N° d'ordre 646

THÈSE de DOCTORAT
UNIVERSITÉ JEAN MONNET, SAINT-ÉTIENNE

Spécialité : Analyse Numérique

Présentée par : Françoise TISSEUR

pour obtenir le diplôme de

Docteur de l'Université Jean Monnet

MÉTHODES NUMÉRIQUES POUR LE CALCUL D'ÉLÉMENTS SPECTRAUX

Étude de la précision, la stabilité et la parallélisation

Mémoire soutenu le 28 mai 1997 devant le jury composé de

Mme. Filomena d'ALMEIDA	Universidade do Porto
Mr. Zhajoun BAI	University of Kentucky, Rapporteur
Mr. Bernard PHILIPPE	IRISA de Rennes, Rapporteur
Mr. Frédéric DESPREZ	Ecole Normale Supérieure de Lyon
Mr. Mario AHUES	Université Jean Monnet, Directeur de Thèse
Mr. Alain LARGILLIER	Université Jean Monnet, Directeur de Thèse

Je remercie vivement Filomena d'Almeida pour s'être déplacée de Porto et me faire l'honneur de présider ce jury.

Je remercie mes directeurs de thèse Mario Ahues et Alain Largillier qui ont eu la patience et la gentillesse de me guider tout au long de ce travail. Je les remercie pour l'ensemble des remarques qui m'ont permis d'améliorer la qualité de ce document.

Je tiens à exprimer mes sincères remerciements à mes rapporteurs Zhaojun Bai et Bernard Philippe. Je leur suis reconnaissante pour le temps et le travail qu'ils ont accordés à l'évaluation de ce mémoire.

Je remercie Frédéric Desprez pour sa participation au jury, pour ses nombreuses suggestions et ses encouragements sur mon travail de parallélisation.

Un grand merci à Bernard Tourancheau, Stéphane Domas, Makan Pourzandi, à mes amis du LIP pour la gentillesse de leur accueil et pour leur aide. Mes excuses à ceux que je n'ai pas cités mais qui se reconnaitront ici, qu'ils sachent que je leur suis reconnaissante.

Je remercie Nick pour son soutien, l'attention particulière qu'il a porté à mes travaux, pour sa constante disponibilité à mon égard et ses précieux conseils.

Pour finir, et c'est par là qu'il aurait fallu commencer, je remercie mes parents pour leur soutien tout au long de mes études. Je leur dédie cette thèse. Je crois qu'ils sont fiers de moi autant que je suis fière d'eux.

Table des matières

Liste des tableaux

Table des figures

Chapitre 1

Introduction Générale

Les vibrations sont partout et bien sûr aussi, les valeurs propres (ou fréquences) qui leur sont associées.

(Parlett, The Symmetric Eigenvalue Problem.)

Le problème de valeurs propres intervient dans de nombreux contextes allant du traitement du signal en temps réel [105], à l'analyse des réseaux électriques [101], à la modélisation acoustique et électromagnétique des guides d'ondes [68],[102]. Ce problème se rencontre également en mécanique quantique, ingénierie, hydrodynamique, physique du plasma. Kerner [66, p. 47–79] en donne d'ailleurs d'intéressantes applications.

Les différences entre tous ces problèmes disparaissent lorsque leur formulation devient suffisamment abstraite, c'est-à-dire lorsque le problème pratique est réduit à l'unique tâche de calculer les éléments propres d'une matrice carrée réelle ou complexe.

Il existe une très grande variété d'algorithmes pour calculer les éléments propres de matrices. Le choix d'un algorithme dépend des propriétés et de la structure de la matrice : matrice symétrique ou matrice non symétrique, matrice dense ou matrice creuse, de ce que l'on veut calculer : toutes les valeurs propres ou seulement quelques unes, tous les vecteurs propres associés ou juste une partie, les sous-espaces invariants, de l'ordinateur que l'on possède : machine séquentielle, vectorielle ou parallèle.

Cette thèse porte sur les méthodes numériques pour le calcul des éléments propres de matrices denses et leurs parallélisations. C'est un sujet relativement vaste, la littérature y est abondante et la recherche très active tant du point de vue séquentiel que du point de vue parallèle. Pour ces raisons, nous débuterons ce mémoire par un bref état de l'art de l'algèbre linéaire numérique pour le problème de valeurs propres de matrices denses. Cette synthèse reflète une part importante du travail réalisé et permet d'en situer le contexte.

> *Néanmoins, il y a une distinction qu'il faut retenir : certaines matrices ont des valeurs propres réelles, d'autres pas $\cdots$ il y a des avantages à traiter les deux cas séparément.*
>
> (Parlett, The Symmetric Eigenvalue Problem.)

1.1 Algèbre Linéaire pour le Problème de Valeurs Propres. État de l'Art des Différentes Méthodes.

L'algèbre linéaire pour le problème de valeurs propres est encore aujourd'hui un domaine de recherche extrêmement actif. Nous donnons plusieurs raisons à cela.

- Premièrement, nous n'avons pas encore entièrement compris certains algorithmes classiques. Par exemple, nous n'avons toujours pas de preuve générale pour la convergence de l'algorithme QR [113]. Sa stabilité directe est encore mal comprise [88], [117].

- Deuxièmement, les nouvelles architectures d'ordinateurs nous forcent à implanter de nouveaux algorithmes, à modifier les anciens, et à remettre à l'ordre du jour de vieilles méthodes que l'on avait auparavant écartées. En particulier, les ordinateurs parallèles nous obligent à modifier notre manière classique de compter le nombre d'opérations d'un algorithme pour en donner sa complexité. Ainsi, certains algorithmes considérés inefficaces en séquentiel car trop coûteux, sont à nouveau intéressants.
- Troisièmement, les applications ont stimulé le développement de nouvelles théories et techniques. En chimie, les problèmes de dynamique moléculaire requièrent la résolution de problèmes de valeurs propres de matrices larges symétriques et complexes [62], [78], [99]. Il n'y pas vraiment d'algorithme disponible et approprié pour ce problème dans les bibliothèques d'algèbre linéaire numérique.

Dans la suite, nous présentons les recherches faites en algèbre linéaire numérique autour du problème de valeurs propres de matrices denses durant ces dernières années, ainsi que les algorithmes actuellement utilisés par les bibliothèques d'algèbre linéaire numérique et les investigations faites du coté parallélisme.

1.1.1 Le Problème de Valeurs Propres Symétrique

Matrices Tridiagonale

Les algorithmes *divide and conquer* (diviser pour mieux régner) représentent un développement important et récent dans le calcul des valeurs propres d'une matrice symétrique tridiagonale T. Leur objectif est de diviser le problème en deux sous-problèmes de taille plus petite. La matrice tridiagonale T est écrite sous la forme

$$T = \begin{pmatrix} T_{11} & 0 \\ 0 & T_{22} \end{pmatrix} + \alpha v v^T$$

où T_{11}, respectivement T_{22}, diffère de T seulement par son dernier, respectivement premier, coefficient diagonal. Les éléments propres de T_{11} et T_{22} sont construits et conduisent à

$$T_{11} = Q_1 D_1 Q_1^T, \quad T_{22} = Q_2 D_2 Q_2^T.$$

Nous avons alors

$$\begin{aligned} T &= \begin{pmatrix} Q_1 D_1 Q_1^T & 0 \\ 0 & Q_2 D_2 Q_2^T \end{pmatrix} + \alpha v v^T \\ &= \begin{pmatrix} Q_1 & 0 \\ 0 & Q_2 \end{pmatrix} (D + \alpha \tilde{v}\tilde{v}^T) \begin{pmatrix} Q_1 & 0 \\ 0 & Q_2 \end{pmatrix}^T \end{aligned}$$

où $\tilde{v} = \mathrm{diag}(Q_1, Q_2)^T v$ et $D = \mathrm{diag}(D_1, D_2)$. Les éléments propres de la matrice perturbée de rang 1, $D + \alpha\tilde{v}\tilde{v}^T$, sont les racines de *l'équation séculaire*

$$f(\lambda) = 1 + \alpha \sum_{j=1}^{n} \frac{\tilde{v}_j^2}{d_j - \lambda} = 0.$$

Cette équation est obtenue en annulant le polynôme caractéristique. Pour chaque valeur propre λ_i, un vecteur propre unitaire correspondant est donné par

$$\tilde{q}_i = \left(\frac{\tilde{v}_1}{d_1 - \lambda_i}, \ldots, \frac{\tilde{v}_n}{d_n - \lambda_i} \right)^T \Bigg/ \sqrt{\sum_{j=1}^{n} \frac{\tilde{v}_j^2}{(d_j - \lambda_i)^2}} \; .$$

Cela nous permet de construire la décomposition $D + \alpha\tilde{v}\tilde{v}^T = \widetilde{Q}\widetilde{D}\widetilde{Q}$. Ainsi, $T = U\widetilde{D}U^T$ où $U = \mathrm{diag}(Q_1, Q_2)\widetilde{Q}$. Cet algorithme est attribué à Cuppen [28], la résolution efficace de l'équation séculaire à Bunch, Nielsen et Sorensen [21] basée sur des travaux de Golub [47]. Cet algorithme, pourtant simple et attractif, était considéré instable. Pour garantir l'orthogonalité des vecteurs propres, l'équation séculaire devait être résolue en doublant la précision des données [10]. Récemment, Gu et Eisenstat ont trouvé un moyen d'implanter cet algorithme ne nécessitant pas la précision étendue [51]. Comme le fait remarquer Demmel [32], il a fallu plusieurs années de recherches pour contourner le manque de précision due à l'arithmétique en nombres flottants. Même en séquentiel, cet algorithme est beaucoup plus performant que l'algorithme QR. Il nécessite cependant plus de stockage mémoire. Il a été incorporé à la librairie LAPACK (voir TAB 1.1.1). Par sa nature, l'algorithme Divide and Conquer possède un fort potentiel de parallélisme. Une analyse et implantation parallèle a été faite par Dongarra et Sorensen [37].

Lorsque seule une partie des éléments propres est souhaitée, il est intéressant d'utiliser la méthode de bissection suivie de l'itération inverse. Cela permet de construire soit les valeurs propres appartenant à un intervalle donné, soit les valeurs propres comprises entre la $i^{ème}$ et la $j^{ème}$ lorsqu'elles ont été numérotées par modules décroissants (voir TAB 1.1.1). Soit $a_1, \ldots, a_n$, les éléments diagonaux de T et $b_1, \ldots, b_{n-1}$ les éléments sur et sous-diagonaux. Ces coefficients sont reliés entre eux par la suite récurrente de Sturm

$$d_i = (a_i - \sigma)d_{i-1} - b_{i-1}^2 d_{i-2}, \tag{1.1.1}$$

où d_i est le déterminant de la sous-matrice principale d'ordre i de la matrice $T - \sigma I$ et $d_0 = 1$. L'idée de base consiste à compter le nombre de fois $count(\sigma)$ où la suite $(d_i)_{i=1,n}$ change de signe. La fonction $count(\sigma)$ est égale au nombre de valeurs propres de T inférieures à σ. Lorsque $count(\sigma)$ n'est pas strictement croissante, l'algorithme de bissection peut donner des résultats incorrects (cf. implantation de EISPACK [104]). Demmel, Dhillon et Ren [33] ont résolu ce problème et proposent des corrections pour différentes implantations de la fonction $count(\sigma)$ et différentes hypothèses sur l'arithmétique utilisée.

Les méthodes précédentes ainsi que la méthode QR implicite shiftée et sa variante QL s'appliquent à des matrices préalablement réduites sous forme tridiagonale. Cette étape de tridiagonalisation est maintenant effectuée par bloc grâce à la représentation WY d'une série de produits de matrices de Householder introduite par Bischof et Van Loan [19]. De nombreuses recherches sont toujours en cours du coté parallélisme [18], [26], [70],.

L'algorithme QR et la méthode de bissection peuvent s'écrire sous la forme d'une récurrence à trois termes (voir équation (1.1.1)). Un moyen intéressant d'introduire le parallélisme dans de telles récurrences est de les réécrire sous la forme d'une récurrence à deux termes à l'aide d'un vecteur :

$$\begin{bmatrix} d_i \\ d_{i-1} \end{bmatrix} = \begin{bmatrix} a_i - \sigma & -b_{i-1}^2 \\ 1 & 0 \end{bmatrix} . \begin{bmatrix} d_{i-1} \\ d_{i-2} \end{bmatrix} = L_i \begin{bmatrix} d_{i-1} \\ d_{i-2} \end{bmatrix} = L_i L_{i-1} \dots L_1 . \begin{bmatrix} d_0 \\ d_{-1} \end{bmatrix} .$$

Cette récurrence est alors évaluée en $\log_2(n)$ étapes en utilisant l'opération préfixe parallèle également appelée *scan* [31]. Ces types d'algorithmes sont cependant numériquement instables bien que, en pratique, les instabilités apparaissent rarement [83]. Ceci constitue un exemple pour lequel l'augmentation du parallélisme entraîne une perte de stabilité.

L'itération inverse pour le calcul des valeurs propres est facilement parallélisable. Malheureusement, contrairement aux algorithmes QR ou divide and conquer, elle ne garantit rien sur l'orthogonalité des vecteurs propres construits surtout lorsque l'on est en présence de valeurs propres très voisines. L'approche séquentielle de Jessup et Ipsen [65] consiste à orthogonaliser chaque itérée pendant l'itération inverse par rapport aux vecteurs propres précédemment construits. Cette approche n'est pas parallélisable. Leur version parallèle [63] consiste à calculer tous les vecteurs propres en parallèle et à utiliser un Gram-Schmidt modifié pipeliné pour l'orthogonalisation. Le prix à payer pour garantir l'orthogonalité des vecteurs propres réduit le parallélisme.

ScaLAPACK propose une version parallèle de la méthode de bissection et de l'itération inverse. L'intervalle d'étude est partagé en intervalles disjoints distribués aux processeurs. Chaque processeur évalue de manière séquentielle la suite de Sturm sur ses données. Pour l'itération inverse, la phase de réorthogonalisation est effectuée par Gram-Schmidt et limitée aux valeurs propres localisées sur un même processeur pour éviter les communications. L'orthogonalité des vecteurs propres n'est donc pas assurée en présence de clusters de valeurs propres [34].

Parlett, Fernando et Dhillon travaillent actuellement sur un algorithme qui élimine la phase de réorthogonalisation tout en garantissant une grande précision et une grande efficacité quelle que soit la distribution des valeurs propres.

Matrices Denses

La vieille méthode de Jacobi, contrairement aux méthodes précédentes, opère sur la matrice d'origine. Elle a longtemps été écartée à cause de sa convergence trop lente. Ces dernières années montrent un regain d'intérêt pour son utilisation. Il y a deux raisons à cela. Premièrement, Demmel et Veselić ont montré que cette méthode est plus stable que la méthode QR [35]. Il faut tout de même mettre des restrictions sur ce résultat. Il n'est valable que pour une implantation appropriée de Jacobi [103] et pour une certaine classe de matrices incluant les matrices symétriques définies positives. En particulier, il est démontré que Jacobi construit les valeurs propres à la même précision que les incertitudes présentes dans les données initiales. Ce résultat ne peut être obtenu par les algorithmes passant par une tridiagonalisation de la matrice à cause de la propagation des erreurs d'arrondi. La réduction peut introduire des erreurs importantes sur les valeurs propres de petite taille. Tous ces résultats proviennent d'un nouveau style de théorie des perturbations introduit par Barlow et Demmel [9] et poursuivi par Slapničar [103] et Mathias [82].

Cette méthode n'est pas implantée dans LAPACK car son noyau principal utilise des BLAS de niveau 1 (cf Table 6.3.1) peu efficaces. Des recherches sont actuellement menées pour une implantation par blocs permettant l'introduction des BLAS de niveau 3. Il n'est cependant pas prouvé que l'analyse d'erreur développée dans [35], [103], soit encore valable.

Le second intérêt de Jacobi est son fort potentiel de parallélisme [46], [69], [95], [93]. Les rotations pour éliminer les paires d'éléments hors diagonaux peuvent être effectuées en parallèle en respectant un ordre convenablement choisi. L'algorithme de Jacobi reste un algorithme de parallélisme de grains fins. Littlefield et Maschoff proposent une version parallèle de Jacobi compétitive avec les versions parallèles de QR ou de bissection dans le cas où le nombre de processeurs est grand [79].

Dans le but d'exploiter les BLAS de niveau 3, de nouveaux algorithmes sont actuellement développés. Auslander et Tsao [4] et Ledermann, Tsao et Turnbull[74] ont proposé un algorithme basé sur des produits matrice-matrice pour construire les sous-espaces invariants d'une matrice A. Soit α un annhilateur de sous-espaces de rang r, c'est-à-dire une transformation qui transporte exactement $n - r$ valeurs propres de A sur 0. Alors, les colonnes de $\alpha(A)$ engendrent un sous-espace invariant par A de dimension r. Soit X, une matrice orthogonale dont les colonnes engendrent le même sous-espace vectoriel que $\alpha(A)$ et $Q = [X, Y]$ une matrice orthogonale. Alors,

$$Q^T A Q = \begin{bmatrix} A_1 & 0 \\ 0 & A_2 \end{bmatrix} \text{ avec } A_1 \in \mathbb{R}^{r\times r.}$$

Ce processus peut être appliqué récursivement, conduisant à un algorithme divide and conquer. Comme annhilateur de sous-espaces, ils utilisent des polynômes $\mathcal{B}_k$ de degré $2k+1$ définis sur $[0, 1]$ par

$$\mathcal{B}_k(x) = \frac{\int_0^x t^k(1-t)^k dt}{\int_0^1 t^k(1-t)^k dt},$$

et tels que

$$\lim_{i\to\infty} \mathcal{B}_k^{(j)}(x) = \chi(x)$$

où

$$\mathcal{B}_k^{(j)}(x) = \underbrace{\mathcal{B}_k(\mathcal{B}_k(\ldots(\mathcal{B}_k(x))))}_{j \text{ fois}} \quad \text{et} \quad \chi(x) = \begin{cases} 0, \text{ si } 0 \le x < \frac{1}{2}; \\ \frac{1}{2}, \text{ si } x = \frac{1}{2}; \\ 1, \text{ si } \frac{1}{2} < x \le 1. \end{cases}$$

Type d'algorithme	**Type de calculs**	**Propriétés**
QR/QL hybride implicite shifté	Toutes les valeurs propres et/ou tous les vecteurs propres	Traite de manière plus efficace les matrices gradées [50].
divide and conquer	Toutes les valeurs propres et tous les vecteurs propres	Beaucoup plus rapide que QR/QL. Nécessite plus de place mémoire.
Factorisation de Cholesky + QR bidiagonal	Toutes les valeurs propres et/ou tous les vecteurs propres	Pour matrice symétrique définies positives. Très précise.
Bissection et Itération inverse	Toutes ou une partie des valeurs propres et vecteurs propres	Calcul les valeurs propres d'un intervalle bien défini. La précision peut être ajustée.

TAB. 1.1.1: Liste des routines disponibles dans LAPACK pour calculer les valeurs propres et vecteurs propres de matrices symétriques tridiagonales .

C'est la construction de cet annhilateur qui utilise de nombreux produits matrice-matrice.

Toujours dans le but de profiter de l'efficacité des BLAS de niveau 3, Yau et Lu proposent une méthode d'accélération polynômiale réduisant le problème de valeurs propres à une série de produits matriciels [123]. L'étude de la parallélisation de leur méthode fera l'objet de le seconde partie de cette thèse.

1.1.2 Le Problème de Valeurs Propres non Symétrique

Depuis plus de trente ans, la méthode QR reste la méthode de choix pour calculer les valeurs propres d'une matrice dense non symétrique de taille modérée.

La première partie de cette thèse est une étude détaillée de cette méthode .

Quelques recherches ont été menées sur les bornes d'erreurs et les estimations du nombre de conditionnement pour la construction des éléments propres, dans le but de les inclure dans les bibliothèques d'algèbre linéaire [8]. L'estimation de ces conditionnements nécessite un réordonnancement de la forme de Schur construite par QR de manière à faire apparaître dans le coin en haut à gauche de la matrice les valeurs propres souhaitées. La validité de ces estimateurs repose sur la fiabilité de l'algorithme de réordonnancement. Or, jusqu'à maintenant, les versions itératives [107] ou directes de cet algorithme étaient peu fiables surtout dans le cas de valeurs propres très proches. LAPACK utilise une version améliorée de l'algorithme direct, garantie stable [5].

Le problème non symétrique de décomposition spectrale est l'un des problèmes les plus difficiles à résoudre sur machines parallèles. Toutes les parallélisations de QR requièrent un parallélisme de grain fin et semblaient difficiles à mettre en oeuvre. Récemment, Henry et Van de Geijn [55] ont montré que l'algorithme QR sur matrices de Hessenberg peut être parallélisé sur machines à mémoire distribuée jusqu'à 100 processeurs.

Un certain nombre d'alternatives à l'algorithme QR ont été étudiées, notamment la réduction à une forme tridiagonale non symétrique suivie par des itérations LR. Cet algorithme requiert des transformations non unitaires qui peuvent être très mal conditionnées et détruire la stabilité [44].

La méthode de Jacobi non symétrique peut être implantée avec des transformations orthogonales mais au prix d'une convergence très lente (linéaire). La convergence quadratique est possible avec des transformations non unitaires mais avec toutes les instabilités que cela peut entraîner.

Des recherches sur les algorithmes divide and conquer pour le cas non symétrique ont également été entreprises. Comme pour l'algorithme QR, ils s'appliquent à une matrice préalablement réduite sous forme de Hessenberg. Contrairement au cas symétrique, les valeurs propres des deux sous-matrices issues de la perturbation de rang 1 n'encadrent pas les valeurs propres de la matrice initiale complexe. Jessup montre que toutes les propriétés intéressantes de l'algorithme divide and conquer de Cuppen sont perdues lorsque cet algorithme est adapté de manière naturelle au cas non symétrique [64]. Les sous problèmes résultant des modifications de rang 1 peuvent être très mal conditionnés alors que le problème initial peut être bien conditionné. Des difficultés de convergence apparaissent lors de la résolution de l'équation séculaire. Pour regrouper les deux sous problèmes, Dongarra et Sidani utilisent une méthode de Newton. Les éléments propres des sous matrices servent à construire le point de départ des itérations. Li, Zeng et Cong [76] utilise une méthode de continuation. Ces méthodes sont intéressantes par leur fort potentiel de parallélisme. Leur désavantage réside dans le fait que même si la matrice initiale est bien conditionnée, rien ne prouve que les matrices intermédiaires le soient.

Une autre approche appelée *spectrale divide and conquer* a également été étudiée. Soit $A \in \mathbb{R}^{n\times n}$, $Q_1 \in \mathbb{C}^{n\times p}$ dont les colonnes, orthogonales, engendrent un sous-espace invariant par A, de dimension p et Q_2 telle que $Q = [Q_1, Q_2]$ soit unitaire. Alors,

$$Q^*AQ = \begin{pmatrix} B_{11} & B_{12} \\ 0 & B_{22} \end{pmatrix},$$

et les valeurs propres de $B_{11} \in \mathbb{C}^{p\times p}$ sont les valeurs propres de A correspondant au sous-espace invariant engendré par Q_1. Ce processus est répété sur B_{11} et B_{22} jusqu'à ce que les valeurs propres soient trouvées. Comment construire Q ? La fonction *signe de matrice*, notée $\mathrm{sign}(\cdots)$, permet de construire la projection spectrale sur les sous-espaces invariants correspondant aux valeurs propres de A

avec une partie réelle positive, respectivement négative :

$$P^{\pm} = \frac{1}{2}\left(I + \operatorname{sign}(A)\right).$$

Un moyen de construire Q est d'utiliser une factorisation QR de $I + \operatorname{sign}(A)$ révélant le rang de P^{+}. La matrice B_{11} contiendra les valeurs propres de partie réelle positive et B_{22} celles de partie réelle négative. Le spectre est ainsi divisé suivant l'axe imaginaire. De manière plus générale, en utilisant la factorisation QR de la matrice $\operatorname{sign}(\alpha A + \beta I)$, on peut diviser le spectre le long d'autres lignes et cela permet de rester en arithmétique réelle lorsque A est réelle. Bai et Demmel [7] ont développé une routine basée sur cette fonction de matrice permettant de ne calculer que les valeurs propres et vecteurs propres de la région du plan complexe qui nous intéressent.

1.2 Présentation de la Thèse

Cette thèse est divisée en deux parties. L'une traite de la méthode QR pour le problème de valeurs propres non symétrique, l'autre de la méthode de Yau et Lu pour le problème de valeurs propres symétrique.

1.2.1 La Méthode QR

La méthode QR reste encore aujourd'hui la méthode à choisir pour le calcul des valeurs propres de matrices générales, denses, non symétriques. Depuis sa conception, cette méthode à été soumise de à nombreuses évolutions justifiant sa remarquable longévité.

Le premier chapitre est une présentation détaillée de la méthode, son histoire, ses évolutions, les résultats de convergence associés, l'intérêt du shift et les difficultés liées à la parallélisation.

Il est souvent dit que l'algorithme QR est inversement stable. Cependant, aucune preuve rigoureuse n'existe dans la littérature. Le second chapitre est une étude détaillée de la propagation des erreurs d'arrondis dans l'algorithme. Nous utilisons des notations modernes et donnons une borne pour la stabilité inverse de l'algorithme, adaptée aux versions actuellement utilisées.

L'algorithme QR est un algorithme itératif. Il construit une suite de matrices semblables convergeant vers une forme de Schur. Quand peut-on stopper les itérations ? Quand peut-on déflater ? Le troisième chapitre est une étude des critères d'arrêts et de déflation de la méthode QR. Nous proposons de nouveaux critères basés sur des considérations mathématiques. Nous montrons par exemple que les critères d'arrêt et de déflation doivent tenir compte de la partie triangulaire supérieure stricte de la matrice, de la taille des éléments sous-diagonaux et de la distance entre les éléments diagonaux.

1.2.2 La Méthode de Yau et Lu

La seconde partie porte sur l'étude et la parallélisation de la méthode de Yau et Lu. Le but de cette méthode est de réduire le problème symétrique de valeurs propres en une série de produits matrice-matrice ayant une grande efficacité sur calculateurs haute-performance.

Au chapitre 1, nous proposons une formulation plus claire, plus détaillée de la méthode proposée en [123]. Nous avons justifié quelques résultats, fait quelques corrections et illustré la dernière partie par un exemple simple.

Nous présentons notre algorithme et sa complexité au chapitre 2.

Les tests numériques présentés en [123] ne sont pas assez nombreux pour justifier d'une implantation parallèle. Le troisième chapitre est consacré à la validation numérique de la méthode. Nous nous sommes plus particulièrement attachés à tester la précision des résultats.

A notre connaissance, aucune recherche du coté parallélisme n'a encore été entreprise en ce qui concerne cette méthode. Le quatrième chapitre propose une parallélisation de la méthode. Les machines cibles sont les machines à mémoire distribuée. Nous présentons les premiers résultats obtenus sur la Paragon d'Intel.

1.3 Notations

Nous décrivons les principales notations utilisées dans cette thèse. Nous utilisons

les lettres capitales romanes et grecques	A, D, H, Δ, Q	*pour les matrices*
les lettres minuscules indicées	$a_{ij}, d_{i,j}, h_{i,j}, q_{i,j}$	*pour les coefficients des matrices,*
les lettres minuscules romanes	x, y, z, u, v	*pour les vecteurs,*
les lettres minuscules grecques	$\alpha, \beta, \gamma, \varepsilon$	*pour les scalaires réels ou complexes.*

Ces conventions de notations ont été introduites par Householder [60] et conseillées dans [56].

Nous noterons

$\mathrm{Sp}(A)$	le spectre de A,
$\mathrm{rg}(A)$	le rang de A,
$< x_1, \ldots, x_n >$	l'espace engendré par $x_1, \ldots, x_n$.
$\Delta = \mathrm{diag}\{\delta_1, \ldots, \delta_n\}$	la matrice diagonale d'éléments diagonaux δ_i, $i \leq n$,
$\mathrm{Ker}(A)$	le noyau de l'application linéaire A,
$\mathrm{Im}(A)$	l'espace image de l'application linéaire A.

Nous noterons $\hat{x}$ la quantité x construite en précision finie.

Première partie

L'algorithme QR

Chapitre 2

L'algorithme QR

L'algorithme QR, attribué a Francis et Kublanovskaya, date des années soixante et reste de nos jours l'algorithme à utiliser pour le calcul des valeurs propres de matrices denses, sans structures particulières, réelles ou complexes et de taille modérée.

Nous débutons ce chapitre par quelques notes historiques sur la méthode QR. Depuis sa conception, cet algorithme a subi de nombreuses modifications. Nous nous proposons de montrer l'évolution de l'algorithme QR, de sa forme de base à ses formes actuelles, tout en expliquant les motivations qui ont conduit à une telle évolution. Aucune preuve de convergence globale de l'algorithme QR n'existe. Nous rappelons les hypothèses sous lesquelles certains résultats de convergence ont été établis [73]. Cela nous permet de démontrer un théorème de convergence concernant les implémentations de QR utilisées dans la pratique. Tous ces résultats montrent que la convergence dépend de la stratégie de shiftage utilisée. Pour ces raisons, nous discutons, dans un dernier paragraphe, des avantages et inconvénients liés à l'utilisation des shifts.

2.1 Historique

En 1955, l'Allemand Rutishauser réalise un grand pas en avant dans le domaine du calcul d'éléments propres grâce à l'algorithme LR [98]. Cet algorithme consiste à réduire une matrice générale à une forme triangulaire par l'intermédiaire de transformations non unitaires. Dès 1959, Wilkinson utilise cet algorithme avec shift d'origine et l'applique principalement à des matrices de Hessenberg complexes.

La même année, Francis, thésard de C. Strachey est chargé de programmer cette méthode. Cet informaticien connaît l'existence d'autres types de factorisations. Remplacer la factorisation LR par une factorisation QR, rien de plus aisé pour lui [87]. Il développe ainsi l'algorithme QR, basé sur des transformations unitaires. A la même époque et de manière complètement indépendante, la Russe V.N. Kublanovskaya étudie ce même type de transformations [67]. Francis ne publie ses résultats qu'en 1961 [43]. Il y décrit aussi bien les techniques du double shift complexe conjugué que la prise en compte de petits éléments sous-diagonaux consécutifs (splitting). Il ne semble pas avoir poursuivi d'études sur le sujet.

Avant 1970, les preuves de convergence de l'algorithme QR utilisent largement les déterminants [60, 67, 85, 121]. Le caractère géométrique de l'itération QR est apprécié pour la première fois dans la thèse de Buurema [22] ainsi que dans les travaux de Parlett et Poole [89]. Ceux-ci utilisent la connexion entre la méthode QR et l'itération inverse pour donner une explication de la convergence rapide de l'algorithme QR avec shift. Plus tard, Watkins [114] adopte une autre approche en utilisant la connexion de l'algorithme QR et de l'itération de sous-espaces. Avec Elsner, ils élargissent les preuves de convergence à la classe des algorithmes GR [113].

2.2 Les Différentes Versions de QR

L'objectif de cette section est de montrer l'évolution de l'algorithme QR, de sa forme de base à sa forme actuelle tout en expliquant les motivations qui ont conduit à une telle évolution.

2.2.1 L'Algorithme QR de Base

Soit $A \in \mathbb{C}^{n \times n}$ une matrice dont on souhaite calculer les éléments propres. L'algorithme QR de base est défini comme suit :

$A_1 = A$

pour $k = 1, 2, \ldots$

$A_k = Q_k R_k \quad$ (factorisation QR)

$A_{k+1} \quad = R_k Q_k$

fin

Nous avons donc

$$A_{k+1} = Q_k^* A_k Q_k = Q_k^* Q_{k-1}^* \ldots Q_1 A Q_1 \ldots Q_{k-1} Q_k.$$

Sous certaines hypothèses (voir §2.3), la suite de matrices unitairement semblables A_k converge vers la forme de Schur de la matrice A, en révélant les valeurs propres approchées, par ordre de module décroissant, sur la diagonale. L'accumulation des transformations unitaires Q_k fournit une base de Schur approchée de A. Évidemment, l'algorithme QR de base est trop inefficace pour être un outil attractif. Deux raffinements suffisent à le rendre compétitif [121] :

1- Une réduction préliminaire de A sous forme de Hessenberg permet de rabaisser rapidement le coût de chaque étape QR. De plus, l'itération QR de base préserve la forme de Hessenberg.

2- L'utilisation des shifts permet dans certains cas de réduire le nombre d'itérations nécessaires pour atteindre la convergence (cf. §2.4).

2.2.2 L'Algorithme QR avec Shift

L'algorithme QR avec shift d'origine appliqué à une matrice de Hessenberg est décrit comme suit :

$H_1 = Q_0^* A Q_0$ (Réduction sous forme de Hessenberg)
pour $k = 1, 2, \ldots$
déterminer un scalaire μ_k
$H_k - \mu_k I = Q_k R_k$ (factorisation QR)
$H_{k+1} = R_k Q_k + \mu_k I$
fin

Les shifts μ_k ont pour but d'accélérer la convergence. Si $\mu \equiv \mu_k$ est fixé d'une itération à l'autre et si les valeurs propres sont ordonnées telles que

$$|\lambda_1 - \mu| > |\lambda_2 - \mu| > \cdots > |\lambda_n - \mu|,$$

alors le $i^{ème}$ élément de la sous-diagonale des matrices itérées H_k converge vers zéro linéairement avec le taux

$$\frac{|\lambda_{i+1} - \mu|}{|\lambda_i - \mu|}.$$

Si μ est plus proche de λ_n que des autres valeurs propres, la $(n-1)^{ème}$ entrée sous-diagonale converge vers zéro rapidement. Si $\mu = h_{nn}^{(k)}$ et si $h_{n,n-1}^{(k)}$ converge vers zéro, alors on peut montrer que la convergence est quadratique [49]. Au §2.4, nous montrerons les avantages et inconvénients liés à l'utilisation des shifts pour accélérer la convergence.

