Algèbre matricielle rapide en calcul formel et calcul numérique

Skander Belhaj

THÈSE EN COTUTELLE

présentée et soutenue publiquement le 7 mai 2010
au Centre International de Rencontres Mathématiques (CIRM), Marseille
pour l'obtention du grade de

Docteur de l'Université de Franche-Comté – France
et
l'Université de Tunis El Manar – Tunisie
(spécialité Mathématiques Appliquées)

par

Skander BELHAJ

Algèbre matricielle rapide en calcul formel et calcul numérique

dirigée par Henri Lombardi, Mohamed Jaoua et Amel Ben Abda

Composition du Jury

Président :	Nabil Gmati	Université de Tunis El Manar
Rapporteurs :	Dario Andrea Bini	Université de Pise, Italie
	Bernard Mourrain	INRIA, Sophia Antipolis
Membres :	Jean-Claude Yakoubsohn	Université Paul Sabatier, Toulouse III
	Mohamed Jaoua	Université de Nice-Sophia Antipolis
	Amel Ben Abda	Université de Tunis El Manar
	Henri Lombardi	Université de Franche-Comté

Laboratoire de Mathématiques de Besançon -UMR CNRS 6623- 16, route de Gray 25030 Besançon – France
LAMSIN -Ecole National d'Ingénieurs de Tunis- BP 37, 1002 Tunis – Tunisie

Table des matières

À Amira, ma chère épouse.

<< Il vaut mieux savoir tout chercher que chercher à tout savoir. >>
[P. Mendelson]

Remerciements

La réalisation de cette thèse fut une opportunité merveilleuse de rencontrer et d'échanger avec de nombreuses personnes. Je ne saurais pas les citer toutes sans dépasser le nombre de pages raisonnablement admis dans ce genre de travail. Je reconnais que chacune a, à des degrés divers, mais avec une égale bienveillance, apporté une contribution positive à sa finalisation. Mes dettes de reconnaissance sont, à ce point de vue, énormes à leur égard.

Je pense particulièrement à Henri Lombardi pour la finesse de ses attitudes sur le plan aussi bien humain que scientifique. Ses remarques successives ont permis d'améliorer les différentes versions de ce travail. Il a toujours trouvé le juste équilibre entre la liberté qu'il m'a laissée dans le choix des grandes orientations et dans la détermination des pistes à suivre, d'une part, et un soutien total et sans faille dans les moments délicats, d'autre part. De lui, j'ai toujours reçu non seulement les encouragements dont le doctorant a tant besoin, mais aussi les précieux conseils pratiques que seul un homme, ayant des qualités humaines comme lui, peut amener à prodiguer. Grâce à son approche respectueuse de la personne humaine, je me suis continuellement senti à l'aise. Je lui en sais infiniment gré.

Je remercie également Mohamed Jaoua pour ses conseils, sa simplicité et sa disponibilité. Je suis fier d'être parmis ses étudiants et je resterai admiratif face à l'immensité de ses connaissances. Merci pour tous les efforts dont vous avez fait pour que cette thèse marche.

Je remercie ensuite l'ensemble des membres du jury, qui m'ont fait l'honneur de bien vouloir étudier avec attention mon travail : Dario Andrea Bini et Bernard Mourrain pour avoir accepté d'être rapporteurs de cette thèse et pour m'avoir fourni une multitude de pistes de recherche pour les prochaines années ; Jean-Claude Yakoubsohn pour avoir accepté d'examiner cette thèse ; Amel Ben Abda pour avoir accepté mon invitation à participer à ce jury et de m'avoir assurer l'acheminement de diriger cette thèse ; et enfin Nabil Gmati pour m'avoir fait l'honneur d'accepter de présider ce jury.

Mes remerciements s'adressent ensuite à Claude Brezinski (Directeur de mon DEA) qui, le premier, m'a encouragé sur la voie de l'enseignement et de la recherche. Ses nombreux conseils m'ont été précieux.

Cette recherche a été éclairée tout au long de son déroulement par Gema Maria Díaz Toca qui ne m'a jamais compté son soutien et ses conseils depuis de nombreuses années. Elle a cru en l'intérêt de mon thème de recherche dès l'origine et m'a toujours encouragé à poursuivre dans ce sens. Qu'elle trouve ici toute la gratitude que je lui

témoigne.

A l'UTM, j'ai pu bénéficier du soutien de nombreuses personnes. Je remercie les enseignants et toute l'équipe du LAMSIN, particulièrement Henda El Fekih, pour ses nombreux conseils et son implication auprès des doctorants. Je pense aussi particulièrement à Salem Dridi, Raoudha Jelassi et Safouène Benhadj pour leur aide et leur gentillesse.

Je tiens à remercier aussi toute l'équipe du Laboratoire de mathématiques de Besançon qui m'a si bien accueilli. Je pense particulièrement à Mariette Jobard, Rachel Langlet, Odile Henry et Catherine Vuillemenot ainsi que Catherine Pagani pour leur gentillesse et leur efficacité.

J'ai également une pensée pour mes étudiants qui m'ont témoigné encouragements et compréhension durant mes derniers mois de thèse.

Enfin, je me tourne vers ma famille. Mes pensées affectueuses se dirigent vers mes parents Meriem et Ali et ma soeur Wejdène qui m'ont soutenu par leur présence tout au long de cette aventure doctorale. Ils ont toujours été à mes côtés et ont su m'écouter et m'encourager. Ils ont tout mis en oeuvre pour que je puisse mener à bien ce travail. J'espère qu'ils retrouveront dans ce travail les valeurs de liberté, d'autonomie et de fidélité qu'ils m'ont fait partager. J'adresse également mes remerciements les plus affectueux à Tata Bahija, Aam Salah, Yesser, Sana, Noura, Houssine, Bouthaina et Moncef pour leur soutien inconditionnel, leurs encouragements sans faille et surtout pour ce qu'ils représentent pour moi. Je remercie particulièrement Amira, mon épouse. Ce travail te doit beaucoup... Qu'il soit pour toi le témoignage de mon infinie reconnaissance pour ces années de compréhension, de privations et d'efforts communs. Finalement, j'envoie un `<< bisou volant >>` à Oussama qui n'est sans doute pas du comprendre pourquoi je n'avais pas beaucoup le temps de jouer avec lui durant ces derniers mois.

Skander BELHAJ Tunis, le 27 Mai 2010.

Introduction

Le calcul formel a pour objet d'étudier les manipulations symboliques effectives d'objets mathématiques. Il se situe ainsi naturellement à l'interface des mathématiques, de l'informatique et de différents domaines d'applications. La recherche dans ce domaine a pour ambition le développement des outils mathématiques pour la conception et l'analyse des algorithmes pour la solution efficace des problèmes informatiques.

Il s'avère nécessaire de s'interroger sur la corrélation entre le domaine du calcul formel et de l'algèbre matricielle : les matrices structurées (tel que Toeplitz, Hankel, Frobenius, Sylvester, Bézout, Vandermonde, ...) [62] sont omniprésentes en calcul numérique et algébrique dans les domaines des sciences, de l'ingénierie, de la communication et des statistiques.

Ma thèse intitulée **"Algèbre matricielle rapide en calcul formel et calcul numérique"**, sous la direction du Prof. H. LOMBARDI (Université de Franche-Comté), Prof. A. BEN ABDA (Université de Tunis El Manar) et Prof. M. JAOUA (Université de Nice Sophia-Antipolis) a pour objectif essentiel : la conception de nouveaux algorithmes rapides en calcul formel et calcul numérique [25, 62] via les matrices structurées.

Notre contribution consiste essentiellement à concevoir et développer différents types d'algorithmes rapides en rapport direct avec les matrices de Toeplitz/Hankel.

La présentation de notre recherche et son analyse sera structurée de la manière suivante :

Nous commençons notre mémoire par la présentation des différentes notations et terminologies utilisées par la suite.

Le Chapitre 1 contient quelques méthodes classiques couramment utilisées pour calculer l'inverse d'une matrice triangulaire de Toeplitz dans une approche exacte : l'Algorithme 1.3.1 : Algorithme d'inversion via substitution requiert environ $n\,(n+1)$ opérations arithmétiques et la méthode d'inversion via division polynomiale qui se fait grâce à une inversion dans un anneau de développement limité nécessite $O(n\log(n))$ opérations (environ 10 transformées de Fourier rapides de n-vecteurs : FFT(n)). Nous présentons aussi les différents travaux théoriques et empiriques portant sur le calcul de l'inverse d'une matrice triangulaire de Toeplitz dans une approche approximative : l'Algorithme 1.4.1 : Algorithme de Fu-Rong Lin, Wai-Ki Ching et M. K. Ng [60] basé sur l'interpolation polynomiale trigonométrique nécessite 2 transformées de Fourier rapides de $2n$-vecteurs : FFT$(2n)$ et une transformation rapide de cosinus de $2n$-vecteurs : DCT$(2n)$, l'Algorithme 1.4.2 : Algorithme de Bini [21] et enfin l'Algorithme

1.4.3 : Algorithme de Bini révisé amélioré par Fu-Rong Lin, Wai-Ki Ching et M. K. Ng nécessite 2 transformées de Fourier rapides de $2n$-vecteurs : FFT($2n$) (environ 2 exécutions de l'algorithme de Bini et $\frac{4}{5}$ de l'algorithme via Interpolation), la plus efficace de point de vue précision et rapidité.

Dans le Chapitre 2, nous considérons un algorithme de diagonalisation par blocs approchée d'une matrice carrée réelle de Hankel H dans laquelle la matrice de passage A est triangulaire supérieure. Cette diagonalisation nous donne une matrice diagonale par blocs

$$D = {}^tAHA = \begin{pmatrix} lH_1 & & & \\ & lH_2 & & \\ & & \ddots & \\ & & & lH_n \end{pmatrix}$$

où chaque bloc lH_j, pour $j = 1, 2, ..., n$ est une matrice triangulaire inférieure de Hankel. Nous nous intéressons au problème de la factorisation par blocs approchée d'une matrice réelle de Hankel. Ce problème a été étudié par Phillips [67], Rissanen [68], Kung [57], Gragg et Lindquist [46], Pal et Kailath [64], Bini et Pan [25], Bultheel et Van Barel [28] et récemment par Ben Atti et Diaz-Toca [20]. Bien que les algorithmes cités ci-dessus semblent être rapides, ils présentent quelques problèmes d'efficacités numériques. En effet, quand nous perturbons la matrice de Hankel, la diagonalisation par blocs donne des blocs 1×1 au lieu des blocs $m \times m$; Ainsi, nous cherchons dans un voisinage de la matrice de Hankel (qui a subi une perturbation) une matrice qui a une diagonalisation par blocs capable de se faire avec des matrices de passage triangulaire de déterminant pas trop grand. Afin d'atténuer cette difficulté, il est donc nécessaire d'introduire le concept de diagonalisation par blocs approchée de la matrice de Hankel. Cette diagonalisation nous donne la matrice diagonale par blocs approchée

$$D_\varepsilon = {}^tA_\epsilon H A_\epsilon = \begin{pmatrix} lH_{\epsilon_1} & \Theta_{1,2} & \cdots & \Theta_{1,n} \\ \Theta_{2,1} & lH_{\epsilon_2} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \Theta_{n-1,n} \\ \Theta_{n,1} & \cdots & \Theta_{n,n-1} & lH_{\epsilon_n} \end{pmatrix}$$

où chaque bloc lH_{ϵ_j} est une matrice triangulaire inférieure de Hankel (par rapport à l'anti-diagonale) approchée pour $j = 1, 2, ..., n$, chaque matrice $\Theta_{j,k}$ est très proche de la matrice nulle pour $j, k = 1, 2, ..., n$ et $j \neq k$ et A_ϵ est une matrice triangulaire supérieure approchée. Nous présentons une nouvelle méthode (Algorithme 2.2.2) inspirée de [20] et basée sur le calcul approché de l'inverse de la matrice triangulaire supérieure de Toeplitz via les techniques du Chapitre 1 en $O(n^2)$ opérations. Nous présentons également une variante rapide (Algorithme 2.3) de factorisation approchée par blocs d'une matrice réelle de Hankel basée sur le complément de Schur [25, 45, 62]. Des exemples numériques ont été présentés afin de mettre en évidence les avantages potentiels de la nouvelle diagonalisation par blocs approchée par rapport à la méthode classique de factorisation approchée, en terme d'efficacités numériques et en terme de temps.

Le Chapitre 3 est consacré à l'application de l'algorithme de diagonalisation par blocs approchée, présenté dans [6], à la matrice de Hankel associée à deux polynômes premiers entre eux u et v. Soient

$$u(x) = \sum_{k=0}^{n} u_k x^k \text{ et } v(x) = \sum_{k=0}^{m} v_k x^k$$

deux polynômes dans $\mathbb{R}[x]$ avec $\deg(u(x)) = n$ et $\deg(v(x)) = m$ et $m < n$. L'algorithme d'Euclide classique appliqué à $u(x)$ et $v(x)$ génère une suite des quotients $\{q_k(x)\}$ et des restes $\{r_k(x)\}$, de la manière suivante :

$$\begin{aligned} &r_{-1}(x) = u(x), \quad r_0(x) = v(x) \\ &r_{k-2}(x) = r_{k-1}(x)\, q_k(x) - r_k(x), \quad \deg(r_k) < \deg(r_{k-1}),\ k = 1, \ldots, K \end{aligned}$$

où $-r_k(x)$ est le reste de la division de $r_{k-2}(x)$ par $r_{k-1}(x)$ et $r_K(x)$ est le Plus Grand Commun Diviseur (PGCD) de $u(x)$ et $v(x)$. La question du calcul de la suite des quotients et des restes a été étudiée par Habicht, Collins et Brown (voir [47, 34, 31, 32]) via des méthodes efficaces basées sur des variantes de l'algorithme Euclide. Naturellement, l'algorithme classique s'avère inefficace lorsqu'il est appliqué à des polynômes de départ perturbés. Afin d'éviter ce problème, Noda et Sasaki (voir [71, 59]) sont parmi les premiers à introduire des versions numériques efficaces de l'algorithme d'Euclide.

Plusieurs articles ont été consacrés à décrire la relation naturelle entre l'algorithme d'Euclide classique et la factorisation par blocs des matrices de Hankel. En effet, Bini et Gemignani (voir [25, 22, 43, 23]) ont calculé la suite des quotients et des restes apparue durant l'exécution de l'Algorithme d'Euclide appliqué à $u(x)$ et $v(x)$ via la factorisation LU par blocs (voir [46, 49]) de la matrice de Hankel $H(u, v)$, associée à $u(x)$ et $v(x)$. Récemment, Ben Atti et Diaz-Toca [39] ont calculé la suite des quotients et des restes via une approche différente de la version classique basée sur la factorisation LU par blocs de la matrice de Hankel $H(u, v)$ [20].

Naturellement, la belle relation entre l'algorithme d'Euclide classique et la diagonalisation par blocs LU des matrices de Hankel $H(u, v)$ semble être égarée quand les polynômes de départ sont perturbés. A vrai dire, on se heurte à un problème de conditionnement. C'est pourquoi une nouvelle notion de diagonalisation par blocs approchée de la matrice de Hankel $H(u, v)$ est introduite dans ce chapitre en s'inspirant des techniques de [6]. Nous proposons aussi un algorithme révisé qui permet de préserver la belle relation entre l'algorithme d'Euclide classique et la factorisation par blocs des matrices de Hankel en générant la suite des quotients et des restes. Enfin, nous illustrons notre approche par un exemple.

Le chapitre 4 est consacré à l'analyse de la diagonalisation par blocs approchée des matrices de Hankel et de Bézout associées à deux polynômes $u(x)$ et $v(x)$ dans $\mathbb{R}[x]$ ($\deg(u(x)) = n$, $\deg(v(x)) = m$ et $m < n$) et de leurs connexions avec l'Algorithme d'Euclide. En effet, Bini et Gemignani [25, 22, 43, 23] introduisent un algorithme parallèle pour la diagonalisation par blocs de la matrice de Bézout $JB(u, v)J$ et génèrent la suite des quotients et des restes apparue durant l'exécution de l'Algorithme d'Euclide appliqué à $u(x)$ et $v(x)$. Cette diagonalisation par blocs de $JB(u, v)J$ peut

être obtenue à partir de la diagonalisation par blocs de $H(u, v)$ et vice versa en se basant sur la relation suivante [58] :

$$B(u, v) = B(u, 1)H(u, v)B(u, 1).$$

Naturellement, quand les polynômes de départ sont perturbés, la belle relation entre l'algorithme d'Euclide (ou la diagonalisation par blocs de $H(u, v)$) et la diagonalisation par blocs des matrices de Bézout $JB(u, v)J$ est conservée. Par conséquent, une nouvelle notion de diagonalisation par blocs approchée de la matrice de Bézout $JB(u, v)J$, inspirée des travaux de Bini et Gemignani [25, 22, 43, 23], a été introduite et sa relation avec l'algorithme d'Euclide ainsi que la diagonalisation par blocs approchée des matrices de Hankel a été vérifiée dans ce chapitre. Des comparaisons numériques montrent l'efficacité des algorithmes.

La technique considérée pour parvenir à la mise en œuvre pertinente des algorithmes du Chapitre 2 est utilisée dans le Chapitre 5 afin de concevoir deux nouvelles variantes de diagonalisation par blocs d'une matrice de Hankel à coefficients complexes. Bien que les algorithmes décrites dans le Chapitre 2 semblent être les premiers à étudier le cas approché, ils présentent encore des problèmes d'efficacités numériques. En effet, quand les coefficients de la matrice de Hankel perturbé sont complexes, nous ne préservons pas la diagonalisation par blocs de la matrice de Hankel. Par conséquent, nous étudions ce problème dans le cas complexe en proposant deux algorithmes : Algorithme 5.1.2 et Algorithme 5.2.1. Nous montrons ensuite que les algorithmes sont intéressants du point de vue empirique. Enfin, nous présentons un exemple qui permet de préserver la relation étroite entre l'Algorithme d'Euclide et la diagonalisation par blocs de la matrice de Hankel dans le cas complexe.

L'un des buts de cette thèse est de certifier théoriquement les algorithmes présentés dans les précédents chapitres (cela veut dire que, dans le voisinage des données défini par la précision, les blocs sont optimaux (i.e. les plus gros possibles) et que d'autres calculs donnant des blocs trop petits entraîneraient inévitablement des erreurs grossières dues à des divisions par des nombres trop petits). Nous n'avons pas pu aboutir à notre but, parce que ce problème s'est révélé énormément plus difficile qu'on pensait à priori. Légitimement, nous présentons dans le Chapitre 6 une étude expérimentale conséquente de la propagation des erreurs dans les étapes intermédiaires des algorithmes issus de cette thèse. Nous comparons plus particulièrement l'efficacité basée sur une étude statistique sur le comportement de l'erreur absolue appliquée à une vingtaine de matrices de départ "exactes", réelles, complexes malconditionnées que l'on perturbe aléatoirement 500 fois. Nous comparons par la suite la stabilité numérique des algorithmes de diagonalisation par blocs de la matrice de Hankel. De plus, nous examinons les différents algorithmes en terme de temps de calculs (en seconde). Nous étudions également l'influence d'appliquer l'inversion approchée d'une matrice triangulaire de Toeplitz au lieu de l'inversion exacte dans les étapes intermédiaires des algorithmes de diagonalisations par blocs de matrices de Hankel. Nous comparons aussi l'approche fondée sur Hankel avec celle basée sur des matrices

de Bézout tant pour leur efficacité que pour leur stabilité numérique. Enfin, nous analysons le rôle de ε dans l'exécution des algorithmes, en mesurant le nombre des blocs au moyen de notre méthode et la méthode du complément de Schur, avec et sans la technique SVD. Notre apport expérimental constitue une tentative de théoriser les calculs sur ordinateur (efficacité et stabilité numérique) et leurs temps d'exécution.

Dans l'Annexe A nous faisons quelques rappels d'algèbre matricielle numérique et donnons, un aperçu détaillé des différentes méthodes utilisées pour calculer la transformée de Fourier rapide, FFT (Fast Fourier Transform) et de la transformée en cosinus discrète, DCT (Discrete Cosine Transform) [2, 25, 1, 26].

L'Annexe B contient les codes MATLAB des algorithmes expérimentés. Nous avons choisi le logiciel de Calcul Numérique MATLAB essentiellement pour des raisons de commodité. Le langage de programmation qui lui est rattaché est proche de celui de nombreux autres langages classiques, permettant de définir et de présenter de manière lisible et efficace les algorithmes considérés. Les autres langages de calcul numérique généralistes auraient pu aussi bien faire l'affaire. Il n'y aura d'ailleurs aucun mal à implémenter dans un de ces langages les algorithmes présentés dans cette thèse.

Notations et préliminaires

Une matrice de **Hankel** est une matrice (pas nécessairement carrée) $H = (h_{ij})$ dont les coefficients sont constants sur les diagonales montantes : $h_{ij} = h_{pq}$ si $i + j = p + q$.

Les matrices de Hankel fournissent un exemple de **matrices structurées**. L'autre exemple le plus important est celui des matrices de **Toeplitz** $T = (t_{ij})$, celles dont les coefficients sont constants sur les diagonales descendantes : $t_{ij} = t_{pq}$ si $i - j = p - q$.

Remarquons qu'une matrice de Hankel carrée d'ordre n est une matrice symétrique et que les produits HJ_n et J_nH d'une matrice de Hankel H carrée d'ordre n par la matrice de Hankel particulière J_n (qu'on appelle aussi anti-unité : comporte seulement des uns sur l'anti-diagonale) sont des matrices de Toeplitz. Cette matrice de permutation d'ordre n permet de renverser l'ordre des n colonnes (respectivement des n lignes) d'une matrice lorsque celle-ci est multipliée à droite (respectivement à gauche) par la matrice J_n : c'est pourquoi on l'appelle matrice de renversement ou encore matrice d'arabisation du fait qu'elle permet d'écrire de droite à gauche les colonnes que l'on lit de gauche à droite et inversement.

Inversement, les produits TJ_n et J_nT d'une matrice de Toeplitz T carrée d'ordre n par la matrice J_n sont des matrices de Hankel.

Une matrice structurée est déterminée par la donnée de beaucoup moins de coefficients qu'une matrice ordinaire de même taille. Par exemple une matrice de Hankel (respectivement de Toeplitz) de type (n, p) est déterminée par la donnée de $(n + p - 1)$ coefficients : ceux des premières lignes et dernières (respectivement premières) colonnes.

Cela rend ces matrices particulièrement importantes pour les grands calculs d'algèbre matricielle.

Dans tous ce qui suit nous désignons par :

- $\mathbb{K} = \mathbb{R}$ ou $\mathbb{C}$.
- $H(S) \in \mathbb{K}^{n\times n}$ la matrice de Hankel associée à une liste S de taille $(2n-1)$. C'est-à-dire la première ligne est donnée par les n premiers termes de S et la dernière colonne est donnée par les n derniers termes de S.
- $lH(S) \in \mathbb{K}^{n\times n}$ la matrice triangulaire inférieure de Hankel (par rapport à l'anti-diagonale) associée à une liste S telle que la dernière colonne est définie par S.
- $H(S; m; n) \in \mathbb{K}^{m\times n}$ la matrice de Hankel associée à une liste S de taille $(m + n - 1)$. La première ligne est donnée par les n premiers termes de S et la dernière colonne est donnée par les m derniers termes de S.
- $T(S; m; n) \in \mathbb{K}^{m\times n}$ la matrice de Toeplitz associée à une liste S de taille

$(m+n-1)$. La première ligne est donnée par les m premiers termes de S et la dernière colonne est donnée par les n derniers termes de S.

- $uT(S) \in \mathbb{K}^{n\times n}$ la matrice triangulaire supérieure de Toeplitz associée à une liste S telle que la première ligne est définie par S.
- Soit $Z_n \in \mathbb{K}^{n\times n}$. La matrice $Z_n = uT(0,1,0,\ldots,0)$ de *shift* vers le haut, c'est-à-dire;

$$Z_n = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ \vdots & & & \ddots & 1 \\ 0 & \cdots & \cdots & 0 & 0 \end{pmatrix}.$$

est nilpotente.

- Soit $p \in \mathbb{N}$. Soit $\Sigma_p \in \mathbb{K}^{p\times p}$, $\Sigma_p = [\varepsilon_{j,k}]_{j,k=1}^{p}$, où toutes les entrées de Σ_p sont zéro sauf que $\varepsilon_{j+k,j} = \varepsilon_k$ avec $j+k = 1,2,...,p$;

$$\Sigma_p = \begin{pmatrix} 0 & \cdots & \cdots & \cdots & 0 \\ \varepsilon_1 & \ddots & & & \vdots \\ \varepsilon_2 & \ddots & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \varepsilon_{p-1} & \cdots & \varepsilon_2 & \varepsilon_1 & 0 \end{pmatrix}.$$

- Soit $p \in \mathbb{N}$. Soit $\Sigma \in \mathbb{K}^{p\times p}$, $\Sigma = [\varepsilon_{j,k}]_{j,k=1}^{p}$, où toutes les entrées de Σ sont zéro sauf pour $\varepsilon_{j+k,j}$ avec $j+k = 1,2,...,p$;

$$\Sigma = \begin{pmatrix} 0 & \cdots & \cdots & \cdots & 0 \\ \varepsilon_{2,1} & \ddots & & & \vdots \\ \varepsilon_{3,1} & \varepsilon_{3,2} & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \varepsilon_{p,1} & \cdots & \varepsilon_{p,p-2} & \varepsilon_{p,p-1} & 0 \end{pmatrix}.$$

- $J_p = lH(1,\underbrace{0,...,0}_{p-1})$, $p \in \mathbb{N}$.
- Soit $P \in \mathbb{K}^{n\times m}$, $\widetilde{P} = J_m^t P J_n$.
- $T(S;m;n) = J_m H(S;m;n)$.
- $uT(S) = J_l lH(S)$, où l est la taille de L.
- Soit $a \in \mathbb{R}$. Soit $\mu > 0$, $V(a,\mu) =]a-\mu; a+\mu[$ est un voisinage de a.

Chapitre 1

Inversion rapide des matrices triangulaires de Toeplitz

1.1 Introduction

Il s'agit dans ce chapitre de décrire et d'analyser certaines méthodes, plus ou moins classiques, pour calculer l'inverse des matrices triangulaires de Toeplitz. Il existe deux approches afin de calculer l'inverse des matrices triangulaires de Toeplitz : inversion exacte [24], [37] et inversion approchée [21], [25], [60]. Une méthode simple pour trouver l'inverse d'une matrice triangulaire de Toeplitz basée sur la substitution, requiert environ $n\,(n+1)$ opérations arithmétiques. Une autre méthode d'inversion exacte d'une matrice triangulaire de Toeplitz se fait grâce à une inversion dans un anneau de développement limité nécessite $O(n\log(n))$ opérations. Plus spécifiquement, environ 10 transformées de Fourier rapides de n-vecteurs : FFT(n) [37], [25].

Dans ce chapitre, nous nous allons traiter le problème de calcul de l'inverse d'une matrice triangulaire inférieure (ou supérieure) de Toeplitz dans une approche approximative. Les techniques utilisées sont purement numériques mais en même temps utilisent la division polynomiale, qui est un outil fondamental dans le calcul formel et aussi équivalent à l'inversion de la matrice triangulaire de Toeplitz. Les algorithmes proposés ont des techniques similaires à celles des algorithmes dans [65, 66] basés sur la division polynomiale.

Ainsi, l'objectif recherché est de comparer ces algorithmes et de dégager le meilleur possible, c'est à dire le plus rapide théoriquement et pratiquement, occupant le moins d'espace possible afin d'appliquer cette technique dans le Chapitre 2.

1.2 Origine des matrices de Toeplitz

Dans plusieurs domaines scientifiques, la modélisation mathématique de divers problèmes conduit, éventuellement après une étape de discrétisation, ou de linéarisation pour les problèmes non linéaires, à la résolution *numérique* de systèmes linéaires de très grande taille. Cette taille peut atteindre des milliers d'équations (par exemple lors de la résolution par discrétisation des équations aux dérivées partielles à partir

des phénomènes physiques). Pour cela, il est naturel de concevoir des algorithmes rapides, efficaces et stables.

Il existe plusieurs algorithmes de résolution de systèmes linéaires qui nécessitent $O(n^3)$ opérations arithmétiques pour une matrice quelconque d'ordre n [45, 1]. En effet, nous abordons seulement des systèmes linéaires correspondant à des matrices possédant une structure particulière qui peuvent être traités par des algorithmes plus rapides. Cette structure spéciale conduit à la naissance des *matrices structurées* (symétrique, bande, circulante, Toeplitz, Hankel, Bézout, Frobenius, Sylvester, Vandermonde, . . .) [62]. Spécifiquement, les *matrices de Toeplitz* sont omniprésentes dans plusieurs problèmes des sciences, de l'ingénierie (par exemple, le traitement de signal et d'image [38]).

En exploitant la structure de la matrice de Toeplitz, la complexité des algorithmes de résolution de systèmes linéaires et/ou d'inverser la matrice de Toeplitz peut être réduite de $O(n^3)$ à $O(n^2)$ opérations arithmétiques (voire $O(n\log(n))$ opérations pour les algorithmes de type *divide-and-conquer*).

Dans ce qui suit, nous introduisons des méthodes rapides pour calculer l'inverse de la *matrice triangulaire supérieure de Toeplitz*

$$T_n = \begin{pmatrix} t_0 & t_1 & \cdots & t_{n-1} \\ & t_0 & \ddots & \vdots \\ & & \ddots & t_1 \\ & & & t_0 \end{pmatrix},$$

avec $t_j \in \mathbb{R}$, pour $j = 0, 1, ..., n-1$ et $t_0 \neq 0$.

Ainsi, nous introduisons quelques propriétés d'une matrice triangulaire supérieure de Toeplitz.

Propriété 1.2.1 Chaque matrice triangulaire supérieure de Toeplitz est représentée comme une combinaison linéaire des puissances de matrices de shift vers le haut :

$$T_n = uT\left(t_0, t_1, \ldots, t_{n-1}\right) = t_0 I + t_1 Z_n + t_2 Z_n^2 + \cdots + t_{n-1} Z_n^{n-1}.$$

Propriété 1.2.2 Pour toute matrice triangulaire supérieure de Toeplitz T_n, nous définissons le polynôme :

$$\tau_n\left(z\right) = \sum_{j=0}^{n-1} t_j z^j = t_0 + t_1 z + \cdots + t_{n-1} z^{n-1}. \tag{1.1}$$

Le développement en série de Maclaurin de $\tau_n^{-1}\left(z\right)$ est donné par

$$\tau_n^{-1}\left(z\right) = \sum_{j=0}^{\infty} b_j z^j = b_0 + b_1 z + \cdots + b_{n-1} z^{n-1} + \cdots . \tag{1.2}$$

Alors

$$T_n^{-1} = \begin{pmatrix} b_0 & b_1 & \cdots & b_{n-1} \\ & b_0 & \ddots & \vdots \\ & & \ddots & b_1 \\ & & & b_0 \end{pmatrix}. \tag{1.3}$$

Ainsi afin d'obtenir T_n^{-1}, nous avons besoin seulement de calculer les coefficients b_j pour $j = 1, 2, ..., n-1$.

Propriété 1.2.3 Remplaçant z dans (1.1) et (1.2) par ρz, nous obtenons

$$\tau_{n,\rho}(z) = \tau_n(\rho z) = \sum_{j=0}^{n-1} \left(t_j \rho^j\right) z^j \quad \text{et} \quad \tau_{n,\rho}^{-1}(z) = \tau_n^{-1}(\rho z) = \sum_{j=0}^{\infty} \left(b_j \rho^j\right) z^j \ .$$

De plus,

$$\begin{pmatrix} t_0 & \rho t_1 & \cdots & \rho^{n-1} t_{n-1} \\ & t_0 & \ddots & \vdots \\ & & \ddots & \rho t_1 \\ & & & t_0 \end{pmatrix}^{-1} = \begin{pmatrix} b_0 & \rho b_1 & \cdots & \rho^{n-1} b_{n-1} \\ & b_0 & \ddots & \vdots \\ & & \ddots & \rho b_1 \\ & & & b_0 \end{pmatrix}.$$

Nous pouvons choisir $\rho \in]0, 1[$ tel que

$$\sum_{j=0}^{\infty} \left(b_j \rho^j\right) < \infty. \tag{1.4}$$

Propriété 1.2.4 L'inverse de la sous-matrice principale dominante $T_n(1:p, 1:p)$ est égale à la sous-matrice principale dominante $T_n^{-1}(1:p, 1:p)$ pour $1 \leq p \leq n$.

1.3 Inversion exacte d'une matrice triangulaire de Toeplitz

1.3.1 Inversion via substitution d'une matrice triangulaire de Toeplitz

Nous rappelons, dans cette partie, un algorithme d'inversion de la matrice triangulaire de Toeplitz basé sur la substitution, requiert environ $n(n+1)$ opérations arithmétiques [45].

Ainsi, l'algorithme de cette méthode est décrit comme suit :

Algorithme 1.3.1 (Inversion via substitution d'une matrice triangulaire de Toeplitz) On donne une liste $t = [t_0, t_1, \ldots, t_{n-1}]$, cet algorithme permet de calculer l'inverse d'une matrice triangulaire supérieure (ou inférieure) de Toeplitz s par la méthode de substitution.

1. Poser $n = \text{taille}(t)$ et $s = \text{zeros}(n, 1)$
2. Poser $s(1) = \frac{1}{t(1)}$
3. Pour $k = 2 : n$

   $s(k:n) = s(k:n) + s(k-1) * t(2 : n-k+2)$

   $s(k) = -s(k)/t(1)$

   FinPour

1.3.2 Inversion via division polynomiale d'une matrice triangulaire de Toeplitz

Une autre méthode d'inversion exacte d'une matrice triangulaire de Toeplitz se fait grâce à une inversion dans un anneau de développement limités [25], ce qui revient à une division de 1 par un autre polynôme dans $\mathbb{K}[x]$ selon les puissances croissantes et sans tenir compte des puissances trop grandes. Une telle inversion dans l'anneau des développements limités est nettement plus rapide qu'une inversion générale de matrice. Précisément :

Lemme 1.3.2 Si $T = uT(\alpha_1, \ldots, \alpha_m)$ avec $\alpha_1 \neq 0$, $T^{-1} = uT(\mu_1, \ldots, \mu_m)$, en posant

$$S(x) = \alpha_1 + \alpha_2 x + \cdots + \alpha_m x^{m-1} = \sum_{k=1}^{m} \alpha_k x^{k-1},$$

$$Q(x) = \mu_1 + \mu_2 x + \cdots + \mu_m x^{m-1} = \sum_{k=1}^{m} \mu_k x^{k-1},$$

on obtient $S(x)\,Q(x) = 1 \bmod x^m$. Autrement dit $Q(x)$ est le quotient dans la division suivant les puissances croissantes de 1 par $S(x)$ à l'ordre $(m-1)$: $Q = 1/S \bmod x^m$.

Preuve. La démonstration est dans le livre [25]. ■

1.4 Inversion approchée d'une matrice triangulaire de Toeplitz

1.4.1 Inversion via interpolation d'une matrice triangulaire de Toeplitz

Dans cette partie, nous présentons l'algorithme de Fu-Rong Lin, Wai-Ki Ching et M. K. Ng [60] qui permet de calculer l'inverse d'une matrice triangulaire de Toeplitz basé sur l'interpolation polynomiale trigonométrique. Cette méthode génère l'inverse d'une matrice triangulaire à haute précision et à faible coût (nécessite deux FFT et une transformation rapide de cosinus (DCT) de $2n$-vecteurs).

En remplaçant z par e^{ix}, avec $i^2 = -1$ et $x \in \mathbb{R}$, alors $\tau_n(e^{ix})$ est un polynôme trigonométrique. Par conséquent, pour récupérer les b_j, nous pensons directement à calculer les coefficients de Fourier de $1/\tau_n(e^{ix})$:

$$b_j = \frac{1}{2\pi} \int_0^{2\pi} \frac{1}{\tau_n(e^{ix})} e^{ix} dx \quad \text{pour } j = 1, 2, ..., n-1. \tag{1.5}$$

Malheureusement, il est difficile de calculer b_j efficacement en utilisant la formule ci-dessus car le calcul explicite de τ_n^{-1} est généralement inconnu. Conséquemment, nous considérons d'interpoler τ_n^{-1} (1.2) via l'interpolation polynomiale trigonométrique.

Nous notons qu'il est difficile de calculer b_j via l'interpolation polynomiale algébrique car les matrices de Vandermonde d'ordre élevée sont très mal-conditionnées. Nous posons

$$\tau_\rho(\theta) \equiv \tau_{n,\rho}\left(e^{-i\theta}\right) = \sum_{j=0}^{n-1}\left(t_j\rho^j\right)e^{-ij\theta} = \tau_\rho^{(r)}(\theta) + i\tau_\rho^{(i)}(\theta), \tag{1.6}$$

avec $\tau_\rho^{(r)}(\theta)$ et $\tau_\rho^{(i)}(\theta)$ sont les parties réelles et imaginaires de $\tau_\rho(\theta)$, respectivement. Plus précisément, en utilisant (1.2), nous avons

$$h_\rho(\theta) \equiv \tau_\rho^{-1}(\theta) = \sum_{j=0}^{\infty}\left(b_j\rho^j\right)e^{-ij\theta}.$$

En particulier, la partie réelle de $h_\rho(\theta)$ est donnée par

$$h_\rho^{(r)}(\theta) = \sum_{j=0}^{\infty}\left(b_j\rho^j\right)\cos(j\theta) = \frac{\tau_\rho^{(r)}(\theta)}{\left(\tau_\rho^{(r)}(\theta)\right)^2 + \left(\tau_\rho^{(i)}(\theta)\right)^2}. \tag{1.7}$$

Nous avons $h_\rho^{(r)}(\theta)$ est une fonction paire et 2π-périodique. Pour obtenir des valeurs approchées $\widehat{b}_j$ de b_j pour $j = 0, 1, ..., n-1$, nous interpelons $h_\rho^{(r)}$ par une fonction de l'ensemble Π_{n-1}, où Π_m est l'ensemble de tous les polynômes trigonométriques paires de degré $\leq m$. Nous utilisons les points équidistants suivants :

$$\theta_j = \frac{2j-1}{2n}\pi, \quad j = 1, ..., n$$

comme des nœuds d'interpolation. Ainsi, l'avantage d'utiliser ces points d'interpolation est, d'une part, $\widehat{b}_j$ peut être calculé efficacement via une transformation rapide de cosinus (DCT) et, d'autre part, le polynôme d'interpolation trigonométrique peut approximer efficacement la fonction de départ.

Soit

$$P_{n-1}(\theta) = \sum_{j=0}^{n-1} c_j\cos(j\theta)$$

est le polynôme d'interpolation de $h_\rho^{(r)}(\theta)$. En utilisant les conditions d'interpolation $P_{n-1}(\theta_j) = h_\rho^{(r)}(\theta_j)$, $j = 1, ..., n$, nous avons

$$C\begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \end{pmatrix} = \begin{pmatrix} h_\rho^{(r)}(\theta_1) \\ h_\rho^{(r)}(\theta_2) \\ \vdots \\ h_\rho^{(r)}(\theta_n) \end{pmatrix} \tag{1.8}$$

avec $C = (c_{jk})_{j,k=1,2,...,n}$ tel que $c_{jk} = \cos\left(\frac{(k-1)(2j-1)\pi}{2n}\right)$, $j, k = 1, 2, ..., n$.

Notons que $C = (\Phi^c)' D$, avec Φ^c est la matrice de transformation discrète de cosinus et

$$D = \operatorname{diag}\left(\sqrt{n}, \sqrt{\frac{n}{2}} I_{n-1}\right) = \begin{pmatrix} \sqrt{n} & & & \\ & \sqrt{\frac{n}{2}} & & \\ & & \ddots & \\ & & & \sqrt{\frac{n}{2}} \end{pmatrix},$$

nous remarquons que si les valeurs de $h_\rho^{(r)}(\theta_j)$, $j = 1, 2, ..., n$ sont données, alors c_j peut être obtenu en utilisant une DCT de n-vecteurs (DCT(n)). Finalement, $\widehat{b}_j$ peut être obtenu en $O(n)$ divisions via $\widehat{b}_j = c_j \rho^{-j}$, $j = 0, 1, ..., n-1$.

Regardons maintenant l'efficacité du polynôme d'interpolation P_{n-1}. Nous avons

$$\left\| P_{n-1} - h_\rho^{(r)} \right\|_\infty \leq \left(2 + \frac{2}{\pi} \log(n)\right) E_{n-1}\left(h_\rho^{(r)}\right),$$

avec $E_m(h_r) = \min\limits_{P \in \Pi_m} \|P - h_r\|_\infty$ est l'erreur de la meilleure approximation dans Π_m [27, 69].

Lemme 1.4.1 Soit f de classe $\mathcal{C}^p$ alors $E_{m-1}(f) \leq \frac{\pi}{2}\left(\frac{1}{m}\right)^p \left\| f^{(p)} \right\|_\infty$.

Preuve. La preuve est basée sur le théorème de Jackson (voir [35]). ■

Ainsi, nous proposons le théorème suivant pour déterminer l'efficacité de $\widehat{b}_j$, $j = 0, 1, ..., n-1$.

Théorème 1.4.1 Soient $\sum\limits_{j=0}^{\infty} (b_j \rho^j) z^j$ le développement en série de Maclaurin de $\tau_n^{-1}(\rho z)$ et $\rho \in]0, 1[$ tel que $\sum\limits_{j=0}^{\infty} |b_j \rho^j| < \infty$. Soit $P_{n-1}(\theta) = \sum\limits_{k=0}^{n-1} c_k \cos(k\theta)$ le polynôme d'interpolation de $h_\rho^{(r)}(\theta)$ avec les points d'interpolation sont $\theta_j = \frac{2j-1}{2n}\pi$, $j = 1, ..., n$ et $\widehat{b}_k = c_k \rho^{-k}$, $k = 0, 1, ..., n-1$. Alors,

$$\begin{aligned} \widehat{b}_0 &= b_0 + \sum_{m=1}^{\infty} (-1)^m \rho^{2mn} b_{2mn}, \\ \widehat{b}_k &= b_k + \sum_{m=1}^{\infty} (-1)^m \left(\rho^{2mn} b_{2mn+k} + \rho^{2(mn-k)} b_{2mn-k}\right), \quad k = 1, 2, ..., n-1. \end{aligned} \tag{1.9}$$

Preuve. On considère le polynôme d'interpolation de $\cos(k\theta)$ avec points d'interpolation θ_j, $j = 1, 2, ..., n$ et $k \in \mathbb{N}$. Soient $m \in \mathbb{N}$ et $k \in \mathbb{N}$, nous avons pour $j = 1, 2, ..., n$,

$$\cos\left((2mn \pm k)\frac{2j-1}{2n}\pi\right) = (-1)^m \cos\left(\frac{(2j-1)k}{2n}\pi\right).$$

En particulier, $\cos((2mn \pm k)\theta j) = (-1)^m$ et $\cos((2m+1)n\theta j) = 0$. Ainsi, les polynômes d'interpolation de $\cos(2mn\theta)$, $\cos((2m+1)n\theta)$ et $\cos((2mn \pm k)\theta)$ sont

$(-1)^m$, 0 et $(-1)^m \cos(k\theta)$, respectivement. Ceci nous permet de conclure

$$\widehat{b}_0 = b_0 + \sum_{m=1}^{\infty} (-1)^m \rho^{2mn} b_{2mn},$$
$$c_k = c_k\rho^k + \rho^k \sum_{m=1}^{\infty} (-1)^m \left(\rho^{2mn} b_{2mn+k} + \rho^{2(mn-k)} b_{2mn-k}\right), \quad k = 1, 2, ..., n-1.$$

Finalement, en remplaçant c_k par $\widehat{b}_k\rho^k$, nous concluons (1.9). ■

D'après (1.9), nous remarquons lorsque ρ est petit, $\widehat{b}_k$ est proche de b_k, $k = 0, 1, ..., n-1$. En effet, mettant dans (1.9) ρ^{2n} en facteur (on peut choisir ρ tel que ρ^{2n} est très proche de 0), nous obtenons

$$\begin{aligned}
\widehat{b}_0 &= b_0 + \rho^{2n} \sum_{m=1}^{\infty} (-1)^m \rho^m b_{2mn} \approx b_0, \\
\widehat{b}_k &= b_k + \sum_{m=1}^{\infty} (-1)^m \left(\rho^{2mn} b_{2mn+k} + \rho^{2(mn-k)} b_{2mn-k}\right) \\
&= b_k + \rho^{2n} \sum_{m=1}^{\infty} (-1)^m \rho^m b_{2mn+k} + \sum_{m=1}^{\infty} (-1)^m \rho^{2(mn-k)} b_{2mn-k} \\
&\approx b_k + \rho^{2(n-k)} \left(-b_{2n-k} + \rho^{2n} b_{4n-k} - \rho^{4n} b_{6n-k} + \cdots\right) \\
&\approx b_k - \rho^{2(n-k)} b_{2n-k} = b_k - \rho^{2(n-k)} b_{n+(n-k)}, \quad k = 1, 2, ..., n-1.
\end{aligned}$$

Numériquement, nous ne pouvons pas mettre ρ très petit car c_k/ρ^k ne peut pas être calculé pour k très large. Si ρ est très proche de 1, $\widehat{b}_k$ peut être une bonne approximation de b_k pour k petit car $\rho^{2(n-k)}$ est très proche de zéro. De plus, $\widehat{b}_k$ n'est pas une bonne approximation de b_k pour k proche de n (par exemple, $\widehat{b}_{n-1} - b_{n-1} \approx -\rho^2 b_{n+1}$).

Pour confirmer les résultats du Théorème 1.4.1, Figure 1.4.1 représente les erreurs de calcul de l'inverse d'une matrice triangulaire définie par :

$$t_k = \frac{1}{(k+1)^2}, \ k = 0, 1, \ldots, 511, \ \rho = 1 \text{ et } \rho = 2^{-36/512}.$$

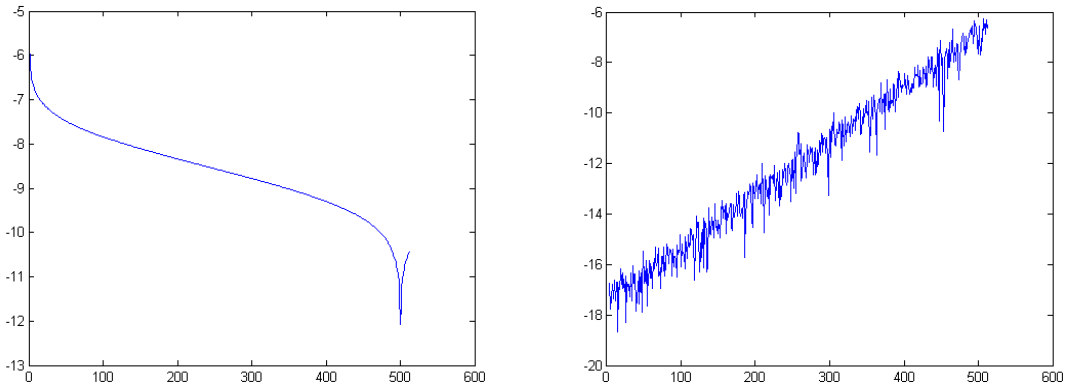


Figure 1.4.1. $\log_{10}(|\widehat{b}-b|)$ pour $t_k = 1/(k+1)^2$, $k = 0, 1, \ldots, 511$, $\rho = 1$ (figure gauche) et $\rho = 2^{-36/512}$ (figure droite).

b représente la première ligne de l'inverse de la matrice T_n obtenu par la méthode de substitution (Algorithme 1.4.1) et $\widehat{b}$ est la première ligne de l'inverse approché de la matrice T_n obtenu par la méthode d'interpolation.

Nous observons qu'avec un bon choix de ρ, la solution approchée est efficace, spécifiquement, pour k loin de n. Ce qui vérifie que les résultats numériques coïncident bien avec les résultats théoriques.

Pour améliorer l'efficacité des résultats numériques, nous pouvons interpoler $h_\rho^{(r)}$ par un polynôme trigonométrique de degré $n + n_0$, $n_0 > 0$. Dans la suite, nous remplaçons n_0 par n.

Ainsi, l'algorithme de cette méthode est le suivant :

Algorithme 1.4.1 (**Inversion via interpolation d'une matrice triangulaire de Toeplitz**) Soit $[t_j]_{j=0}^{n-1}$ la première ligne de la matrice triangulaire de Toeplitz T_n, cet algorithme permet de calculer la première ligne $\widehat{b} = \left[c_k/\rho^k\right]_{k=0}^{n-1}$ de $(T_n)^{-1}$.

1. Choisir ρ et calculer $\widetilde{t}_k = \rho^k t_k$ pour $k = 0, 1, ..., n-1$.
2. Calculer $\tau_\rho(\theta_j) = \sum_{l=0}^{n-1} \widetilde{t}_l e^{-il\theta_j}$, avec $\theta_j = (2j-1)\pi/4n$ pour $j = 0, 1, ..., n-1$.
3. Calculer $h_j = h_\rho^{(r)}(\theta_j) = \tau_\rho^{(r)}(\theta_j)/((\tau_\rho^{(r)}(\theta_j))^2 + (\tau_\rho^{(i)}(\theta_j))^2)$ pour $j = 1, ..., 2n$.
4. Résoudre $C \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{2n-1} \end{pmatrix} = \begin{pmatrix} h_0 \\ h_1 \\ \vdots \\ h_{2n} \end{pmatrix}$, avec $C = (c_{jk})_{j,k=1,2,...,n}$ tel que $c_{jk} = \cos\left(\frac{(k-1)(2j-1)\pi}{4n}\right)$, $j, k = 1, 2, ..., 2n$.
5. Calculer $\widehat{b} = \left[c_k/\rho^k\right]_{k=0}^{n-1}$.

Pour conclure cette section, nous discutons la complexité de l'Algorithme 1.4.1. En effet, on a

$$\tau_\rho(\theta_{2j}) = \sum_{l=0}^{n-1} \widetilde{t}_l e^{-il\theta_{2k}} = \sum_{l=1}^{n} \left(\widetilde{t}_l e^{-3\pi i(l-1)/4n}\right) e^{-2\pi i(l-1)(j-1)/2n}.$$

Alors, les valeurs de $\tau_\rho(\theta_{2j})$ pour $j = 1, \ldots, n$ sont calculées via FFT$(2n)$. De même, $\tau_\rho(\theta_{2j-1})$ pour $j = 1, \ldots, n$ sont calculées via FFT$(2n)$. La complexité de l'étape 4 et l'étape 5 de l'Algorithme 1.4.1 est DCT$(2n)$. D'où, la complexité de l'Algorithme 6 nécessite environ deux FFT$(2n)$ et une DCT$(2n)$. Les essais numériques montrent que $\rho = 2^{-18/n}$ est un meilleur choix.

1.4.2 Inversion via Bini d'une matrice triangulaire de Toeplitz

Pour calculer rapidement et efficacement l'inverse de la matrice triangulaire supérieure de Toeplitz, Bini [21] a proposé une méthode approchée. Soit $Z_n = [z_{jk}]_{j,k=1}^n$, avec toutes les entrées de Z_n sont nulles sauf pour $z_{j+1,j} = 1$, $j = 1, 2, ..., n-1$. Nous avons

$$T_n = \sum_{j=0}^{n-1} t_j Z_n^j.$$

L'idée de Bini est d'approximer Z_n par $Z_n^{(\varepsilon)} = \left[z_{jk}^{(\varepsilon)}\right]_{j,k=1}^n$, où $z_{jk}^{(\varepsilon)} = z_{jk}$ quand $(j,k) \neq (1,n)$ et $z_{1n}^{(\varepsilon)} = \varepsilon^n$ (ε est un petit nombre positif) et pour calculer efficacement l'inverse de

$$T_n^{(\varepsilon)} = \sum_{j=0}^{n-1} t_j \left(Z_n^{(\varepsilon)}\right)^j.$$

en utilisant la transformation de Fourier rapide.
L'inverse de $T_n^{(\varepsilon)}$ peut être calculé rapidement en utilisant la diagonalisation de Fourier

$$(T_n^{(\varepsilon)})^{-1} = (D_n^{(\varepsilon)})^{-1} F_n^* D_n^{-1} F_n D_n^{(\varepsilon)}, \tag{1.10}$$

où $D_n^{(\varepsilon)} = \text{diag}(1, \varepsilon, ..., \epsilon^{n-1})$, $D_n = \text{diag}(d)$ avec $d = \sqrt{n} F_n D_n^{(\varepsilon)} [t_j]_{j=0}^{n-1}$ et F_n est la matrice de Fourier de taille $n \times n$.

Voici l'Algorithme de Bini:

Algorithme 1.4.2 (**Inversion via Bini d'une matrice triangulaire de Toeplitz**) Soit $[t_j]_{j=0}^{n-1}$ la dernière colonne de la matrice triangulaire de Toeplitz T_n et soit $\varepsilon \in (0;1)$, cet algorithme permet de calculer la dernière colonne $\widetilde{\mu}^{(\varepsilon)} = \left[\mu_j^{(\varepsilon)}\right]_{j=0}^{n-1}$ de $(T_n^{(\varepsilon)})^{-1}$.

1. Calculer $\widetilde{t}_j = t_j \varepsilon^j$ pour $j = 0, 1, ..., n-1$
2. Calculer $d = (\sqrt{n} F_n) \left[\widetilde{t}_j\right]_{j=0}^{n-1}$
3. Calculer $d = [c_j]_{j=0}^{n-1} = [1/d_j]_{j=0}^{n-1}$
4. Calculer $f = (F_n^*/\sqrt{n})c$
5. Calculer $[b_j]_{j=0}^{n-1} = [f_j/\varepsilon^j]_{j=0}^{n-1}$.

Il n'est pas difficile de vérifier que pour $k = 0, 1, \ldots, n-1$,

$$\sum_{j=0}^{\infty} \left(b_j^{(\epsilon)} \epsilon^j \right) e^{-2\pi i j k/n} = \left(\sum_{j=0}^{\infty} \left(t_j \epsilon^j \right) e^{-2\pi i j k/n} \right)^{-1} = \frac{1}{\tau_\epsilon \left(2\pi k/n \right)}, \tag{1.11}$$

avec τ_ϵ est définie par (1.6). Ceci nous permet de conclure que l'Algorithme de Bini est équivalent à interpoler $1/\tau_\epsilon$ par un polynôme d'interpolation trigonométrique de degré inférieur à n avec $\theta_k = 2k\pi/n$, $k = 1, \ldots, n$ sont les points d'interpolation. Similairement du Théorème 5, nous proposons le théorème suivant concernant l'erreur obtenu par $b^{(\epsilon)}$.

Théorème 1.4.2 Soient $\sum_{j=0}^{\infty} \left(b_j \epsilon^j \right) z^j$ le développement en série de Maclaurin de $\tau_n^{-1}(\epsilon z)$ et $\epsilon \in \,]0, 1[$ tel que $\sum_{j=0}^{\infty} |b_j \epsilon^j| < \infty$. Soit $b^{(\epsilon)}$ est la première ligne de l'inverse approché de la matrice triangulaire de Toeplitz obtenue via l'Algorithme de Bini. Alors $b_j^{(\epsilon)} - b_j = O\left(\epsilon^n\right)$, $j = 0, 1, \ldots, n-1$. C'est-à-dire,

$$b_j^{(\varepsilon)} = b_j + \varepsilon^n \sum_{k=1}^{\infty} \varepsilon^{(k-1)n} b_{j+kn}, \quad j = 0, 1, ..., n-1. \tag{1.12}$$

Preuve. La preuve est similaire au Théorème 1.4.1. ■

Théoriquement, lorsque ε est très petit, l'inverse approché est plus précis. D'autre part, si ε est proche de zéro, $D_n^{(\varepsilon)}$ sera très mal-conditionnée pour n grand. Par conséquent, nous devons choisir un ε approprié.

Nous illustrons l'effet des erreurs d'arrondis par un simple exemple. Soit T_n la matrice triangulaire inférieure de Toeplitz définie par sa dernière colonne :

$$t_0 = 1, \; t_1 = -1, \; t_j = 0 \text{ pour } j = 2, \ldots, n-1.$$

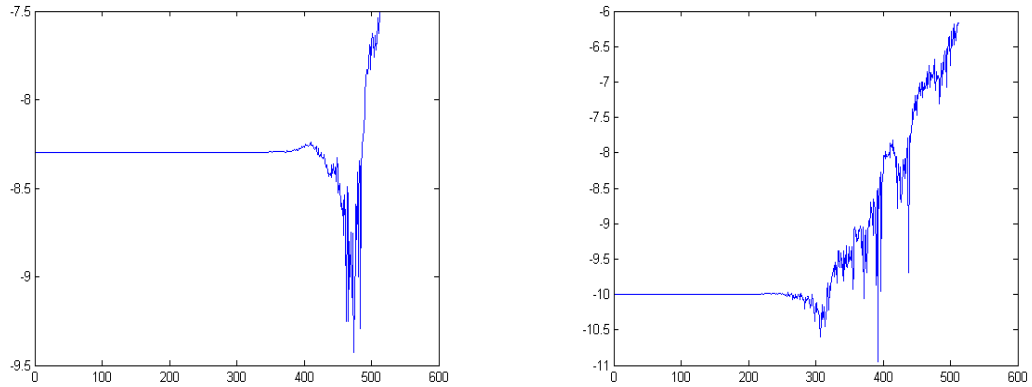


Figure 1.4.2. $\log_{10}\left(\left| \widetilde{b}^{(\varepsilon)} - b \right| \right)$ pour $t_0 = 1$, $t_1 = -1$, $t_j = 0$ avec $j = 2, 3, \ldots, 511$, $\epsilon = (0.5 \times 10^{-8})^{\frac{1}{n}}$ (figure gauche) et $\epsilon = (1.0 \times 10^{-10})^{\frac{1}{n}}$ (figure droite).