2.2.3 Itérations QR Explicites Multishiftées

Pour une matrice donnée A réduite sous forme de Hessenberg $H = Q_0^* A Q_0$, m étapes de l'algorithme QR explicite avec shifts $\mu_1, \ldots, \mu_m$ produisent les relations suivantes, où l'on a défini $H_1 = H$:

$$H_{m+1} = R_m Q_m + \mu_m I \tag{2.2.1}$$

$$
\begin{aligned}
&= Q_m^* H_m Q_m \\
&= Q_m^* Q_{m-1}^* \dots Q_1^* H_1 Q_1 \dots Q_{m-1} Q_m.
\end{aligned}
$$

Commençons par définir trois matrices que nous utiliserons par la suite :

$$
p_m(H) = (H - \mu_m I) \dots (H - \mu_1 I),
$$

$$
Q = Q_1 \dots Q_{m-1} Q_m, \quad R = R_m R_{m-1} \dots R_1. \tag{2.2.2}
$$

Remarquons que Q est unitaire et R triangulaire supérieure. Notons que

$$
\begin{aligned}
Q^* p_m(H) &= Q_m^* \dots Q_1^* (H - \mu_m I) Q_1 \dots Q_{m-1} Q_{m-1}^* \dots Q_1^* (H - \mu_{m-1} I) \\
&\qquad \dots (H - \mu_2 I) Q_1 Q_1^* (H - \mu_1 I) \qquad (2.2.3) \\
&= Q_m^* (H_m - \mu_m I) Q_{m-1}^* (H_{m-1} - \mu_{m-1} I) \dots Q_1^* (H_1 - \mu_1 I) \\
&= R_m R_{m-1} \dots R_1 \\
&= R.
\end{aligned}
$$

Cela montre que Q dans (2.2.2) fournit une factorisation QR de $p_m(H)$. Si $p_m(H)$ est régulière, c'est-à-dire si aucun des shifts n'est une valeur propre de H et si l'on impose à la diagonale de R d'être réelle et positive, Q et R sont définies de manière unique par la factorisation QR de $p_m(H)$. Le facteur unitaire Q dans la factorisation QR de $p_m(H)$ est essentiellement égal à la matrice unitaire Q définie dans (2.2.2). Nous pouvons travailler directement avec $Q = Q_1 \dots Q_{m-1} Q_m$ sans former les matrices unitaires individuelles Q_i et effectuer simultanément m itérations de l'algorithme QR. Notons enfin que H_{m+1} est encore une matrice de Hessenberg. Nous appellerons cette opération *itération QR explicite de multiplicité m.*

2.2.4 Itération QR Implicite Multishiftée

Si la matrice $p_m(H)$ est construite explicitement, l'algorithme devient très coûteux. L'algorithme QR implicite effectue la transformation $H_{m+1} = Q^* H Q$ de manière plus efficace en évitant la construction de $p_m(H)$.

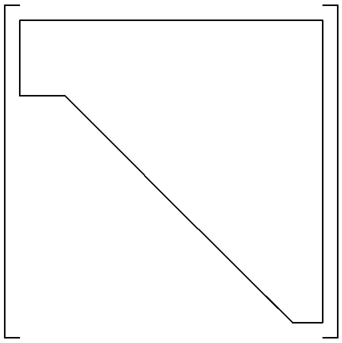

FIG. 2.2.1: Renflement introduit dans le coin en haut à gauche de la matrice de Hessenberg après la transformation $P_1^* H P_1$.

Soit P_1 une matrice de Householder choisie de façon à ce qu'elle annule tous les éléments mis à part le premier de la première colonne de $p_m(H)$ (voir §3.2 pour la construction d'une telle P_1) :

$$P_1^* p_m(H) e_1 = \alpha_1 e_1, \quad e_1 = (1, 0, \dots, 0)^*. \tag{2.2.4}$$

Formons la matrice $P_1^* H P_1$ qui peut ne plus être sous forme de Hessenberg. La matrice $P_1^* H P_1$ est alors remise sous la forme de Hessenberg supérieure G à l'aide de $n-2$ matrices de Householder $P_2, \dots, P_{n-1}$ chacune ayant l'élément en position $(1, 1)$ égal à 1. A la fin, nous obtenons

$$G = P^* H P, \quad P = P_1 P_2 \dots P_{n-1}. \tag{2.2.5}$$

La transformation $P_1^* H P_1$ crée un *renflement* de taille $m + 1$ en haut de la matrice (voir Figure 2.2.1). Le reste de l'itération QR implicite consiste à chasser le renflement à l'aide d'une réduction sous forme de Hessenberg standard (voir Figure 2.2.2). Comme les colonnes sont remises à jour une par une, les nouveaux éléments sous-diagonaux non nuls sont rajoutés dans le coin en bas à droite du renflement et celui-ci est chassé vers le coin en bas à droite de la matrice. En fait, le rôle du renflement est de transporter les shifts [118]. Watkins et Elsner [113], [112] parlent d'algorithme de poursuite.

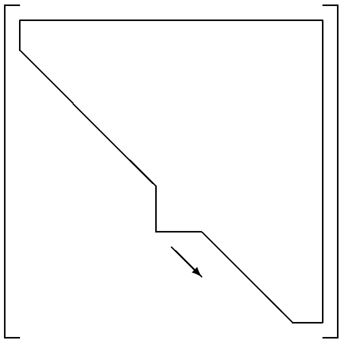

FIG. 2.2.2: Le renflement est chassé dans le coin en bas à droite de la matrice.

Nous suivons l'approche de Miminis et Paige [84] pour montrer que G en (2.2.5) et H_{m+1} en (2.2.1) sont essentiellement égales. D'abord, notons que d'après (2.2.4) et d'après la forme des P_i, nous avons

$$p_m(H)e_1 = P_1\alpha_1 e_1 = P\alpha_1 e_1,$$

et d'après (2.2.3),

$$p_m(H)e_1 = QRe_1,$$

si bien que les premières colonnes de P et Q sont identiques à un facteur multiplicatif près, de module unité.

Théorème 1 *(Théorème Q implicite).*
Soit $H \in \mathbb{C}^{n\times n}$. Soient $Q, P \in \mathbb{C}^{n\times n}$ unitaires telles que Q^*HQ et P^*HP sont des matrices de Hessenberg supérieures irréductibles. Si $Pe_1 = Qe_1$ alors $P = QD$ où $D = \operatorname{diag}(\pm 1, \ldots, \pm 1)$.

Preuve. Soit $K = Q^*HQ$, $G = P^*HP$ et $W = P^*Q = (w_1, \ldots, w_n)$. Alors, $WK = GW$ et pour $i = 2, \ldots, n$, nous avons

$$k_{i,i-1}w_i = Gw_{i-1} - \sum_{j=1}^{i-1} k_{j,i-1}w_j.$$

Puisque $w_1 = P^*Qe_1 = P^*Pe_1 = e_1$ et que G est Hessenberg supérieure, nous en concluons que W est triangulaire supérieure. Comme W est unitaire, nous avons $w_i = \pm e_i$ pour $i = 2, \ldots, n$. Mais $w_i = P^* q_i$ si bien que $p_i = \pm q_i$ pour $i = 1, \ldots, n-1$. $\square$

Le théorème Q implicite montre que H_{m+1} et G sont essentiellement égales :

$$\begin{aligned} G &= P^* H P \\ &= D^* Q^* H Q D \quad \text{avec } Q \text{ définie par (2.2.2)} \\ &= D^* H_{m+1} D. \end{aligned}$$

Il s'en suit que nous pouvons effectuer la transition de H à H_{m+1} en $O(n^2)$ flops si nous

- construisons $p_m(H)e_1$, la première colonne de

$$p_m(H) = (H - \mu_m I) \ldots (H - \mu_1 I),$$

- construisons la matrice de Householder P_1 telle que $P_1(p_m(H)e_1) = \alpha e_1$ et formons $P_1^* H P_1$,
- réduisons $P_1^* H P_1$ en une forme de Hessenberg supérieure à l'aide de $n-2$ matrices de Householder.

Ces opérations forment une *itération QR implicite de multiplicité m*. Du point de vue calculs, l'approche implicite est beaucoup moins coûteuse que si l'on avait formé explicitement $p_m(H)$. Les shifts sont utilisés pour construire la première colonne de $p_m(H)$. Cette opération nécessite $O(m^3)$ flops parce que H est Hessenberg supérieure et que seulement les $m+1$ premières composantes de $p_m(H)e_1$ sont non nulles. En général, $m \ll n$. De plus, cette approche permet d'utiliser des shifts complexes conjugués tout en continuant à travailler dans le champ des réels. Tous les algorithmes QR multishiftés diffèrent uniquement dans la définition du polynôme p_m. Ce polynôme est appelé *polynôme des shifts*.

2.3 Convergence de l'Algorithme QR

L'algorithme QR est un ensemble de n méthodes d'itération de sous-espaces emboîtés les uns dans les autres avec un changement de coordonnées à chaque itération. Cette remarque est la clé qui permet de comprendre pourquoi l'algorithme QR converge.

Dans ce paragraphe, nous nous proposons d'étudier la convergence de l'algorithme QR de base ou multishifté, appliqué à une matrice de Hessenberg irréductible $H \in \mathbb{C}^{n\times n}$. Nous commencerons par montrer la connexion entre QR et l'itération de sous-espaces. Cela nous permettra d'étudier la convergence des sous-espaces générés par QR vers les sous-espaces invariants de H.

2.3.1 Itérations de Sous-espaces et Algorithme QR

La connexion entre l'itération de sous-espaces et l'algorithme QR est connue depuis longtemps [121]. Cependant, elle n'a été utilisée que plus tard pour comprendre l'algorithme QR et donner des preuves de convergence [22, 89, 114]

Soit S_0 un sous-espace quelconque de $\mathbb{C}^n$ de dimension r et U une base orthonormale de S_0. Soit (p_i) une suite de polynômes dont le degré est inférieur à n et

$$\widetilde{p}_k = p_k \dots p_2 p_1, \tag{2.3.1}$$

pour tout k. La méthode d'itération de sous-espaces non stationnaire consiste à construire une suite de sous-espaces (S_k) de la manière suivante :

$U = Q_0 R_0$ (factorisation QR)

pour $k = 1, 2, \dots$, faire

$U_k = p_k(H) Q_{k-1} = Q_k R_k$ (factorisation QR)

fin

Si S_k est le sous-espace engendré par les colonnes de Q_k alors

$$S_k = p_k(H)S_{k-1} = \widetilde{p}_k(H)S_0. \tag{2.3.2}$$

Sous certaines conditions sur A et S_0, les sous-espaces S_k convergent vers un sous-espace invariant par A [23].

Maintenant, nous allons voir comment les algorithmes QR définis précédemment peuvent être interprétés comme des itérations de sous-espaces. De manière générale, la $k^{ème}$ étape d'un algorithme QR commence avec la factorisation QR

$$p_k(H_k) = Q_k R_k. \tag{2.3.3}$$

Nous supposerons $p_k(H_k)$ régulière. Soient $q_1, q_2 \ldots, q_n$ les colonnes de Q_k et $e_1, e_2, \ldots, e_n$ la base canonique de $\mathbb{C}^n$. Puisque R_k est triangulaire supérieure et non singulière, nous déduisons de (2.3.3) que les r premières colonnes de $p_k(H_k)$ engendrent le même sous-espace vectoriel que les r premières colonnes de Q_k, c'est-à-dire,

$$p_k(H_k) < e_1, e_2, \ldots, e_r > \; = \; < q_1, q_2, \ldots, q_r > \quad \text{pour } i = 1, \ldots, n.$$

Ainsi, $< q_1, q_2, \ldots, q_r >$ est le sous-espace obtenu après une itération de sous-espace appliqué à $< e_1, e_2, \ldots, e_r >$.
L'itération QR se termine en effectuant la transformation

$$H_{k+1} = Q_k^* H_k Q_k. \tag{2.3.4}$$

Cela revient à effectuer à un changement de coordonnées. Les vecteurs qui étaient représentés par $q_1, q_2, \ldots, q_r$ dans l'ancien système, sont maintenant représentés par $Q_k^* q_1, Q_k^* q_2, \ldots, Q_k^* q_r$ dans le nouveau système. Ces derniers vecteurs ne sont autres que $e_1, e_2, \ldots, e_r$. En résumé, chaque itération QR effectue une itération de sous-espace sur $< e_1, e_2, \ldots, e_r >$, ce qui construit $< q_1, q_2, \ldots, q_r >$. Un changement de coordonnées retransforme les vecteurs $< q_1, q_2, \ldots, q_r >$ en $< e_1, e_2, \ldots, e_r >$. Au lieu d'avoir une matrice fixée et une suite de sous-espaces, on a un sous-espace fixé et une suite de matrices.

Soit

$$\widetilde{Q}_k := Q_1 Q_2 \ldots Q_k \quad \text{et} \quad \widetilde{R}_k := R_k \ldots R_2 R_1.$$

Alors, de (2.3.4) et en posant $H_0 = H$, nous avons

$$H_{k+1} = \widetilde{Q}_k^* H_0 \widetilde{Q}_k \quad \text{pour tout} \quad k \geq 1.$$

Les matrices $\widetilde{Q}_k$ et $\widetilde{R}_k$ satisfont également l'égalité

$$\widetilde{p}_k(H_0) = \widetilde{Q}_k \widetilde{R}_k.$$

Cela montre que l'espace engendré par les k premières colonnes de $\widetilde{Q}_k$ est justement

$$\widetilde{p}_k(H_0) < e_1, \ldots, e_r > := S_k, \tag{2.3.5}$$

et cet espace est le résultat de k itérations de sous-espaces à partir de $S_0 = < e_1, \ldots, e_r >$. Si l'itération de sous-espace converge, $\widetilde{p}_k(H_0) < e_1, \ldots, e_r >$ est proche d'un sous-espace invariant par H et dans ce cas, $H_{k+1} = \widetilde{Q}_k^* H_0 \widetilde{Q}_k$ sera proche d'une forme triangulaire par blocs. Remarquons enfin que la suite des sous-espaces $\widetilde{p}_k(H_0) < e_1, \ldots, e_r >$ dépend uniquement des polynômes p_k. L'algorithme convergera plus ou moins vite suivant le choix des p_k, c'est-à-dire, le choix des shifts.

2.3.2 Convergence des Sous-espaces Générés par QR

Soit H une matrice de Hessenberg supérieure irréductible et $\lambda_1, \lambda_2, \ldots, \lambda_n$ ses valeurs propres. Soient M et N les sous-espaces invariants par H associés à $\{\lambda_1, \lambda_2, \ldots, \lambda_r\}$ et $\{\lambda_{r+1}, \ldots, \lambda_n\}$ respectivement.

Nous souhaitons étudier la convergence des sous-espaces S_k définis par (2.3.5) vers le sous-espace invariant M où $S_0 = < e_1, \ldots, e_r >$.

Soit U une base orthonormale de M et V une base orthonormale de N. Il existe deux matrices $\theta_M \in \mathbb{C}^{r\times r}$ et $\theta_N \in \mathbb{C}^{(n-r)\times(n-r)}$ telles que

$$HU = U\theta_M \qquad \text{et} \qquad HV = V\theta_N$$

avec $\mathrm{Sp}(\theta_M) = \{\lambda_1, \lambda_2, \ldots, \lambda_r\}$ et $\mathrm{Sp}(\theta_N) = \{\lambda_{r+1}, \ldots, \lambda_n\}$.
Soit P la projection spectrale sur M et U_0 une base de S_0. Alors U_0 admet la décomposition unique

$$U_0 = UF + VG \tag{2.3.6}$$

où F et G sont telles que $PU_0 = UF$ et $(I-P)U_0 = VG$.
Si l'on considère l'itération de sous-espace non stationnaire $U_k = \widetilde{p}_k(H)U_0$, alors, d'après (2.3.6), nous avons

$$\begin{aligned} U_k &= \widetilde{p}_k(H)UF + \widetilde{p}_k(H)VG \\ &= U\widetilde{p}_k(\theta_M)F + V\widetilde{p}_k(\theta_N)G. \end{aligned}$$

Si l'on suppose que, pour $1 \leq j \leq r$ et tout k, $\widetilde{p}_i(\lambda_j) \neq 0$, alors $\widetilde{p}_k(\theta_M)$ est une matrice inversible.
Remarquons que puisque H est une matrice de Hessenberg irréductible, nous avons $\mathrm{rg}(PU_0) = \mathrm{rg}(U)$. Or $PU_0 = UF$ donc F est inversible.
Ainsi,

$$U_kF^{-1}\widetilde{p}_k(\theta_M)^{-1} = U + V\widetilde{p}_k(\theta_N)GF^{-1}\widetilde{p}_k(\theta_M)^{-1}. \tag{2.3.7}$$

Pour montrer que la suite de sous-espaces S_k converge vers le sous-espace invariant M, il nous suffira de montrer qu'il existe une matrice régulière F_k d'ordre r telle que

$$\lim_{k\to\infty} \|U - U_kF_k\|_2 = 0.$$

Théorème 2 *Soit H une matrice de Hessenberg supérieure irréductible et diagonalisable. Soit M, le sous-espace invariant de dimension r associé aux valeurs* *propres* $\lambda_1, \dots, \lambda_r$. *Soit* $\widetilde{p}_k$ *une suite de polynômes telle que pour tout* k *et tout* j *vérifiant* $1 \le j \le r$ *on a* $\widetilde{p}_k(\lambda_j) \neq 0$. *Posons*

$$r_k = \frac{\max\limits_{r+1 \le j \le n} |\widetilde{p}_k(\lambda_j)|}{\min\limits_{1 \le j \le r} |\widetilde{p}_k(\lambda_j)|}. \tag{2.3.8}$$

Si la suite (r_k) *converge vers* 0, *alors la suite des sous-espaces* $S_k = \widetilde{p}_k(H) < e_1, \dots, e_r >$ *converge vers* M.

Preuve. D'après (2.3.7), nous avons

$$\begin{aligned} \|U - U_k F^{-1}\widetilde{p}_k(\theta_M)^{-1}\|_2 &= \|V\widetilde{p}_k(\theta_N)GF^{-1}\widetilde{p}(\theta_M)^{-1}\|_2 \\ &\le \|GF^{-1}\|\|\widetilde{p}_k(\theta_N)\|_2\|\widetilde{p}_k(\theta_M)^{-1}\|_2. \end{aligned}$$

Si H est diagonalisable, il existe $X = (X_M, X_N)$ régulière telle que

$$\theta_M = X_M^{-1} D_M X_M \qquad \text{et } \theta_n = X_N^{-1} D_n X_N.$$

Ainsi,

$$\|\widetilde{p}_k(\theta_N)\|_2 \le \kappa(X_N) \max_{r+1 \le j \le n} |\widetilde{p}_k(\lambda_j)|$$

et

$$\|\widetilde{p}_k(\theta_M)^{-1}\|_2 \le \kappa(X_M) \min_{1 \le k \le r} |\widetilde{p}_k(\lambda_j)|^{-1}.$$

Par conséquent,

$$\|U - U_k F^{-1}\widetilde{p}_k(\theta)^{-1}\|_2 \le \|GF^{-1}\|_2 \kappa(X)^2 \; r_k,$$

et, il existe une matrice $F_k = F^{-1}\widetilde{p}_k(\theta)^{-1}$ régulière d'ordre r telle que, si $r_k \to 0$, on a $\|U - U_k F_k\|_2 \to 0$ lorsque $k \to \infty$.
(Si H est normale, X peut être choisie telle que $\kappa(X) = 1$, sinon, $\kappa(X) > 1$ et peut devenir très grand si H approche une matrice défective). □

La plupart des preuves de convergence des sous-espaces générés par l'algorithme QR utilise l'hypothèse suivante :

$$< e_1, \dots, e_r > \cap < v_{r+1}, \dots, v_n > \; = \{0\}, \quad 1 \le r \le n-1, \tag{2.3.9}$$

où les v_i, $i = r+1, \ldots, n$ engendrent le sous-espace invariant N. Nous préférons utiliser l'hypothèse

$$H \text{ est une matrice de Hessenberg irréductible,} \tag{2.3.10}$$

Même si elle est plus restrictive et implique (2.3.9). Elle permet en effet d'obtenir un résultat clair et adapté aux situations rencontrées dans la pratique. Si H n'est pas irréductible, le problème de valeurs propres est réduit en plusieurs problèmes de valeurs propres de taille plus petite mettant en jeu des matrices de Hessenberg irréductibles.

L'hypothèse "H diagonalisable" dans le théorème, n'est pas restrictive. Toute matrice non diagonalisable est arbitrairement proche d'une matrice diagonalisable. En pratique, les matrices non-diagonalisables se comportent dans la plupart des cas comme des matrices diagonalisables.

Le théorème suivant est un résultat de convergence global de l'algorithme QR vers une forme triangulaire par blocs.

Théorème 3 *Plaçons-nous sous les mêmes hypothèses que celles du Théorème 2. Considérons la suite des itérés H_k de l'algorithme QR appliqué initialement à $H_0 = H$. Si la suite des sous-espaces générés par QR converge, les matrices H_k tendent vers une forme triangulaire par blocs, et il existe une constante C telle que*

$$|h_{r+1,r}^{(k+1)}| \leq C\|H\|_2 r_k \quad \text{pour tout } k,$$

où r_k est défini en (2.3.8).

Preuve. Écrivons

$$H_{k+1} = \begin{pmatrix} H_{11}^{(k+1)} & H_{12}^{(k+1)} \\ H_{21}^{(k+1)} & H_{22}^{(k+1)} \end{pmatrix}$$

où $H_{11}^{(k+1)} \in \mathbb{C}^{r\times r}$ et considérons la partition $\widetilde{Q}_k = [\widetilde{Q}_1^{(k)}, \widetilde{Q}_2^{(k)}]$ où $\widetilde{Q}_1^{(k)} \in \mathbb{C}^{n\times r}$. D'après le théorème précédent, il existe une base orthonormale $\widetilde{U}_k$ du sous-espace

M, invariant par H, et une constante C dépendant de M telles que

$$\|\widetilde{Q}_1^{(k)} - \widetilde{U}_k\|_2 \leq Cr_k.$$

Nous avons un résultat analogue pour le supplémentaire orthogonal $S_k^\perp$ de S_k dans $\mathbb{C}^n$. Il existe une base orthonormale $\widetilde{V}_k$ de N telle que,

$$\|\widetilde{Q}_2^{(k)} - \widetilde{V}_k\|_2 \leq Cr_k.$$

De $H_{k+1} = \widetilde{Q}^{(k)*} H \widetilde{Q}^{(k)}$ nous tirons

$$\begin{aligned} H_{21}^{(k+1)} &= \widetilde{Q}_2^{(k)*} H \widetilde{Q}_1^{(k)} \\ &= (\widetilde{Q}_2^{(k)} - \widetilde{V}_k)^* H \widetilde{Q}_1^{(k)} + \widetilde{V}_k^* H (\widetilde{Q}_1^{(k)} - \widetilde{U}_k) + \widetilde{V}_k^* H \widetilde{U}_k. \end{aligned}$$

Le sous-espace M est invariant par H donc $\widetilde{V}_k^* H \widetilde{U}_k = 0$. Nous avons

$$\begin{aligned} |h_{r+1,r}^{(k+1)}| &= \|H_{21}^{(k+1)}\|_2 \\ &\leq \|\widetilde{Q}_2^{(k)} - \widetilde{V}_k\|_2 \|H\|_2 \|\widetilde{Q}_1^{(k)}\|_2 + \|\widetilde{V}_k^*\|_2 \|H\|_2 \|\widetilde{Q}_1^{(k)} - \widetilde{U}_k\|_2 \\ &\leq 2C\|H\|_2 r_k. \end{aligned}$$

Si la suite des sous-espaces générés par QR converge, $|h_{r+1,r}^{(k+1)}| \to 0$ et les matrices itérées H_k tendent vers une forme triangulaire par blocs. $\square$

Si l'on choisit $p_k(H_k) = H_k$ pour tout k, nous aurons l'itération QR de base et $\widetilde{p}_k(H) = H^k$ où $\widetilde{p}_k$ est défini en (2.3.1) Si l'on suppose les valeurs propres de H sont de modules distincts et ordonnées telles que

$$|\lambda_1| \geq \ldots \geq |\lambda_r| > |\lambda_{r+1}| \geq |\lambda_n|,$$

alors

$$\rho = \frac{\max\limits_{r+1 \leq j \leq n} |\lambda_j|}{\min\limits_{1 \leq j \leq r} |\lambda_j|} = \frac{|\lambda_{r+1}|}{|\lambda_r|} < 1.$$

Ainsi, $r_k = \rho^k$ et $S_k \to M$ linéairement avec ρ comme taux de convergence.

Corollaire 1 *L'algorithme QR de base appliqué à une matrice de Hessenberg supérieure irréductible et dont les valeurs propres sont à modules séparés, converge. Les valeurs propres sont obtenues par ordre de module décroissant.*

Les théorèmes que nous venons de donner sont des théorèmes de convergence globale. Ils supposent que QR avec une stratégie de shiftage donnée, converge.

2.4 QR et le Shift

L'objectif de cette section est d'étudier les bénéfices et inconvénients du shift dans l'algorithme QR.

2.4.1 Quelques Stratégies

Le choix des polynômes p_k est appelé *stratégie de shiftage.* On utilise le mot shift car si le degré m de p_k vaut 1, on a une translation ou shift. Parmi les stratégies les plus utilisées, on trouve :

– **stratégie du shift de Francis généralisé**
  p_k est le polynôme caractéristique de la sous-matrice principale inférieure de taille $m \times m$ de H_k. C'est la stratégie la plus utilisée [3, 6, 39]. Si $m = 2$, nous retrouvons l'algorithme de Francis avec double shift.
– **stratégie du shift de Wilkinson généralisé**
  $p_k = (x - \mu^{(k)})^m$ où $\mu^{(k)}$ est la valeur propre de la sous-matrice principale inférieure de taille $m \times m$ de H_k la plus proche de $h_{nn}^{(k)}$. C'est une généralisation du shift de Wilkinson.
– **stratégie du shift de Rayleigh généralisé**
  $p_k = (x - \mu_1^{(k)}) \ldots (x - \mu_m^{(k)})$. Il y a deux possibilités :
  – les $\mu_l^{(k)}$, $l = 1, \ldots, m$ sont les éléments diagonaux de la sous-matrice

principale inférieure de taille $m \times m$ de H_k, ou bien
- $\mu_1^{(k)} = \cdots = \mu_m^{(k)} = h_{nn}^{(k)}$.

C'est une généralisation du shift de Rayleigh.

- **stratégie du shift de Francis réutilisé**
$p_k = \left[(x - \mu_1^{(k)})(x - \mu_2^{(k)})\right]^{m/2}$ où $\mu_1^{(k)}, \mu_2^{(k)}$ sont les valeurs propres de la sous-matrice principale inférieure de taille 2×2 de H_k. Cela revient à réutiliser $m/2$ fois le shift de Francis. Van de Geijn montre que, lorsque l'algorithme QR converge, cette stratégie a un taux de convergence inférieur à celui de la stratégie du shift de Francis généralisé [109].

Les shifts utilisés dans les stratégies proposées ci-dessus se veulent être des approximations des valeurs propres. Ce choix est justifié dans les paragraphes suivants et au théorème 4.

2.4.2 Pourquoi Utiliser les Shifts ?

Il y a plusieurs motivations derrière l'utilisation du shift.

Accélération de la Convergence

Les Théorèmes 2 et 3 montrent que la convergence de l'algorithme QR est gouverné par le taux

$$r_k = \frac{\max\limits_{r+1 \le j \le n} |\widetilde{p}_k(\lambda_j)|}{\min\limits_{1 \le j \le r} |\widetilde{p}_k(\lambda_j)|}.$$

Convenablement choisis, les shifts peuvent accélérer la convergence. Cependant, personne n'a encore été capable d'établir une stratégie de shiftage qui soit garantie de converger pour toutes les matrices et qui converge rapidement.

En [71], nous nous sommes intéressés au comportement du taux de convergence en fonction des shifts (simples et doubles) pour des matrices 3×3 dans le cas élémentaire d'un QR stationnaire ($p_k = p$ pour tout k). Nous voulions, entre

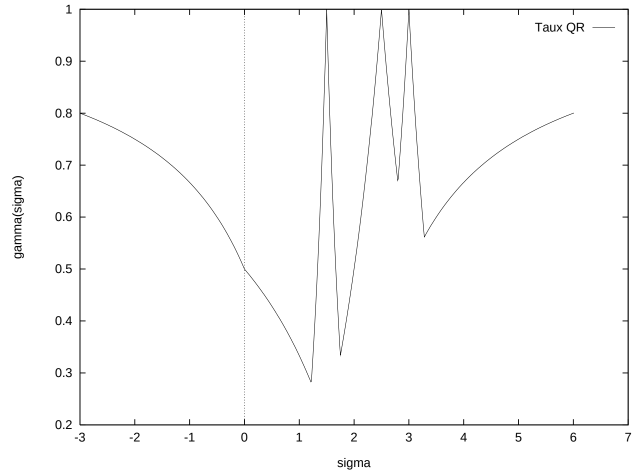

FIG. 2.4.1: Graphe de $\rho(\sigma)$ pour $\lambda_1 = 1.0$, $\lambda_2 = 2.0$, $\lambda_3 = 4.0$.

autre, mettre en évidence l'intérêt du shift, localiser le shift optimum, étudier sa position relative. La Figure 2.4.1 représente l'évolution du taux de convergence $\rho(\mu)$ en fonction du shift μ où

$$\rho(\mu) = \max_{1 \leq i < 3} \frac{|p(\lambda_{i+1})|}{|p(\lambda_i)|}$$

avec $p(x) = x - \mu$ et les λ_i ordonnés tels que

$$|p(\lambda_1)| \geq |p(\lambda_2)| \geq |p(\lambda_3)|.$$

La fonction $\rho(\mu)$ possède des minima locaux en des points non dérivables. Si $\mathrm{Sp}(H) = \{1, 2, 4\}$, le minimum est atteint en $\mu^* = 1.21$ et vaut $\rho(\mu^*) = 0.28$ alors que $\rho(0) = 0.5$, c'est-à-dire lorsqu'on n'utilise pas de shift. Remarquons que μ^* est le minimum le plus proche de la plus petite valeur propre en module. Cela n'est pas toujours vérifié. Ce qu'il faut remarquer, c'est que nous passons très rapidement d'un bon taux de convergence à un taux de convergence non optimal voir même stationnaire. Pour l'exemple précédent, $\rho(\mu) = 1$ en $\mu = \frac{3}{2}, \frac{5}{2}, 3$ (voir Figure 2.4.1).

Nous sommes arrivés aux mêmes conclusions dans le cas de matrices à spectre complexe et dans le cas du double shift. Même dans des cas très simples, il semble très difficile de localiser les shifts optimaux et de proposer une stratégie.

De leur coté, Batterson et Smillie [14] se sont fixés une stratégie. Ils ont étudié la translation d'origine et ont montré qu'elle se comportait de manière chaotique. Batterson a exhibé une suite de matrices pour lesquelles l'algorithme de Francis avec double shift n'est pas capable d'approcher une solution [12].

A priori, il n'existe pas encore de stratégie miracle ! Cependant, Watkins et Elsner ont montré que dans le cas où l'algorithme converge, l'utilisation du shift de Francis généralisé conduit à une convergence quadratique [113, Th.6.2].

Déflation

Le but d'une stratégie de shiftage est de construire des itérées qui déflatent, c'est-à-dire qui permettent de réduire la taille du problème, en utilisant le moins d'itérations possible.

Théorème 4 *Soit $p_{m,k}$ le polynôme des shifts, de degré m de l'itération k :*

$$p_{m,k}(H_k) = (H_k - \mu_1)\dots(H_k - \mu_m).$$

Supposons que s des m shifts soient des valeurs propres de H_k. La matrice $p_{m,k}(H_k)$ est de rang $n-s$ et l'itérée suivante $H_{k+1} = \widetilde{Q}_k^* H_k \widetilde{Q}_k$ à la forme

$$\begin{pmatrix} H_{11}^{(k)} & H_{12}^{(k)} \\ 0 & H_{22}^{(k)} \end{pmatrix}.$$

Les valeurs propres de $H_{22}^{(k)} \in \mathbb{R}^{s\times s}$ sont les s shifts qui correspondent aux valeurs propres de H_k.

Preuve. Puisque H_k est Hessenberg irréductible, la matrice H_k a seulement un bloc de Jordan associé à chaque valeur propre. Soit $J = X^{-1}H_kX$, $J = \mathrm{diag}\,\{J_1, J_2, \dots, J_r\}$ la décomposition sous forme de Jordan de H_k. Puisque les blocs correspondent à des valeurs propres distinctes, chaque shift peut être une valeur propre d'au plus un bloc de Jordan. Soit λ_i, $i = 1, \dots, r$, les valeurs propres associées au bloc J_i et n_i la dimension de ce bloc. Soit ν_i le nombre de shifts égaux à λ_i et $s_i = \min\{\nu_i, n_i\}$. Alors, $s = \sum_{i=1}^{r} s_i$. Pour chaque i, considérons

$$p_{m,k}(J_i) = (J_i - \mu_1)\dots(J_i - \mu_m).$$

Le facteur $J_i - \mu_l$ est régulier si et seulement si $\mu_l \neq \lambda_i$. Si $\mu_l = \lambda_i$, alors $J_i - \mu_l = N$ où N est la matrice nilpotente avec des 1 sur la sur-diagonale et des zéros partout ailleurs. Puisque ν_i des facteurs sont égaux à N, $p_{m,k}(J_i)$ ła forme $MN^{\nu_i} = MN^{s_i}$ où M est régulière. Donc,

$$\dim\left(\mathrm{Ker}(p_{m,k}(J_i))\right) = s_i \quad \text{et} \quad \dim\left(\mathrm{Ker}(p_{m,k}(J))\right) = \sum_{i=1}^{r} s_i = s.$$

Ainsi,

$$\mathrm{rg}\left(p_{m,k}(H_k)\right) = \mathrm{rg}\left(p_{m,k}(J)\right) = n - s.$$

Soit $R = Q^* p_{m,k}(H_k)$ défini comme en (2.2.3). La matrice R est de rang s et à la forme

$$R = \begin{pmatrix} R_{11} & R_{12} \\ 0 & 0 \end{pmatrix} \quad \text{où } R_{11} \in \mathbb{R}^{s \times s}.$$

Nous en déduisons la forme par blocs de H_{k+1}.