Ainsi, la dernière colonne de $(T_n)^{-1}$ est donnée par $b_j = 1$ pour $j = 0, 1, \ldots, n-1$ et la dernière colonne de $(T_n^{(\varepsilon)})^{-1}$ est donnée par $b_j^{(\varepsilon)} = \frac{1}{(1-\epsilon^n)}$ pour $j = 0, 1, \ldots, n-1$.

C'est-à-dire, $b_j^{(\varepsilon)} - b_j = \frac{\epsilon^n}{(1-\epsilon^n)} \approx \epsilon^n$ pour tout j. Les résultats numériques des erreurs de $\widetilde{b}^{(\varepsilon)}$ de l'algorithme de Bini pour $\epsilon = (0.5 \times 10^{-8})^{\frac{1}{n}}$ et $\epsilon = (1.0 \times 10^{-10})^{\frac{1}{n}}$ sont représentées dans la Figure 1.

D'après la Figure 1.4.2, nous montrons que $\widetilde{b}^{(\varepsilon)}$ est moins précis pour j proche de n.

1.4.3 Inversion via Bini révisée d'une matrice triangulaire de Toeplitz

En se basant sur les discussions ci-dessus, Lin, Ching et Ng [60] ont révisé l'algorithme de Bini pour obtenir une inversion approchée plus précise en incluant la matrice triangulaire de Toeplitz d'ordre n dans une autre matrice triangulaire de Toeplitz d'ordre $2n$. Ce qui donne un résultat plus précis par rapport aux inversions approchées et encore plus rapide par rapport aux inversions exactes. L'algorithme peut être énoncé comme suit :

Algorithme 1.4.3 (Inversion via Bini révisée d'une matrice triangulaire de Toeplitz) Soit $[t_j]_{j=0}^{n-1}$ la dernière colonne de la matrice triangulaire de Toeplitz T_n et soit $\varepsilon \in]0, 1[$, cet algorithme permet de calculer la dernière colonne $\widetilde{\mu}^{(\varepsilon)} = \left[\mu_j^{(\varepsilon)}\right]_{j=0}^{n-1}$ de $(T_n^{(\varepsilon)})^{-1}$.

1. Calculer $\widetilde{t}_j = t_j \varepsilon^j$ pour $j = 0, 1, ..., n-1$
2. Poser $\widetilde{t}_j = 0$ pour $j = n,\ n+1,\ \ldots, 2n-1$
3. Calculer $d = (\sqrt{n} F_{2n}) \left[\widetilde{t}_j\right]_{j=0}^{2n-1}$
4. Calculer $c = [c_j]_{j=0}^{2n-1} = [1/d_j]_{j=0}^{2n-1}$
5. Calculer $f = (F_{2n}^*/\sqrt{2n})c$
6. Calculer $[b_j]_{j=0}^{n-1} = [f_j/\varepsilon^j]_{j=0}^{n-1}$.

Il est évident que la complexité de l'algorithme de Bini révisé soit environ deux FFT($2n$). Les essais numériques montrent que $\epsilon = (0.5 \times 10^{-8})^{\frac{1}{n}}$ et $\epsilon = 10^{-\frac{5}{n}}$ sont des meilleurs choix pour l'algorithme de Bini et l'algorithme de Bini révisé, respectivement.

Remarque 1.4.3 L'algorithme de Bini peut être déduit à partir des algorithmes [65, 66] basés sur la division polynomiale. Nous décrivons brièvement leur idée. On définit deux polynômes

$$T(z) = \sum_{j=0}^{n-1} t_{n-1-j} z^j \text{ et } B(z) = \sum_{j=0}^{n-1} b_{n-1-j} z^j,$$

associé à la première ligne de T_n et $(T_n)^{-1}$, respectivement. On peut vérifier facilement que

$$R(z) = z^{2n-2} - T(z)\,B(z)$$

est un polynôme de degré inférieur à $n-1$. Ainsi, $B(z)$ et $R(z)$ sont le polynôme quotient et le polynôme reste de la division de deux polynômes, z^{2n-2} par $T(z)$.

1.5 Exemples numériques

Dans cette section, nous examinons l'efficacité et l'exactitude de l'algorithme de Bini, l'algorithme de Bini révisé et l'algorithme via Interpolation, pour six séquences différentes des matrices triangulaires inférieures de Toeplitz :

(i) $t_j = \frac{1}{(j+1)^3}$, $j = 0, \ldots, n-1$ (Table 1.5.1),

(ii) $t_j = \frac{1}{(j+1)^2}$, $j = 0, \ldots, n-1$ (Table 1.5.2),

(iii) $t_j = \frac{1}{(j+1)}$, $j = 0, \ldots, n-1$ (Table 1.5.3),

(iv) $t_j = \frac{1}{\log(j+2)}$, $j = 0, \ldots, n-1$ (Table 4),

(v) $t_0 = 1$, $t_1 = 1$, $t_j = 0$ pour $j = 2, \ldots, n-1$ (Table 1.5.5),

(vi) $t_0 = 1$, $t_1 = -2$, $t_2 = 1$, $t_j = 0$ pour $j = 3, \ldots, n-1$ (Table 1.5.6).

Nous notons que les séquences de (1)−(iv) sont bien-conditionnées (pour $n = 4096$, le conditionnement de ces séquences sont 1.4, 2.3, 16.8 et 799.5, respectivement), la séquence (v) a comme inverse $\mu_j = (-1)^j$ et la séquence (vi) a comme inverse $\mu_j = j+1$, $j = 0, \ldots, n-1$. Dans les tables suivantes, nous estimons l'erreur relative de l'inverse approché

$$\frac{\|\widetilde{\mu} - \mu\|_1}{\|\mu\|_1},$$

où $\widetilde{\mu}$ est l'inverse approché de la dernière colonne et μ est l'inverse exact de la dernière colonne. La deuxième ligne et la troisième ligne des tables représentent l'exactitude des inverses calculés par l'algorithme de Bini et l'algorithme de Bini révisé, respectivement. Pour les séquences de (i)−(iv), les inverses exacts sont inconnus, ainsi nous pouvons juste placer μ à la première colonne de l'inverse calculé au moyen de l'Algorithme 1.3.1.

Table 1.5.1.

n	**128**	**256**	**512**	**1024**	**2048**	**4096**
Bini	1.0e-8	1.9e-8	2.5e-8	3.8e-8	5.0e-8	7.0e-8
Bini révisée	4.5e-12	9.0e-12	1.2e-11	1.9e-11	3.3e-11	4.6e-11
Interpolation	2.7e-11	3.1e-11	4.6e-11	7.5e-11	1.2e-10	1.6e-10

Table 1.5.2.

n	**128**	**256**	**512**	**1024**	**2048**	**4096**
Bini	9.5e-9	1.4e-8	2.5e-8	3.3e-8	5.6e-8	7.6e-8
Bini révisée	5.5e-12	8.5e-12	1.3e-11	2.1e-11	3.1e-11	4.2e-11
Interpolation	2.3e-11	2.6e-11	4.4e-11	7.2e-11	8.9e-11	1.5e-10

Table 1.5.3.

n	**128**	**256**	**512**	**1024**	**2048**	**4096**
Bini	5.0e-9	2.0e-8	2.9e-8	4.2e-8	5.6e-8	8.2e-8
Bini révisée	7.3e-12	1.2e-12	1.7e-11	2.3e-11	3.3e-11	4.7e-11
Interpolation	2.0e-11	3.7e-11	4.8e-11	6.6e-11	9.8e-11	1.6e-10

Table 1.5.4.

n	**128**	**256**	**512**	**1024**	**2048**	**4096**
Bini	1.5e-8	2.1e-8	3.5e-8	6.4e-8	1.1e-7	1.9e-7
Bini révisée	7.2e-12	1.8e-11	3.4e-11	4.8e-11	7.4e-11	1.3e-11
Interpolation	2.2e-11	3.7e-11	7.6e-11	1.1e-10	1.7e-10	3.4e-10

Table 1.5.5.

n	**128**	**256**	**512**	**1024**	**2048**	**4096**
Bini	5.2e-9	5.7e-9	5.3e-9	5.7e-9	5.9e-9	9.2e-9
Bini	1.0e-10	1.0e-10	1.0e-10	1.0e-10	1.0e-10	9.7e-11
Interpolation	1.4e-12	7.0e-12	5.8e-12	8.9e-12	1.1e-11	7.4e-11

Table 1.5.6.

n	**128**	**256**	**512**	**1024**	**2048**	**4096**
Bini	1.4e-8	1.5e-8	1.7e-8	2.3e-8	6.0e-8	1.1e-7
Bini révisée	5.0e-10	5.0e-10	5.0e-10	4.7e-10	4.0e-10	9.2e-10
Interpolation	6.8e-12	1.7e-11	2.8e-11	1.0e-10	9.7e-10	4.0e-9

1.6 Conclusion

Nous avons proposé une étude comparative étendue afin de calculer l'inverse d'une matrice triangulaire de Toeplitz basé sur la transformée de Fourier rapide (FFT). Nous remarquons que l'algorithme de Bini révisé et l'algorithme via Interpolation sont plus précis par rapport à l'algorithme de Bini. De plus, l'algorithme de Bini révisé nécessite deux FFT($2n$) (environ deux exécutions de l'algorithme de Bini et $\frac{4}{5}$ de l'algorithme via Interpolation). Ainsi, l'approche via Bini révisée est un résultat à haute précision et à faible coût (plus précis par rapport aux inversions approchées et plus rapide par rapport aux inversions exactes).

Le prochain objectif consiste à répondre à la problématique principale de recherche, à savoir la diagonalisation approchée de la matrice de Hankel. Nous réaliserons ce travail basé sur les techniques d'inversion rapide des matrices triangulaires de Toeplitz dans le Chapitre 2.

Chapitre 2

Diagonalisation par blocs de la matrice de Hankel

2.1 Introduction

Dans ce chapitre, nous considérons un algorithme de diagonalisation par blocs approchée d'une matrice carrée réelle de Hankel H dans laquelle la matrice de passage A est triangulaire supérieure. Cette diagonalisation nous donne une matrice diagonale par blocs

$$D = {}^tAHA = \begin{pmatrix} lH_1 & & & \\ & lH_2 & & \\ & & \ddots & \\ & & & lH_n \end{pmatrix} \tag{2.1}$$

où chaque bloc lH_j, pour $j = 1, 2, ..., n$ est une matrice triangulaire inférieure de Hankel.

Ce problème a été étudié par Phillips [67], Rissanen [68], Kung [57], Gragg et Lindquist [46], Pal et Kailath [64], Bini et Pan [25], Bultheel et Van Barel [28] et récemment par Ben Atti et Diaz-Toca [20].

Phillips [67] et Rissanen [68] sont les premiers à traiter ce problème. Ensuite, Kung [57] ainsi que Gragg et Lindquist [46] ont considéré (2.1) dans le contexte des problèmes de réalisation partielle. Plusieurs relations entre les matrices de Hankel et les réalisations partielles ont été abordées dans [50, 56, 46]. Ces techniques sont en connexion directe avec la théorie des moments [3], qui a des applications dans divers domaines, y compris la géométrie algorithmique [44]. De plus, Pal et Kailath [64] ont considéré la factorisation des matrices de Hankel (2.1) comme une application particulière : le calcul du "rank profil" et de l'inertie [61]. Récemment, Ben Atti et Diaz-Toca [20] ont présenté l'algorithme (2.1), qui donne une preuve algébrique simple du théorème de Frobenius, lequel donne la signature d'une matrice de Hankel réelle à partir des signes des mineurs principaux dominants.

Bien que les algorithmes cités ci-dessus semblent être rapides, ils présentent quelques problèmes d'efficacités numériques. En effet, quand nous perturbons la matrice de Hankel, la diagonalisation par blocs donne des blocs 1×1 au lieu des blocs $m \times m$;

Ainsi, nous cherchons dans un voisinage de la matrice de Hankel (qui a subie une perturbation) une matrice qui a une diagonalisation par blocs capable de se faire avec des matrices de passage triangulaire de déterminant pas trop grand.

Pour résoudre ce problème, il est donc nécessaire d'introduire la notion de diagonalisation par blocs approchée de la matrice de Hankel. Cette diagonalisation nous donne la matrice diagonale par blocs approchée

$$D_\varepsilon = {}^tA_\epsilon H A_\epsilon = \begin{pmatrix} lH_{\epsilon_1} & \Theta_{1,2} & \cdots & \Theta_{1,n} \\ \Theta_{2,1} & lH_{\epsilon_2} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \Theta_{n-1,n} \\ \Theta_{n,1} & \cdots & \Theta_{n,n-1} & lH_{\epsilon_n} \end{pmatrix} \tag{2.2}$$

où chaque bloc lH_{ϵ_j} est une matrice triangulaire inférieure de Hankel (en ce qui concerne l'antidiagonal) approchée pour $j = 1, 2, ..., n$, chaque matrice $\Theta_{j,k}$ est très proche de la matrice nulle pour $j, k = 1, 2, ..., n$ et $j \neq k$ et A_ϵ est une matrice triangulaire supérieure approchée.

Cette nouvelle méthode est inspirée de [20] et basée sur le calcul approché de l'inverse de la matrice triangulaire supérieure de Toeplitz via les techniques du Chapitre 1 en $O(n^2)$ opérations.

Nous proposons ensuite une variante rapide de la factorisation par blocs approchée basée sur le complément de Schur [25, 45, 62] d'une matrice carrée réelle de Hankel. Une comparaison numérique a été élaborée pour mettre en évidence les avantages potentiels de la nouvelle diagonalisation par blocs approchée en ce qui concerne la méthode classique de la factorisation approchée, en termes d'efficacité numérique et en termes de coût informatique. Enfin, des résultats numériques ont illustré l'efficacité de nouvelle approche.

Il est à noter qu'une version abrégée en anglais de ce chapitre a fait l'objet d'un article intitulé << A fast method to block-diagonalize a Hankel matrix >> publié dans la revue "Numerical Algorithms", (voir [6]) et de nombreuses communications dans des congrès internationaux, (voir [15, 16, 17, 18]).

2.1.1 Exemple et Préliminaires

Dans l'exemple 2.1.1, nous introduisons la première étape d'une diagonalisation par blocs approchée de la matrice de Hankel à coefficients réels au moyen d'une matrice de changement de base Toeplitz-supérieure.

Exemple 2.1.1 Considérons la matrice de Hankel à coefficients réels d'ordre 5 suivante :

$$h = H\,(0,0,1,1,2,2,3,4,11) = \begin{pmatrix} 0 & 0 & 1 & 1 & 2 \\ 0 & 1 & 1 & 2 & 2 \\ 1 & 1 & 2 & 2 & 3 \\ 1 & 2 & 2 & 3 & 4 \\ 2 & 2 & 3 & 4 & 11 \end{pmatrix} = \left(\begin{array}{ccc|cc} 0 & 0 & 1 & 1 & 2 \\ 0 & 1 & 1 & 2 & 2 \\ 1 & 1 & 2 & 2 & 3 \\ \hline 1 & 2 & 2 & 3 & 4 \\ 2 & 2 & 3 & 4 & 11 \end{array}\right).$$

Nous perturbons h (en rajoutant, par exemple, 10^{-13} à chaque composante de h), nous obtenons :

$$\widetilde{h} = \begin{pmatrix} 10^{-13} & 10^{-13} & 1 & 1 & 2 \\ 10^{-13} & 1 & 1 & 2 & 2 \\ 1 & 1 & 2 & 2 & 3 \\ 1 & 2 & 2 & 3 & 4 \\ 2 & 2 & 3 & 4 & 11 \end{pmatrix} = \left(\begin{array}{ccc|cc} 10^{-13} & 10^{-13} & 1 & 1 & 2 \\ 10^{-13} & 1 & 1 & 2 & 2 \\ 1 & 1 & 2 & 2 & 3 \\ \hline 1 & 2 & 2 & 3 & 4 \\ 2 & 2 & 3 & 4 & 11 \end{array}\right).$$

Nous construisons à partir de h une matrice triangulaire supérieure de Toeplitz t :

$$t = uT(1,1,2,2,3) = \begin{pmatrix} 1 & 1 & 2 & 2 & 3 \\ 0 & 1 & 1 & 2 & 2 \\ 0 & 0 & 1 & 1 & 2 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} = \left(\begin{array}{ccc|cc} 1 & 1 & 2 & 2 & 3 \\ 0 & 1 & 1 & 2 & 2 \\ 0 & 0 & 1 & 1 & 2 \\ \hline 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{array}\right).$$

L'inverse de t est alors :

$$t^{-1} = uT(1,-1,-1,1,0) = \begin{pmatrix} 1 & -1 & -1 & 1 & 0 \\ 0 & 1 & -1 & -1 & 1 \\ 0 & 0 & 1 & -1 & -1 \\ 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$= \left(\begin{array}{ccc|cc} 1 & -1 & -1 & 1 & 0 \\ 0 & 1 & -1 & -1 & 1 \\ 0 & 0 & 1 & -1 & -1 \\ \hline 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 1 \end{array}\right).$$

Nous utilisons t^{-1} comme matrice de changement de base pour $\widetilde{h}$ et nous obtenons :

$$h' = \left(\begin{array}{ccc|cc} 1.000e\text{-}13 & 1.000e\text{-}13 & 1 & 0 & 9.992e\text{-}14 \\ 1.000e\text{-}13 & 1 & -1 & 1.0003e\text{-}13 & -1.0003e\text{-}13 \\ 1 & -1 & -1 & -1.0003e\text{-}14 & -9.992e\text{-}14 \\ \hline 0 & 9.992e\text{-}14 & -9.992e\text{-}14 & 9.9838e\text{-}27 & 1 \\ 9.992e\text{-}14 & -9.992e\text{-}14 & -9.992e\text{-}14 & 1 & 5 \end{array}\right).$$

Ainsi, nous obtenons une réduction de la matrice perturbée de Hankel :

- Le premier bloc diagonal est une matrice triangulaire inférieure approchée suivant l'anti-diagonale.
- Le deuxième bloc diagonal est une matrice de Hankel approchée.
- Les coefficients de h' sont déterminés de manière exacte si nous inversons la

matrice triangulaire supérieure de Toeplitz d'ordre 7 suivante :

$$T = uT\,(1,1,2,2,3,4,11) = \left(\begin{array}{ccc|cccc} 1 & 1 & 2 & 2 & 3 & 4 & 11 \\ 0 & 1 & 1 & 2 & 2 & 3 & 4 \\ 0 & 0 & 1 & 1 & 2 & 2 & 3 \\ \hline 0 & 0 & 0 & 1 & 1 & 2 & 2 \\ 0 & 0 & 0 & 0 & 1 & 1 & 2 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right).$$

Nous trouvons :

$$T^{-1} = uT\,(1,-1,-1,1,0,-1,5) = \left(\begin{array}{ccc|cccc} 1 & -1 & -1 & 1 & 0 & -1 & 5 \\ 0 & 1 & -1 & -1 & 1 & 0 & -1 \\ 0 & 0 & 1 & -1 & -1 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 & -1 & -1 & 1 \\ 0 & 0 & 0 & 0 & 1 & -1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right).$$

Nous voyons donc que les coefficients du deuxième bloc Hankel approché sont obtenus à partir des 4 derniers coefficients de la première ligne de T^{-1} à un signe $(-)$ près.

Dans la suite, nous allons montrer que l'Exemple 2.1.1 se généralise.

Lemme 2.1.1 Soient $n \in \mathbb{N}^*$, $h = H\,(h_1, ..., h_{2n-1})$ est une matrice réelle de Hankel $n \times n$. Supposons que $h_j = \varepsilon_j$ tel que $\varepsilon_j \in V(0,\mu)$ pour $j = 1, 2, ..., p-1$ et $h_p \notin V(0,\mu)$. Alors

$$h = H\,(\varepsilon_1, ..., \varepsilon_{p-1}, h_p, ..., h_{2n-1}) = \begin{pmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{pmatrix}, \tag{2.3}$$

où

$$h_{11} = lH\,(h_p, ..., h_{2p-1}) + J_p\Sigma_p, \quad h_{22} = H\,(h_{2p+1}, ..., h_{2n-1}),$$
$$h_{12} = H\,(h_{p+1}, ..., h_{n+p-1}; p; n-p), \quad h_{21} = {}^t h_{12}.$$

Nous construisons successivement à partir de h les deux matrices suivantes :

– Une matrice triangulaire inférieure (suivant l'anti-diagonale) $\mathcal{H}$ d'ordre $(2n-p)$,

$$\mathcal{H} = lH\,(h_p, ..., h_{2n-1}) = \begin{pmatrix} 0 & 0 & H_{13} \\ 0 & h_{11} & h_{12} \\ H_{31} & {}^t h_{12} & h_{22} \end{pmatrix}, \tag{2.4}$$

où $H_{31} = H_{13} = lH(h_p, ..., h_{n-1})$

– Une matrice carrée triangulaire supérieure de Toeplitz T,

$$T = J_{2n-p}\mathcal{H} = uT\,(h_p, ..., h_{2n-1}) = \begin{pmatrix} t_{11} & t_{12} & t_{13} \\ 0 & t_{22} & t_{23} \\ 0 & 0 & t_{33} \end{pmatrix}, \tag{2.5}$$

où $t_{11} = t_{33} = uT\,(h_p, ..., h_{n-1})$, $t_{22} = J_p h_{11}$, $t_{13} = J_{n-p} h_{22}$, $t_{12} = J_{n-p}{}^t h_{12}$, et $t_{23} = J_p h_{12}$.

Lemme 2.1.2 Soit T une matrice triangulaire supérieure, non singulière avec un coefficient diagonal non nul. Alors $T^{-1} = uT(\mu_1, ..., \mu_{2n-p})$ et admet la décomposition par blocs suivante :

$$T^{-1} = \begin{pmatrix} (T^{-1})_{11} & (T^{-1})_{12} & (T^{-1})_{13} \\ 0 & (T^{-1})_{22} & (T^{-1})_{23} \\ 0 & 0 & (T^{-1})_{33} \end{pmatrix} = \begin{pmatrix} t_{11}^{-1} & \widetilde{P} & Q \\ 0 & t_{22}^{-1} & P \\ 0 & 0 & t_{11}^{-1} \end{pmatrix}, \tag{2.6}$$

où

$$P = T(\mu_2, ..., \mu_n; p; n-p), \quad \widetilde{P} = J_{n-p}\,{}^tPJ_p,$$

$$t_{22}P + t_{23}t_{11}^{-1} = 0_{(p,n-p)}, \quad h_{11}P + Mt_{11}^{-1} = 0_{(p,n-p)}$$

$$t_{11}\widetilde{P} + t_{12}t_{22}^{-1} = 0_{(n-p,p)}, \quad t_{11}Q + t_{12}P + t_{13}t_{11}^{-1} = 0_{(n-p,n-p)}.$$

Preuve. Il suffit de remarquer que T est Toeplitz-supérieure avec un coefficient diagonal non nul. Elle est régulière et a pour inverse une matrice de même type. Pour les plus amples détails, voir [19]. ■

2.2 Diagonalisation par blocs approchée de la matrice de Hankel via Toeplitz

2.2.1 Réduction au moyen d'une matrice triangulaire supérieure de Toeplitz

Nous commençons la présentation d'un résultat intermédiaire qui mène à l'algorithme de diagonalisation par blocs approchée de la matrice de Hankel via Toeplitz (Algorithme 2.2.2).

Théorème 2.2.1 Soit $h = H(\varepsilon_1, ..., \varepsilon_{p-1}, h_p, ..., h_{2n-1})$, où $\varepsilon_j \in V(0, \mu)$ pour $j = 1, 2, ..., p-1$ avec $h_p \notin V(0, \mu)$ et

$$T = uT(h_p, ..., h_{2n-1}), \quad T^{-1} = uT(\mu_1, ..., \mu_{2n-p}),$$

$$t = uT(h_p, ..., h_{n+p-1}), \quad t^{-1} = uT(\mu_1, ..., \mu_n).$$

Alors

$$h' = {}^t\left(t^{-1}\right) ht^{-1} = \begin{pmatrix} h'_{11} & \epsilon' \\ {}^t(\epsilon') & h'_{22} \end{pmatrix}, \tag{2.7}$$

où

$$h'_{11} = lH(\mu_1, ..., \mu_p) + {}^t\left(t_{22}^{-1}\right) J_p\Sigma_p t_{22}^{-1}, \tag{2.8}$$

$$h'_{22} = -H(\mu_{r+2}, ..., \mu_{2p-r}) + {}^tPJ_p\Sigma_pP, \tag{2.9}$$

$$\epsilon' = {}^t\left(t_{22}^{-1}\right) J_p\Sigma_pP. \tag{2.10}$$

Preuve. Nous remarquons que t est une sous-matrice de la matrice T car

$$T = \begin{pmatrix} t_{11} & t_{12} & t_{13} \\ 0 & & t \end{pmatrix} \text{ avec } t = \begin{pmatrix} t_{22} & t_{23} \\ 0 & t_{11} \end{pmatrix}.$$

Alors,

$$\begin{aligned} {}^t\left(t^{-1}\right) h t^{-1} &= \begin{pmatrix} {}^t\left(t_{22}^{-1}\right) & 0 \\ {}^tP & {}^t\left(t_{11}^{-1}\right) \end{pmatrix} \begin{pmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{pmatrix} \begin{pmatrix} t_{22}^{-1} & P \\ 0 & t_{11}^{-1} \end{pmatrix} \\ &= \begin{pmatrix} {}^t\left(t_{22}^{-1}\right) & 0 \\ {}^tP & {}^t\left(t_{11}^{-1}\right) \end{pmatrix} \begin{pmatrix} h_{11}t_{22}^{-1} & h_{11}P + h_{12}t_{11}^{-1} \\ h_{21}t_{22}^{-1} & h_{21}P + h_{22}t_{11}^{-1} \end{pmatrix} \end{aligned}$$

et comme $TT^{-1} = I$ et $T = J\mathcal{H}$, nous avons :

$$\begin{aligned} h_{11}t_{22}^{-1} &= J_p\left(J_p h_{11}\right) t_{22}^{-1} = J_p\left(\Sigma_p + t_{22}\right) t_{22}^{-1} = J_p\Sigma_p t_{22}^{-1} + J_p, \\ h_{11}P + h_{12}t_{11}^{-1} &= J\left(\left(Jh_{11}\right)P + Jh_{12}t_{11}^{-1}\right) = J\left(\left(\Sigma_p + t_{22}\right)P + Jh_{12}t_{11}^{-1}\right) \\ &= J\left(\Sigma_p P\right) + J\left(t_{22}P + Jh_{12}t_{11}^{-1}\right) = J_p\Sigma_p P, \\ h_{21}t_{22}^{-1} &= J\left(Jh_{21}\right) t_{22}^{-1} = Jt_{12}t_{22}^{-1} = -Jt_{11}\widetilde{P}, \\ h_{21}P + h_{22}t_{11}^{-1} &= J\left(Jh_{21}P + Jh_{22}t_{11}^{-1}\right) = J\left(t_{12}P + t_{13}t_{11}^{-1}\right) = -Jt_{11}Q. \end{aligned}$$

Ainsi,

$$\begin{aligned} {}^t\left(t^{-1}\right) h t^{-1} &= \begin{pmatrix} {}^t\left(t_{22}^{-1}\right) & 0 \\ {}^tP & {}^t\left(t_{11}^{-1}\right) \end{pmatrix} \begin{pmatrix} J_p\Sigma_p t_{22}^{-1} + J_p & J_p\Sigma_p P \\ -Jt_{11}\widetilde{P} & -Jt_{11}Q \end{pmatrix} \\ &= \begin{pmatrix} {}^t\left(t_{22}^{-1}\right)\left(J_p + J_p\Sigma_p t_{22}^{-1}\right) & {}^t\left(t_{22}^{-1}\right) J_p\Sigma_p P \\ {}^tP\left(J_p\Sigma_p t_{22}^{-1} + J_p\right) - {}^t\left(t_{11}^{-1}\right) Jt_{11}\widetilde{P} & {}^tPJ_p\Sigma_p P - {}^t\left(t_{11}^{-1}\right) Jt_{11}Q \end{pmatrix}. \end{aligned}$$

Comme

$$\begin{aligned} {}^t\left(t_{22}^{-1}\right)\left(J_p + J_p\Sigma_p t_{22}^{-1}\right) &= {}^t\left(t_{22}^{-1}\right) J_p + {}^t\left(t_{22}^{-1}\right) J_p\Sigma_p t_{22}^{-1} \\ &= {}^t(t_{22}^{-1})J_p + {}^t\left(t_{22}^{-1}\right) J_p\Sigma_p t_{22}^{-1}, \\ {}^tP\left(J_p\Sigma_p t_{22}^{-1} + J_p\right) - {}^t\left(t_{11}^{-1}\right) Jt_{11}\widetilde{P} &= {}^tPJ_p - {}^t\left(t_{11}^{-1}\right) Jt_{11}\widetilde{P} + {}^tPJ_p\Sigma_p t_{22}^{-1} \\ &= \left({}^tPJ_p - Jt_{11}^{-1}t_{11}J^tPJ\right) + {}^tPJ_p\Sigma_p t_{22}^{-1} \\ &= {}^tPJ_p\Sigma_p t_{22}^{-1}, \\ {}^tPJ_p\Sigma_p P - {}^t\left(t_{11}^{-1}\right) Jt_{11}Q &= {}^tPJ_p\Sigma_p P - JQ, \end{aligned}$$

et on a alors

$${}^t\left(t^{-1}\right) h t^{-1} = \begin{pmatrix} {}^t\left(t_{22}^{-1}\right) J_p + {}^t\left(t_{22}^{-1}\right) J_p\Sigma_p t_{22}^{-1} & {}^t\left(t_{22}^{-1}\right) J_p\Sigma_p P \\ {}^tPJ_p\Sigma_p t_{22}^{-1} & -JQ + {}^tPJ_p\Sigma_p P \end{pmatrix}.$$

De plus,

$$t_{22}^{-1} = uT\left(\mu_1, ..., \mu_p\right), \quad Q = T\left(\mu_{p+2}, ..., \mu_{2n-p}\right).$$

D'où,

$${}^t\left(t_{22}^{-1}\right) J_p = uH\left(\mu_1, ..., \mu_p\right) \text{ et } -JQ = -H\left(\mu_{p+2}, ..., \mu_{2n-p}\right).$$

■

Exemple 2.2.1 Considerons la liste S suivante de taille 17 qui définit une matrice de Hankel h :

$S = [0, \frac{1}{3}, -\frac{1}{3}, -\frac{1}{9}, \frac{5}{9}, -\frac{11}{27}, -\frac{4}{9}, \frac{89}{81}, \frac{7}{81}, -\frac{584}{243}, -\frac{104}{243}, \frac{12667}{1458}, \frac{1018}{243}, -\frac{193345}{4374}, -\frac{36133}{2187}, \frac{3042241}{13122}, \frac{853193}{26244}]$.

$$h = \begin{pmatrix} 0 & \frac{1}{3} & -\frac{1}{3} & -\frac{1}{9} & \frac{5}{9} & -\frac{11}{27} & -\frac{4}{9} & \frac{89}{81} & \frac{7}{81} \\ \frac{1}{3} & -\frac{1}{3} & -\frac{1}{9} & \frac{5}{9} & -\frac{11}{27} & -\frac{4}{9} & \frac{89}{81} & \frac{7}{81} & -\frac{584}{243} \\ -\frac{1}{3} & -\frac{1}{9} & \frac{5}{9} & -\frac{11}{27} & -\frac{4}{9} & \frac{89}{81} & \frac{7}{81} & -\frac{584}{243} & -\frac{104}{243} \\ -\frac{1}{9} & \frac{5}{9} & -\frac{11}{27} & -\frac{4}{9} & \frac{89}{81} & \frac{7}{81} & -\frac{584}{243} & -\frac{104}{243} & \frac{12667}{1458} \\ \frac{5}{9} & -\frac{11}{27} & -\frac{4}{9} & \frac{89}{81} & \frac{7}{81} & -\frac{584}{243} & -\frac{104}{243} & \frac{12667}{1458} & \frac{1018}{243} \\ -\frac{11}{27} & -\frac{4}{9} & \frac{89}{81} & \frac{7}{81} & -\frac{584}{243} & -\frac{104}{243} & \frac{12667}{1458} & \frac{1018}{243} & -\frac{193345}{4374} \\ -\frac{4}{9} & \frac{89}{81} & \frac{7}{81} & -\frac{584}{243} & -\frac{104}{243} & \frac{12667}{1458} & \frac{1018}{243} & -\frac{193345}{4374} & -\frac{36133}{2187} \\ \frac{89}{81} & \frac{7}{81} & -\frac{584}{243} & -\frac{104}{243} & \frac{12667}{1458} & \frac{1018}{243} & -\frac{193345}{4374} & -\frac{36133}{2187} & \frac{3042241}{13122} \\ \frac{7}{81} & -\frac{584}{243} & -\frac{104}{243} & \frac{12667}{1458} & \frac{1018}{243} & -\frac{193345}{4374} & -\frac{36133}{2187} & \frac{3042241}{13122} & \frac{853193}{26244} \end{pmatrix}.$$

La matrice de passage exacte et la matrice diagonale par blocs exacte sont respectivement les suivantes :

$$t^{-1}_{exacte} = \begin{pmatrix} 3 & 3 & 4 & 0 & 0 & 1 & 0 & -5 & 1 \\ 0 & 3 & 3 & 4 & 0 & 0 & 1 & 0 & -5 \\ 0 & 0 & 3 & 3 & 4 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 3 & 3 & 4 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 3 & 3 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3 & 3 & 4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 3 & 3 & 4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 \end{pmatrix},$$

$$h'_{exacte} = \begin{pmatrix} \mathbf{0} & \mathbf{3} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \mathbf{3} & \mathbf{3} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{0} & \mathbf{0} & \mathbf{-1} & \mathbf{0} & \mathbf{5} & \mathbf{-1} & \mathbf{-25} \\ 0 & 0 & \mathbf{0} & \mathbf{-1} & \mathbf{0} & \mathbf{5} & \mathbf{-1} & \mathbf{-25} & \mathbf{21/2} \\ 0 & 0 & \mathbf{-1} & \mathbf{0} & \mathbf{5} & \mathbf{-1} & \mathbf{-25} & \mathbf{21/2} & \mathbf{124} \\ 0 & 0 & \mathbf{0} & \mathbf{5} & \mathbf{-1} & \mathbf{-25} & \mathbf{21/2} & \mathbf{124} & \mathbf{-81} \\ 0 & 0 & \mathbf{5} & \mathbf{-1} & \mathbf{-25} & \mathbf{21/2} & \mathbf{124} & \mathbf{-81} & \mathbf{-609} \\ 0 & 0 & \mathbf{-1} & \mathbf{-25} & \mathbf{21/2} & \mathbf{124} & \mathbf{-81} & \mathbf{-609} & \mathbf{545} \\ 0 & 0 & \mathbf{-25} & \mathbf{21/2} & \mathbf{124} & \mathbf{-81} & \mathbf{-609} & \mathbf{545} & \mathbf{11873/4} \end{pmatrix}.$$

La matrice h est perturbée (rajouter $\varepsilon = 1.0e$-13 à toutes ces composantes). Nous obtenons alors la nouvelle liste perturbée S = [0.000000000000001, 0.003333333333334, -0.003333333333332, -0.001111111111110, 0.005555555555557, -0.004074074074073, -0.004444444444443, 0.010987654320989, 0.000864197530865, -0.024032921810699, -0.004279835390946, 0.086879286694103, 0.041893004115227, -0.442032464563328, -0.165217192501142, 2.318427831123306, 0.325100213382107].

En utilisant le Théorème 2.2.1, nous obtenons les résultats suivants :

$$t^{-1} = \begin{pmatrix} 3 & 3 & 4 & -6.6e\text{-}16 & -1.3e\text{-}15 & 1 & -2.3e\text{-}15 & -5 & 1 \\ 0 & 3 & 3 & 4 & -6.6e\text{-}16 & -1.3e\text{-}15 & 1 & -2.3e\text{-}15 & -5 \\ 0 & 0 & 3 & 3 & 4 & -6.6e\text{-}16 & -1.3e\text{-}15 & 1 & -2.3e\text{-}15 \\ 0 & 0 & 0 & 3 & 3 & 4 & -6.6e\text{-}16 & -1.3e\text{-}15 & 1 \\ 0 & 0 & 0 & 0 & 3 & 3 & 4 & -6.6e\text{-}16 & -1.3e\text{-}15 \\ 0 & 0 & 0 & 0 & 0 & 3 & 3 & 4 & -6.6e\text{-}16 \\ 0 & 0 & 0 & 0 & 0 & 0 & 3 & 3 & 4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 \end{pmatrix},$$

Ainsi,

$$h' = \left(\begin{array}{cc|ccccccc} \mathbf{9e\text{-}13} & \mathbf{3} & 1.2e\text{-}12 & 0 & -8.8e\text{-}16 & 3e\text{-}13 & 0 & -1.5e\text{-}12 & 3e\text{-}13 \\ \mathbf{3} & \mathbf{3} & 1.2e\text{-}12 & -8.8e\text{-}16 & -8.8e\text{-}16 & 3e\text{-}13 & 0 & -1.5e\text{-}12 & 3e\text{-}13 \\ \hline 1.2e\text{-}12 & 1.2e\text{-}12 & \mathbf{1.6e\text{-}12} & \mathbf{1.1e\text{-}15} & \mathbf{-1} & \mathbf{4e\text{-}13} & \mathbf{5} & \mathbf{-1} & \mathbf{-25} \\ -1.6e\text{-}16 & -1.6e\text{-}16 & \mathbf{1.1e\text{-}15} & \mathbf{-1} & \mathbf{1.8e\text{-}15} & \mathbf{5} & \mathbf{-1} & \mathbf{-25} & \mathbf{21/2} \\ 0 & 0 & \mathbf{-1} & \mathbf{2.7e\text{-}15} & \mathbf{5} & \mathbf{-1} & \mathbf{-25} & \mathbf{21/2} & \mathbf{124} \\ 3e\text{-}13 & 3e\text{-}13 & \mathbf{4e\text{-}13} & \mathbf{5} & \mathbf{-1} & \mathbf{-25} & \mathbf{21/2} & \mathbf{124} & \mathbf{-81} \\ 0 & 0 & \mathbf{5} & \mathbf{-1} & \mathbf{-25} & \mathbf{21/2} & \mathbf{124} & \mathbf{-81} & \mathbf{-609} \\ -1.5e\text{-}12 & -1.5e\text{-}12 & \mathbf{-1} & \mathbf{-25} & \mathbf{21/2} & \mathbf{124} & \mathbf{-81} & \mathbf{-609} & \mathbf{545} \\ 3e\text{-}13 & 3e\text{-}13 & \mathbf{-25} & \mathbf{21/2} & \mathbf{124} & \mathbf{-81} & \mathbf{-609} & \mathbf{545} & \mathbf{11873/4} \end{array}\right).$$

Observons que la réduction converge au cas exacte (c.à.d. sans rajouter $\varepsilon = 1.0e\text{-}13$ à toutes les composantes de la matrice de Hankel):

$$h' \simeq \begin{pmatrix} lH\,(0,3,3) & 0 \\ 0 & -H(0,0,1,0,-5,1,25,-\frac{21}{2},-124,81,609,-545,-\frac{11873}{4}) \end{pmatrix}.$$

Dans cet exemple, nous utilisons l'inversion de MATLAB (qui n'est autre que la méthode de la substitution) pour calculer l'inverse de la matrice triangulaire supérieure de Toeplitz t. Pour minimiser le coût de la méthode, nous proposons de calculer cette inversion par le biais de la méthode rapide de Bini révisée (voir Chapitre 1) en $O\,(n^2)$ opérations arithmétiques. Ainsi, nous considérons l'algorithme rapide suivant pour calculer la réduction (2.7) :

Algorithme 2.2.1 (Réduction approchée via Toeplitz d'une matrice de Hankel) Soit $S = [\varepsilon_1, ..., \varepsilon_{p-1}, h_p, ..., h_{2n-1}]$ une liste où $\varepsilon_j \in V(0)$ pour $j = 1, 2, ..., p-1$ et $h_p \notin V(0, \mu)$, cet algorithme calcule la réduction approchée d'une matrice de Hankel via la matrice triangulaire supérieure de Toeplitz.

1. On définie une matrice de Hankel $h = H\,(\varepsilon_1, ..., \varepsilon_{p-1}, h_p, ..., h_{2n-1})$
2. On définie une matrice triangulaire supérieure de Toeplitz $t = uT\,(h_p, ..., h_{n+p-1})$
3. Calculer t^{-1}via l'Algorithme 1.4.3 (Bini révisée)
4. Calculer $h' = {}^t\,(t^{-1})\,ht^{-1}$

5. Poser $h'_{11} = h'(1:p, 1:p)$ et $h'_{22} = h'(p+1:n, p+1:n)$.

Pour réitérer l'Algorithme 2.2.1, nous devons avoir h'_{22} une matrice de Hankel, or $-H(\mu_{r+2}, ..., \mu_{2p-r})$ est une matrice de Hankel et ${}^tPJ_p\Sigma_pP$ est seulement une matrice symétrique. Nous imposons, alors

$$h'_{22} = -H(\mu_{r+2}, ..., \mu_{2p-r}) + \Theta \tag{2.11}$$

où Θ est une matrice de Hankel construite à partir de la première colonne et la dernière ligne de ${}^tPJ_p\Sigma_pP$.

2.2.2 Diagonalisation par blocs au moyen de matrices triangulaires supérieures

Lorsque nous réitérons les étapes de l'Algorithme 2.2.1 avec la matrice de Hankel approchée h'_{22} (2.11), nous obtenons la matrice diagonale par blocs approchée D_ε (2.2) où chaque bloc est une matrice triangulaire inférieure de Hankel et nous considérons l'agorithme suivant :

Algorithme 2.2.2 (Diagonalisation par blocs approchée de la matrice de Hankel) Soit $S = [\varepsilon_1, ..., \varepsilon_{p-1}, h_p, ..., h_{2n-1}]$ une liste où $\varepsilon_j \in V(0, \mu)$ pour $j = 1, 2, ..., p-1$ et $h_p \notin V(0, \mu)$, cet algorithme calcule la diagonalisation par blocs approchée d'une matrice de Hankel via la matrice triangulaire supérieure de Toeplitz

1. On définie une matrice de Hankel $h = H(\varepsilon_1, ..., \varepsilon_{p-1}, h_p, ..., h_{2n-1})$
2. Tanque (taille$(S) \geq 2$) faire

   On appelle l'Algorithme 2.2.1

   Si $(n > p)$

   Poser $S = [h'(p+1, p+1:n)\ h'(n, p+2:n)]$

   Fin Si

   Fin Tanque.

Exemple 2.2.2 Nous proposons d'utiliser les mêmes hypothèses de l'exemple 2.1.1. Ainsi, ce processus donne le résultat suivant (D1, D2, D3 et D4 représentent les

blocs de la diagonalisation approchée) :

```
Step 1:

D1 =

       9e-013            3
            3            3

H22 =

  1.6011e-012  1.3323e-015           -1 -6.0704e-012            5           -1
  6.6613e-016           -1 -1.2942e-011            5           -1          -25
           -1 -1.2943e-011            5           -1          -25         10.5
 -6.0698e-012            5           -1          -25         10.5          124
            5           -1          -25         10.5          124          -81
           -1          -25         10.5          124          -81         -609
          -25         10.5          124          -81         -609          545

Step 2:

D2 =

  1.6011e-012  1.3323e-015           -1
  1.3323e-015           -1 -6.0292e-010
 -1           -6.0292e-010           -5

H22 =

  3.3125e-012          0.5  5.4729e-009           -1
  0.5           5.626e-009           -1  2.6364e-008
  5.4729e-009           -1  2.6359e-008         -1.5
 -1            2.6364e-008         -1.5         10.5

Step 3:

D3 =

   1.325e-011            2
            2 -2.1892e-008

H22 =

  1.9308e-007          -14
          -14           42

Step 4:

Set

D4=

  1.9308e-007          -14
          -14           42

End STOP.
```

2.3 Diagonalisation par blocs approchée par la méthode de Schur

La méthode usuelle de Schur est l'une des plus célèbres techniques pour calculer la décomposition

$$h = LD^tL \tag{2.12}$$

d'une matrice de Hankel en $O(n^2)$ opérations ([46], [25], [28] et [73]). Dans cette partie, nous donnons une version de cette factorisation dans un point de vue "approximatif".

Avant d'énoncer la nouvelle factorisation approchée d'une matrice réelle de Hankel via le complément de Schur, nous rappelons le cas "exact" appliquée à une matrice quelconque.

Définition 2.3 Considérons une matrice partitionnée en quatre blocs

$$M = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$$

où la sous-matrice A est carrée et inversible. Le complément de Schur de A dans M, noté (M/A), est la matrice

$$(M/A) = D - CA^{-1}B.$$

Remarque 2.3 Quand la matrice M est carrée, la notion de complément de Schur est reliée à l'élimination de Gauss par blocs

$$M = \begin{pmatrix} A & B \\ C & D \end{pmatrix} = \begin{pmatrix} I & 0 \\ CA^{-1} & I \end{pmatrix} \begin{pmatrix} A & B \\ 0 & (M/A) \end{pmatrix}.$$

De plus, on a la formule de Schur

$$\det M = \det A. \det (M/A).$$

Voir [73] pour plus de détails sur le complément de Schur.

Théorème 2.3 Soit h une matrice réelle symétrique vérifiant la partition suivante

$$h = \begin{pmatrix} h_{11} & {}^th_{21} \\ h_{21} & h_{22} \end{pmatrix} \tag{2.13}$$

où

$$h_{11} = lH(h_1, ..., h_p) + J_p\Sigma_p,\ h_{21} \in \mathbb{R}^{(n-p)\times p} \text{ et } h_{22} \in \mathbb{R}^{(n-p)\times(n-p)},$$

on considère le complément de Schur de h_{11}

$$h_{sc} = h_{22} - h_{21} (lH(h_1, ..., h_p))^{-1} {}^th_{21}$$

et la matrice d'élimination

$$l = \begin{pmatrix} I_{p\times p} & 0_{p\times(n-p)} \\ h_{21} (lH(h_1, ..., h_p))^{-1} & I_{(n-p)\times(n-p)} \end{pmatrix}.$$

Alors

$$h' = l^{-1} h \left(l^{-1}\right)^t = \begin{pmatrix} h'_{11} & \epsilon' \\ {}^t(\epsilon') & h'_{sc} \end{pmatrix} \tag{2.14}$$

où

$$\begin{aligned} h'_{11} &= h_{11},\ \epsilon' = -J_p \Sigma_p{}^t \left(h_{21} \left(lH\left(h_1, ..., h_p\right)\right)^{-1}\right), \\ h'_{sc} &= h_{sc} + \left(h_{21} \left(lH\left(h_1, ..., h_p\right)\right)^{-1}\right) J_p \Sigma_p{}^t \left(h_{21} \left(lH\left(h_1, ..., h_p\right)\right)^{-1}\right). \end{aligned}$$

Preuve. Nous allons montrer que

$$h = l h'^t l.$$

On a

$$\begin{aligned} lh' &= \begin{pmatrix} I_{p\times p} & 0_{p\times(n-p)} \\ h_{21}\left(lH\left(h_1, ..., h_p\right)\right)^{-1} & I_{(n-p)\times(n-p)} \end{pmatrix} \begin{pmatrix} lH\left(h_1, ..., h_p\right) + J_p\Sigma_p & \epsilon' \\ \left(\epsilon'\right)^t & h'_{sc} \end{pmatrix} \\ &= \begin{pmatrix} lH\left(h_1, ..., h_p\right) + J_p\Sigma_p & \epsilon' \\ h_{21} + h_{21}\left(lH\left(h_1, ..., h_p\right)\right)^{-1} J_p\Sigma_p + {}^t(\epsilon') & h_{21}\left(lH\left(h_1, ..., h_p\right)\right)^{-1}\epsilon' + h'_{sc} \end{pmatrix} \\ &= \begin{pmatrix} lH\left(h_1, ..., h_p\right) + J_p\Sigma_p & \epsilon' \\ h_{21} & h_{21}\left(lH\left(h_1, ..., h_p\right)\right)^{-1}\epsilon' + h'_{sc} \end{pmatrix} \\ &= \begin{pmatrix} lH\left(h_1, ..., h_p\right) + J_p\Sigma_p & \epsilon' \\ h_{21} & h_{sc} \end{pmatrix}. \end{aligned}$$

Ainsi

$$\begin{aligned} \left(lh'\right){}^t l &= \begin{pmatrix} h_{11} & \epsilon' \\ h_{21} & h_{sc} \end{pmatrix} \begin{pmatrix} I_{p\times p} & \left(\left(lH\left(h_1, ..., h_p\right)\right)^{-1}\right)^t h_{21}^t \\ 0_{(n-p)\times p} & I_{(n-p)\times(n-p)} \end{pmatrix} \\ &= \begin{pmatrix} h_{11} & {}^t h_{21} + J_p\Sigma_p{}^t \left(h_{21}\left(lH\left(h_1, ..., h_p\right)\right)^{-1}\right){}^t h_{21} + \epsilon' \\ h_{21} & h_{21}{}^t \left(\left(lH\left(h_1, ..., h_p\right)\right)^{-1}\right){}^t h_{21} + h_{21}\left(lH\left(h_1, ..., h_p\right)\right)^{-1}\epsilon' + h'_{sc} \end{pmatrix} \\ &= \begin{pmatrix} h_{11} & {}^t h_{21} \\ h_{21} & h_{21}{}^t \left(\left(lH\left(h_1, ..., h_p\right)\right)^{-1}\right){}^t h_{21} + h'_{sc} \end{pmatrix}. \end{aligned}$$

■

Exemple 2.3 Nous considérons les mêmes hypothèses de l'exemple 2.1.1. Alors, le résultat suivant a été élaboré

$$h_{11} = \begin{pmatrix} 1e\text{-}13 & \frac{1}{3} \\ \frac{1}{3} & -\frac{1}{3} \end{pmatrix},$$

$$h_{22} = \begin{pmatrix} \frac{5}{9} & -\frac{11}{27} & -\frac{4}{9} & \frac{89}{81} & \frac{7}{81} & -\frac{584}{243} & -\frac{104}{243} \\ -\frac{11}{27} & -\frac{4}{9} & \frac{89}{81} & \frac{7}{81} & -\frac{584}{243} & -\frac{104}{243} & \frac{12667}{1458} \\ -\frac{4}{9} & \frac{89}{81} & \frac{7}{81} & -\frac{584}{243} & -\frac{104}{243} & \frac{12667}{1458} & \frac{1018}{243} \\ \frac{89}{81} & \frac{7}{81} & -\frac{584}{243} & -\frac{104}{243} & \frac{12667}{1458} & \frac{1018}{243} & -\frac{193345}{4374} \\ \frac{7}{81} & -\frac{584}{243} & -\frac{104}{243} & \frac{12667}{1458} & \frac{1018}{243} & -\frac{193345}{4374} & -\frac{36133}{2187} \\ -\frac{584}{243} & -\frac{104}{243} & \frac{12667}{1458} & \frac{1018}{243} & -\frac{193345}{4374} & -\frac{36133}{2187} & \frac{3042241}{13122} \\ -\frac{104}{243} & \frac{12667}{1458} & \frac{1018}{243} & -\frac{193345}{4374} & -\frac{36133}{2187} & \frac{3042241}{13122} & \frac{853193}{26244} \end{pmatrix},$$

et

$$h_{12} = h_{21}^t = \begin{pmatrix} -\frac{1}{3} & -\frac{1}{9} & \frac{5}{9} & -\frac{11}{27} & -\frac{4}{9} & \frac{89}{81} & \frac{7}{81} \\ -\frac{1}{9} & \frac{5}{9} & -\frac{11}{27} & -\frac{4}{9} & \frac{89}{81} & \frac{7}{81} & -\frac{584}{243} \end{pmatrix}.$$

Ainsi,

$$h_{sc} = \begin{pmatrix} 0 & 5.5e\text{-}17 & -0.11111 & 0.11111 & 0.59259 & -0.85185 & -2.716 \\ 5.5e\text{-}17 & -0.11111 & 0.22222 & 0.48148 & -1.4444 & -1.8642 & 7.7716 \\ -0.11111 & 0.22222 & 0.51852 & -1.4815 & -2.0617 & 8.0556 & 8.1564 \\ 0.11111 & 0.48148 & -1.4815 & -2.0123 & 8.8951 & 7.1029 & -46.92 \\ 0.59259 & -1.4444 & -2.0617 & 8.8951 & 6.5267 & -46.245 & -19.896 \\ -0.85185 & -1.8642 & 8.0556 & 7.1029 & -46.245 & -20.713 & 239.46 \\ -2.716 & 7.7716 & 8.1564 & -46.92 & -19.896 & 239.46 & 33.734 \end{pmatrix},$$

$$l = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1.3333 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1.3333 & -0.33333 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0.44444 & 1.6667 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ -2.5556 & -1.2222 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1.963 & -1.3333 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 3.5556 & 3.2963 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ -6.9506 & 0.25926 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix},$$

et

$$h'_{11} = \begin{pmatrix} \mathbf{1e}\text{-}13 & \frac{\mathbf{1}}{\mathbf{3}} \\ \frac{\mathbf{1}}{\mathbf{3}} & -\frac{\mathbf{1}}{\mathbf{3}} \end{pmatrix},$$

$$\epsilon' = \begin{pmatrix} 1.3e\text{-}13 & -1.3e\text{-}13 & -4.4e\text{-}14 & 2.6e\text{-}13 & -2e\text{-}13 & -3.6e\text{-}13 & 7e\text{-}13 \\ -1.1e\text{-}16 & 2.3e\text{-}17 & 4.4e\text{-}16 & 2.8e\text{-}16 & -2.7e\text{-}15 & -7.5e\text{-}16 & 1.3e\text{-}14 \end{pmatrix},$$

$$h'_{sc} = \begin{pmatrix} \mathbf{1.8e\text{-}13} & \mathbf{-1.8e\text{-}13} & \mathbf{-0.11111} & 0.11111 & 0.59259 & -0.85185 & -2.716 \\ \mathbf{-1.8e\text{-}13} & \mathbf{-0.11111} & \mathbf{0.22222} & 0.48148 & -1.4444 & -1.8642 & 7.7716 \\ \mathbf{-0.11111} & \mathbf{0.22222} & \mathbf{0.51852} & -1.4815 & -2.0617 & 8.0556 & 8.1564 \\ 0.11111 & 0.48148 & -1.4815 & -2.0123 & 8.8951 & 7.1029 & -46.92 \\ 0.59259 & -1.4444 & -2.0617 & 8.8951 & 6.5267 & -46.245 & -19.896 \\ -0.85185 & -1.8642 & 8.0556 & 7.1029 & -46.245 & -20.713 & 239.46 \\ -2.716 & 7.7716 & 8.1564 & -46.92 & -19.896 & 239.46 & 33.734 \end{pmatrix}.$$

Cet exemple utilise l'inversion de MATLAB pour calculer l'inverse de la matrice triangulaire inférieure de Hankel $lH(h_1, ..., h_p)$. Afin de minimiser le coût de cette méthode, nous proposons de calculer cette inversion par le biais de la méthode rapide de Bini révisée (voir Chapitre 1) en $O(n^2)$ opérations arithmétiques en utilisant $(lH(h_1, ..., h_p))^{-1} = (uT(h_1, ..., h_p))^{-1} J_p$. Ainsi, nous considérons l'algorithme rapide suivant pour calculer la diagonalisation par blocs :

Algorithme 2.3 (Diagonalisation par blocs approchée via Schur pour une matrice de Hankel) Soit h une matrice symétrique définie comme (2.13), cet algorithme calcule la diagonalisation par blocs approchée de h via Schur.