Supposons que les valeurs propres de H_k soient distinctes. Les valeurs propres de $H_{11}^{(k)}$ sont les valeurs propres de $H_k|_{\mathrm{Im}(p_{m,k}(H_k))}$. L'image de $p_{m,k}(H_k)$ est le sous-espace invariant par H dont les valeurs propres associées ne sont pas parmi les shifts. Ainsi, les valeurs propres de $H_{22}^{(k)}$ sont les valeurs propres restantes de H, c'est-à-dire, les valeurs propres correspondant aux shifts. Si A a des valeurs propres multiples, $H_{11}^{(k)}$ et $H_{22}^{(k)}$ peuvent avoir des valeurs propres communes. Si λ_i est une valeur propre de J de multiplicité s_i, alors λ_i est une valeur propre de $J|_{\mathrm{Im}(p_{m,k}(J))}$ de multiplicité $n_i - s_i$. Ainsi, λ_i est une valeur propre de $H_{22}^{(k)}$ de multiplicité s_i. $\square$

Le théorème précédent montre que choisir un polynôme tel que $p_{m,k}(H_k)$ soit singulière est souhaitable puisqu'il permet au problème de déflater après une itération. La matrice $H_{11}^{(k)}$ de valeurs propres inconnues est séparée de $H_{22}^{(k)}$ de valeurs propres connues. Notons que dans ce cas, nous perdons la relation entre l'algorithme QR multishifté explicite et l'algorithme QR multishifté implicite (voir §2.2.4) puisque $H_{22}^{(k)}$ n'est pas complètement défini par l'algorithme QR implicite multishifté. En fait, cela n'est pas un problème. La matrice $H_{11}^{(k)}$ est la matrice que l'on aurait obtenue après $n - s$ étapes d'un QR explicite. Donc, sur cette matrice de valeurs propres inconnues, la relation implicite existe toujours et c'est sur ce bloc que les itérations QR continuent.

Implémentation par Blocs

L'algorithme QR repose sur des opérations vecteur-vecteur, c'est-à-dire, des opérations de niveau 1. Les opportunités de vectorisation dans un tel algorithme sont limitées. Grâce au multishift, l'algorithme QR peut être réorganisé de manière à faire apparaître des opérations matrice-vecteur ou matrice-matrice, lesquelles conduisent à des implémentations plus efficaces sur machines vectorielles ou parallèles [6], [38]. L'idée est de chasser un renflement de taille $m \times m$ plutôt qu'un renflement de taille 1×1 ou 2×2 comme dans les algorithmes avec shifts simples ou doubles. Pour chasser le renflement, les matrices de Householder sont regroupées et appliquées par blocs [6]. Dubrulle [38] propose d'utiliser la représentation WY de Bischof et Van Loan [19] pour obtenir un algorithme plus riche en opérations de niveau 3.

Pipelinage

Les shifts vont à l'encontre de la parallélisation. Puisque les données pour calculer les shifts sont parmi celles qui sont calculées à la fin de chaque itération

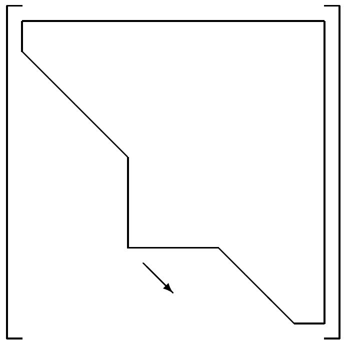

FIG. 2.4.2: Le renflement de taille $m = 3k$ est chassé dans le coin en bas à droite de la matrice.

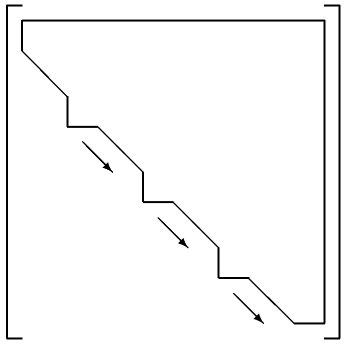

FIG. 2.4.3: QR pipeliné. Les 3 renflements de taille k sont chassés dans le coin droit de la matrice.

QR, toute nouvelle itération ne peut être débutée avant que la précédente ne soit terminée. Autrement dit, le shift empêche le pipelinage des itérations. Cependant, si l'on considère une implémentation par blocs et si le nombre de shifts $m = lk$ est un multiple de la taille k des blocs, alors l'itération QR de multiplicité m peut être pipelinée en l itérations QR de multiplicité k. Cette opération est illustrée en Figures 2.4.2 et 2.4.3. Du point de vue parallélisme, cela signifie que le processeur $p + 1$ n'a pas besoin d'attendre que le processeur p ait terminé sa tâche avant de commencer la sienne. Cela permet également de recouvrir les communications par des calculs. Il faudra cependant vider le pipeline toutes les l itérations [115], [116].

Henry et Van de Geijn [55] ont montré que l'algorithme QR standard (i.e. avec double shift de Francis) et un stockage "Hankel-wrapped" [110] n'est pas *scalable* dans la mesure où pour maintenir un niveau d'efficacité donné, les dimensions du problème doivent augmenter linéairement avec le nombre de processeurs. Cependant, des implémentations scalables de QR peuvent être obtenues si l'on utilise le multishift avec m qui augmente en fonction de la racine carrée du nombre de processeurs [54].

2.4.3 Les Shifts dans la Pratique

Depuis les 30 dernières années, la plupart des bibliothèques d'algèbre linéaire numériques fournissent l'algorithme QR de Francis avec double shift implicite [49, Algorithm 7.5.2] pour résoudre des problèmes de valeurs propres pour des matrices réelles denses . Les routines hqr de Eispack [104] et `xLAHQR` de Lapack [3] utilisent en plus du double shift de Francis, un shift exceptionnel qui est une sorte de perturbation du polynôme caractéristique [13], et cela, lorsque la convergence est trop lente. En pratique, cette stratégie fonctionne bien.

Récemment, Day [29] a exhibé une classe de matrices pour laquelle cet algorithme faillit ou converge trop lentement. Par exemple, la matrice de Demmel

$$H(\eta) = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & \eta & 0 \\ 0 & -\eta & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \qquad \eta < 1,$$

demande plus de 10^4 itérations en arithmétique à double précision et les valeurs propres construites ne sont pas précises. Pour ce type de matrices, Day propose une nouvelle stratégie de shiftage utilisant à la fois le shift de Francis et le shift de Wilkinson [29]. La nouvelle version de LAPACK devrait tenir compte de ses travaux.

La remarquable longévité de l'algorithme QR a été récemment étendue grâce aux versions mutishiftées. La routine `xGEEV` de la librairie LAPACK, utilise le multishift de Francis généralisé avec $1 \leq m \leq 6$. La valeur de m est en général pas trop grande car toute implémentation par blocs nécessite un surcoût de calculs. Il faut trouver un compromis entre le surcoût et l'efficacité due à l'utilisation des BLAS de niveau 2 ou 3. Le choix de LAPACK est $m = 6$.

Il devrait y avoir prochainement, une version de l'algorithme QR parallèle disponible dans Scalapack et utilisant le multishift pipeliné.

2.5 Conclusions

Nous avons montré l'évolution de l'algorithme QR, de sa version de base, à ses formes actuelles. Cette évolution est due à l'utilisation des shifts et l'introduction des BLAS de niveau 3 pour améliorer l'efficacité. Aucune preuve de convergence globale de l'algorithme QR n'a été produite à ce jour. Il ne semble pas y avoir de stratégie de shiftage miracle. Cependant, lorsque QR commence à converger, certaines stratégies comme la stratégie de Francis généralisée conduisent à une convergence quadratique ($m \geq 1$). Donc, pourquoi ne pas utiliser les shifts ? En pratique, le degré du polynôme des shifts n'est pas très grand. Le multishift semble faire apparaître des déflations prématurées. L'étude de la "forward stability" de QR est toujours un problème ouvert et son étude pourrait expliquer ces types d'instabilités [88], [117].

De nouveaux résultats de convergence pourraient être obtenus en utilisant des techniques provenant des systèmes dynamiques [13], [12].

Chapitre 3

Stabilité de l'algorithme QR

Depuis plus de 40 ans, l'algorithme QR s'avère être la méthode de choix pour le calcul des valeurs propres d'une matrice A, dense, réelle ou complexe, de taille modérée. Cette remarquable longévité s'est vue récemment prolongée par l'introduction du multishift. En arithmétique exacte complexe, les itérations QR construisent une suite de matrices unitairement semblables qui, lorsqu'il y a convergence, tendent vers une forme de Schur T de A.

Supposons que $\widehat{T}$ soit une approximation de T construite en arithmétique à précision finie. La qualité de $\widehat{T}$ peut se mesurer au moyen de l'erreur absolue ou relative, pour une norme appropriée :

$$E_{abs}(\widehat{T}) = \|\widehat{T} - T\| \ , \ E_{rel}(\widehat{T}) = \frac{\|\widehat{T} - T\|}{\|T\|}.$$

C'est ce que l'on appelle *l'erreur directe*.

Au lieu de nous intéresser à l'erreur directe, nous pouvons nous demander *pour quel ensemble de données, nous avons résolu de manière exacte notre problème ?* Pour cela, nous considérons que $\widehat{T}$ est la forme de Schur exacte d'une matrice perturbée $A + \Delta A$ et nous voulons borner $\|\Delta A\|$. La norme de ΔA est appelée *erreur inverse*. L'*analyse de l'erreur inverse* est le processus pour borner l'erreur inverse d'une solution calculée. Si l'erreur inverse n'est pas plus grande que les incertitudes inhérentes aux données, alors la solution peut difficilement être critiquée.

Au chapitre 1, nous avons vu que pour l'algorithme QR implicite multishifté, chaque nouvel itéré est obtenu par transformation unitaire. Cette transformation unitaire est une accumulation de matrices de Householder. Un moyen d'obtenir une borne pour l'erreur inverse est d'utiliser l'analyse d'erreurs d'arrondi développée par Wilkinson il y a trente ans [120, 121].

Le but de ce chapitre est de présenter une analyse d'erreurs d'arrondi pour l'algorithme QR en utilisant des notations modernes et un format unifié comme celui développé dans [57]. Nos résultats prennent en compte les versions multishiftées de QR.

Ce chapitre est organisé de la manière suivante :

Dans la section 3.1, nous rappelons le modèle standard des opérations en arithmétique à précision finie et donnons quelques notations et résultats utiles dans la suite.

La section 3.2 porte sur les matrices de Householder et présente une analyse d'erreurs d'arrondi pour les transformations de Householder.

Dans la section 3.3, nous donnons des bornes pour l'erreur inverse de l'algorithme QR multishifté implicite.

Les conclusions sont données dans la section 3.4.

3.1 Arithmétique à précision finie

Pour analyser les effets des erreurs d'arrondi sur un algorithme, nous avons besoin de faire des hypothèses sur la précision des opérations de base. Ayant défini un modèle, nous l'appliquerons à l'analyse d'erreur pour le produit scalaire, analyse dont nous aurons besoin ultérieurement. Cela nous permettra d'introduire la notation "$\gamma_{\mathbf{u}}(n)$" qui est de plus en plus utilisée et facilite les études d'analyse d'erreurs.

3.1.1 Le modèle standard

Nous utilisons le *modèle standard des opérations en arithmétique à nombres flottants* :

$$(3.1.1) \qquad fl(x \operatorname{op} y) = (x \operatorname{op} y)(1+\delta), \qquad |\delta| \leq \mathbf{u}, \quad \operatorname{op} = +, -, *, /,$$

où $\mathbf{u}$ est l'unité d'erreur d'arrondi. Par exemple, pour l'arithmétique IEEE qui est très largement utilisées, on a $\mathbf{u} = 2^{-53} \approx 1.1 \times 10^{-16}$ pour la double précision. Pour certaines machines qui ne possèdent pas de chiffre supplémentaire, dit *de garde,* pour stocker les résultats intermédiaires dans l'addition ou la soustraction, le modèle (3.1.1) doit être remplacée pour $+, -$ par

$$(3.1.2) \qquad fl(x \pm y) = x(1+\delta_1) \pm y(1+\delta_2), \qquad |\delta_i| \leq \mathbf{u}, \; i = 1, 2.$$

Tous les résultats que nous donnons par la suite restent valides pour le modèle (3.1.2) avec seulement quelques changements mineurs dans les termes constants.

En ce qui concerne les opérations de base en arithmétique complexe, les bornes pour les erreurs d'arrondi sont de la même forme que celles qu'on obtient avec le modèle standard (3.1.1) défini pour l'arithmétique réelle. La seule différence réside dans les constantes qui sont plus grandes [57, Lemma 3.5].

Nous travaillerons toujours en arithmétique réelle, mais nos bornes seront valides pour l'arithmétique complexe à condition que les constantes soient augmentées de manière appropriée.

3.1.2 Quelques notations et résultats utiles

Le lemme suivant est un outil de base pour notre analyse. C'est un moyen pratique de garder la trace des termes d'ordre plus élevé en $\mathbf{u}$, l'unité d'arrondi.

Lemme 1 *Si* $|\delta_i| \leq \mathbf{u}$, $\rho_i = \pm 1$ *pour* $i = 1, \ldots, n$ *et* $n\mathbf{u} < 1$ *alors il existe* θ_n *tel que*

$$\prod_{i=1}^{n}(1+\delta_i)^{\rho_i} = 1 + \theta_n \qquad \textit{et} \qquad |\theta_n| \leq \frac{n\mathbf{u}}{1-n\mathbf{u}}.$$

Preuve. La preuve se fait par récurrence. Supposons que le résultat soit vrai pour $n-1$. Alors, pour $\rho_n = 1$ nous avons

$$\prod_{i=1}^{n}(1+\delta_i)^{\rho_i} = (1+\delta_n)(1+\theta_{n-1}) = 1 + \theta_n$$

avec $\theta_n = \delta_n + (1+\delta_n)\theta_{n-1}$, et

$$\begin{aligned} |\theta_n| &\leq \mathbf{u} + (1+\mathbf{u})\frac{(n-1)\mathbf{u}}{1-(n-1)\mathbf{u}} \\ &= \frac{\mathbf{u}(1-(n-1)\mathbf{u}) + (1+\mathbf{u})(n-1)\mathbf{u}}{1-(n-1)\mathbf{u}} \\ &\leq \frac{n\mathbf{u}}{1-n\mathbf{u}}. \end{aligned}$$

Pour $\rho_n = -1$ nous avons

$$\prod_{i=1}^{n}(1+\delta_i)^{\rho_i} = (1+\delta_n)^{-1}(1+\theta_{n-1}) = 1 + \theta_n$$

avec $\theta_n = \dfrac{\theta_{n-1} - \delta_n}{1+\delta_n}$, et

$$\begin{aligned} |\theta_n| &\leq \frac{\frac{(n-1)\mathbf{u}}{1-(n-1)\mathbf{u}} + \mathbf{u}}{1-\mathbf{u}} \\ &= \frac{(n-1)\mathbf{u} + \mathbf{u}(1-(n-1)\mathbf{u})}{(1-(n-1)\mathbf{u})(1-\mathbf{u})} \\ &\leq \frac{n\mathbf{u}}{1-n\mathbf{u}}. \qquad \square \end{aligned}$$

Comme dans [57], pour des raisons pratiques, nous introduisons la constante

$$\boxed{\gamma_{\mathbf{u}}(n) = \frac{n\mathbf{u}}{1-n\mathbf{u}}},$$

et nous faisons l'hypothèse raisonnable que $n\mathbf{u} < 1$ ($n < 10^{16}$ si l'on utilise l'arithmétique IEEE en double précision).

Bien sûr, il existe d'autres manières d'écrire les bornes (voir par exemple Forsythe et Moler [42, p. 92]), mais les notations θ_n et $\gamma_{\mathbf{u}}(n)$ sont pratiques pour garder la trace des termes du second ordre et d'ordres plus élevés en $\mathbf{u}$. Le lemme suivant fournit les règles nécessaires pour manipuler les termes $1+\theta_n$ et $\gamma_{\mathbf{u}}(n)$.

Lemme 2 *Pour tout entier positif k, on considère θ_k et $\gamma_{\mathbf{u}}(k)$ définis comme précédemment. Les relations suivantes sont valables quelques soient les entiers positifs j et k :*

$$(1+\theta_k)(1+\theta_j) = 1+\theta_{k+j}, \tag{3.1.3}$$

$$\frac{1+\theta_k}{1+\theta_j} = \begin{cases} 1+\theta_{k+j} & j \leq k, \\ 1+\theta_{k+2j} & j > k, \end{cases} \tag{3.1.4}$$

$$k\gamma_{\mathbf{u}}(j) \leq \gamma_{\mathbf{u}}(jk), \tag{3.1.5}$$

$$\gamma_{\mathbf{u}}(k)+\mathbf{u} \leq \gamma_{\mathbf{u}}(k+1), \tag{3.1.6}$$

$$\gamma_{\mathbf{u}}(k)+\gamma_{\mathbf{u}}(j)+\gamma_{\mathbf{u}}(k)\gamma_{\mathbf{u}}(j) \leq \gamma_{\mathbf{u}}(k+j). \tag{3.1.7}$$

Preuve. Les relations (3.1.3), (3.1.5), et (3.1.6) sont triviales. Pour le résultat concernant la division, nous avons

$$\frac{1+\theta_k}{1+\theta_j} = 1+\theta \quad \text{avec} \quad \theta = \frac{\theta_k-\theta_j}{1+\theta_j}.$$

Alors,

$$|\theta| \leq \frac{\frac{k\mathbf{u}}{1-k\mathbf{u}}+\frac{j\mathbf{u}}{1-j\mathbf{u}}}{1-\frac{j\mathbf{u}}{1-j\mathbf{u}}} = \frac{(k+j)\mathbf{u}-2kj\mathbf{u}^2}{(1-2j\mathbf{u})(1-k\mathbf{u})}.$$

Si $j \leq k$,

$$\frac{(k+j)\mathbf{u}-2kj\mathbf{u}^2}{(1-2j\mathbf{u})(1-k\mathbf{u})} \leq \frac{(k+j)\mathbf{u}}{1-(k+j)\mathbf{u}} = \gamma_{\mathbf{u}}(k+j),$$

sinon,

$$\begin{aligned}\frac{(k+j)\mathbf{u}-2kj\mathbf{u}^2}{(1-2j\mathbf{u})(1-k\mathbf{u})} &= \frac{(k+j)\mathbf{u}-2kj\mathbf{u}^2}{1-(2j+k)\mathbf{u}+2jk\ u^2} \\ &\leq \frac{(k+2j)\mathbf{u}}{1-(k+2j)\mathbf{u}} = \gamma_{\mathbf{u}}(k+2j).\end{aligned}$$

Pour le dernier résultat,

$$\begin{aligned}
\gamma_{\mathbf{u}}(k)+\gamma_{\mathbf{u}}(j)+\gamma_{\mathbf{u}}(k)\gamma_{\mathbf{u}}(j) &= \frac{k\mathbf{u}(1-j\mathbf{u})+j\mathbf{u}(1-k\mathbf{u})+kj\mathbf{u}^2}{(1-k\mathbf{u})(1-j\mathbf{u})} \\
&= \frac{(k+j)\mathbf{u}-kj\mathbf{u}^2}{1-(k+j)\mathbf{u}+kj\mathbf{u}^2} \\
&\leq \frac{(k+j)\mathbf{u}}{1-(k+j)\mathbf{u}} = \gamma_{\mathbf{u}}(k+j). \qquad \square
\end{aligned}$$

Dans la suite, toutes les quantités calculées seront notées avec un chapeau.

3.1.3 Analyse d'erreurs pour le produit scalaire

Le prochain résultat fournit une borne d'erreur pour le produit scalaire effectué en arithmétique flottante.

Lemme 3 *Considérons le produit scalaire x^Ty où $x, y \in \mathbb{R}^n$. Alors,*

$$fl(x^Ty) = x^T(y+\Delta y), \qquad |\Delta y| \leq \gamma_{\mathbf{u}}(n)|y|.$$

Preuve. Utilisons le modèle standard (3.1.1). Nous avons

$$\widehat{s}_1 = fl(x_1y_1) = x_1y_1(1+\delta_1), \quad |\delta_1| \leq \mathbf{u},$$

et pour $k = 2, \ldots, n$,

$$\widehat{s}_k = fl(\widehat{s}_{k-1}+x_ky_k) = (\widehat{s}_{k-1}+x_ky_k(1+\delta_k))(1+\varepsilon_k), \quad |\delta_k| \leq \mathbf{u}, |\varepsilon_k| \leq \mathbf{u}.$$

Pour $k = n$, nous avons

$$fl(x^Ty) = x_1y_1(1+\delta_1)\prod_{j=2}^{n}(1+\varepsilon_j) + \sum_{k=2}^{n} x_ky_k(1+\delta_k)\prod_{j=k}^{n}(1+\varepsilon_j)$$

et en appliquant le lemme 2, nous avons

$$fl(x^Ty) = x_1y_1(1+\theta_n) + x_2y_2(1+\theta_n^{'}) + x_3y_3(1+\theta_{n-1}) + \cdots + x_ny_n(1+\theta_2),$$

où $|\theta_k| \leq \gamma_{\mathbf{u}}(k), k = 2, \ldots, n$.

Cela montre que le produit scalaire calculé est le produit scalaire exact pour un ensemble de données perturbées $x_1, x_2, \ldots, x_n, y_1(1+\theta_n), y_2(1+\theta_n^{'}), \ldots, y_n(1+\theta_2)$. Ainsi,

$$fl(x^T y) = x^T(y + \Delta y), \quad |\Delta y| \leq \gamma_{\mathbf{u}}(n)|y|. \qquad \square$$

3.2 Analyse d'erreur inverse pour les transformations de Householder

Les calculs avec les matrices de Householder sont très stables. Wilkinson [121, p. 152–160] montra que les vecteurs de Householder construits en arithmétique à précision finie sont très proches de ceux obtenus en arithmétique exacte et que la matrice construite lorsqu'on applique une matrice de Householder à une matrice donnée, est la matrice exacte d'une très petite perturbation de la matrice d'origine.

L'analyse de Wilkinson suppose que l'on utilise pour le produit scalaire, une accumulation en extra (double) précision [119, 122]. Nous ne ferons pas cette hypothèse dans notre analyse car de telles accumulations sont de nos jours rarement utilisées. Comme conséquence de cela, nos bornes auront un facteur n supplémentaire.

Parlett [86, Sections 6.5, 6.6] donne également une bonne discussion de la stabilité inverse d'une suite de transformations unitaires.

Dans cette section, nous définissons tout d'abord les matrices de Householder et donnons une analyse d'erreur pour leur construction en utilisant le format unifié proposé par Higham [57].

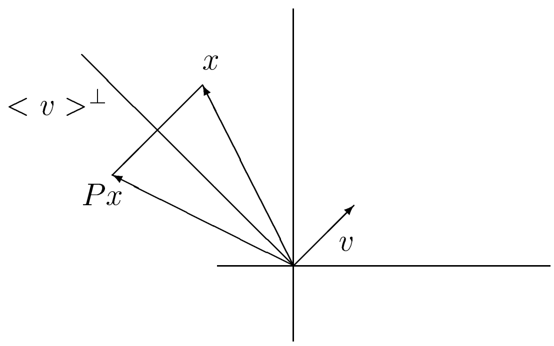

FIG. 3.2.1: P reflète x à travers l'hyperplan $< v >^{\perp}$.

3.2.1 Matrices de Householder élémentaires

Les matrices de la forme $I - 2ww^*$, $w \in \mathbb{C}^n$ ont été utilisées dans les mathématiques classiques depuis très longtemps. Il semble que leurs premières applications en analyse numérique ont été faites par Feller et Forsythe [41] pour faire de la déflation sur des matrices symétriques. Malheureusement, cette application particulière n'attira pas beaucoup l'attention. Leur popularité date de leur utilisation par Householder pour la triangularisation unitaire [59]. Ce dernier suggéra également de les utiliser pour la réduction d'une matrice symétrique à une forme tridiagonale. Il les appelait " *matrices élémentairement hermitiennes*".

Nous utiliserons les termes *matrices de Householder* pour désigner une matrice de la forme

$$P = I - \beta vv^*, \qquad \beta = \frac{2}{v^*v} \quad (v \neq 0).$$

Les matrices P possèdent les propriétés d'être hermitiennes et unitaires :

$$PP^* = P^2 = (I - \beta vv^*)^2 = I - 2\beta vv^* + \beta^2 vv^*vv^* = I \quad \text{car} \quad \beta = \frac{2}{v^*v}.$$

Soit $x \in \mathbb{C}^n$. Alors, $Px = x - \left(2\frac{v^*x}{v^*v}\right)v$. La matrice P est souvent appelée *réflecteur de Householder* car elle reflète x à travers l'hyperplan $< v >^{\perp}$. La figure 3.2.1 illustre cela en dimension 2.

Soient x, y deux vecteurs donnés de $\mathbb{C}^n$. Existe-t-il une matrice P telle que $Px = y$? Puisque $\|P\|_2 = 1$, alors $\|x\|_2 = \|y\|_2$. Nous avons

$$Px = y \equiv x - \left(2\frac{v^*x}{v^*v}\right) v = y.$$

Cette équation est de la forme $av = x - y$. Puisque P est indépendante du changement d'échelle de v, on peut prendre $a = 1$.
Avec $v = x - y$ nous avons

$v^*v = x^*x + y^*y - 2x^*y = -2v^*y$

et

$v^*x = x^*x - y^*x = -y^*v = \frac{1}{2}v^*v.$

Ainsi,

$$Px = x - \left(2\frac{v^*x}{v^*v}\right) v = x - v = y.$$

Notre intérêt est dans le choix $y = \pm\|x\|_2 e_1$. Cela fait des matrices de Householder un outil puissant pour introduire des zéros à l'intérieur des vecteurs. Considérons par exemple la mise sous forme de Hessenberg H d'une matrice $A \in \mathbb{C}^{n\times n}$:

$$U^*AU = H, \quad U^*U = I.$$

La transformation U est un produit de $n-2$ matrices de Householder $P_1, \ldots, P_{n-2}$. Le rôle de P_k est d'annuler la $k^{ème}$ colonne en dessous de la sous-diagonale. La figure 3.2.2 illustre le processus pour une matrice générique 5×5.

3.2.2 Matrices de Householder en nombres flottants

Nous restreignons notre étude à l'arithmétique réelle finie. En ce qui concerne le choix du signe dans la définition de y, $(y = \pm\|x\|_2 e_1)$, remarquons que si x est proche d'un multiple de e_1, alors $v = x - \text{sign}(x_1)\|x\|_2 e_1$ a une norme petite. Par conséquent, on peut penser que le facteur $\beta = \dfrac{2}{v^T v}$ aura une erreur relative très grande. Il est donc habituel de choisir $y = -\text{ sign }(x_1)\|x\|_2 e_1$.

$$A = \begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{bmatrix} \overset{P_1}{\to} \begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \end{bmatrix}$$

$$\overset{P_2}{\to} \begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ 0 & * & * & * & * \\ 0 & 0 & * & * & * \\ 0 & 0 & * & * & * \end{bmatrix} \overset{P_3}{\to} \begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ 0 & * & * & * & * \\ 0 & 0 & * & * & * \\ 0 & 0 & 0 & * & * \end{bmatrix}$$

FIG. 3.2.2: Réduction sous forme de Hessenberg. Le symbole $\overset{P_k}{\to}$ signifie que l'on effectue une pre et post multiplication de la matrice par P_k.

Lemme 4 *Soit $x \in \mathbb{R}^m$. Considérons la construction de $\beta \in \mathbb{R}$ et $v \in \mathbb{R}^m$ tels que $Px = -\operatorname{sign}(x_1)\|x\|_2 e_1$ où $P = I - \beta v v^T$ est une matrice de Householder avec $\beta = \dfrac{2}{v^T v}$:*

$$\begin{aligned} v &= x, \\ s &= \operatorname{sign}(x_1)\|x\|_2, \\ v_1 &= v_1 + s, \\ \beta &= \frac{1}{s v_1}. \end{aligned}$$

En arithmétique à nombres flottants, les $\widehat{\beta}$ et $\widehat{v}$ construits satisfont

$$\widehat{\beta} = \beta + \Delta\beta, \quad |\Delta\beta| \le \gamma_{\mathbf{u}}(2m+3)|\beta|, \tag{3.2.1}$$

$$\widehat{v} = v + \Delta v, \quad |\Delta v| \le \gamma_{\mathbf{u}}(m+1)|v|. \tag{3.2.2}$$

Preuve. Il est immédiat que les formules données pour v et β construisent bien les matrices de Householder. Le lemme 3 donne

$$fl(x^T x) = x^T x(1+\theta_m), \quad |\theta_m| \le \gamma_{\mathbf{u}}(m),$$

et ainsi

$$fl(\|x\|_2) = (x^T x)^{1/2}(1+\theta_m)^{1/2}(1+\delta), \quad |\delta| \le \mathbf{u}.$$

Alors, pour $m > 1$ nous pouvons montrer que

$$\widehat{s} = s(1 + \theta_m^{'}) \quad \text{où} \quad |\theta_m^{'}| \leq \gamma_{\mathbf{u}}(m).$$

Pour des raisons pratiques de notation, nous définissons $w = v_1 + s$. Ainsi,

$$\begin{aligned} \widehat{w} &= (v_1 + \widehat{s})(1 + \delta) \quad \text{avec} \quad |\delta| \leq \mathbf{u} \\ &= w(1 + \theta_{m+1}) \quad \text{où} \quad |\theta_{m+1}| \leq \gamma_{\mathbf{u}}(m + 1). \end{aligned}$$

Finalement, en utilisant les égalités (3.1.3) et (3.1.4) du lemme 2,

$$\begin{aligned} fl(\widehat{s}\widehat{w}) &= sw(1 + \theta_{2m+2}) \quad \text{où} \quad |\theta_{2m+2}| \leq \gamma_{\mathbf{u}}(2m + 2), \\ fl(1/(\widehat{s}\widehat{w})) &= \frac{1 + \delta}{sw(1 + \theta_{2m+2})} = \beta(1 + \theta_{4m+5}), \end{aligned}$$

où $|\theta_{4m+5}| \leq \gamma_{\mathbf{u}}(4m + 5)$. □

Lemme 5 *Soient $x, v \in \mathbb{R}^m$. Nous considérons la construction de $y = Px = x - \beta(v^T x)v$ où $P = I - \beta vv^T$ est une matrice de Householder. Supposons que la construction soit faite en utilisant des $\widehat{\beta}$ et $\widehat{v}$ construits et satisfaisant* (3.2.1) *et* (3.2.2). *Le vecteur $\widehat{y}$ construit satisfait*

$$\widehat{y} = P(x + \Delta x), \quad \|\Delta x\|_2 \leq \gamma_{\mathbf{u}}(cm)\|x\|_2,$$

où c est une petite constante entière.

Preuve. Nous donnons les grandes lignes de la preuve, en omettant les détails. D'après le lemme 3, nous avons

$$fl(\widehat{v}^T x) = \widehat{v}^T(x + \Delta x), \quad |\Delta x| \leq \gamma_{\mathbf{u}}(m)|x|,$$

d'où

$$fl(\widehat{v}^T x) = v^T x + w, \quad |w| \leq \gamma_{\mathbf{u}}(2m + 1)|v||v^T||x|,$$

et

$$\widehat{w} := fl(\widehat{\beta}\widehat{v}(\widehat{v}^T x)) = \beta v(v^T x) + \Delta w, \quad |\Delta w| \leq \gamma_{\mathbf{u}}(4m + 5)\beta|v||v^T||x|.$$

Alors

$$\widehat{y} = fl(x - \widehat{w}) = x - \beta v(v^T x) + \Delta y, \quad |\Delta y| \le \mathbf{u}|x|\gamma_{\mathbf{u}}(4m+6)\beta|v||v^T||x|.$$

Comme $\|I + \beta|v||v^T|\|_2 \le 3$, nous avons,

$$\|\Delta y\|_2 \le \gamma_{\mathbf{u}}(12m+18)\ \|x\|_2. \qquad \square$$

Nous pouvons également considérer les matrices de Householder P telles que $Px = +\ \mathrm{sign}\ (x_1)\|x\|_2 e_1$ (l'autre choix de signe) pourvu que les formules du lemme 4 soient modifiées de manière appropriée pour éviter les problèmes de compensation catastrophique [57, p. 383]. Le lemme 5 et nos résultats restent vrais pour cet autre choix de P.

D'un point de vue pratique, lorsqu'on implante la transformation

$$y = Px = -\ \mathrm{sign}\ (x_1)\|x\|_2 e_1,$$

on ne construit pas explicitement $y_2, \ldots, y_m$, mais on les pose égal à zéro. Il est facile de voir que les bornes du lemme 5 sont toujours valables dans cette situation. En fait, en forçant $y_2 = \cdots = y_m = 0$, on fait réduire l'erreur.

Nous allons maintenant considérer la pre et post-multiplication d'une matrice par une matrice de Householder approchée.

Lemme 6 *Soit* $A \in \mathbb{R}^{m\times m}$ *et soit* $P = I - \beta vv^T \in \mathbb{R}^{m\times m}$ *une matrice de Householder. Supposons que la construction de* PA *soit effectuée en utilisant les* $\widehat{\beta}$ *et* $\widehat{v}$ *satisfaisant* (3.2.1) *et* (3.2.2), *respectivement. Alors,*

$$fl(PA) = P(A + \Delta A), \qquad \|\Delta A\|_F \le \gamma_{\mathbf{u}}(cm), \tag{3.2.3}$$

$$fl(P^T AP) = P^T(A + \Delta A)P, \qquad \|\Delta A\|_F \le \gamma_{\mathbf{u}}(c' m). \tag{3.2.4}$$

Preuve. Soit a_j, la $j^{ème}$ colonne de A. D'après le lemme 5, nous avons

$$fl(Pa_j) = P(a_j + \Delta a_j), \qquad \|\Delta a_j\|_2 \le \gamma_{\mathbf{u}}(cm)\|a_j\|_2.$$

Ainsi, $\widehat{B} := fl(PA) = P(A + \Delta A)$, où

$$\|\Delta A\|_F \leq \sum_{j=1}^{m} \|\Delta a_j\|_2^2 \leq \gamma_{\mathbf{u}}(cm)^2 \sum_{j=1}^{m} \|a_j\|_2^2 = \gamma_{\mathbf{u}}(cm)^2 \|A\|_F^2,$$

c'est-à-dire, $\|\Delta A\|_F \leq \gamma_{\mathbf{u}}(cm)\|A\|_F$. De la même manière,

$$\begin{aligned} fl(\widehat{B}P) &= (\widehat{B} + \Delta\widehat{B})P \\ &= P^T(A + \Delta A + P\Delta\widehat{B})P \end{aligned}$$

avec $\|\Delta\widehat{B}\|_F \leq \gamma_{\mathbf{u}}(cm)\|\widehat{B}\|_F$ et

$$\begin{aligned} \|\Delta A + P\Delta\widehat{B}\|_F &\leq \|\Delta A\|_F + \gamma_{\mathbf{u}}(cm)\|\widehat{B}\|_F \\ &\leq (\gamma_{\mathbf{u}}(cm) + \gamma_{\mathbf{u}}(cm) + \gamma_{\mathbf{u}}(cm)^2)\|A\|_F \\ &\leq \gamma_{\mathbf{u}}(2cm)\|A\|_F, \end{aligned}$$

d'après (3.1.7). $\square$

Notons que le lemme 6 exprime le résultat construit comme une transformation orthogonale semblable exacte d'une matrice perturbée, c'est-à-dire, en termes de l'erreur inverse.