1. Définir une matrice symétrique h vérifiant (2.13)
2. Poser $lh_{11} = lH\left(h_1, ..., h_p\right)$ et $ut_{11} = uT\left(h_1, ..., h_p\right)$
3. On appelle l'Algorithme 1.4.3 (Bini révisée) pour calculer $\left(ut_{11}\right)^{-1}$
4. Calculer $\left(lh_{11}\right)^{-1} = \left(ut_{11}\right)^{-1} J_p$
5. Poser $h_{11} = h(1:p, 1:p)$, $h_{21} = h(p+1:n, 1:p)$ and $h_{22} = h(p+1:n, p+1:n)$
6. Calculer $h_{sc} = h_{22} - h_{21}\left(lh_{11}\right)^{-1\,t} h_{21}$
7. Poser $l = \left[I\left(p,p\right), h_{21}\left(lh_{11}\right)^{-1}; 0\left(n-p,p\right), I\left(n-p,n-p\right)\right]$
8. Calculer $h' = l^{-1} h^t \left(l^{-1}\right)$
9. Poser $h'_{11} = h'(1:p, 1:p)$ et $h = h'_{sc} = h'(p+1:n, p+1:n)$ et revenir à l'étape **1.**

Exemple 2.4 Nous proposons d'utiliser de nouveau l'exemple 2.1.1. Ainsi, le nouveau processus via Schur affiche le résultat suivant (D1, D2, D3 et D4 représentent les blocs de la diagonalisation approchée) :

```
Step 1:
D1 =

            1e-013      0.33333
          0.33333     -0.33333
Sc1 =

       -1.9784e-013 -6.5892e-014     -0.11111      0.11111      0.59259     -0.85185       -2.716
       -6.5892e-014     -0.11111      0.22222      0.48148      -1.4444      -1.8642       7.7716
           -0.11111      0.22222      0.51852      -1.4815      -2.0617       8.0556       8.1564
            0.11111      0.48148      -1.4815      -2.0123       8.8951       7.1029       -46.92
            0.59259      -1.4444      -2.0617       8.8951       6.5267      -46.245      -19.896
           -0.85185      -1.8642       8.0556       7.1029      -46.245      -20.713       239.46
             -2.716       7.7716       8.1564       -46.92      -19.896       239.46       33.734
Step 2:
D2 =

       -1.9784e-013 -6.5892e-014     -0.11111
       -6.5892e-014     -0.11111      0.22222
           -0.11111      0.22222      0.51852
Sc2 =

        1.3899e-011     0.055556    -0.055556     -0.40741
           0.055556     -0.11111     -0.35185      0.94444
          -0.055556     -0.35185      0.81481       2.0617
           -0.40741      0.94444       2.0617      -5.9321
Step 3:
D3 =

        1.3899e-011     0.055556
           0.055556     -0.11111
Sc3 =

       -8.5885e-010     -0.38889
           -0.38889       1.9444
Step 4:
D4 =

       -8.5885e-010     -0.38889
           -0.38889       1.9444
```

2.4 Comparaison

Nous clarifions la corrélation entre la méthode approchée classique et la nouvelle méthode de diagonalisation par blocs approchée.

ourA =

3	3	-4	-1.6381e-012	-20	6	-1.0659e-010	8	1.0278e-007
0	3	-3	-4	-15	-34	6	-71	8
0	0	-3	-3	-19	-24	-34	-45	-71
0	0	0	-3	-3	-38	-24	-110	-45
0	0	0	0	-3	-6	-38	-36	-110
0	0	0	0	0	-6	-6	-50	-36
0	0	0	0	0	0	-6	-6	-50
0	0	0	0	0	0	0	-6	-6
0	0	0	0	0	0	0	0	-6

Aschur =

1	0	-1.3333	1.3333	0.44444	-2.5556	1.963	3.5556	-6.9506
0	1	-1	-0.33333	1.6667	-1.2222	-1.3333	3.2963	0.25926
0	0	1	0	0	-4	-1.6667	27.5	-1.4444
0	0	0	1	0	-6.3333	2.3333	32.111	-21.056
0	0	0	0	1	-1	-5.3333	7.6667	24.444
0	0	0	0	0	1	0	-8.3333	2.3333
0	0	0	0	0	0	1	-1	-7.3333
0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	1

ourD =

9e-013	**3**	-1.199e-012	1.0214e-014	-6.0334e-012	-1.3545e-012	2.7812e-011	2.2001e-010	1.8433e-009
3	**3**	-1.1884e-012	-2.8422e-014	-7.6088e-012	2.6461e-011	2.5171e-010	2.0634e-009	1.1777e-007
-1.1984e-012	-1.188e-012	**1.6002e-012**	**1.5543e-015**	**-1**	-3.1894e-012	-1.2079e-013	-6.418e-012	3.3623e-011
9.6589e-015	-2.8977e-014	**2.4425e-015**	**-1**	**-7.8981e-013**	-1.2079e-013	-1.6112e-012	3.3623e-011	9.2884e-010
-6.0318e-012	-7.6052e-012	**-1**	**-8.0336e-013**	**-5**	-1.7647e-011	3.3158e-011	8.9656e-010	-3.2657e-009
-1.3496e-012	2.6461e-011	-3.2006e-012	-1.2368e-013	-1.7618e-011	**3.8418e-012**	**2**	-1.5824e-010	1.5469e-010
2.7816e-011	2.517e-010	-1.3345e-013	-1.6123e-012	3.3203e-011	**2**	**-3.2741e-010**	1.5487e-010	-4.22e-010
2.2001e-010	2.0634e-009	-6.3889e-012	3.3629e-011	8.9694e-010	-1.5831e-010	1.5591e-010	**8.2989e-012**	**-14**
1.8434e-009	1.1777e-007	3.3608e-011	9.2885e-010	-3.2656e-009	1.5553e-010	-4.2163e-010	**-14**	**42**

Dschur =

1e-013	**0.33333**	1.3334e-013	-1.3334e-013	-4.4409e-014	-9.9754e-014	1.0014e-013	-1.3212e-013	1.3106e-013
0.33333	**-0.33333**	1.9784e-013	6.5947e-014	-3.2979e-013	1.1212e-012	-1.3194e-012	2.3411e-012	-2.6037e-012
1.3338e-013	1.9778e-013	**1.775e-013**	**-1.7712e-013**	**-0.11111**	-2.2149e-013	-1.3618e-012	-5.2913e-013	-2.5704e-012
-1.3335e-013	6.6026e-014	**-1.7757e-013**	**-0.11111**	**0.22222**	-1.9854e-012	1.3456e-013	-5.9903e-012	2.4727e-012
-4.4368e-014	-3.2987e-013	**-0.11111**	**0.22222**	**0.51852**	-1.4504e-012	4.1767e-012	-1.5881e-012	8.9599e-012
-9.9951e-014	1.1215e-012	-2.1952e-013	-1.9862e-012	-1.4555e-012	**1.0655e-013**	**0.055556**	7.671e-011	-2.0956e-011
9.9917e-014	-1.319e-012	-1.3626e-012	1.3483e-013	4.1835e-012	**0.055556**	**-0.11111**	8.0302e-012	3.4204e-011
-1.3478e-013	2.3421e-012	-5.2657e-013	-6.0009e-012	-1.6001e-012	7.6692e-011	8.0635e-012	**1.742e-013**	**-0.38889**
1.3438e-013	-2.6064e-012	-2.5753e-012	2.4763e-012	8.9862e-012	-2.0953e-011	3.4203e-011	**-0.38889**	**1.9444**

Une étude comparative de l'Algorithme via Toeplitz par rapport à l'Algorithme via Schur a été établie :

- Dans les étapes intermédiaires des deux approches, la dimension et la structure inférieure de Hankel du bloc ont été conservées.
- Dans la première étape des deux algorithmes, le bloc h'_{11} défini dans (2.11) et (2.14) de la méthode via Toeplitz est la première sous-matrice principale de h de déterminant non nul et le bloc h'_{11} de la méthode via Schur est l'inverse de la première sous-matrice principale de h de déterminant non nul.
- Les deux approches nécessitent le calcul de la matrice de passage, définie par (2.2), pour obtenir la matrice diagonale par blocs.
- Dans les étapes intermédiaires, la méthode via Toeplitz calcule la réduction de h par une seule inversion de la matrice triangulaire de Toeplitz comparée de deux inversions, dans les étapes 3 et 8 de l'Algorithme via Schur, de la matrice triangulaire de Toeplitz. Ainsi, la méthode via Toeplitz est plus rapide que la méthode via Schur.
- Dans [46], nous trouvons un résultat affirmant qu'une matrice de Hankel est LU-équivalente à une matrice diagonale par blocs si les matrices triangulaires supérieures comme en (2.2), ont des 1 sur les diagonales principales. Ainsi, la matrice triangulaire supérieure définie comme en (2.2) est LU-équivalente "approchée" à une matrice diagonale par blocs.
- Puis, nous faisons la même observation que dans le papier [20]. En effet, nous pouvons stocker des uns dans les diagonales des matrices triangulaires supérieures (2.2) en les multipliant par $\mathrm{diag}(((A_\epsilon)_{11})^{-1}, ((A_\epsilon)_{22})^{-1}, \ldots, ((A_\epsilon)_{nn})^{-1})$ où $(A_\epsilon)_{jj}$ sont les entrées sur la diagonale principale A_ϵ définie comme en (2.2). Ainsi, le fait d'avoir des uns sur la diagonale mène à introduire une nouvelle définition d'une approche LU-équivalence "approchée".

2.5 Exemples numériques

Dans cette section, nous proposons de mettre en évidence les avantages potentiels de notre diagonalisation par blocs approchée en ce qui concerne la méthode classique de factorisation approchée, en termes d'efficacité numérique et en termes de temps.

- Tous les algorithmes ont été implémentés sur MATLAB 7.0.4.287 (R2007A) et ont été exécutés sur un ordinateur portable Intel(R) Core(TM)2 CPU T5600 avec un processeur 1.83 GHz et 2046 Mb de RAM.
- $\varepsilon = (1.0 \times 10^{-10})^{\frac{1}{n}}$ et $\varepsilon = (0.5 \times 10^{-8})^{\frac{1}{p}}$ sont les meilleurs choix (voir Chapitre 1) pour l'Algorithme révisé de Bini de l'approximation de Toeplitz et l'approximation de Schur, respectivement.
- Dans les exemples, nous introduisons la matrice de Hankel à partir d'une perturbation (rajouter $k \times 10^{-13}$ où k est choisi aléatoirement dans $[-1, 1]$ pour tous les coefficients de la matrice de Hankel "exacte"), dont les blocs (de différentes tailles) de la matrice de Hankel "exacte" sont connues.
- L'erreur relative de la méthode via le complément de Schur et la méthode via

Toeplitz sont définis, respectivement par :

$$Error_{Schur\ approchée} = \frac{\|D_{Schur\ approchée} - D_{Schur\ exacte}\|_1}{\|D_{Schur\ exacte}\|_1}, \tag{2.15}$$

et

$$Erreur_{Toeplitz\ approchée} = \frac{\|D_{Toeplitz\ approchée} - D_{Toeplitz\ exacte}\|_1}{\|D_{Toeplitz\ exacte}\|_1}. \tag{2.16}$$

2.5.1 Efficacité

Nous présentons quelques résultats numériques illustrant l'efficacité de l'algorithme via Toeplitz en ce qui concerne l'algorithme via Schur pour 7 exemples de matrices de Hankel comme en (2.3). En effet, nous dévoilons les erreurs relatives de notre algorithme (2.15) et l'algorithme de complément de Schur (2.16) respectivement. Les résultats de ces expériences sont donnés dans les tableaux suivants :

n	$\|D_{\textbf{Toeplitz exacte}}\|_1$	$\|D_{\textbf{Toeplitz approchée}}\|_1$	$\textbf{Error}_{\textbf{Toeplitz approchée}}$
9	56	56.00000000626214	1.119218685983936e-10
12	11	10.99999999999135	1.695132023798473e-8
25	6	6.00000000000213	6.869035939271882e-6
38	6	6.00000000001302	4.416662500537031e-6
54	22	22.00000000003274	3.557926557557788e-5
75	7	7.00000000078459	2.632791201223558e-7
100	7	7.00000000113276	2.694290060522763e-7

n	$\|D_{\textbf{Schur exacte}}\|_1$	$\|D_{\textbf{Schur approchée}}\|_1$	$\textbf{Error}_{\textbf{Schur approchée}}$
9	2.33333333333327	2.33333333299368	4.578777833076956e-10
12	0.56250000000000	0.56250000000000	5.640961041485247e-8
25	0.87695312500000	0.87696422784116	1.258481465590000e-3
38	0.62500000000000	0.62500000108309	9.125857869527466e-5
54	14.86111111111165	14.88540173365934	1.634509180780000e-3
75	0.95312500000000	0.95312500001764	8.816858555558903e-5
100	0.95312500000000	0.95312500001764	8.816858555558903e-5

Nous proposons de même de perturber les matrices de Hankel pour savoir le comportement des erreurs d'arrondi et leur effet sur le rendement. Nous considérons

$$S = [0, \frac{1}{3}, -\frac{1}{3}, -\frac{1}{9}, \frac{5}{9}, -\frac{11}{27}, -\frac{4}{9}, \frac{89}{81}, \frac{7}{81}, -\frac{584}{243}, -\frac{104}{243}, \frac{12667}{1458}, \frac{1018}{243}, \\ -\frac{193345}{4374}, -\frac{36133}{2187}, \frac{3042241}{13122}, \frac{853193}{26244}]$$

et nous rajoutons 10^{-k} pour toutes les entrées de matrice de Hankel avec $k = 7, \ldots, 27$. Nous donnons alors le tableau suivant pour montrer l'efficacité de l'algorithme via Toeplitz et de l'algorithme via le complément de Schur.

k	$\mathbf{Error}_{\mathbf{Sc\ approximate}}$	$\mathbf{Error}_{\mathbf{Our\ approximate}}$
8	1.2824e-005	6.9946e-005
9	1.4636e-005	7.0289e-006
10	1.5323e-005	7.2427e-007
11	1.5390e-005	6.1916e-008
12	1.5399e-005	2.9813e-008
13	1.5399e-005	4.2168e-008
14	1.5398e-005	2.0506e-008
15	1.5398e-005	2.5140e-008
16	1.5398e-005	2.2178e-008
17, . . . , **27**	1.5399e-005	2.6585e-008

Les matrices par blocs exactes via Toeplitz et par l'intermédiaire du complément de Schur ont été traitées via le Théorème 2.1 et le Théorème 2.2 respectivement pour $\varepsilon_j = 0$.

Ainsi,

$$D_{Toep\ exacte} = \begin{pmatrix} \mathbf{0} & \mathbf{3} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \mathbf{3} & \mathbf{3} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{0} & \mathbf{0} & \mathbf{-1} & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{0} & \mathbf{-1} & \mathbf{0} & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{-1} & \mathbf{0} & \mathbf{-5} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathbf{0} & \mathbf{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathbf{2} & \mathbf{0} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{0} & \mathbf{-14} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{-14} & \mathbf{42} \end{pmatrix},$$

et

$$D_{Schur\ exacte} = \begin{pmatrix} \mathbf{0} & \mathbf{\frac{1}{3}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \mathbf{\frac{1}{3}} & \mathbf{\frac{1}{3}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{0} & \mathbf{0} & \mathbf{-\frac{1}{9}} & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{0} & \mathbf{-\frac{1}{9}} & \mathbf{\frac{2}{9}} & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{-\frac{1}{9}} & \mathbf{\frac{2}{9}} & \mathbf{\frac{14}{27}} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathbf{0} & \mathbf{\frac{1}{18}} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathbf{\frac{1}{18}} & \mathbf{-\frac{1}{9}} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{0} & \mathbf{-\frac{7}{18}} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{-\frac{7}{18}} & \mathbf{\frac{35}{18}} \end{pmatrix}.$$

Dans l'exemple ci-dessus, nous remarquons que les deux algorithmes sont efficaces à partir d'un certain rang $k = 8$ (pour $k = 1, \ldots, 7$ nous ne gardons pas la diagonalisation par blocs de la matrice de Hankel).

Par la suite, nous proposons de donner une étude conséquente afin de prouver l'efficacité de la diagonalisation par blocs approchée des matrices de Hankel. Ainsi, nous confirmons "numériquement" que les deux approches sont efficaces et la perturbation ci-dessus préserve le résultat à partir d'une certaine tolérance additive.

1. $n = 12$

k	$\mathbf{Error_{Sc\ approximate}}$	$\mathbf{Error_{Our\ approximate}}$
8	0.028008	5.2062e-005
9	0.0024594	5.1776e-006
10	0.00024071	5.0560e-007
11	2.8203e-005	3.2538e-008
12	3.2080e-006	2.5582e-009
13	1.8040e-007	3.3921e-008
14	6.6724e-007	2.7802e-009
15	4.2572e-007	5.3467e-009
16	1.9281e-007	2.4491e-009
17	5.6468e-008	2.1702e-008
18, . . . , **27**	5.6410e-008	2.1706e-008

2. $n = 25$

k	$\mathbf{Error_{Sc\ approximate}}$	$\mathbf{Error_{Our\ approximate}}$
8	1.0615	9.9076
9	1.1322	14.366
10	0.22833	0.056658
11	0.006935	0.0057348
12	0.0043216	0.00057443
13	0.0030606	5.7743e-005
14	0.00106	6.4048e-006
15	0.00019381	1.1168e-006
16	0.0002444	1.0467e-006
17	0.001273	1.1245e-006
18	0.0012589	1.0913e-006
19	0.001257	1.0783e-006
20, . . . , **27**	0.0012585	1.0786e-006

3. $n = 54$

k	$\mathbf{Error_{Sc\ approximate}}$	$\mathbf{Error_{Our\ approximate}}$
8	0.96872	1.3969
9	1.1848	0.81142
10	0.028001	0.19966
11	0.0022468	0.015705
12	0.0013586	0.0015361
13	0.0007464	0.00015355
14	0.0006854	1.5148e-005
15	0.0001409	1.3452e-006
16	0.00053803	4.1316e-007
17	0.0011497	3.8677e-007
18	0.0016119	2.6140e-007
19	0.0016121	2.0456e-007
20	0.001608	2.0311e-007
21	0.0016333	2.0358e-007
22	0.0016333	2.0344e-007
23	0.0016345	2.0333e-006
24, . . . , **27**	0.0016345	2.0357e-006

4. $n = 75$

k	$\mathbf{Error_{Sc\ approximate}}$	$\mathbf{Error_{Our\ approximate}}$
8	5.5232	0.094507
9	0.23497	0.008876
10	0.00047249	0.00088254
11	4.55850e-005	8.8144e-005
12	1.18120e-005	8.7425e-006
13	1.15270e-005	1.0422e-006
14	8.62350e-006	4.0980e-007
15	3.96300e-006	4.3134e-007
16	4.36230e-006	2.3560e-007
17	1.03582e-005	5.7684e-007
18	9.38140e-006	6.5964e-007
19	6.77500e-006	6.3389e-007
20	6.77210e-006	6.3579e-007
21	6.77210e-006	6.3523e-007
22, . . . , **27**	6.77210e-006	6.3678e-007

5. $n = 100$

k	$\mathbf{Error}_{\mathbf{Sc\ approximate}}$	$\mathbf{Error}_{\mathbf{Our\ approximate}}$
8	0.55409	0.038271
9	417.42	0.004102
10	0.00042123	0.00041318
11	8.3883e-005	4.1261e-005
12	0.00012432	3.6762e-006
13	0.00014097	1.0226e-006
14	0.00011841	4.5262e-007
15	5.7383e-005	1.0547e-006
16	5.7170e-005	7.4818e-007
17	0.00014047	7.6256e-007
18	0.00012606	4.7645e-007
19, . . . , **27**	8.8169e-005	5.7163e-007

2.5.2 Temps de calcul

Il est clair que les deux méthodes de diagonalisation par blocs nécessitent $O(n^2)$ opérations. Par conséquent, il s'avère intéressant de savoir le temps d'exécution nécessaire pour chaque méthode. Dans le tableau suivant, nous examinons les temps de calculs (en sec) de notre diagonalisation approchée par blocs et de la méthode approchée de complément de Schur pour 7 tailles différentes de matrices de Hankel.

n	$\mathbf{Temps}_{\mathbf{Schur\ approchée}}$	$\mathbf{Temps}_{\mathbf{Our\ approchée}}$
9	0.015353	0.008857
12	0.010140	0.005759
25	0.008062	0.006055
54	0.017775	0.009857
75	0.019114	0.017255
100	0.034030	0.029891
200	0.102005	0.108442
512	1.296836	1.076396
750	4.356359	3.311806
1024	9.074938	7.480839

Ainsi, nous observons que le diagonalisation classique par blocs de complément de Schur nécessite plus de temps que l'approche via Toeplitz.

2.6 Conclusion

Nous avons tenté d'exposer dans ces quelques pages, le concept central de cette thèse, à savoir la diagonalisation par blocs approchée de la matrice de Hankel à coefficients réels. Nous avons comparé la nouvelle approche par la méthode usuelle du

complément de Schur. Des résultats numériques ont montrés l'efficacité et la rapidité des algorithmes.

Une application de la diagonalisation par blocs approchée de la matrice de Hankel à coefficients réels à l'algorithme d'Euclide sera réalisée dans le Chapitre 3.

Chapitre 3

Diagonalisation par blocs de la matrice de Hankel et algorithme d'Euclide

Soient

$$u(x) = \sum_{k=0}^{n} u_k x^k \text{ et } v(x) = \sum_{k=0}^{m} v_k x^k \tag{3.1}$$

deux polynômes dans $\mathbb{R}[x]$ avec $\deg(u(x)) = n$ et $\deg(v(x)) = m$ et $m < n$. L'algorithme d'Euclide classique appliqué à $u(x)$ et $v(x)$ génère une suite des quotients $\{q_k(x)\}$ et des restes $\{r_k(x)\}$, de la manière suivante :

$$\begin{array}{l} r_{-1}(x) = u(x), \quad r_0(x) = v(x) \\ r_{k-2}(x) = r_{k-1}(x)\, q_k(x) - r_k(x), \quad \deg(r_k(x)) < \deg(r_{k-1}(x)) \quad k = 1, \ldots, K \end{array} \tag{3.2}$$

où $-r_k(x)$ est le reste de la division de $r_{k-2}(x)$ par $r_{k-1}(x)$ et $r_K(x)$ est le Plus Grand Commun Diviseur (PGCD) de $u(x)$ et $v(x)$.

La question du calcul de la suite des quotients et des restes a été étudiée par Habicht, Collins et Brown (voir [47, 34, 31, 32]) via des variantes de l'algorithme d'Euclide. Dans ces algorithmes, l'objectif est de préserver tous les calculs dans l'anneau des coefficients des polynômes $u(x)$ et $v(x)$, tout en maîtrisant la taille des objets intermédiaires. Naturellement, l'algorithme classique s'avère inefficace lorsqu'il est appliqué à des polynômes de départ perturbés. Afin d'éviter ce problème, Noda et Sasaki (voir [71, 59]) sont parmi les premiers à introduire des versions numériques efficaces de l'algorithme d'Euclide.

Plusieurs articles ont été consacrés à décrire la relation naturelle entre l'algorithme d'Euclide classique et la factorisation par blocs des matrices de Hankel. En effet, Bini et Gemignani (voir [25, 22, 43, 23]) ont calculé les coefficients des polynômes (3.2) via une approche révisée en se basant sur la factorisation LU par blocs (voir [46, 49]) de la matrice de Hankel $H(u, v)$, associée à $u(x)$ et $v(x)$. Récemment, Ben Atti et Diaz-Toca [39] ont calculé tous les coefficients des polynômes (3.2) via une approche différente de la version classique basée sur la factorisation LU par blocs de la matrice de Hankel $H(u, v)$ [20].

Naturellement, la belle relation entre l'algorithme d'Euclide classique et la diagonalisation par blocs LU des matrices de Hankel $H(u,v)$ semble être égarée quand les polynômes de départ (3.1) sont perturbés. A vrai dire, on se heurte à un problème de conditionnement. C'est pourquoi une nouvelle notion de diagonalisation par blocs approchée de la matrice de Hankel $H(u,v)$ est introduite dans ce chapitre en s'inspirant des techniques de [6]. Nous proposons aussi un algorithme révisé qui permet de préserver la belle relation entre l'algorithme d'Euclide classique et la factorisation par blocs des matrices de Hankel en générant la suite des quotients et des restes. Enfin, nous illustrons l'approche au moyen de toeplitz par un exemple.

Nous tenons à signaler enfin qu'une version abrégée en anglais de ce chapitre a fait l'objet d'un article intitulé << Block factorization of a Hankel matrix and Euclidean Algorithm >> accepté pour publication dans la revue "Mathematical Modelling of Natural Phenomena", (voir [7]) et de nombreuses communications dans des congrès internationaux, (voir [12, 13, 14]).

3.1 Matrice de Hankel associée à deux polynômes

$u(x) = u_n x^n + u_{n-1}x^{n-1} + \cdots + u_1 x + u_0$ et $v(x) = v_m x^m + v_{m-1}x^{m-1} + \cdots + v_1 x + v_0$ sont deux polynômes dans $\mathbb{R}[x]$ de degré n et m, respectivement, telle que $0 < m < n$.

La série formelle $R(x)$ associée à la fonction $\frac{v(x)}{u(x)}$ est donnée par :

$$R(x) = \frac{v(x)}{u(x)} = \sum_{k=0}^{\infty} h_k x^{-k}. \tag{3.3}$$

Définition 3.1 On appelle matrice de Hankel associée à deux polynômes $u(x)$ et $v(x)$, la matrice carrée d'ordre n, définie par $H(h_1, h_2, \ldots, h_{2n-1})$ et notée par $H(u,v) = (h_{i+j-1})_{1\leq i,j\leq n}$:

$$H(u,v) = \begin{pmatrix} h_1 & h_2 & \cdots & h_n \\ h_2 & h_3 & \cdots & h_{n+1} \\ \vdots & \vdots & \ddots & \vdots \\ h_n & h_{n+1} & \cdots & h_{2n-1} \end{pmatrix}.$$

Plusieurs publications ont détaillé la structure des matrices de Hankel ainsi obtenues et leurs propriétés (voir par exemple [51, 25, 62, 40, 5]).

Réciproquement, toute matrice de Hankel régulière, peut être considérée comme étant la matrice de Hankel associée à deux polynômes premiers entre eux :

Proposition 3.1 Pour toute matrice régulière de Hankel H d'ordre n, il existe deux polynômes $u(x)$ et $v(x)$ premiers entre eux, $\deg(u) = n$, $\deg(v) = m$ et $m < n$, tels que $H = H(u,v)$. Les polynômes $u(x)$ et $v(x)$ sont en relation avec H par les équations suivantes :

$$\begin{aligned} {}^tH(u_0, \ldots, u_{n-1}) &= -u_n {}^t(h_{n+1}, \ldots, h_{2n}), \\ {}^t(v_{n-1}, \ldots, v_0) &= lT(h_1, \ldots, h_n) {}^t(u_n, \ldots, u_1) \end{aligned}$$

où h_{2n} est un nombre fixé dans $\mathbb{R}$ et $lT(h_1, \ldots, h_n)$ est une matrice triangulaire inférieure de Toeplitz définie par la liste d'éléments $[h_1, \ldots, h_n]$.

Preuve. La démonstration est dans l'article [22]. ■

Ceci nous permet de conclure l'algorithme suivant :

Algorithme 3.1 (Construire la matrice de Hankel H à partir de deux polynômes) Soit $u(x)$ et $v(x)$ deux polynômes dans $\mathbb{R}[X]$ premiers entre eux, tel que $\deg(u) = n$, $\deg(v) = m$ et $0 < m < n$, cet algorithme construit la matrice de Hankel $H(u, v)$ à partir de deux polynômes.

1. On stock $u(x)$ et $v(x)$ dans deux matrices lignes de taille $1 \times n$, respectivement
$$U = [u_0, \ldots, u_{n-1}, u_n] \text{ et } V = [v_0, \ldots, v_{m-1}, v_m]$$
2. Poser $V = [V \text{ zeros}(1, n - m - 1)]$
3. Poser $U_0 = U(1)$, $U = U(2 : n)$ et $n = \text{taille}(U)$
4. Poser $H = \text{zeros}(1, 2n - 1)$ et $H(1) = V(n)/U(n)$
5. Pour $i = 2 : n$
$H(i) = (V(n - i + 1) - H(1 : i - 1)^t U(n - i + 1 : n - 1))/U(n)$
Fin.
6. Poser $U = [U_0\ U]$
7. Pour $i = 1 : n - 1$
$H(n + i) = -(H(i : i + n - 1) * {}^t U(1 : n))/U(n + 1)$
Fin.

Exemple 3.1 Soient

$$u(x) = x^4 + 13x^3 + 23x^2 + 21x - 1,$$
$$v(x) = x^2 + 12x + 10.$$

La matrice de Hankel associée à $u(x)$ et $v(x)$ est la suivante :

$$H(u, v) = \begin{pmatrix} 0 & 1 & -1 & 0 \\ 1 & -1 & 0 & 2 \\ -1 & 0 & 2 & -4 \\ 0 & 2 & -4 & 5 \end{pmatrix}.$$

Supposons qu'on connait tous les coefficients de la matrice de Hankel H et tous les coefficients du polynôme $u(x)$, l'algorithme suivant génère tous les coefficients du polynôme $v(x)$:

Algorithme 3.2 (Construire $v(x)$ à partir de $u(x)$ et la matrice de Hankel H) Soit H une matrice de Hankel et $u(x) = \sum_{k=0}^{n} u_k x^k$ un polynôme dans $\mathbb{R}[x]$ de degré n, cet algorithme génère tous les coefficients du polynôme $v(x) = \sum_{k=0}^{m} v_k x^k$.

1. Poser $H = [h_{n-m+2}, \ldots, h_{n-1}]$ et $U = [u_0, \ldots, u_{n-1}, u_n]$

2. Poser $n = \text{taille}(U)$, $U_0 = U(1)$ et $U = U(2:n)$
3. Poser $n = \text{taille}(U)$
4. Calculer $U = J_n * {}^tU$
5. Calculer $V = {}^t(J_n * lT(H(1,1:n)) * U)$.

Remarque 3.1 Considérons la matrice de Hankel inférieure :

$$H = lH\left(h_n, ..., h_{2n-1}\right).$$

La Proposition 3.1 nous permet de conclure,

$$\begin{pmatrix} v_{n-1} \\ v_{n-2} \\ \vdots \\ v_0 \end{pmatrix} = \begin{pmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \\ h_n & \cdots & 0 \end{pmatrix} \begin{pmatrix} u_n \\ u_{n-1} \\ \vdots \\ u_1 \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ h_n u_n \end{pmatrix}.$$

Ainsi, $v(x)$ est une constante de $\mathbb{R}$. Si on prend $v(x) = 1$ alors $v_0 = 1$ et $u_n = 1/h_n$. Ainsi, H représente une matrice de Hankel associée à $u(x)$ et $v(x) = 1$:

$$H = H\left(u, 1\right).$$

Il est important de signaler que les matrices de Hankel sont étroitement liées aux matrices de Bézout et ceci a été prouvé dans la littérature.

3.2 Algorithme d'Euclide approché

Rappelons que l'algorithme d'Euclide classique pour le calcul du polynôme PGCD est comme suit. Soient

$$u(x) = \sum_{k=0}^{n} u_k x^k \text{ et } v(x) = \sum_{k=0}^{m} v_k x^k$$

deux polynômes dans $\mathbb{R}[x]$ tels que $\deg(u(x)) = n$, $\deg(v(x)) = m$ et $m < n$. L'algorithme d'Euclide classique appliqué à $u(x)$ et $v(x)$ génère une suite des quotients polynomiales $\{q_k(x)\}$ et des restes polynomiales $\{r_k(x)\}$, de la manière suivante :

$$\begin{array}{l} r_{-1}(x) = u(x), \quad r_0(x) = v(x) \\ r_{k-2}(x) = r_{k-1}(x)\, q_k(x) - r_k(x), \quad \deg\left(r_k(x)\right) < \deg\left(r_{k-1}(x)\right) \quad k = 1, \ldots, K \end{array}$$

où $-r_k(x)$ est le reste de la division de $r_{k-2}(x)$ par $r_{k-1}(x)$. L'algorithme s'arrête quand le reste $r_{K+1}(x) = 0$; alors $r_K(x)$ est le Plus Grand Commun Diviseur (PGCD) de $u(x)$ et $v(x)$.

L'algorithme d'Euclide dans le cas approché peut être reformulé par la simple conversion de tous les "petits" coefficients par 0 après chaque division polynomiale (où les "petits" coefficients sont des coefficients dont la valeur absolue est plus petite qu'une tolérance fixée presque nulle). Malheureusement, cette méthode naïve est très

instable comme il a été prouvé dans [71]. Cependant, il est possible de concevoir des versions stables basées sur l'approche d'Euclide.

L'un des premiers travaux dans cette direction a été donné dans [59], où Noda et Sasaki introduisent une règle de normalisation sur les restes et calculent le PGCD approché comme suit :

Algorithme 3.3.1 (Algorithme d'Euclide approché) Soient $u(x)$ et $v(x)$ deux polynômes dans $\mathbb{R}[x]$ tels que $\deg(u(x)) = n$, $\deg(v(x)) = m$ et $m < n$, cet algorithme calcule la suite des polynômes quotients approchés et des polynômes restes approchés avec une tolérance inférieure à un petit nombre positif ϵ, $0 < \epsilon \ll 1$.

1. Calculer q_k comme le quotient de la division de r_{k-1} par r_k.
2. Poser $r_{k-1} = q_k r_k + \max\{1, \|q_k\|_\infty\} . r_{k+1}$.
3. Appliquer récursivement l'Algorithme 3.3 pour $k = 0, 1, \ldots, K$ et on s'arrête quand le polynôme reste $|r_{K+1}| < \epsilon$.

Remarque 3.3.1

Il convient de souligner que toute variante de l'algorithme d'Euclide approchée (voir, par exemple, [52, 53]) peut être utilisée. Nous proposons d'implémenter l'Algorithme 3.3 pour faire une comparaison avec la méthode au moyen de Toeplitz (voir section 3.5).

3.3 Diagonalisation par blocs approchée de $H(u, v)$

Dans le Chapitre 2, nous avons prouvé que l'Algorithme 2.2 nous permet de générer une matrice diagonale par blocs approchée D_ε telle que :

$$D_\varepsilon = {}^tA_\epsilon H A_\epsilon = \begin{pmatrix} lH_{\epsilon_1} & \Theta_{1,2} & \cdots & \Theta_{1,n} \\ \Theta_{2,1} & lH_{\epsilon_2} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \Theta_{n-1,n} \\ \Theta_{n,1} & \cdots & \Theta_{n,n-1} & lH_{\epsilon_n} \end{pmatrix}$$

où chaque bloc lH_{ϵ_j} est une matrice triangulaire inférieure de Hankel (en ce qui concerne l'antidiagonale) approchée pour $j = 1, 2, ..., n$, chaque matrice $\Theta_{j,k}$ est très proche de la matrice nulle pour $j, k = 1, 2, ..., n$ et $j \neq k$ et A_ϵ est une matrice triangulaire supérieure approchée.

3.3.1 Réduction approchée de $H(u, v)$

Nous introduisons, dans cette partie, la corrélation entre l'algorithme d'Euclide approché appliqué à deux polynômes $u(x)$ et $v(x)$ et la diagonalisation par blocs approchée de $H(u, v)$.

A partir de la Remarque 3.1 et (2.7), nous réécrivons le Théorème 2.2.1 appliqué à une matrice de Hankel associée à $u(x)$ et $v(x)$ comme le suivant :

Théorème 3.3.2 Soit $H(u,v) = H(\varepsilon_1, ..., \varepsilon_{n-m-1}, h_{n-m}, ..., h_{2n-1})$ une matrice perturbée (approchée) de Hankel associée à deux polynômes premiers entre eux dans $\mathbb{R}[x]$, $u(x) = \sum_{k=0}^{n} u_k x^k$ et $v(x) = \sum_{k=0}^{m} v_k x^k$, tels que $\deg(u(x)) = n$, $\deg(v(x)) = m$ et $m < n$. Soient

$$T = uT(h_{n-m}, ..., h_{2n-1}), \quad T^{-1} = uT(\mu_1, ..., \mu_{n-m}),$$

$$t = uT(h_{n-m}, ..., h_{2n-m-1}), \quad t^{-1} = uT(\mu_1, ..., \mu_n).$$

Alors

$$^t\left(t^{-1}\right) H(u,v)\, t^{-1} = \begin{pmatrix} \widetilde{H}(q,1) & \epsilon' \\ (\epsilon')^t & \widetilde{H}(v,r) \end{pmatrix}, \tag{3.4}$$

où

$$\widetilde{H}(q,1) = H(q,1) + {}^t\left(t_{22}^{-1}\right) J_{n-m}\Sigma_{n-m}t_{22}^{-1},$$

$$\widetilde{H}(v,r) = H(v,r) + {}^tPJ_{n-m}\Sigma_{n-m}P, \quad \epsilon' = {}^t\left(t_{22}^{-1}\right) J_{n-m}\Sigma_{n-m}P,$$

$q(x)$ et $r(x)$ sont le polynôme quotient et reste de la division de $\frac{u(x)}{v(x)}$.

Preuve. $H(u,v)$ est définie par les $(2n-1)$ premiers termes de

$$\frac{v(x)}{u(x)} = \sum_{k=0}^{\infty} h_k x^{-k}$$

et de plus $(uT(h_{n-m}, ..., h_{2n-1}))^{-1}$ est donnée par les $(n-m)$ premiers termes de

$$\frac{u(x)}{v(x)} = \mu_1 x^{n-m} + \cdots + \mu_{n-m+1} + \sum_{k=1}^{\infty} \mu_{n-m+1+k} x^{-k} = q(x) - \frac{r(x)}{v(x)}.$$

Alors,

$$\begin{aligned} h'_{11} &= lH(\mu_1, ..., \mu_{n-m}) + {}^t\left(t_{22}^{-1}\right) J_{n-m}\Sigma_{n-m}t_{22}^{-1} \text{ (Théorème 2.2.1 (Chapitre 2))} \\ &= H(q,1) + {}^t\left(t_{22}^{-1}\right) J_{n-m}\Sigma_{n-m}t_{22}^{-1} \text{ (Remarque 3.2).} \\ h'_{22} &= -H(\mu_{n-m+2}, ..., \mu_{n+m}) + {}^tPJ_{n-m}\Sigma_{n-m}P \text{ (Théorème 2.2.1 (Chapitre 2))} \\ &= H(v,r) + {}^tPJ_{n-m}\Sigma_{n-m}P. \end{aligned}$$

■

En réitérant le Théorème 3.3.2, nous obtenons l'algorithme suivant qui calcule la suite des quotients approchés et des restes approchés apparaissant au cours de l'exécution de l'Algorithme 3.3 :

Algorithme 3.3.2 (Diagonalisation par blocs approchée de $H(u,v)$) Soient $u(x) = \sum_{k=0}^{n} u_k x^k$ et $v(x) = \sum_{k=0}^{m} v_k x^k$ deux polynômes dans $\mathbb{R}[x]$ de degré n et m, respectivement, où $m < n$, cet algorithme calcule la suite des quotients et des restes approchés apparus au cours de l'exécution de l'Algorithme 3.3 avec une tolérance inférieure à un petit nombre positif ϵ, $0 < \epsilon \ll 1$.

1. On appelle l'Algorithme 3.1 pour construire $H = H(\varepsilon_1, ..., \varepsilon_{n-m-1}, h_{n-m}, ..., h_{2n-1})$ (La première étape : $\varepsilon_1 = \cdots = \varepsilon_{n-m-1} = 0$)
2. Définir la matrice triangulaire supérieure de Toeplitz

$$t = uT(h_{n-m}, ..., h_{2n-m-1}) \text{ pour } h_{n-m} >> \epsilon$$

3. Calculer t^{-1} via l'Algorithme 1.4.3 (Bini révisée).
4. Calculer $h' = {}^t(t^{-1})\, Ht^{-1}$.
5. Poser $h'_{11} = h'(1 : n-m, 1 : n-m)$ et $h'_{22} = h'(n-m+1 : n, n-m+1 : n)$.
6. On récupère tous les coefficients du polynôme quotient à partir de l'étape 3.
7. On récupère tous les coefficients du polynôme reste à partir de l'Algorithme 3.2 en utilisant les étapes 1. et 5.
8. On exécute récursivement l'Algorithme 3.3.2 avec

$$H = H\left(h'_{22}(1 : m, 1)\ \ h'_{22}(m, 1 : m)\right),$$

afin d'obtenir tous les coefficients apparus dans le processus d'Euclide.

Remarque 3.3.2 Nous observons dans le Théorème 3.3.2 que $u(x)$ et $v(x)$ sont premiers entre eux. Nous pouvons préserver la généralité sans cette hypothèse. En effet, si $r_K(x) = \epsilon\text{-PGCD}(u(x), v(x))$[1] est un polynôme de degré positif alors la suite des polynômes restes $\{r_k(x)\}$ peut être récupérée via la suite polynômiale $\{\widetilde{r}_k(x)\}$ générée de l'algorithme d'Euclide approché appliqué à $u(x)/r_k(x)$ et $v(x)/r_k(x)$.

3.4 Exemple Numérique

Nous utilisons ici l'exemple de [39].
Soient

$$u(x) = 6x^9 + 24x^8 + 44x^7 + 162x^6 + 60x^5 + 273x^4 + 32x^3 + 193x^2 - 70x - 10,$$
$$v(x) = 2x^7 + 6x^6 + 6x^5 + 40x^4 - 28x^3 + 65x^2 - 19x - 2.$$

L'Algorithme 3.1 construit la matrice de Hankel suivante : $H(u, v) = H(0, 0.33333, -0.33333, -0.11111, 0.55556, -0.40741, -0.44444, 1.0988, 0.08642, -2.4033, -0.42798, 8.6879, 4.1893, -44.203, -16.522, 231.84, 32.51)$. Nous introduisons les polynômes de départs à partir de la perturbation (rajouter $k.10^{-13}$ pour tous les coefficients des

[1] Soit p et q des polynômes de degrés respectifs n et m et soit ϵ un nombre positif. On appelle :
- ϵ-diviseur (ou diviseur approché) : tout diviseur des polynômes perturbés $\widehat{p}$ et $\widehat{q}$ vérifiant

$$\deg \widehat{p} \leqslant n,\ \deg \widehat{q} \leq m,\ \|p - \widehat{p}\| \leq \epsilon \text{ et } \|q - \widehat{q}\| \leq \epsilon.$$

- ϵ-PGCD (PGCD approché [42, 41]) : un ϵ-diviseur de degré maximum.

polynômes de départ "exacte" avec k est aléatoirement définit dans $]0,1[$) des polynômes de départ "exacts" (les polynômes quotients et les polynômes restes sont définis "exactement").

Les erreurs de la suite des quotients et des restes entre l'algorithme d'Euclide approché et la méthode de diagonalisation par blocs approchée de la matrice de Hankel sont notées respectivement par :

$$Error_Q^A = \|Q_{\text{approx EA}} - Q_{\text{Our approx}}\|_1 \text{ et } Error_R^A = \|R_{\text{approx EA}} - R_{\text{Our approx}}\|_1 .$$

De plus, les erreurs de la suite des quotients "exactes" et des restes "exactes" entre l'algorithme d'Euclide approché et la méthode de diagonalisation par blocs approchée de la matrice de Hankel sont notées respectivement par :

$$Error_Q = \|Q_{\text{exact}} - Q_{\text{Our approx}}\|_1 \text{ et } Error_R = \|R_{\text{exact}} - R_{\text{Our approx}}\|_1 .$$

Dans ce qui suit, nous montrons le parallélisme entre la diagonalisation par blocs approchée de $H(u,v)$ et l'algorithme d'Euclide approché (pour tous les étapes).
Etape 1 : Diagonalisation par blocs approchée de $H(u,v)$
$H(q_1,1) = H(0,\mathbf{3},\mathbf{3})$,
$H(v,r_1) = H(-5.6e\text{-}15,\ 1.2e\text{-}14,\ -1, 2.2e\text{-}13,\ 5,\ -1,\ -25,\ 10.5,\ 124,\ -81,\ -609,\ 545,\ 2968.2)$,
$q_1(x) = \mathbf{3}x^2 + \mathbf{3}x + \mathbf{3.9999}$,
$r_1(x) = -2.0000x^4 - 5.9999x^3 + 3.9999x^2 - 12x + 1.9999$
Etape 1 : Algorithme d'Euclide approché
$r_{-1}(x) = r_0(x)\,q_1(x) - r_1(x), q_1(x) = \mathbf{2.9999}x^2 + \mathbf{3}x + \mathbf{3.9999}$,
$r_1(x) = -2.0000x^4 - 5.9999x^3 + 3.9999x^2 - 11.9999x + 2.0000$
Etape 2 : Diagonalisation par blocs approchée de $H(u,v)$
$H(q_2,1) = H(-5.6e\text{-}15, 1.2e\text{-}14, \mathbf{-1}, \mathbf{-2.2e\text{-}13}, \mathbf{-5})$,
$H(r_1,r_2) = H(-2.3e\text{-}11,\ 0.5,\ -3.2e\text{-}10,\ -1,\ -5.3e\text{-}9,\ -1.5,\ 10.5)$,
$q_2(x) = \mathbf{-0.9999}x^3 \mathbf{-2.2019e\text{-}13}x^2 \mathbf{-4.9999}x \mathbf{+0.9999}$,
$r_2(x) = -1.0000x^2 - 3.0000x + 3.9999$
Etape 2 : Algorithme d'Euclide approché
$r_0(x) = r_1(x)\,q_2(x) - r_2(x), q_2(x) = \mathbf{-0.9999}x^3 \mathbf{+3.7334e\text{-}11}x^2 \mathbf{-4.9999}x \mathbf{+0.9999}$,
$r_2(x) = -0.9999.x^2 - 2.9999x + 3.9999$
Etape 3 : Diagonalisation par blocs approchée de $H(u,v)$
$H(q_3,1) = H(-9.5\text{-}11, \mathbf{2}, \mathbf{1.2e\text{-}9})$,
$H(r_2,r_3) = H(-2.6e\text{-}8,\ -14,\ 42)$,
$q_3(x) = \mathbf{1.9999}x^2 + \mathbf{1.2842e\text{-}9}x + \mathbf{3.9999}$, $r_3(x) = 13.9999$
Etape 3 : Algorithme d'Euclide approché
$r_1(x) = r_2(x)\,q_3(x) - r_3(x), q_3(x) = \mathbf{1.9999}x^2 - \mathbf{3.5625e\text{-}8}x + \mathbf{3.9999}$, $r_3(x) = 13.9999$
Nous présentons également dans la Table 3.1, les erreurs entre la méthode de diagonalisation par blocs approchée de la matrice de Hankel et l'algorithme d'Euclide approché et les erreurs entre la méthode de diagonalisation par blocs approchée de la matrice de Hankel et la suite des quotients "exactes" et des restes "exactes".

Table 3.1 : L'efficacité de l'algorithme de diagonalisation par blocs de la matrice de Hankel

Etapes	$Error_Q^A$	$Error_R^A$	$Error_Q$	$Error_R$
1	$4.88498130 \times 10^{-15}$	$8.54649684 \times 10^{-13}$	$4.35207425 \times 10^{-14}$	$5.87818682 \times 10^{-12}$
2	$2.36009367 \times 10^{-09}$	$4.35233378 \times 10^{-08}$	$2.30907670 \times 10^{-09}$	$4.08386090 \times 10^{-08}$
3	$6.24723485 \times 10^{-07}$	$2.14263267 \times 10^{-06}$	$7.72895518 \times 10^{-06}$	$2.64003742 \times 10^{-05}$

3.5 Conclusion

Dans ce chapitre, nous avons calculé la suite des restes approchée et la suite des quotients approchée apparues au cours de l'exécution de l'algorithme d'Euclide approché appliqué à deux polynômes, dans $\mathbb{R}[x]$, $u(x)$ et $v(x)$ de degrés n et m, respectivement, $m < n$ via une nouvelle approche basée sur les techniques du Chapitre 1.

Nous allons étudier dans le chapitre suivant un algorithme de diagonalisation par blocs approchée de la matrice de Bézout $JB(u, v)J$ via la méthode de complément de Schur afin de calculer la suite des restes approchée et la suite des quotients approchée apparues au cours de l'exécution de l'algorithme d'Euclide approché.

Chapitre 4

Diagonalisation par blocs des matrices de Hankel et de Bézout : connexion avec l'algorithme d'Euclide

Ce chapitre est consacré à l'analyse de la diagonalisation par blocs approchée des matrices de Hankel et de Bézout associées à deux polynômes $u(x)$ et $v(x)$ dans $\mathbb{R}[x]$ ($\deg(u(x)) = n$, $\deg(v(x)) = m$ et $m < n$) et de leurs connexions avec l'Algorithme d'Euclide.

En effet, Bini et Gemignani [25, 22, 43, 23] introduisent un algorithme parallèle pour la diagonalisation par blocs de la matrice de Bézout $JB(u,v)J$ et génèrent la suite des quotients et des restes apparue durant l'exécution de l'Algorithme d'Euclide appliqué à $u(x)$ et $v(x)$. Cette diagonalisation par blocs de $JB(u,v)J$ peut être obtenue à partir de la diagonalisation par blocs de $H(u,v)$ et vice versa en se basant sur la relation suivante (la preuve est dans [58] page 475) :

$$B(u,v) = B(u,1)H(u,v)B(u,1).$$

Naturellement, quand les polynômes (3.1) sont perturbés, la belle relation entre l'algorithme d'Euclide (ou la diagonalisation par blocs de $H(u,v)$) et la diagonalisation par blocs des matrices de Bézout $JB(u,v)J$ est conservée. Par conséquent, une nouvelle notion de diagonalisation par blocs approchée de la matrice de Bézout $JB(u,v)J$, inspirée des travaux de Bini et Gemignani [25, 22, 43, 23], a été introduite et sa relation avec l'algorithme d'Euclide ainsi que la diagonalisation par blocs approchée des matrices de Hankel a été vérifiée dans ce chapitre. De plus, des expérimentations numériques illustrent l'efficacité des méthodes.

Nous tenons à signaler enfin que ce chapitre a fait l'objet de nombreuses communications dans des congrès internationaux, (voir [12, 13, 14]).

4.1 Matrice de Bézout associée à deux polynômes

Définition 4.1 On appelle matrice de Bézout associée à deux polynômes $u(x)$ et $v(x)$, la matrice symétrique

$$B(u,v) = \begin{pmatrix} c_{0,0} & \cdots & c_{0,n-1} \\ \vdots & & \vdots \\ c_{n-1,0} & \cdots & c_{n-1,n-1} \end{pmatrix}, \tag{4.1}$$

où les $c_{i,j}$ sont définis par l'expression de Cayley :

$$b(x,y) = \frac{u(x)\,v(y) - u(y)\,v(x)}{x-y},$$

telle que la fraction rationnelle $b(x,y)$ est une fonction à deux variables qui s'écrit de la manière suivante :

$$b(x,y) = \sum_{i,j=0}^{n-1} c_{i,j} x^i y^j.$$

Remarque 4.1 A partir de la Remarque 3.1 ($v(x) = 1$), nous observons que la matrice $B(u,1)$ est une matrice triangulaire supérieure de Hankel :

$$B(u,1) = uH(u_1,\ldots,u_n).$$

Nous présentons par la suite quelques propriétés essentielles de la matrice de Bézout :

Théorème 4.1 Nous avons les propriétés suivantes :

1. $B(u,v)$ est une matrice symétrique d'ordre n.
2. $B(u,v) = -B(v,u)$.
3. $B(u,v)$ est linéaire par rapport à u et v :

$$B(\alpha u + \beta f, v) = \alpha B(u,v) + \beta B(f,v),$$
$$B(u, \alpha v + \beta g) = \alpha B(u,v) + \beta B(u,g),$$

pour tout f et g deux polynômes et pour tout α et β deux scalaires.

Dans la suite, nous proposons différentes représentations de la matrice de Bézout.

Proposition 4.1 (Représentation matricielle) La matrice de Bézout est représentée par :

$$B(u,v) = uH(u_1,\ldots,u_n)\,uT(v_0,\ldots,v_{n-1}) - uH(v_1,\ldots,v_n)\,uT(u_0,\ldots,u_{n-1})$$

telle que pour $m < n$ les coefficients $v_{m+1},\ldots,v_n$ sont supposés nuls.

Preuve. (Voir [58]). ■

Comme conséquence de la proposition 4.1 :

Proposition 4.2 (Formule récursive) Les composantes de la matrice de Bézout $B(u,v)$ sont récursivement calculées avec l'équation suivante :

$$b_{i,j+1} = b_{i+1,j} + u_i v_j - u_j v_i$$

tels que $b_{n+1,k} = b_{0,k} = 0$ pour tout i, j et $v_{m+1} = \cdots = v_n = 0$ (Proposition 3.1).

Dans le reste de ce paragraphe, nous dévoilons la liaison entre la matrice de Bézout et de Hankel.

Proposition 4.3 La formule qui relie les matrices de Bézout et de Hankel est la suivante :

$$B(u,v) = B(u,1) H(u,v) B(u,1).$$

Théorème 4.2 Nous avons les propriétés suivantes :

1. $\text{rang}(B(u,v)) = \text{rang}(H(u,v))$
2. $\text{sig}(B(u,v)) = \text{sig}(H(u,v))$
3. $\deg(\text{PGCD}(u,v)) = i \iff \text{rang}(H(u,v)) = n - i$
4. $B(u,1)^{-1} = H(u,1)$.

Dans ce qui suit, nous désignons par $u(x)$ et $v(x)$ deux polynômes premiers entre eux de $\mathbb{R}[x]$. Nous proposons dans la section suivante d'affirmer la relation intime entre la diagonalisation par blocs approchée de $H(u,v)$ et l'algorithme d'Euclide approché.

4.2 Diagonalisation par blocs approchée de $JB(u,v)J$

Nous explorons, dans cette section, la méthode approchée de diagonalisation par blocs de la matrice $JB(u,v)J$, basée sur les résultats de Bini et Gemignani [25, 22, 43, 23].

4.2.1 Réduction approchée de $JB(u,v)J$

La proposition suivante explique comment les polynômes générés par l'algorithme d'Euclide approché peuvent être obtenue par le biais de la méthode de complément de Schur appliquée à la matrice de permutation de Bézout approchée $JB(u,v)J$:

Théorème 4.2.1 Soit

$$u(x) = \sum_{k=0}^{n} u_k x^k \text{ et } v(x) = \sum_{k=0}^{m} v_k x^k$$

deux polynômes de $\mathbb{R}[x]$ tels que $\deg(u(x)) = n$, $\deg(v(x)) = m$ et $m < n$. Soit H une matrice de Hankel définie comme dans le Théorème 2.1 et soit $0 = m_0 < m_1 < \cdots < m_l = n$ des entiers naturels tels que $\det(H_{m_i}) \notin V(0,\mu)$ pour $i = 1, \ldots, l$ et $\det(H_k) \in V(0,\mu)$ sinon. Soient

$$P_{m_i} = (JB(u,v)J)_{m_i} \text{ et } \widetilde{P}_{m_i} = P_{m_i} + J\Sigma.$$

Alors $\det\left(\widetilde{P}_{m_i}\right) \notin V(0,\mu)$ pour $i = 1,\ldots,l$ et $\det\left(\widetilde{P}_k\right) \in V(0,\mu)$ sinon. De plus, si S_{m_i} est le complément de Schur de P_{m_i} et $\widetilde{S}_{m_i}$ est le complément de Schur de $\widetilde{P}_{m_i}$ dans $JB(u,v)J$, nous avons les relations suivantes :

$$\begin{aligned} S_{m_i} &= Y - XP_{m_i}^{-1t}X = JB(r_i, r_{i+1})J, \\ \widetilde{S}_{m_i} &= S_{m_i} + \left({}^tXP_{m_i}^{-1}\right) J\Sigma^t \left({}^tXP_{m_i}^{-1}\right), \end{aligned}$$

où $\{r_i(x)\}_{1,\ldots,l}$ est la suite des polynômes restes obtenue via l'algorithme d'Euclide appliqué à $u(x)$ et $v(x)$.

Preuve. La preuve est pratiquement la même que dans l'article [22]. ■

Théorème 4.2.2 Sous les hypothèses du Théorème 4.2.1, on a

$$\begin{aligned} JB(u,v)J &= \begin{pmatrix} \widetilde{P}_{m_i} & {}^tX \\ X & Y \end{pmatrix} = \begin{pmatrix} I & 0 \\ XP_{m_i}^{-1} & I \end{pmatrix} \begin{pmatrix} \widetilde{P}_{m_i} & \Theta \\ {}^t\Theta & \widetilde{S}_{m_i} \end{pmatrix} \begin{pmatrix} I & P_{m_i}^{-1t}X \\ 0 & I \end{pmatrix} \\ &= \begin{pmatrix} T & 0 \\ XP_{m_i}^{-1}T & I \end{pmatrix} \begin{pmatrix} JB(u^{(i)}, v^{(i)})J + J\Sigma & \widetilde{\Theta} \\ {}^t\widetilde{\Theta} & \widetilde{S}_{m_i} \end{pmatrix} \begin{pmatrix} {}^tT & {}^tTP_{m_i}^{-1t}X \\ 0 & I \end{pmatrix} \end{aligned}$$

où

$$\begin{aligned} S_{m_i} &= Y - XP_{m_i}^{-1t}X = JB(r_i, r_{i+1})J, \\ \widetilde{S}_{m_i} &= S_{m_i} + \left({}^tXP_{m_i}^{-1}\right) J\Sigma^t \left({}^tXP_{m_i}^{-1}\right), \\ T &= lT\left(u_n, \ldots, u_{n-m_i+1}\right) \left(B(u^{(i)}, 1)\right)^{-1} J, \\ \Theta &= -J\Sigma^t \left(XP_{m_i}^{-1}\right), \ \widetilde{\Theta} = T^{-1}\Theta, \end{aligned}$$

et les polynômes $u^{(i)}(x)$, $v^{(i)}(x)$ vérifient

$$H(u,v)_{m_i} = H\left(u^{(i)}, v^{(i)}\right).$$

Preuve. La preuve est déduite à partir de l'article [22] et le Théorème 4.2.1. ■

A partir du Théorème 4.2.2, nous présentons un algorithme parallèle pour une diagonalisation par blocs approchée de $JB(u,v)J$.