Nous mentionnons aussi que, pour la construction de PA, Higham [57, Lemma 18.3] montre que $fl(PA) = P(A + \Delta A)$ avec une borne par composantes de la forme

$$|\Delta A| \leq \gamma_{\mathbf{u}}(cm^2)G|A|, \quad \|G\|_F = 1.$$

Pour des transformations de similarité (pre et post-multiplication) P^TAP, cette borne doit être modifiée et prend la forme

$$|\Delta A| \leq \gamma_{\mathbf{u}}(cm^2)G_1|A|G_2, \quad \|G_1\|_F = 1, \quad \|G_2\|_F = 1.$$

Cette borne est semblable à celle donnée en (3.2.4). Ainsi, il n'y a pas d'avantage à utiliser l'analyse par composantes dans l'analyse d'erreur inverse de l'algorithme QR.

3.3 Stabilité inverse de l'algorithme QR

La version implicite de l'algorithme QR se doit d'être appliquée à une forme de Hessenberg. Nous commencerons par étudier les propriétés pour la propagation des erreurs d'arrondi lors de la réduction sous forme de Hessenberg puis l'erreur inverse associée à une itération QR multishiftée implicite. Ces résultats nous permettront de proposer une borne pour l'erreur inverse de l'algorithme QR.

3.3.1 Réduction sous forme de Hessenberg

Toute matrice réelle de taille $n \times n$ est unitairement semblable à une matrice de Hessenberg et la réduction peut se faire avec $n-2$ transformations de Householder.

Théorème 5 *Soit $A \in \mathbb{R}^{n\times n}$ et $H = U_0^T A U_0$ sa réduction sous forme de Hessenberg par des transformations de Householder. La forme de Hessenberg construite satisfait*

$$\widehat{H} = W^T(A+\Delta A)W \qquad avec \qquad \|\Delta A\|_F \leq \gamma_{\mathbf{u}}(cn^2),$$

où W est une matrice réelle orthogonale et c est une constante entière de petite taille.

Preuve. Soit P_k la matrice utilisée pour annuler la $k^{ème}$ colonne en dessous de la sous-diagonale. Le lemme 6 donne

$$\widehat{A}_1 = fl(P_1^T A P_1) = P_1^T(A+\Delta A_1)P_1, \qquad \|\Delta A_1\|_F \leq \gamma_{\mathbf{u}}(c(n-1))\|A\|_F.$$

Alors,

$$\begin{aligned} \widehat{A}_2 = fl(P_2^T \widehat{A}_1 P_2) &= P_2^T(\widehat{A}_1 + \Delta\widehat{A}_1)P_2 \text{ o`u } \|\Delta\widehat{A}_1\|_F \leq \gamma_{\mathbf{u}}(c(n-2))\|\widehat{A}_1\|_F, \\ &= P_2^T P_1^T (A+\Delta A_2)P_1P_2 \end{aligned}$$

où

$$\|\Delta A_2\|_F \leq \|\Delta A_1\|_F + \|\Delta\widehat{A}_1\|_F$$

$$
\begin{aligned}
&\leq \; [\gamma_{\mathbf{u}}(c(n-1)) + \gamma_{\mathbf{u}}(c(n-2))(1+\gamma_{\mathbf{u}}(c(n-1)))]\|A\|_F \\
&\leq \; \gamma_{\mathbf{u}}(c[(n-1)+(n-2)])\|A\|_F,
\end{aligned}
$$

en utilisant (3.1.7). De la même manière, nous trouvons

$$
\begin{aligned}
\widehat{H} &= fl(P_{n-2}^T \widehat{A}_{n-3} P_{n-2}) \\
&= P_{n-2}^T P_{n-3}^T \dots P_1^T (A + \Delta A) P_1 \dots P_{n-3} P_{n-2} \\
&:= W^T (A + \Delta A) W,
\end{aligned}
$$

où W est une matrice orthogonale et $\|\Delta A\|_F \leq \gamma_{\mathbf{u}}(q)$, avec

$$
q = c \sum_{k=1}^{n-2} (n-k) \leq c \, \frac{n(n-1)}{2} \leq c' n^2. \qquad \square
$$

Remarquons que, en général, nous n'aurons pas $W = U_0$ car les matrices P_k sont définies à partir de $\widehat{A}_{k-1}$ et non à partir de A_{k-1}.

3.3.2 Algorithme QR multishifté

Le prochain résultat fournit une borne pour l'erreur inverse d'une itération QR implicite multishiftée.

Avant chaque étape QR, nous mettons à zéro tous les éléments de la sous-diagonale qui satisfont un certain critère de déflation (voir chapitre 3). Nous recherchons ensuite le plus grand entier r et le plus grand entier s tels que

$$
H = \begin{array}{cc}
\begin{pmatrix} H_{11} & H_{12} & H_{13} \\ 0 & H_{22} & H_{23} \\ 0 & 0 & H_{33} \end{pmatrix} & \begin{array}{c} q \\ r \\ s \end{array} \\
\begin{array}{ccc} q & \quad r & \quad s \end{array} &
\end{array}, \qquad q + r + s = n,
$$

où H_{33} est triangulaire supérieure et H_{22} est Hessenberg irréductible. Une itération QR implicite est alors effectuée sur $H_{22} \in \mathbb{R}^{r \times r}$, produisant

$$
G = P^T H P, \qquad P^T P = I,
$$

où $P = P_1 \dots P_{r-2}$ est une accumulation de $r-2$ matrices de Householder.

Théorème 6 *Une itération de l'algorithme QR implicite de multiplicité m appliquée à une matrice de Hessenberg $H \in \mathbb{R}^{r\times r}$ construit la forme de Hessenberg $\widehat{G}$ telle que*

$$\widehat{G} = U^T(H+\Delta H)U, \qquad \|\Delta H\|_F \leq \gamma_{\mathbf{u}}(crm)\ \|H\|_F,$$

où U est une matrice rélle orthogonale.

Preuve. Nous donnons les grandes lignes en omettant les détails. Une itération de l'algorithme QR implicite s'effectue en deux étapes. Premièrement, nous construisons la matrice $P_1^T H P_1$ qui crée un renflement dans le coin gauche en haut de la matrice H. La matrice construite $\widehat{H}_1$ satisfait

$$\widehat{H}_1 = fl(P_1^T H P_1) = P_1(H+\Delta H_1)P_1, \quad \|\Delta H_1\|_F \leq \gamma_{\mathbf{u}}(cm)\|H\|_F.$$

La seconde étape consiste à chasser le renflement de taille $m+1$ à l'aide de $r-2$ matrices de Householder. La matrice obtenue satisfait

$$\widehat{G} := U^T(H+\Delta H)U \quad \|\Delta H\|_F \leq \gamma_{\mathbf{u}}(c^{'}rm)\|H\|_F.$$

La preuve est très semblable à celle du théorème 5. $\square$

Théorème 7 *L'algorithme QR implicite de multiplicité m appliqué à une matrice de Hessenberg $H \in \mathbb{R}^{n\times n}$, construit une matrice quasi-triangulaire supérieure $\widehat{T}$ telle que*

$$\widehat{T} = Q^T(H+\Delta H)Q, \quad Q^TQ = I$$

et

$$\|\Delta H\|_F \leq \sum_{j=1}^{p} \gamma_{\mathbf{u}}(cm \sum_{k=1}^{j} r_k)\ \|H\|_F$$

où p est le nombre d'itérations et r_k est la taille du problème à l'itération k.

Preuve. Soit $\widehat{H}_k$ la matrice construite au début de l'itération k. Supposons que tous les éléments $\widehat{h}^{(k)}_{i+1,i}$ qui sont mis à zéro à la $k^{ème}$ étape satisfont

$$|\widehat{h}^{(k)}_{i+1,i}| \leq c^{'}\mathbf{u}\|\widehat{H}_k\|_F.$$

Nous avons

$$\widehat{H}_{k+1} = Q_k^T(\widehat{H}_k + \varDelta H_k)Q_k$$

où $\varDelta H_k$ contient les éléments de la sous-diagonale éventuellement négligés et les erreurs d'arrondi faites durant les transformations semblables correspondant à la $k^{ème}$ itération QR implicite de multiplicité m et satisfaisant

$$\begin{aligned}\|\varDelta H_k\|_F &\leq \gamma_{\mathbf{u}}(cr_km)\ \|\widehat{H}_k\|_F + c^{'}\mathbf{u}\|\widehat{H}_k\|_F \\ &\leq \gamma_{\mathbf{u}}(c^{''}r_km)\ \|\widehat{H}_k\|_F.\end{aligned}$$

Mais,

$$\begin{aligned}\|\widehat{H}_k\|_F &\leq \|\widehat{H}_{k-1}\|_F + \|\varDelta H_{k-1}\|_F \\ &\leq (1+\gamma_{\mathbf{u}}(cr_{k-1}m))\|\widehat{H}_{k-1}\|_F \\ &\leq (1+\gamma_{\mathbf{u}}(cr_{k-1}m))(1+\gamma_{\mathbf{u}}(cr_{k-2}m))\|\widehat{H}_{k-2}\|_F \\ &\vdots \\ &\leq \prod_{i=1}^{k-1}(1+\gamma_{\mathbf{u}}(cr_im))\|H\|_F \\ &\leq (1+\gamma_{\mathbf{u}}(cm\sum_{i=1}^{k-1} r_i))\|H\|_F,\end{aligned}$$

donc, en utilisant (3.1.7),

$$\|\varDelta H_k\|_F \leq \gamma_{\mathbf{u}}(cm\sum_{i=1}^{k} r_i)\|H\|_F.$$

Finalement, si p est le nombre total d'itérations QR, la dernière matrice itérée satisfait

$$\widehat{H}_{p+1} \quad = \quad Q_p^T(\widehat{H}_p + \varDelta H_p)Q_p$$

$$
\begin{aligned}
&= Q_p^T(Q_{p-1}^T(\widehat{H}_{p-1} + \Delta H_{p-1})Q_{p-1} + \Delta H_p)Q_p \\
&= Q_p^T Q_{p-1}^T(\widehat{H}_{p-1} + \Delta H_{p-1} + Q_{p-1}\Delta H_p Q_{p-1}^T)Q_{p-1}Q_p \\
&\vdots \\
&= Q_p^T \dots Q_1^T \left(\widehat{H}_1 + \Delta H_1 + \sum_{i=2}^{p} \left(\left(\prod_{j=1}^{i-1} Q_j \right) \Delta H_i \left(\prod_{j=1}^{i-1} Q_j \right)^T \right) \right) Q_1 \dots Q_p \\
&:= Q^T(H + \Delta H)Q
\end{aligned}
$$

où

$$\|\Delta H\|_F \leq \sum_{j=1}^{p} \|\Delta H_j\|_F \leq \sum_{j=1}^{p} \gamma_{\mathbf{u}}(cm \sum_{k=1}^{j} r_k)\|H\|_F. \qquad \square$$

Nous voyons que la borne pour l'erreur inverse sur l'algorithme QR implicite multishifté dépend essentiellement de la taille des matrices à chaque itération et du nombre d'itérations QR.

Supposons que l'on utilise l'algorithme QR standard, c'est-à-dire avec double shift de Francis. Des observations empiriques ont montré que, en moyenne, seulement deux itérations sont nécessaires entre chaque déflation. Ainsi, avec $m = 2$ et $p \simeq 2n$, nous aurons certainement

$$\|\Delta H\|_F \leq \gamma_{\mathbf{u}}(c' n^3)\ \|H\|_F.$$

3.4 Conclusion

Nous avons donné des bornes pour l'erreur inverse associée à l'algorithme QR implicite multishifté. Elles montrent que l'algorithme est inversement stable. Ce résultat plaisant est dû à l'utilisation de transformations semblables basées sur des matrices de Householder. Bien sûr, nos résultats sont applicables seulement lorsque l'algorithme QR converge ; pour un exemple de non convergence de l'algorithme QR implicite multishifté, voir Day [29].

De nos jours, la plupart des librairies d'algèbre linéaire utilisent les transformations de Householder par groupe. Une forme est la représentation WY de Bischof et Van Laon. Cette sorte de représentation permet l'utilisation de routines de produit matrice-matrice. Nos résultats restent vrais avec l'utilisation de la technique WY. L'utilisation des routines BLAS de niveau 3 affecte la stabilité seulement au travers des constantes dans les bornes pour l'erreur inverse. Voir [57, 7, Section 18.4] pour les détails d'analyse d'erreur de la technique WY.

Notre analyse d'erreur produit une *borne* sur l'erreur inverse. Si l'on est intéressé seulement par $\widehat{T}$, alors l'erreur inverse peut être définie par

$$\mu_\nu(A) = \min_{Q^TQ=I} \|A - Q\widehat{T}Q^T\|_\nu, \quad \nu = 2, F.$$

Malheureusement, ce problème de Procrustes n'a pas de solution explicite connue et c'est une tâche non triviale que d'évaluer numériquement $\mu_\nu(A)$.

Cette analyse ne présente pas de borne d'erreur sur les valeurs propres construites. Pour cela, nous avons besoin de la théorie des perturbations. Pour construire les bornes d'erreur directe efficacement, nous avons besoin d'utiliser des techniques d'estimation du conditionnement [8].

3.5 Commentaires sur de Précédents Travaux

En [1], [2], nous avons adopté une approche différente pour étudier la stabilité de l'algorithme QR.

Nous supposons que l'algorithme construit une matrice quasi triangulaire supérieure $\widehat{T}$ telle que

$$\widehat{T} = \widehat{Q}^* H \widehat{Q} \tag{3.5.1}$$

où $\widehat{Q}$ est l'accumulation des transformations utilisées pour construire $\widehat{T}$ à partir de H. En général, $\widehat{Q}$ n'est pas unitaire. Soit $\widetilde{T}$ la partie triangulaire supérieure de $\widehat{T}$. Notre but est de montrer qu'il existe une matrice $\widetilde{H}$ et une matrice unitaire

$\widetilde{Q}$ telles que

$$\widetilde{T} = \widetilde{Q}^* \widetilde{H} \widetilde{Q}$$

c'est-à-dire $\widetilde{T}$ est la forme de Schur exacte de $\widetilde{H}$ et que si $\widehat{Q}^*\widehat{Q} - I$ vérifie

$$\|\widehat{Q}^*\widehat{Q} - I\|_2 \leq \gamma_{\mathbf{u}}(ckn)$$

où k est le nombre de transformations de Householder utilisées dans la construction de $\widehat{Q}$, alors

$$\|\widetilde{Q} - \widehat{Q}\|_2 \leq \gamma_{\mathbf{u}}(ckn) \text{ et } \|\widetilde{H} - H\|_F \leq \gamma_{\mathbf{u}}(ckn)\|H\|_F.$$

Pour cela, nous avons à résoudre un problème de minimisation appelé problème de Procrustes unitaire pour borner $\|\widetilde{Q} - \widehat{Q}\|_2$ et la théorie des perturbations pour borner $\|\widetilde{H} - H\|_F$.

L'écriture (3.5.1) est ambiguë. Si l'on considère que

$$\widehat{Q} = fl(Q_1 Q_2 \dots Q_k),$$

il n'est pas clair que $\widehat{T}$ puisse s'écrire sous la forme $\widehat{Q}^* H \widehat{Q}$. De plus, en pratique, la matrice $\widehat{Q}$ n'est pas toujours formée.

Nous préferons l'approche de ce chapitre qui ne passe que par des résultats sur la propagation des erreurs d'arrondi pour borner l'erreur inverse.

Chapitre 4

Un Nouveau Critère d'Arrêt pour l'Algorithme QR

4.1 Introduction

Au cours de ces trente dernières années, l'algorithme QR a subi de nombreuses modifications afin d'accélérer sa convergence, améliorer la précision des résultats et l'efficacité des calculs. Cependant, très peu de travaux ont été menés sur le critère d'arrêt de cet algorithme itératif [40].

En pratique, le test de convergence des éléments sous-diagonaux de la matrice itérée est couplé avec un critère de partitionnement qui amène à remplacer le problème d'origine par deux problèmes de taille plus petite. Si le coefficient $h_{i,i-1}$ est "suffisamment petit " pour un indice i, $2 \leq i \leq n$, alors $h_{i,i-1}$ est mis a zéro et la matrice en cours H est remplacée par

$$\begin{array}{cc} \begin{pmatrix} H_{11} & H_{12} \\ 0 & H_{22} \end{pmatrix} & \begin{array}{c} {\scriptstyle i-1} \\ {\scriptstyle n-i+1} \end{array} \\ \begin{array}{cc} {\scriptstyle i-1} & {\scriptstyle n-i+1} \end{array} & \end{array} .$$

On dit alors que le problème *découple* en deux problèmes mettant maintenant en jeu les matrices H_{11} et H_{22}.

Soit i, le plus grand entier tel que le coefficient $h_{i,i-1}$ soit négligeable.

- Si $i = n$, h_{nn} est considéré comme une valeur propre approchée. Une *déflation* est effectuée en supprimant la dernière ligne et dernière colonne de la matrice en cours.
- Si $i = n-1$, les valeurs propres de la sous-matrice 2×2 dans le coin inférieur droit sont considérées comme des approximations de valeurs propres de la matrice d'origine. L'algorithme effectue une déflation d'ordre $n-2$ sur la matrice en cours en supprimant les deux dernières lignes et colonnes.
- Si $i < n-1$, les itérations QR continuent sur la sous-matrice correspondant aux lignes et colonnes de i à n.

Nous allons maintenant clarifier ce que l'on veut dire par "suffisamment petit". Supposons que les éléments de la matrice initiale H soient connus avec une incertitude de l'ordre de $\mathbf{u}\|H\|_2$ où $\mathbf{u}$ désigne la précision machine. Supposons qu'à la $k^{ème}$ itération QR nous ayons

$$|h_{i,i-1}^{(k)}| \leq \mathbf{u}\|H\|_F. \tag{4.1.1}$$

Alors, puisque H est unitairement semblable à H_k, annuler $h_{i,i-1}^{(k)}$ correspond à une perturbation de l'ordre de $\mathbf{u}\|H\|_F$ ou moins, ce qui est négligeable.

Avec l'utilisation de ce critère d'arrêt, l'algorithme QR est inversement stable (voir chapitre 2). Cependant, Stewart [106] fait remarquer que le critère (4.1.1) n'est pas satisfaisant pour les *matrices gradées* dont la taille des éléments décroit de gauche à droite et dont la forme est illustrée ici par

$$A = \begin{pmatrix} 1 & 10^{-2} & 10^{-4} & 10^{-6} \\ 10^{-2} & 10^{-4} & 10^{-6} & 10^{-8} \\ 10^{-4} & 10^{-6} & 10^{-8} & 10^{-10} \\ 10^{-6} & 10^{-8} & 10^{-10} & 10^{-12} \end{pmatrix}.$$

De telles matrices peuvent avoir des valeurs propres beaucoup plus petites que $\mathbf{u}\|A\|_2$ et insensibles face à de petites perturbations relatives des éléments de la matrice. Suppososns que $h_{nn}^{(k)}$ converge vers une valeur propre plus petite que $\mathbf{u}\|A\|_2$. Le critère (4.1.1), trop grossier, peut mettre l'élément $h_{n,n-1}^{(k)}$ à zéro alors qu'il est plus grand en valeur absolue que $h_{nn}^{(k)}$. Les valeurs propres de petite taille ne sont pas construites avec une grande précision relative. Il est donc préférable d'avoir un critère qui teste si les éléments sous-diagonaux sont négligeables face aux éléments diagonaux locaux.

La plupart des implantations de l'algorithme QR [104], [3] utilisent un critère proposé par Wilkinson [81]. L'élément de la sous-diagonale en position $(i, i-1)$ de la matrice de Hessenberg H_k à l'itération k est mis à zéro si

$$|h_{i,i-1}^{(k)}| \leq \mathbf{u}(|h_{i-1,i-1}^{(k)}| + |h_{i,i}^{(k)}|). \tag{4.1.2}$$

Notons que le critère (4.1.2) implique le critère (4.1.1) nécessaire pour la stabilité inverse de l'algorithme QR.

Le critère (4.1.2) est essentiellement un test heuristique motivé par la prise en compte des matrices gradées. De nombreuses années d'expériences ont montré qu'il se comporte bien en pratique. Cependant, il semble qu'il n'y ait pas de théorie mathématique pour le justifier.

Dans ce chapitre, nous suggérons un nouveau critère de déflation basé sur quelques considérations mathématiques. Toutes les idées et tous les résultats de ce travail ont été motivés par l'exemple élémentaire 2×2 suivant. Les valeurs propres exactes λ_1, λ_2 de la matrice

$$H = \begin{pmatrix} \tilde{\lambda}_1 & M \\ \varepsilon & \tilde{\lambda}_2 \end{pmatrix},$$

où ε est petit, M grand [1] et $\tilde{\lambda}_1 \neq \tilde{\lambda}_2$, sont données par,

$$\lambda_1 = \frac{1}{2}\left(\tilde{\lambda}_1 + \tilde{\lambda}_2 + (\tilde{\lambda}_1 - \tilde{\lambda}_2)\sqrt{1 + \frac{4\varepsilon M}{(\tilde{\lambda}_1 - \tilde{\lambda}_2)^2}}\right),$$
$$\lambda_2 = \frac{1}{2}\left(\tilde{\lambda}_1 + \tilde{\lambda}_2 - (\tilde{\lambda}_1 - \tilde{\lambda}_2)\sqrt{1 + \frac{4\varepsilon M}{(\tilde{\lambda}_1 - \tilde{\lambda}_2)^2}}\right).$$

Les valeurs propres exactes sont proches des valeurs propres approchées $\tilde{\lambda}_1$ et $\tilde{\lambda}_2$ si et seulement si $\dfrac{|\varepsilon M|}{|\tilde{\lambda}_1 - \tilde{\lambda}_2|^2}$ est très petit. Cela suggère que le partitionnement et le critère d'arrêt doivent prendre en compte, simultanément, trois paramètres :

1) la taille des éléments de la sous-diagonale,

2) la taille des éléments de la partie triangulaire supérieure stricte,

3) la distance entre deux éléments diagonaux consécutifs.

Dans la suite, nous présentons une analyse qui généralise cet exemple 2×2. Au paragraphe 3, nous proposons un critère de déflation pour l'algorithme QR implicite multishifté. Nous illustrons les performances de ce critère numériquement en utilisant l'implémentation de LAPACK pour le double shift QR implicite.

4.2 Approximation par les Eléments Diagonaux

Considérons l'algorithme QR de base en arithmétique complexe à précision finie, appliqué à une matrice de Hessenberg irréductible. Nous disons qu'il converge (voir [85], [48]) lorsqu'il construit une suite de matrices de Hessenberg irréductibles unitairement semblables pour lesquelles la première sous-diagonale tend vers 0, la partie strictement supérieure reste bornée et la diagonale principale tend vers les valeurs propres.

[1] Lorsque l'on réduit une matrice de Hessenberg supérieure par transformations unitaires, M ne peut pas devenir plus grand que $\|H\|$

Considérons une matrice de Hessenberg de la forme

$$H_n(\varepsilon) = \begin{pmatrix} h_{11} & h_{12} & \dots & h_{1n} \\ \eta_1(\varepsilon) & h_{22} & & \vdots \\ & \ddots & \ddots & \vdots \\ 0 & & \eta_{n-1}(\varepsilon) & h_{nn} \end{pmatrix},$$

où les éléments de la sous-diagonale ont été écrits comme fonction de ε et satisfont $\eta_i(0) = 0$.

L'analyse de ce paragraphe ne fait aucune référence aux itérations QR, mais par la suite, nous regarderons $H_n(\varepsilon)$ comme un itéré QR.

Soit $\lambda(\varepsilon)$ une valeur propre de $H_n(\varepsilon)$ donc de H_n, simple. Si la fonction λ : $\varepsilon \in \mathbb{C} \mapsto \lambda(\varepsilon) \in \mathbb{C}$ est analytique dans un voisinage de 0 et si $\lambda(0) = h_{ii}$, alors, un développement de MacLaurin au premier ordre conduit à

$$|\lambda(\varepsilon) - h_{ii}| \;\leq\; |\lambda'(0)\,\varepsilon| + O(|\varepsilon|^2). \tag{4.2.1}$$

Si les valeurs propres sont défectives, il faut augmenter l'ordre du développement. La valeur $\lambda^{'}(0)$ mesure la fiabilité de l'élément diagonal h_{ii} comme approximation d'une valeur propre de H_n. Cette mesure peut être utilisée pour suggérer un critère d'arrêt de l'algorithme QR de base.

Dans ce qui suit, nous proposons deux manières d'établir une expression simple et calculable en pratique, pour $\lambda^{'}(0)$. La première utilise le polynôme caractéristique de H_n tandis que la seconde utilise des résultats provenant de la théorie des perturbations.

4.2.1 Polynôme caractéristique

Nous commençons par démontrer quelques résultats utiles pour démontrer le théorème. Le premier concerne le polynôme caractéristique de H_n.

Lemme 7 *Pour $n > 2$, le polynôme caractéristique de $H_n(\varepsilon)$, noté $p_n(\varepsilon, \cdot)$, est donné par*

$$\begin{aligned} p_n(\varepsilon, x) &= (h_{nn} - x)p_{n-1}(\varepsilon, x) - \eta_{n-1}(\varepsilon)\ h_{n-1,n}\ p_{n-2}(\varepsilon, x) \\ &\quad + \eta_{n-1}(\varepsilon)\eta_{n-2}(\varepsilon)\Delta_{n-2}(\varepsilon, x), \end{aligned}$$

où $\Delta_{n-2}(\varepsilon, x)$ est le déterminant d'une matrice d'ordre $n-2$ dépendant de ε et de x.

Preuve. Développons $\det(H_n(\varepsilon) - xI)$ par rapport à sa dernière ligne. Nous obtenons

$$p_n(\varepsilon, x) = (h_{nn} - x)p_{n-1}(\varepsilon, x) - \eta_{n-1}(\varepsilon)q_{n-1}(\varepsilon, x) \tag{4.2.2}$$

où $q_{n-1}(\varepsilon, x)$ est défini par

$$q_{n-1}(\varepsilon, x) = \det \begin{pmatrix} h_{11} - x & \dots & \dots & h_{1n} \\ \eta_1(\varepsilon) & \ddots & & \vdots \\ & \ddots & h_{n-2,n-2} - x & h_{n-2,n} \\ 0 & & \eta_{n-2}(\varepsilon) & h_{n-1,n} \end{pmatrix}.$$

De la même manière, développons $q_{n-2}(\varepsilon, x)$ par rapport à sa dernière ligne. Cela conduit à

$$q_{n-1}(\varepsilon, x) = h_{n-1,n}p_{n-2}(\varepsilon, x) - \eta_{n-2}(x)\Delta_{n-2}(\varepsilon, x), \tag{4.2.3}$$

où $\Delta_{n-2}(\varepsilon, x)$ est défini par

$$\Delta_{n-2}(\varepsilon, x) = \det \begin{pmatrix} h_{11} - x & \dots & \dots & h_{1n} \\ \eta_1(\varepsilon) & \ddots & & \vdots \\ & \ddots & h_{n-3,n-3} - x & h_{n-3,n} \\ 0 & & \eta_{n-3}(\varepsilon) & h_{n-2,n} \end{pmatrix}.$$

□

Soit f définie par $f(\varepsilon) := p_n(\varepsilon, \lambda(\varepsilon))$. Pour ε assez petit, $f(\varepsilon) = 0$ et par suite,

$$f'(\varepsilon) = \frac{\partial p_n}{\partial \varepsilon}(\varepsilon, \lambda(\varepsilon)) + \lambda'(\varepsilon)\frac{\partial p_n}{\partial x}(\varepsilon, \lambda(\varepsilon)) = 0.$$

La dernière égalité appliquée à $\lambda(0) = h_{ii}$ conduit à

$$\lambda'(0)\frac{\partial p_n}{\partial x}(0, h_{ii}) = -\frac{\partial p_n}{\partial \varepsilon}(0, h_{ii}). \tag{4.2.4}$$

Les lemmes suivants concernent le calcul de $\dfrac{\partial p_n}{\partial x}(0, h_{ii})$ et $\dfrac{\partial p_n}{\partial \varepsilon}(0, h_{ii})$.

Lemme 8 *Supposons que les η_i soient des fonctions différentiables en* 0. *Pour $n \geq 2$, nous avons*

$$\frac{\partial p_n}{\partial \varepsilon}(0, x) = -\sum_{i=1}^{n-1} \eta_i^{'}(0)\ h_{i,i+1} \left\{ \prod_{\substack{j=1 \\ j \neq i, i+1}}^{n} (h_{jj} - x) \right\}, \tag{4.2.5}$$

Preuve. La preuve se fait par récurrence sur n. Si $\eta_1^{'}(0)$ existe, alors, pour $n = 2$,

$$\frac{\partial p_n}{\partial \varepsilon}(0, x) = -\eta_1^{'}(0)h_{12}.$$

D'après le lemme 7, pour $n \geq 3$,

$$p_n(\varepsilon, x) = (h_{nn} - x)p_{n-1}(\varepsilon, x) - \eta_{n-1}(\varepsilon)\ h_{n-1,n}\ p_{n-2}(\varepsilon, x) + \eta_{n-1}(\varepsilon)\eta_{n-2}(\varepsilon)\Delta_{n-2}(\varepsilon, x).$$

Puisque les fonctions η_i sont différentiables au voisinage de 0,

$$\begin{aligned} \frac{\partial p_n}{\partial \varepsilon}(\varepsilon, x) &= (h_{nn} - x)\frac{\partial p_{n-1}}{\partial \varepsilon}(\varepsilon, x) - \eta_{n-1}^{'}(\varepsilon)\ h_{n-1,n}p_{n-2}(\varepsilon, x) \\ &\quad - \eta_{n-1}(\varepsilon)\ h_{n-1,n}\frac{\partial p_{n-2}}{\partial \varepsilon}(\varepsilon, x) \\ &\quad + \eta_{n-1}^{'}(\varepsilon)\eta_{n-2}(\varepsilon)\Delta_{n-2}(\varepsilon, x) + \eta_{n-1}(\varepsilon)\eta_{n-2}^{'}(\varepsilon)\Delta_{n-2}(\varepsilon, x) \\ &\quad + \eta_{n-1}(\varepsilon)\eta_{n-2}(\varepsilon)\frac{\partial \Delta_{n-2}}{\partial \varepsilon}(\varepsilon, x). \end{aligned}$$

Or, pour $n \geq 1$,

$$p_n(0, x) = \prod_{j=1}^{n}(h_{jj} - x), \tag{4.2.6}$$

et puisque $\eta_i(0) = 0$, évaluer $\frac{\partial p_n}{\partial \varepsilon}(\varepsilon, x)$ en $\varepsilon = 0$, conduit à

$$\frac{\partial p_n}{\partial \varepsilon}(0, x) = -(h_{nn} - x)\sum_{i=1}^{n-2} \eta_i^{'}(0)h_{i,i+1} \left\{ \prod_{\substack{j=1 \\ j \neq i, i+1}}^{n-1} (h_{jj} - x) \right\}$$

$$- \eta_{n-1}^{'}(0) h_{n-1,n} \prod_{j=1}^{n-2} (h_{jj} - x)$$

$$= \; - \sum_{i=1}^{n-1} \eta_i^{'}(0) \; h_{i,i+1} \left\{ \prod_{\substack{j=1 \\ j \neq i,i+1}}^{n} (h_{jj} - h_{ii}) \right\}.$$

□

Lemme 9 *Pour $n \geq 1$,*

$$\frac{\partial p_n}{\partial x}(0, h_{ii}) = - \prod_{\substack{j=1 \\ j \neq i}}^{n} (h_{jj} - h_{ii}). \tag{4.2.7}$$

Preuve. Dérivons (4.2.6) par rapport à x. Nous obtenons

$$p_n'(0, x) = - \sum_{k=1}^{n} \prod_{\substack{j=1 \\ j \neq i}}^{n} (h_{jj} - x),$$

d'où le résultat en $x = h_{ii}$. □

La proposition suivante fournit une expression simple pour $\lambda^{'}(0)$.

Théorème 8 *Supposons que $\lambda'(0)$ existe, que les $\eta_i(.)$ sont des fonctions différentiables en 0 et que les h_{ii} sont tous distincts. Alors,*

Si $\lambda(0) = h_{11}, \qquad \lambda'(0) = \dfrac{\eta_1'(0) h_{12}}{h_{11} - h_{22}}.$

Si $\lambda(0) = h_{nn}, \qquad \lambda'(0) = \dfrac{\eta_{n-1}'(0) h_{n-1,n}}{h_{n-1,n-1} - h_{nn}}.$

Si $\lambda(0) = h_{ii}$ *pour un* i *tel que* $2 \leq i \leq n - 1$,

$$\lambda'(0) = \frac{\eta_{i-1}'(0) h_{i-1,i}}{h_{i,i} - h_{i-1,i-1}} + \frac{\eta_i'(0) h_{i,i+1}}{h_{i,i} - h_{i+1,i+1}}.$$

Preuve. Si les h_{ii} sont tous distincts alors, d'après (4.2.7), $\dfrac{\partial p_n}{\partial x}(0, h_{ii})$ est non nul. Si les η_i sont dérivables en 0, les équations (4.2.4) et (4.2.5) conduisent à

$$\lambda'(0) \;\; = \;\; \frac{-\dfrac{\partial p_n}{\partial \varepsilon}(0, h_{ii})}{\dfrac{\partial p_n}{\partial x}(0, h_{ii})}$$

$$= \frac{\sum_{i=1}^{n-1} \eta_i^{'}(0)\ h_{i,i+1} \left\{ \prod_{\substack{j=1 \\ j\neq i,i+1}}^{n} (h_{jj} - h_{ii}) \right\}}{\sum_{i=1}^{n} \prod_{\substack{j=1 \\ j\neq i}}^{n} (h_{jj} - h_{ii})}.$$

Après quelques simplifications, nous obtenons les résultats de la Proposition 8.