Comparons cette méthode avec la méthode de diagonalisation par blocs approchée de la matrice de Hankel. Nous commençons le calcul du premier bloc de matrice diagonale (une matrice de Hankel inférieure). Nous notons par $D_{Bézout}$ la matrice diagonale par blocs de $JB(u,v)J$. Soit D_{11} le premier bloc. Alors

$$m_1 = n - m, \ h_{11} = H(u,v)_{n-m} = lH\left(u_{n-m}, \ldots, u_{2n-2m-1}\right)$$

et, à partir de la Remarque 4.1, nous choisissons $v^{(1)}(x) = 1$ tel que

$$H(u,v)_{n-m} = H\left(u^{(1)}, 1\right).$$

Ainsi,

$$\begin{aligned}\widetilde{P}_{m_i} &= JB(u^{(1)}, v^{(1)})J + J\Sigma = JB(u^{(1)}, 1)J + J\Sigma \\ &= JH(u^{(1)}, 1)^{-1}J + J\Sigma = Jh_{11}^{-1}J + J\Sigma \\ &= JB(q_1, 1)J + J\Sigma.\end{aligned}$$

Donc,

$$D_{11} = JB(q_1, 1)J + J\Sigma.$$

Successivement, nous obtenons une matrice supérieure approchée $A_{Bézout}$ et $D_{Bézout}$ tels que

$${}^tA_{Bézout}JB(u, v)JA_{Bézout} = D_{Bézout}.$$

De plus, les matrices successive $B(r_i, r_{i+1})$ génèrent directement la suite des restes approchées. Finalement, nous obtenons les mêmes blocs D_{ii} pour les deux méthodes basées sur les matrices de Bézout et de Hankel, à l'exception du dernier.

Remarque 4.2.1 Nous observons dans le Théorème 4.2.2 que $u(x)$ et $v(x)$ sont premiers entre eux. Nous pouvons préserver la généralité sans cette hypothèse. En effet, si $r_K(x) = \epsilon\text{-PGCD}(u(x), v(x))$[1] est un polynôme de degré positif alors la suite des polynômes restes $\{r_k(x)\}$ peut être récupérée via la suite polynômiale $\{\widetilde{r}_k(x)\}$ générée de l'algorithme d'Euclide approché appliqué à $u(x)/r_k(x)$ et $v(x)/r_k(x)$.

4.3 Expérimentation numérique

Tous les algorithmes de ce chapitre ont été implantés sous MATLAB 7.0.4.287 (R2007A) et ont été exécutés sur un ordinateur portable Intel(R) Core(TM)2 CPU T5600 avec un processeur 1.83 GHz et 2046 Mb de RAM ainsi appliqués à plusieurs polynômes couple (générées aléatoirement) $(u(x),\ v(x))$ de degrés $(n,\ m)$, où $m < n$.

Dans les exemples, nous introduisons les polynômes de départ à partir de la perturbation (rajouter $k.10^{-13}$ pour tous les coefficients des polynômes de départ "exacte" avec k est aléatoirement défini dans $]0, 1[$) des polynômes de départ "exacte" (les polynômes quotient et les polynômes reste sont définis "exactement").

- deg : représente le degré des polynômes quotients approchés et/ou restes approchés. Ce résultat est particulièrement intéressant lorsque le degré du polynôme quotient approché et/ou le polynôme reste approché est très sensible au choix de la tolérance ϵ.
- Echec : quand l'algorithme ne reconnaît pas le degré du polynôme quotient "exacte" ou le degré du polynôme reste "exacte".

[1] Soit p et q des polynômes de degrés respectifs n et m et soit ϵ un nombre positif. On appelle :
- ϵ-diviseur (ou diviseur approché) : tout diviseur des polynômes perturbés $\widehat{p}$ et $\widehat{q}$ vérifiant

$$\deg\widehat{p} \leqslant n,\ \deg\widehat{q} \leq m,\ \|p - \widehat{p}\| \leq \epsilon \text{ et } \|q - \widehat{q}\| \leq \epsilon.$$

- ϵ-PGCD (PGCD approché [42, 41]) : un ϵ-diviseur de degré maximum.

4.3.1 Comparaison des algorithmes

Nous utilisons ici l'exemple de [39].

Soient

$$u(x) = 6x^9 + 24x^8 + 44x^7 + 162x^6 + 60x^5 + 273x^4 + 32x^3 + 193x^2 - 70x - 10,$$
$$v(x) = 2x^7 + 6x^6 + 6x^5 + 40x^4 - 28x^3 + 65x^2 - 19x - 2.$$

L'Algorithme 3.1 construit la matrice de Hankel suivante :

$$H(u,v) = \begin{pmatrix} 0 & 0 & 0.33333 & -0.11111 & 0.55556 & -0.40741 & -0.44444 & 1.0988 & 0.08642 \\ 0 & 0.33333 & -0.11111 & 0.55556 & -0.40741 & -0.44444 & 1.0988 & 0.08642 & -2.4033 \\ 0.33333 & -0.11111 & 0.55556 & -0.40741 & -0.44444 & 1.0988 & 0.08642 & -2.4033 & -0.42798 \\ -0.11111 & 0.55556 & -0.40741 & -0.44444 & 1.0988 & 0.08642 & -2.4033 & -0.42798 & 8.6879 \\ 0.55556 & -0.40741 & -0.44444 & 1.0988 & 0.08642 & -2.4033 & -0.42798 & 8.6879 & 4.1893 \\ -0.40741 & -0.44444 & 1.0988 & 0.08642 & -2.4033 & -0.42798 & 8.6879 & 4.1893 & -44.203 \\ -0.44444 & 1.0988 & 0.08642 & -2.4033 & -0.42798 & 8.6879 & 4.1893 & -44.203 & -16.522 \\ 1.0988 & 0.08642 & -2.4033 & -0.42798 & 8.6879 & 4.1893 & -44.203 & -16.522 & 231.84 \\ 0.08642 & -2.4033 & -0.42798 & 8.6879 & 4.1893 & -44.203 & -16.522 & 231.84 & 32.51 \end{pmatrix}.$$

De plus,

$$JB(u,v)J = \begin{pmatrix} 0 & 12 & 36 & 36 & 240 & -168 & 390 & -114 & -12 \\ 12 & 84 & 180 & 384 & 792 & -282 & 1446 & -468 & -48 \\ 36 & 180 & 324 & 963 & 932 & 150 & 2006 & -744 & -68 \\ 36 & 384 & 936 & 1544 & 4992 & -2722 & 8628 & -2726 & -264 \\ 240 & 792 & 932 & 4992 & -1960 & 6756 & 16 & -984 & -60 \\ -168 & -282 & 150 & -2722 & 6756 & -8908 & 9041 & -2447 & -146 \\ 390 & 1446 & 2006 & 8628 & 16 & 9041 & 5037 & -2714 & -344 \\ -114 & -468 & -744 & -2726 & -984 & -2447 & -2714 & 539 & 264 \\ -12 & -48 & -68 & -264 & -60 & -146 & -146 & 264 & -50 \end{pmatrix}.$$

Dans ce qui suit, nous présentons l'exécution des algorithmes des trois méthodes : diagonalisation par blocs approchée de $H(u,v)$ et de $JB(u,v)J$ ainsi qu'Euclide approché.

Etape 1 : Algorithme d'Euclide approché
$r_{-1}(x) = r_0(x)\,q_1(x) - r_1(x)$, avec
$q_1^E(x) = \mathbf{2.9999}x^2 + \mathbf{3}x + \mathbf{3.9999}$
$r_1^E(x) = -2.0000x^4 - 5.9999x^3 + 3.9999x^2 - 11.9999x + 2.0000$

Etape 1 : Diagonalisation par blocs approchée de $H(u,v)$

$$ {}^{t}t_1^{-1} H(u,v)\, t_1^{-1} = \begin{pmatrix} \widetilde{H}(q_1,1) & \epsilon_1 \\ {}^{t}\epsilon_1 & \widetilde{H}(v,r_1) \end{pmatrix} $$

$$ t_1^{-1} = \begin{pmatrix} 3 & 3 & 4 & 2.6096e\text{-}12 & -1.1019e\text{-}10 & 1 & -1.4518e\text{-}10 & -5 & 1 \\ 0 & 3 & 3 & 4 & 2.6096e\text{-}12 & -1.1019e\text{-}10 & 1 & -1.4518e\text{-}10 & -5 \\ 0 & 0 & 3 & 3 & 4 & 2.6096e\text{-}12 & -1.1019e\text{-}10 & 1 & -1.4518e\text{-}10 \\ 0 & 0 & 0 & 3 & 3 & 4 & 2.6096e\text{-}12 & -1.1019e\text{-}10 & 1 \\ 0 & 0 & 0 & 0 & 3 & 3 & 4 & 2.6096e\text{-}12 & -1.1019e\text{-}10 \\ 0 & 0 & 0 & 0 & 0 & 3 & 3 & 4 & 2.6096e\text{-}12 \\ 0 & 0 & 0 & 0 & 0 & 0 & 3 & 3 & 4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 \end{pmatrix}. $$

$$ \widetilde{H}(q_1,1) = \begin{pmatrix} 0 & \mathbf{3} \\ \mathbf{3} & \mathbf{3} \end{pmatrix}, $$

$$ \widetilde{H}(v,r_1) = \begin{pmatrix} -2.6e\text{-}12 & 1.1e\text{-}11 & -1 & 1.4e\text{-}10 & 5 & -1 & -25 \\ 1.1e\text{-}11 & -1 & 1.4e\text{-}10 & 5 & -1 & -25 & 21/2 \\ -1 & 1.4e\text{-}10 & 5 & -1 & -25 & 21/2 & 124 \\ 1.4e\text{-}10 & 5 & -1 & -25 & 21/2 & 124 & -81 \\ 5 & -1 & -25 & 21/2 & 124 & -81 & -609 \\ -1 & -25 & 21/2 & 124 & -81 & -609 & 545 \\ -25 & 21/2 & 124 & -81 & -609 & 545 & 2968.3 \end{pmatrix}, $$

$q_1^H(x) = \mathbf{3}x^2 + \mathbf{3}x + \mathbf{3.9999}$

$r_1^H(x) = -2.0000x^4 - 5.9999x^3 + 3.9999x^2 - 12x + 1.9999$

Etape 1 : Diagonalisation par blocs approchée de $JB(u,v)\,J$

$$ {}^{t}b_1^{-1} JB(u,v)\, J b_1^{-1} = \begin{pmatrix} \widetilde{H}(q_1,1) & \epsilon_1 \\ {}^{t}\epsilon_1 & \widetilde{H}(v,r_1) \end{pmatrix} $$

$$ b_1^{-1} = \begin{pmatrix} 2 & 6 & 6 & 40 & -28 & 65 & -19 & -2 & 6.6e\text{-}16 \\ 0 & 2 & 6 & 6 & 40 & -28 & 65 & -19 & -2 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}. $$

$$ J\widetilde{B}(q_1,1)\, J = \begin{pmatrix} 0 & \mathbf{3} \\ \mathbf{3} & \mathbf{3} \end{pmatrix}, $$

$$ J\widetilde{B}(v,r_1)\, J = \begin{pmatrix} -9.2e\text{-}9 & 7.8e\text{-}9 & -4 & -12 & 8 & -24 & 4 \\ 7.8e\text{-}9 & -4 & -24 & -28 & 1.9e\text{-}7 & -68 & 12 \\ -4 & -24 & -40 & -36 & -44 & -60 & 12 \\ -12 & -28 & -36 & -340 & 230 & -506 & 76 \\ 8 & 1.9e\text{-}7 & -44 & 230 & -228 & 298 & -68 \\ -24 & -68 & -60 & -506 & 298 & -772 & 138 \\ 4 & 12 & 12 & 76 & -68 & 138 & -62 \end{pmatrix}, $$

$q_1^B(x) = \mathbf{2.9999}x^2 + \mathbf{3.0000}x + \mathbf{3.9999}$

$r_1^B(x) = -2.0000x^4 - 5.9999x^3 + 3.9999x^2 - 11.9999x + 2.0000$

Etape 2 : Algorithme d'Euclide approché
$r_0(x) = r_1(x)\, q_2(x) - r_2(x)$, avec
$q_2^E(x) = \mathbf{0.9999}x^3 + \mathbf{3.7334e\text{-}11}x^2 - \mathbf{4.9999}x + \mathbf{0.9999}$
$r_2^E(x) = -1.0000x^2 - 2.9999x + 3.9999$

Etape 2 : Diagonalisation par blocs approchée de $H(u,v)$

$$ {}^t\left(t_2^{-1}\right) H(v,r_1)\, t_2^{-1} = \begin{pmatrix} \widetilde{H}(q_2,1) & \epsilon_2' \\ {}^t(\epsilon_2') & \widetilde{H}(r_1,r_2) \end{pmatrix} $$

$$ t_2^{-1} = \begin{pmatrix} -1 & -1.4517e\text{-}10 & -5 & 1 & 1.6254e\text{-}8 & -0.4998 & 2.5116e\text{-}7 \\ 0 & -1 & -1.4517e\text{-}10 & -5 & 1 & 1.6254e\text{-}8 & -0.4998 \\ 0 & 0 & -1 & -1.4517e\text{-}10 & -5 & 1 & 1.6254e\text{-}8 \\ 0 & 0 & 0 & -1 & -1.4517e\text{-}10 & -5 & 1 \\ 0 & 0 & 0 & 0 & -1 & -1.4517e\text{-}10 & -5 \\ 0 & 0 & 0 & 0 & 0 & -1 & -1.4517e\text{-}10 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{pmatrix}.. $$

$$ \widetilde{H}(q_2,1) = \begin{pmatrix} -2.6099e\text{-}12 & 1.1019e\text{-}11 & \mathbf{-1} \\ 1.1019e\text{-}11 & \mathbf{-1} & -1.0079e\text{-}11 \\ \mathbf{-1} & -1.0079e\text{-}11 & \mathbf{-5} \end{pmatrix}, $$

$$ \widetilde{H}(r_1,r_2) = \begin{pmatrix} -1.6367e\text{-}8 & 0.5 & -2.5107e\text{-}7 & -1 \\ 0.5 & -2.5107e\text{-}7 & -1 & -3.8538e\text{-}6 \\ -2.5107e\text{-}7 & -1 & -3.8538e\text{-}6 & -1.5 \\ -1 & -3.8538e\text{-}6 & -1.5 & 10.5 \end{pmatrix}, $$

$q_2^H(x) = -\mathbf{0.9999}x^3 - \mathbf{8.2589e\text{-}11}x^2 - \mathbf{4.9999}x + \mathbf{0.9999}$

$r_2^H(x) = -1.0000x^2 - 2.9999x + 3.9999$

Etape 2 : Diagonalisation par blocs approchée de $JB(u,v)\,J$

$$ {}^tb_2^{-1} JB(v,r_1)\, Jb_2^{-1} = \begin{pmatrix} J\widetilde{B}(q_2,1)\, J & \epsilon_1 \\ {}^t\epsilon_1 & J\widetilde{B}(r_1,r_2)\, J \end{pmatrix} $$

$$ b_2^{-1} = \begin{pmatrix} -2 & -6 & 4 & -12 & 2 & 5.6e\text{-}14 & 1.7e\text{-}15 \\ 0 & -2 & -6 & 4 & -12 & 2 & 3.1e\text{-}14 \\ 0 & 0 & -2 & -6 & 4 & -12 & 2 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}. $$

$$ J\widetilde{B}(q_2,1)\, J = \begin{pmatrix} 0 & 0 & \mathbf{-1} \\ 0 & \mathbf{-1} & -3.5e\text{-}8 \\ \mathbf{-1} & -3.5e\text{-}8 & \mathbf{-5} \end{pmatrix}, $$

$$ J\widetilde{B}(r_1,r_2)\, J = \begin{pmatrix} -1.1e\text{-}5 & 2 & 6 & -8 \\ 2 & 12 & 10 & -24 \\ 6 & 10 & -48 & 18 \\ -8 & -24 & 18 & -42 \end{pmatrix}, $$

$q_2^B(x) = -\mathbf{0.9999}x^3 - \mathbf{3.5625e\text{-}8}x^2 - \mathbf{4.9999}x + \mathbf{0.9999}$

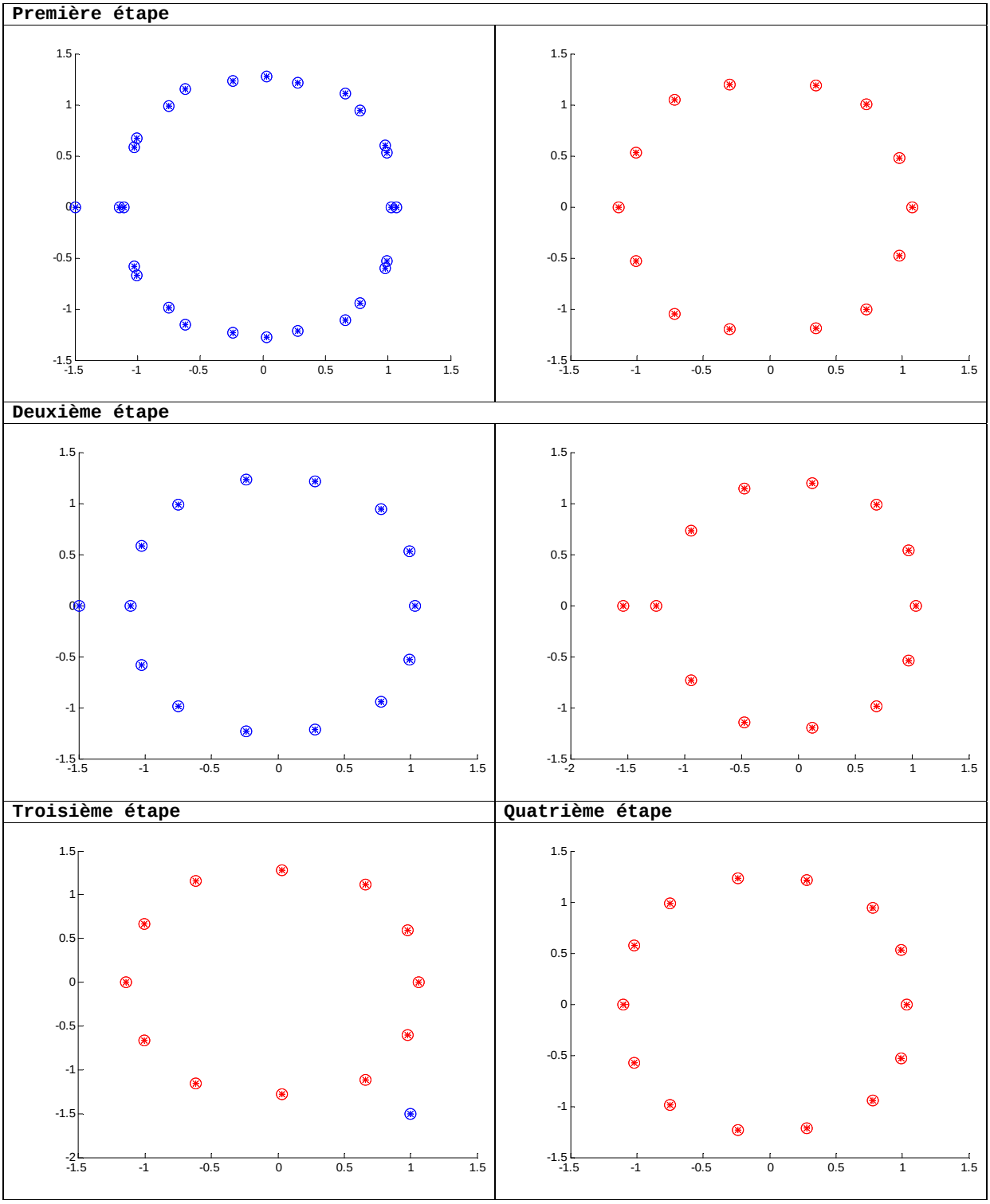
$r_2^B(x) = -1.0000x^2 - 2.9999x + 3.9999$

Etape 3 : Algorithme d'Euclide approché
$r_1(x) = r_2(x)\, q_3(x) - r_3(x)$, avec
$q_3^E(x) = \mathbf{1.9999}x^2 - \mathbf{3.5625e\text{-}8}x + \mathbf{3.9999}$
$r_3^E(x) = \textit{13.9999}$

Etape 3 : Diagonalisation par blocs approchée de $H(u,v)$
$${}^t\left(t_3^{-1}\right) H(r_1,r_2)\, t_3^{-1} = \begin{pmatrix} \widetilde{H}(q_3,1) & \epsilon_2' \\ {}^t(\epsilon_2') & \widetilde{H}(r_2,r_3) \end{pmatrix}$$
$$t_3^{-1} = \begin{pmatrix} \mathbf{2} & 1.0043e\text{-}6 & 4 & 1.9432e\text{-}5 \\ 0 & \mathbf{2} & 1.0043e\text{-}6 & 4 \\ 0 & 0 & \mathbf{2} & 1.0043e\text{-}6 \\ 0 & 0 & 0 & \mathbf{2} \end{pmatrix}.$$
$$\widetilde{H}(q_3,1) = \begin{pmatrix} -6.5468e\text{-}8 & \mathbf{2} \\ \mathbf{2} & 1.0043e\text{-}6 \end{pmatrix},$$
$$\widetilde{H}(r_2,r_3) = \begin{pmatrix} -1.9694e\text{-}5 & -14 \\ -14 & 42 \end{pmatrix},$$
$q_3^H(x) = \mathbf{1.9999}x^2 + \mathbf{6.4096e\text{-}7}x + \mathbf{3.9999}$
$r_3^H(x) = \textit{13.9999}$

Etape 3 : Diagonalisation par blocs approchée de $JB(u,v)\,J$
$${}^t b_3^{-1} JB(r_1,r_2)\, J b_3^{-1} = \begin{pmatrix} J\widetilde{B}(q_3,1)\, J & \epsilon_1 \\ {}^t\epsilon_1 & J\widetilde{B}(r_2,r_3)\, J \end{pmatrix}$$
$$b_3^{-1} = \begin{pmatrix} -1 & -3 & 4 & -3.7356e\text{-}15 \\ 0 & -1 & -3 & 4 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$
$$J\widetilde{B}(q_3,1)\, J = \begin{pmatrix} 0 & \mathbf{1.9999} \\ \mathbf{1.9999} & 4.9382e\text{-}5 \end{pmatrix}$$
$$J\widetilde{B}(r_2,r_3)\, J = \begin{pmatrix} -8.7452e\text{-}4 & -13.9992 \\ -13.9992 & -42.0007 \end{pmatrix}$$
$q_3^B(x) = \mathbf{1.9999}x^2 - \mathbf{4.9382e\text{-}5}x + \mathbf{3.9997}$
$r_3^B(x) = \textit{13.9991}$

Figure 4.3.3



°,+, × représentent les racines des polynômes quotients approchés (bleu) et des polynômes restes approchés (rouge) par l'algorithme d'Euclide, et la diagonalisation par blocs approchée de H(u,v) et de B(u,v), respectivement.

4.3.2 Sensibilité des degrés par rapport à la tolérance

Soient $u(x)$ et $v(x)$ deux polynômes, générés aléatoirement, de degrés 54 et 40, respectivement. L'algorithme d'Euclide approchée nous permet de générer la suite des restes et des quotients approchées en 4 étapes. Les degrés des polynômes quotients "exacte" $\{\deg(q_1), \deg(q_2), \deg(q_3), \deg(q_4)\}$ et les degrés des polynômes restes "exacte" $\{\deg(r_1), \deg(r_2), \deg(r_3), \deg(r_4)\}$ sont $\{14, 13, 12, 14\}$ et $\{27, 15, 1, 0\}$, respectivement. Ainsi, nous illustrons les degrés des polynômes quotients et restes approchés pour différentes tolérances ϵ.

Etape 1

ϵ	$\deg q_1^B$	$\deg q_1^H$	$\deg q_1^E$	$\deg r_1^B$	$\deg r_1^H$	$\deg r_1^E$
$10^{-2}, \ldots, 10^{-9}$	14	14	14	27	27	27
$10^{-10}, \ldots, 10^{-15}$	14	14	14	27	27	Echec
$10^{-16}, \ldots, 10^{-20}$	14	14	14	27	27	Echec

Etape 2

ϵ	$\deg q_2^B$	$\deg q_2^H$	$\deg q_2^E$	$\deg r_2^B$	$\deg r_2^H$	$\deg r_2^E$
$10^{-2}, \ldots, 10^{-8}$	13	13	13	15	15	15
$10^{-9}, \ldots, 10^{-11}$	13	13	13	15	Echec	Echec
$10^{-12}, \ldots, 10^{-20}$	13	Echec	Echec	15	Echec	Echec

Etape 3

ϵ	$\deg q_3^B$	$\deg q_3^H$	$\deg q_3^E$	$\deg r_3^B$	$\deg r_3^H$	$\deg r_3^E$
$10^{-2}, \ldots, 10^{-6}$	12	12	12	1	1	1
$10^{-7}, \ldots, 10^{-9}$	12	12	12	1	1	Echec
$10^{-10}, \ldots, 10^{-20}$	12	Echec	Echec	1	Echec	Echec

Etape 4

ϵ	$\deg q_4^B$	$\deg q_4^H$	$\deg q_4^E$	$\deg r_4^B$	$\deg r_4^H$	$\deg r_4^E$
$10^{-2}, \ldots, 10^{-6}$	14	14	14	0	0	0
$10^{-7}, 10^{-8}$	14	14	14	0	0	Echec
$10^{-9}, \ldots, 10^{-20}$	14	Echec	Echec	0	Echec	Echec

Les résultats montrent que la méthode basée sur Bézout et la méthode basée sur Hankel sont plus efficaces par rapport à la méthode d'Euclide approchée.

4.3.3 Comparaison des racines

Nous comparons les racines des polynômes quotients approchés et des polynômes restes approchés avec une tolérance $\epsilon \in]10^{-6}, 10^{-2}[$.

Figure 4.3.3 représente les racines des polynômes quotients approchés et polynômes restes approchés par les trois méthodes.

4.4 Conclusion

Dans ce chapitre, nous avons calculé la suite des restes approchés et la suite des quotients approchés apparues au cours de l'exécution de l'algorithme d'Euclide

approché appliqué à deux polynômes, dans $\mathbb{R}[x]$, $u(x)$ et $v(x)$ de degrés n et m, respectivement, $m < n$ via une nouvelle approche basée sur les techniques du Chapitre 1. Nous avons conçu aussi un algorithme de diagonalisation par blocs approchée de la matrice de Bézout $JB(u, v)J$ via la méthode de complément de Schur afin de calculer la suite des restes approchés et la suite des quotients approchés apparues au cours de l'exécution de l'algorithme d'Euclide approché.

Les résultats de ce chapitre nous permettent d'étendre l'approche de [12, 13, 14, 11, 10] dans le cas de polynômes de départ à valeurs dans $\mathbb{C}[x]$. Dans cette direction, nous obtenons la suite des restes approchée et la suite des quotients approchée apparues au cours de l'exécution de l'algorithme d'Euclide approché appliqué à deux polynômes, dans $\mathbb{C}[x]$, $u(x)$ et $v(x)$ de degrés n et m, respectivement, $m < n$. Nous réaliserons ce travail dans le Chapitre 5.

Chapitre 5

Diagonalisation par blocs de la matrice de Hankel à coefficients complexes

Une matrice $n \times n$ de Hankel est de la forme :

$$H = \begin{pmatrix} h_1 & h_2 & \cdots & h_n \\ h_2 & h_3 & \cdots & h_{n+1} \\ \vdots & \vdots & \ddots & \vdots \\ h_n & h_{n+1} & \cdots & h_{2n-1} \end{pmatrix}, \tag{5.1}$$

dont tous les éléments dans les mêmes anti-diagonale sont identiques. Cette matrice est symétrique (mais pas Hermitienne si elle est complexe). Tout au long de ce chapitre, la notation ${}^tM$ désigne la transposée de M et non le conjugué.

Les matrices de Hankel à coefficients complexes jouent un rôle important en traitement du signal [54]. En effet, les scalaires définis par (5.1) représentent un signal généré par une somme d'un nombre fini d'exponentielles r

$$h_k = \sum_{l=1}^{r} \lambda_l^k d_l, \quad k = 1, ..., 2n-1$$

où λ_l et d_l sont les sous-jacents modes et poids, respectivement. Lorsque le signal est altéré par le bruit, des matrices de Hankel "perturbées" sont produites.

Dans ce chapitre, nous introduisons le problème de diagonalisation par blocs de la matrice de Hankel "perturbée" et étudions le cas complexe. Il existe une vaste littérature sur la diagonalisation par blocs d'une matrice symétrique de Hankel ; pour certaines références, voir Phillips [67], Rissanen [68], Kung [57], Gragg et Lindquist [46], Pal et Kailath [64], Bini et Pan [25], Bultheel et Van Barel [28], Ben Atti et Diaz-Toca [20] et récemment par Belhaj [6].

Bien que l'algorithme décrit dans [6] semble être le premier à étudier le cas approché, il ne préserve pas la même taille des blocs dans le cas complexe. Par conséquent, nous reprenons ce problème dans le cas complexe.

Le nouvel algorithme est une variante du Chapitre 2. Nous comparons également l'approche de diagonalisation par blocs approchée de la matrice de Hankel par rapport à une variante basée sur le complément de Schur (voir références [46], [25] et [28] par exemple) adaptée à une matrice $n \times n$ de Hankel à coefficients complexes. Enfin, nous présentons un exemple qui permet de préserver la relation étroite entre l'algorithme d'Euclide et la méthode de diagonalisation par blocs approchée de la matrice de Hankel dans le cas complexe.

Nous tenons à signaler enfin qu'une version abrégée en anglais de ce chapitre a fait l'objet d'un article intitulé << Computing the block factorization of complex Hankel matrices >>, publié dans la revue "Computing", (voir [8]) et une communication dans la 16th International Linear Algebra Society (ILAS) Conference, (voir [9]).

5.1 Diagonalisation par blocs approchée d'une matrice de Hankel complexe via Toeplitz

Dans le Chapitre 2, nous avons montré que l'Algorithme 2.2.2 nous permet de générer une matrice diagonale par blocs approchée D_ε telle que :

$$D_\varepsilon = {}^tA_\epsilon H A_\epsilon = \begin{pmatrix} lH_{\epsilon_1} & \Theta_{1,2} & \cdots & \Theta_{1,n} \\ \Theta_{2,1} & lH_{\epsilon_2} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \Theta_{n-1,n} \\ \Theta_{n,1} & \cdots & \Theta_{n,n-1} & lH_{\epsilon_n} \end{pmatrix} \tag{5.2}$$

où chaque bloc lH_{ϵ_j} est une matrice triangulaire inférieure de Hankel (en ce qui concerne l'antidiagonal) approchée pour $j = 1, 2, ..., n$, chaque matrice $\Theta_{j,k}$ est très proche de la matrice nulle pour $j, k = 1, 2, ..., n$ et $j \neq k$ et A_ϵ est une matrice triangulaire supérieure approchée.

Nous introduisons ainsi une variante de cette factorisation pour une matrice de Hankel complexe.

5.1.1 L'algorithme pour une matrice complexe de Hankel via Toeplitz

Nous proposons une version révisée du Théorème 2.2.1 (Chapitre 2) afin de mettre en place un algorithme de diagonalisation par blocs approchée d'une matrice complexe de Hankel.

Théorème 5.1.1 Soit $h = H(\varepsilon_1, ..., \varepsilon_{p-1}, h_p, ..., h_{2n-1})$, où $\varepsilon_j = \varepsilon_j^{(r)} + i\varepsilon_j^{(i)}$, $\varepsilon_j^{(r)} \in V(0, \mu)$ et $\varepsilon_j^{(i)} \in V(0, \mu)$ pour $j = 1, 2, ..., p-1$ avec $h_j \notin V(0, \mu)$ et

$$T = uT(h_p, ..., h_{2n-1}), \quad T^{-1} = uT(\mu_1, ..., \mu_{2n-p}),$$

$$t = uT(h_p, ..., h_{n+p-1}), \quad t^{-1} = uT(\mu_1, ..., \mu_n).$$

Alors

$$h' = {}^t\left(t^{-1}\right) h t^{-1} = \begin{pmatrix} h'_{11} & \epsilon' \\ {}^t(\epsilon') & h'_{22} \end{pmatrix}, \tag{5.3}$$

où

$$h'_{11} = lH\left(\mu_1, ..., \mu_p\right) + {}^t\left(t_{22}^{-1}\right) J_p \Sigma_p t_{22}^{-1},$$
$$h'_{22} = -H\left(\mu_{p+2}, ..., \mu_{2n-p}\right) + {}^t P J_p \Sigma_p P,$$
$$\epsilon' = {}^t\left(t_{22}^{-1}\right) J_p \Sigma_p P.$$

Preuve. Nous appliquons la même démonstration de [6]. ■

Pour calculer la diagonalisation par blocs approchée d'une matrice complexe de Hankel, nous appliquons l'algorithme suivant afin de récupérer la taille des blocs définie dans (2.2).

Algorithme 5.1.1 (Compteur de la taille des blocs) Soit $S = [\varepsilon_1, ..., \varepsilon_{p-1}, h_p, ..., h_{2n-1}]$ où $\varepsilon_j = \varepsilon_j^{(r)} + i\varepsilon_j^{(i)}$, $\varepsilon_j^{(r)} \in V(0, \mu)$ et $\varepsilon_j^{(i)} \in V(0, \mu)$ pour $j = 1, 2, ..., p-1$ avec $h_p \notin V(0, \mu)$, cet algorithme compte la taille des blocs c définie dans (2.2) avec une tolérance inférieure à un petit nombre positif ϵ $(0 < \epsilon \ll 1)$.

1. Poser $c = 0$
2. Pour $k = 1 :$ taille(S)
   Si $(|\text{real}\left(S(k)\right)| \leq \epsilon$ et $|\text{Im}\left(S(k)\right)| \leq \epsilon)$
   $c = c + 1$
   Sinon
   Sortir
   FinSi
   FinPour

Remarque 5.1.1 Algorithme 5.1.1 permet d'éviter l'utilisation de la technique SVD [45] pour compter la taille des blocs, qui nécessite plus d'opérations arithmétiques à l'égard à notre processus.

Ainsi, nous proposons l'algorithme rapide pour le calcul de la factorisation par blocs approchée comme suit :

Algorithme 5.1.2 (Algorithme de diagonalisation par blocs approchée d'une matrice de Hankel complexe via Toeplitz) Soit $S = [\varepsilon_1, ..., \varepsilon_{p-1}, h_p, ..., h_{2n-1}]$ où $\varepsilon_j = \varepsilon_j^{(r)} + i\varepsilon_j^{(i)}$, $\varepsilon_j^{(r)} \in V(0, \mu)$ et $\varepsilon_j^{(i)} \in V(0, \mu)$ pour $j = 1, 2, ..., p-1$ avec $h_p \notin V(0, \mu)$, cet algorithme calcule la diagonalisation par blocs approchée d'une matrice de Hankel complexe via Toeplitz.

1. Chercher p via Algorithme 5.1.1
2. Définir une matrice triangulaire de Toeplitz $t = uT\left(h_p, ..., h_{n+p-1}\right)$
3. Calculer t^{-1} via Lemme 1.3.2 (Division polynomiale, Chapitre 1)
4. Calculer $h' = {}^t\left(t^{-1}\right) h t^{-1}$
5. Poser $h'_{11} = h'(1 : p, 1 : p)$ et $h'_{22} = h'(p + 1 : n, p + 1 : n)$
6. Appliquer récursivement Algorithme 5.1.2 à $S = [h'_{22}\left(1 : n - p, 1\right) h'_{22}\left(n - p, 1 : n - p\right)]$, afin de récupérer (5.3).

5.2 Diagonalisation par blocs approchée d'une matrice de Hankel complexe via Schur

La méthode usuelle de Schur est l'une des plus célèbres techniques pour calculer la décomposition

$$h = LD^tL$$

d'une matrice de Hankel en $O(n^2)$ opérations ([46], [25] et [28]). Dans cette partie, nous dévoilons une nouvelle méthode de cette factorisation dans une approche approximative.

5.2.1 L'algorithme pour une matrice complexe de Hankel via Schur

Nous avons le résultat suivant.

Théorème 5.2.1 Soit h une matrice symétrique complexe vérifiant la partition suivante

$$h = \begin{pmatrix} h_{11} & {}^th_{21} \\ h_{21} & h_{22} \end{pmatrix} \tag{5.4}$$

où

$$h_{11} = lH(h_1, ..., h_p) + J_p\Sigma_p,\ h_{21} \in \mathbb{C}^{(n-p)\times p} \text{ et } h_{22} \in \mathbb{C}^{(n-p)\times(n-p)},$$

on considère le complément de Schur de h_{11}

$$h_{sc} = h_{22} - h_{21}\left(lH(h_1, ..., h_p)\right)^{-1}{}^th_{21}$$

et la matrice d'élimination

$$l = \begin{pmatrix} I_{p\times p} & 0_{p\times(n-p)} \\ h_{21}\left(lH(h_1, ..., h_p)\right)^{-1} & I_{(n-p)\times(n-p)} \end{pmatrix}.$$

Alors

$$lh'^tl = h \ \text{ où } \ h' = \begin{pmatrix} h'_{11} & \epsilon' \\ (\epsilon')^t & h'_{sc} \end{pmatrix} \tag{5.5}$$

où

$$\begin{aligned} h'_{11} &= h_{11},\ \epsilon' = -J_p\Sigma_p{}^t\left(h_{21}\left(lH(h_1, ..., h_p)\right)^{-1}\right), \\ h'_{sc} &= h_{sc} + \left(h_{21}\left(lH(h_1, ..., h_p)\right)^{-1}\right)J_p\Sigma_p{}^t\left(h_{21}\left(lH(h_1, ..., h_p)\right)^{-1}\right). \end{aligned}$$

Preuve. Nous appliquons la même démonstration de [6]. ■

La réitération du Théorème 5.2.1 définit la matrice diagonale par blocs approchée D_ε (5.2) où chaque bloc est une matrice triangulaire approchée de Hankel. Ainsi, nous proposons l'algorithme suivant :

Algorithme 5.2.1 (Diagonalisation par blocs approchée via Schur pour une matrice de Hankel) Soit h une matrice symétrique complexe définie par (5.4), cet algorithme calcule la diagonalisation par blocs approchée de h via Schur.

1. Définir la matrice symétrique complexe de h
2. Poser $lh_{11} = lH\left(h_1, ..., h_p\right)$ et $ut_{11} = uT\left(h_1, ..., h_p\right)$
3. Calculer $\left(ut_{11}\right)^{-1}$ via Lemme 1.3.2 (Division polynomiale, Chapitre 1)
4. Calculer $\left(lh_{11}\right)^{-1} = \left(ut_{11}\right)^{-1} J_p$
5. Poser $h_{11} = h(1:p, 1:p)$, $h_{21} = h(p+1:n, 1:p)$ et $h_{22} = h(p+1:n, p+1:n)$
6. Calculer $h_{sc} = h_{22} - h_{21}\left(lh_{11}\right)^{-1} {}^t h_{21}$
7. Poser $l = \left[I\left(p,p\right), h_{21}\left(lh_{11}\right)^{-1}; 0\left(n-p,p\right), I\left(n-p,n-p\right)\right]$
8. Poser $invl = \left[I\left(p,p\right), -h_{21}\left(lh_{11}\right)^{-1}; 0\left(n-p,p\right), I\left(n-p,n-p\right)\right]$
9. Calculer $h' = \left(invl\right) h^t \left(invl\right)$
10. Poser $h'_{11} = h'(1:p, 1:p)$ et $h'_{sc} = h'(p+1:n, p+1:n)$.

5.3 Exemples Numériques

Un exemple numérique est présenté dans cette section pour montrer l'exécution des étapes intermédiaires des processus. Nous proposons également des exemples numériques qui illustrent l'efficacité et le coût faible de la diagonalisation par blocs approchée au moyen de Toeplitz par rapport à la méthode classique de factorisation approchée de Schur.

- Tous les algorithmes ont été implémentés sur MATLAB 7.0.4.287 (R2007A) et ont été exécutés sur un ordinateur portable Intel(R) Core(TM)2 CPU T5600 avec un processeur 1.83 GHz et 2046 Mb de RAM.
- Dans les exemples, nous introduisons la matrice de Hankel à partir d'une perturbation (rajouter $\left(k^{(r)} + ik^{(i)}\right) \times 10^{-13}$ où $k^{(r)}$, $k^{(i)}$ sont choisis aléatoirements dans $[-1, 1]$) pour tous les coefficients de la matrice de Hankel "exacte"), dont les blocs (de différents tailles) de la matrice de Hankel "exacte" sont connus.
- L'erreur relative de la méthode via le complément de Schur et la méthode via Toeplitz sont définie, respectivement par :

$$Error_{Sc} = \frac{\left\|D_{Sc\ approx} - D_{Sc}\right\|_1}{\left\|D_{Sc}\right\|_1},\ Error_{Our} = \frac{\left\|D_{Our\ approx} - D_{Our}\right\|_1}{\left\|D_{Our}\right\|_1}. \quad (5.6)$$

5.3.1 Comparaison des algorithmes

Nous comparons l'approche au moyen de Toeplitz par rapport à la diagonalisation par blocs approchée classique de la matrice complexe de Hankel (Algorithme 4.3). Nous décrivons également les différences et les similarités entre ces deux méthodes par le biais d'un exemple.

On considère la liste S suivante de taille 13 qui définie la matrice de Hankel :

$S = [0,\ \frac{1}{6} - \frac{1}{6}i,\ -\frac{1}{6} + \frac{1}{6}i,\ \frac{1}{6} + \frac{4}{9}i,\ -\frac{1}{6} - \frac{19}{18}i,\ -\frac{103}{108} + \frac{59}{108}i,\ -\frac{143}{45} + \frac{49}{45}i,\ -\frac{3911}{1620} - \frac{173}{45}i,$
$-\frac{10892}{2025} + \frac{6889}{900}i,\ \frac{815831}{48600} - \frac{242717}{48600}i,\ -\frac{5083067}{243000} - \frac{4592431}{243000}i,\ -\frac{9589}{13500} + \frac{42814517}{729000}i,\ \frac{496841}{6750} - \frac{7561091}{121500}i]$.

Figures 5.3.1 et 5.3.2 décrivent toutes les étapes exécutées avec l'Algorithme 5.1.1 et Algorithme 5.2.1 respectivement. Dans la Figure 5.3.3, nous proposons les matrices diagonales par blocs ainsi que les matrices triangulaires (5.2), obtenues par les deux méthodes.

Ainsi, toutes les propriétés du Chapitre 1 des deux méthodes appliquées à une matrice de Hankel coefficients réels sont vérifiées pour une matrice de Hankel à coefficients complexes.

5.3.2 Efficacité

Quelques résultats numériques sont présentés pour illustrer l'efficacité de notre algorithme à l'égard de l'algorithme de Schur approché pour 11 différentes séquences définissant les matrices de Hankel (5.4).

Ainsi, l'erreur relative de notre algorithme et celle de l'algorithme du complément de Schur (5.6) sont données dans les tableaux suivants :

n	$\|D_{\mathbf{Our}}\|_1$	$\|D_{\mathbf{Our\ approx}}\|_1$	$\mathbf{Error_{Our}}$
7	8.4852813742385700	8.4852813742339990	5.706774496276349e-13
14	23.000000000000000	23.000000221676345	9.697543542689079e-09
28	23.000000000000000	23.000000192887367	8.393834710724900e-09
56	10.000000000003041	10.000000003571856	2.729703512635876e-08
112	12.000000000000000	12.000000000082560	3.166115405614226e-08
224	15.000000000000000	15.000000000104892	5.453164647193843e-06
336	23.236067977499783	23.236068029831394	3.982568713893646e-08
448	23.236067977499783	23.236068720381791	3.475003625625449e-07
672	12.000000000000000	12.000000000003219	3.739884346894936e-06
896	12.000000000000000	12.000000000334289	9.270214916390855e-07
1120	13.000000000000000	13.000000053823175	9.069382833634979e-06

n	$\|D_{\mathbf{Sc}}\|_1$	$\|D_{\mathbf{Sc\ approx}}\|_1$	$\mathbf{Error_{Sc}}$
7	310.9203966784305	310.92039667837630	2.213926888702992e-13
14	36.000000000000000	36.000000000122682	2.192140674609713e-08
28	8.000000000000000	8.0000000000226170	1.179146052406410e-09
56	7.228045704137195	7.2280457034719530	6.270644695469078e-08
112	138.0665927567458	138.06659276952500	2.809358098521050e-08
224	92.000000000000000	92.000000003949523	1.858406019570196e-06
336	3.000000000000000	3.0000000000011880	1.716036350304927e-07
448	5.500000000000000	5.5000000000201320	9.448631662438197e-07
672	1161.000000000000	1160.9756804024230	4.563332870795622e-05
896	1161.000000000000	1161.0275947690960	3.972672785950243e-05
1120	63.000000000000000	63.000000095200178	1.611067508970338e-05

```
Step 1:

D1 =

  1.0801e-24 +1.800e-12i   3           +        3i
  3          +        3i   3           +        3i

H22 =

  2.5297e-11 +2.0400e-11i  2.5296e-011 +2.0409e-11i -0.10000000 -0.30000000i  2.5185e-11 +2.1216e-11i -0.28000 +0.46000i
  2.5297e-11 +2.0408e-11i -0.100000000 -0.30000000i  2.5073e-11 +2.0887e-11i -0.28000000 +0.46000000i  0.40000 -0.30000i
 -0.10000000 -0.30000000i  2.50790e-11 +2.0880e-11i -0.28000000 +0.46000000i  0.40000000 -0.30000000i  0.90600 -0.14200i
  2.5186e-11 +2.1225e-11i -0.280000000 +0.46000000i  0.40000000 -0.30000000i  0.90600000 -0.14200000i -1.66530 -0.37600i
 -0.28000000 +0.46000000i  0.400000000 -0.30000000i  0.90600000 -0.14200000i -1.66530000 -0.37600000i -0.51422 -0.58378i

Step 2:

D2 =

 -7.9978e-11  -3.1498e-10i -7.9913e-11  -3.1505e-10i          -1 +         3i
 -7.9913e-11  -3.1505e-10i -1           +         3i  3.9584e-10 -1.2961e-10i
 -1           +         3i  3.9584e-010 -1.2961e-10i          -5 +         2i

H22 =

 -8.8373e-08  -1.7003e-07i  15          -3.6869e-007i
 15           -3.6869e-07i  47          +          1i


Step 3:

Set
D3 =

 -8.8373e-08  -1.7003e-07i  15          -3.6869e-007i
 15           -3.6869e-07i  47          +          1i

End STOP.
```

Figure 5.3.1 décrit les étapes exécutées avec l'Algorithme 5.1.1

Step 1:

D1 =

```
  1e-13       +    1e-13i  0.16667000 - 0.16667000i
  0.16667000  -0.1666700i -0.16667000 + 0.16667000i
```

Sc2 =

```
  1.1335e-12 -7.3319e-13i  6.6613e-16 -6.7224e-013i -0.016667 +0.0055556i  0.016667 -0.0055556i  0.029259 -0.019630i
  4.4409e-16 -6.7213e-13i -0.01666700 +0.005555600i  0.033333 -0.0111110i  0.012593 -0.0140740i -0.095556 +0.057778i
 -0.016667   +0.00555560i  0.03333300 -0.011111000i  0.016296 -0.0492590i -0.099259 +0.0929630i  0.118650 +0.033000i
  0.016667   -0.00555560i  0.01259300 -0.014074000i -0.099259 +0.0929630i  0.060938 -0.0382960i  0.117420 -0.235520i
  0.029259   -0.01963000i -0.09555600 +0.057778000i  0.118650 +0.0330000i  0.117420 -0.2355200i -0.530060 +0.256700i
```

Step 2:

D2 =

```
  1.1335e-12 -7.3319e-13i  6.6613e-16 -6.7224e-013i -0.016667 +0.0055556i
  4.4409e-16 -6.7213e-13i -0.01666700 +0.005555600i  0.033333 -0.0111110i
 -0.01666700 +0.00555560i  0.03333300 -0.011111000i  0.016296 -0.0492590i
```

Sc3 =

```
  1.1745e-11 +1.2695e-11i   0.00022222 +0.00029630i
  0.00022222 +0.00029630i  -0.00112100 -0.00153580i
```

Step 3:

Set

D3 =

```
  1.1745e-11 +1.2695e-11i   0.00022222 +0.00029630i
  0.00022222 +0.00029630i  -0.00112100 -0.00153580i
```

End STOP.

Figure 5.3.2 décrit les étapes exécutées avec l'Algorithme 5.2.1

ourA =

```
 3        +          3i   3          +        3i -11         + 33i          8.9042e-10 +   3.51e-09i -55 +          22i      54         +7.4128e-09i   -7.0254e-09+8.2121e-09i
 0                      3          +        3i -12         +        6i -11          +         33i -21 -          9i     -40         +        37i   54          +7.723e-09i
 0                      0                      -12         + 6i          -12          +          6i -32 +         24i     -6         +         6i  -40          +        37i
 0                      0                        0                       -12          +          6i -12 +          6i     -32         +        24i   -6          +         6i
 0                      0                        0                         0                        -12 +          6i     -12         +         6i  -32          +        24i
 0                      0                        0                         0                          0                    -12         +         6i  -12          +         6i
 0                      0                        0                         0                          0                     0                      -12          +         6i
```

Aschur =

```
 1                      0                     -1.8333     +1.83330000i  1.83330   -1.8333i        -1.8333     -4.8889i    1.7667     +11.5780000i  10.5570     -5.975900i
 0                      1                     -1          +6.0013e-13i -0.83333   +1.8333i         2.6667     -3.6667i   -4.5000     -1.22220000i   6.2667     +12.80000i
 0                      0                      1                        0                          0                     -0.6000     +0.20000000i  -3.3000     +0.933330i
 0                      0                      0                        1                          0                     -2.9333     +0.53333000i   2.3333     -0.333330i
 0                      0                      0                        0                          1                     -1          -3.3801e-11i  -1.9333     +0.533330i
 0                      0                      0                        0                          0                      1                         0
 0                      0                      0                        0                          0                      0                         1
```

ourD =

```
 1.0802e-24 +1.8000e-12i  3          +         3i -1.3202e-11 +6.5994e-12i -1.7764e-15 -3.5527e-15i -2.3096e-11 -9.9121e-12i  1.6191e-11 +1.6193e-11i  1.4211e-14 +2.8422e-14i
 3          +          3i  3          +         3i -1.3204e-11 +6.5938e-12i  5.3291e-15 +         0i -2.3110e-11 -9.9192e-12i  1.6229e-11 +1.6200e-11i  1.4211e-14 -1.4211e-14i
-1.3210e-11 +6.5994e-12i -1.3210e-11 +6.5994e-12i -7.9974e-11 -3.1499e-10i -7.9899e-11 -3.1507e-10i -1          +         3i -1.1073e-10 -1.7310e-10i -4.3246e-10 +2.7716e-10i
-6.6613e-16 -6.6613e-16i -2.6645e-15 +2.6645e-15i -7.9926e-11 -3.1509e-10i -1          +         3i  3.9470e-10 -1.2643e-10i -4.3288e-10 +2.7699e-10i -6.2412e-12 +1.2683e-12i
-2.3106e-11 -9.9054e-12i -2.3126e-11 -9.9094e-12i -1          +         3i  3.9466e-10 -1.2645e-10i -5          +         2i  9.6982e-11 -3.3320e-10i -8.3117e-10 -2.5917e-10i
 1.6196e-11 +1.6191e-11i  1.6191e-11 +1.6186e-11i -1.1081e-10 -1.7306e-10i -4.3259e-10 +2.7695e-10i  9.7296e-11 -3.3306e-10i -3.9309e-10 -7.5767e-10i 15          -3.6869e-07i
-1.5987e-14 +5.3291e-15i -1.5987e-14 +2.6645e-14i -4.3271e-10 +2.7669e-10i -6.2101e-12 +1.1866e-12i -8.3156e-10 -2.5996e-10i 15          -3.6869e-07i 47         +         1i
```

Dschur =

```
 1.0000e-13 +         0i  0.16667000 -0.16667000i  3.8335e-13 +3.8339e-13i  0.61111    +0.61111000i -1.2222     -1.22220000i -0.16296000 +0.48889000i  1.3333000 -0.37778000i
 0.16667000 -0.16667000i -0.16667000 +0.16667000i  0.61111000 +0.61111000i -1.2222     -1.22220000i -0.40741    -0.40741000i  1.17040000 +0.11111000i -2.9383000 -1.41730000i
 1.8337e-13 -1.8331e-13i -1.2297e-16 -1.3231e-16i  1.8055e-12 -7.3302e-13i -6.7097e-13 -6.7268e-13i -0.016667   +0.00555560i  1.2333e-12 -6.7316e-14i -0.0059259 -0.01777800i
-1.8318e-13 +1.8345e-13i  5.4566e-17 +7.1422e-17i -6.7179e-13 -6.7131e-13i -0.016667   +0.00555560i  0.033333   -0.01111100i -0.00592590 -0.01777800i  0.0155560 +0.04666700i
 1.8323e-13 +4.8937e-13i  2.4037e-16 -6.0667e-16i -0.01666700 +0.00555560i  0.033333   -0.01111100i  0.016296   -0.04925900i  0.00962960 +0.02888900i  0.0347650 -0.03595100i
-3.5942e-13 -1.7971e-13i  2.8413e-16 +3.4758e-16i -1.5323e-13 -3.6255e-12i -2.0504e-12 -5.1192e-12i -1.5366e-12 +1.4229e-12i  1.0277e-11 -8.0346e-12i  0.0002222 +0.00029630i
 3.6071e-13 +1.8105e-13i -7.6359e-16 -9.4461e-16i  3.6051e-12 -9.1697e-13i  6.5573e-13 +1.3588e-12i  1.5422e-12 -1.4312e-12i  0.00022222 +0.00029630i -0.0011210 -0.00153580i
```

Figure 5.3.3 Les matrices diagonales par blocs et les matrices triangulaires supérieures.

En outre, une étude expérimentale de l'efficacité du problème confirme "numériquement" que les deux approches sont efficaces et la perturbation ci-dessus (pour $k = 1, ..., 13$, nous ne préservons pas les diagonalisation par blocs) ne diverge pas le résultat à partir une certaine tolérance additive.

5.3.3 Temps de calcul

Il s'avère intéressant de savoir le temps d'exécution nécessaire pour chaque méthode. Dans le tableau suivant, nous illustrons les temps de calculs (en sec) de notre diagonalisation approchée par blocs et de l'approche approximative de complément de Schur pour 11 différentes matrices complexes perturbées de Hankel.

n	**Time$_{\text{Sc approximate}}$**	**Time$_{\text{Our approximate}}$**
7	0.011534	0.008796
14	0.004748	0.002351
28	0.013804	0.002813
56	0.012679	0.011248
112	0.037908	0.035668
224	0.268646	0.249651
336	0.880709	0.816654
448	1.593345	1.580948
672	6.347841	6.006527
896	14.153723	14.048224
1120	28.963264	28.857474

D'où, nous concluons que la factorisation via Schur nécessite un peu plus de temps par rapport à l'approche au moyen de Toeplitz.

5.3.4 Application à l'algorithme d'Euclide

Soient $u(x) = 2ix^7 + (6i - 7)x^6 - (17 + 5i)x^5 - (64 + 21i)x^4 - (211i + 93)x^3 - (329 + 334i)x^2 - (490 + 645i)x - 751$, and $v(x) = 2ix^5 + (-1 + 6i)x^4 + (1 + 30i)x^3 + (15 + 42i)x^2 + (44 + 76i)x + 67$.

Alors, $H(u, v) = H(0, 1, \text{-}3i, 2, \text{-}39i, \text{-}94, \text{-}1\text{-}152i, \text{-}1502\text{+}10i, 50\text{+}2837i, \text{-}8069\text{-}140i, \text{-}171\text{+}55458i, 76781\text{+}58i, 2920\text{+}(765777/2)i)$.
Dans ce qui suit, nous montrons la connexion entre la diagonalisation par blocs approchée de $H(u, v)$ et l'algorithme d'Euclide (pour toutes les étapes).
Etape 1 : Diagonalisation par blocs de $H(u, v)$
$H(q_1, 1) = H(0, 1, 3i)$,
$H(v, r_1) = H((-9.6809 - 6.3949i)e\text{-}13,\ (1 + 5.5397i)e\text{-}13,\ (-4.8404\ -3.1974i)e\text{-}12,\ -8.6832e\text{-}17,\ (9.6875 + 6.3949i)e\text{-}013,\ -3.011e\text{-}13 + 2.5686e\text{-}12i,\ -2 + 8.0178e\text{-}11i,\ -1 - 4.7778e\text{-}10i,\ 12 - 0.5i)$,
$q_1(x) = 1x^2 + (-1.5e\text{-}13 + 3i)x - 11 - 4.7398e\text{-}13i$,
$r_1(x) = (3.4904e\text{-}12 + 2i)x^3 + (3 + 4i)x^2 + (6 + 10i)x + 14 + 2.555e\text{-}11i$.
Etape 1 : Algorithme d'Euclide

$r_{-1}(x) = r_0(x)\, q_1(x) - r_1(x)$,
$q_1(x) = 1x^2 + (-1.501e\text{-}13 + 3i)x - 11 - 4.7606e\text{-}13i$,
$r_1(x) = (1.1227e\text{-}11 + 2i)x^3 + (3 + 4i)x^2 + (6 + 10i)x + 14 - 3.1896e\text{-}11i$.
Etape 2 : Diagonalisation par blocs de $H(u, v)$
$H(q_2, 1) = H(8.4427e\text{-}12 - 4.4421e\text{-}11i, 1 - 1.5502e\text{-}10i, 1 + 2i)$,
$H(r_1, r_2) = H(8.4427e\text{-}12 - 4.4421e\text{-}11i,\ 1 - 1.5502e\text{-}10i,\ 4.2213e\text{-}11\ -2.221e\text{-}10i,$
$8.4427e\text{-}11 - 4.4421e\text{-}10i,\ -1.8705e\text{-}10 - 0.5i)$,
$q_2(x) = (1 + 3.1296e\text{-}12i)x^2 + (1 + 2i)x + 5 + 3.1296e\text{-}12i$,
$r_2(x) = (2.8121e\text{-}10 - 2i)x - 3 - 4.7869e\text{-}10i$.
Etape 2 : Algorithme d'Euclide
$r_0(x) = r_1(x)\, q_2(x) - r_2(x)$,
$q_2(x) = (1 + 5.5633e\text{-}12i)x^2 + (1 + 2i)x + 5 + 9.468e\text{-}12i$
$r_2(x) = (2.8301e\text{-}11 - 2i)x - 3 + 2.693e\text{-}11i$.
Etape 3 : Diagonalisation par blocs de $H(u, v)$
$H(q_3, 1) = H(8.4427e\text{-}12 - 4.4421e\text{-}11i, 1 - 1.5502e\text{-}10i, 2 - 5.2256e\text{-}10i)$,
$H(r_2, r_3) = H(1 + 5.147e\text{-}10i)$,
$q_3(x) = (1 - 6.6174e\text{-}11i)x^2 + (2 - 3.4487e\text{-}10i)x + 5 - 7.694e\text{-}10i$,
$r_3(x) = 1 + 5.147e\text{-}10i$.
Etape 3 : Algorithme d'Euclide
$r_1(x) = r_2(x)\, q_3(x) - r_3(x)$.
$q_3(x) = (1 - 1.9764e\text{-}11i)x^2 + (2 - 2.6777e\text{-}11i)x + 5 - 1.3356e\text{-}11i$,
$r_3(x) = 1 - 1.4282e\text{-}10i$.