□

4.2.2 Théorie des perturbations

Dans ce paragraphe, nous utilisons quelques résultats classiques provenant de la théorie de fonctions. Nous supposons toujours que les $\eta_i(\varepsilon)$ sont des fonctions différentiables au voisinage de zéro. Réécrivons $H_n(\varepsilon)$ sous la forme

$$H_n(\varepsilon) = H_n(0) + \varepsilon E(\varepsilon)$$

où

$$E(\varepsilon) = \begin{pmatrix} 0 & \cdots & \cdots & 0 \\ \eta_1'(\varepsilon) + O(\varepsilon^2) & \ddots & & \vdots \\ & \ddots & & \vdots \\ 0 & & \eta_{n-1}'(\varepsilon) + O(\varepsilon^2) & 0 \end{pmatrix}.$$

Supposons que λ est une valeur propre simple de $H_n(\varepsilon) \in \mathbb{C}^{n\times n}$ et que x et y vérifient $H_n(0)x = \lambda x$ et $y^* H_n(0) = \lambda y^*$ avec $\|x\|_2 = \|y\|_2 = 1$.

On peut montrer que dans un voisinage de zéro, il existe des fonctions différentiables $x(\varepsilon)$ et $\lambda(\varepsilon)$ telles que

$$H_n(\varepsilon)x(\varepsilon) = \lambda(\varepsilon)x(\varepsilon),$$

oú $x(0) = x$ et $\lambda(0) = \lambda$.

En dérivant par rapport à ε et en mettant ε à 0 dans le résultat, nous obtenons

$$H_n(0)x'(0) + E(0)x = \lambda'(0)x + \lambda x'(0).$$

Multiplier les deux membres de l'égalité par y^* conduit à

$$y^*E(0)x = \lambda'(0)y^*x.$$

Puisque λ est une valeur propre simple, $y^*x \neq 0$ et $\lambda'(0)$ est alors donné par

$$\lambda'(0) = \frac{y^*E(0)x}{y^*x}. \tag{4.2.8}$$

Lemme 10 *Supposons que $\lambda = h_{ii}$ et $h_{ii} \neq h_{jj}, j \neq i$. Alors x et y sont définis par*

$$\begin{cases} x_k = -\dfrac{1}{\overline{h_{kk}} - \overline{h_{ii}}} \displaystyle\sum_{j=k+1}^{i} \overline{h_{kj}} x_j, \ \text{pour } k < i, \\ x_k = 0 \ \text{pour } k > i \end{cases}$$

$$\begin{cases} y_k = 0 \ \text{pour } k < i, \\ y_k = -\dfrac{1}{h_{kk} - h_{ii}} \displaystyle\sum_{j=k+1}^{i} h_{kj} x_j, \ \text{pour } k > i. \end{cases}$$

x_i et y_i sont choisis tels que $\|x\|_2 = \|y\|_2 = 1$.

Preuve. Le vecteur propre à droite x (respectivement à gauche y) est solution du système triangulaire supérieur (respectivement inférieur) $(H_n(0) - h_{ii}I)x = 0$ et $(H_n(0) - h_{ii}I)^*y = 0$. □

Maintenant, évaluons $\lambda'(0)$ en utilisant (4.2.8). Tout d'abord, nous avons

$$E(0)x = \begin{pmatrix} 0 \\ \eta_1'(0)x_1 \\ \vdots \\ \eta_{i-1}'(0)x_{i-1} \\ \eta_i'(0)x_i \\ 0 \\ \vdots \\ 0 \end{pmatrix} \quad \text{et} \quad y = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ y_i \\ y_{i+1} \\ \vdots \\ y_n \end{pmatrix},$$

si bien que $y^*E(0)x = \sum_{k=1}^{n-1} \eta_k'(0)x_k\overline{y_{k+1}}$.

Si $1 < i < n$, alors

$$\begin{aligned} y^*E(0)x &= \eta_{i-1}'(0)x_{i-1}\overline{y_i} + \eta_i'(0)x_i\overline{y_{i+1}} \\ &= -\eta_{i-1}'(0)\frac{h_{i-1,i}}{h_{i-1,i-1} - h_{ii}}x_i\overline{y_i} - \eta_i'(0)\frac{h_{i+1,i}}{h_{i+1,i+1} - h_{ii}}x_i\overline{y_i}, \end{aligned}$$

et $y^*x = x_i\overline{y_i}$. Ainsi,

$$\lambda'(0) = \eta_{i-1}'(0)\frac{h_{i-1,i}}{h_{ii} - h_{i-1,i-1}} + \eta_i'(0)\frac{h_{i+1,i}}{h_{ii} - h_{i+1,i+1}}.$$

Si $i = 1$, $y^*E(0)x = \eta_1'(0)\overline{y_2}$ et par suite

$$\lambda'(0) = \eta_1'(0)\frac{h_{12}}{h_{11} - h_{22}}.$$

Si $i = n$, $y^*E(0)x = \eta_{n-1}'(0)x_{n-1}$ et par suite

$$\lambda'(0) = \eta_{n-1}'(0)\frac{h_{n-1,n}}{h_{n-1,n-1} - h_{nn}}.$$

Nous obtenons la même expression que celle du Théorème 8.

Le théorème suivant fournit une borne pour la distance entre les coefficients diagonaux de $H_n(\varepsilon)$ et ses valeurs propres.

Théorème 9 *Sous les mêmes hypothèses que celles de la Proposition* 8 *et les conventions* $h_{00} = -\infty$, $h_{n+1,n+1} = +\infty$, *nous avons, pour* $i = 1, \ldots, n$,

$$|\lambda(\varepsilon) - h_{ii}| \leq \frac{|\eta_{i-1}(\varepsilon)|\ |h_{i-1,i}|}{|h_{i,i} - h_{i-1,i-1}|} + \frac{|\eta_i(\varepsilon)|\ |h_{i,i+1}|}{|h_{i,i} - h_{i+1,i+1}|} + O(|\varepsilon|^2). \tag{4.2.9}$$

Preuve. Il suffit d'appliquer les résultats de la Proposition 8 à l'inégalité (4.2.1) avec

$$\varepsilon\eta_i'(0) = \eta_i(\varepsilon) + O(\varepsilon^2).$$

□

De même que pour le petit exemple 2×2 de l'introduction, nous remarquons que la validité des éléments diagonaux en tant qu'approximation des valeurs propres dépend de trois paramètres :

1) la taille des éléments sur-diagonaux,

2) la taille des éléments sous-diagonaux,

3) la distance entre deux éléments diagonaux successifs.

Remarquons enfin que d'un point de vue pratique, la borne (4.2.9) peut être facilement évaluée puisque, mis à part le $O(|\varepsilon|^2)$, toutes les quantités mises en jeu sont connues à chaque itération QR.

4.3 Critère de Déflation

Dans cette partie, nous proposons un critère de déflation pour l'algorithme QR. Nous souhaitons négliger un élément de la sous diagonale seulement si l'élément adjacent diagonal est suffisamment proche d'une valeur propre de la matrice donnée. Ainsi, nous souhaitons négliger $h_{i,i-1}^{(k)}$ si

$$\frac{|\lambda(\varepsilon) - h_{ii}^{(k)}|}{|h_{ii}^{(k)}|} \leq \mathbf{u}.$$

Le Théorème 9 montre que cet ordre de précision est obtenu à l'étape k si

$$|h_{i,i-1}^{(k)}|\ |h_{i-1,i}^{(k)}| \leq \mathbf{u}|h_{ii}^{(k)}||h_{ii}^{(k)} - h_{i-1,i-1}^{(k)}| \tag{4.3.1}$$

(rappelons que $\eta_i(\varepsilon) = h_{i,i-1}^{(k)}$). Notons que le terme $h_{ii}^{(k)}$ dans le membre droit de l'inégalité rend ce test approprié pour les matrices gradées. Notre critère de déflation est tel que les conditions (4.3.1) et (4.1.1) soient vérifiées. Nous imposons la dernière condition de manière à assurer que la stabilité inverse de l'algorithme est maintenue.

Ce critère peut facilement être implanté dans LAPACK [3] en modifiant la routine `xLAHQR`. Cette routine est une implantation de l'algorithme QR implicite avec double shift. C'est une version améliorée de la procédure `hqr2` (voir [81], [91], [104]).

Après les lignes

```
*
*     Perform QR iterations on rows and columns ILO to I until a
*     submatrix of order 1 or 2 splits off at the bottom because a
*     subdiagonal element has become negligible.
*
      DO 130 ITS = 0, ITN
*
*        Look for a single small subdiagonal element.
*
```

remplacer les lignes

```
*
      DO 130 ITS = 0, ITN
*
*        Look for a single small subdiagonal element.
*
         DO 20 K = I, L + 1, -1
            TST1 = ABS( H( K-1, K-1 ) ) + ABS( H( K, K ) )
            IF( TST1.EQ.ZERO )
     $         TST1 = DLANHS( '1', I-L+1, H( L, L ), LDH, WORK )
            IF( ABS( H( K, K-1 ) ).LE.MAX( ULP*TST1, SMLNUM ) )
     $         GO TO 30
   20    CONTINUE
   30    CONTINUE
         L = K
         IF( L.GT.ILO ) THEN
*
*           H(L,L-1) is negligible
```

```
*
            H( L, L-1 ) = ZERO
         END IF
```

par les lignes

```
 *
      DO 130 ITS = 0, ITN
*
*        Look for a single small subdiagonal element.
*
         DO 20 K = I, L + 1, -1
            IF( ABS( H( K, K-1 )).GT. (ULP*NORM)) GO TO 20
            TST1 = ABS( H( K-1, K-1 ) - H( K, K ) )*
     $           ABS( H( K, K ) )
            SS  =  ABS( H( K-1, K ) )
            IF( SS.EQ.ZERO )  GO TO 30
            IF( ABS( H( K, K-1 )).LE.MAX( ULP*TST1/SS,SMLNUM ))
     $         GO TO 30
   20    CONTINUE
   30    CONTINUE
         L = K
         IF( L.GT.ILO ) THEN
*
*           H(L,L-1) is negligible
*
            H( L, L-1 ) = ZERO
         END IF
```

où `NORM` est la norme $\|\|\|_1$ de la matrice H et `ULP` est la précision machine.

Voici un petit exemple où le nouveau crière de déflation améliore la précision. Soit H définie par

$$H = \begin{pmatrix} 1 & M & 0 \\ \varepsilon & 1+d & M \\ 0 & \varepsilon & 1+2d \end{pmatrix}.$$

Pour $\varepsilon = 1.1e-8$, $M = 1.1e5$, $d = 1.e-2$, le test classique de `SLAHQR` (en simple précision) déflate immédiatement L'erreur relative sur les valeurs propres construites est de l'ordre de $0.4e-1$. Avec le nouveau critère de déflation, trois itérations QR supplémentaires sont nécessaires. L'erreur relative obtenue est de l'ordre de $0.7e-7$.

Nous avons testé ce critère d'arrêt sur un ensemble de matrices test construites par la routine `xLATME` de la librairie LAPACK. Ces matrices sont de la forme UTU^T où U est une matrice orthogonale aléatoire, $T = D + N$ avec D, matrice diagonale, choisie suivant un mode et un paramètre ρ :

mode=0 : D est définie par l'utilisateur.

mode=1 : la matrice D est définie par

$$D(1) = 1 \text{ et } D(2:N) = 1/\rho.$$

mode=2 : la matrice D est définie par

$$D(1:N-1) = 1 \text{ et } D(N) = 1/\rho.$$

mode=3 : la matrice D est définie par

$$D(i) = \rho^{\frac{i-1}{n-1}}.$$

Distribution géométrique des valeurs propres de 1 à ρ.

mode=4 : la matrice D est définie par

$$D(i) = 1 - \frac{i-1}{n-1}(1-\rho).$$

	Matrices diagonalisables			
	Mode 0		Mode 2	
n	test classique	nouveau test	test classique	nouveau test
20	43	44	30	20
40	96	91	49	44
60	144	140	80	71
80	174	173	95	111
100	211	210	135	157
	Mode 3		Mode 5	
	test classique	nouveau test	test classique	nouveau test
20	34	34	49	49
40	69	69	96	96
60	113	116	119	124
80	149	145	168	164
100	185	184	207	203

TAB. 4.3.1: Matrices construites par `DLATME`. Pour MODE=0, D est remplie aléatoirement par des nombres compris dans l'intervalle $[-1, 1]$. Tests effectués en double precision.

Distribution arithmétique des valeurs propres de 1 à ρ.

mode=5 : la matrice D est définie de manière aléatoire telle que $D(i) \in [\rho, 1], i = 1, \ldots, n$, et le logarithme des $D(i)$ est uniformément distribué.

Les deux tests de déflation sont très semblables en ce qui concerne le nombre d'itérations(voir Table 4.3.1). Le nouveaux critère d'arrêt semble demander un peu moins d'itérations lorsque la matrice est diagonalisable et peut demander un

	Matrices diagonalisables		Matrices non-normales	
n	test classique	nouveau test	test classique	nouveau test
20	43	43	44	44
40	85	84	99	101
60	89	82	155	154
80	124	111	201	209
100	158	147	244	245

TAB. 4.3.2: Matrices construites par `DLATME` avec Mode=2 et $\rho = 1.0e-3$. Tests effectués en double précision.

n	test classique		nouveau test	
	Nb. d'itérations	Erreur relative	Nb. d'itérations	Erreur relative
20	50	0.79D-01	51	0.79D-01
40	83	0.11D+01	86	0.41D+00
60	127	0.20D+01	124	0.20D+01

Tab. 4.3.3: Matrices construites par `DLATME` avec MODE=1 ($D(1) = 1, D(2 : n) = 10^{-2}$), $\rho = 1.0D + 02$.

n	test classique		nouveau test	
	Nb. d'itérations	Erreur relative	Nb. d'itérations	Erreur relative
4	4	0.11D-03	4	0.11D-03
8	10	0.76D-02	10	0.76D-02
10	15	0.58D-02	19	0.58D-02
12	17	0.13D-02	21	0.13D-02

Tab. 4.3.4: Matrices gradées.

peu plus lorsqu'elle est non-normale (voir Table 4.3.2).

Dans le but de comparer la précision des valeurs propres construites par l'un ou l'autre de ces critères, nous avons exécuté les deux tests en simple précision et comparé les résultats avec ceux qui sont obtenus en double précision avec la routine `DLAHQR` (voit Table 4.3.3). Parfois, le nouveau critère demande plus d'itérations mais fournit une meilleure précision. Parfois, il demande moins d'itérations et fournit la même précision.

Nous avons également fait des tests sur des matrices gradées. Comme nous l'avions espéré, les résultats sont très semblables à ceux qui sont obtenus avec le critère classique en ce qui concerne la précision et le nombre d'itérations (voir Table 4.3.4).

4.4 Conclusion

Notre analyse suggère que, en pratique, un critère de déflation pour l'algorithme QR devrait prendre en compte la taille des éléments sous-diagonaux, la taille des éléments de la partie strictement supérieure et la distance entre deux éléments diagonaux consécutifs. Le critère que nous avons proposé a ces propriétés et nos expériences numériques indiquent qu'il est compétitif, en pratique, avec le test traditionnel (heuristiquement motivé).

Bien qu'étant de forme assez différente, il peut paraître surprenant que ces critères aient un comportement aussi proche. En fait, dans des cas non critiques, si l'on pense en terme de magnitude, les deux tests sont la plupart du temps semblables.

Annexe

Conditionnement des Nombres de Conditionnement Spectraux

Le *nombre de conditionnement spectral* d'une matrice A pour le calcul d'une valeur propre λ, noté $\mathrm{Cond}(A, \lambda)$, mesure la sensibilité de cette valeur propre face à une perturbation de A. On dit que le problème est *mal conditionné* si son nombre de conditionnement est grand.

Soit $\tilde{\lambda}$ une valeur propre simple approchées de A. Nous pouvons considérer $\tilde{\lambda}$ comme étant une valeur propre construite par un algorithme. Si l'algorithme de construction de $\tilde{\lambda}$ est régressivement stable, $\tilde{\lambda}$ est une valeur propre exacte d'une matrice $\widetilde{A}$ telle que $\dfrac{\|A-\widetilde{A}\|}{\|A\|}$ est petit.

Posons $\Delta A = A - \widetilde{A}$. Le conditionnement gouverne une relation entre l'erreur directe et l'erreur inverse :

$$\frac{|\lambda - \tilde{\lambda}|}{|\lambda|} \leq \mathrm{Cond}(A, \lambda) \frac{\|\Delta A\|}{\|A\|} + O\left(\frac{\|\Delta A\|^2}{\|A\|^2}\right).$$

Ainsi, pour un problème mal conditionné, la solution construite peut avoir une erreur directe importante même si elle a été calculée avec une petite erreur régressive.

Si l'on utilise un algorithme régressivement stable pour résoudre le problème de valeurs propres, nous pouvons utliser une estimation de $\mathrm{Cond}(A, \lambda)$ pour borner l'erreur des solutions calculées. Si λ est une valeur propre simple, alors

$$\mathrm{Cond}(A,\lambda) = \|P\|_2$$

où P est la projection spectrale associée à λ. Z. Bai, J. Demmel and A. McKenney [8] ont développé des algorithmes pour construire une estimation de $\|P\|_2$ pour des matrices réduites sous forme de Schur canonique.

Dans cette annexe, nous nous intéressons plus particulièrement au conditionnement du calcul de $\mathrm{Cond}(A,\lambda)$. Ce problème est-il bien conditionné ?

Afin d'obtenir une borne pour

$$\left|\mathrm{Cond}(A,\lambda) - \mathrm{Cond}(\widetilde{A},\tilde{\lambda})\right| = \left|\|P\|_2 - \|\widetilde{P}\|_2\right|$$

en termes de ΔA, nous utilisons la définition de la projection spectrale suivante :

$$P = -\frac{1}{2\pi i}\int_{\Gamma_\lambda} R(z)dz$$

où $R(z) = (A - zI)^{-1}$ est la résolvante sur le contour Γ_λ. Nous choisissons la courbe Γ_λ telle qu'elle entoure λ et $\tilde{\lambda}$. Nous avons

$$\begin{aligned}
|\,\mathrm{Cond}(A,\lambda) - \mathrm{Cond}(\widetilde{A},\tilde{\lambda})| &\leq \|P - \tilde{P}\|_2 \\
&= \frac{1}{2\pi}\|\int_{\Gamma_\lambda}\Big(R(z) - \tilde{R}(z)\Big)\ dz\|_2 \\
&= \frac{1}{2\pi}\|\int_{\Gamma_\lambda}\Big(R(z)(\widetilde{A} - zI)\tilde{R}(z) - R(z)(A - zI)\tilde{R}(z)\Big)\ dz\|_2 \\
&= \frac{1}{2\pi}\|\int_{\Gamma_\lambda}\tilde{R}(z)(\Delta A)R(z)\ dz\|_2.
\end{aligned}$$

Mais,

$$\begin{aligned}
\tilde{R}(z) &= R(z)[I - (\Delta A)R(z)]^{-1} \\
&= R(z)[I + (\Delta A)R(z) + O((\Delta A)^2)].
\end{aligned}$$

Ainsi,

$$\|P - \tilde{P}\|_2 = \frac{1}{2\pi}\|\int_{\Gamma_\lambda} R(z)(\Delta A)R(z)dz\|_2 + O(\|\Delta A\|_2^2)$$

$$\begin{aligned}
&\leq \frac{1}{2\pi}\left[\int_{\Gamma_\lambda} \|R(z)\|_2^2 |dz|\right] \|\Delta A\|_2 + O(\|\Delta A\|_2^2) \\
&\leq \frac{1}{2\pi}\max_{z\in\Gamma_\lambda} \|R(z)\|_2^2 \operatorname{long}(\Gamma_\lambda)\ \|\Delta A\|_2 + O(\|\Delta A\|_2^2)
\end{aligned}$$

où $\operatorname{long}(\Gamma_\lambda)$ est la longueur du contour de Γ_λ. Soit $r_1 = \min_i |\lambda - \lambda_i|$ où les λ_i, $2 \leq i \leq n$ sont les valeurs propres de A autres que λ. Nous définissons Γ_λ comme étant un cercle aoutour de λ de rayon $r = r_1/2$. Nous supposons que $|\tilde{\lambda} - \lambda| < r$. Alors,

$$\max_{z\in\Gamma_\lambda} \|R(z)\|^2 = \max_{z\in\Gamma_\lambda} \max_{\mu\in Sp(A)-\lambda} [1/(|z-\mu|)]^2 = 1/r^2.$$

Ainsi,

$$\left|\|P\|_2 - \|\widetilde{P}\|_2\right| \leq \frac{2}{\min_i |\lambda - \lambda_i|} \|\Delta A\|_2 + O(\|\Delta A\|_2^2).$$

Notons que si $\operatorname{Cond}(A, x)$ désigne le nombre de condition pour le vecteur propre associé à la valeur propre λ, il est montré dans [106, p.296–297] qu'il existe un vecteur propre x pour lequel

$$\operatorname{Cond}(A, x)^{-1} \leq \min_i |\lambda - \lambda_i|$$

donc

$$|\operatorname{Cond}(A, \lambda) - \operatorname{Cond}(\widetilde{A}, \tilde{\lambda})| \leq \operatorname{Cond}(A, x)\|\Delta A\|_2 + O(\|\Delta A\|_2^2).$$

Nous obtenons un résultat proche du résulat plus général de Demmel [30] qui clame que pour certains problèmes d'analyse numérique, *"le nombre de condition du nombre de condition est le nombre de condition".* Dans ce cas, nous montrons que le nombre de condition du nombre de condition pour le calcul des valeurs propres est borné par le nombre de condition pour le calcul des vecteurs propres.

Deuxième partie

Algorithme de Yau et Lu

Chapitre 5

Méthode de Yau et Lu

Il existe de nombreuses méthodes pour résoudre le problème de valeurs propres de matrices denses symétriques (voir [86] et §1.1). Avec l'arrivée des machines parallèles, les efforts se sont concentrés dans un premier temps sur la parallélisation des méthodes traditionnelles en séquentiel [108], [100].

Ces méthodes consistent en trois étapes [49] : tout d'abord, la matrice est réduite sous forme tridiagonale, puis les valeurs propres et vecteurs propres de cette matrice sont construits ; finalement, les vecteurs propres sont retransformés via la transformation inverse utilisée pour la tridiagonalisation. Bien que les algorithmes séquentiels pour les étapes de tridiagonalisation et transformation inverse soient bien compris, les solveurs tridiagonales de valeurs propres font encore partie des domaines de recherche actifs [33], [52], [65]. En ce qui concerne les algorithmes parallèles, il y a eu beaucoup plus de travaux effectués sur la parallélisation du solveur tridiagonal que sur la parallélisation de la tridiagonalisation elle même. Les premières recherches pour la parallélisation du solveur tridiagonal ont commencé avec les études de Huang sur la multisection [61] et ont continué sur plusieurs autres types de solveurs tridiagonaux sur une grande variété d'ordinateurs à mémoire partagée ou distribuée [11], [15], [37], [63], [75], [80].

Dans le but d'utiliser pleinement les avantages du parallélisme, d'autres types de méthodes basées sur des multiplications matricielles sont actuellement étudiées. Auslander et Tsao [4] et Lederman, Tsao et Turnbull [74] ont un algorithme basé sur des produits matrice-matrice et utilisant une transformation polynômiale des valeurs propres. Dans le même esprit, Yau [1] et Lu proposent une méthode d'accélération polynômiale réduisant le problème de valeur propre en une série de produits matriciels.

Nous avons choisi d'étudier la méthode de Yau et Lu et d'en proposer une parallélisation. Nous nous sommes, dans un premier temps, attachés à réexpliquer la méthode en essayant de la simplifier le plus possible, en l'illustrant par des exemples simples. Nous avons corrigé quelques erreurs et rajouté quelques preuves.

5.1 Présentation de la Méthode

Soit A une matrice dense, réelle et symétrique. La symétrie simplifie le problème de valeurs propres dans la mesure où toutes les valeurs propres de A sont réelles, et, il existe une base orthonormale de vecteurs propres dans laquelle A est diagonale (Théorème spectral pour les matrices symétriques [121]).

Nous noterons $\mathrm{Sp}(A) = \{\lambda_1, \lambda_2, \ldots, \lambda_n\}$ l'ensemble des valeurs propres de A ordonnées telles que

$$\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n,$$

et $X = (x_1, x_2, \ldots, x_n)$ la base orthonormale de vecteurs propres telle que

$$X^T A X = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n).$$

[1] Médaille Field, 1982.

5.1.1 Méthode d'Accélération Polynômiale

Les méthodes d'*accélération polynômiale* consistent à exhiber un polynôme de A tel qu'il atteigne un maximum pour l'une des valeurs propres de A et soit proche de zéro pour les autres.

Soit v_0 une vecteur de $\mathbb{R}^n$. Son développement dans la base X des vecteurs propres de A est donné par $v_0 = \alpha_1 x_1 + \alpha_2 x_2 + \cdots + \alpha_n x_n$. Soit P_N un polynôme de degré $N-1$. Nous avons

$$\begin{aligned} P_N(A)v_0 \quad &= P_N(A)\alpha_1 x_1 + P_N(A)\alpha_2 x_2 + \cdots + P_N(A)\alpha_n x_n \\ &= \alpha_1 P_N(\lambda_1)x_1 + \alpha_2 P_N(\lambda_2)x_2 + \cdots + \alpha_n P_N(\lambda_n)x_n. \end{aligned}$$

Si P_N est proche de zéro pour toutes les valeurs propres de A mis à part λ_k et si $\alpha_k \neq 0$, alors

$$\frac{P_N(A)v_0}{\alpha_k P_N(\lambda_k)} \simeq x_k.$$

Ainsi, le vecteur $P_N(A)v_0 \in \mathbb{R}^n$ peut être considéré comme une approximation du vecteur propre x_k associé à la valeur propre λ_k. Le polynôme P_N accélère les composantes de v_0 dans la direction de x_k.

Il y a plusieurs difficultés dans cette approche. Apparemment, elle ne construit qu'un seul couple d'éléments propres à la fois. Pour chacun d'eux, il faut déterminer un nouveau polynôme P_N ou translater le point où $P_N(A)$ atteint son maximum et répéter le processus. Tout cela est trop coûteux pour être intéressant en pratique. Pour contourner ces difficultés Yau et Lu proposent d'incorporer la Transformée de Fourier Rapide (TFR) dans ce processus.

Soit une fonction f telle que $f(A)$ soit une matrice unitaire. Alors, les valeurs propres de $f(A)$ sont toutes sur le cercle unité. Elles peuvent donc s'écrire $e^{i\theta_k}$, $k = 1, \ldots, n$. Soit P_N un polynôme de degré $N-1$ tel que P_N soit proche de 0 sur le cercle unité sauf au voisinage de 1. Définissons

$$u(\lambda) := P_N(e^{-i\lambda} f(A))v_0 = \sum_{j=1}^{n} \alpha_j P_N(e^{-i(\lambda-\theta_j)})x_j. \tag{5.1.1}$$

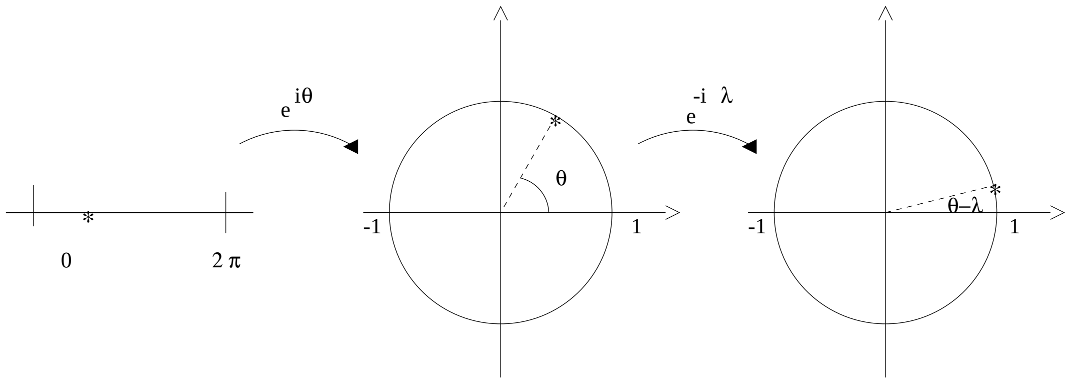

FIG. 5.1.1: Sp(A) est transporté sur le cercle unité et la rotation $e^{-i\lambda}$ fait tourner le cercle.

Supposons qu'on puisse choisir λ tel que $e^{i\theta_k}$ soit proche de $e^{i\lambda}$ et les autres valeurs propres de $f(A)$ soient éloignées de $e^{i\lambda}$. Les coefficients de x_j seront petits sauf pour $j = k$. Le vecteur $u(\lambda)$ ira dans la direction de x_k et pourra être pris comme une approximation du vecteur propre x_k associé à la valeur propre $e^{i\theta_k}$.
Si $P_N(z) = \sum_{j=0}^{N-1} \beta_j z^j$, alors

$$u(\lambda) = \sum_{j=0}^{N-1} \beta_j v_j e^{-ij\lambda} \quad \text{où} \quad v_j = f(A)^j v_0.$$

La fonction $u(\lambda)$ s'écrit sous la forme d'une série de Fourier. Ainsi, si l'on construit d'abord les vecteurs v_j, $u(\lambda)$ peut être évaluée en de nombreuses valeurs de λ équiréparties sur le cercle unité et de manière simultanée, grâce à la TFR.

De manière imagée, cette méthode transporte le spectre de A sur le cercle unité et fait tourner ce cercle de manière à ce que chacune des valeurs propres soit proche de $z = 1$, point en lequel P_N atteint un maximum. Pour faire tourner le cercle d'un angle λ, il suffit d'appliquer la rotation $e^{-i\lambda}$ (voir figure 5.1.1).

Pour appliquer l'approche précédente au problème de valeurs propres symétrique, il faut définir la matrice unitaire $f(A)$, trouver un polynôme P_N et construire les coefficients de Fourier $\beta_j v_j$. Parmi les $u(\lambda)$ construits par la TFR, nous avons besoin de sélectionner ceux qui peuvent être regardés comme une bonne approximation des vecteurs propres. Dans le cas de valeurs propres proches ou multiples, Yau et Lu proposent un rafinenemt local avec possibilité de rajouter des vecteurs propres de manière à ce que toutes les valeurs propres soient construites.

5.1.2 Méthode de Yau et Lu

Définition de la Matrice Unitaire

Nous voulons relier la matrice A à une matrice unitaire. Il s'agit donc d'une matrice dont tout le spectre est sur le cercle unité. Un choix naturel est $f(A) = e^{iA}$.

La fonction $f(z) = e^{iz}$ est analytique à l'intérieur d'un contour fermé Γ qui entoure $\mathrm{Sp}(A)$. Nous définissons e^{iA} comme étant la matrice

$$e^{iA} = \frac{1}{2i\pi} \oint_{\Gamma} e^{iz}(zI - A)^{-1} dz.$$

C'est la version matricielle du théorème intégral de Cauchy. Puisque A est symétrique, elle est diagonalisable et

$$\begin{aligned} e^{iA} &= \frac{1}{2i\pi} \oint_{\Gamma} e^{iz}(zI - XDX^{-T})^{-1} dz \\ &= X \operatorname{diag}\left(\frac{1}{2i\pi} \oint_{\Gamma} e^{iz}(z-\lambda_1)^{-1} dz, \ldots, \frac{1}{2i\pi} \oint_{\Gamma} e^{iz}(z-\lambda_n)^{-1} dz \right) X^{-T} \\ &= X \operatorname{diag}\left(e^{i\lambda_1}, \ldots, e^{i\lambda_n} \right) X^{-T}. \end{aligned}$$

Ainsi, $\mathrm{Sp}(e^{iA}) = \{e^{i\lambda_1}, \ldots, e^{i\lambda_n}\}$. La matrice A étant symétrique, ses valeurs propres sont réelles et donc $\|e^{iA}\|_2 = 1$. Il n'est pas difficile de montrer que

$$e^{iA} = \cos A + i \sin A.$$

Dans la suite, nous imposons au spectre de A d'être contenu dans $[0, 2\pi)$. Ainsi, f réalise une bijection de $[0, 2\pi)$ sur le cercle unité.

Lorsque A est symétrique, le problème de l'exponentielle est un problème bien conditionné. Une application directe des résultats de Van Loan [111, th. 4] conduit à

$$\|e^{i(A+E)} - e^{iA}\|_2 \leq \|E\|_2 e^{\|E\|_2}.$$

Construction du Polynôme P_N

Nous voulons construire un polynôme à coefficients réels β_j et de degré $N-1$,

$$P_N(z) := \sum_{j=0}^{N-1} \beta_j z^j$$

tel que sa partie réelle atteigne son maximum en $z=1$ et soit proche de 0 sur le cercle unité sauf au voisinage de $z=1$. La notation P_N au lieu de P_{N-1} n'est pas logique, mais nous avons décidé de conserver les notations originelles de Yau et Lu.

Nous noterons $\mathbb{P}_k$ l'ensemble des polynômes de degré k. Le polynôme de Chebyshev $T_k \in \mathbb{P}_k$ est défini par :

$$\begin{aligned} T_k(x) &= \cos(k \arccos x), \text{ si } |x| \leq 1, \\ T_k(x) &= \cosh(k \operatorname{arg}\cosh x), \text{ si } |x| > 1. \end{aligned}$$

Pour $z = e^{i\theta}$, nous avons

$$\begin{aligned} \mathrm{Re}(P_N(z)) &= \beta_0 + \beta_1 \cos(\theta) + \beta_2 \cos(2\theta) + \cdots + \beta_{N-1} \cos((N-1)\theta) \\ &= \beta_0 + \beta_1 T_1(\cos\theta) + \beta_2 T_2(\cos\theta) + \cdots + \beta_{N-1} T_{N-1}(\cos\theta) \\ &=: Q_N(\cos\theta). \end{aligned} \tag{5.1.2}$$

Soit δ un réel de petite taille. Les polynômes de Chebyshev ont la propriété de réaliser l'optimum du problème :

$$\min_{\substack{Q_N \in \mathbb{P}_{N-1} \\ Q_N(1)=1}} \max_{x \in [-1, \cos(\delta\pi)]} |Q_N(x)|,$$

et cet optimum est atteint par :

$$Q_N(x) = \frac{T_{N-1}\left(\dfrac{2x+1-\cos(\delta\pi)}{1+\cos(\delta\pi)}\right)}{T_{N-1}\left(\dfrac{3-\cos(\delta\pi)}{1+\cos(\delta\pi)}\right)}.$$

C'est une conséquence du théorème d'alternance de Chebyshev (voir [72, Section 3.5]).