5.4 Conclusion

Ce chapitre propose une nouvelle méthode pour factoriser une matrice carrée complexe de Hankel. Notre approche est comparée à la factorisation par blocs approchée classique d'une matrice carrée complexe de Hankel. Une étude numérique montre que les algorithmes sont efficaces.

Chapitre 6

Une étude expérimentale

L'un des buts de cette thèse est de certifier les algorithmes présentés dans les précédents chapitres. En effet, une étude théorique pour des algorithmes numériques est incontestablement beaucoup plus poussée, cela doit prendre en compte les questions de précisions aussi bien dans les données que dans les calculs : Avec quelle précision les données devraient être connues ? Avec quelle précision les calculs devraient être faits ? Ici certifier un résultat, cela veut dire que, dans le voisinage des données défini par la précision, les blocs sont optimaux (C'est-à-dire, les plus gros possibles) et que d'autres calculs donnant des blocs trop petits feraient inévitablement des erreurs grossières dues à des divisions par des nombres trop petits. Nous n'avons pas pu aboutir à notre but, parce que ce problème s'est révélé énormément plus difficile qu'on pensait à priori.

Dans les chapitres précédents, nous avons développé des heuristiques sans pouvoir les certifier théoriquement. Nous analysons donc la propagation des erreurs dans les étapes intermédiaires des algorithmes élaborés au moyen d'une étude expérimentale conséquente qui légitime nos heuristiques dans une certaine mesure. D'une part, nous comparons l'efficacité des différents algorithmes et nous proposons une étude statistique appliquées à une vingtaine de matrices de Hankel réelles ou complexes perturbées d'une manière aléatoire ; C'est-à-dire, nous additionnons $\left(k^{(r)} + ik^{(i)}\right) \times 10^{-15}$ ($k^{(i)} = 0$, dans le cas réel) à chaque composante de la matrice de Hankel "exacte" : les blocs (de différentes tailles) de la matrice diagonale sont précisément connues, avec $k^{(r)}$, $k^{(i)}$ sont choisis aléatoirement dans $[-1, 1]$. D'autre part, nous comparons la stabilité numérique des algorithmes de diagonalisation par blocs de la matrice de Hankel ; De toute évidence, il s'agit d'un calcul en virgule flottante. Nous examinons par la suite les différents algorithmes en terme de temps (en seconde). Nous étudions également l'influence d'appliquer l'inversion approchée d'une matrice triangulaire de Toeplitz au lieu de l'inversion exacte dans les étapes intermédiaires de l'algorithme de diagonalisations par blocs de matrices de Hankel. Nous comparons encore l'approche fondée sur les matrices de Hankel avec celle basée sur les matrices le Bézout tant pour leur efficacité et leur stabilité numérique. Enfin, nous analysons le rôle de ε dans l'exécution des algorithmes.

Ce chapitre a été évalué en tenant compte des points suivants :

– Tous les algorithmes ont été implémentés sous MATLAB 7.0.4.287 (R2007A) et

ont été exécutés sur un ordinateur portable Intel(R) Core(TM)2 CPU T5600 avec un processeur 1.83 GHz et 2046 Mb de RAM.

– L'erreur relative de la méthode via le complément de Schur et la méthode via Toeplitz sont définie, respectivement par :

$$Error^R_{Sc} = \frac{\|H_{Sc\ approx} - H_{Sc\ exact}\|_1}{\|H_{Sc\ exact}\|_1},\ Error^R_{Our} = \frac{\|H_{Our\ approx} - H_{Our\ exact}\|_1}{\|H_{Our\ exact}\|_1}. \tag{6.1}$$

– L'erreur absolue de la méthode via le complément de Schur et la méthode via Toeplitz sont définie, respectivement par :

$$Error^A_{Sc} = \|D_{Sc\ approx} - D_{Sc}\|_1\ ,\ Error^A_{Our} = \|D_{Our\ approx} - D_{Our}\|_1. \tag{6.2}$$

– **Bn** et **Mv** représentent le nombre de perturbations Bonnes et Mauvaises sur les 500 perturbations choisies au hasard, respectivement.

– **Echec** : quand le nombre des blocs ne coïncide pas avec le nombre des blocs "exacte".

6.1 Efficacité

Nous tirons au hasard des perturbations en grand nombre, de diverses tailles, afin de donner des résultats statistiques sur le comportement des différents algorithmes de diagonalisations par blocs approchées de la matrice de Hankel. Nous ferons cela pour une vingtaine de matrices de départ "exactes", réels, complexes mal-conditionnées que nous les perturbons 500 fois. Le teste effectué mesure la différence des matrices diagonales par blocs au moyen du complément de Schur et Toeplitz, respectivement donnée par (6.2) par rapport à une tolérance fixe $\varepsilon = 10^{-5}$.

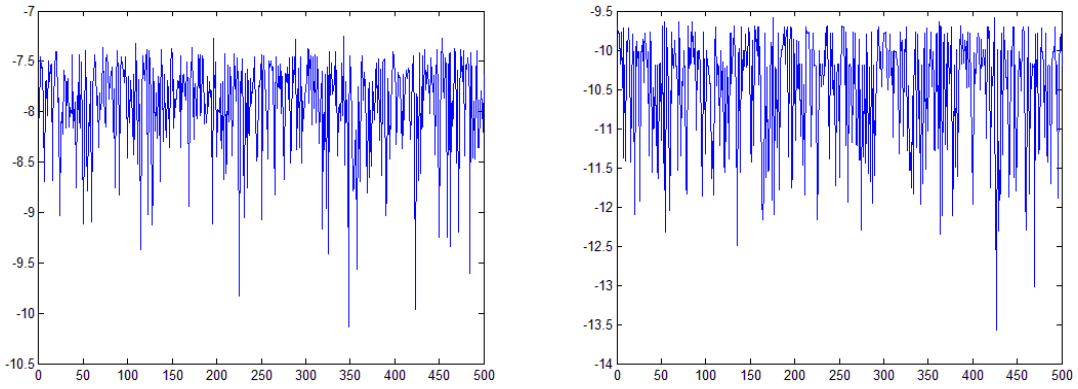


Figure 6.1.1 : $\log_{10}\left(Error^A_{Our}\right)$ (Figure gauche) et $\log_{10}\left(Error^A_{Sc}\right)$ (Figure droite) pour une matrice de Hankel d'ordre 9.

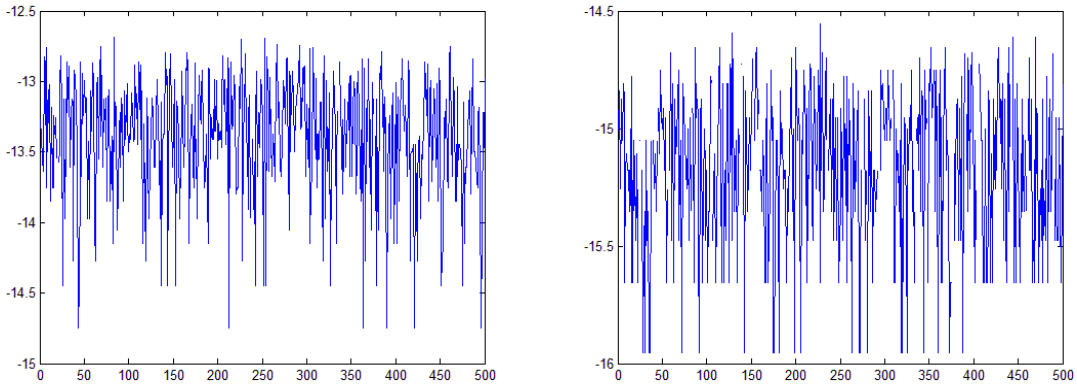


Figure 6.1.2 : $\log_{10}\left(Error^{A}_{Our}\right)$ (Figure gauche) et $\log_{10}\left(Error^{A}_{Sc}\right)$ (Figure droite) pour une matrice de Hankel d'ordre 12.

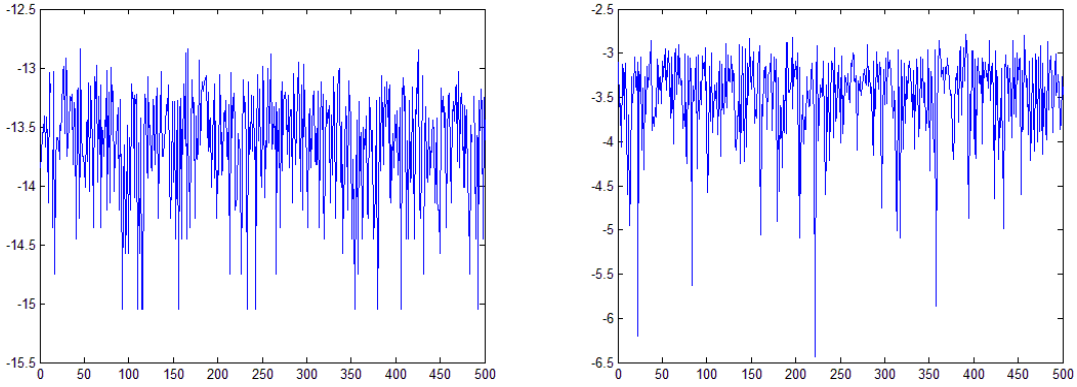


Figure 6.1.3 : $\log_{10}\left(Error^{A}_{Our}\right)$ (Figure gauche) et $\log_{10}\left(Error^{A}_{Sc}\right)$ (Figure droite) pour une matrice de Hankel d'ordre 25.

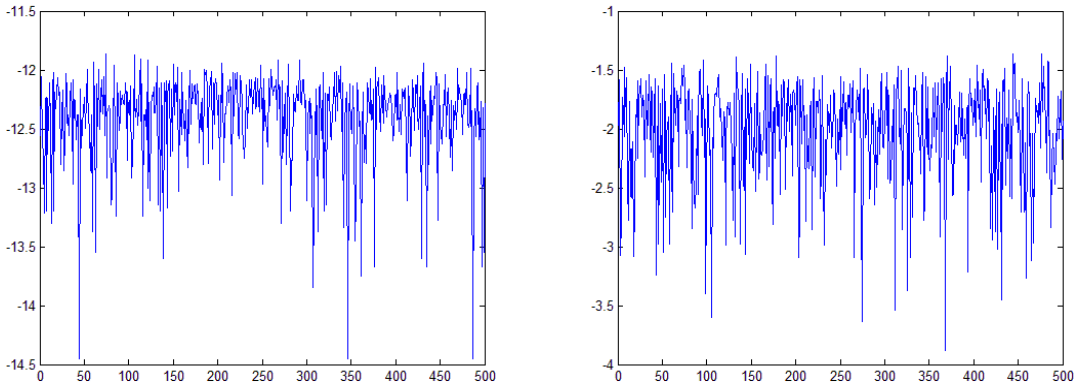


Figure 6.1.4 : $\log_{10}\left(Error^{A}_{Our}\right)$ (Figure gauche) et $\log_{10}\left(Error^{A}_{Sc}\right)$ (Figure droite) pour une matrice de Hankel d'ordre 54.

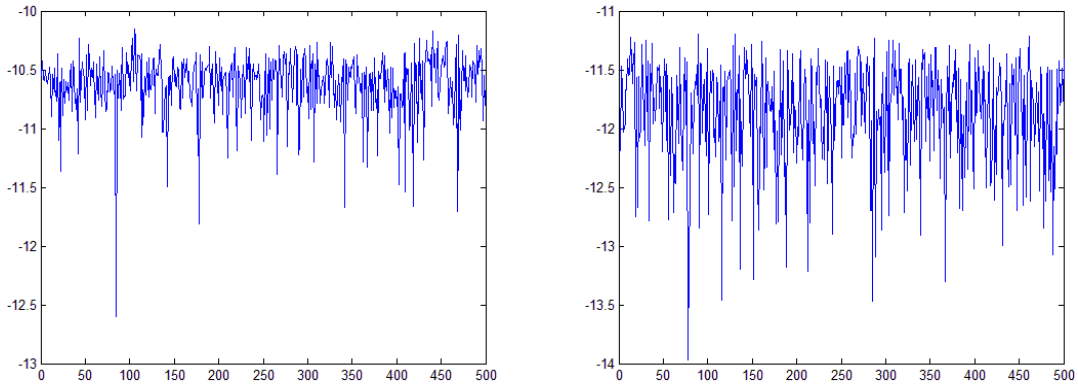


Figure 6.1.5 : $\log_{10}\left(Error^{A}_{Our}\right)$ (Figure gauche) et $\log_{10}\left(Error^{A}_{Sc}\right)$ (Figure droite) pour une matrice de Hankel d'ordre 56.

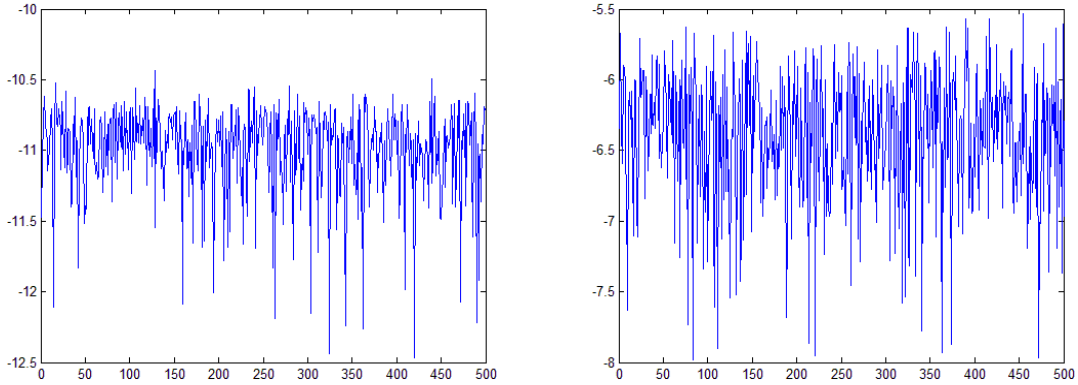


Figure 6.1.6 : $\log_{10}\left(Error^{A}_{Our}\right)$ (Figure gauche) et $\log_{10}\left(Error^{A}_{Sc}\right)$ (Figure droite) pour une matrice de Hankel d'ordre 75.

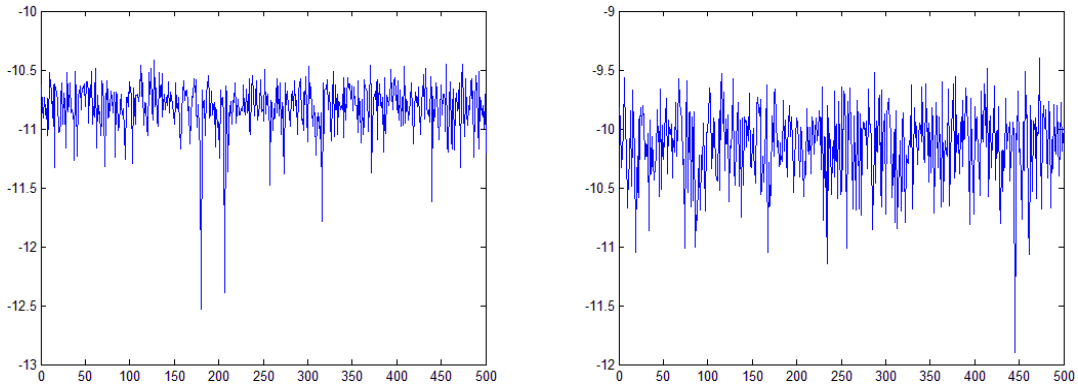


Figure 6.1.7 : $\log_{10}\left(Error^{A}_{Our}\right)$ (Figure gauche) et $\log_{10}\left(Error^{A}_{Sc}\right)$ (Figure droite) pour une matrice de Hankel d'ordre 100.

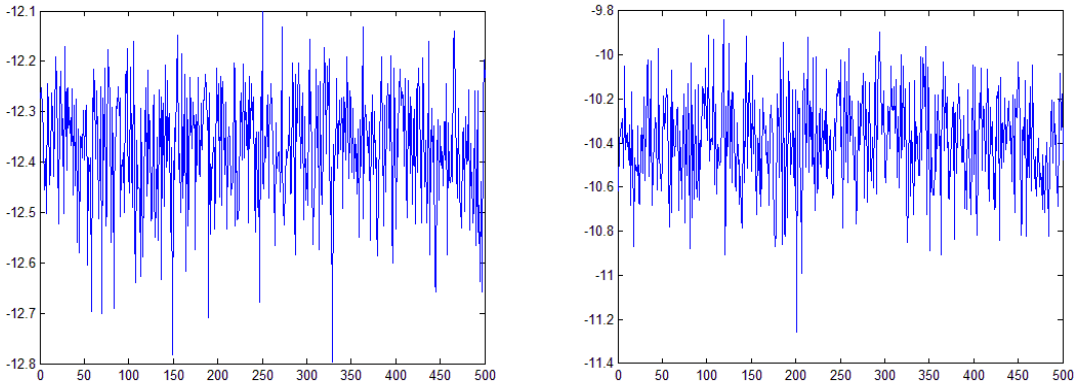


Figure 6.1.8 : $\log_{10}\left(Error^{A}_{Our}\right)$ (Figure gauche) et $\log_{10}\left(Error^{A}_{Sc}\right)$ (Figure droite) pour une matrice de Hankel d'ordre 112.

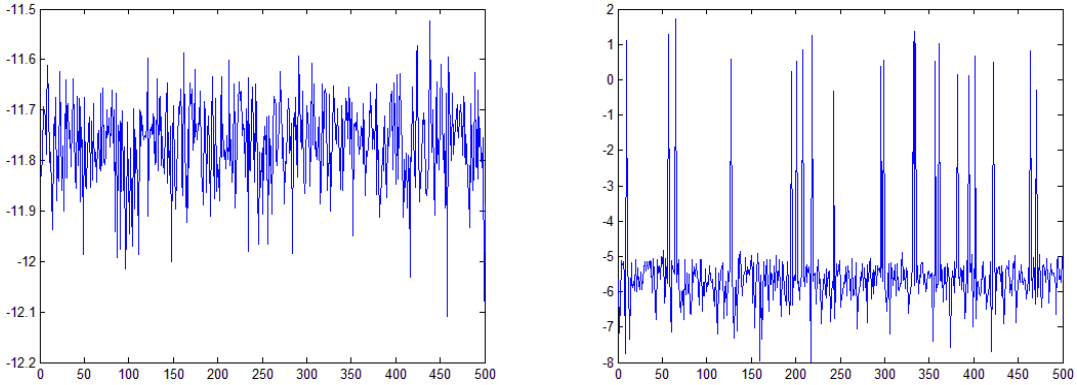


Figure 6.1.9 : $\log_{10}\left(Error^{A}_{Our}\right)$ (Figure gauche) et $\log_{10}\left(Error^{A}_{Sc}\right)$ (Figure droite) pour une matrice de Hankel d'ordre 150.

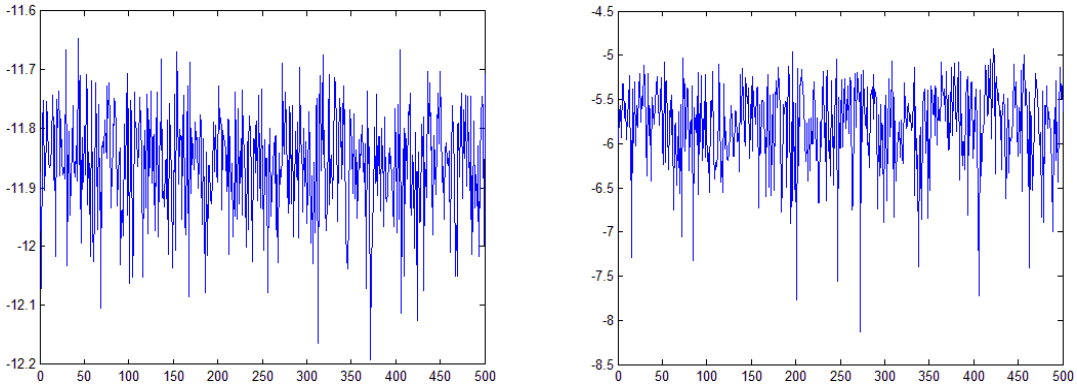


Figure 6.1.10 : $\log_{10}\left(Error^{A}_{Our}\right)$ (Figure gauche) et $\log_{10}\left(Error^{A}_{Sc}\right)$ (Figure droite) pour une matrice de Hankel d'ordre 200.

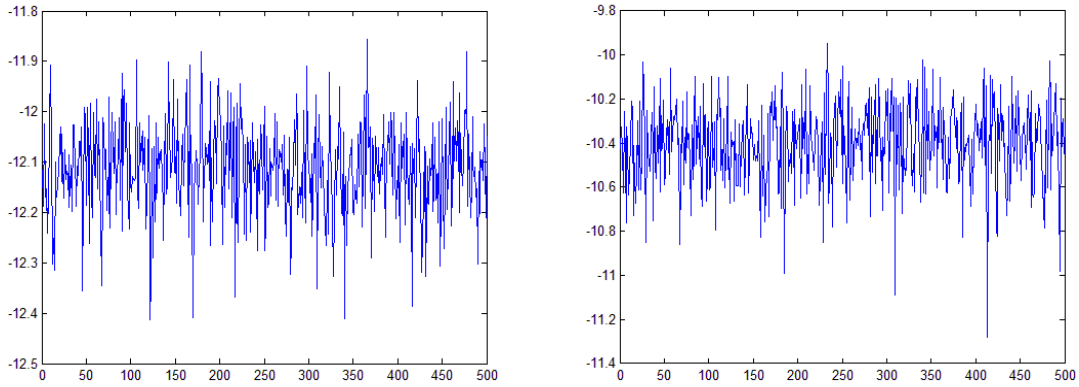


Figure 6.1.11 : $\log_{10}\left(Error^{A}_{Our}\right)$ (Figure gauche) et $\log_{10}\left(Error^{A}_{Sc}\right)$ (Figure droite) pour une matrice de Hankel d'ordre 242.

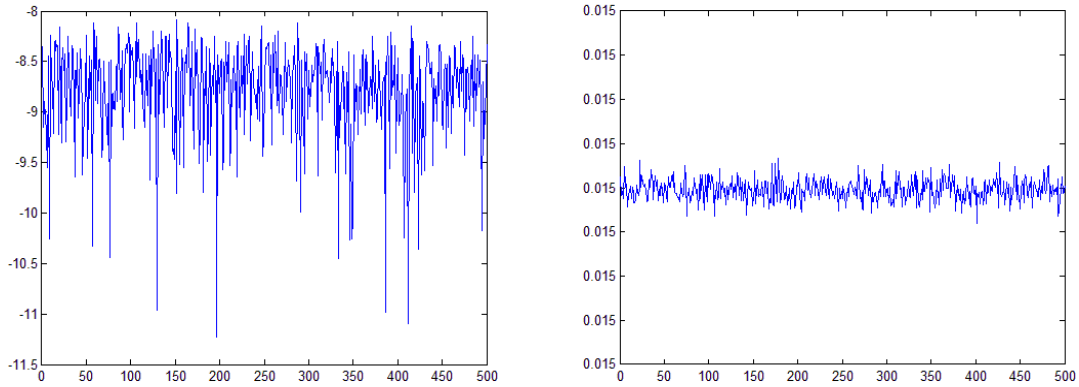


Figure 6.1.12 : $\log_{10}\left(Error^{A}_{Our}\right)$ (Figure gauche) et $\log_{10}\left(Error^{A}_{Sc}\right)$ (Figure droite) pour une matrice de Hankel d'ordre 336.

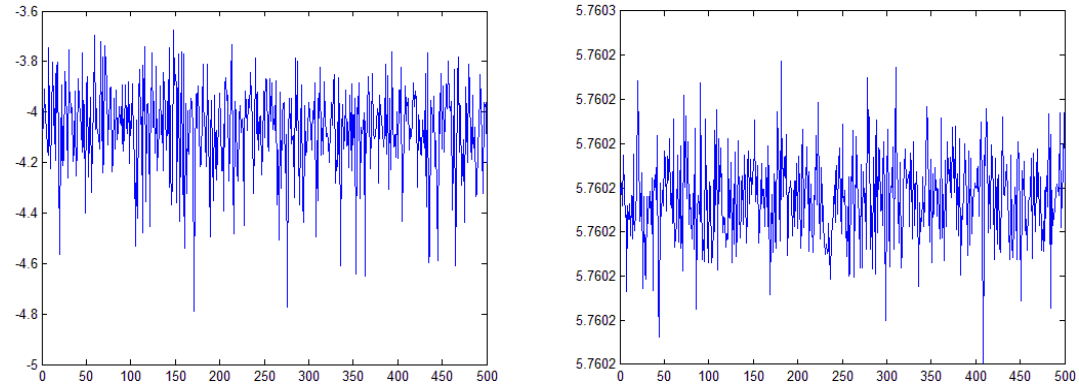


Figure 6.1.13 : $\log_{10}\left(Error^{A}_{Our}\right)$ (Figure gauche) et $\log_{10}\left(Error^{A}_{Sc}\right)$ (Figure droite) pour une matrice de Hankel d'ordre 400.

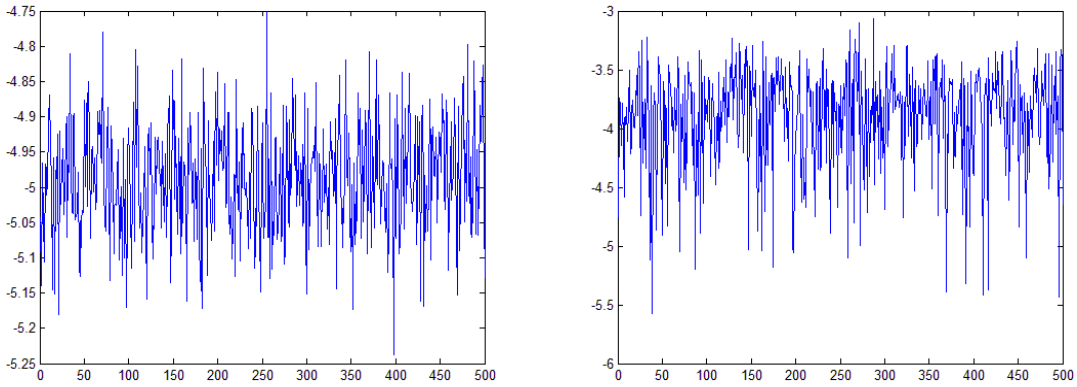


Figure 6.1.14 : $\log_{10}\left(Error^{A}_{Our}\right)$ (Figure gauche) et $\log_{10}\left(Error^{A}_{Sc}\right)$ (Figure droite) pour une matrice de Hankel d'ordre 448.

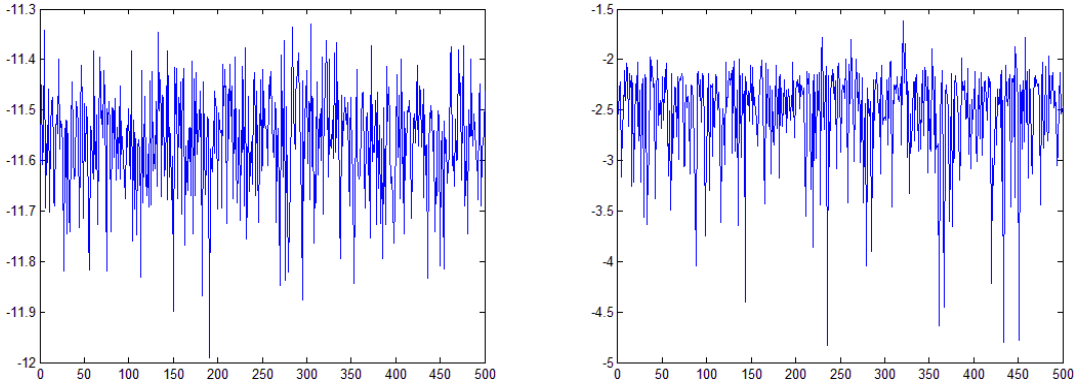


Figure 6.1.15 : $\log_{10}\left(Error^{A}_{Our}\right)$ (Figure gauche) et $\log_{10}\left(Error^{A}_{Sc}\right)$ (Figure droite) pour une matrice de Hankel d'ordre 500.

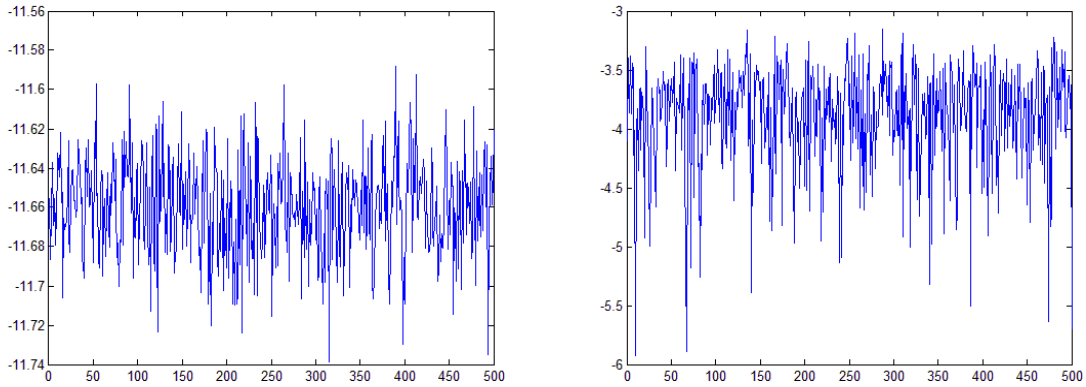


Figure 6.1.16 : $\log_{10}\left(Error^{A}_{Our}\right)$ (Figure gauche) et $\log_{10}\left(Error^{A}_{Sc}\right)$ (Figure droite) pour une matrice de Hankel d'ordre 672.

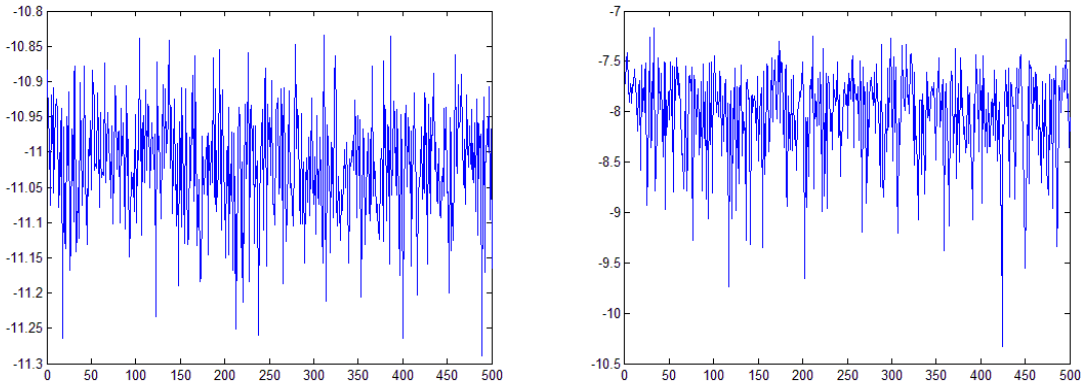


Figure 6.1.17 : $\log_{10}\left(Error^{A}_{Our}\right)$ (Figure gauche) et $\log_{10}\left(Error^{A}_{Sc}\right)$ (Figure droite) pour une matrice de Hankel d'ordre 750.

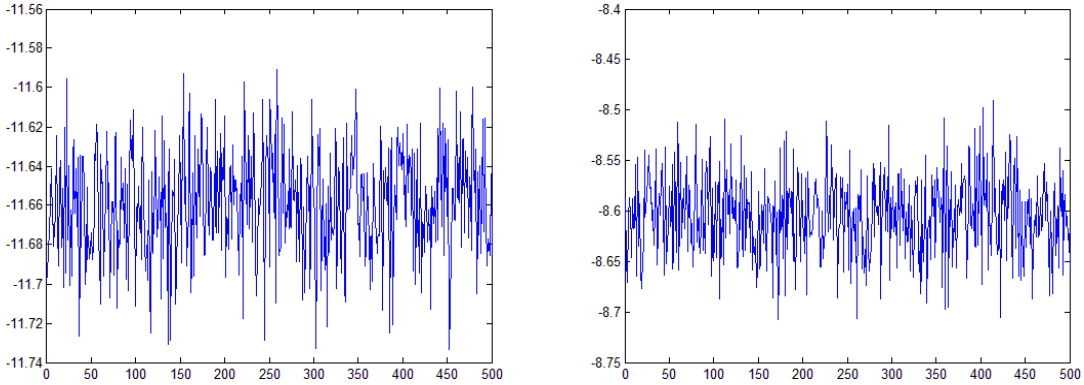


Figure 6.1.18 : $\log_{10}\left(Error^{A}_{Our}\right)$ (Figure gauche) et $\log_{10}\left(Error^{A}_{Sc}\right)$ (Figure droite) pour une matrice de Hankel d'ordre 896.

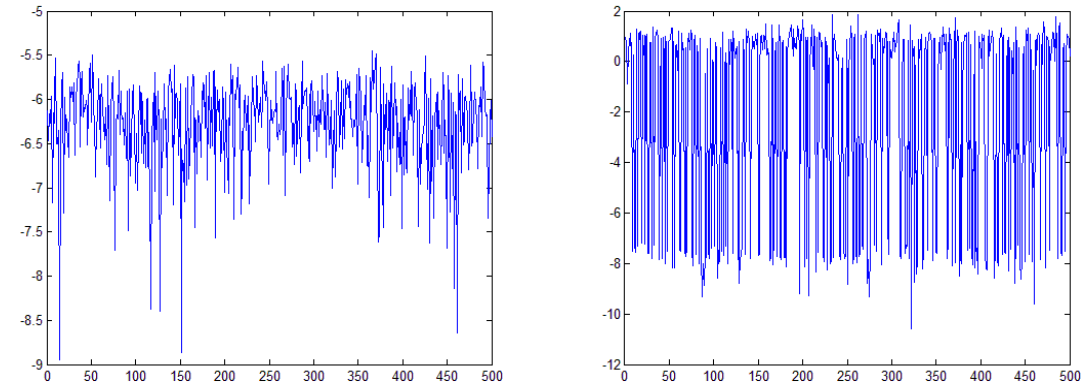


Figure 6.1.19 : $\log_{10}\left(Error^{A}_{Our}\right)$ (Figure gauche) et $\log_{10}\left(Error^{A}_{Sc}\right)$ (Figure droite) pour une matrice de Hankel d'ordre 912.

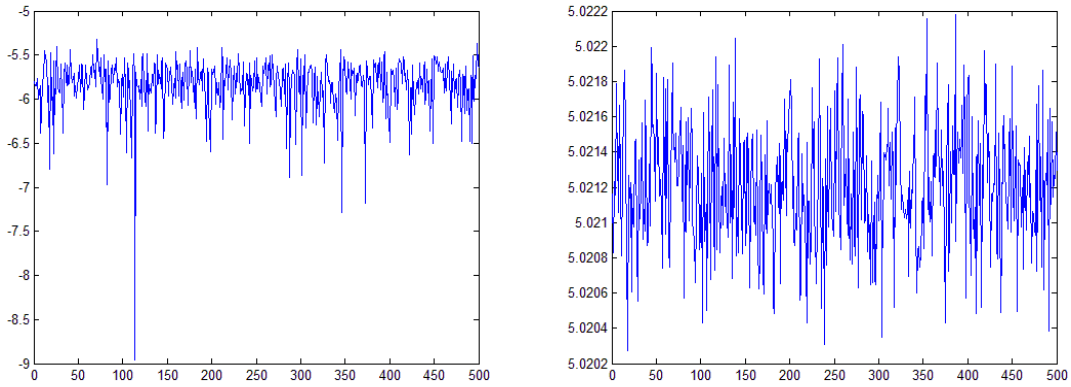


Figure 6.1.20 : $\log_{10}\left(Error^{A}_{Our}\right)$ (Figure gauche) et $\log_{10}\left(Error^{A}_{Sc}\right)$ (Figure droite) pour une matrice de Hankel d'ordre 1024.

Les Figures 6.1.1-20 représentent le comportement d'erreurs

$$\epsilon_1 = \log_{10}\left(Error^{A}_{Our}\right) \text{ et } \epsilon_2 = \log_{10}\left(Error^{A}_{Sc}\right)$$

pour des matrices de Hankel "exacte" par rapport aux 500 perturbations tirées au hasard :

$$\epsilon_3 = \left(k^{(r)} + ik^{(i)}\right) \times 10^{-15}$$

($k^{(i)} = 0$, dans le cas réel), avec $k^{(r)}$, $k^{(i)}$ sont choisis aléatoirement dans $[-1, 1]$.

Nous détaillons par la suite le comportement de ϵ_1 et ϵ_2, par rapport au 500 perturbations tirées au hasard : ϵ_3 et par rapport à une tolérance ϵ aléatoire, pour une matrice de Hankel d'ordre 100.

ϵ		$..., \mathbf{10^{-13}}$	$\mathbf{10^{-12}}$	$\mathbf{10^{-11}}$	$\mathbf{10^{-10}}, ...$
ϵ_1	**Bn**	0	1	70	500
	Mv	500	499	430	0

ϵ		$..., \mathbf{10^{-13}}$	$\mathbf{10^{-12}}$	$\mathbf{10^{-11}}$	$\mathbf{10^{-10}}$	$\mathbf{10^{-9}}, ...$
ϵ_2	**Bn**	0	1	11	311	500
	Mv	500	499	489	189	0

Nous faisons la même chose pour plus d'une vingtaine de matrices de Hankel. Ainsi, la conclusion tirée est la suivante : Sur 500 perturbations aléatoires :

- Pour $|\epsilon| > 10^{-10}$: Notre diagonalisation approchée par blocs et la diagonalisation approchée par blocs via le complément de Schur sont souvent bonnes par rapport à l'exacte ;
- Pour $10^{-13} < |\epsilon| < 10^{-10}$: Notre diagonalisation approchée par blocs et la diagonalisation approchée par blocs via le complément de Schur sont souvent assez bonnes par rapport à l'exacte ;
- Au delà de $\epsilon = 10^{-13}$: Notre diagonalisation approchée par blocs et la diagonalisation approchée par blocs via le complément de Schur sont souvent mauvaises par rapport à l'exacte.

6.2 Stabilité

Pour étudier la stabilité numériques des différents algorithmes, nous donnons dans le Tableau 6.2.1.1 l'erreur relative (6.1), en arithmétique flottante à 15 chiffres, appliquée à plusieurs matrices de Hankel.

n	$Error^R_{Sc}$	$Error^R_{Our}$
9	1.510307622052787e-012	5.109279845177994e-010
12	4.809398388692218e-012	5.764191878422235e-011
25	1.747953348257615e-009	7.421166662905649e-012
54	1.233053124848335e-006	6.109332631185916e-011
75	1.309215931690797e-009	1.282297161691120e-010
100	3.670817931625318e-010	2.299119730115441e-010
150	1.237035121810071e-009	1.401548852003868e-012
200	2.253404023014869e-010	2.616922818264540e-011
400	5.567600917584636e-012	1.459114304302858e-012
512	9.208559684775261e-011	4.559640870404542e-012
750	5.592250775411315e-010	3.950105207782781e-009
912	3.868641897579982e-011	1.025429171691832e-011
1024	1.698861841529736e-009	3.734510428892953e-012

Table 6.2.1.1 : l'erreur relative (6.1), en arithmétique flottante à 15 chiffres, pour des matrices de Hankel.

Ainsi, nous remarquons que la méthode au moyen de Toeplitz et la méthode du complément de Schur sont numériquement stables.

Nous représentons par la suite le comportement de la stabilité pour les coefficients de quelques matrices de Hankel (Figure 6.2.1.1-13).

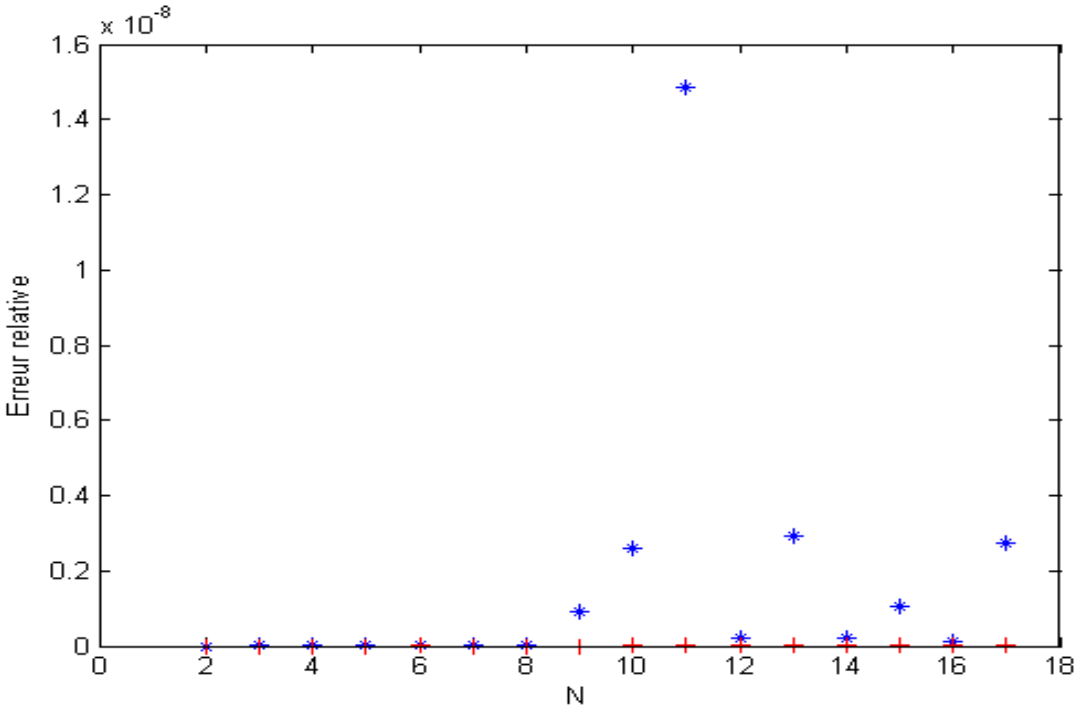


Figure 6.2.1.1 : Erreur relative ($+$: $Schur$, $*$: $Toeplitz$)
pour une matrice de Hankel d'ordre 9.

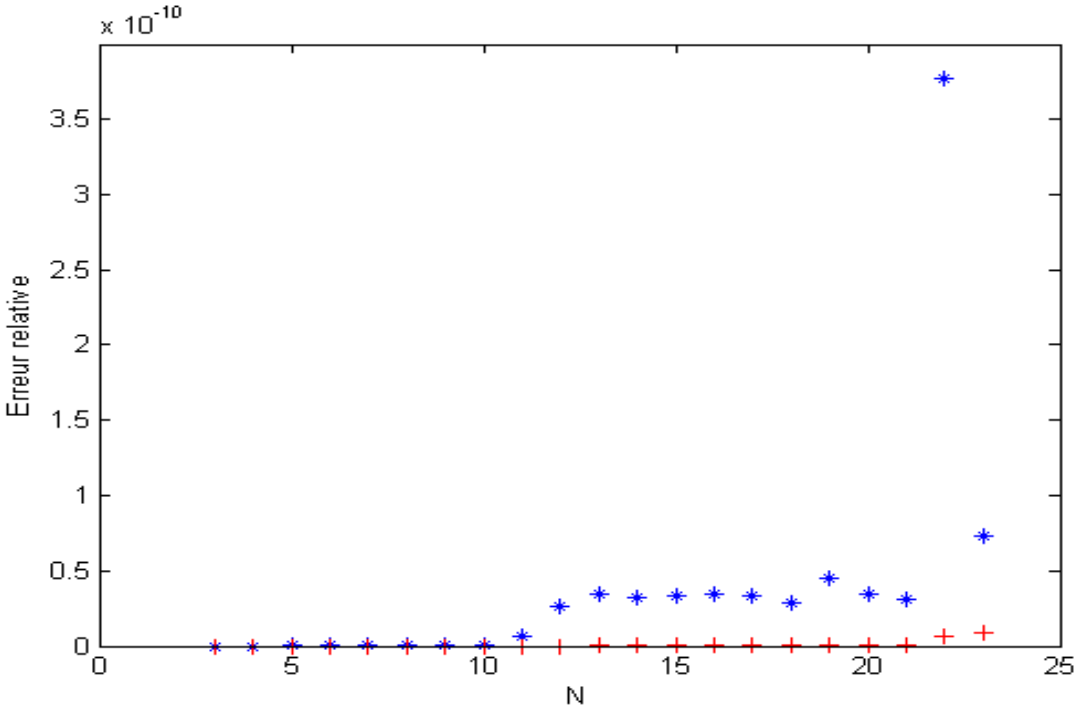


Figure 6.2.1.2 : Erreur relative ($+$: $Schur$, $*$: $Toeplitz$)
pour une matrice de Hankel d'ordre 12.

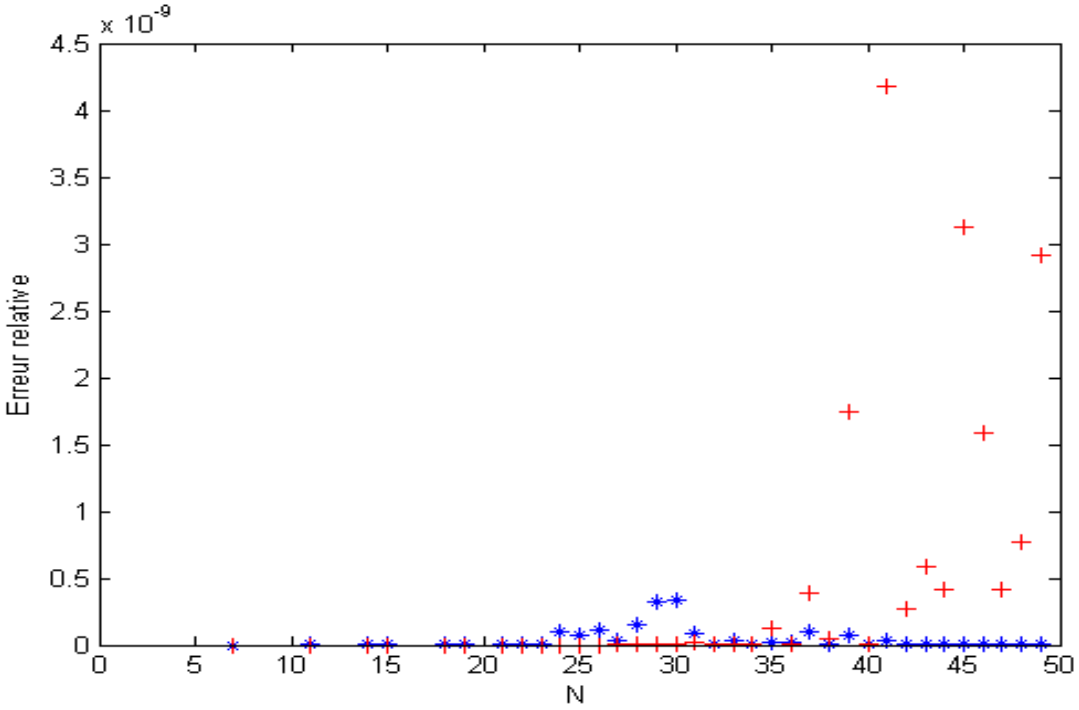


Figure 6.2.1.3 : Erreur relative ($+$: $Schur$, $*$: $Toeplitz$)
pour une matrice de Hankel d'ordre 25.

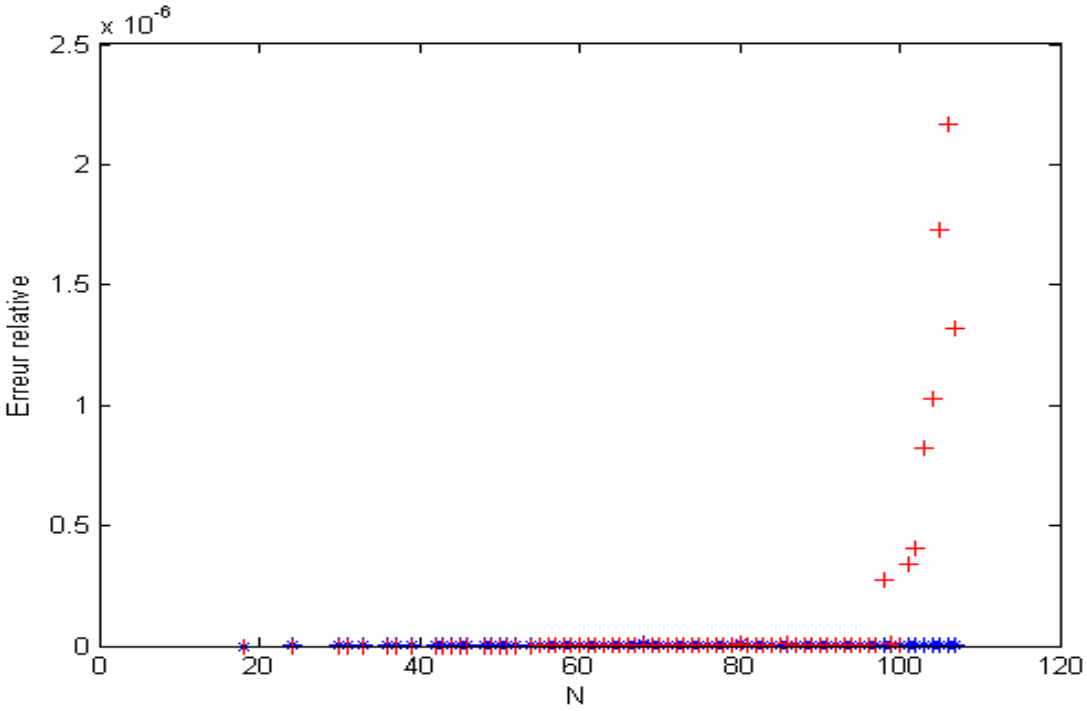


Figure 6.2.1.4 : Erreur relative ($+$: $Schur$, $*$: $Toeplitz$)
pour une matrice de Hankel d'ordre 54.

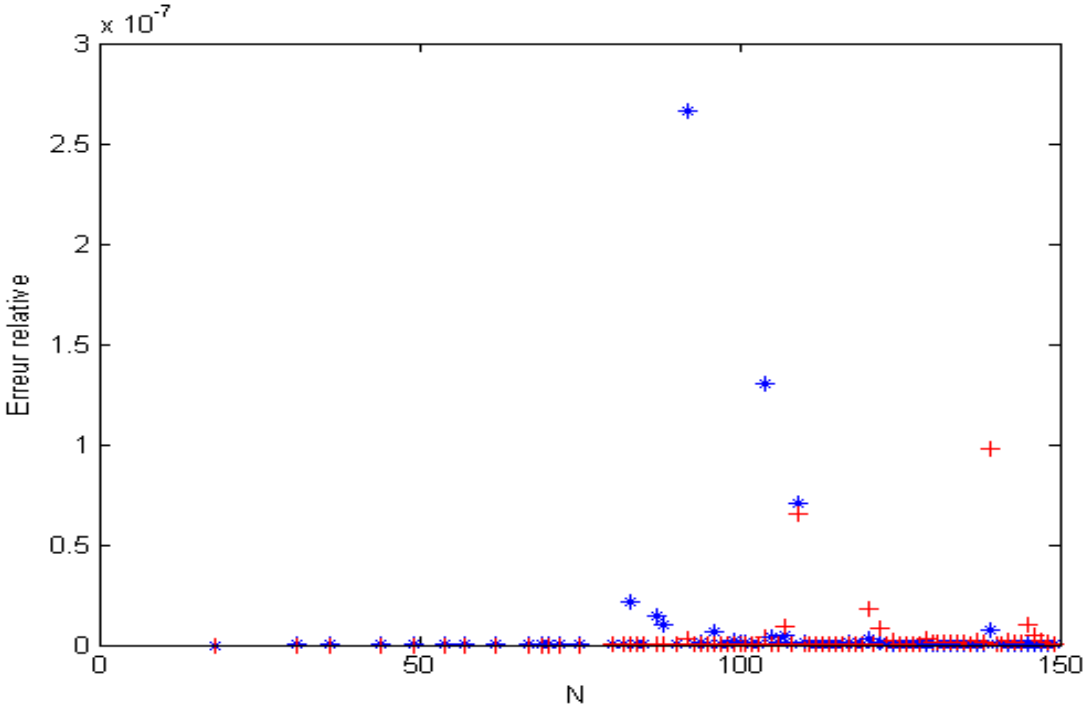


Figure 6.2.1.5 : Erreur relative ($+$: $Schur$, $*$: $Toeplitz$)
pour une matrice de Hankel d'ordre 75.

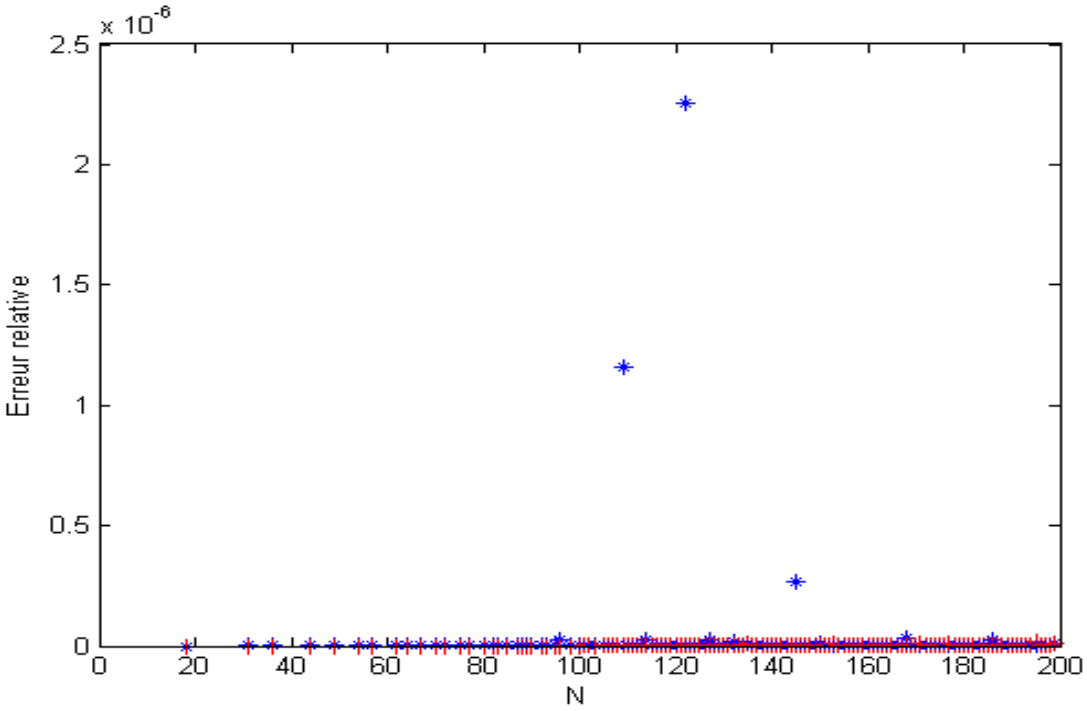


Figure 6.2.1.6 : Erreur relative ($+$: $Schur$, $*$: $Toeplitz$)
pour une matrice de Hankel d'ordre 100.