Les coefficients β_j, $j = 0, \ldots, N-1$ peuvent être définis par des relations de récurrence. Pour cela, posons

$$\kappa_N(\delta) = |Q_N(x)|_\infty = \frac{1}{\left|T_{N-1}\left(\dfrac{3-\cos(\delta\pi)}{1+\cos(\delta\pi)}\right)\right|}, \qquad \text{pour } x \in [-1, \cos(\delta\pi)],$$

$$a = a(\delta) = \frac{2}{1+\cos(\delta\pi)} \qquad \text{et} \qquad b = b(\delta) = \frac{1-\cos(\delta\pi)}{1+\cos(\delta\pi)}.$$

Alors,

$$Q_N(x) = \kappa_N(\delta) T_{N-1}(ax+b). \tag{5.1.3}$$

Définissons les coefficients $\beta_j^{(k)}$ pour $k = 1, \ldots, N-1$, par

$$T_k(ax+b) = \sum_{j=0}^{k} \beta_j^{(k)} T_j(x).$$

Ainsi,

$$\begin{aligned} T_0(ax+b) &= \sum_{j=0}^{0} \beta_j^{(0)} T_j(x) = \beta_0^{(0)} = 1, \quad \text{et}, \\ T_1(ax+b) &= \beta_0^{(1)} T_0(x) + \beta_1^{(1)} T_1(x) = \beta_0^{(1)} + \beta_1^{(1)} x = ax+b, \end{aligned}$$

ce qui donne pour les deux premiers ensembles des coefficients

$$\boxed{\beta_0^{(0)} = 1} \qquad \text{et} \qquad \boxed{\beta_0^{(1)} = b \ , \ \beta_1^{(1)} = a.}$$

Posons $\beta_j^{(k)} = 0$ pour $j > k$. Alors, grâce à la formule

$$T_{j+k} = 2T_j T_k - T_{j-k}, \quad j \geq k,$$

nous obtenons

$$\begin{aligned} T_{k+1}(ax+b) &= 2T_k(ax+b)T_1(ax+b) - T_{k-1}(ax+b) \\ &= \sum_{j=0}^{k} \left[2a\beta_j^{(k)} T_j(x) T_1(x) + 2b\beta_j^{(k)} T_j(x)\right] T_0(x) - \sum_{j=0}^{k-1} \beta_j^{(k-1)} T_j(x) \end{aligned}$$

$$
\begin{aligned}
= \quad & \left[2b\beta_0^{(k)} + a\beta_1^{(k)} - \beta_0^{(k-1)}\right] T_0(x) \\
& + \left[a(2\beta_1^{(k)} + \beta_2^{(k)}) + 2b\beta_1^{(k)} - \beta_1^{(k-1)}\right] T_1(x) \\
& + \sum_{j=2}^{k+1} \left(a\beta_{j+1}^{(k)} + a\beta_{j-1}^{(k)} + 2b\beta_j^{(k)} - \beta_j^{(k-1)}\right) T_j(x).
\end{aligned}
$$

Or, par definition,

$$
T_{k+1}(ax+b) \quad = \quad \sum_{j=0}^{k+1} \beta_j^{(k+1)} T_j(x).
$$

ce qui conduit aux relations de récurrence suivantes pour le calcul des coefficients $\beta_j^{(k)}$, $k = 2, \ldots, N-1$:

$$
\boxed{
\begin{aligned}
&\beta_0^{(k+1)} = a\beta_1^{(k)} + 2b\beta_0^{(k)} - \beta_0^{(k-1)}, \\
&\beta_1^{(k+1)} = a\left(2\beta_0^{(k)} + \beta_2^{(k)}\right) + 2b\beta_1^{(k)} - \beta_1^{(k-1)}, \\
&\beta_j^{(k+1)} = a\left(\beta_{j-1}^{(k)} + \beta_{j+1}^{(k)}\right) + 2b\beta_j^{(k)} - \beta_j^{(k-1)}, \text{ pour } 2 \leq j \leq k, \\
&\beta_j^{(k+1)} = 0, \text{ pour } j > k+1.
\end{aligned}
}
$$

Finalement, $\beta_j = \kappa_N(\delta)\, \beta_j^{(N-1)}$, pour $j = 0, \ldots, N-1$.

Dans la suite, nous verrons que le degré du polynôme dépend de la précision désirée sur les vecteurs propres approchés. Nous verrons dans la partie sélection comment choisir la valeur de N.

Construction des Coefficients de Fourier et Calcul des $u(\lambda)$

Dans notre situation, le calcul des $u(\lambda)$ est facilité si l'on remarque que

$$
u(\lambda) = P_N(e^{-i\lambda}e^{iA})v_0 = \sum_{j=0}^{N-1} \beta_j v_j e^{-ij\lambda} \quad \text{où} \quad v_j = e^{ijA}v_0, \quad j = 0, \ldots, N-1.
$$

Nous obtenons le développement en série de Fourier de la fonction u. Si nous calculons d'abord les N coefficients de Fourier $\beta_j v_j$ de la fonction u, la technique de la TFR permet d'évaluer plusieurs vecteurs $u(\lambda)$ à la fois. C'est un des intérêts de la transformation utilisée dans cette méthode.

Les vecteurs $v_j, j = 1, \ldots N-1$, dépendent d'un vecteur initial v_0 de $\mathbb{R}^n$. Remarquons que, pour $j \geq 1$,

$$\begin{aligned} v_{j+1} &= e^{iA}v_j = \cos(A)v_j + i\sin(A)v_j \\ &= 2\cos(A)v_j - \cos(-A)v_j - i\sin(-A)v_j \\ &= 2\cos(A)v_j - e^{-iA}v_j \\ &= 2\cos(A)v_j - v_{j-1}. \end{aligned} \tag{5.1.4}$$

Cette formule de récurrence permet d'éviter le calcul des puissances successives de e^{iA}.

Tous les vecteurs v_j à l'exception du vecteur initial v_0 sont complexes. Mais puisque l'on résout un problème de valeurs propres de matrices réelles symétriques, nous ne tiendrons compte que de la partie réelle des approximations $u(\lambda)$ des vecteurs propres. Pour $\lambda_l = \dfrac{l\pi}{N}$ et $l = 0, 1, \ldots, 2N-1$, définissons u_l par

$$u_l = \operatorname{Re}(u(\lambda_l)) = \operatorname{Re}\left(\sum_{j=0}^{N-1} \beta_j v_j e^{-ij\frac{l\pi}{N}}\right).$$

D'apr'es (5.1.1), nous avons aussi

$$u_l = \operatorname{Re}\sum_{k=1}^{n} \alpha_k P_N(e^{i(\lambda_k - \frac{l\pi}{N})}x_k).$$

Choix du Degré N

Le nombre de vecteurs construits u_l va dépendre de la précision souhaitée sur les vecteurs propres approchés.

Supposons que les valeurs propres de A soient distinctes. Soit λ_l la valeur propre de A la plus proche de $\dfrac{2l\pi}{N}$ vérifiant $|\lambda_l - \frac{2l\pi}{N}| \leq \delta\pi$. Alors,

$$\begin{aligned}
\|u_l - \alpha_l x_l\|_2 &= \| \operatorname{Re} \sum_{j=1}^{n} \alpha_j P_N(e^{i(\lambda_j - \frac{2l\pi}{N})}) x_j \ - \alpha_l x_l\|_2 \\
&\leq \| \sum_{\substack{j=1 \\ j\neq l}}^{n} \alpha_j \operatorname{Re}\Big(\ P_N(e^{i(\lambda_j - \frac{l\pi}{N})})\Big)\, x_j\|_2 + |\alpha_l|\ |\operatorname{Re}\Big(\ P_N(e^{i(\lambda_l - \frac{2l\pi}{N})})\Big) - 1|.
\end{aligned}$$

Or, si $\theta > \delta\pi$, d'après (5.1.2) et (5.1.3) $|\operatorname{Re}(P_N(e^{i\theta}))| \leq \kappa_N(\delta)$ et si $\|v_0\|_2 = 1$, alors

$$\| \sum_{\substack{j=1 \\ j\neq l}} \alpha_j \operatorname{Re}\Big(\ P_N(e^{i(\lambda_j - \frac{l\pi}{N})})\Big)\, x_j\|_2 \leq \kappa_N(\delta).$$

Par définition de P_N, $|\operatorname{Re}\Big(\ P_N(e^{i(\lambda_l - \frac{2l\pi}{N})})\Big) - 1|$ est proche de 0.

Ainsi,

$$\|u_l - \alpha_l x_l\|_2 = O(\kappa_N(\delta)).$$

Si l'on souhaite une précision ε sur les vecteurs propres approchés, nous choisirons N tel que $\kappa_N(\delta) \leq \varepsilon$.

Supposons qu'il existe plusieurs valeurs propres $\lambda_{k_1}, \ldots, \lambda_{k_r}$ de A telles que $|\lambda_{k_i} - \frac{2l\pi}{N}| \leq \delta\pi$. Alors, u_l approchera une combinaison linéaire des vecteurs propres $x_{k_1}, \ldots, x_{k_r}$ et de la même manière, nous aurons

$$\|u_l - \sum_{i=i_1}^{i_r} \alpha_{k_i} x_{k_i}\|_2 \simeq O(\kappa_N(\delta)).$$

5.1.3 Construction des Eléments Propres

Sélection et Raffinement

Nous avons maintenant construit les vecteurs

$$u_l \ = \ \operatorname{Re}(u(\lambda_l)) = \operatorname{Re}\left(\sum_{j=0}^{N-1} \beta_j v_j e^{ij\frac{l\pi}{N}}\right), \quad l = 0, \ldots, 2N-1.$$

Le but de cette partie est de proposer une méthode qui construit les éléments propres de A en tirant le plus d'informations possible des u_l calculés.

Pour cela, nous allons commencer par regrouper les u_l en clusters orthogonaux les uns aux autres et sélectionner les vecteurs les plus valables dans chacun des clusters.

Pour $k = 1, \ldots, n$ et avec $\lambda_k \in \mathrm{Sp}(A)$, notons I_k la suite d'entiers consécutifs définies par

$$I_k = \left\{ l, \text{ tels que } 0 \leq l \leq 2N-1, \text{ et } \left| \lambda_k - \frac{l\pi}{N} \right| \leq \delta\pi \right\}.$$

Si les valeurs propres λ_k et λ_{k+1} sont proches l'une de l'autre, voire multiples, alors $I_k \cap I_{k+1} \neq \emptyset$ et pour tout $l \in I_k \cup I_{k+1}$, u_l pourra s'écrire comme une combinaison linéaire des deux vecteurs propres x_k et x_{k+1}.

En pratique, les ensembles I_k ne peuvent être construits puisque les λ_k sont des inconnues. Par contre, en utilisant des informations sur les u_l, nous pouvons déterminer des réunions d'ensembles I_k. Pour un l fixé, remarquons que

$$|\lambda_k - \frac{l\pi}{N}| > \delta\pi, \quad k = 1, \ldots, n \quad \Rightarrow \quad \|u_l\|_2 = \|\sum_{j=1}^{n} \alpha_j P_N(e^{i(\lambda_k - \frac{j\pi}{N})}) x_j\|_2 \leq \kappa_N(\delta),$$

et par contraposition,

$$\|u_l\|_2 > \kappa_N(\delta) \quad \Rightarrow \quad \exists k, \quad 1 \leq k \leq n, \quad \left| \lambda_k - \frac{l\pi}{N} \right| \leq \delta\pi.$$

Nous pourrons considérer u_l comme une approximation d'un vecteur propre de A ou d'une combinaison linéaire de vecteurs propres de A si $\|u_l\|_2 > \kappa_N(\delta)$.

Soit $\varepsilon = c\kappa_N(\delta)$ un petit multiple de $\kappa_N(\delta)$, c désignant une constante entière. Nous allons donc regrouper les u_l en n_c clusters de la manière suivante :
le j^{ème} cluster sera défini par ses indices de début et fin $\{l_j^d, l_j^f\}$ tels que

$$\begin{aligned} \|u_l\|_2 > \varepsilon & \qquad \forall l \in \{l_j^d, l_j^f\}, \\ \|u_l\|_2 \leq \varepsilon & \qquad \text{si } l = l_{j-1}^d \text{ ou } l = l_{j+1}^f. \end{aligned}$$

Les valeurs de l telles que $\|u_l\|_2 \leq \varepsilon$ servent de séparateurs entre les différents clusters. Ces derniers sont orthogonaux entre eux à la précision ε. A l'intérieur de chaque cluster, nous ne retenons que les vecteurs dont la norme euclidienne atteint un maximum local et n'est pas trop proche de ε. Les vecteurs sont rangés par norme décroissante dans

$$W_j = [w_1^j, \ldots, w_{p_j}^j]$$

où p_j désigne le nombre de vecteurs retenus dans le cluster j. Notons p le nombre total de vecteurs retenus :

$$p = p_1 + \cdots + p_{n_c}.$$

Si $p < n$, il est nécessaire de compléter la base pour construire tous les éléments propres de A. Soit $W = [w_{p+1}, \ldots, w_{p_n}]$ un ensemble de $n-p$ vecteurs aléatoires. Nous orthonormalisons W de la manière suivante :

$$w_k^T w_l^j = 0 \quad \text{pour} \quad j = 1, \ldots, n_c, \quad l = 1, \ldots, p, \quad k = p+1, \ldots, n,$$

(orthonormalisation par rapport aux clusters déja construits.)

$$w_k^T w_l = \delta_{kl} \quad \text{pour} \quad k, l = p+1, \ldots, n,$$

(orthonormalisation de W).

et résolvons le problème de valeurs propres symétriques $W^T AW$ de taille $(n-p) \times (n-p)$. Nous obtenons

$$W^T AW = Q \begin{pmatrix} \tilde{\mu}_{p+1} & & \\ & \ddots & \\ & & \tilde{\mu}_n \end{pmatrix} Q^T.$$

Soit $\tilde{W} = WQ = [\tilde{w}_{p+1}, \ldots, \tilde{w}_{p_n}]$. Chacun de ces $n-p$ vecteurs est soit combiné avec l'un des n_c clusters soit laissé seul comme un vecteur propre individuel. Pour le vecteur $\tilde{w}_k$ nous définissons

$$\tilde{l}_k = \frac{N\tilde{\mu}_k}{\pi}.$$

S'il existe un entier j tel que $k_j^d \leq \tilde{l}_k \leq k_j^f$, alors $\tilde{w}_k$ appartient au $j^{ème}$ cluster. Dans le cas contraire, $(\tilde{\mu}_k, \tilde{w}_k)$ est un couple d'éléments propres de A. Cette situation arrive lorsque le vecteur initial v_0 n'a pas de composantes sur ce vecteur propre (ou la composante est négligeable). Nous noterons toujours p_j le nombre de vecteurs dans le j^{ème} cluster sachant que p_j a pu être augmenté.

Orthonormalisons chaque cluster W_j et résolvons pour chacun d'eux le problème de valeur propre réduit

$$W_j^T A W_j = Q_j \begin{pmatrix} \mu_1^j & & \\ & \ddots & \\ & & \mu_{p_j}^j \end{pmatrix} Q_j^T.$$

Alors, les vecteurs colonnes de $\tilde{W}_j = W_j Q_j$ sont les vecteurs propres de A dans ce cluster et les valeurs propres correspondantes sont $\mu_1^j, \ldots, \mu_{p_j}^j$.

Exemples

Illustrons cette méthode par un exemple de petite taille . Considérons la matrice tridiagonale de taille 4×4 provenant de la discrétisation du Laplacien avec un schéma aux différences finies à 5 points :

$$A = \begin{pmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{pmatrix}.$$

Les valeurs propres exactes de A sont

$$\lambda_k = 2 - 2\cos\frac{k\pi}{5}, \qquad k = 1, \ldots, 4,$$

et les vecteurs propres sont

$$x_i^{(k)} = \sin\frac{k\pi i}{5}, \quad 1 \leq i \leq 4.$$

Pour approximation des valeurs propres extrêmes de A, nous avons pris

$$a = 0.23552233703038 \qquad \text{et} \qquad b = 3.7644776629696.$$

La nouvelle matrice A translatée et réduite de manière à ce que $\mathrm{Sp}(A) \in [0, 2\pi)$, aura pour valeurs propres

$$\lambda_1 = 0.2607379674896,$$

$$\lambda_2 = 2.0412042743339,$$

$$\lambda_3 = 4.2419814573419,$$

$$\lambda_4 = 6.0224475519381.$$

Si $\delta = \dfrac{1}{n} = \dfrac{1}{4}$ et $\kappa_N(\delta) = 10^{-9}$, nous aurons besoin de prendre $N = 32$. Les ensembles

$$I_k = \left\{ l \text{ tel que } 0 \leq l \leq 2N-1 \text{ et } \left| \frac{l\pi}{N} - \lambda_1 \right| < \delta\pi \ [2\pi] \right\}$$

sont donnés par

$$\begin{aligned} I_1 &= \{0 \leq l \leq 11\} \cup \{60 \leq l \leq 63\}, \\ I_2 &= \{15 \leq l \leq 29\}, \\ I_3 &= \{36 \leq l \leq 50\}, \\ I_4 &= \{54 \leq l \leq 63\} \cup \{0 \leq l \leq 5\}. \end{aligned}$$

Nous nous trouvons dans le cas où $I_1 \cup I_4 \neq \emptyset$. Dans la partie sélection, nous devrions voir apparaître un cluster de taille 2 regroupant deux vecteurs qui engendrent le sous-espace invariant associéà $e^{i\lambda_1}$ et $e^{i\lambda_4}$, valeurs propres proches sur le cercle unité.

Nous avons exécuté l'algorithme sur cette matrice A en prenant pour vecteur initial v_0,

$$v_0 = (0.5, 0, 0.5, 0)^T$$

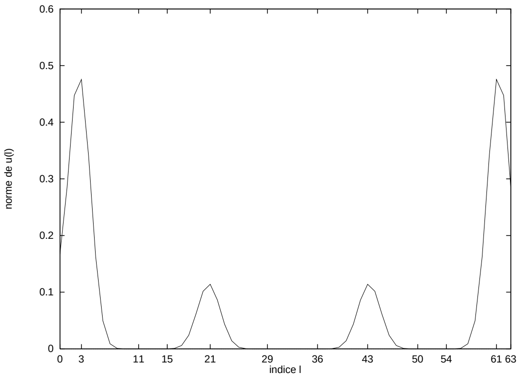

FIG. 5.1.2: Evolution de la norme de $u_l = P_N(e^{i\frac{l\pi}{N}}e^{iA})v_0$ en fonction de l.

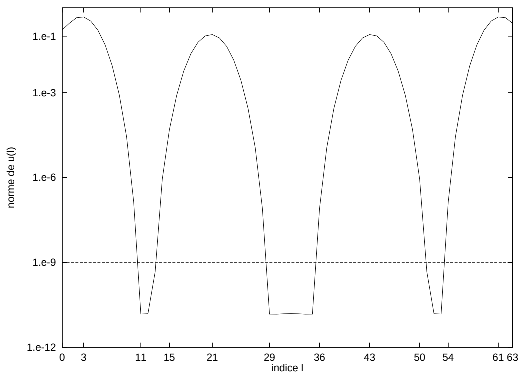

FIG. 5.1.3: Evolution de $\log_{10}\|u_l\|_2$. Positionnement par rapport à $\kappa_N(\delta)$.

et avec le degré N tel que $\kappa_N(\delta) \leq 10^{-9}$. Il est facile de vérifier que ce vecteur ne possède pas de composantes négligeables sur la base des vecteurs propres de A.

Les Figures 5.1.2 et 5.1.3 représentent l'évolution de la norme des vecteurs u_l, $l = 0, \ldots, 2N-1$. L'étape de sélection conduit à la formation de trois clusters marqués par les indices

$$0 \leq l \leq 10 \text{ et } 54 \leq l \leq 63 \quad \text{pour le cluster 1,}$$
$$14 \leq l \leq 28 \quad \text{pour le cluster 2,}$$
$$36 \leq l \leq 50 \quad \text{pour le cluster 3.}$$

La figure 5.1.3 montre les valeurs de l telles que $\|u_l\|_2 \leq 10^{-9}$ c'est-à-dire, les valeurs qui servent de séparateurs entre les différents clusters. Ces valeurs de l correspondent à ce que l'on avait prédit par la théorie (voir les valeurs des I_k). Les maxima locaux ont pour indices $l = 4, 62, 22, 44$, ce qui conduit à la formation des sous-espaces

$$W_1 = [u_4, u_{62}], \quad W_2 = [u_{22}] \quad W_3[u_{44}].$$

Remarquons que la norme des vecteurs u_4 et u_{62} est plus grande que celle des

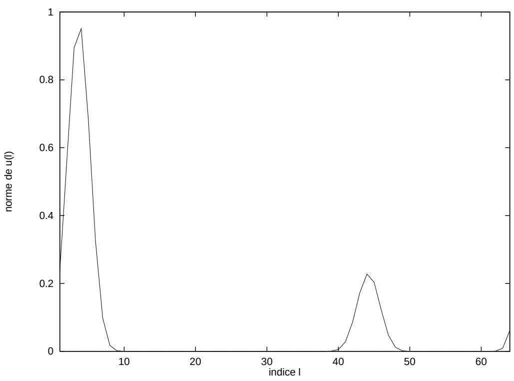

FIG. 5.1.4: Evolution de la norme de $u_l = P_N(e^{i\frac{l\pi}{N}} e^{iA})v_0$ en fonction de l .

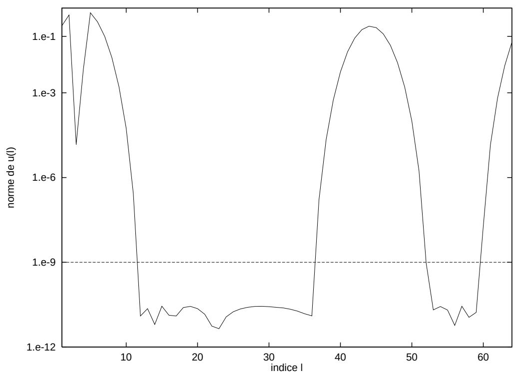

FIG. 5.1.5: Evolution de $\log_{10} \|u_l\|_2$. Positionnement par rapport à $\kappa_N(\delta)$.

vecteurs u_{22} et u_{44}. Les vecteurs u_4 et u_{62} sont combinaisons linéaires des vecteurs propres correspondant aux valeurs propres λ_1 et λ_4 et l'accélération polynômiale agit dans ces deux directions.

Finalement, il ne reste plus qu'à résoudre trois petits problèmes de valeurs propres de matrices symétriques :

$$A_1 = W_1^T A W_1, \quad A_2 = W_2^T A W_2, \quad A_3 = W_3^T A W_3$$

dont les solutions sont immédiates.

Le choix du vecteur initial est important. Si l'on prend

$$v_0 = \frac{1}{2}(1, 1, 1, 1)^T,$$

les composantes de v_0 sur les vecteurs propres x_1 et x_3 sont négligeables. Les Figures 5.1.4 et 5.1.5 montrent l'évolution de la norme des vecteurs u_l, $l = 0, \ldots, 2N-1$. L'étape de sélection conduit à la formation de 2 clusters marqués par les indices

$$0 \leq l \leq 11 \text{ et } 59 \leq l \leq 63 \quad \text{pour le cluster 1,}$$

$$36 \leq l \leq 50 \quad \text{pour le cluster 2.}$$

Les maxima locaux ont pour indices $l = 4, 44$, ce qui conduit à la formation des sous-espaces

$$W_1 = [u_4], \text{ et } W_2[u_{44}].$$

Nous devons alors complèter la base en générant deux vecteurs aléatoires $W = (w_1, w_2)$ que l'on orthonormalise avec u_4 et u_{44}. La résolution de $W^T AW$ conduit à la construction de deux couples d'éléments propres $(\tilde{\mu}_1, \tilde{w}_1)$ et $(\tilde{\mu}_2, \tilde{w}_2)$. Or, $E(\frac{N\tilde{\mu}_1}{\pi}) = 61$ donc le couple $(\tilde{\mu}_1, \tilde{w}_1)$ appartient au premier cluster. Pour le second couple, $E(\frac{N\tilde{\mu}_2}{\pi}) = 20$. C'est une valeur qui n'entre dans aucun cluster donc ce couple est considéré comme un couple d'éléments propres de A.

Chapitre 6

Algorithme Séquentiel

Les principales étapes de l'algorithme de Yau et Lu sont présentées dans le Tableau 6.0.1 ci-dessous et seront reprises dans la suite de ce chapitre.

I - Accélération polynômiale

a – Construire la matrice unitaire e^{iA}.
b – Construire les coefficients $\beta_j, j = 0, \ldots, N-1$ du polynôme P_N.
c – Construire les vecteurs $\beta_j v_j = \beta_j B^j v_0, j = 0, \ldots, N-1$.
d – Calculer les $u_l = P_N(e^{\frac{il\pi}{N}} e^{iA}) v_0 = \sum_{j=0}^{N-1} \beta_j v_j e^{ij\frac{l\pi}{N}}, j = 0, \ldots, N-1$.
à l'aide de la TFR.

II - Construction des éléments propres

a – Construire les sous-espaces invariants $W_i, i = 1, \ldots, n_c$.
– Regrouper les u_l en n_c clusters orthogonaux.
– Sélectionner dans chaque cluster les u_l les plus valables et construire les $W_i, i = 1, \ldots, n_c$.
– Compléter la base des sous-espaces invariants si nécessaire.
b – Résoudre dans chaque sous-espace W_i un problème de valeurs propres symétrique :

$$A_i = W_i^T A W_i, \quad A_i = Q_i^T D_i Q_i, \quad X_i = W_i Q_i$$

Tab. 6.0.1: Algorithme de Yau et Lu.

6.1 Accélération Polynômiale

6.1.1 Construction de la Matrice Unitaire e^{iA}

Un moyen d'approcher la fonction de matrice $f(A) = e^{iA}$ est d'utiliser une troncature de la série de Taylor.

Soit $e^{iz} = \sum_{k=0}^{\infty} (-i)^k \frac{z^k}{k!}$ le développement en série de Taylor de f. Alors,

$$\begin{aligned} e^{iA} &= X \operatorname{diag}\left(e^{i\lambda_1}, \ldots, e^{i\lambda_n}\right) X^T \\ &= X \operatorname{diag}\left(\sum_{k=0}^{\infty} (-i)^k \frac{\lambda_1^k}{k!}, \ldots, \sum_{k=0}^{\infty} (-i)^k \frac{\lambda_n^k}{k!}\right) X^T \\ &= X \sum_{k=0}^{\infty} (-i)^k \frac{D^k}{k!} X^T \\ &= \sum_{k=0}^{\infty} (-i)^k \frac{A^k}{k!} \end{aligned}$$

Pour une meilleure efficacité de l'algorithme, nous préférons une distribution plus ou moins uniforme des valeurs propres de e^{iA}. Pour cette raison, nous translatons et réduisons le spectre de A de manière à ce que $\mathrm{Sp}(A) \subset [0, 2\pi)$. Si a et b sont des approximations des valeurs propres extrêmes de A obtenues par une méthode quelconque (théorème de Gershgorin, méthode de Rojo et Soto [97]), ...), alors nous redéfinissons

$$A := \frac{2}{b-a}(A - aI).$$

Le théorème suivant borne les erreurs qui interviennent lorsque e^{iA} est approchée par une série de Taylor tronquée.

Théorème 10 *Soit $A \in \mathbb{R}^{n\times n}$ symétrique telle que $Sp(A) \in [0, 2\pi)$ et $Y = \frac{A}{\pi} - I$. Alors,*

$$\|e^{iA} - \sum_{k=0}^{q} (-i)^{k+1} \frac{\pi^k}{k!} Y^k\|_2 \leq \frac{n\pi^{q+1}}{(q+1)!}.$$

Preuve. Si $\mathrm{Sp}(A) \in [0, 2\pi)$ alors $\mathrm{Sp}(Y) \subset [-1, 1)$ et $e^{iA} = -e^{i\pi Y}$. Définissons les matrices $B(s)$ et $E(s)$ par

$$B(s) = -e^{i\pi Y s} = \sum_{k=0}^{q} (-i)^k \frac{\pi^k}{k!} A^k s^k + E(s), \quad 0 \leq s \leq 1. \tag{6.1.1}$$

L'élément $b_{ij}(s)$ est nécessairement analytique et

$$b_{ij}(s) = \sum_{k=0}^{q} \frac{b_{ij}^{(k)}(0)}{k!} s^k + \frac{b_{ij}^{(q+1)}(\varepsilon_{ij})}{(q+1)!} s^{q+1}, \tag{6.1.2}$$

où ε_{ij} est tel que $0 \leq \varepsilon_{ij} \leq 1$. En comparant les puissances de s en (6.1.1) et (6.1.2), nous concluons que

$$e_{ij}(s) = \frac{b_{ij}^{(q+1)}(\varepsilon_{ij})}{(q+1)!} s^{q+1}.$$

Mais $b_{ij}^{(q+1)}(s)$ est l'élément en position (i, j) de $(i\pi Y)^k\ e^{i\pi Y s}$ et ainsi,

$$\begin{aligned}
|e_{ij}(s)| &\leq \max_{0 \leq s \leq 1} \frac{b_{ij}^{(q+1)}(s)}{(q+1)!} \\
&\leq \max_{0 \leq s \leq 1} \frac{\|(i\pi Y)^{(q+1)}\ e^{i\pi Y s}\|_2}{(q+1)!} \\
&\leq \frac{\pi^{(q+1)} \|Y\|_2^{(q+1)}}{(q+1)!} \max_{0 \leq s \leq 1} \|e^{i\pi Y s}\|_2 \\
&\leq \frac{\pi^{(q+1)}}{(q+1)!}.
\end{aligned}$$

On en déduit que $$\|e^{iA} - \sum_{k=0}^{q} (-i)^{k+1} \frac{\pi^k}{k!} Y^k\|_2 \leq \frac{n\pi^{q+1}}{(q+1)!}. \qquad \square$$

Pour la construction de e^{iA}, nous utiliserons le développement en série de Taylor des fonctions paires $\cos \pi x$ et $\dfrac{\sin(\pi x)}{x}$. Nous obtenons

$$\begin{aligned}
\cos(A) &= \cos(\pi I + \pi Y) = \cos(\pi I)\cos(\pi Y) - \sin(\pi I)\sin(\pi Y) = -\cos(\pi Y) \\
&= \sum_{k=0}^{\infty} \frac{(\pi Y)^{2k}(-1)^k}{(2k)!}
\end{aligned}$$

D'après le théorème précédent, nous pourrons tronquer la série à $q = 28$ si la double précision est exigée. Ainsi,

$$\cos(A) \simeq -\sum_{k=0}^{14} (-1)^k c_{2k} Y^{2k}, \quad c_k = \frac{\pi^k}{k!}. \tag{6.1.3}$$

Il est préférable de réécrire cette somme sous la forme

$$\cos(A) \simeq -\sum_{k=0}^{3}(Y^8)^k B_k, \quad B_k = \begin{cases} c_{8k}I + c_{8k+2}Y^2 + \cdots + c_{8k+6}Y^6, & k = 0, 1, 2 \\ c_{8k}I + \cdots + c_{8k+4}Y^4, & k = 3, \end{cases}$$

pour avoir un processus de sommation optimal avec la règle de Horner. Ainsi, $\cos(A)$ peut être construit avec 7 produits matriciels au lieu de 13 si la règle de Horner était directement appliquée à (6.1.3). Cette technique est discutée dans Paterson et Stockmeyer [90].

Nous n'avons pas adopté l'approche de Yau et Lu. Ceux-ci utilisent un développement suivant les polynômes de Tchebycheff des fonctions paires $\cos \pi x$ et $\dfrac{\sin x}{x}$ pour $x \in [-1, 1]$. Cela demande le même nombre de multiplications matricielles et les coefficients du développement sont à calculer par intégration numérique [94].

6.1.2 Construction du Polynôme P_N

Pour le degré du polynôme P_N défini au paragraphe 5.1.2, nous choisissons le plus petit entier N puissance de 2 (à cause de la TFR) satisfaisant

$$\kappa_N(\delta) = \frac{1}{T_{N-1}\left(\dfrac{3-\cos(\delta\pi)}{1+\cos(\delta\pi)}\right)} \leq \varepsilon,$$

où ε désigne la précision souhaitée sur les vecteurs propres et

$$T_{N-1}(x) = \frac{1}{2}\left((x + \sqrt{x^2} - 1)^k + (x + \sqrt{x^2} - 1)^{-k}\right), \quad \text{pour} \quad |x| > 1.$$

Les coefficients $\beta_j, j = 0, N-1$ du polynôme P_N sont obtenus par les relations de récurrence suivantes :

$\beta_0^{(0)} = 1,$

$\beta_0^{(1)} = b\ ,\ \beta_1^{(1)} = a,$

pour $k = 2, \ldots, N-1$:

$$\beta_0^{(k+1)} = a\beta_1^{(k)} + 2b\beta_0^{(k)} - \beta_0^{(k-1)},$$

$$\beta_1^{(k+1)} = a\left(2\beta_0^{(k)} + \beta_2^{(k)}\right) + 2b\beta_1^{(k)} - \beta_1^{(k-1)},$$

$$\beta_j^{(k+1)} = a\left(\beta_{j-1}^{(k)} + \beta_{j+1}^{(k)}\right) + 2b\beta_j^{(k)} - \beta_j^{(k-1)} \text{ pour } j > 1,$$

$$\beta_j^{(k+1)} = 0 \text{ pour } j > k+1.$$

où

$$a = \frac{2}{1+\cos(\delta\pi)}, \qquad b = \frac{1-\cos(\delta\pi)}{1+\cos(\delta\pi)}.$$

Finalement, nous obtenons les coefficients $\beta_0, \beta_1, \ldots, \beta_{N-1}$ du polynôme P_N, en multipliant $\beta_0^{(N-1)}, \beta_1^{(N-1)}, \ldots, \beta_{N-1}^{(N-1)}$ par $\kappa_N(\delta)$. Cette construction, de nature vectorisable, s'effectue en $O(N^2)$ opérations de manière efficace sur machines vectorielles et parallèles.