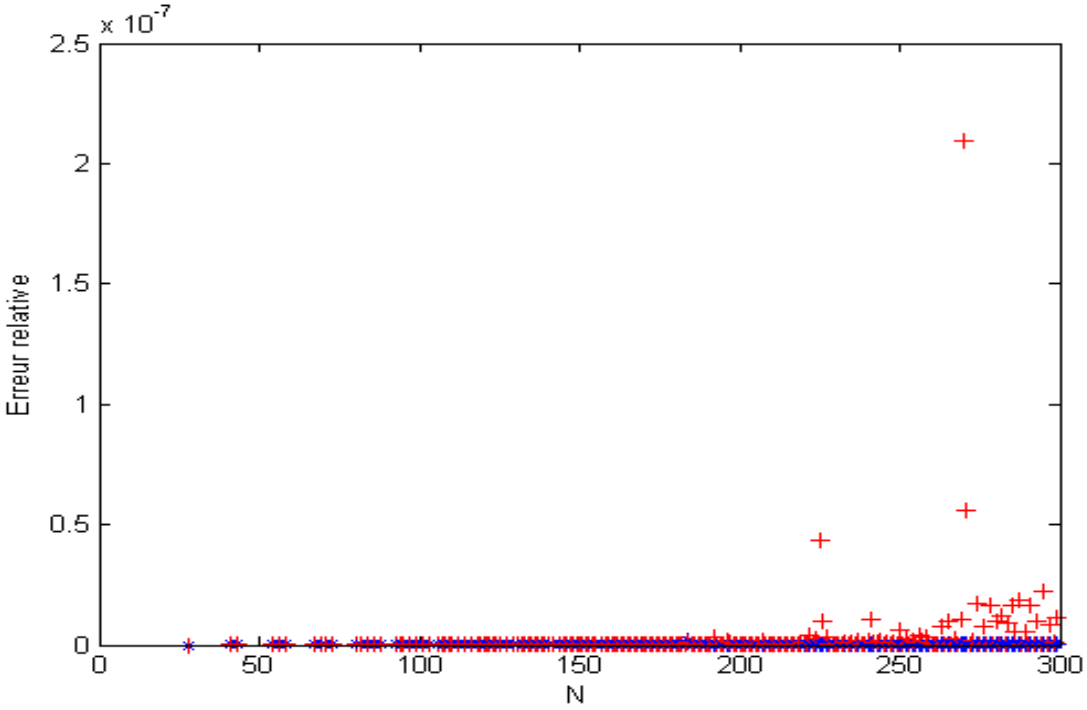


Figure 6.2.1.7 : Erreur relative ($+$: $Schur$, $*$: $Toeplitz$)
pour une matrice de Hankel d'ordre 150.

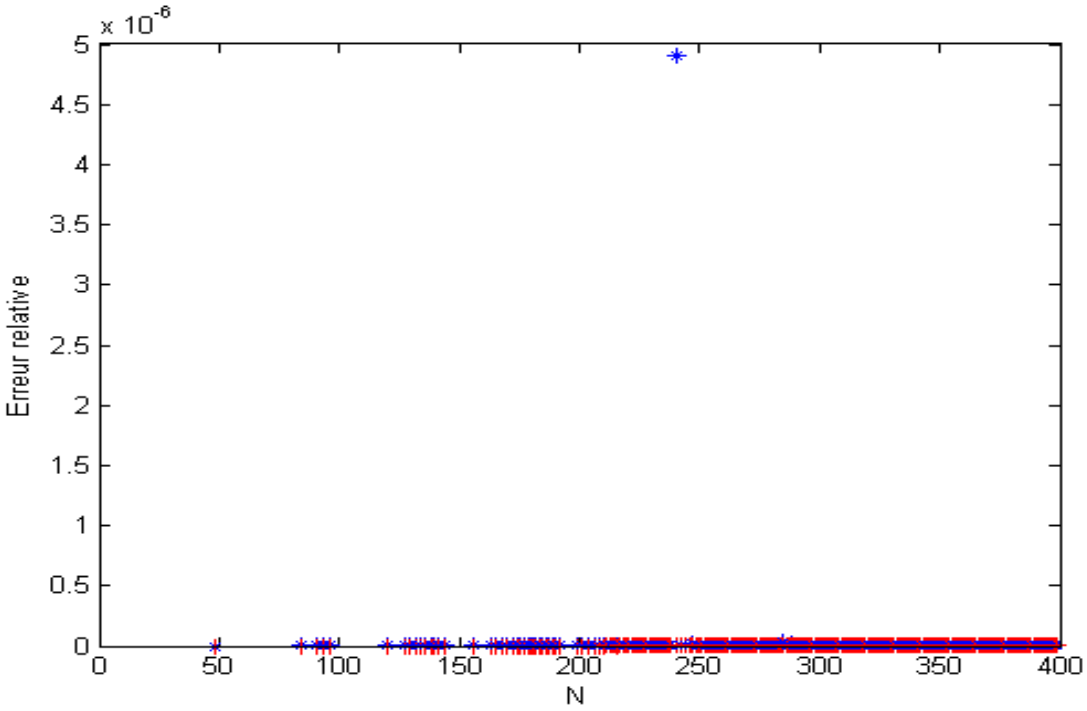


Figure 6.2.1.8 : Erreur relative ($+$: $Schur$, $*$: $Toeplitz$)
pour une matrice de Hankel d'ordre 200.

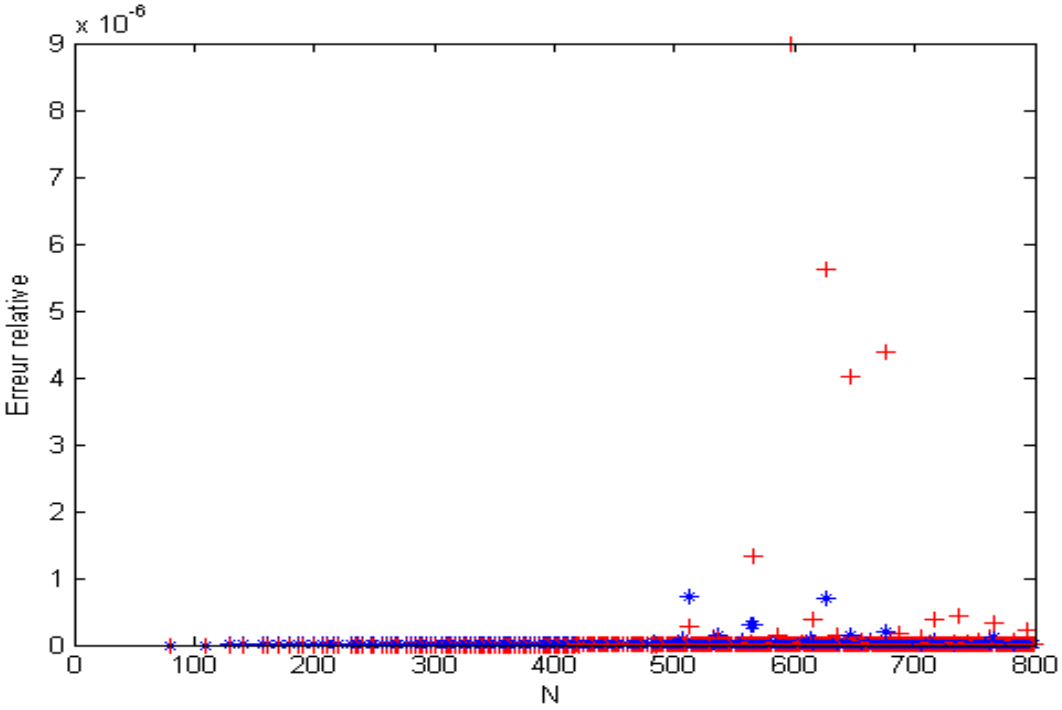


Figure 6.2.1.9 : Erreur relative ($+$: $Schur$, $*$: $Toeplitz$)
pour une matrice de Hankel d'ordre 400.

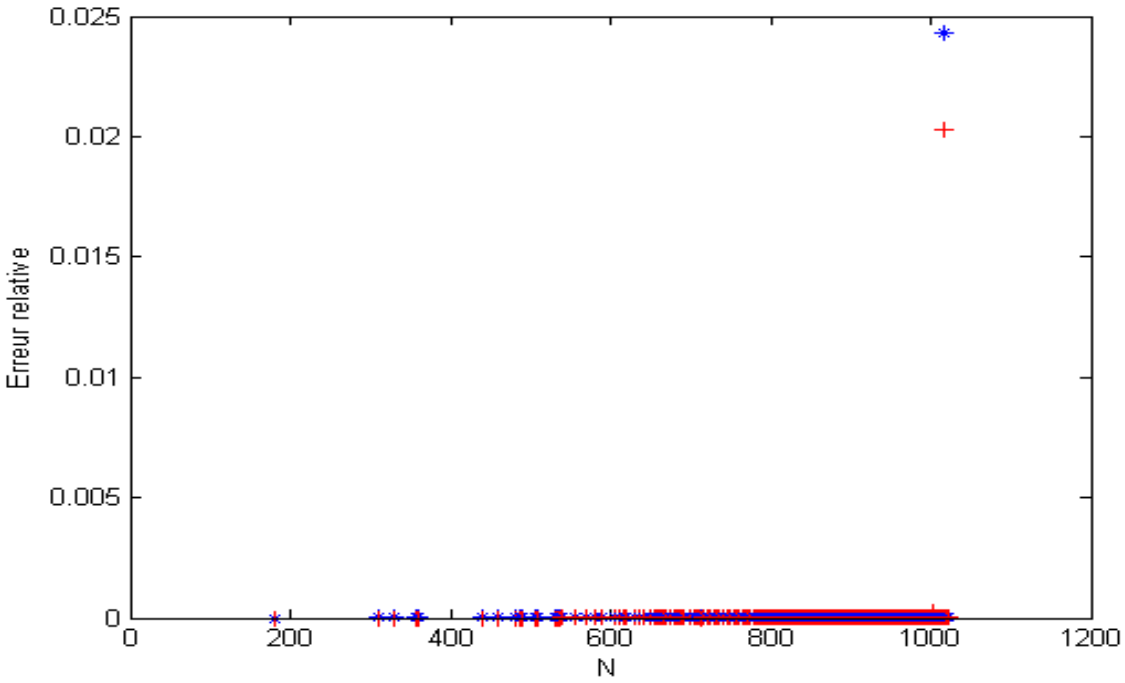


Figure 6.2.1.10 : Erreur relative ($+$: $Schur$, $*$: $Toeplitz$)
pour une matrice de Hankel d'ordre 512.

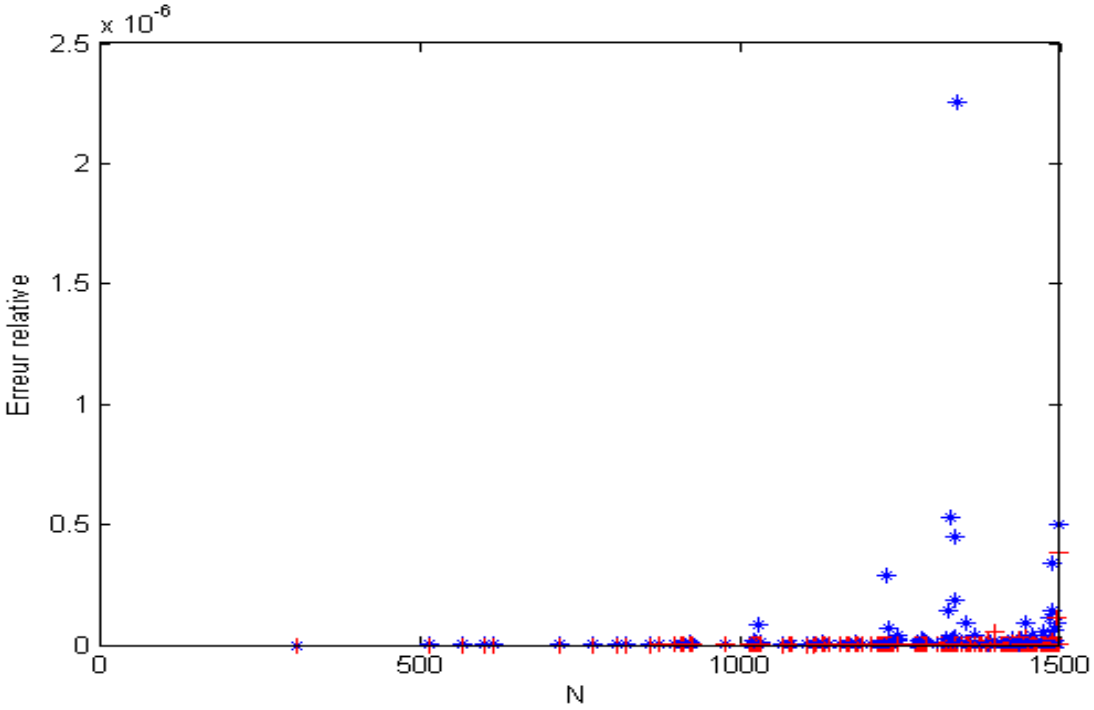


Figure 6.2.1.11 : Erreur relative ($+$: $Schur$, $*$: $Toeplitz$)
pour une matrice de Hankel d'ordre 750.

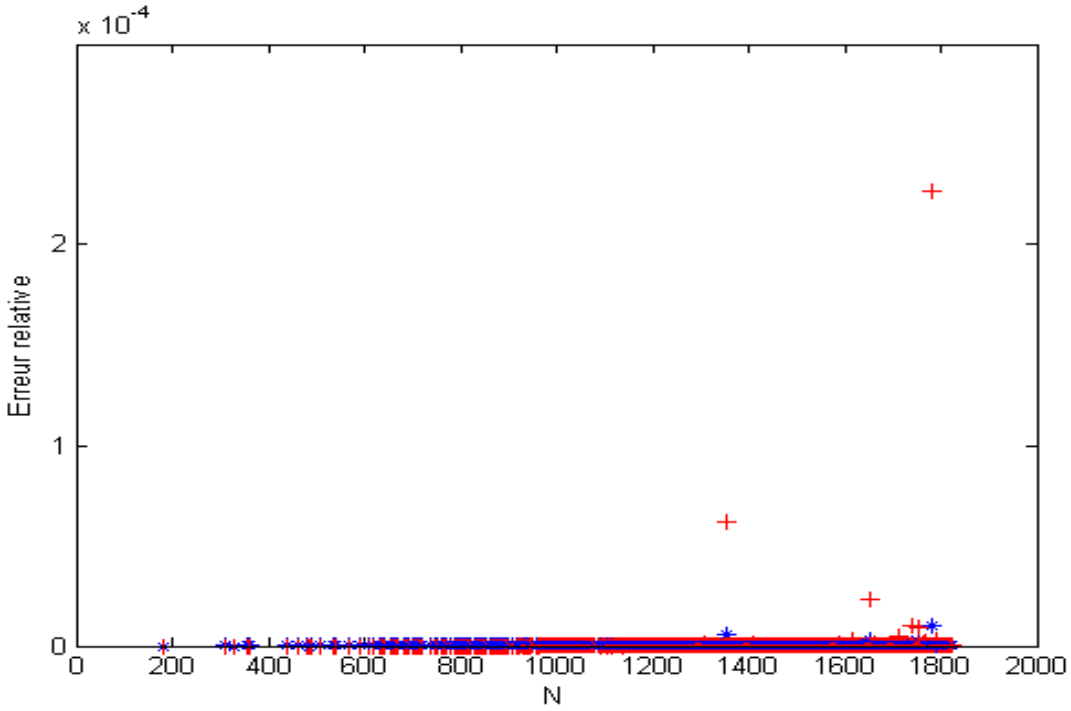


Figure 6.2.1.12 : Erreur relative ($+$: $Schur$, $*$: $Toeplitz$)
pour une matrice de Hankel d'ordre 912.

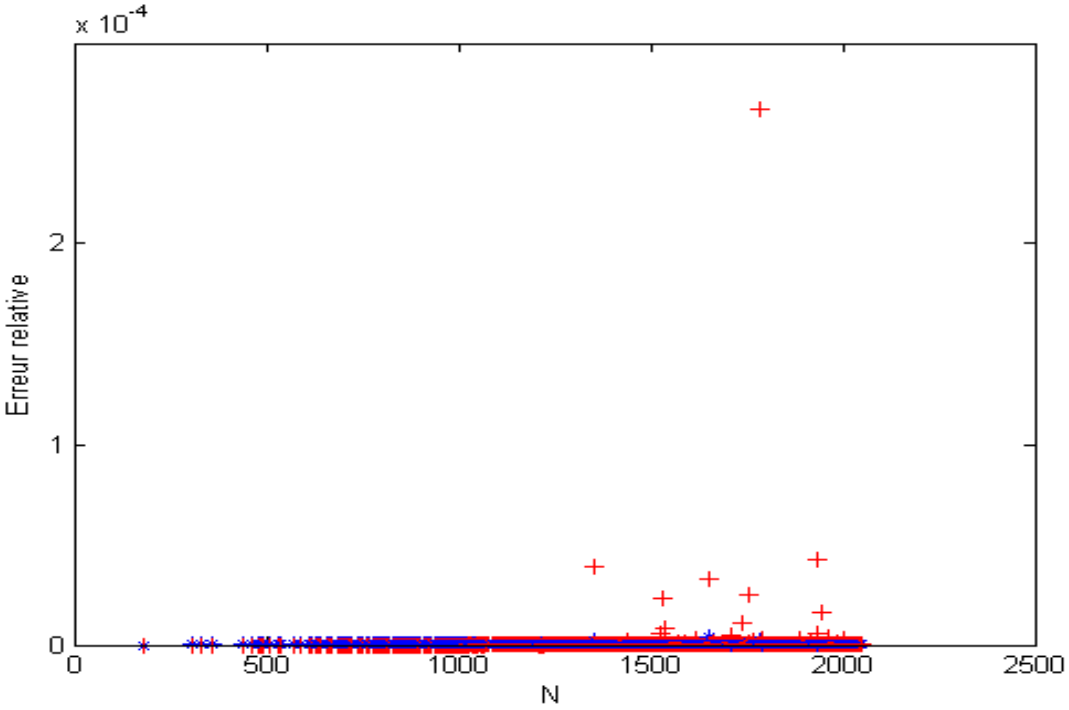


Figure 6.2.1.13 : Erreur relative ($+$: *Schur*, $*$: *Toeplitz*)
pour une matrice de Hankel d'ordre 1024.

6.3 Temps de calcul

Table 6.3.1.1 et Table 6.3.1.2 examinent les temps de calcul (en seconde) de la diagonalisation approchée par blocs au moyen de Toeplitz et la diagonalisation approchée par blocs via le complément de Schur, respectivement, pour des matrices de Hankel avec et sans la SVD par rapport à une tolérance fixe $\varepsilon = 10^{-5}$.

n	$\mathbf{Temps}^{\mathbf{SVD}}_{\mathbf{Our\ appr}}$	$\mathbf{Temps}^{\mathbf{SVD}}_{\mathbf{Our\ exact}}$	$\mathbf{Temps}_{\mathbf{Our\ exact}}$	$\mathbf{Temps}_{\mathbf{Our\ appr}}$
7	0.013784	0.015731	0.012106	0.013959
9	0.005041	0.009369	0.006737	0.003778
12	0.003694	0.006646	0.006274	0.003319
14	0.018998	0.008483	0.010342	0.015461
25	0.010001	0.016168	0.011962	0.006746
28	0.048379	0.035011	0.051968	0.026789
54	0.040909	0.038674	0.021410	0.023656
56	0.043103	0.068067	0.054062	0.056698
75	0.065573	0.066932	0.044845	0.048835
100	0.116058	0.120270	0.072986	0.069949
112	0.185891	0.149440	0.293679	0.176010
150	0.261549	0.255269	0.181619	0.157348
200	0.476486	0.401817	0.281143	0.271091
242	1.149094	0.760889	1.366672	0.936976
336	9.680564	4.092368	2.030033	1.863021
400	5.901191	3.818207	1.151003	1.134041
448	10.276467	5.903743	4.589068	4.801048
500	6.790398	4.590035	2.183397	2.180653
672	105.021364	56.535336	12.354240	12.138629
750	30.372912	22.561397	5.527707	5.387997
896	125.227031	75.885985	30.199463	26.784229
912	53.094547	41.047456	9.046823	9.584548
1024	57.091103	46.090088	12.966118	12.959254
1120	152.067646	102.173770	55.338688	54.851836

Table 6.3.1.1 : Temps de calcul (en sec) de notre diagonalisation approchée par blocs pour des matrices de Hankel.

n	$\text{Temps}^{\text{SVD}}_{\text{Schur appr}}$	$\text{Temps}^{\text{SVD}}_{\text{Schur exact}}$	$\text{Temps}_{\text{Schur exact}}$	$\text{Temps}_{\text{Schur appr}}$
7	0.014190	0.016560	0.018965	0.011146
9	0.004275	0.006533	0.005034	0.002154
12	0.022002	0.005399	0.005536	0.002173
14	0.013738	0.012452	0.049608	0.005168
25	0.006538	0.008040	0.005822	0.003169
28	0.029112	0.050293	0.016122	0.008922
54	0.034352	0.033973	0.015061	0.010197
56	0.051475	0.057963	0.010327	0.011143
75	0.046530	0.423010	0.017437	0.015490
100	0.058025	0.043565	0.026248	0.024111
112	0.170153	0.601587	0.056575	0.061901
150	0.473925	0.457818	0.049533	0.045800
200	0.249735	0.216793	0.086301	0.082909
242	2.405415	1.868428	0.191766	0.333824
336	9.198089	4.424583	0.894088	0.907123
400	4.894257	4.292934	0.558503	0.861849
448	10.045758	6.004023	1.477244	1.771594
500	3.637118	3.507734	1.096444	1.274711
672	107.695967	59.327398	5.156941	6.529818
750	29.304004	21.330485	3.435513	3.421549
896	127.460937	79.220462	14.839375	14.031458
912	49.795891	41.798264	5.266518	5.131951
1024	53.911682	44.731867	7.040860	7.442701
1120	155.248917	104.534042	27.831622	27.363059

Table 6.3.1.2 : Temps de calcul (en sec) de la diagonalisation approchée par blocs via le complément de Schur pour des matrices de Hankel.

D'où, nous observons que le temps d'exécution nécessaire par notre méthode est meilleure par rapport à la méthode de complément de Schur.

Nous comparons par la suite le comportement du temps de calcul, répété 20 fois, des différents algorithmes appliqués à quelques matrices de Hankel (Figure 6.3.1.1-10).

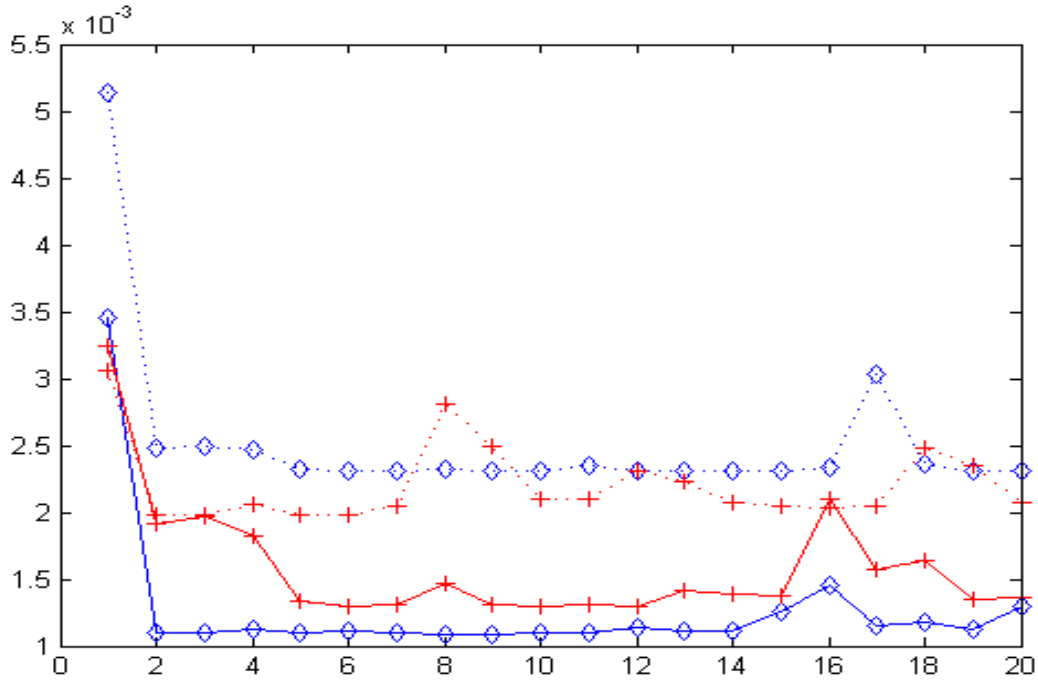


Figure 6.3.1.1 : Temps de calcul ($+$: *Schur* , $\Diamond$: *Toeplitz*)
(— : Sans SVD, $\cdots$: Avec SVD) pour une matrice de Hankel d'ordre 7.

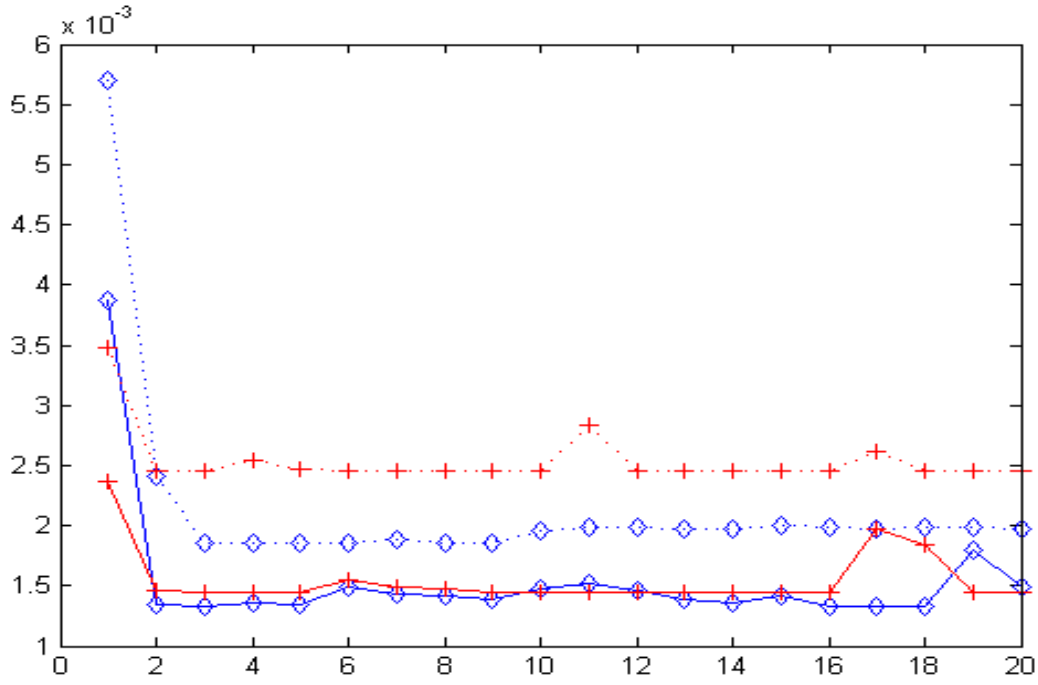


Figure 6.3.1.2 : Temps de calcul ($+$: *Schur* , $\Diamond$: *Toeplitz*)
(— : Sans SVD, $\cdots$: Avec SVD) pour une matrice de Hankel d'ordre 12.

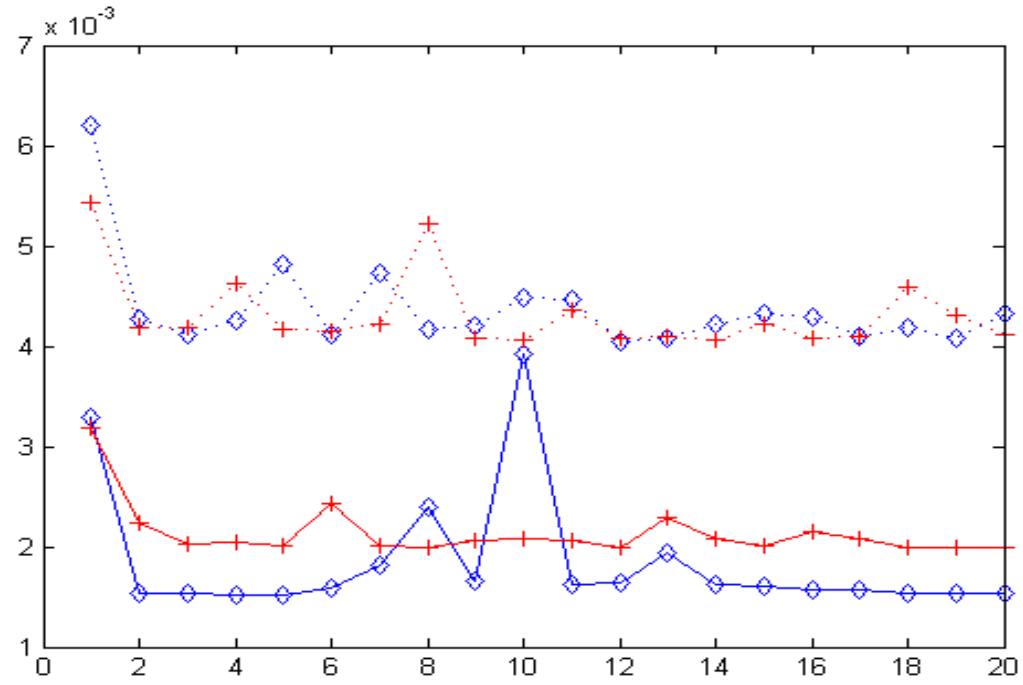


Figure 6.3.1.3 : Temps de calcul ($+$: *Schur* , $\Diamond$: *Toeplitz*)
(— : Sans SVD, $\cdots$: Avec SVD) pour une matrice de Hankel d'ordre 25.

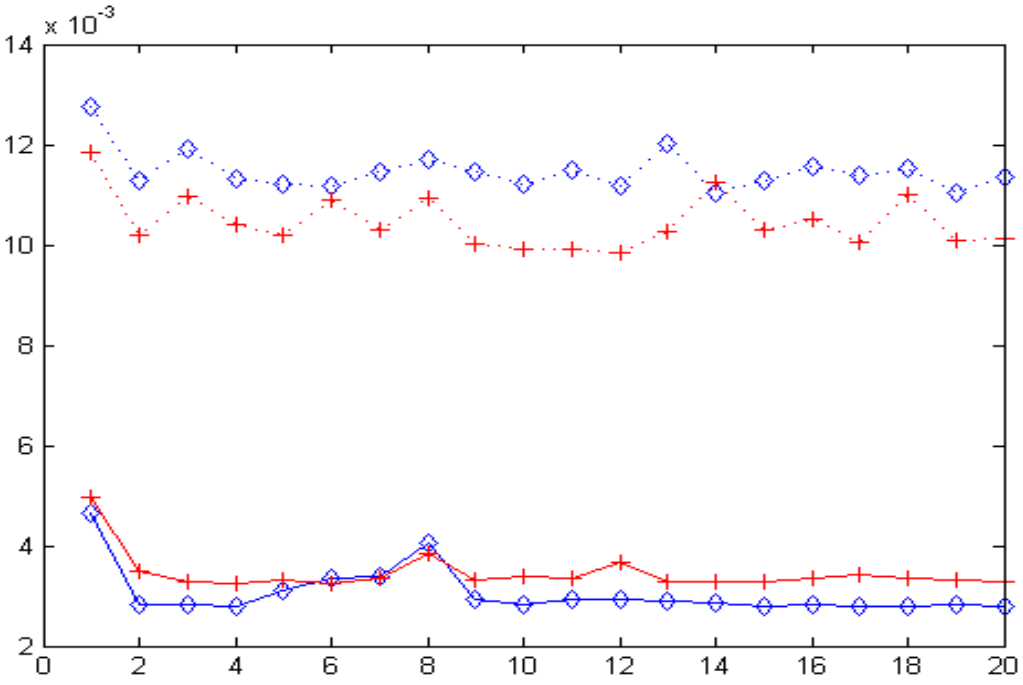


Figure 6.3.1.4 : Temps de calcul ($+$: *Schur* , $\Diamond$: *Toeplitz*)
(— : Sans SVD, $\cdots$: Avec SVD) pour une matrice de Hankel d'ordre 56.

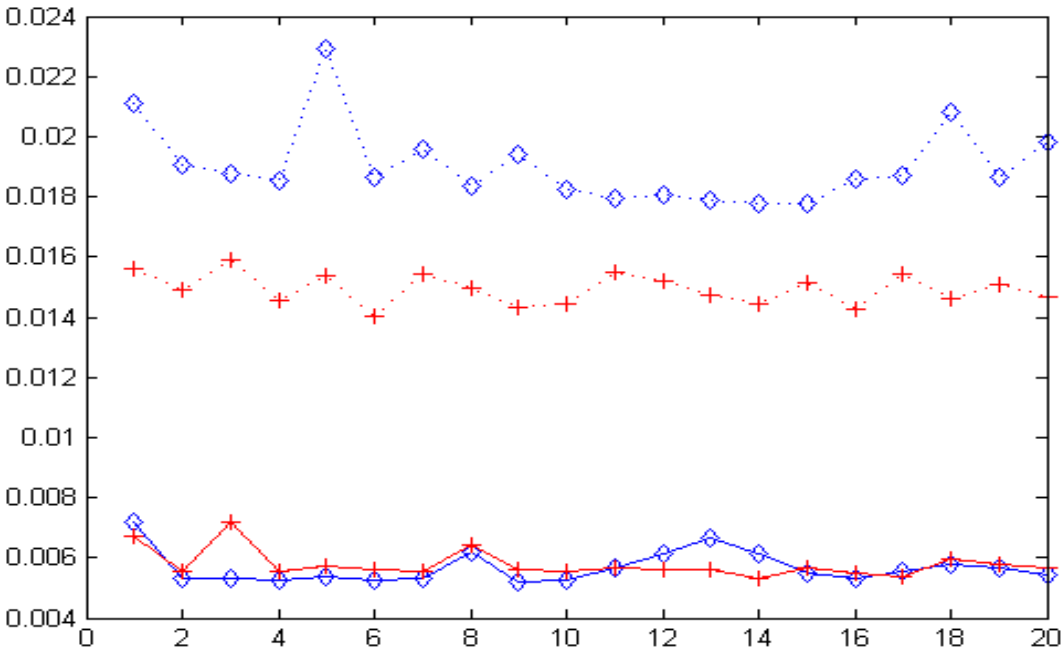


Figure 6.3.1.5 : Temps de calcul ($+$: *Schur* , $\Diamond$: *Toeplitz*)
(— : Sans SVD, $\cdots$: Avec SVD) pour une matrice de Hankel d'ordre 75.

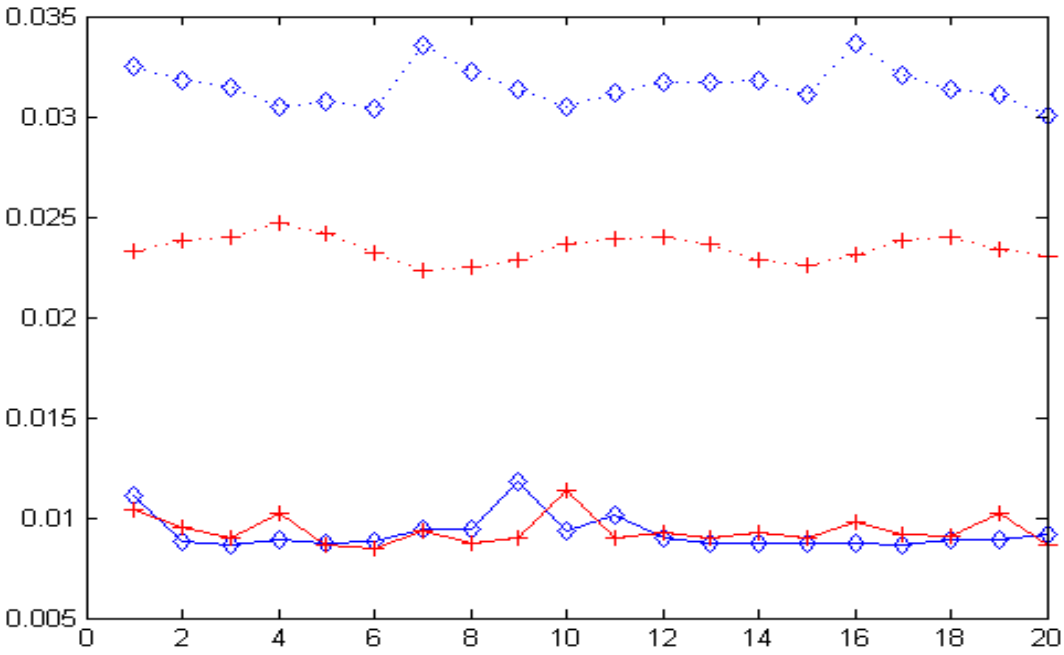


Figure 6.3.1.6 : Temps de calcul ($+$: *Schur* , $\Diamond$: *Toeplitz*)
(— : Sans SVD, $\cdots$: Avec SVD) pour une matrice de Hankel d'ordre 112.

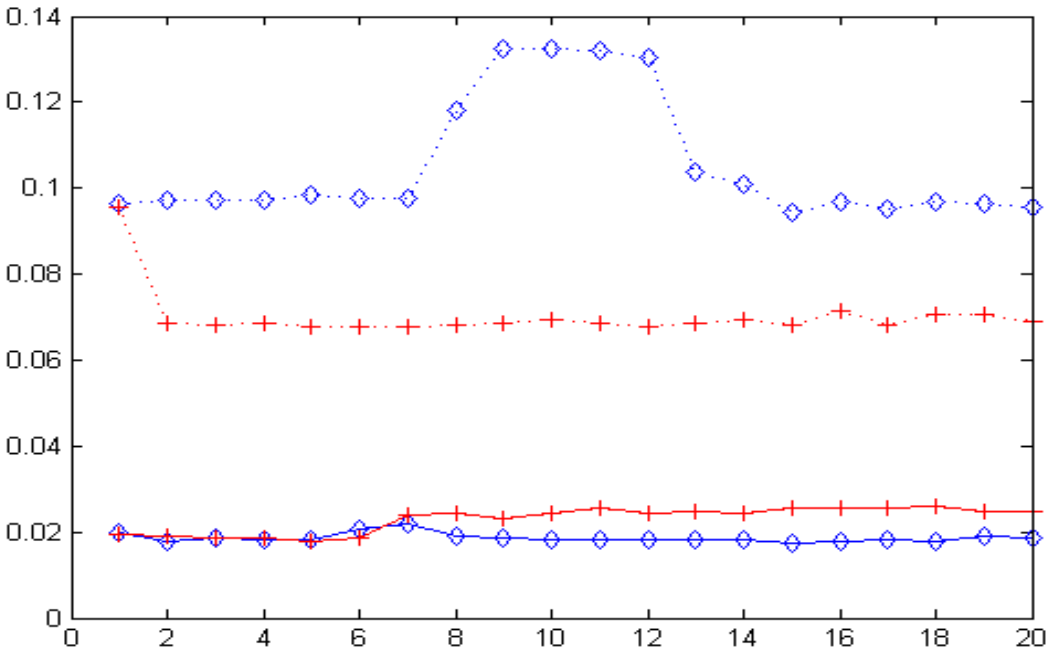


Figure 6.3.1.7 : Temps de calcul ($+$: *Schur* , $\Diamond$: *Toeplitz*)
(— : Sans SVD, $\cdots$: Avec SVD) pour une matrice de Hankel d'ordre 150.

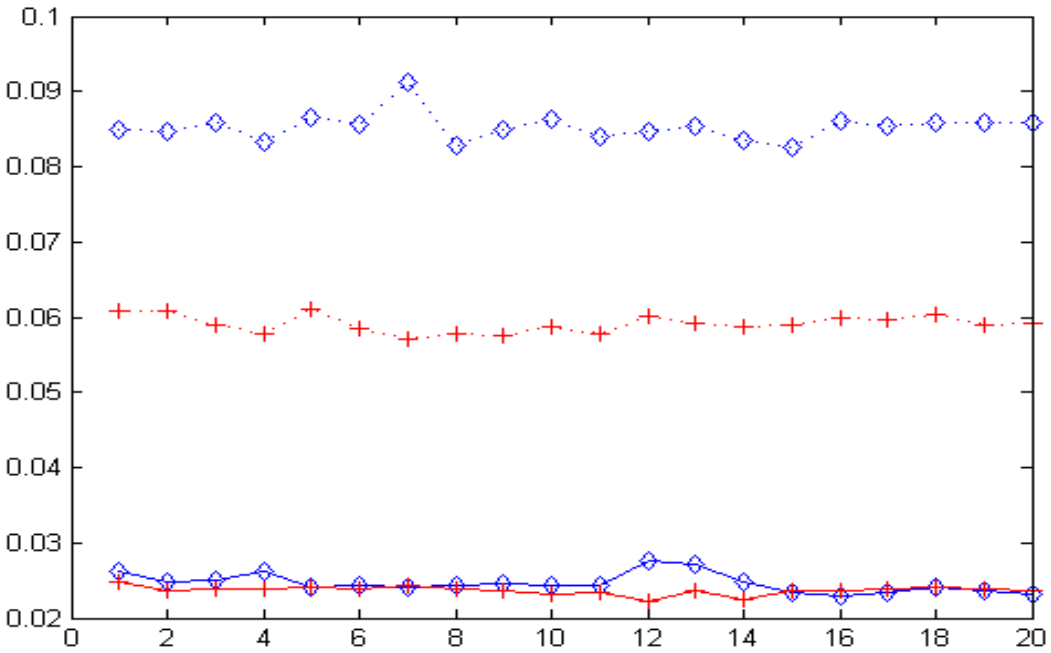


Figure 6.3.1.8 : Temps de calcul ($+$: *Schur* , $\Diamond$: *Toeplitz*)
(— : Sans SVD, $\cdots$: Avec SVD) pour une matrice de Hankel d'ordre 200.

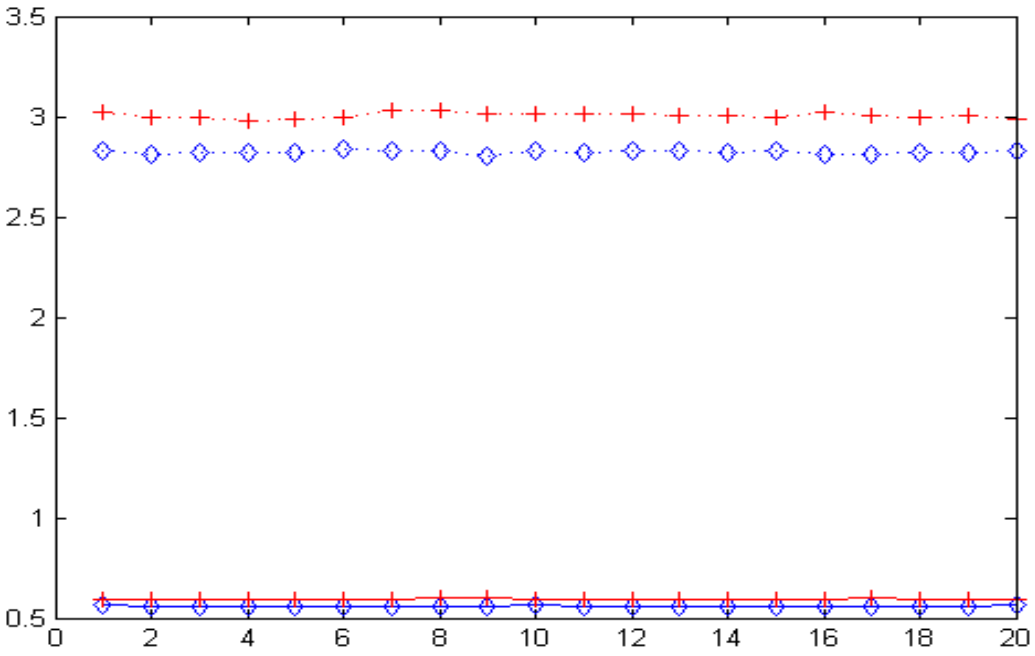


Figure 6.3.1.9 : Temps de calcul ($+$: $Schur$, $\Diamond$: $Toeplitz$)
(— : Sans SVD, $\cdots$: Avec SVD) pour une matrice de Hankel d'ordre 400.

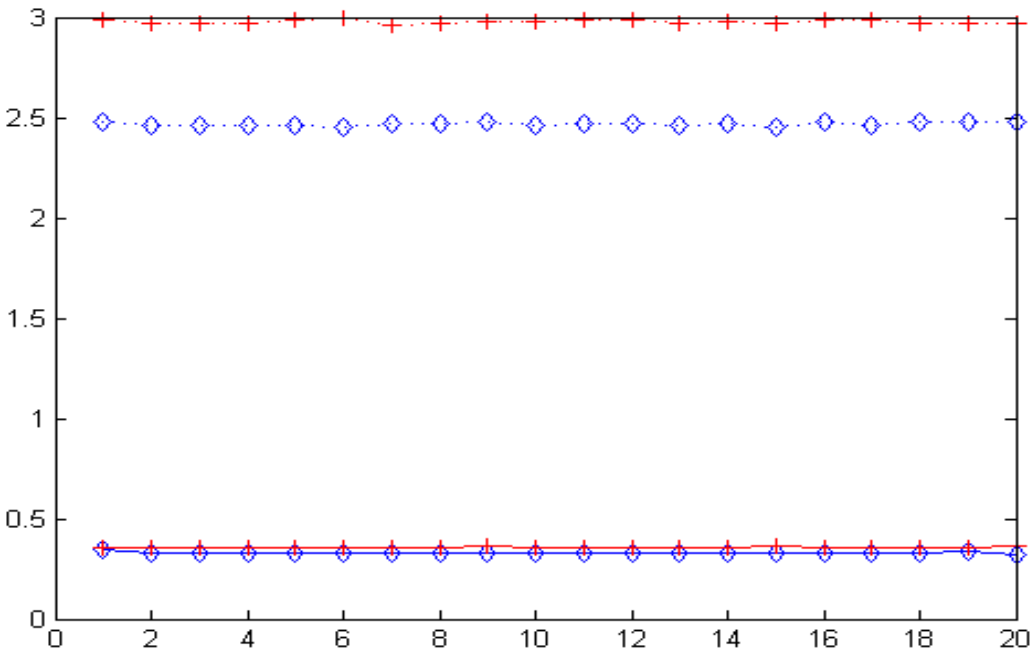


Figure 6.3.1.10 : Temps de calcul ($+$: $Schur$, $\Diamond$: $Toeplitz$)
(— : Sans SVD, $\cdots$: Avec SVD) pour une matrice de Hankel d'ordre 448.

6.4 Influence d'inverser les Toeplitz-supérieures

Nous allons maintenant expliquer l'influence d'appliquer l'inversion approchée d'une matrice triangulaire de Toeplitz au lieu de l'inversion exacte dans les étapes intermédiaires de l'Algorithme 2.2.2 de diagonalisation par blocs de matrices de Hankel.

Nous étudions ainsi le comportement de la matrice diagonale par blocs approchée par rapport à l'exacte en représentant, dans les Figures 6.4.1-10, l'erreur absolue au moyen de trois inversions : Bini, Matlab et Substitution données respectivement par :

$$\log_{10}(\|D_{Bini} - D_{exacte}\|_1),\ \ \log_{10}(\|D_{Matlab} - D_{exacte}\|_1),\ \ \log_{10}(\|D_{Sub} - D_{exacte}\|_1)$$

tels que D_{Bini}, D_{Matlab} et D_{Sub} sont les matrices diagonales par blocs, 50 fois perturbées aléatoirement, au moyen de Bini, Matlab et Substitution, respectivement et D_{exacte} est la matrice diagonale par blocs exacte (Sans perturber les données initiales).

Nous rappelons que cette comparaison a été appliquée à quelques matrices de Hankel.

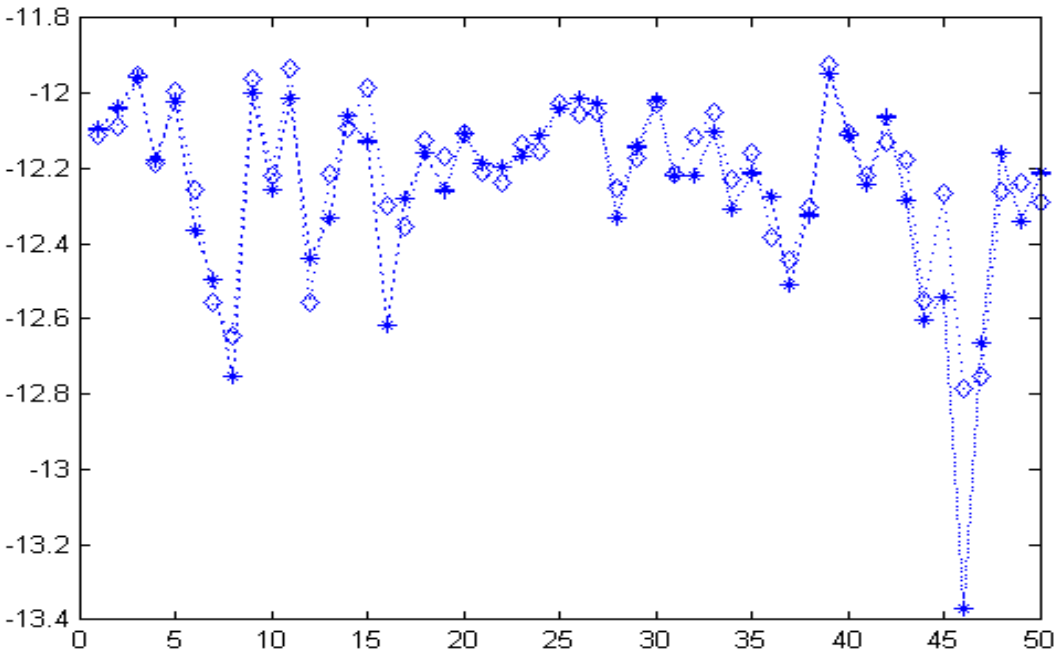


Figure 6.4.1.1 : Efficacité de l'inversion des Toeplitz-supérieurs de l'Algorithme 2.2.2 ($\Diamond$: Bini, $+$: Substitution, $*$: Matlab) pour une matrice de Hankel d'ordre 54.

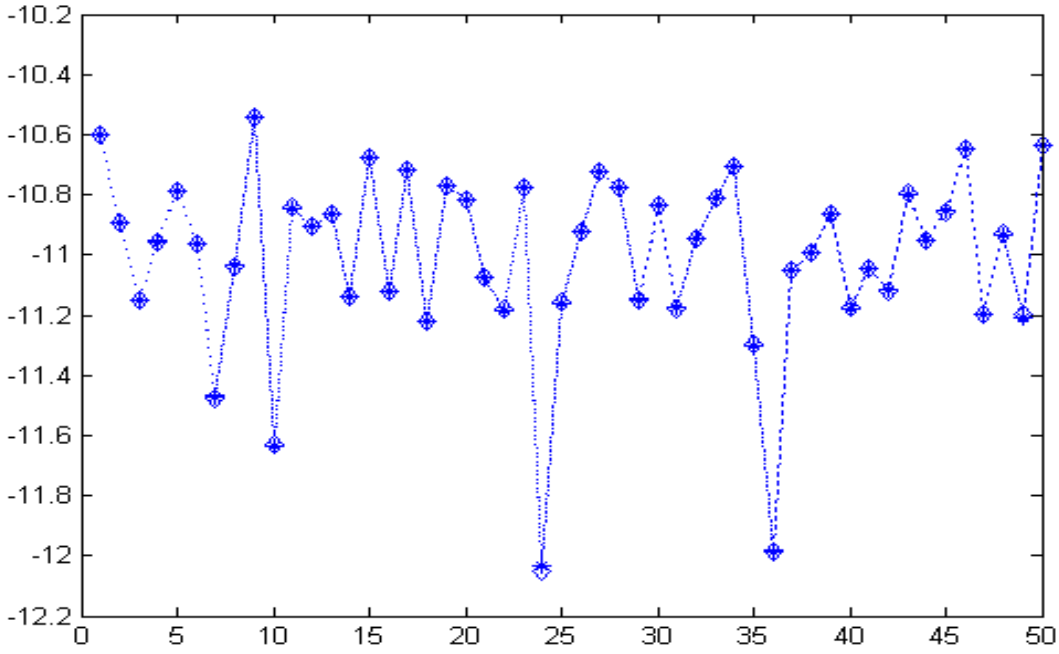


Figure 6.4.1.2 : Efficacité de l'inversion des Toeplitz-supérieurs de l'Algorithme 2.2.2
($\Diamond$: Bini, $+$: Substitution, $*$: Matlab) pour une matrice de Hankel d'ordre 75.

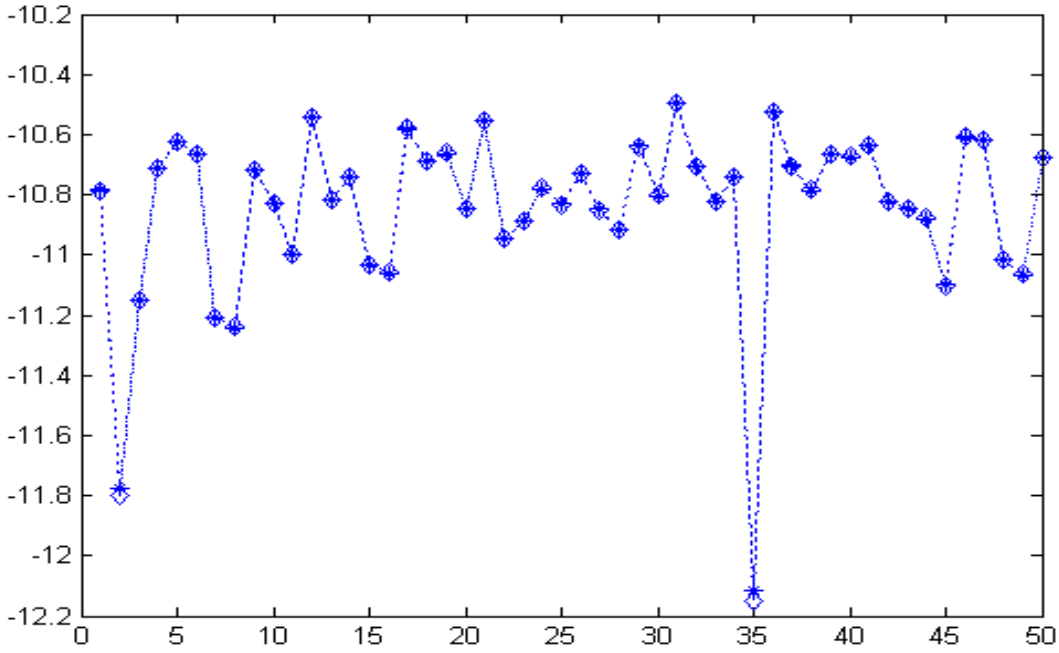


Figure 6.4.1.3 : Efficacité de l'inversion des Toeplitz-supérieurs de l'Algorithme 2.2.2
($\Diamond$: Bini, $+$: Substitution, $*$: Matlab) pour une matrice de Hankel d'ordre 100.

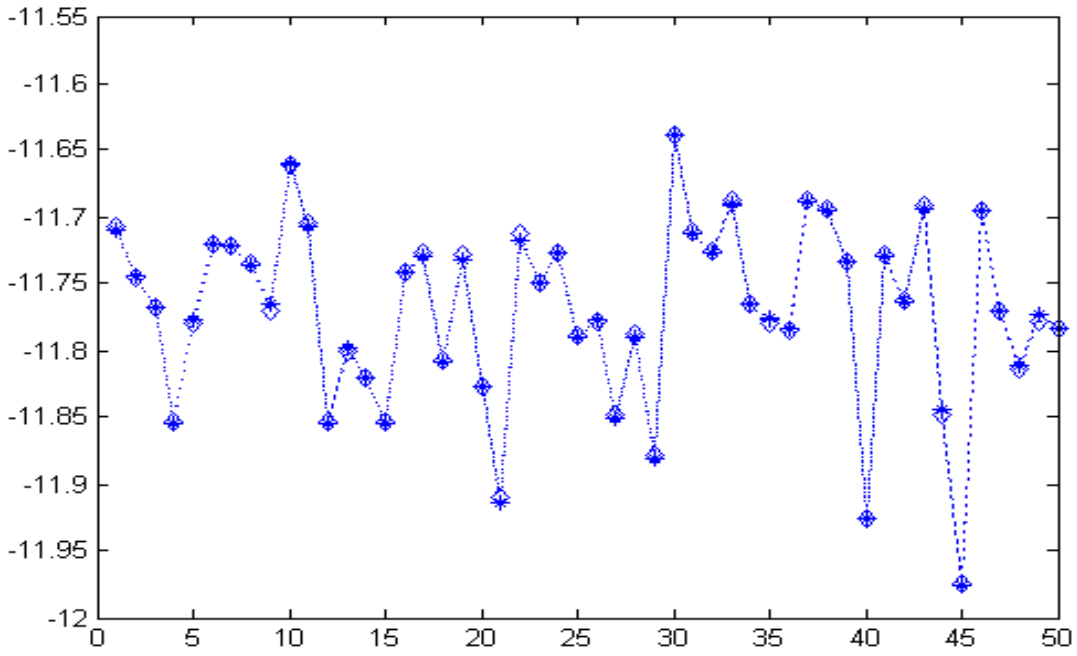


Figure 6.4.1.4 : Efficacité de l'inversion des Toeplitz-supérieurs de l'Algorithme 2.2.2 ($\Diamond$: Bini, $+$: Substitution, $*$: Matlab) pour une matrice de Hankel d'ordre 150.

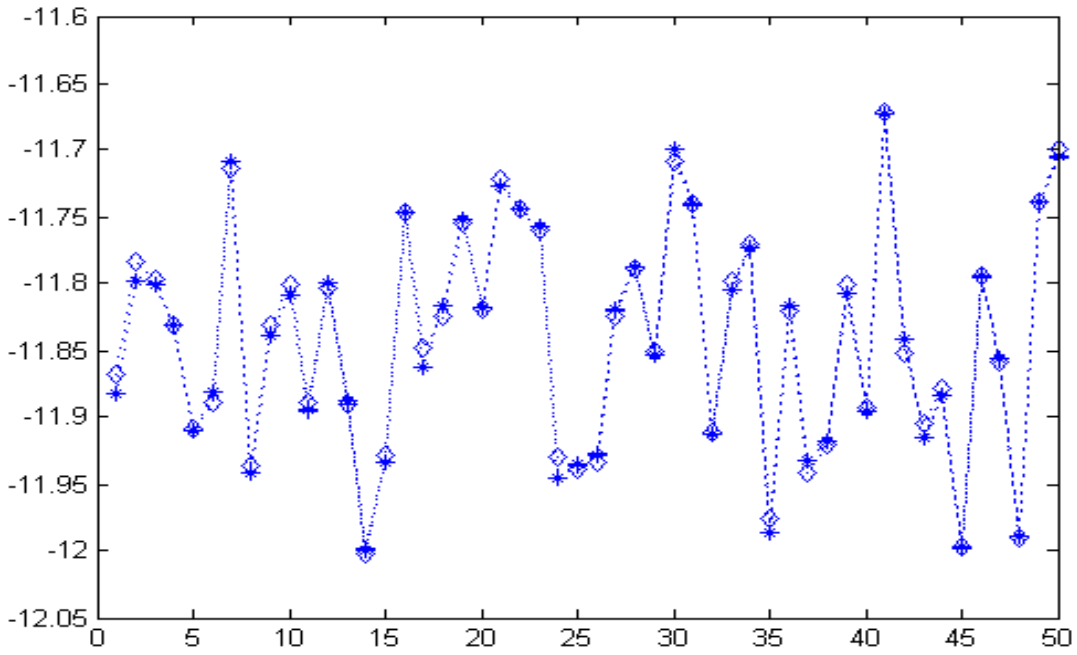


Figure 6.4.1.5 : Efficacité de l'inversion des Toeplitz-supérieurs de l'Algorithme 2.2.2 ($\Diamond$: Bini, $+$: Substitution, $*$: Matlab) pour une matrice de Hankel d'ordre 200.

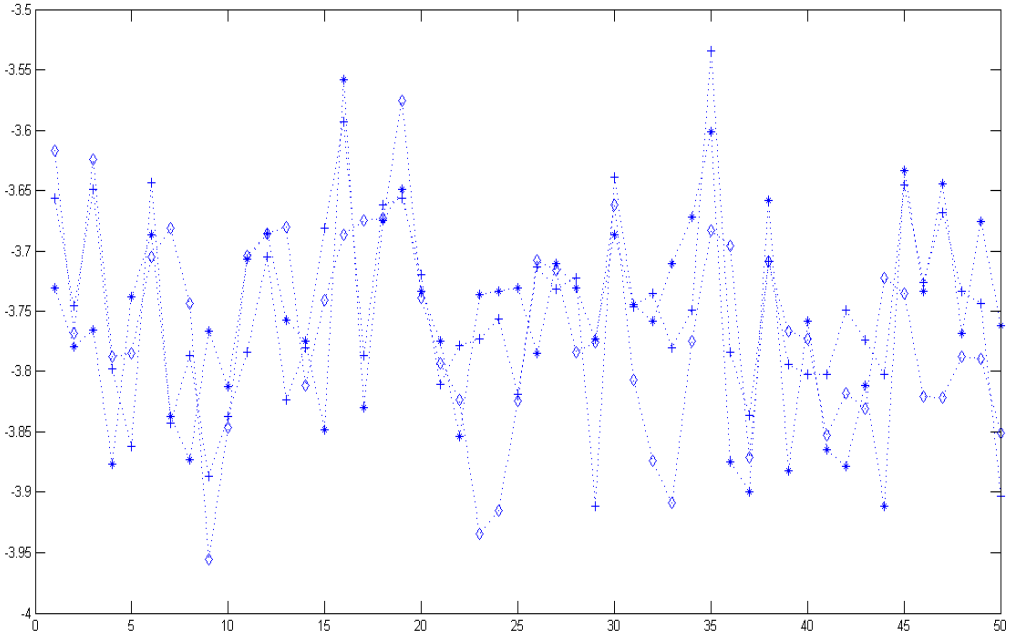


Figure 6.4.1.6 : Efficacité de l'inversion des Toeplitz-supérieurs de l'Algorithme 2.2.2
($\Diamond$: Bini, $+$: Substitution, $*$: Matlab) pour une matrice de Hankel d'ordre 400.

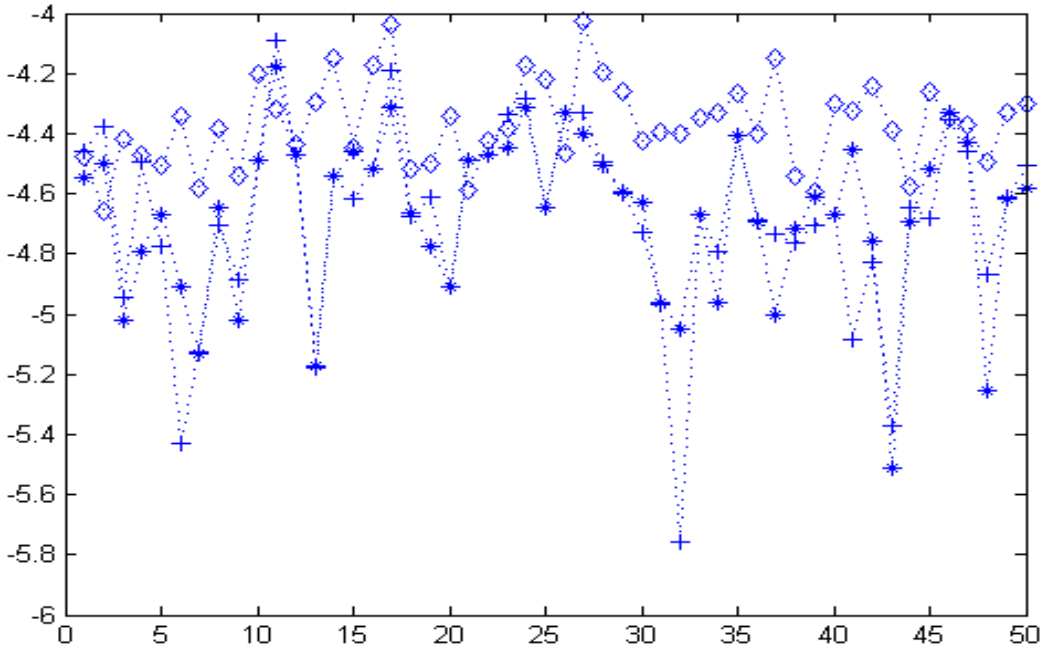


Figure 6.4.1.7 : Efficacité de l'inversion des Toeplitz-supérieurs de l'Algorithme 2.2.2
($\Diamond$: Bini, $+$: Substitution, $*$: Matlab) pour une matrice de Hankel d'ordre 512.

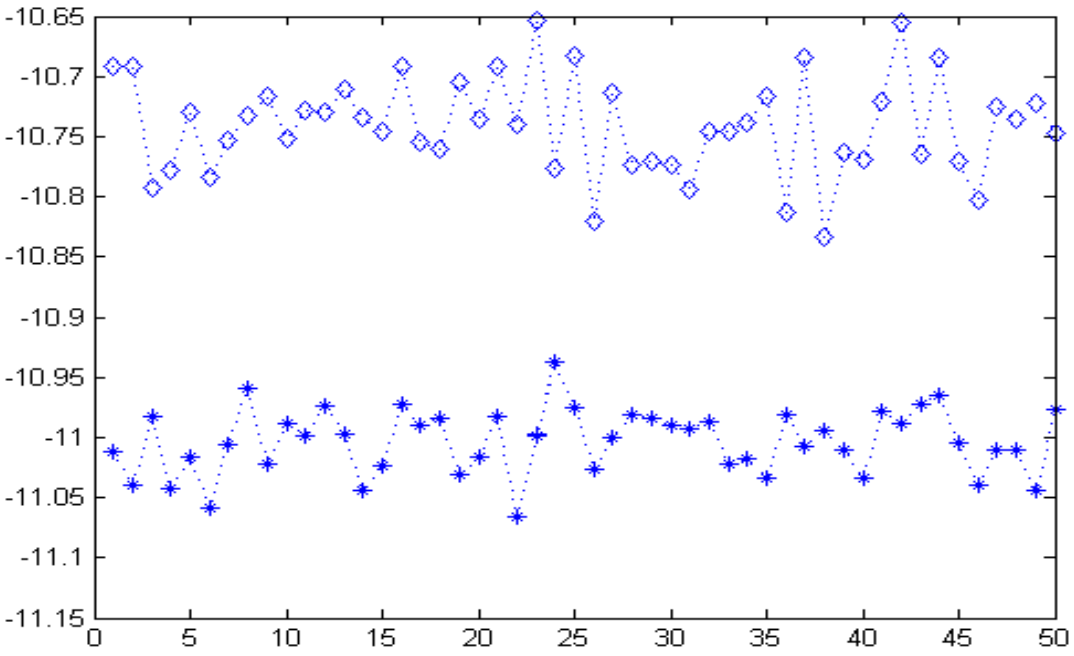


Figure 6.4.1.8 : Efficacité de l'inversion des Toeplitz-supérieurs de l'Algorithme 2.2.2 ($\Diamond$: Bini, $+$: Substitution, $*$: Matlab) pour une matrice de Hankel d'ordre 750.

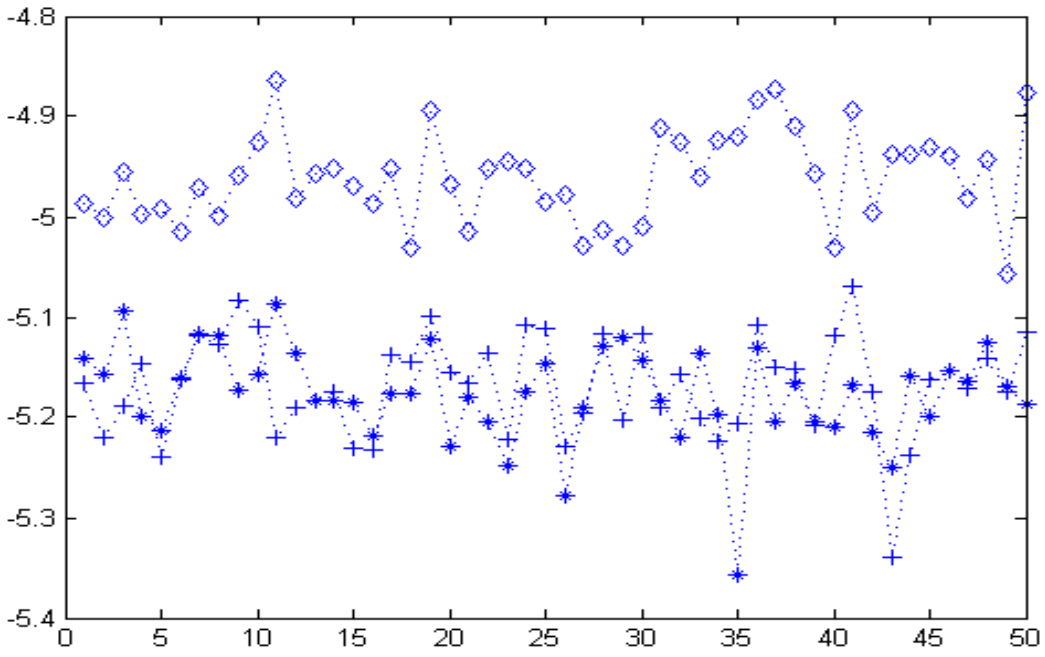


Figure 6.4.1.9 : Efficacité de l'inversion des Toeplitz-supérieurs de l'Algorithme 2.2.2 ($\Diamond$: Bini, $+$: Substitution, $*$: Matlab) pour une matrice de Hankel d'ordre 912.

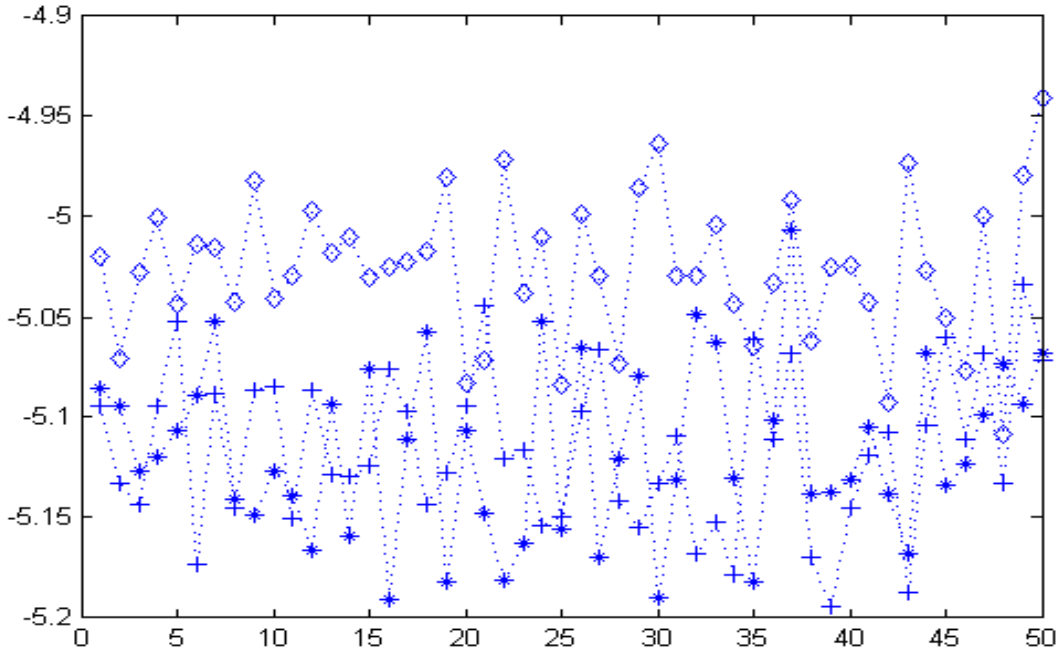


Figure 6.4.1.10 : Efficacité de l'inversion des Toeplitz-supérieurs de l'Algorithme 2.2.2 ($\Diamond$: Bini, $+$: Substitution, $*$: Matlab) pour une matrice de Hankel d'ordre 1024.

Nous remarquons alors que l'utilisation de la méthode d'inversion approchée de la matrice triangulaire de Toeplitz préserve le résultat final et n'a pratiquement pas d'influence sur les étapes intermédiaires.

6.5 Comparaison par rapport à l'approche via Bézout

Nous comparons l'approche fondée sur la diagonalisation par blocs approchée des matrices de Hankel $H(u,v)$ associée à deux polynômes

$$u(x) = \sum_{k=0}^{n} u_k x^k \text{ et } v(x) = \sum_{k=0}^{m} v_k x^k$$

avec $\deg(u(x)) = n$ et $\deg(v(x)) = m$ et $m < n$, avec celui basé sur les matrices le Bézout $JB(u,v)J$ tant pour leur efficacité et leur stabilité numérique.

L'objectif essentiel de cette partie est d'améliorer la stabilité numérique de la diagonalisation par blocs approchée au moyen de la matrice de Bézout parce que les coefficients de la matrice de Bézout sont bornés par $\Phi = 2n\mu\nu$ (avec $\|u\|_\infty \leq \mu$ et $\|v\|_\infty \leq \nu$) et les coefficients de la matrice de Hankel $H(u,v)$ peuvent s'accroître d'une manière exponentielle.

De plus, si u et v sont à coefficients entiers, les calculs dans les étapes intermédiaires peuvent être préservés en nombres entiers. Pour plus de détails, voir [25].

6.5.1 Efficacité

Nous mesurons la différence des matrices diagonales par blocs au moyen de Bézout et Hankel, donnée par (6.2) pour une tolérance fixe $\varepsilon = 10^{-5}$. Nous vérifions cela pour plusieurs matrices de Hankel $H(u, v)$ et Bézout $JB(u, v)J$ associées à deux polynômes u et v perturbés 50 fois.

Nous représentons alors l'efficacité de la diagonalisation par blocs approchée au moyen de la matrice de Hankel $H(u, v)$ et Bézout $JB(u, v)J$ dans les Figures 6.5.1.1-8.

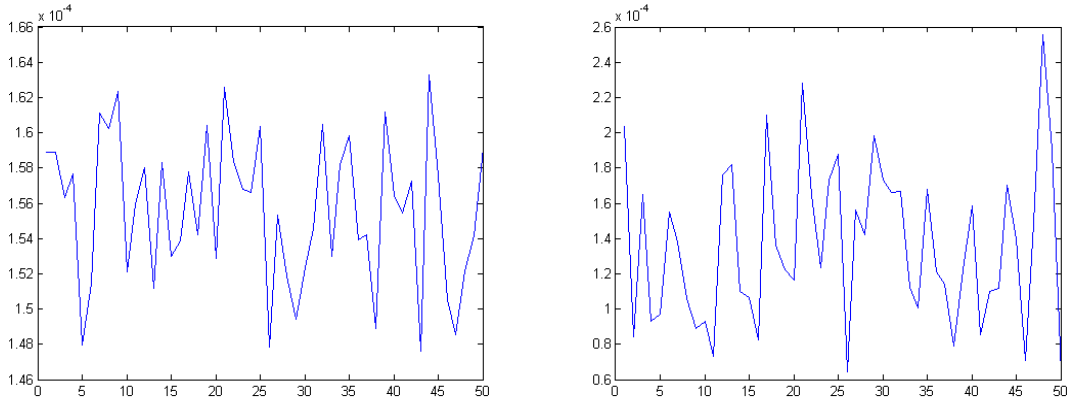


Figure 6.5.1.1 : $Error^{A}_{Hankel}$ (Figure gauche) et $Error^{A}_{Bézout}$ (Figure droite) pour $n = 1024$.

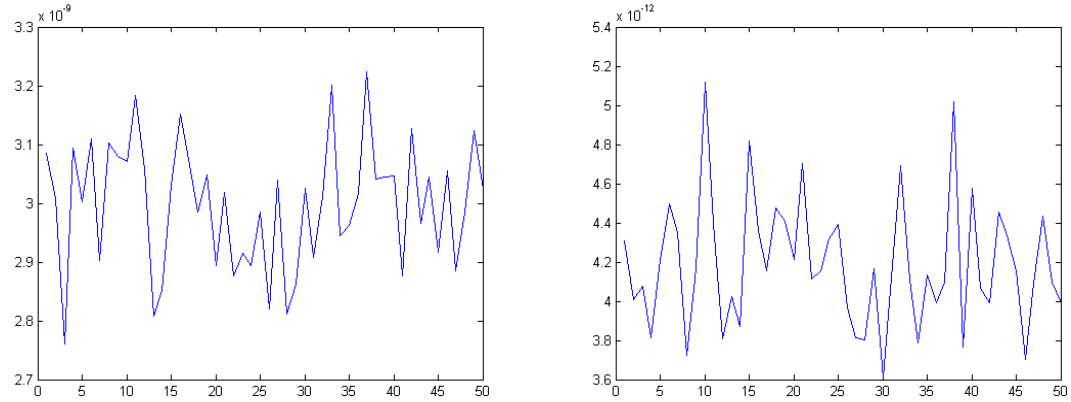


Figure 6.5.1.2 : $Error^{A}_{Hankel}$ (Figure gauche) et $Error^{A}_{Bézout}$ (Figure droite) pour $n = 1224$.

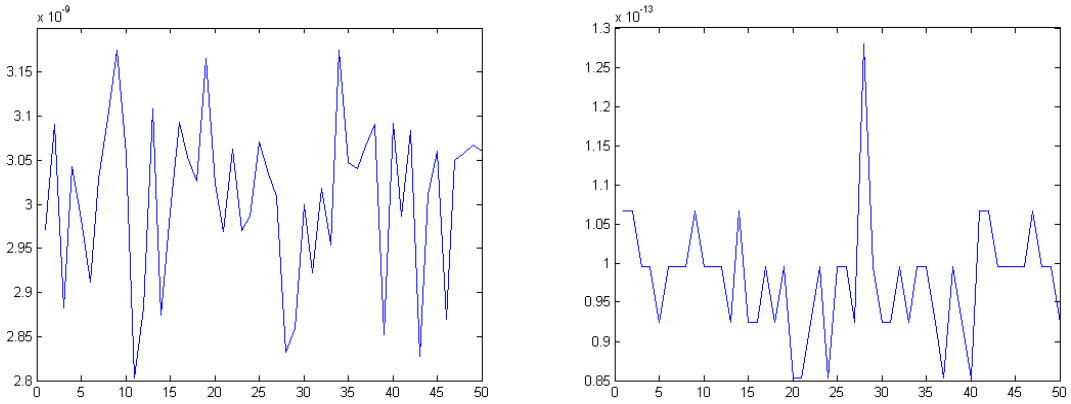


Figure 6.5.1.3 : $Error^{A}_{Hankel}$ (Figure gauche) et $Error^{A}_{Bézout}$ (Figure droite) pour $n = 1424$.

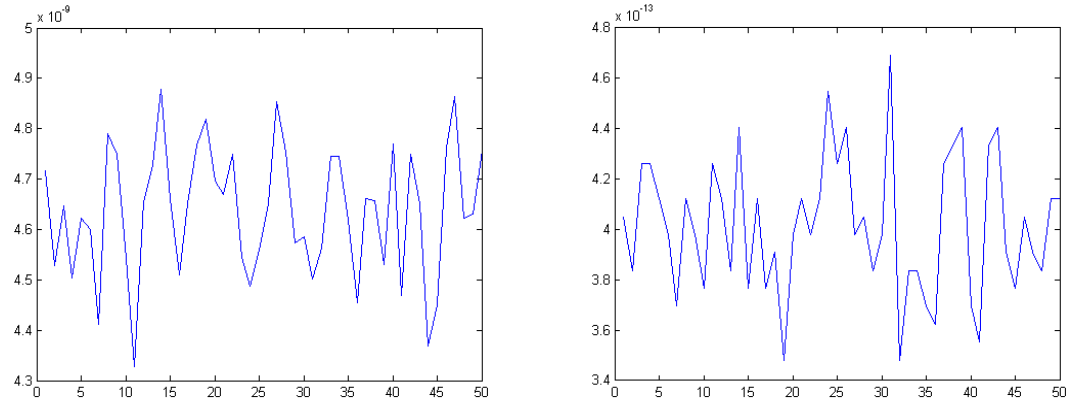


Figure 6.5.1.4 : $Error^{A}_{Hankel}$ (Figure gauche) et $Error^{A}_{Bézout}$ (Figure droite) pour $n = 1624$.

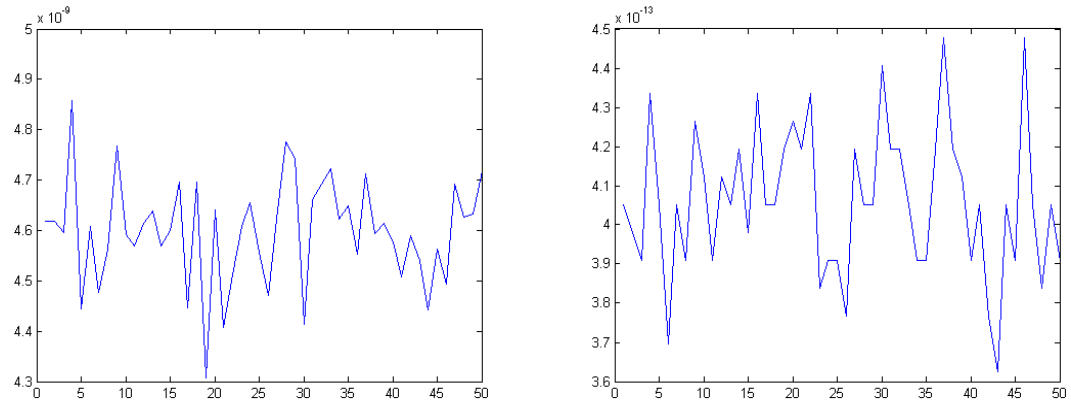


Figure 6.5.1.5 : $Error^{A}_{Hankel}$ (Figure gauche) et $Error^{A}_{Bézout}$ (Figure droite) pour $n = 1824$.

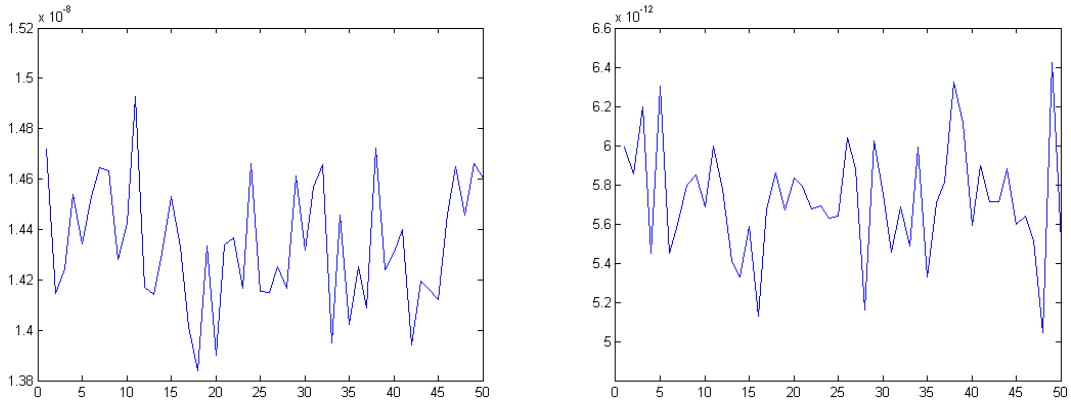


Figure 6.5.1.6 : $Error^{A}_{Hankel}$ (Figure gauche) et $Error^{A}_{Bézout}$ (Figure droite) pour $n = 2048$.

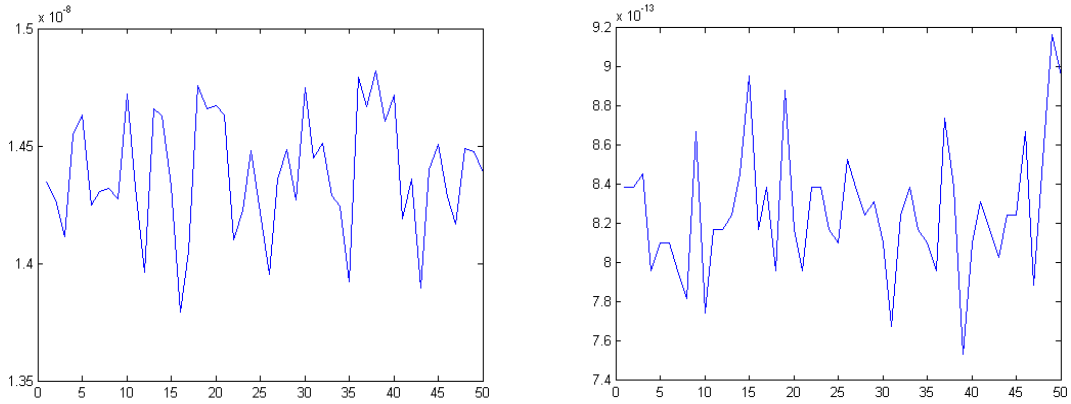


Figure 6.5.1.6 : $Error^{A}_{Hankel}$ (Figure gauche) et $Error^{A}_{Bézout}$ (Figure droite) pour $n = 2448$.

6.5.2 Stabilité

Nous donnons dans le Tableau 6.6.1 l'erreur relative (6.1), en arithmétique flottante à 15 chiffres, appliquée à des matrices de Hankel $H(u, v)$ et Bézout $JB(u, v)J$ associées à deux polynômes u et v de départ exactes.

n	$Error^{R}_{Our\ Hankel}$	$Error^{R}_{Sc\ Bézout}$
1024	5.784074308862707e-012	1.086190389822859e-008
1224	7.377238376056904e-007	1.503623817980172e-007
1424	1.229383772472314e-007	5.902898664239044e-012
1624	1.402838670935590e-006	4.631154488009333e-007
1824	1.552356261630665e-007	8.760591568775624e-008
2048	1.482326667465912e-005	1.042891806603727e-009
2448	3.863794639419495e-008	1.162851904025527e-008

Table 6.2.1.1 : l'erreur relative (6.1), en arithmétique flottante à 15 chiffres, pour des matrices de Hankel.

D'où, l'approche de diagonalisation par blocs au moyen des matrices de Hankel et Bézout sont numériquement stables, voir la méthode fondée sur les matrices de Bézout semble être plus stable quand la taille des matrices n augmente.

Nous représentons dans les Figures 6.2.2.1-7 le comportement de la stabilité pour les coefficients de la matrice de Hankel obtenus au moyen des deux méthodes.

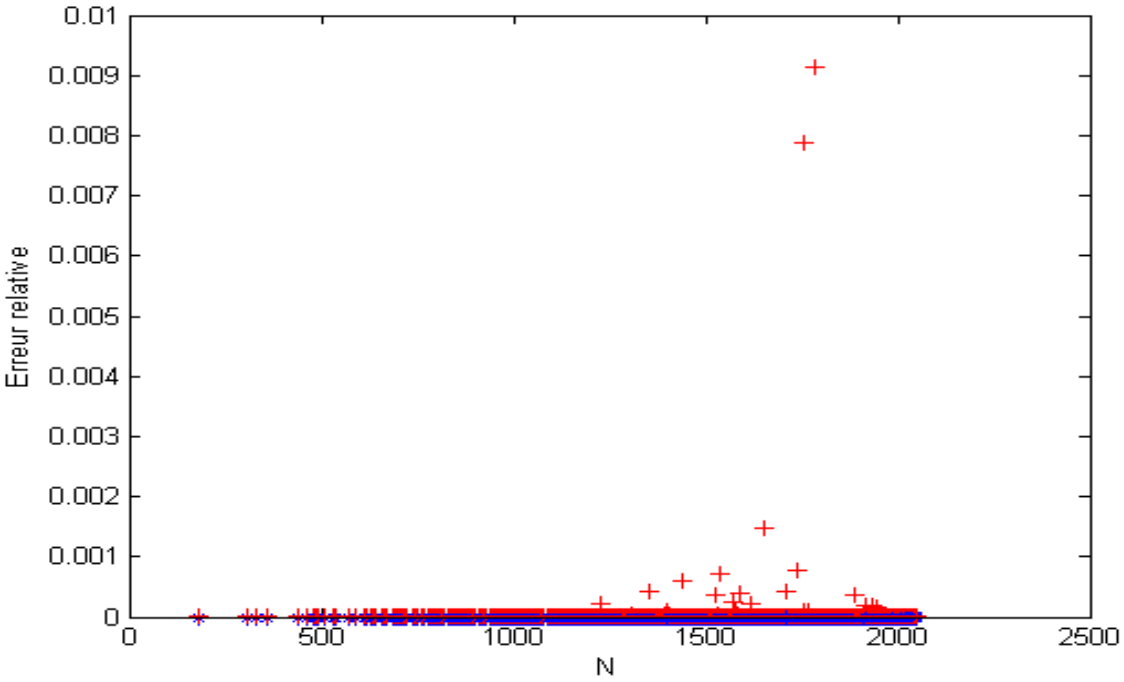


Figure 6.5.2.1 : Erreur relative ($*$: *Hankel*, $+$: *Bézout*) pour $n = 1024$.

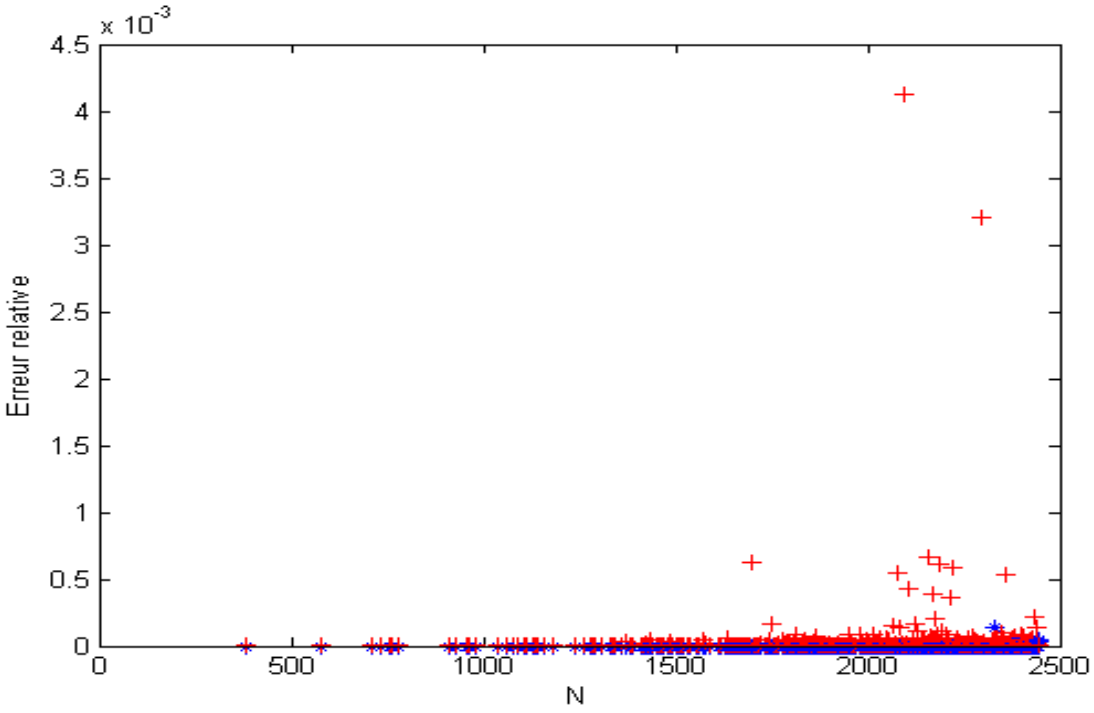


Figure 6.5.2.2 : Erreur relative ($*$: *Hankel*, $+$: *Bézout*) pour $n = 1224$.

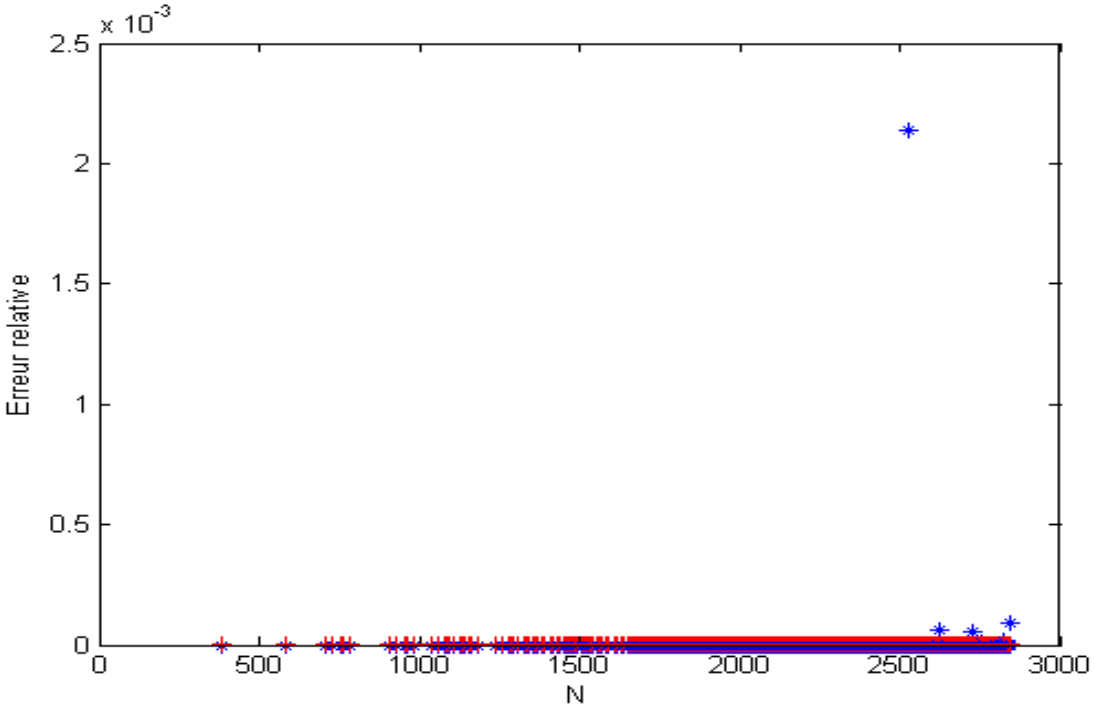


Figure 6.5.2.3 : Erreur relative ($*$: $Hankel$, $+$: $Bézout$) pour $n = 1424$.

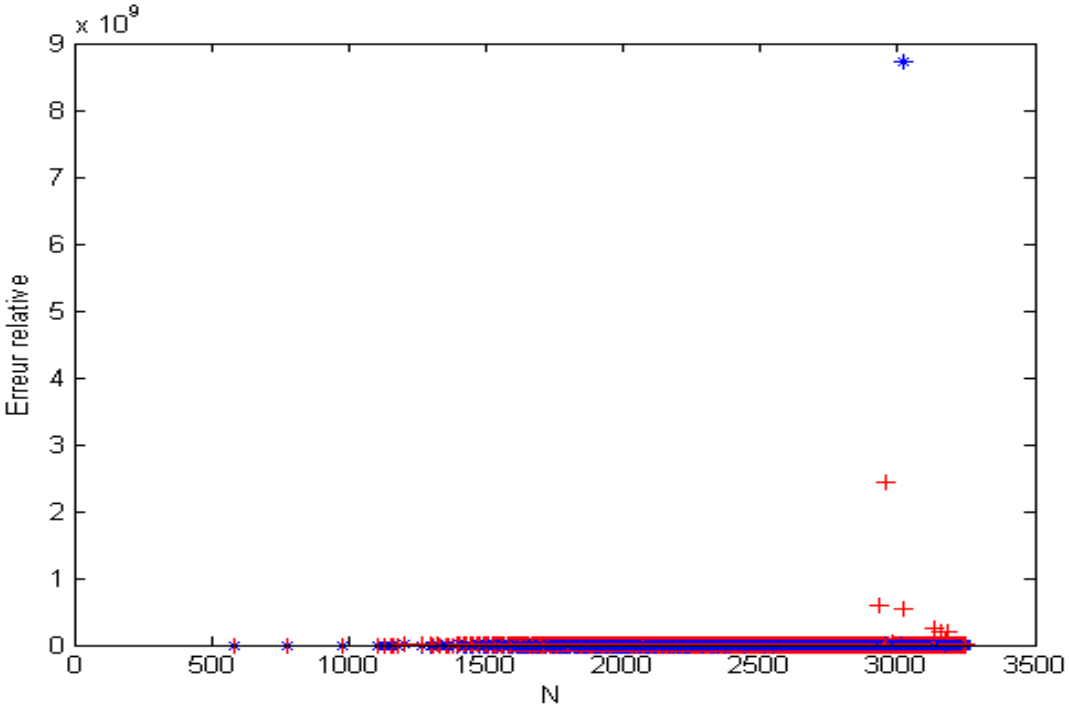


Figure 6.5.2.4 : Erreur relative ($*$: $Hankel$, $+$: $Bézout$) pour $n = 1624$.

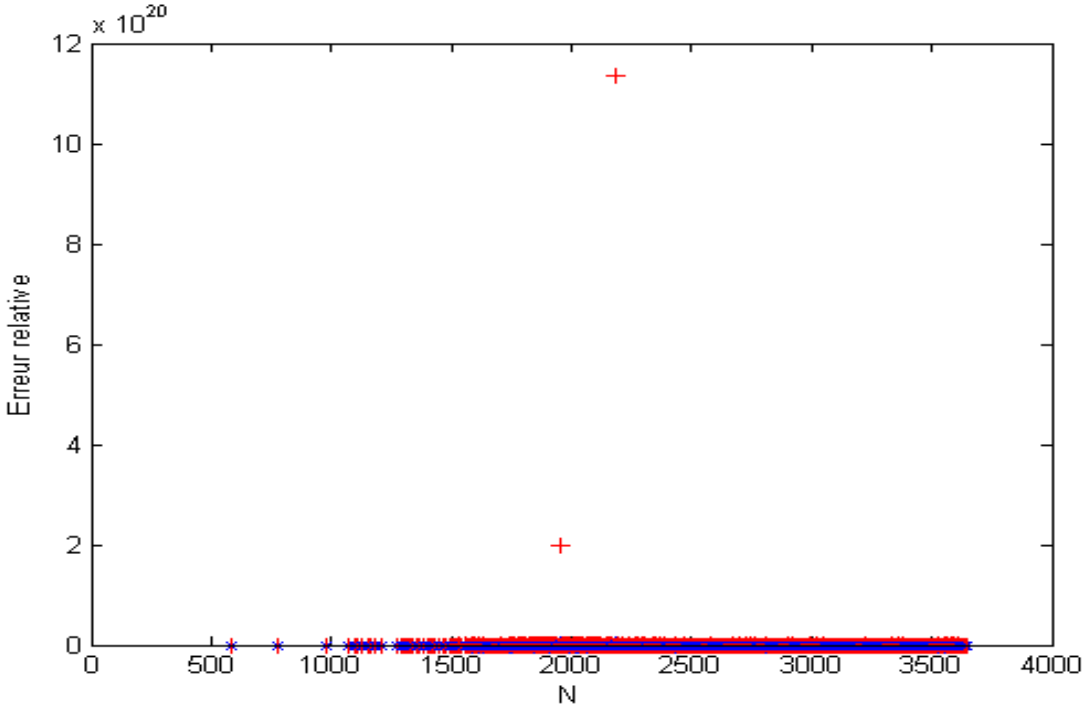


Figure 6.5.2.5 : Erreur relative ($* : Hankel$, $+ : Bézout$) pour $n = 1824$.

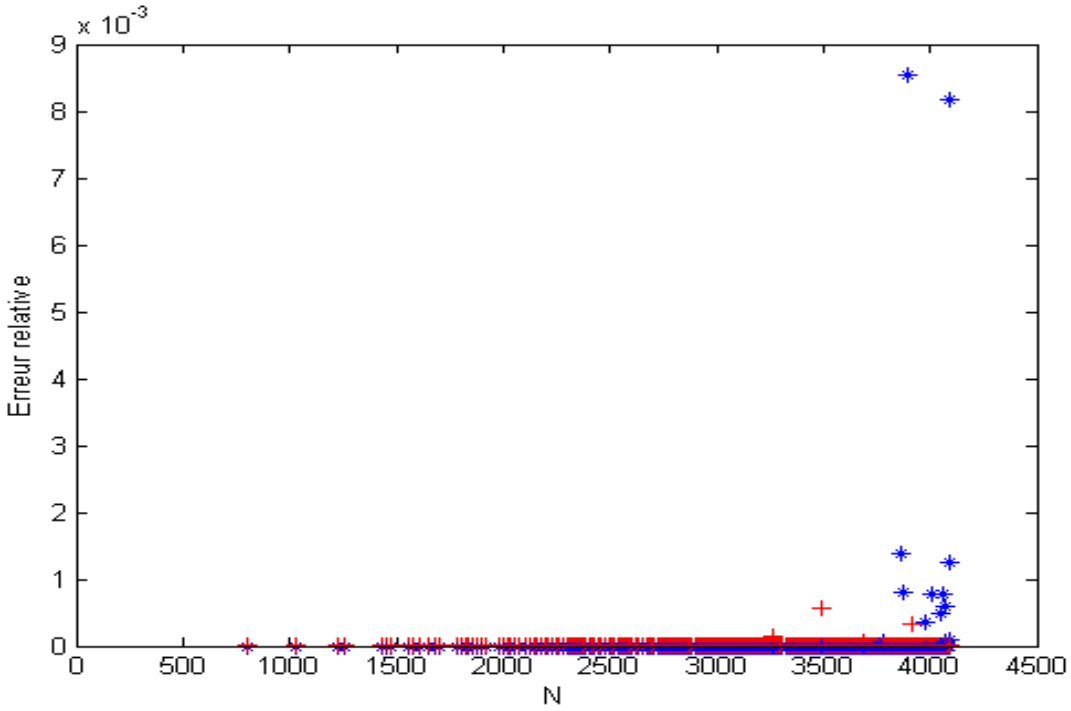


Figure 6.5.2.6 : Erreur relative ($* : Hankel$, $+ : Bézout$) pour $n = 2048$.

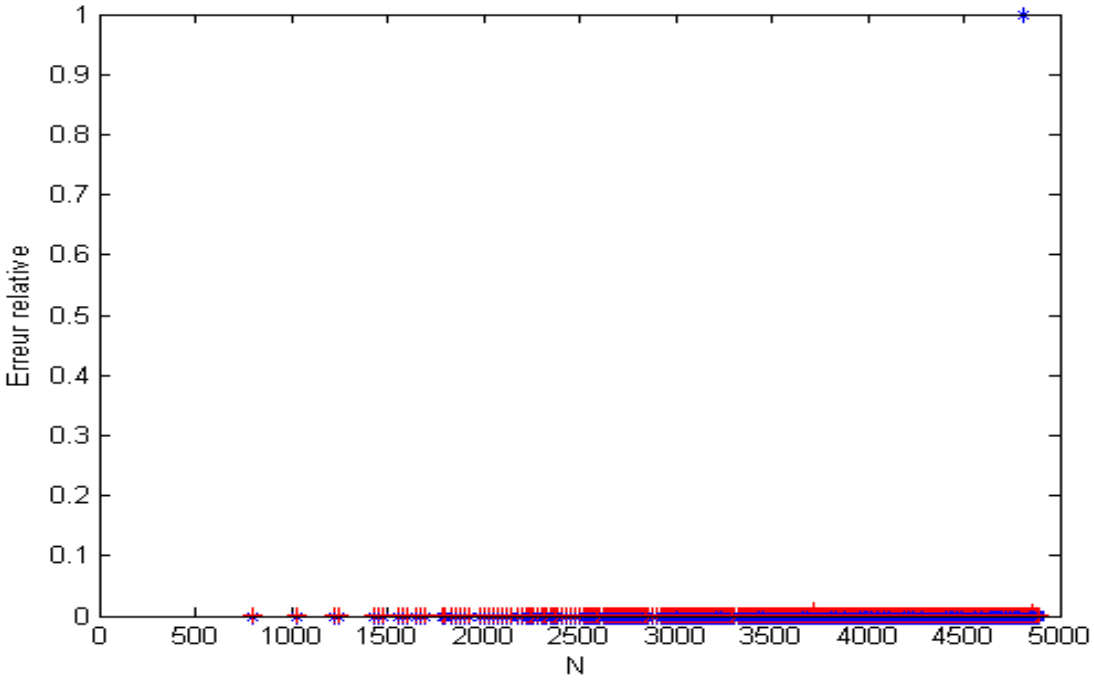


Figure 6.5.2.7 : Erreur relative ($*$: *Hankel*, $+$: *Bézout*) pour $n = 2448$.

6.6 Sensibilité par rapport à la tolérance ϵ

Nous analysons le rôle de ε dans l'exécution des algorithmes. En effet, le teste effectué mesure le nombre des blocs des matrices diagonales par blocs au moyen de la méthode du complément de Schur et notre méthode par rapport à une tolérance ε aléatoire, avec et sans la SVD.

1. $n = 9$, $n_{D_{exact}}^{SVD} = 4$.

ϵ	$\mathbf{n}_{\mathbf{D_{Sc\ approx}}}^{\mathbf{SVD}}$	$\mathbf{n}_{\mathbf{D_{Sc\ approx}}}$	$\mathbf{n}_{\mathbf{D_{our\ approx}}}$	$\mathbf{n}_{\mathbf{D_{our\ approx}}}^{\mathbf{SVD}}$
$,\ldots,\mathbf{10^{-12}}$	**Echec**	**Echec**	**Echec**	**Echec**
$\mathbf{10^{-11}},\ldots,\mathbf{10^{-04}}$	4	4	4	4
$\mathbf{10^{-03}}$	**Echec**	4	4	4
$\mathbf{10^{-02}}$	**Echec**	4	4	**Echec**

Table 6.6.1 : Le nombre des blocs au moyen de notre méthode et de la méthode du complément de Schur pour une matrice de Hankel d'ordre 9.

2. $n = 12$, $n_{D_{exact}}^{SVD} = 5$.

ϵ	$\mathbf{n}_{\mathbf{D_{Sc\ approx}}}^{\mathbf{SVD}}$	$\mathbf{n}_{\mathbf{D_{Sc\ approx}}}$	$\mathbf{n}_{\mathbf{D_{our\ approx}}}$	$\mathbf{n}_{\mathbf{D_{our\ approx}}}^{\mathbf{SVD}}$
$,\ldots,\mathbf{10^{-16}}$	**Echec**	**Echec**	**Echec**	**Echec**
$\mathbf{10^{-15}},\ldots,\mathbf{10^{-12}}$	**Echec**	5	5	**Echec**
$\mathbf{10^{-11}},\ldots,\mathbf{10^{-4}}$	5	5	5	5
$\mathbf{10^{-3}},\mathbf{10^{-2}}$	**Echec**	5	5	**Echec**

Table 6.6.2 : Le nombre des blocs au moyen de notre méthode et de la méthode du complément de Schur pour une matrice de Hankel d'ordre 12.

3. $n = 25$, $n_{D_{exact}}^{SVD} = 6$.

ϵ	$\mathbf{n}_{\mathbf{D_{Sc\ approx}}}^{\mathbf{SVD}}$	$\mathbf{n_{D_{Sc\ approx}}}$	$\mathbf{n_{D_{our\ approx}}}$	$\mathbf{n}_{\mathbf{D_{our\ approx}}}^{\mathbf{SVD}}$
$,\ldots,\mathbf{10^{-11}}$	**Echec**	**Echec**	**Echec**	**Echec**
$\mathbf{10^{-10}},\ldots,\mathbf{10^{-4}}$	6	6	6	6
$\mathbf{10^{-3}},\mathbf{10^{-2}}$	**Echec**	6	6	**Echec**

Table 6.6.3 : Le nombre des blocs au moyen de notre méthode et de la méthode du complément de Schur pour une matrice de Hankel d'ordre 25.

4. $n = 54$, $n_{D_{exact}}^{SVD} = 6$.

ϵ	$\mathbf{n}_{\mathbf{D_{Sc\ approx}}}^{\mathbf{SVD}}$	$\mathbf{n_{D_{Sc\ approx}}}$	$\mathbf{n_{D_{our\ approx}}}$	$\mathbf{n}_{\mathbf{D_{our\ approx}}}^{\mathbf{SVD}}$
$,\ldots,\mathbf{10^{-11}}$	**Echec**	**Echec**	**Echec**	**Echec**
$\mathbf{10^{-10}},\ldots,\mathbf{10^{-5}}$	6	6	6	6
$\mathbf{10^{-4}},\ldots,\mathbf{10^{-2}}$	**Echec**	6	6	**Echec**

Table 6.6.4 : Le nombre des blocs au moyen de notre méthode et de la méthode du complément de Schur pour une matrice de Hankel d'ordre 54.

5. $n = 56$, $n_{D_{exact}}^{SVD} = 4$.

ϵ	$\mathbf{n}_{\mathbf{D_{Sc\ approx}}}^{\mathbf{SVD}}$	$\mathbf{n_{D_{Sc\ approx}}}$	$\mathbf{n_{D_{our\ approx}}}$	$\mathbf{n}_{\mathbf{D_{our\ approx}}}^{\mathbf{SVD}}$
$,\ldots,\mathbf{10^{-11}}$	**Echec**	**Echec**	**Echec**	**Echec**
$\mathbf{10^{-10}},\ldots,\mathbf{10^{-8}}$	4	4	4	4
$\mathbf{10^{-7}},\ldots,\mathbf{10^{-2}}$	**Echec**	4	4	**Echec**

Table 6.6.5 : Le nombre des blocs au moyen de notre méthode et de la méthode du complément de Schur pour une matrice de Hankel d'ordre 56.

6. $n = 75$, $n_{D_{exact}}^{SVD} = 6$.

ϵ	$\mathbf{n}_{\mathbf{D_{Sc\ approx}}}^{\mathbf{SVD}}$	$\mathbf{n_{D_{Sc\ approx}}}$	$\mathbf{n_{D_{our\ approx}}}$	$\mathbf{n}_{\mathbf{D_{our\ approx}}}^{\mathbf{SVD}}$
$,\ldots,\mathbf{10^{-11}}$	**Echec**	**Echec**	**Echec**	**Echec**
$\mathbf{10^{-10}},\ldots,\mathbf{10^{-6}}$	6	6	6	6
$\mathbf{10^{-5}},\ldots,\mathbf{10^{-2}}$	**Echec**	6	6	**Echec**

Table 6.6.6 : Le nombre des blocs au moyen de notre méthode et de la méthode du complément de Schur pour une matrice de Hankel d'ordre 75.

7. $n = 100$, $n_{D_{exact}}^{SVD} = 6$.

ϵ	$\mathbf{n}_{\mathbf{D_{Sc\ approx}}}^{\mathbf{SVD}}$	$\mathbf{n_{D_{Sc\ approx}}}$	$\mathbf{n_{D_{our\ approx}}}$	$\mathbf{n}_{\mathbf{D_{our\ approx}}}^{\mathbf{SVD}}$
$,\ldots,\mathbf{10^{-11}}$	**Echec**	**Echec**	**Echec**	**Echec**
$\mathbf{10^{-10}},\ldots,\mathbf{10^{-5}}$	6	6	6	6
$\mathbf{10^{-4}},\ldots,\mathbf{10^{-2}}$	**Echec**	6	6	**Echec**

Table 6.6.7 : Le nombre des blocs au moyen de notre méthode et de la méthode du complément de Schur pour une matrice de Hankel d'ordre 100.

A partir de ces expérimentations, l'utilisation de la SVD ne garantit pas la taille des blocs initiale. Ainsi, la technique utilisée pour la recherche des blocs paraît plus efficace.

Chapitre 7

Généralisations et perspectives

La thèse est déstinée à concevoir des algorithmes rapides en rapport direct avec les matrices structurées, plus particulièrement les matrices de Toeplitz et Hankel.

D'une part, plusieurs algorithmes sont sérieusement améliorés dans cette thèse. Spécifiquement, la thèse comprend :

1. L'analyse des différents travaux théoriques et empiriques portant sur le calcul de l'inverse d'une matrice triangulaire de Toeplitz.
2. Un algorithme rapide de diagonalisation par blocs approchée d'une matrice de Hankel à coefficients réels via la matrice triangulaire supérieure de Toeplitz
3. Un algorithme rapide de diagonalisation par blocs approchée d'une matrice de Hankel à coefficients réels via le complément de Schur.
4. Un algorithme rapide, basé sur la diagonalisation par blocs approchée d'une matrice de Hankel via la matrice triangulaire supérieure de Toeplitz, qui calcule la suite des quotients et la suite des restes apparues dans l'algorithme d'Euclide approché.
5. Un algorithme rapide, basé sur la diagonalisation par blocs approchée d'une matrice de Bézout via le complément de Schur, qui calcule la suite des quotients et la suite des restes apparus dans l'algorithme d'Euclide approché.
6. Une variante de l'algorithme rapide de diagonalisation par blocs approchée d'une matrice de Hankel via la matrice triangulaire supérieure de Toeplitz dans le cas complexe.
7. Une variante de l'algorithme rapide de diagonalisation par blocs approchée d'une matrice de Hankel via le complément de Schur dans le cas complexe.

D'autre part, l'un des buts de cette thèse estt de certifier théoriquement les algorithmes présentés. On n'a pas pu aboutir à notre but, parce que ce problème s'est révélé énormement plus difficile qu'on pensait à priori. Ceci est remplacé par une étude expérimentale conséquente qui légitime nos heuristiques dans une certaine mesure.

Ainsi, les perspectives de cette thèse :

– Une étude rigoureuse de la propagation des erreurs : En arithmétique exacte, la propagation des erreurs de troncature introduites à chaque étape de la récursivité,

et, si l'arithmétique à virgule flottante est utilisée, une analyse des erreurs d'arrondi.

- Il serait agréable d'avoir une idée sur la façon dont ces erreurs se propagent dans les étapes intermédiaires au moyen du complément de Schur.
- Comment mesurer théoriquement la qualité des algorithmes de factorisation par rapport à ces deux objectifs :

  1. Avoir des matrices de passage qui n'aient pas des coefficients trop grands.
  2. Donner de bonnes approximations de matrices par blocs Hankel-inférieures.

- Chercher une diagonalisation par blocs approchée de la matrice de Hankel en se basant sur des divisions polynomiales approchées (ce qui évitera d'inverser la matrice triangulaire de Toeplitz).
- Concevoir une version de cette diagonalisation par blocs approchée pour une matrice de Hankel par blocs et de Toeplitz par blocs.
- Il serait bien de trouver des applications où ces décompositions approchées peuvent être utiles.