6.1.3 Construction des Vecteurs $v_j = e^{ijA}v_0$

Soit v_0 un vecteur quelconque de $\mathbb{R}^n$. Le vecteur suivant,

$$v_1 = \cos(A)v_0 + i\sin(A)v_0,$$

est facilement obtenu grâce à la construction de $\cos A = -\cos(\pi X)$. La partie imaginaire est basée sur le développement de

$$\sin(A)v_0 = -\sin(\pi Y)v_0 = -Y\sin(\pi Y)Y^{-1}v_0.$$

La fonction $\dfrac{\sin(\pi x)}{x}$ est paire et $\sin(\pi Y)Y^{-1}$ partage les mêmes puissances de Y que $\cos(\pi Y)$. Nous ne construirons pas explicitement la matrice $Y\sin(\pi Y)Y^{-1}$ mais construirons directement la partie imaginaire du vecteur v_1 par des produits matrice-vecteur :

$$v_{temp} = -\sin(\pi Y)Y^{-1}v_0 = -\sum_{k=0}^{14}(-1)^k c_{2k+1}Y^{2k}v_0, \qquad Im(v_1) = Yv_{temp}.$$

Les $N-2$ vecteurs v_j restants sont construits par la formule de récurrence

$$v_{j+1} = 2\cos(A)v_j - v_{j-1} \tag{6.1.4}$$

valable pour $j \geq 1$. Cette construction s'effectue en environ $16n^3$ opérations, soit un coût du même ordre de grandeur que celui de la réduction orthogonale d'une matrice symétrique en une matrice tridiagonale (méthode de Lanczos par exemple).

Dans le but d'utiliser l'efficacité des BLAS de niveau 3, nous reformulons l'algorithme précédent de manière à faire apparaître des produits matrice-matrice. Pour cela, nous utilisons la relation suivante, valable pour $p \geq 1$ et j tel que $1 \leq j \leq 2^{p-1}$:

$$v_{2^{p-1}+j} = 2C_p v_j - \overline{v_{2^{p-1}-j}}, \qquad \text{où } C_p = C_{p-1}^2 - I, \quad C_1 = \cos(A).$$

Cette relation n'est pas triviale. Elle est obtenue par récurrence sur p et utilise la relation (6.1.4). Nous obtenons le nouvel algorithme :

étape 0 : $v_1 = \cos(A)v_0 + i\sin(A)v_0$

étape 1 : $C_1 = \cos(A)$

$v_2 = 2C_1v_1 - v_0$

étape 2 : $C_2 = 2C_1^2 - I$

$(v_3, v_4) = 2C_2(v_1, v_2) - \overline{(v_1, v_0)}$

étape 3 : $C_3 = 2C_2^2 - I$

$(v_5, v_6, v_7, v_8) = 2C_3(v_1, v_2, v_3, v_4) - \overline{(v_3, v_2, v_1, v_0)}$

$\vdots \qquad \vdots$

$\vdots \qquad \vdots$

étape p : $C_p = 2C_{p-1}^2 - I$

$(v_{\frac{N}{2}+1}, \dots, v_N) = 2C_p(v_1, \dots, v_{\frac{N}{2}}) - \overline{(v_{\frac{N}{2}-1}, \dots, v_0)}$

Tous les vecteurs v_j sont obtenus en $p = \log_2(N)$ étapes.

6.1.4 Evaluation des u_l par la TFR

Les vecteurs u_l définis par

$$u_l = Re\left(\sum_{j=0}^{N-1} \beta_j v_j e^{\frac{-ijk\pi}{N}}\right), \qquad k = 0, 1, \dots, 2N-1,$$

sont évalués par la TFR. Cet algorithme attribué à [27] Cooley et Tukey découpe récursivement la suite à transformer en sous-suites d'indices pairs et impairs permettant de ramener la complexité à un ordre de grandeur en $O(m\log_2(m))$ au lieu de $O(m^2)$, si m est le nombre d'éléments de la suite.

Les implémentations usuelles de la TFR construisent m transformées de Fourier à partir d'une suite de m valeurs. Puisque nous ne sommes intéressés que par la partie réelle des u_l, nous voulons construire $2N$ vecteurs réels u_l à partir de N vecteurs complexes $\beta_l v_l$ et recouvrir l'espace mémoire de la matrice $V = [v_0, v_1, \ldots, v_{N-1}]$ par $U = [u_0, u_1, \ldots, u_{2n-1}]$. Pour contourner cette difficulté, nous avons dupliqué les données après l'étape de réordonnancement (bit reversal) dans la TFR et ne stockons que les $2N$ parties réelles obtenues. La construction de la matrice U s'effectue en $O(nN\log_2(N))$ opérations.

6.2 Construction des Eléments Propres

6.2.1 Sélection

La sélection commence par le calcul de la norme de tous les vecteurs u_l, $\quad l = 0, 1, \ldots, 2N-1$ obtenus par TFR en environ $2Nn$ multiplications et additions.

Soit ε_κ un petit multiple de

$$\kappa_N(\delta) = \frac{1}{T_{N-1}\left(\dfrac{3-\cos(\delta\pi)}{1+\cos(\delta\pi)}\right)}.$$

Les vecteurs $u_l, l = 0, 1, \ldots, 2N-1$ sont regroupés en n_c clusters $\{k_j^d, k_j^f\}$ pour $j = 1, 2, \ldots, n_c$ tels que :

$$\begin{aligned} \|u_k\|_2 &> \varepsilon_\kappa && \text{pour } k \in \{k_j^d, k_j^f\} \\ \|u_k\|_2 &\leq \varepsilon_\kappa && \text{sinon} \end{aligned}$$

Dans chaque cluster, ne conserver que les vecteurs dont la norme 2 atteint un maximum local et les classer par norme décroissante. Regrouper les premiers et derniers clusters s'ils ne sont pas orthogonaux ($nc := nc - 1$).

6.2.2 Construction de W

Soit $p = p_1 + p_2 + \cdots + p_{n_c}$, le nombre total de vecteurs retenus. Si $p < n$, générer $n - p$ vecteurs aléatoires et les orthonormaliser avec les colonnes de

$[W_1, W_2, \ldots, W_{n_c}]$ pour obtenir $W = [W_{p+1}, W_{p+2}, \ldots, W_n]$. Résoudre le problème de valeurs propres symétrique de taille $(n-p) \times (n-p)$:

$$W^T A W = Q \begin{pmatrix} \tilde{\mu}_{p+1} & & & \\ & \tilde{\mu}_{p+2} & & \\ & & \ddots & \\ & & & \tilde{\mu}_n \end{pmatrix} Q^T,$$

en utilisant n'importe quel algorithme de résolution de problème de valeurs propres symétrique (`xSYEVX` de LAPACK par exemple). Soient $\tilde{w}_{p+1}, \tilde{w}_{p+2}, \ldots, \tilde{w}_{p+n}$, les $n-p$ vecteurs colonnes de la matrice WQ. Pour $l = p+1, p+2, \ldots, n$, s'il existe un entier j tel que $k_j^d \leq \frac{N\tilde{\mu}_l}{\pi} \leq k_j^f$, alors ajouter le vecteur $\tilde{w}_l$ au $j^{ème}$ cluster. Nous aurons $w_{p_j+1}^j := \tilde{w}_l$, $p_j := p_{j+1}$.
Dans le cas contraire, $(\tilde{\mu}_l, \tilde{w}_l)$ est considéré comme un couple d'éléments propres de A.

Pour chaque cluster, si $p_j > 1$, déterminer les éléments propres de la matrice réelle symétrique de taille $p_j \times p_j$:

$$A_j = W_j A W_j^T = Q_j \begin{pmatrix} \mu_1^j & & & \\ & \mu_2^j & & \\ & & \ddots & \\ & & & \mu_{p_j}^j \end{pmatrix} Q_j^T,$$

Alors, μ_i^j sera une valeur propre de A et un vecteur propre unitaire associé sera donné par la $i^{ème}$ colonne de la matrice unitaire $W_j Q_j$. Si $p_j = 1$, la valeur propre du $j^{ème}$ cluster est raffinée en calculant le quotient de Rayleigh : ${w_1^j}^T A w_1^j = \mu_1^j$.

Il n'est pas possible d'estimer le coût du calcul de la partie raffinement dans la mesure où le travail à effectuer dépend directement du problème initial.

Si les valeurs propres sont très voisines les unes des autres, et si v_0 n'a pas de composantes sur certains clusters, le nombre de vecteurs aléatoires à rajouter

pour compléter la base est important. Une orthonormalisation par factorisation QR Householder coûte cher : $O(n^3)$ opérations.
L'algorithme QL avec shifts implicites nécessite quant à lui $3n^3$ opérations pour la construction des valeurs propres et des vecteurs propres. Cependant, dans la pratique, la taille des sous-problèmes à résoudre est petite par rapport à n.

6.3 Complexité

Pour présenter le coût des opérations, nous suivons Golub et Van Loan [49, p. 19]. Par exemple, élever une matrice d'ordre n au carré prend $2n^3$ opérations.

La construction de la matrice $\cos A$ requiert 7 produits matrice-matrice soit $7n^3$ opérations puisque ce sont des matrices symétriques et commutantes. Nous ne construisons pas $\sin A$ puisque seulement le calcul de $\sin Av_0$ est nécessaire. La partie la plus coûteuse de l'algorithme est la construction des vecteurs v_j, $j = 1, N-1$ nécessaires pour la TFR. Si l'on utilise la formule de récurrence (6.1.4), la construction de chaque vecteur v_j nécessite une multiplication entre une matrice réelle symétrique et un vecteur complexe, soit $4Nn^2$ opérations. Si l'on réarrange les calculs comme présentés au §6.1.3, nous avons besoin de $\log_2(N)-1$ opérations matricielles supplémentaires. En contre partie, le calcul des v_j est réalisé à l'aide de produits matrice-matrice c'est-à-dire, des opérations de niveau 3 permettant une meilleure efficacité et un plus grand potentiel pour le parallélisme.

La construction des u_l est très efficaces parce que la TFR peut être faite en $nN\log_2(N)$ opérations. Pour la construction des éléments propres, le calcul des vecteurs u_l demande $2Nn^2$ opérations.

Si la base est incomplète, nous effectuons une factorisation QR qui nécessite $8n^3/3$ opérations puisque Q est formée explicitement.

La dernière étape formant les n_c à travers la transformation W^TAW nécessite 2 multiplications matricielles et la mise à jour des vecteurs propres $2n^3$ opérations

BLAS	Nb. ref. mémoire	Nb. op.	Reuse ratio
niveau1	$3n$	$2n$	$3/2$
niveau2	$mn + n + 2m$	$2mn$	2
niveau3	$2mn + mk + kn$	$2mnk$	$n/2$

TAB. 6.3.1: Paramètre de granularité des différents niveaux de BLAS pour des matrices de taille $m \times n, n \times k, m \times k$.

supplémentaires.

Le nombre total d'opérations est donc

$$\left(\frac{38}{3} + \log_2(N)\right) n^3 + 6Nn^2 + \log_2(N)Nn.$$

Bien sûr, cet algorithme est quatre à cinq fois plus coûteux que l'algorithme QR pour des matrices symétriques. Cependant, remarquons que les étapes a) et c) de la première partie de l'algorithme (voir Table 6.0.1) sont toutes des produits matrice-matrice. Il est facile de montrer que

$$\frac{\text{nb. total d'op. impliquant des produits matriciels}}{\text{nb. total d'op}} \gtrsim \frac{28 + \log_2(N)}{38 + \log_2(N)}.$$

Cet algorithme est motivé par le fait que les opérations matricielles peuvent être implémentées de manière plus efficace que les opérations matrice-vecteur ou vecteur-vecteur. Le "reuse ratio" défini par le rapport entre le nombre de flops et la taille des références mémoire, borne les performances (voir TAB 6.3.1).

Bien que le nombre d'opérations de cet algorithme soit plus grand que celui des méthodes standard passant par la tridiagonalisation, son implémentation sur machines parallèles peut s'avérer intéressante.

Dans le cadre du projet PRISM, [17], Auslander et Tsao [4] proposent un algorithme pour approcher les sous-espaces invariants d'une matrice à travers la construction d'un polynôme de matrice possédant des propriétés spéciales (annihilateur de sous-espaces invariants de rang r). Cette méthode est plus coûteuse que la méthode de Yau et Lu. Le nombre d'opérations est chiffré à

$$\frac{1}{3}n^3\left(16N + \frac{176}{3}\right) \text{ avec } N \simeq 25.$$

6.4 Commentaires

Pour la construction de la base de sous-espaces invariants (voir §6.2.2), nous pouvons utiliser

$$\min_{Q} \left\{ \|W - Q\|_F : Q^T Q = I \right\}$$

dont la solution optimale Q_{opt} est donnée par la décomposition polaire de W. La matrice orthogonale Q_{opt} peut être construite en utilisant un schéma riche en produits matrice-matrice (voir Higham et Schreiber [58] et Philippe [92]).

Il est très difficile de faire une analyse rigoureuse de la stabilité de cet algorithme. Il est cependant montré [57] que chacune de ses composantes est numériquement stable : évaluation d'un polynôme par la méthode de Horner, produits matriciels, transformée de Fourier rapide. Il est difficile de trouver la matrice dont on a effectivement résolu le problème. Yau et Lu tentent de discuter des effets de la précision finie. Ils ne fournissent aucun résultat significatif en terme d'analyse d'erreur et stabilité inverse. Nous avons validé la précision de cet algorithme par une série de tests numériques.

Chapitre 7

Résultats Numériques

Puisqu'il est très difficile de faire une étude théorique globale de la stabilité de la méthode de Yau et Lu, il est nécessaire d'insister sur les tests numériques. Les résultats présentés par Yau et Lu ne sont pas assez nombreux pour être convainquants et justifier un travail d'implémentation parallèle.

Dans ce chapitre, nous nous sommes principalement intéressés à la précision des résultats fournis par notre algorithme. Au paragraphe 7.1, nous montrons que si le résidu et le défaut d'orthogonalité sont petits, alors la décomposition spectrale construite est la décomposition spectrale exacte d'une petite perturbation de A. Malheureusement, nous n'avons pas de bornes théoriques pour le résidu et le défaut d'orthogonalité associé à notre algorithme. Pour valider numériquement cet algorithme, nous avons dû mesurer ces deux types d'erreurs pour un certain nombre de matrices décrites au paragraphe 7.2. Nous discutons des résultats obtenus au paragraphe suivant.

7.1 Mesure de la Précision

Pour mesurer la précision des éléments propres construits par un algorithme, il y a deux types d'erreurs intéressantes. Nous noterons e_{res} le résidu normalisé

$$e_{res} = \max_i \frac{\|A\widehat{x}_i - \widehat{\lambda}_i \widehat{x}_i\|_2}{|\widehat{\lambda}_{\max}|}, \quad \|\widehat{x}_i\|_2 = 1,$$

où les $\widehat{\lambda}_i$ sont les valeurs propres construites et les $\widehat{x}_i$, les vecteurs propres construits correspondants. Nous noterons e_{orth} le défaut d'orthogonalité

$$e_{orth} = \max_i \|(\widehat{X}\widehat{X}^T - I)e_i\|_2$$

où $\widehat{X} = [\widehat{x}_1, \ldots, \widehat{x}_n]$ est la matrice des vecteurs propres construits.

Le théorème suivant montre que si le résidu et le défaut d'orthogonalité sont petits, alors les valeurs propres construites sont les valeurs propres exactes d'une petite perturbation de A.

Théorème 11 *Soit A une matrice symétrique et $\widehat{D} = diag(\widehat{\lambda}_1, \ldots, \widehat{\lambda}_n)$, $\widehat{X} = (\widehat{x}_1, \ldots, \widehat{x}_n)$ sa décomposition spectrale construite par un algorithme. Alors, il existe une matrice orthogonale $Q \in \mathbb{R}^{n\times n}$ telle que*

$$A + \Delta A = Q\widehat{D}Q^T$$

avec

$$\|\Delta A\|_2 \leq \sqrt{n}\ |\widehat{\lambda}_{\max}|\ e_{res}(1 + e_{orth}) + (|\lambda_{\max}| + |\widehat{\lambda}_{\max}|)(2\sqrt{n}\ e_{orth} + n\ e_{orth}^2).$$

Preuve. Un résultat de Higham [57, p. 389] montre que

$$\min_Q \left\{\|\widehat{X} - Q\|_2, QQ^T = I\right\} \leq \|XX^T - I\|_2,$$

et que ce minimum est atteint pour Q égale au facteur orthogonal de la décomposition polaire de $\widehat{X}$. En définissant Q cette manière, nous avons

$$\widehat{X} = Q + E, \qquad \|E\|_2 \leq \|XX^T - I\|_2 \leq \sqrt{n}\ e_{orth}$$

car $\|XX^T - I\|_2 \leq \sqrt{n}\ \max_i \|(XX^T - I)e_i\|_2 = sqrtne_{orth}$.

Définissons

$$r_i = A\widehat{x}_i - \widehat{\lambda}_i\widehat{x}_i \quad \text{et} \quad R = [r_1, \ldots, r_n].$$

Alors,

$$r_i^T\widehat{x}_j = \widehat{x}_i^T A\widehat{x}_j - \widehat{\lambda}_i\widehat{x}_i^T\widehat{x}_j,$$

si bien que

$$\begin{aligned} R\widehat{X} &= \widehat{X}^T A\widehat{X} - \widehat{D}\widehat{X}^T\widehat{X} \\ &= (Q+E)^T A(Q+E) - \widehat{D}(Q+E)^T(Q+E) \\ &= Q^T AQ - \widehat{D} + \Delta, \end{aligned} \tag{7.1.1}$$

où

$$\Delta = Q^T AE + E^T AQ + E^T AE - \widehat{D}(E^T Q + Q^T E + E^T E).$$

Nous avons

$$\begin{aligned} \|\Delta\|_2 &\leq 2\|A\|_2\|E\|_2 + \|E\|_2^2\|A\|_2 + \|\widehat{D}\|_2\left(2\|E\|_2 + \|E\|_2^2\right) \\ &\leq \left(\|A\|_2 + \|\widehat{D}\|_2\right)\left(2\|E\|_2 + \|E\|_2^2\right). \end{aligned}$$

De (7.1.1), nous obtenons

$$\|Q^T AQ - \widehat{D}\|_2 \leq \|R\|_2\|\widehat{X}\|_2 + \|\Delta\|_2.$$

Or,

$$\|R\|_2 \leq \sqrt{n}\ \max_i \|A\widehat{x}_i - \widehat{\lambda}_i\widehat{x}_i\|_2 = \sqrt{n}|\widehat{\lambda}_{\max}|\ e_{res},$$

donc

$$\begin{aligned} \|Q^T AQ - \widehat{D}\|_2 &\leq \sqrt{n}|\widehat{\lambda}_{\max}|\ e_{res}\|\widehat{X}\|_2 + \|\Delta\|_2 \\ &\leq \sqrt{n}|\widehat{\lambda}_{\max}|\ e_{res}\,(1 + e_{orth}) + \left(\|A\|_2 + \|\widehat{D}\|_2\right)(2\sqrt{n}\ e_{orth} + n\ e_{orth}^2) \end{aligned}$$

comme annoncé. $\square$

Ce théorème montre que e_{res} et e_{orth} bornent l'erreur régressive sur les valeurs propres construites λ_i.

De plus, un résultat standard de la théorie des perturbations dit que si $\widehat{\lambda} \in \mathrm{Sp}(A+E)$, alors il existe un $\lambda \in \mathrm{Sp}(A)$ tel que [49, cor. 8.1.6]

$$|\lambda - \widehat{\lambda}| \leq \|\Delta A\|_2.$$

Ainsi, la précision des valeurs propres construites est aussi bornée en fonction des quantités e_{res} et e_{orth}.

7.2 Matrices Test

Pour tester la stabilité et la précision de notre algorithme, nous avons utilisé les matrices test symétriques suivantes.

Type 1 : Matrices à coefficients aléatoires uniformément distribués sur $[-1, 1]$.

Type 2 : Matrices générées par la routine `DLATMS` de LAPACK [3] et [57, p. 299]. Ces matrices sont de la forme UDU^T où U est une matrice orthogonale aléatoire et D est une matrice diagonale choisie suivant un mode et un paramètre ρ :

mode=3 la matrice D est définie par

$$D(i) = \rho^{\frac{i-1}{n-1}}.$$

Distribution géométrique des valeurs propres de 1 à ρ.

mode=4 la matrice D est définie par

$$D(i) = 1 - \frac{i-1}{n-1}(1-\rho).$$

Distribution arithmétique des valeurs propres de 1 à ρ.

mode=5 la matrice D est définie de manière aléatoire telle que $D(i) \in [\rho, 1], i = 1, \ldots, n$, et le logarithme des $D(i)$ est uniformément distribué.

Type 3 : Matrices provenant de la discrétisation du laplacien sur un domaine carré en utilisant un schéma à cinq points. Soit m le nombre de points de discrétisation. Alors A est d'ordre $n = m^2$ et peut être mise sous la forme tridiagonale par blocs suivante :

$$A = \begin{bmatrix} G & -I & & \\ -I & G & \ddots & \\ & \ddots & \ddots & -I \\ & & -I & G \end{bmatrix},$$

où I est la matrice identité d'ordre m et G, d'ordre m aussi, est définie par

$$G = \begin{bmatrix} 4 & -1 & & 0 \\ -1 & 4 & \ddots & \\ & \ddots & \ddots & -1 \\ 0 & & -1 & 4 \end{bmatrix}.$$

Les éléments propres de A sont connus explicitement et la plupart des valeurs propres sont de multiplicité 2.

Type 5 : Matrice proposée par Yau et Lu :

$$A(i,j) = \sin(3000(i+j-2)^2)).$$

Cette matrice possède un spectre bien reparti (cf. FIG 7.2.1).

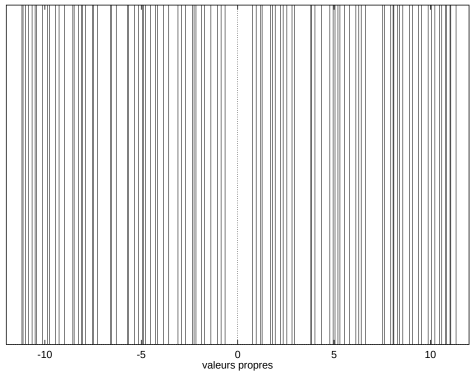

FIG. 7.2.1: Répartition du spectre de la matrice $A(i,j) = \sin(3000(i+j-2)^2))$.

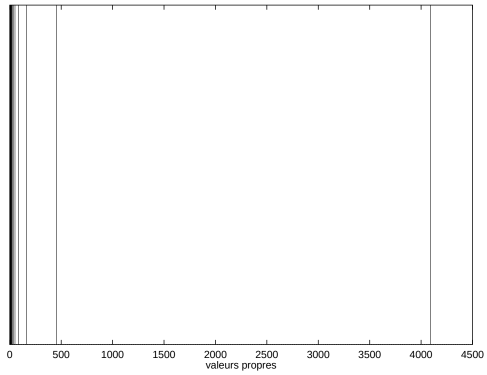

FIG. 7.2.2: Répartition du spectre de la matrice de type 6.

type 6 : Matrice de la forme

$$A = \begin{pmatrix} n & n-1 & n-2 & \cdots & 2 & 1 \\ n-1 & n-1 & \vdots & & \vdots & \vdots \\ n-2 & & & & \vdots & \vdots \\ \vdots & & & & & \\ 2 & 2 & & 2 & 2 & 1 \\ 1 & 1 & & 1 & 1 & 1 \end{pmatrix},$$

dont les valeurs propres, (voir fig 7.2.2,) très mal réparties, sont données par

$$\lambda_i = \frac{1}{2}\left(1 - \cos\frac{2i-1}{2n+1}\pi\right)^{-1}.$$

7.3 Tests Numériques

Tous les tests présentés dans ce paragraphe ont été effectués sur une station Sun SPARC 512 MP, en arithmétique IEEE double précision avec une précision machine $\mathbf{u} = 2^{-53} \approx 1.1 \times 10^{-16}$. Le code est écrit en Fortran.

Nous avons généré un très grand nombre de matrices aléatoires (type 1) et calculé pour chacune d'elles, e_{res} et e_{orth}. Les résultats sont présentés aux Figures

7.3.1 et 7.3.2. Les mesures du résidu et du défaut d'orthogonalité sont proches de la précision machine ce qui signifie que l'on a résolu un problème assez proche du problème initial.

Les matrices de type 2 (mode 2,3,4) et type 6 ont des valeurs propres très voisines les unes des autres. Nous conservons encore une bonne précision pour les résidus et défaut d'orthogonalité (voir Fig.7.3.3 et Fig.7.3.4).

La matrice issue de la discrétisation du laplacien est symétrique définie positive. Une factorisation de Cholesky de A (au préalable réduite sous forme tridiagonale) et suivie par des itérations QR bidiagonales, construit des résultats très précis pour ce type de matrices. Cet algorithme est disponible dans LAPACK sous le nom `DPTEQR`. Les figures 7.3.5 et 7.3.6 comparent la précision des résultats de cet algorithme avec le nôtre. Nos résultats sont seulement 10 fois moins précis que ceux obtenus par l'un des meilleurs algorithmes.

La taille et le nombre de sous-problèmes à résoudre dans la dernière partie de l'algorithme dépendent de la matrice initiale. Pour différents types de matrices, nous avons représenté la répartition des clusters suivant leur taille. Un spectre bien séparé conduit à de nombreux clusters de petite taille (Figures 7.3.8 et 7.3.9). La matrice du laplacien (type 3) possède des valeurs propres doubles. Pour elle, nous avons un grand nombre de clusters de dimension 2 (voir Figure 7.3.10). La Figure 7.3.11 correspond à une distribution géométrique du spectre. Le cluster de dimension 33 correspond aux valeurs propres de petite taille qui sont très proches les unes des autres.

Cet algorithme est mal adapté aux matrices du type 6. Par exemple, pour $n = 100$, l'algorithme repère les quatre clusters (voir Figure 7.2.2) mais ne conserve que 7 vecteurs propres. Il faut alors compléter la base avec 93 autres vecteurs et résoudre un problème de valeurs propres de taille 93. Le problème initial n'a pas été réduit !

Le choix du vecteur initial v_0 a une influence sur le nombre de vecteurs propres approchés par l'accélération polynômiale. Ce vecteur doit avoir des composantes non négligeables sur la base des vecteurs propres. Par exemple, les vecteurs propres de la matrice tridiagonale

$$A = \begin{pmatrix} 2 & -1 & & 0 \\ -1 & \ddots & \ddots & \\ & \ddots & \ddots & -1 \\ 0 & & -1 & 2 \end{pmatrix},$$

sont définis par

$$x_i^{(k)} = \sin \frac{k\pi i}{n+1}, \quad 1 \leq i \leq n.$$

Le vecteur initial v_0 choisi tel que

$$v_0 = \frac{1}{\sqrt{n}}(1, 1, \ldots, 1)^T,$$

a des composantes négligeables sur la base des vecteurs propres x_i. Si

$$v_0 = \alpha_1 x_1 + \alpha_2 x_2 + \cdots + \alpha_n x_n,$$

alors $\alpha_{2k} \ll 1$ pour $k = 1, \ldots, n/2$. Par contre, pour des choix de v_0 différents comme par exemple $v_0 = (1, 0, 1, \ldots)^T$, nous approchons presque tous les vecteurs propres.

Les temps d'éxécution varient suivant le type de BLAS utilisées. L'algorithme de Yau et Lu est beaucoup plus lent que l'algorithme QR pour matrice symétrique programmé dans LAPACK (voir Figure 7.3.12). Sur station IBM RISC Sys/6000-550, l'utilisation des BLAS optimisées permet de réduire le rapport temps d'éxécution de Yau et Lu par rapport au temps d'éxécution de QR. La figure 7.3.13 compare les temps d'éxécution de l'algorithme avec BLAS non optimisées (sur station SUN) et BLAS optimisées (sur station IBM RISC Sys/6000-550).

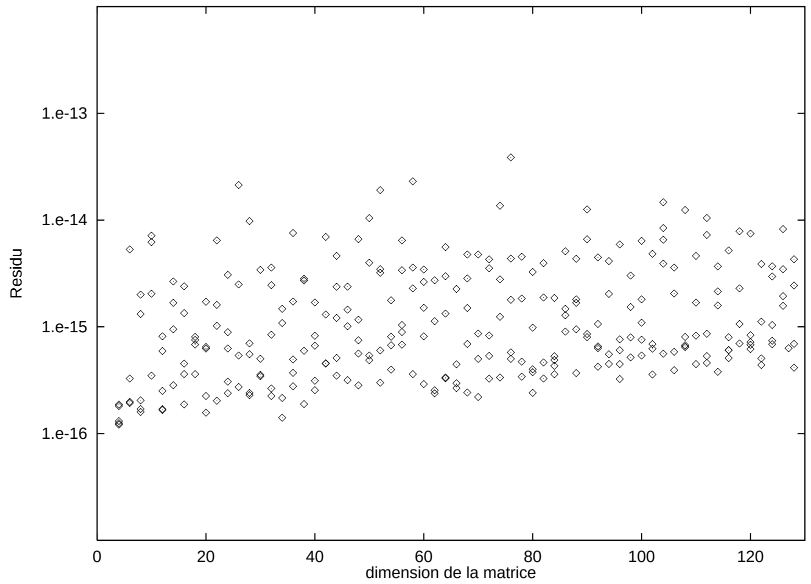

FIG. 7.3.1: Mesure du résidu pour des matrices aléatoires.

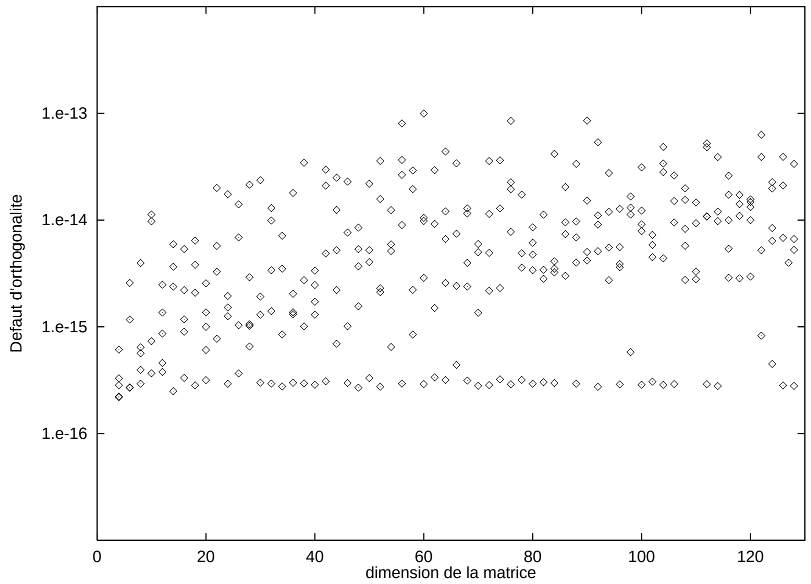

FIG. 7.3.2: Mesure du défaut d'orthogonalité pour des matrices aléatoires de type 1.

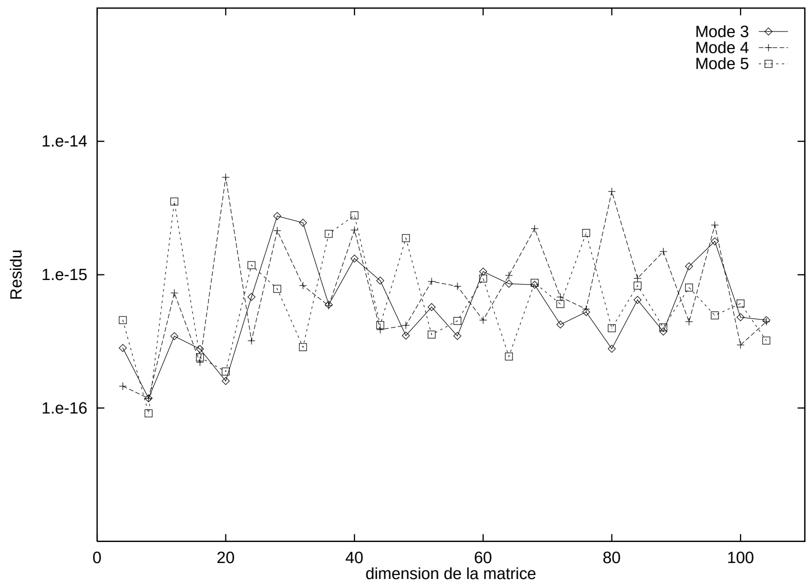

FIG. 7.3.3: Mesure du résidu pour des matrices du type 2.

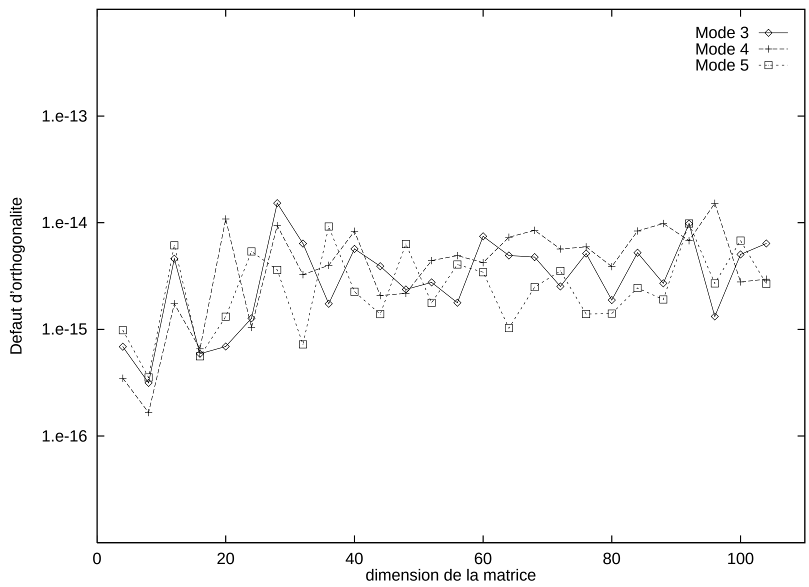

FIG. 7.3.4: Mesure du défaut d'orthogonalité pour des matrices du type 2.