Annexe A : Notions fondamentales

Dans cette section, nous allons faire les rappels nécessaires qui seront très utiles pour cette thèse. Cependant les bases de l'algèbre linéaire seront supposées acquises. Pour plus de détails, se reporter, par exemple, à [2, 25, 1, 26].

A.1 Conditionnement et stabilité

A.1.1 Conditionnement d'un problème

Soit f une fonction de $E_{\mathcal{D}}$ dans $E_{\mathcal{S}}$ où $E_{\mathcal{D}}$ et $E_{\mathcal{S}}$ sont, respectivement, deux espaces vectoriels des données et des solutions sur le corps des nombres réels ou complexes. Considérons le problème du calcul suivant

$$y = f(x)$$

tel que x représente les donnés. *Un problème numérique* sera donc défini comme étant une fonction f (implicite ou explicite) de connexion entre un élément x d'entrée ($x \in E_{\mathcal{D}}$) et un élément x de sortie ($f(x) \in E_{\mathcal{S}}$). Un élément dans $E_{\mathcal{S}}$ ou $E_{\mathcal{D}}$ peut être un nombre, un point de l'espace euclidien, une matrice, ou n'importe quelle entité mathématique plus compliquée.

En pratique, les données sont très souvent tirées de mesures physiques ou d'autres calculs. Elles sont généralement d'une précision limitée. Une question importante qui se pose alors est de savoir dans quelle mesure les résultats du calcul seront sensibles à de légères modifications des données. Si une petite perturbation des données induit un changement important des résultats, il est douteux que les résultats obtenus puissent être fiables. On parle alors de *problème mal-conditionné.*

Le conditionnement mesure la dépendance de la solution d'un problème numérique par rapport aux données du problème, ceci afin de contrôler la validité d'une solution calculée par rapport à ces données. Il s'agit le plus souvent d'une quantité numérique, parfois appelée *nombre de conditionnement.*

De façon plus générale, on peut dire le nombre de conditionnement associé à un problème est une mesure de la difficulté de calcul numérique du problème. Un problème possédant un nombre de conditionnement bas est dit *bien-conditionné* et un problème possédant un nombre de conditionnement élevé est dit *mal-conditionné.*

A.1.2 Stabilité numérique

Le conditionnement d'un problème, ne faisant aucune mention de l'algorithme de résolution utilisé, est donc une propriété purement mathématique du problème, qui n'a rien à voir avec l'arithmétique finie de l'ordinateur, ni avec l'algorithme de résolution utilisé. Il est indépendant de tout algorithme choisi pour le résoudre, et s'évalue en arithmétique exacte. Il existe dans le problème avant même que celui-ci soit résolu numériquement. C'est pourquoi d'ailleurs on l'appelle parfois *la stabilité mathématique* du problème.

La stabilité numérique est une propriété des algorithmes numériques. La définition précise de la stabilité dépend du contexte, mais elle concerne l'exactitude des résultats fournis par un algorithme. Elle décrit comment les erreurs dans les données d'entrée sont propagées à travers un algorithme.

Parfois un calcul peut être réalisé de plusieurs manières, lesquelles sont toutes algébriquement équivalentes et donnent théoriquement le même résultat, mais dans la pratique elles donnent des résultats différents car elles ont différents niveaux de stabilité numérique. Une des tâches communes de l'analyse numérique est d'essayer de trouver les algorithmes les plus robustes, c'est-à-dire ayant la meilleure stabilité numérique.

Dans une méthode stable, les erreurs restent minimes et les résultats produits sont conformes à ceux attendus. Dans une méthode instable, les erreurs de calcul sont amplifiées par le traitement et altèrent le résultat final. Des méthodes instables produisent rapidement des résultats aberrants et sont inutiles pour le traitement numérique.

A.2 Normes et distances

A.2.1 Normes vectorielles

Soit E un espace vectoriel sur $\mathbb{C}$. On appelle *norme* toute application de E dans $\mathbb{R}$, qui à tout $x \in E$, associe le nombre réel $\|x\|$ (appelé norme de x) qui vérifie les trois conditions suivantes

1. $\|x\| \geq 0$ et $\|x\| = 0$ si et seulement si $x = 0 \in E$,
2. $\|\lambda x\| = |\lambda| \, . \, \|x\|$, $\forall \lambda \in \mathbb{C}$,
3. $\|x + y\| \leq \|x\| + \|y\|$, $\forall x, y \in E$.

Comme c'est le cas pour la valeur absolue, une norme sert à mesurer la distance $\|x - y\|$ entre deux éléments. Sur un espace vectoriel, il se peut que l'on puisse définir plusieurs normes. Cependant, si l'espace est de dimension finie, toutes les normes sont équivalentes c'est-à-dire qu'il existera des inégalités entre elles. Par conséquent si deux éléments sont voisins au sens d'une certaine norme (et en particulier si une suite converge pour une certaine norme), il en sera de même pour une autre norme.

Prenons maintenant le cas $E = \mathbb{C}^n$. Soit $x = (x_1, \ldots, x_n)^T$ un vecteur. Les quantités suivantes s'appellent *normes de Hölder* d'indice k

$$\|x\|_k = \left(|x_1|^k + \cdots + |x_n|^k\right)^{1/k}, \quad k = 1, 2, \ldots,$$
$$\|x\|_\infty = \max_{1 \leq i \leq n} |x_i| .$$

Parmi ces normes, les plus utilisées sont $\|x\|_1$, $\|x\|_\infty$ et $\|x\|_2$. On les désigne souvent respectivement sous les termes de norme l_1, l_∞ et l_2. Cette dernière représente la longueur du vecteur x au sens euclidien du terme ; elle s'appelle *norme euclidienne*.

A.2.2 Normes matricielles

Une norme de matrice peut être définie à partir d'une norme pour les vecteurs (mais cela n'est pas obligatoire). Soit $A \in \mathbb{C}^{n \times m}$ une matrice. La quantité

$$\|A\| = \sup_{x \neq 0} \frac{\|Ax\|}{\|x\|}$$

est une norme pour la matrice A. Puisque l'on peut, dans cette définition, remplacer x par αx, où α est un scalaire, on peut toujours choisir x de norme 1 et l'on a donc également

$$\|A\| = \sup_{\|x\|=1} \|Ax\| .$$

Ces normes de matrice vérifient les trois propriétés des normes mais, en plus, il existe deux autres propriétés qui nous serons très utiles

$$\|Ax\| \leq \|A\| . \|x\|$$

et

$$\|AB\| \leq \|A\| . \|B\| .$$

On appelle *multiplicative* toute norme vérifiant cette dernière inégalité.

Les normes de matrices les plus utilisées sont celles qui sont reliées à une norme de Hölder pour les vecteurs, c'est-à-dire

$$\|A\|_k = \sup_{x \neq 0} \frac{\|Ax\|_k}{\|x\|_k}$$

où $\|.\|_k$ est la norme de vecteur de Hölder d'indice k.

Les normes de matrices semblent être difficiles à calculer en pratique puisqu'elles font intervenir une borne supérieure. Cependant, on connaît leurs expressions exactes dans trois cas pour les normes de Hölder

$$\|A\|_1 = \max_{1 \leq j \leq m} \sum_{i=1}^{n} |a_{ij}| ,$$
$$\|A\|_\infty = \max_{1 \leq j \leq n} \sum_{i=1}^{m} |a_{ij}| ,$$
$$\|A\|_2 = \sqrt{\rho\left(A^T A\right)},$$

où $\rho\left(A^TA\right)$ désigne le rayon spectral de la matrice $A^TA \in \mathbb{C}^{m\times m}$, c'est-à-dire sa plus grande valeur propre puisque, A^TA étant symétrique définie positive, celles-ci sont réelles et positives. cette dernière norme s'appelle aussi *norme spectrale.*

Si les matrices considérées sont carrées ($m = n$).

On démontre que, quelle que soit la norme,

$$\rho(A) \leq \|A\|.$$

Si A est symétrique, $\|A\|_2 = \rho(A)$.

On utilise aussi parfois la norme de *Frobenius* car elle est facile à calculer. Elle est définie par

$$\|A\|_F = \left(\sum_{i,j=1}^{m} |a_{ij}|^2\right)^{1/2}.$$

Pour cette norme, on a également

$$\|AB\|_F \leq \|A\|_F . \|B\|_F .$$

De plus

$$\|A\|_F^2 = tr\left({}^tAA\right),$$

où tr désigne la trace d'une matrice, c'est-à-dire la somme des éléments de sa diagonale.

A.3 Conditionnement d'une matrice

Soit $A \in \mathbb{C}^{n\times n}$. Pour toute norme de Hölder, on a

$$\left\|AA^{-1}\right\| = \|I\| \leq \|A\| . \left\|A^{-1}\right\|.$$

Or, la norme de la matrice identité est égale à 1 d'après la définition. Il s'en suit que l'on a l'inégalité (l'indice k est sous-entendu)

$$k(A) = \|A\| . \left\|A^{-1}\right\| \geq 1.$$

Ce nombre $k(A)$ s'appelle de *conditionnement* de A.

On a les propriétés suivantes

1. $k(\lambda A) = k(A)$, $\forall \lambda \neq 0$,
2. $k(A^{-1}) = k(A)$ puisque A et A^{-1} jouent les rôles symétriques dans la définition de $k(A)$,
3. $0 < \frac{1}{\|A^{-1}\|} \leq \frac{\|Ax\|}{\|x\|} \leq \|A\|$, $\forall x$.

La notion de conditionnement d'une matrice est absolument fondamentale pour les méthodes de résolution des systèmes d'équations linéaires. On dit que la matrice A est *bien conditionnée* si $k(A)$ est "voisin" de 1. Si $k(A)$ est "grand" par rapport à 1, on dit que A est *mal conditionnée*. Naturellement les adjectifs "voisin" et "grand" sont subjectifs. Si la précision de l'ordinateur avec laquelle on travaille est de 10^{-7} un conditionnement de 10^5 sera considéré comme "grand". Par contre, si la précision est de 10^{-16}, un tel conditionnement sera "petit".

Remarque A.3 On fait souvent l'erreur de croire qu'une matrice dont le déterminant est très voisin de zéro est mal-conditionnée et cela parce qu'elle est proche d'une matrice singulière. Il n'en est rien. En effet, considérons la matrice diagonale de dimension n dont tous les termes sont égaux à un nombre ϵ. son déterminant vaut ϵ^n alors que son conditionnement est égal à 1.

A.4 SVD et rang approché

En mathématiques, le procédé d'algèbre linéaire de décomposition en valeurs singulières (ou SVD, de l'anglais : Singular Value Decomposition) d'une matrice est un outil important de factorisation des matrices rectangulaires réelles ou complexes. Ses applications s'étendent du traitement du signal aux statistiques, en passant par la météorologie.

Cette technique est l'outil principal de l'analyse numérique pour déterminer le rang d'une matrice en présence de perturbations (rang approché).

La référence classique de la SVD est [45]. Nous rappelons la définition et quelques propriétés de base, qui se révéle utile.

Théorème A.4.1 Soit $A \in \mathbb{C}^{m\times n}$. Alors, il existe deux matrices unitaires $U \in \mathbb{C}^{m\times m}$ et $V \in \mathbb{C}^{n\times n}$ et une matrice diagonale $D = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_p) \in \mathbb{C}^{m\times n}$, avec $p = \min\{m, n\}$, telles que

$$A = UD^H V, \tag{7.1}$$

où $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_p \geq 0$.

La factorisation de (7.1) est appelé la *décomposition en valeurs singulières* de A et les σ_j sont les *valeurs singulières*. Les colonnes $u_1, \ldots, u_m$ de U sont les vecteurs singuliers à gauche de A, alors que les colonnes $v_1, \ldots, v_m$ de V sont les vecteurs singuliers à droite de A.

La SVD permet de donner plusieurs informations sur une matrice A. Si $A = UDV^H$ et r est un paramètre tels que

$$\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_{r+1} = \cdots = 0,$$

alors

$$\begin{aligned} \text{rang}(A) &= r, \\ \ker(A) &= < v_{r+1}, \ldots, v_n >, \\ \text{Im}(A) &= < u_1, \ldots, u_r > . \end{aligned}$$

De plus, A s'écrit

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

Théorème A.4.2 Soit $A \in \mathbb{C}^{m\times n}$. S'il existe deux matrices unitaires $U \in \mathbb{C}^{m\times m}$ et $V \in \mathbb{C}^{n\times n}$ et une matrice diagonale $D = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_p) \in \mathbb{C}^{m\times n}$, avec $p = \min\{m, n\}$, telles que $A = UDV^H$. Poser $k < r = \text{rang}(A)$ et

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

alors

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

Autrement dit, la distance entre A et la matrice "approchée" de A de rang k est σ_{k+1}.

Ceci est un résultat important pour le calcule du rang approché ou numérique d'une matrice, ce qui est fondamentale dans le calcul de la suite des restes et des quotients ainsi que le PGCD. Ainsi, le *rang approché* de A est r tels que les dernières $(n-r)$ valeurs singulières de A sont plus petits que $\mu \|A\|_2$, où μ est la précision machine.

Un raisonnement similaire appliqué à la matrice A perturbée (rajouter ϵ à toutes ces composantes). Dans ce cas, il est logique de calculer le ϵ-*rang* de A : nous disons que $rang(A, \epsilon)$ si

$$\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_{r_\epsilon} > \epsilon \geq \sigma_{r_\epsilon+1} \geq \ldots \geq \sigma_p.$$

A.3.1 SVD des matrices de Bézout et de Hankel

Soit $B(u, v) \in \mathbb{C}^{n\times n}$ une matrice de Bézout associée à deux polynômes $u(x)$ et $v(x)$ et $\text{rang}(B) = k$, alors $n - k = \deg(\text{PGCD}(u, v))$ .Cette propriété est valable pour une matrice de Hankel $H(u, v) \in \mathbb{C}^{n\times n}$, associée à deux polynômes $u(x)$ et $v(x)$.

Il est claire que la notion du rang approché est importante losque nous migrons ce résultat dans le cas "approximatif". En effet, le degré d'un ϵ-PGCD de $B(u, v)$ ou $H(u, v)$ n'est pas trivial. Toutefois, nous introduisons deux lemmes de calcule des bornes d'erreurs. Nous supposons $\|u(x)\|_2 = \|v(x)\|_2 = 1$.

Lemme A.4.1 Soient $u(x)$ et $v(x)$ deux polynômes de degrés respectivement n et m, avec $n \geq m$. Supposons que $\|u(x)\|_2 = \|v(x)\|_2 = 1$. Soit $\epsilon > 0$, $\widehat{u}(x)$ et $\widehat{v}(x)$ deux polynômes de degrés respectivement n et m vérifiant $\|u(x) - \widehat{u}(x)\|_2 \leq \epsilon$, $\|v(x) - \widehat{v}(x)\|_2 \leq \epsilon$ alors, pour une approximation de premier ordre :

$$\|B(u, v) - B(\widehat{u}, \widehat{v})\|_2 \leq 4n\epsilon.$$

Preuve. Soient $a(x) = \widehat{u}(x) - u(x)$ et $b(x) = \widehat{v}(x) - v(x)$, où $\|a(x)\|_2 \leq \epsilon$ et $\|b(x)\|_2 \leq \epsilon$. D'après la linéarité de la matrice de Bézout, nous avons

$$B(\widehat{u}, \widehat{v}) = B(u, v) + B(u, b) + B(a, v) + B(a, b).$$

Ainsi, pour une approximation de premier ordre, nous obtenons

$$B(u,v) - B(\widehat{u},\widehat{v}) \triangleq B(u,b) + B(a,v).$$

De plus

$$\|B(u,v) - B(\widehat{u},\widehat{v})\|_2 \leq \|B(u,b)\|_2 + \|B(a,v)\|_2.$$

Or,

$$\begin{aligned}\|B(u,b)\|_2 &\leq 2n \|u(x)\|_2 \|b(x)\|_2 \leq 2n\epsilon \\ \|B(a,v)\|_2 &\leq 2n \|a(x)\|_2 \|v(x)\|_2 \leq 2n\epsilon.\end{aligned}$$

d'où le résultat. ■

Lemme A.4.2 Supposons que ϵ-PGCD de degré k. Soit $\sigma_1 \geq \ldots \geq \sigma_N$ sont les valeurs singulières de $B(u,v)$. Alors,

$$\sigma_{n-k+1} \leq 4n\epsilon + \mathcal{O}\left(\epsilon^2\right).$$

Preuve. Soient $\widehat{u}(x)$ et $\widehat{v}(x)$ deux polynômes de même degrés vérifiant $\|u(x) - \widehat{u}(x)\|_2 \leq \epsilon$, $\|v(x) - \widehat{v}(x)\|_2 \leq \epsilon$, et $k = \text{PGCD}(\widehat{u},\widehat{v})$. De plus, la matrice $B(\widehat{u},\widehat{v})$ est de rang $n-k$. Ainsi, à partir du Théorème A.3.2, nous avons $\sigma_{n-k+1} \leq \|B(\widehat{u},\widehat{v})\|_2$. Lemme A.3.1 nous montre $\|B(\widehat{u},\widehat{v})\|_2 \leq 4n\epsilon$. ■

Remarque A.4.1 En utilisant la Proposition 3.2.3 du Chapitre 3, il est naturel d'appliquer tous les résultats de cette section à une matrice de Hankel $H(u,v) \in \mathbb{C}^{n\times n}$, associée à deux polynômes $u(x)$ et $v(x)$ avec $n \geq m$.

A.5 FFT

La transformée de Fourier rapide, FFT (Fast Fourier Transform), est l'un des algorithmes dont la publication a provoqué une véritable révolution dans le calcul. C'est, sans doute, l'algorithme le plus important en mathématiques appliquées et ingénierie. Charles Van Loan a écrit dans son livre [72] : "*The fast Fourier transform is one of the truly great computational developments of this century. It has changed the face of science and engineering so that it is not an exaggeration to say that life as we know it would be very different without FFT*". L'importance de la FFT provient de deux facteurs distincts : sa présence dans une pléthore d'applications, dans presque tout les secteurs de la technologie informatique, et la disponibilité d'algorithmes rapides et précis pour la calculer. Il est associé aux noms de James Cooley et John Tuckey qui ont publié cet algorithme en 1965. Il calcule rapidement la transformation de Fourier discrète et il a été redécouvert plusieurs fois depuis Gauss, et surtout par Danielson et Lanczos en 1942. La FFT permet de ramener le calcul de la transformation de Fourier discrète de $O(n^2)$ flops à $O(nlogn)$ flops. Voir [48] pour une histoire plus détaillée sur la FFT.

Définition A.5.1 La matrice de Fourier d'ordre n est la matrice F_n suivante :

$$F_n = \frac{1}{\sqrt{n}} \begin{pmatrix} 1 & \cdots & \cdots & 1 \\ 1 & \omega_n & \cdots & \omega_n^{n-1} \\ \vdots & \ddots & \ddots & \vdots \\ 1 & \omega_n^{n-1} & \cdots & \omega_n^{(n-1)^2} \end{pmatrix} = \frac{1}{\sqrt{n}} \left(\omega_n^{ij}\right)_{0 \le i,j \le n-1}.$$

S'il y a pas de confusion on utilise F à la place de F_n.

Définition A.5.2 La transformation de Fourier discrète d'un vecteur $v = (v_0, ..., v_{n-1})^T$ est le vecteur Fv ; il sera noté par la suite $v = {}^t(\widehat{v}_0, ..., \widehat{v}_{n-1})$.

Proposition A.4.1 La matrice F est symétrique et unitaire.

Preuve. La symétrie est évidente. Ecrivons $FF^* = (a_{ij})_{0 \le i,j \le n}$. Comme ${}^H F = \frac{1}{\sqrt{n}} \left(\omega_n^{-ij}\right)_{0 \le i,j \le n-1}$, alors

$$a_{ij} = \frac{1}{n} \sum_{k=1}^{n} \omega_n^{(i-1)(j-1)} \omega_n^{-(k-1)(j-1)} = \frac{1}{n} \sum_{k=1}^{n} \omega_n^{(i-j)(k-1)}.$$

Ainsi, si $i = j$ alors $a_{i,i} = \frac{1}{n} \sum_{k=1}^{n} 1 = 1$ et si $i \neq j$ alors $a_{i,j} = 0$ comme somme des exposants d'une racine $n^{\text{ième}}$ de l'unité. ■

Remarque A.5.1 La matrice de Fourier F_n est la matrice de Vandermonde associée au vecteur $(1, \omega_n, ..., \omega_n^{n+1})$. Donc, si on associe au vecteur v le polynôme $v(x) = \sum_{i=0}^{n-1} v_i x^i$ alors $\widehat{v}$ est simplement le vecteur dont les composantes correspondent à l'évaluation de $v(x)$ aux racines $n^{\text{ième}}$ de l'unité.

Proposition A.5.2 (Algorithme FFT). Si n est une puissance de 2, alors on peut calculer $\widehat{v}$ en $\frac{3}{2} n \log n$ flops.

Preuve. On pose $n = 2m$, et $v^{[0]} = (v_0, v_2, \ldots, v_{n-2})^T$, $v^{[1]} = (v_1, v_3, \ldots, v_{n-1})^T$; notons $\widehat{v}^{[0]}$, $\widehat{v}^{[1]}$ les transformations de Fourier discrètes de taille m de $v^{[0]}$ et $v^{[1]}$ respectivement. Pour j variant de 0 à $n-1$, on a :

$$\begin{aligned} \widehat{v}_j &= \sum_{k=0}^{n-1} v_k \omega_n^{jk} \\ &= \sum_{k=0}^{m-1} v_{2k} \omega_{2m}^{2jk} + \sum_{k=0}^{m-1} v_{2k+1} \omega_{2m}^{2j(k+1)} \\ &= \sum_{k=0}^{m-1} v_{2k} \omega_{2m}^{jk} + \omega_{2m}^{j} \sum_{k=0}^{m-1} v_{2k+1} \omega_m^{jk} \end{aligned}$$

car $\omega_{2m}^{2jk} = e^{2i\pi.2jk/2m} = e^{2i\pi.jk/m} = \omega_m^{jk}$. Donc

$$\begin{cases} \widehat{v}_j = \widehat{v}_j^{[0]} + \omega_n^j \widehat{v}_j^{[1]} & \text{si } 0 \le j \le m-1 \\ \widehat{v}_j = \widehat{v}_{j-m}^{[0]} + \omega_n^j \widehat{v}_{j-m}^{[1]} & \text{si } m \le j \le n-1 \end{cases}$$

Ainsi pour calculer la transformation de Fourier d'un vecteur de taille n, il suffit de calculer la transformation de Fourier de deux vecteurs de taille $n/2$ chacun, de faire $n/2$ multiplications et n additions. En effet, pour $m \leq j \leq n-1$, $\widehat{v}_j = \widehat{v}^{[0]}_{j-m} + \omega^j_n \widehat{v}^{[1]}_{j-m} = \widehat{v}^{[0]}_{j-m} + \omega^{j-m}_n \widehat{v}^{[1]}_{j-m}$, or le nombre $\omega^{j-m}_n \widehat{v}^{[1]}_{j-m}$ est déjà calculé, donc pour $m \leq j \leq n-1$ il suffit de faire une addition. Si $c(n)$ est le nombre d'opérations arithmétiques nécessaires pour calculer la transformation de Fourier d'un vecteur de taille n, alors

$$c\left(n\right) = 2c\left(n/2\right) + 3n/2$$

Par récurrence et en remarquant que $c(1)$ ne demande aucune opération, on a donc

$$c(n) = \frac{3}{2} n log n.$$

■

A.6 DCT

La Transformée en cosinus discrète, DCT (Discrete Cosine Transform), est une transformation proche de la transformée de Fourier discrète (DFT). Le noyau de projection est un cosinus et génère donc des coefficients réels, contrairement à la DFT, dont le noyau est une exponentielle complexe et qui génère donc des coefficients complexes. On peut cependant exprimer la DCT en fonction de la DFT, qui est alors appliquée sur le signal symétrisé. Voir [33] pour une histoire plus détaillée sur la DCT.

La variante la plus courante de la transformée en cosinus discret est la DCT type-II, souvent simplement appelée "la DCT". Son inverse, qui correspond au type-III est souvent simplement appelée "IDCT".

La DCT est une fonction linéaire inversible $\mathbb{R}^N \longrightarrow \mathbb{R}^N$ ou de manière équivalente une matrice carrée $N \times N$ inversible. Il existe plusieurs légères variantes de la DCT. Voici les quatre types les plus connus.

A.6.1 DCT-I

$$X_k = \frac{1}{2}\left(x_0 + (-1)^k x_{N-1}\right) + \sum_{n=1}^{N-2} x_n \cos\left[\frac{\pi}{N-1} nk\right]$$

On peut rendre cette transformée orthogonale (à une constante multiplicative près) en multipliant x_0 et x_{N-1} par $\sqrt{2}$ et réciproquement X_0 et X_{N-1} par $1/\sqrt{2}$. Cette normalisation casse toutefois la correspondance avec une DFT.

On peut noter que la DCT-I n'est pas définie pour $N \leq 2$, contrairement aux autres types qui sont définis pour tout N positif.

A.6.2 DCT-II

$$X_k = \sum_{n=0}^{N-1} x_n \cos\left[\frac{\pi}{N} n \left(k + \frac{1}{2}\right)\right]$$

Cette variante DCT est la plus courante et la plus utilisée. Elle est généralement simplement appelée "la DCT". De la même manière que pour la DCT-I, on peut rendre cette transformation orthogonale en multipliant X_0 par $1/\sqrt{2}$. Cette forme normalisée est très utilisée en pratique mais casse la correspondance avec la DFT.

A.6.3 DCT-III

$$X_k = \frac{1}{2} x_0 + \sum_{n=1}^{N-1} x_n \cos\left[\frac{\pi}{N} \left(n + \frac{1}{2}\right) k\right]$$

La DCT-III est la transformée inverse de la DCT-II. Elle est plus connue sous le nom de "DCT Inverse" et son acronyme (anglais) "IDCT".

De la même manière que pour la DCT-I, on peut rendre cette transformation orthogonale en multipliant X_0 par $1/\sqrt{2}$. Cette forme normalisée est très utilisée en pratique mais casse la correspondance avec la DFT.

A.6.4 DCT-IV

$$X_k = \sum_{n=0}^{N-1} x_n \cos\left[\frac{\pi}{N} \left(n + \frac{1}{2}\right) \left(k + \frac{1}{2}\right)\right].$$

La DCT-IV est une matrice orthogonale.

A.5.5 Applications

1. La DCT, et en particulier la DCT-II est très utilisée en traitement du signal et de l'image, et spécialement en compression. La DCT possède en effet une excellente propriété de "regroupement" de l'énergie : l'information est essentiellement portée par les coefficients basses fréquences. Pour les images naturelles, la DCT est la transformation qui se rapproche le plus de la transformée de Karhunen-Loève qui fournit une décorrélation optimale des coefficients pour un signal markovien. En pratique, les procédés de compression font donc l'hypothèse qu'une image naturelle peut être modélisée comme la réalisation d'un processus markovien et approximent la transformée de Karhunen-Loève, trop complexe en calcul et dépendante des données, par une DCT. Seuls un petit nombre de coefficients sont non-nuls, et peuvent être utilisés pour reconstruire l'image par transformée inverse (IDCT) lors de la décompression. Le gain en termes de compression vient de la suppression des coefficients nuls ou proches

de zéro. Ce genre de mécanisme est utilisé dans les standards JPEG et MPEG, qui utilisent une DCT 2D sur des blocs de pixels de taille 8×8 (pour des raisons de complexité).

2. Les formats de compression de son avec perte AAC, Vorbis et MP3 utilisent une version modifiée de cette technique, la transformée en cosinus discrète modifiée, TCDM (MDCT en anglais).
3. La DCT est aussi employée pour la résolution de systèmes d'équations différentielles par des méthodes spectrales.

Annexe B : Codes Matlab

Nous présentons, dans les pages qui suivent, les codes MATLAB des algorithmes expérimentés. Ces programmes et la liste des matrices de Hankel "exacte", construite au moyen d'un fichier Maple récupéré de Diaz-Toca [20] sont téléchargables et utilisables librement via le lien suivant http ://sbelhaj.mac125.net.

B.1 Les programmes communs

B.1.1 DCT-II

```
function y = dct2(x)
n = length(x) ;
y(2 :2 :2*n,1) = x ;
y = [y ; zeros(2*n,1)] ;
y = real( fft(y) ) ;
y = y(1 :n) ;
```

B.1.2 Calcul du rang approché

```
function [k] = e_rank(B,epselon)
n=length(B) ;
s_v_d=svd(B) ;
i=0 ;
k=0 ;
while (i<length(s_v_d))
if (s_v_d(i+1)>4*n*epselon)
 k=k+1 ;
end
i=i+1 ;
end
```

B.1.3 Compteur des blocs

```
function r = count(S,epselon)
r=0 ;
for k=1 :length(S)
```

```
if((abs(real(S(k)))<=epselon)&&(abs(real(S(k)))>=0)&&(abs(imag(S(k)))<=epselon)&&(abs(imag(S(k)))>=0))

 r=r+1 ;
else
 break ;
end
end
if (r>=length(S))
r=(length(S)-1)/2 ;
end
```

B.1.4 Compteur des blocs via SVD

```
function [p] = countBlocks(B,epselon)
n=length(B) ;
p=0 ;
for i=1 :n
p=p+1 ;
if (e_rank(B(1 :i,1 :i),epselon) < length(B(1 :i,1 :i)))
else
break ;
end
```

B.1.5 Construction d'une matrice de Hankel via deux polynômes

```
function [H] = POLYtoHANKEL(U,V)
n=length(U) ;
m=length(V) ;
V=[V zeros(1,n-m-1)] ;
U0=U(1) ;
U=U(2 :n) ;
n=length(U) ;
H=zeros(1,2*n-1) ;
H(1)=V(n)/U(n) ;
for i = 2 :n
H(i)=(V(n-i+1)-H(1 :i-1)*U(n-i+1 :n-1).')/U(n) ;
end
U=[U0 U] ;
for i = 1 :n-1
H(n+i)=-(H(i :i+n-1)*U(1 :n).')/U(n+1) ;
end
```

B.1.6 Construction d'une matrice de Bézout via deux polynômes

```
function [B] = POLYtoBEZOUT(U,V)
H=POLYtoHANKEL(U,V) ;
m=length(H) ;
n=(m+1)/2 ;
c=H(1 :n) ;
r=H(n :m) ;
H=Hankel(c,r) ;
c=U(2 :n+1) ;
Bu1=Hankel(c) ;
J=hankel([zeros(1,n-1) 1]) ;
B=Bu1*H*Bu1 ;
B=J*B*J ;
```

B.1.7 Transformation de deux polynômes en une matrice de Hankel

```
function [V] = HANKELtoPOLY(U,H)
n=length(U) ;
U0=U(1) ;
U=U(2 :n) ;
n=length(U) ;
J=hankel([zeros(1,n-1) 1]) ;
U=J*U' ;
V=(J*tril(toeplitz(H(1,1 :n)))*U).' ;
```

B.1.8 Transformation de deux polynômes en une matrice de Bézout

```
function [V] = BEZOUTtoPOLY(U,B)
n=length(U) ;
c=U(2 :n) ;
Bu1=Hankel(c) ;
J=hankel([zeros(1,length(B)-1) 1]) ;
H=inv(Bu1)*J*B*J*inv(Bu1) ;
V = HANKELtoPOLY(U,H) ;
V=V(1 :length(V)-count(H(1, :),1e-6)) ;
```

B.1.9 Euclide de deux polynômes

```
function [Mq,Mr,q,r_1] = euclide(U,V,epselon)
n=length(U) ;
m=length(V) ;
```

```
JU=hankel([zeros(1,n-1) 1]);
U=U*JU
JV=hankel([zeros(1,m-1) 1]);
V=V*JV
temp=0;
temp2=0;
r=0;
i=1;
Mq=zeros(n);
Mr=zeros(n);
while (temp2==0)
r_1=r;
[q,r]=deconv(U,V);
temp=V;
c=0;
for j=1 :length(r)
 if (r(j)>=-epselon && r(j)<=epselon)
  c=c+1;
 end
end
if (c==length(r))
 temp2=1;
 break;
else
 temp2=0;
end
if(length(q)>length(V))
 V=[zeros(1,length(q)-length(V)) V];
else
 if (length(q)<length(V))
  q=[zeros(1,length(V)-length(q)) q];
 end
end
r=[zeros(1,length(V)+length(q)-1-length(r)) r];
U=temp;
U=U(count(U,epselon)+1 :length(U));
q=q(count(q,epselon)+1 :length(q));
r=r(count(r,epselon)+1 :length(r));
V=r;
Mq(i,1 :n) = [Mq(i,1 :n-length(q)) q];
Mr(i,1 :n) = [Mr(i,1 :n-length(r)) r];
i=i+1;
end
end
```

B.2 Les programmes d'inversion de matrices supérieures de Toeplitz

B.2.1 Inversion via FFT Division

```
function u = div_fft(f,g)
n=length(f) ;
m=length(g) ;
k=n-m+1 ; % length of u=f/g
least_c_m=lcm(n,m) ;
f_pad=[zeros(1,least_c_m-n) f] ;
g_pad=[zeros(1,least_c_m-m) g] ;
threshold=1e-10 ;
ff=fft(f_pad) ;
gg=fft(g_pad) ;
if min(abs(gg))<threshold
u=lsdiv(f,g) ;
else
uu=ff./gg ;
uuu=ifft(uu) ;
u=zeros(1,k) ;
u(k)=uuu(1) ;
u(1 :k-1)=uuu(least_c_m-k+2 :least_c_m) ;
end
```

B.2.2 Inversion via Least Square Division

```
function u = lsdiv(f,g)
n=length(f)-1 ;
m=length(g)-1 ;
k=n-m ;
g=fliplr(g).' ;
f=fliplr(f).' ;
C=zeros(n+1,k+1) ;
for i=1 :k+1
C(i :i+m,i)=g ;
end
u=fliplr(lsqr(C,f,1e-10,50).') ;
```

B.2.3 Inversion via substitution

```
function x = recurrent(a)
n = length(a) ;
x = zeros(n,1) ;
x(1) = 1/a(1) ;
```

```
for i = 2 :n
x(i :n) = x(i :n) + x(i-1)*a(2 :n-i+2) ;
x(i) = -x(i)/a(1) ;
end
```

B.2.4 Inversion via interpolation

```
function f = interpolation(a,x)
n = length(a) ;
t = zeros(2*n,1) ;
t(1 :n) = a.*x ;
t = fft(t) ;
b = t(1 :n) ;
c = real(b) ; d = imag(b) ;
e = B./(B.^2+d.^2) ;
f = dct2(e)/sqrt(n/2) ;
f(1) = f(1) /sqrt(2) ;
```

B.2.5 Inversion via Bini

```
function b = bini(a,d)
n = length(a) ;
D = d.^(0 :n-1) ;
D = D' ;
a = a.*D ;
b = fft(a) ;
b = 1./b ;
b = ifft(b) ;
b = b./D ;
```

B.3 Les programmes de factorisation de matrices de Hankel

B.3.1 Réduction de la matrice de Hankel réelle via Toeplitz

```
function [n,p,D,H,tprim,hprim] = reductionViaToeplitz_Bini(S,epselon)
N=length(S) ;
p=count(S,epselon)+1 ;
n=(N+1)/2 ;
c=S(1 :n) ;
r=S(n :N) ;
H=Hankel(c,r) ;
T=triu(Toeplitz(S(p :N))) ;
B=bini([S(p :N) zeros(1,(N-p+1))]',(1.0e-10)^(1/(n))) ;
```

```
t=triu(Toeplitz(S(p :n+p-1)));
b=real(B(1 :n));
tprim=triu(Toeplitz(b));
hprim=tprim'*H*tprim;
D=[hprim(1 :p,1 :p),zeros(p,n-p);zeros(n-p,p),eye(n-p)];
```

B.3.2 Réduction de la matrice de Hankel complexe via Toeplitz

```
function [n,p,D,H,tprim,hprim] = reductionViaToeplitz_recurrent(S,epselon)
p=count(S,epselon)+1;
n=(N+1)/2;
c=S(1 :n);
r=S(n :N);
H=Hankel(c,r);
t=recurrent(S(p :n+p-1).');
tprim=triu(Toeplitz(t));
hprim=tprim.'*H*tprim;
D=[hprim(1 :p,1 :p),zeros(p,n-p);zeros(n-p,p),eye(n-p)];
```

B.3.3 Réduction de la matrice de Hankel réelle via Schur

```
function [n,p,A11,A21,A22,L,S,liste] = SchurComplement_Bini(A,epselon)
n=length(A);
liste=A(1,1 :n);
p=count(liste,epselon)+1;
A11=A(1 :p,1 :p);
J=hankel([zeros(1,p-1) 1]);
T11=J*A11;
B=bini([T11(1,1 :p) zeros(1,p*p)]',(0.5e-8)^(1/(p)));
b=real(B(1 :p));
invT11=triu(Toeplitz(b));
invH11Bini=invT11*J;
x=[zeros(1,p-1) A11(p,1)]';
y=A11(p,1 :p);
A11exact=hankel(x,y);
A21=A(p+1 :n,1 :p);
A22=A(p+1 :n,p+1 :n);
L11=eye(p,p);
L21=A21*invA11exact;
L12=zeros(p,n-p);
L22=eye(n-p,n-p);
S=A22-A21*invA11exact*A21';
A11=[A(1 :p,1 :p),zeros(p,n-p);zeros(n-p,p),eye(n-p)];
```

```
L=[L11,L12 ;L21,L22] ;
invL=[L11,L12 ;-L21,L22] ;
```

B.3.4 Réduction de la matrice de Hankel complexe via Schur

```
function [n,p,A11,A21,A22,L,S,liste] = SchurComplement_recurrent(A,epselon)
n=length(A) ;
liste=A(1,1 :n) ;
p=count_complexe(liste,epselon)+1 ;
A11=A(1 :p,1 :p) ;
J=hankel([zeros(1,p-1) 1]) ;
T11=J*A11 ;
t=recurrent(T11(1,1 :p).') ;
invT11=triu(Toeplitz(t)) ;
x=[zeros(1,p-1) A11(p,1)].' ;
y=A11(p,1 :p) ;
A11exact=hankel(x,y) ;
invA11exact=invT11*J ;
A21=A(p+1 :n,1 :p) ;
A22=A(p+1 :n,p+1 :n) ;
L11=eye(p,p) ;
L21=A21*invA11exact ;
L12=zeros(p,n-p) ;
L22=eye(n-p,n-p) ;
S=A22-A21*invA11exact*A21.' ;
A11=[A(1 :p,1 :p),zeros(p,n-p) ;zeros(n-p,p),eye(n-p)] ;
L=[L11,L12 ;L21,L22] ;
invL=[L11,L12 ;-L21,L22] ;
```

B.3.5 Diagonalisation par blocs de Hankel-réelle via Toeplitz

```
function [A,D] = BlockDiagoHankel_new(S,epselon)
p=count(S,epselon)+1 ;
N=(length(S)+1)/2 ;
m=length(S) ;
n=(m+1)/2 ;
S=S ;
X=eye(N) ;
DBlock=eye(N) ;
while (length(S)>=3)
[n,p,D,H,tprim,hprim] = reductionViaToeplitz_Bini(S,epselon)
if(length(tprim)<N)
tprim=[eye(N-length(tprim)),zeros(N-length(tprim),length(tprim)) ; zeros(length(tprim),N-length(tprim)),tprim] ;
end
if (length(D)>=1)&&(length(D)<N)
```

```
D=[eye(N-length(D)),zeros(N-length(D),length(D)) ; zeros(length(D),N-length(D)),D] ;
end
if(n==p)
D=[eye(N-length(H)),zeros(N-length(H),length(H)) ; zeros(length(H),N-length(H)),H] ;
DBlock=DBlock*D ;
break ;
else
if(n<p)
sprintf('%s','Veillez changer la tolérance') ;
break ;
else
S=[hprim(p+1,p+1 :n) hprim(n,p+2 :n)] ;
end
end
X=X*tprim ;
DBlock=DBlock*D ;
end
A=X ;
D=DBlock ;
```

B.3.6 Diagonalisation par blocs de Hankel-complexe via Toeplitz

```
function [A,D] = BlockDiagoHankel_complexe(S,epselon)
p=count(S,epselon)+1 ;
N=(length(S)+1)/2 ;
m=length(S) ;
n=(m+1)/2 ;
S=S ;
X=eye(N) ;
DBlock=eye(N) ;
while (length(S)>=2)
[n,p,D,H,tprim,hprim] = reductionViaToeplitz_complexe(S,epselon) ;
if(length(tprim)<N)
tprim=[eye(N-length(tprim)),zeros(N-length(tprim),length(tprim)) ; zeros(length(tprim),N-length(tprim)),tprim] ;
end
if (length(D)>=1)&&(length(D)<N)
D=[eye(N-length(D)),zeros(N-length(D),length(D)) ; zeros(length(D),N-length(D)),D] ;
end
if(n==p)
D=[eye(N-length(H)),zeros(N-length(H),length(H)) ; zeros(length(H),N-length(H)),H] ;
DBlock=DBlock*D ;
break ;
else
```

```
if(n<p)
sprintf('%s','Changer la tolerance') ;
break ;
else
S=[hprim(p+1,p+1 :n) hprim(n,p+2 :n)] ;
end
end
X=X*tprim ;
DBlock=DBlock*D ;
end
A=X ;
D=DBlock ;
```

B.3.7 Diagonalisation par blocs de Hankel-réelle via Schur

```
function [A,D] = BlockDiagoHankel_Schur_Bini(S,epselon)
p=count(S,epselon)+1 ;
M=length(S) ;
N=(length(S)+1)/2 ;
n=(M+1)/2 ;
c=S(1 :n) ;
r=S(n :M) ;
A=Hankel(c,r) ;
S=A ;
liste=S ;
X=eye(N) ;
DBlock=eye(N) ;
while (length(liste)>=2)
[n,p,A11,A21,A22,L,S,liste] = SchurComplement_Bini(S,epselon) ;
if(length(L)<N)
L=[eye(N-length(L)),zeros(N-length(L),length(L)) ; zeros(length(L),N-length(L)),L] ;
else
L=L ;
end
if (length(A11)>=1)&&(length(A11)<N)
A11=[eye(N-length(A11)),zeros(N-length(A11),length(A11)) ; zeros(length(A11),N-length(A11)),A11] ;
end
X=X*L ;
DBlock=DBlock*A11 ;
if(n==p)
A11=[eye(N-length(A22)),zeros(N-length(A22),length(A22)) ; zeros(length(A22),N-length(A22)),A22] ;
DBlock=DBlock*A11 ;
break ;
end
end
```

```
A=X ;
D=DBlock ;
```

B.3.8 Diagonalisation par blocs de Hankel-complexe via Schur

```
function [A,D] = BlockDiagoHankel_Schur_complexe(S,epselon)
p=count_complexe(S,epselon)+1 ;
M=length(S) ;
N=(length(S)+1)/2 ;
n=(M+1)/2 ;
c=S(1 :n) ;
r=S(n :M) ;
A=Hankel(c,r) ;
S=A ;
liste=S ;
X=eye(N) ;
DBlock=eye(N) ;
while (length(liste)>=2)
[n,p,A11,A21,A22,L,S,liste] = SchurComplement_recurrent(S,epselon) ;
if(length(L)<N)
L=[eye(N-length(L)),zeros(N-length(L),length(L)) ; zeros(length(L),N-length(L)),L] ;
else
L=L ;
end
if (length(A11)>=1)&&(length(A11)<N)
A11=[eye(N-length(A11)),zeros(N-length(A11),length(A11)) ; zeros(length(A11),N-length(A11)),A11] ;
end
X=X*L ;
DBlock=DBlock*A11 ;
if(n==p)
A11=[eye(N-length(A22)),zeros(N-length(A22),length(A22)) ; zeros(length(A22),N-length(A22)),A22] ;
DBlock=DBlock*A11 ;
break ;
end
end
A=X ;
D=DBlock ;
```

Bibliographie

[1] J. Abdeljaoued, H. Lombardi, Méthodes matricielles. Introduction à la Complexité Algébrique, Springer, collection "Mathématiques et applications" de la SMAI, 2003.

[2] K.E. Atkinson, An Introduction to Numerical Analysis, 2nd edition, John Wiley and Sons, Inc, 1989.

[3] N. I. Akhiezer, M. Krein, Some questions in the theory of moments, AMS, 1962.

[4] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe and H. van der Vorst, editors, Templates for the solution of Algebraic Eigenvalue Problems : A Practical Guide, SIAM, Philadelphia, 2000.

[5] S. Basu, R. Pollack and M.F. Roy. Algorithms in real algebraic geometry. Springer, 2003.

[6] S. Belhaj, A fast method to block-diagonalize a Hankel matrix, Numer Algor 47, (2008) 15-34.

[7] S. Belhaj, Block factorization of a Hankel matrix and Euclidean Algorithm, Math. Model. Nat. Phenom. 5 (7), (2010) 38-44.

[8] S. Belhaj, Computing the block factorization of complex Hankel matrices, Computing 87 (3-4), (2010) 169-186.

[9] S. Belhaj, Computing the block factorization of complex Hankel matrices : application to the Euclidean algorithm, 16th ILAS Conference June 21-25, 2010 Pisa, Italy.

[10] S. Belhaj, Block diagonalization of Hankel and Bézout matrices : connection with the Euclidean Algorithm, Numerical Analysis and Scientific Computing with Applications, May 18-22, 2009 Agadir, Morocco.

[11] S. Belhaj, Factorisation des matrices de Hankel et de Bézout : connexion avec l'algorithme d'Euclide" TAM TAM'09 4ème Colloque sur les Tendances dans les Applications Mathématiques en Tunisie, Algérie, Maroc, 4-8 Mai 2009, Kénitra, Maroc.

[12] S. Belhaj, Factorisation par blocs des matrices de Hankel et l'algorithme d'Euclide, JANO'09, Mohammedia, Maroc, 60-63 (Décembre 2008).

[13] S. Belhaj, Sur le calcul de la suite des restes et des quotients via la matrice de Hankel, CFMCF'01, Kerkennah, Tunisie, (Mai 2008).

[14] S. Belhaj, Block factorization of a Hankel matrix and Euclidean Algorithm, IMACS'09, Lille, France (Mars 2008).

[15] S. Belhaj, Factorisation approchée des matrices de Hankel, EECFTN'07, Monastir, Tunisie (Septembre 2007).

[16] S. Belhaj, Factorisation de la matrice de Hankel, READ'07, Tunis, Tunisie (Juillet 2007).

[17] S. Belhaj, Une méthode rapide de diagonalisation par blocs approchée de la matrice de Hankel, TAM-TAM'07, Alger, Algérie (Avril 2007).

[18] S. Belhaj, Sur une méthode de diagonalisation par blocs approchée de la matrice de Hankel, EJCMI'07, Nancy, France (Mars 2007).

[19] N. Ben Atti, Calcul rapide sur les matrices structurées : les matrices de Hankel, thèse de doctorat, Université de Franche-Comté, Novembre 2008.

[20] N. Ben Atti, G. M. Diaz-Toca, Block diagonalization and LU-equivalence of Hankel matrices, Linear Algebra and its Applications 412 (2006) 247-269.

[21] D. Bini, Parallel solution of certain Toeplitz linear systems, SIAM J. Comput. 13 (1984) 268-276.

[22] D. Bini, L. Gemignani, Fast parallel computation of the polynomial remainder sequence via Bézout and Hankel matrices, SIAM J. Comput. 24 (1995) 63-77.

[23] D. A. Bini, L. Gemignani, Fast fraction-free triangularization of Bézoutians with applications to subresultant chain computation, Linear Algebra Appl. 284 (1-3), (1998) 19-39.

[24] D. Bini, V. Pan, Polynomial division and its computational complexity, J. Complexity 2 (1986) 179-203.

[25] D. Bini, V. Pan, Polynomial and Matrix Computations, Vol. 1, Fundamental Algorithms, Birkhäuser, 1994.

[26] C. Brezinski, M. Redivo Zaglia, Méthodes Numériques Directes de l'Algèbre Matricielle. Ellipses, Paris, 2004.

[27] L. Brutman, Lebesgue functions for polynomial interpolation—a survey, Ann. Numer. Math. 4 (1997) 111-127.

[28] A. Bultheel, M. Van Barel, Linear algebra, rational approximation and orthogonal polynomials, Studies in computational Mathematics, vol 6, Elsevier/North-Holland, Amsterdam, 1997.

[29] P. Boito, Structured Matrix Based Methods for Approximate Polynomial GCD, Ph. D. thesis, Scuola Normale Superiore, Pisa University, Pisa, October 2007.

[30] A. Borodin, J. von zur Gathen, J. Hopcroft, Fast parallel matrix and gcd computation, information and control, 52 (1982) 241-256.

[31] W. S. Brown, On Euclid's and the computation ofpolynomial greate st common divisors, Journal of the ACM 18 (4), (1971) 478-504.

[32] W. S. Brown, J. F. Traub, On Euclid's Algorithm and the Theory of Subresultants, J. Assoc. Comput. Mach. 18, (1971) 505-514.

[33] W. Chen, C.H. Smith, and S.C. Fralick, "A fast computational algorithm for the discrete cosine transform," IEEE Trans. Commun., COM-25, (1977) 1004-1009.

[34] G. E. Collins, Subresultants and reduced polynomial remainder sequences, J. Assoc. Comput. Mach. 14, (1967) 128-142.

[35] E.W. Cheney, Introduction to Approximation Theory, AMS Chelsea Publishing, New York, 1982.

[36] R. E. Cline, R. J. Plemmons, G. Worm, Generalized Inverses of Certain Toeplitz Matrices, Linear Algebra and its Applications 8, (1974) 25-33.

[37] D. Commenges, M. Monsion, Fast inversion of triangular Toeplitz matrices, IEEE Trans. Automat. Control AC-29 (1984) 250-251.

[38] R. Chan, M. K. Ng, Conjugate gradient methods for Toeplitz systems, SIAM Rev. 38 (1996) 427-482.

[39] G. M. Diaz-Toca, N. Ben Atti, Block LU factorization of Hankel and Bézout Matrices and Euclidean Algorithm, Int. J. Comput. Math. 86, (2009) 135-149.

[40] G.M. Diaz–Toca and L. Gonzalez–Vega. Barnett's Theorem about the greatest common divisor of several univariate polynomials through Bézout–like Matrices. Journal of Symbolic Computation 34 (1), (2002) 59-81.

[41] I. Z. Emiris, A. Galligo, H. Lombardi, Certified approximate univariate GCDs, J. Pure Appl. Algebra 117-118, (1997) 229-251.

[42] I. Z. Emiris, A. Galligo, H. Lombardi, Numerical Univariate Polynomial GCD, in : J. Renegar, M. Shub and S. Smale, eds., The Mathematics of Numerical Analysis, Lectures in Applied Mathematics 32 (AMS, Providence, RI, 1996), 323-343.

[43] L. Gemignani, GCD of Polynomials and Bézout Matrices, Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation (Kihei, Maui, HI), ACM Press, New York (1997) 271-277.

[44] G.H. Golub, P. Milanfar, J. Varah, A stable numerical method for inverting shape from moments, Technical Report SCCM-97-10, Stanford University, Sei. Comp. and Comp'l. Math. Pgm, 1997.

[45] G. Golub, Ch. Van Loan, Matrix Computations, J. Hopkins Univ. Press, Baltimore and London. 3ème édition, 1996.

[46] W. B. Gragg, A. Lindquist, On partial realization problem, Linear Algebra Appl. 50 (1983) 277-319

[47] Habicht, W. Eine Verallgemeinerung des Sturmschen Wurzelzahlverfahrens. Comm. Math. Helvetici, 21, (1948) 99-116.

[48] M Heideman, D Johnson, and C. Burrus. Gauss and the history of the fast fourier transform. IEEE ASSP Magazine, 1, (1984) 14-21.

[49] G. Heining, K. Rost, Algebraic methods for Toeplitz-like matrices and operators, Birkhäuser Verlag, Basel, 1984.

[50] G. Heinig, U. Jungnickel, Hankel matrices generated by Markov Parameters, Hankel matrix extension, partial realization, and Pade approximation, in : Operator Theory : Advances and Applications, Birkhauser, Boston, 19 (1986) 231-254.

[51] U. Helmke and P.A. Fuhrmann, Bézoutians, Linear Algebr. Appl., 122/123/124 , (1989) 1039-1097.

[52] V. Hribernig, Sensitivity of Algebraic Algorithms, Ph.D. Thesis at the Technical University of Vienna, 1994.

[53] V. Hribernig, H. J. Stetter, Detection and validation of clusters of polynomial zeros, J. Symb. Comp. 24(6), (1997) 667-681.

[54] A.K. Jain, Fundamentals of Digital Image Processing, Prentice-Hall, Inc., Englewood Cliffs, NY, 1989.

[55] T. Kailath and A.H. Sayed, editors, Fast Reliable Algorithms for Matrices with Structure,SIAM, Philadelphia, 1999.

[56] R.E. Kalman, On partial realizations, transfer functions and canonical forms, Acta Polytech. Stand. MA31 (1979) 9-32.

[57] S. Y. Kung, Multivariable and Multidimensional Systems, PhD thesis, Stanford University, Stanford, CA, June 1977.

[58] P. Lancaster, M. Tismenetsky, The Theory of Matrices, 2nd edition, Computer Science and Applied Mathematics, Academic Press Inc., Orlando, FL (1985).

[59] M.-T. Noda, T. Sasaki, Approximate GCD and its application to ill-conditioned algebraic equations, J. Comput. Appl. Math. 38 (1-3), (1991) 335-351.

[60] Fu-Rong Lin, Wai-Ki Ching, M. K. Ng, Fast inversion of triangular Toeplitz matrices, Theoretical computer Science 315 (2004) 511-523.

[61] D. Pal, Fast algorithms for structured matrices with arbitrary rank profile, PhD thesis, Stanford University, Stanford, 1990.

[62] Victor Y. Pan, Structured matrices and polynomials : Unified Superfast Algorithms, Springer Verlag, 2001.

[63] V. Y. Pan, Z. Q. Chen, Approximate real polynomial division via approximate inversion of real triangular Toeplitz matrices, Applied Mathematics Letters 12 (1999) 1-2.

[64] D. Pal, T. Kailath, Fast Triangular Factorization of Hankel and Related Matrices with Arbitrary Rank Profile, SIAM J. Matrix Anal. Appli. 16 (1990) 451-478.

[65] V. Pan, E. Landowne, A. Sadikou, Univariate polynomial division with a remainder by means of evaluation and interpolation, Proceedings of the Third IEEE Symposium on Parallel and Distributed Processing, December 1991, Dallas, TX, USA, 1991, 212-217.

[66] V. Pan, A. Sadikou, E. Landowne, Polynomial division with a remainder by means of evaluation and interpolation, Inform. Process. Lett. 44 (1992) 149-153.

[67] J. Phillips, The triangular decomposition of Hankel Matrices, Math. Comp. 25 (1971) 599-602.

[68] J. Rissanen, Algorithms for Triangular Decomposition of Block Hankel and Toeplitz Matrices with Applications to Factoring Positive Polynomials, Mathematics of Computation, 27 (1973) 147-154.

[69] T.J. Rivlin, The Lebesgue constants for polynomial interpolation, in : H.C. Garnier, et al., (Eds.), Functional Analysis and its Applications, Springer, Berlin, 1974, 422-437.

[70] A. Schönhage, Asymptotically fast algorithms for the numerical multiplication and division of polynomials with complex coefficients, in Proceedings, EUROCAM, Marseille, 1982.

[71] A. Schönhage, Quasi-GCD Computations, J. Complexity, 1 (1), (1985) 118-137.

[72] Charles Van Loan. Computational frameworks for the fast Fourier transform, volume 10 of Frontiers in Applied Mathematics. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1992.

[73] F.-Z Zhang ed., The Schur Complement and its applications, Springer, New York, 2005.

Résumé

Dans cette thèse, nous visons l'amélioration de quelques algorithmes en algèbre matricielle rapide et plus spécifiquement les algorithmes rapides sur les matrices structurées en calcul formel et numérique. Nous nous intéressons en particulier aux matrices de Hankel et de Toeplitz. Nous introduisons un nouvel algorithme de diagonalisation par blocs approchée de matrices réelles de Hankel. Nous décrivons la relation naturelle entre l'algorithme d'Euclide et notre factorisation par blocs approchée pour les matrices de Hankel associées à deux polynômes, ainsi que pour les matrices de Bézout associées aux mêmes polynômes. Enfin, dans le cas complexe, nous présentons un algorithme révisé de notre diagonalisation par blocs approchée des matrices de Hankel, en calculant la suite des restes et la suite des quotients apparues au cours de l'exécution de l'algorithme d'Euclide.

Mots clés : matrice de Toeplitz, matrice de Hankel, matrice de Bézout, complément de Schur, diagonalisation par blocs, Algorithme d'Euclide

AMS Classification : 15A23, 15B05, 65Fxx, 11Cxx

Abstract

We introduce a new algorithm for the approximate block factorization of real Hankel matrices. We then describe the natural relationship between the Euclidean algorithm and our approximate block factorization, not only for Hankel matrices associated to two polynomials but also for Bézout matrices associated to the same polynomials. Finally, in the complex case, we present a revised algorithm for our approximate block factorization of Hankel matrices by calculating the approximate polynomial quotients and remainders appearing in the Euclidean algorithm.

Keywords : Toeplitz matrix, Hankel matrix, Bézout matrix, Schur complementation, block factorization, Euclidean Algorithm