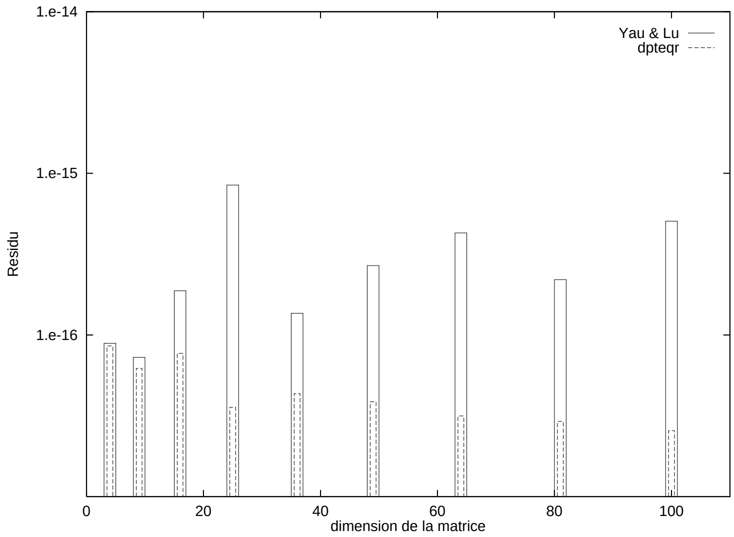

FIG. 7.3.5: Mesure du résidu. Comparaison entre Yau Lu et DPTEQR pour la matrice du laplacien.

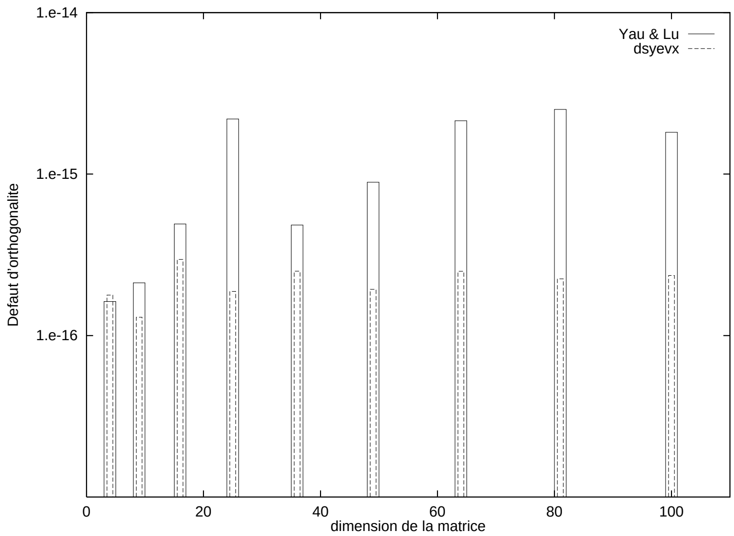

FIG. 7.3.6: Mesure du défaut d'orthogonalité. Comparaison entre Yau Lu et DPTEQR pour la matrice du laplacien.

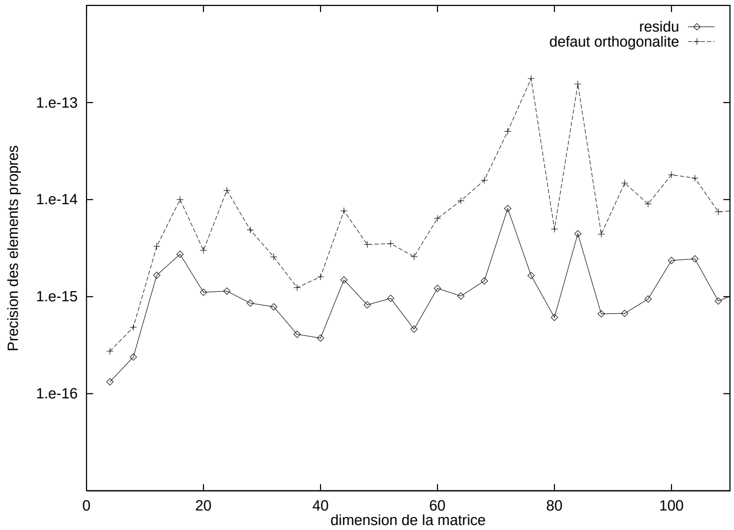

FIG. 7.3.7: Erreur résiduelle et défaut d'orthogonalité pour des matrices de type 5.

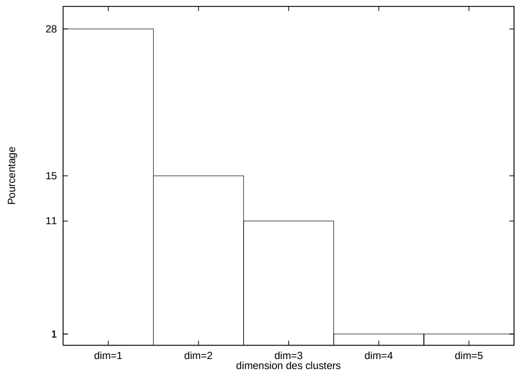

FIG. 7.3.8: Répartition des clusters pour une matrice aléatoire.

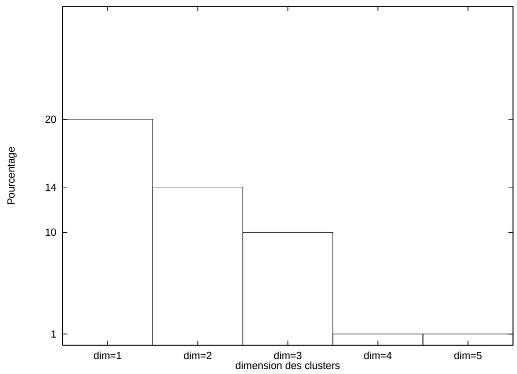

FIG. 7.3.9: Répartition des clusters pour une matrice de type 5

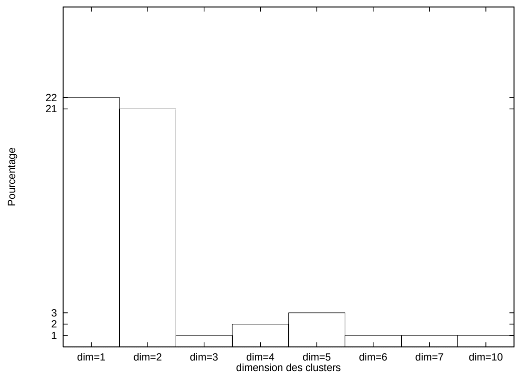

FIG. 7.3.10: Répartition des clusters pour la matrice du L laplacien.

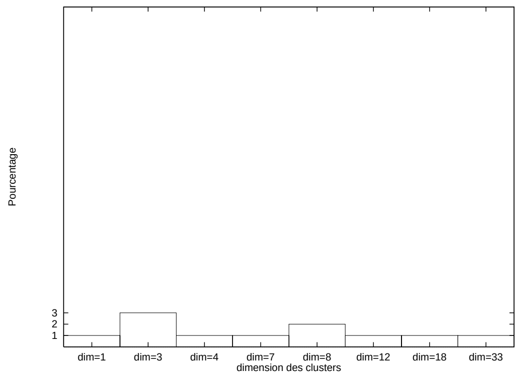

FIG. 7.3.11: Répartition des clusters pour une matrice de type 2, mode 3.

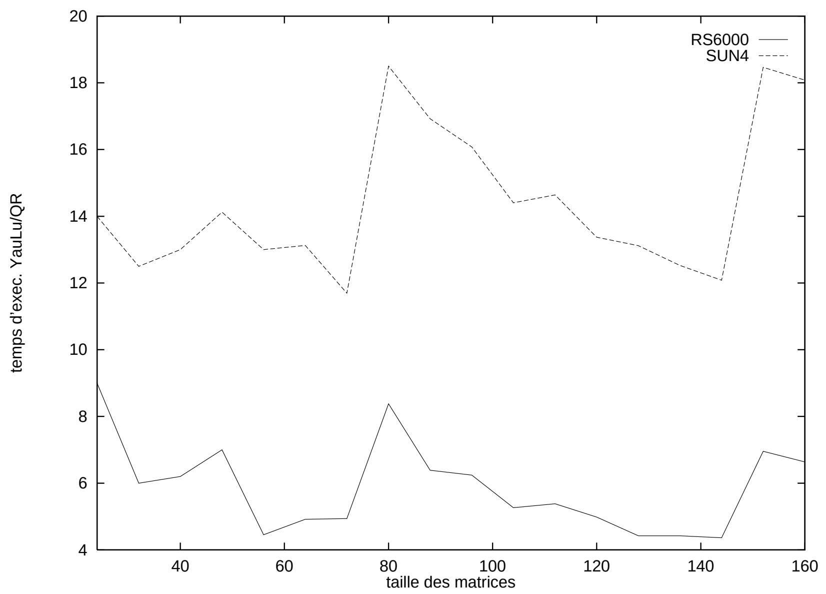

FIG. 7.3.12: Rapport temps d'execution de Yau et Lu / temps d'execution de QR.

7.4 Conclusion

Nous avons testé notre programme sur un grand nombre de matrices. Celui-ci construit des éléments propres à une précision proche de la précision machine. Il n'est pas très bien adapté aux matrices à spectre mal réparti. C'est un problème que rencontrent de nombreuses méthodes. Il serait intéressant de rappeler récursivement l'algorithme sur les clusters de grande taille.

Nous avons voulu comparer nos résultats en temps d'exécution avec ceux de l'algorithme de décomposition en sous-espaces invariants. C'est sans doute le type d'algorithme le plus proche de celui de Yau et Lu. L'ensemble de sous-routines fourni,[1] développé par le projet PRISM [17], est incomplet. Nous avons corrigé quelques "bugs" mais pas tous. Il n'est pas possible de le tester sur des matrices de taille supérieure à 68 ! (version de Novembre 1996). Des comparaisons de temps d'exécution pour des matrices si petites ne sont pas très significatives.

[1] La routine DGISDA est disponible par ftp anonymous à l'adresse `ftp.super.org` dans le répertoire `pub/prism`.

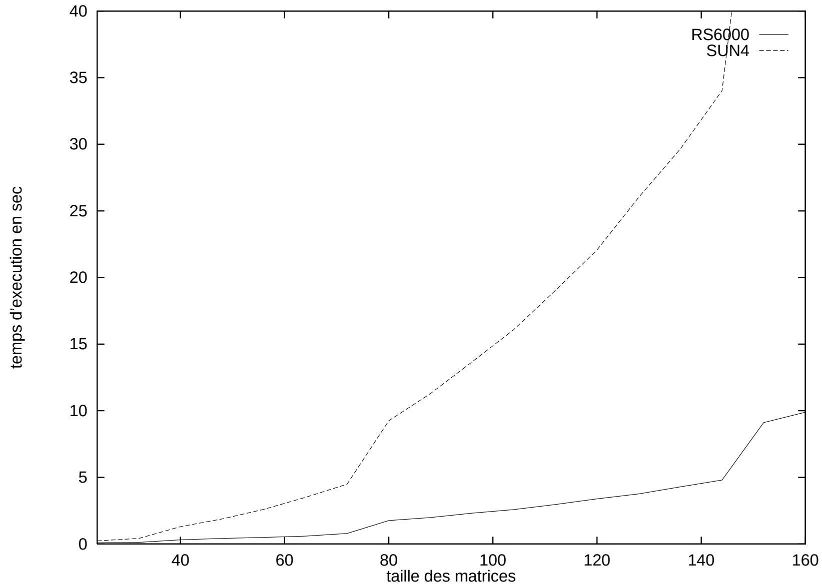

FIG. 7.3.13: Temps d'execution avec BLAS optimisées (RS6000) et BLAS non optimisées (SUN4).

Notre algorithme n'est pas concevable en séquentiel non pas par la précision des résultats qu'il fournit mais parce qu'il est plus coûteux en calculs que les méthodes traditionnelles. L'utilisation des BLAS de niveau 3 peut justifier ce surcoût. Cet algorithme possède un fort potentiel de parallélisme dû aux nombreux produits matriciels et à la réduction du problème initial en une série de sous-problèmes du même type. Aucune recherche du coté parallélisme n'a encore été faite, sur cette méthode. C'est l'objectif du chapitre suivant.

Chapitre 8

Implémentation parallèle

L'algorithme de Yau et Lu est construit autour d'un ensemble de modules manipulant des blocs de matrices et possédant un fort potentiel de parallélisme comme les produits matrice-matrice et la factorisation QR nécessaire pour compléter la base. Des implémentations efficaces de ces modules sont maintenant disponibles sur la plupart des machines.

L'efficacité d'un algorithme parallèle dépend largement du coût dû aux mouvements de données. Le coût dépend aussi bien de la distribution initiale des données sur les processeurs que de l'algorithme utilisé pour les communications inter-processeurs. Au premier paragraphe, nous décrivons la distribution des données que nous avons utilisée pour notre algorithme et expliquons ce choix.

Nous proposons ensuite une implantation parallèle de la méthode de Yau et Lu. L'efficacité de notre code repose sur l'efficacité des produits matrice-matrice. Pour cette raison, et parce qu'il n'existe pas encore de BLAS de niveau 3 parallèle adaptée à notre situation, nous avons développé notre propre produit matriciel parallèle pour des matrices symétriques qui commutent. Nous présentons les résultats obtenus sur la Paragon d'Intel pour ce nouveau produit matrice-matrice et pour le noyau principal de notre algorithme.

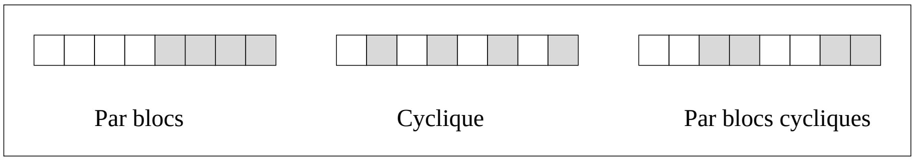

FIG. 8.1.1: Les modes de répartition par blocs, cyclique et par blocs cyclique de huit données sur deux processeurs.

8.1 Distribution

Décider quels seront les éléments de la matrice à assigner à tel ou tel processeur est une étape critique dans la mise en oeuvre d'un algorithme parallèle d'algèbre linéaire.

Nous avons choisi d'utiliser une distribution par *blocs cyclique* des données. Une étude à la fois théorique et pratique [53] a montré que ce type de distribution, pour les algorithmes d'algèbre linéaire dense,

– minimise les communications inter-processeurs,
– conduit à une meilleure scalabilité,
– diminue le temps de repos,
– offre une meilleure répartition de la charge du travail,
– diminue les dépassements de capacité de mémoire,

par rapport aux distributions par lignes ou colonnes.

La distribution par blocs cycliques est paramétrisée par quatre nombres P, Q, r et c. Les paramètres $P \times Q$ désignent la grille de processeurs et les paramètres $r \times c$, la taille des blocs de la matrice. Tous les blocs séparés par un pas fixé suivant les lignes et les colonnes sont assignés à un même processeur. La figure 8.1.2 illustre ce type de distribution dans le cas d'une grille carrée 2×2 de processeurs.

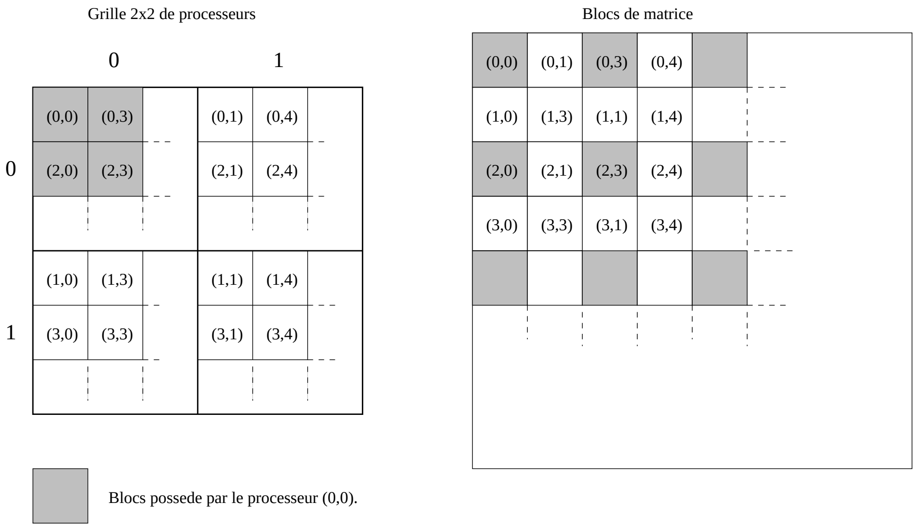

FIG. 8.1.2: Distribution par blocs cyclique sur une grille 2×2 de processeurs.

8.2 Opérations de communication

Notre algorithme emploie trois opérations de communication très connues [45].

La première opération est la communication *point à point* dans laquelle un processeur envoie un message à un autre. La communication point à point est une opération primitive sur tout ordinateur à mémoire distribuée (opération de bas niveau). Nous utilisons ce type de communication pour faire des échanges de données entre deux paires de processeurs.

La deuxième opération de communication utilisée est la *diffusion*. Un processeur envoie le même message à plusieurs autres processeurs (opération de haut niveau).

La troisième opération est la *somme globale* pour laquelle chaque processeur commence avec une valeur et termine avec la somme des valeurs de tous les autres processeurs (opérateur global).

Les deux dernières opérations peuvent être implantées de plusieurs manières et dépendent de l'architecture propre de la machine. Pour avoir un programme portable, nous faisons appel aux BLACS (Basic Linear Algebra Communication

Subroutines) qui sont maintenant implantées sur la plupart des machines parallèles et tiennent compte des spécificités de chacune d'elles. Les BLACS forment une bibliothèque efficace de routines de communications, adaptée au calcul matriciel. Elles ont pour modèle, une ou deux grilles de processus.

8.3 Algorithme de Yau et Lu parallèle

Il y a deux formes de parallélisme dans la méthode de Yau et Lu. La première est un *parallélisme de données* et correspond aux nombreux produits matrice-matrice. Le second est une sorte de *parallélisme fonctionnel*. Il correspond à la réduction du problème initial en une série de problèmes du même type, pouvant être résolus indépendamment les uns des autres sur chaque processeur. C'est pour ces raisons que nous disons que la méthode de Yau et Lu possède un fort potentiel de parallélisme.

Dans ce paragraphe, nous proposons une parallélisation de la méthode de Yau et Lu. Nous séparons les étapes d'accélération polynômiale et la construction des éléments propres.

8.3.1 Accélération polynômiale

Cette première partie est construite autour d'une série d'appels à des produits matrice-matrice et de la TFR.

Construction de la matrice $\cos A$

Paralléliser la construction de la matrice $\cos(A)$ n'est pas difficile. Si l'on dispose d'un produit matrice-matrice parallèle, il suffit de remplacer les appels produit matrice-matrice séquentiel dans l'algorithme présenté au paragraphe 6.1.1 par des appels produit matrice-matrice parallèle. Cependant, il est intéressant

```
/*Initialisation */
A → A_0, c_24 I → C
FOR i = 1 TO 4 DO
    PDSCMM ⇒ A_{2i-2} * A_{2i-2} → A_{2i}
ENDFOR
FOR i = 0 TO 2 DO
    PDMATAD ⇒ c_{24+2i} A_{2i} + C → A_{2i}
ENDFOR

/* Boucle principale */
FOR i = 2 DOWNTO 0 DO
    C_{8i} I → B
    FOR j = 1 TO 3 DO
        PDMATAD ⇒ c_{8i+2j} A_{2j} + B → B
        PDSCMM ⇒ A_8 * C + B → C
    ENDFOR
ENDFOR
```

TAB. 8.3.1: Algorithme de construction de $C = \cos(A)$.

de remarquer que cette construction fait intervenir des produits matrice-matrice entre des puissances de A. Or, la matrice A est symétrique et commute avec ses puissances. Cela signifie que tous les produits matrice-matrice utilisés doivent fournir, en résultat, une matrice symétrique. Il n'est donc pas nécessaire de faire tous les calculs mais seulement la moitié. Dans l'algorithme présenté en Table 8.3.1, le nom `PDSCMM` désigne une routine parallèle possédant ces propriétés. Son implantation sera présentée, plus en détail, au paragraphe 8.4.

Construction des vecteurs de Fourier

La construction des vecteurs de Fourier v_j est la partie la plus coûteuse en calculs. A chaque étape k, $k = 1, \ldots, \log_2(N/2)$, les multiplications matrice-matrice suivantes sont effectuées :

$$\begin{aligned} C_k &= 2C_{k-1} * C_{k-1}, \\ V_k &= 2C_k * U_k - W_k, \end{aligned}$$

```
/* Initialisation */
v_0 → W
PDGEMV ⇒ Cv_0 + iSv_0 → U
PDGEMM ⇒ 2C * U − W → V

/* Boucle principale */
FOR i = 1 TO log_2(N/2) DO
    PDSCMM ⇒ 2C * C − I → C
/*    Mise à jour de W, send/recv entre les colonnes de processeurs :*/
    [perm(V̄), W] → W
/*    Mise à jour en local de U :*/
    [U, V] → U
    PDGEMM ⇒ 2C * U − W → V
ENDFOR
```

TAB. 8.3.2: Algorithme pour la construction des v_j, $j = 0, \ldots, N-1$.

où

$$C_0 = \cos(A), \quad W_0 = v_0, \quad U_0 = v_1,$$

$$W_k = [\mathrm{perm}(\overline{V_{k-1}}), W_{k-1}], \text{ et } U_k = [U_{k-1}, V_{k-1}].$$

Les trois matrices U_k, V_k et W_k sont rectangulaires et leur taille augmente d'une itération à l'autre. La matrice W_k est une permutation des colonnes de V_k. Sa construction nécessite des communications. Ces communications sont des échanges entre paires de processeurs. Ces paires de processeurs sont à déterminer au début de la construction et restent inchangées pendant l'algorithme. Pour une bonne répartition de la charge de travail et pour des raisons que nous expliquerons au paragraphe suivant, nous imposons aux matrices U_k, V_k, W_k d'avoir un partitionnement des blocs, dans la direction des colonnes, égale à deux. Les vecteurs que l'on construit sont complexes. Cependant, pour des raisons pratiques et parce que c'est possible, (aucune multiplication dans le corps des complexes n'est à effectuer), nous préferons continuer à travailler avec l'arithmétique à précision finie pour les réels. L'algorithme est présenté en Table 8.3.2.

Transformée de Fourier 1-D parallèle

Pour la Transformée de Fourier Rapide à une dimension, nous utilisons un algorithme de recouvrement calculs/communications. La structure d'un algorithme de TFR comporte deux parties [20], [96]. La première partie réordonne les données dans un ordre appelé *bit reversal*. Cela s'effectue en permutant des paires d'éléments. La seconde partie est l'implantation en $\log_2(N)$ étapes du lemme de Danielson - Lanczos :

La transformée de Fourier de longueur N peut être réécrite comme la somme de deux transformées de Fourier, chacune de longueur $N/2$, l'une formée des points de numérotation paire et l'autre des points de numérotation impaire.

Le réordonnancement des données ou bit reversal permet d'obtenir les résultats dans l'ordre. Il nécessite biensûr des communications. Dans notre situation, cette étape peut être évitée : il suffit de construire les v_j directement sur la bonne colonne de processeurs. Cela se fait de manière naturelle et transparente si l'on choisit une distribution des v_j par bloc cyclique avec des blocs égaux à 2 dans la direction des colonnes et des blocs de même dimension que ceux de A suivant les lignes.

Illustrons ceci sur un exemple simple. Choisissons $N = 8$. Il faut donc construire $2N$ vecteurs de Fourier soit 16 vecteurs v_j (complexes) :

$$v_0\ v_1\ v_2\ v_3\ v_4\ v_5\ v_6\ v_7\ v_8\ v_9\ v_{10}\ v_{11}\ v_{12}\ v_{13}\ v_{14}\ v_{15}$$

Le réordonnancement en ordre bit reversal de ces vecteurs conduit à

$$v_0\ v_8\ v_4\ v_{12}\ v_2\ v_{10}\ v_6\ v_{14}\ v_1\ v_9\ v_5\ v_{13}\ v_3\ v_{11}\ v_7\ v_{15} \tag{8.3.1}$$

Si la grille de processeurs est telle que $Q = 4$ (4 colonnes de processeurs), alors, d'après l'algorithme 8.3.2 et pour la distribution imposée auparavant, les vecteurs v_j après construction sont répartis comme suit :

colonne 0	colonne 1	colonne 2	colonne 3
$v_0\ v_4\ v_8\ v_{12}$	$v_1\ v_5\ v_9\ v_{13}$	$v_2\ v_6\ v_{10}\ v_{14}$	$v_3\ v_7\ v_{11}\ v_{15}$.

En effectuant une étape de bit reversal sur chaque colonne de processeurs, nous obtenons la distribution suivante :

colonne 0	colonne 1	colonne 2	colonne 3
$v_0\ v_8\ v_4\ v_{12}$	$v_1\ v_9\ v_5\ v_{13}$	$v_2\ v_{10}\ v_6\ v_{14}$	$v_3\ v_{11}\ v_7\ v_{15}$.

Ainsi, si l'on renumérote les colonnes de processeurs, nous avons une distribution par bloc complets de la ligne (8.3.1). C'est une distribution idéale pour la seconde partie de la TFR. Aucune communication n'a été nécessaire pour effectuer ce réordonnancement.

Considérons maintenant la deuxième partie de la Transformée de Fourier Rapide correspondant à l'application du lemme de Danielson-Lancszos. Les calculs peuvent être effectués de manière efficace en utilisant un algorithme de recouvrement calculs/communications : chaque processeur divise ses données en deux blocs. Les calculs sur le bloc 1 (resp. le bloc 2) sont effectués pendant que les données du bloc 2 (resp. bloc 1) sont communiquées. Nous avons représenté en Figure 8.3.1 le schéma en papillon des communications. Ces communications se font entre colonnes de processeurs. En fin d'algorithme, les vecteurs u_l construits sont stockés de manière bloc complet.

8.3.2 Construction des éléments propres

Pour la dernière partie de l'algorithme, chaque processeur sélectionne les vecteurs les plus valables (voir §5.1.3). Lorsque le nombre de vecteurs sélectionné est inférieur à n, nous utilisons une factorisation QR parallèle pour compléter la base. Des implantations parallèles efficaces sont maintenant disponibles [16], [25].

Après projection sur les sous-espaces invariants, chaque processeur peut ensuite résoudre ses petits problèmes de valeurs propres en local, en utilisant une

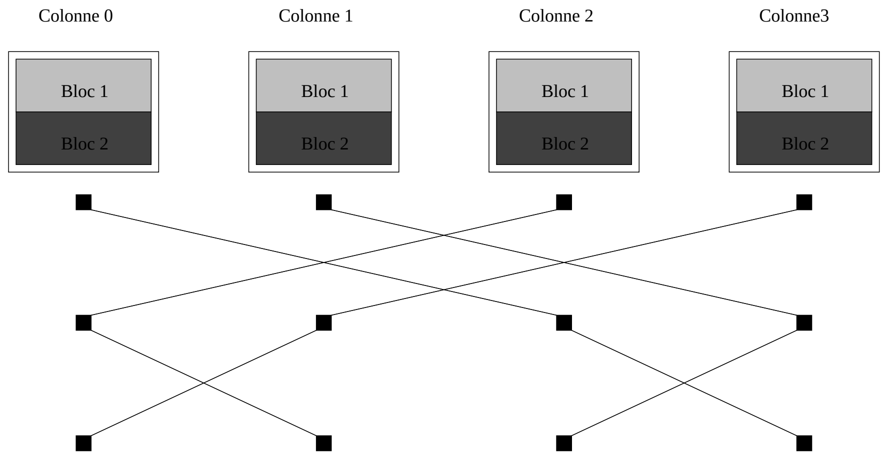

FIG. 8.3.1: Diagramme des communications entre colonnes de processeurs pour la seconde partie de la TFR.

routine standard de résolution du problème symétrique de valeurs propres comme par exemple, `DSYEVX` de LAPACK.

8.4 Produit matrice-matrice symétrique

Au chapitre 6, §6.3, nous avons vu que le coût en calculs de l'algorithme est dominé par des produits matrice-matrice. Ainsi, les performances de notre algorithme dépendront en grande partie des performances du code parallèle utilisé. Nous avons besoin de deux types de produits. Le premier est un produit entre une matrice symétrique et une matrice rectangulaire et le second entre deux matrices symétriques qui commutent. Le résultat doit être symétrique. Donc, seule la moitié des calculs sont nécessaires. Il n'existe pas de code disponible pour ce deuxième type de produit et nous avons dû développer un code parallèle tenant compte de ces avantages.

Notre algorithme est basé sur une idée présentée par Snyder [77]. Nous voulons effectuer l'opération

$$C \leftarrow \alpha A * B + \beta C.$$

Les trois matrices A, B et C sont distribuées de manière bloc-cyclique.

La première étape consiste à construire les blocs diagonaux de C. Seule la partie triangulaire supérieure est construite. Pour cela, il a fallu développer une routine Fortran (`SCMM`) car aucune routine BLAS de niveau 3 n'effectue ce type de calculs. Sur la Paragon d'Intel, cette routine est aussi efficace que les routines de niveau 3 optimisées. Puisque les matrices sont symétriques, chaque bloc diagonal est le produit de deux blocs colonne distribués sur la même colonne de processeurs. Après multiplication, chaque processeur a une partie du résultat. Une somme globale sur cette colonne est nécessaire. Le résultat est laissé sur le processeur possédant le bloc diagonal.

La deuxième étape construit la partie bloc triangulaire supérieure de C. Les blocs colonne A et B ne sont pas toujours sur le même processeur. Il faut diffuser les blocs colonne de A sur les lignes de processeurs et les multiplier par les blocs colonne correspondants de B laissés sur place. Comme précédemment, chaque processeur détient une partie du résultat et une somme globale sur les colonnes est nécessaire. La dernière étape consiste à transposer la partie supérieure de C pour avoir la matrice complète. L'algorithme est présenté en Table 8.4.1 et ses principales différences avec les algorithmes classiques de Fox et Snyder sont résumées dans le Tableau 8.4.2.

8.5 Implantation Parallèle sur la Paragon

8.5.1 iNTEL Paragon

Tous les tests ont été effectués sur une Paragon d'Intel avec 30 noeuds. Chaque noeud est composé de deux processeurs RISC i860, l'un pour les calculs, l'autre pour les communications. Cela permet d'avoir un recouvrement important des calculs par les communications. Le réseau d'interconnexion est une grille 2D avec, comme composant de routage, le iMRC (iNTEL Mesh Routine Chip) et un mode

$N_b = \lceil \frac{M}{B_s} \rceil$
$P_b = \lceil \frac{N_b}{P} \rceil$
/* Construction des blocs diagonaux de C */
FOR $i = 0$ TO N_b DO
    $cur\_row = mod(i, P)$
    $cur\_col = mod(i, Q)$
    $b_r = \lceil \frac{i}{P} \rceil$
    $b_c = \lceil \frac{i}{Q} \rceil$
    IF ($my\_col = cur\_col$) THEN
        construire les blocs diagonaux de C :
        DSCMM $\Rightarrow \alpha A^T_{.,b_c} \times B_{.,b_c} + \beta C_{b_r,b_c} \rightarrow C_{b_r,b_c}$
        somme globale de C_{b_r,b_c} :
        DGSUM2D $\Rightarrow$ le résultat est laissé sur le proc. (cur_row, cur_col)
    ENDIF
ENDFOR
/* Construction des blocs de la partie triangulaire supérieure stricte de C */
FOR $i = 0$ TO $N_b - 1$ DO
    $cur\_row = mod(i, P)$
    $cur\_col = mod(i, Q)$
    $b_r = \lceil \frac{i}{P} \rceil$
    $b_c = \lceil \frac{i}{Q} \rceil$
    IF ($my\_col = cur\_col$) THEN
        DGEBS2D $\Rightarrow$ diffuser $A_{.,b_c}$ à tous les proc. de la ligne cur_row
        construire la $i^{ème}$ ligne de blocs de la partie triang. sup. de C :
        DGEMM $\Rightarrow \alpha A^T_{.,b_c} \times B_{.,(b_c+1,\ldots,P_b)} + \beta C_{b_r,(b_c+1,\ldots,P_b)} \rightarrow C_{b_r,(b_c+1,\ldots,P_b)}$
        somme globale de $C_{b_r,(b_c+1,\ldots,P_b)}$:
        DGSUM2D $\Rightarrow$ le résultat est laissé sur le proc. (cur_row, my_col)
    ELSE
        DGBR2D $\Rightarrow$ recevoir $A_{.,b_c}$
        construire le la $i^{ème}$ ligne de blocs de la partie triang. sup. de C :
        DGEMM $\Rightarrow \alpha A^T_{.,b_c} \times B_{.,(b_c,\ldots,P_b)} + \beta C_{b_r,(b_c,\ldots,P_b)} \rightarrow C_{b_r,(b_c,\ldots,P_b)}$
        somme globale de $C_{b_r,(b_c,\ldots,P_b)}$:
        DGSUM2D $\Rightarrow$ le résultat est laissé sur le proc. (cur_row, my_col)
    ENDIF
ENDFOR
/* transposer et copier la partie triang. supérieure de C dans la partie inférieure.*/

TAB. 8.4.1: Algorithme de multiplication de matrices symétriques communtantes..

Algorithme	PDGEMM	Snyder	PSCMM
Pre processing	Aucun	Transposition of B	Aucun
Construction de C	localement	Somme globale horizontale	Somme globale horizontale
Mouvement de A	Diffusion horizontale	Aucun	Diffusion horizontale
Mouvement de B	Rotation verticale	Rotation Verticale	Aucun
Post processing	Aucun	Transposition de B	Transposition de C

TAB. 8.4.2: Comparaison des types de communications utilisées pour le produit matrice-matrice de ScaLAPACK, de Fox et de `PSCMM`.

de communication worm-hole. La bande passante est de 69 Mbytes et la latence de 60 μs. L'architecture de la Paragon est illustrée à la Figure 8.5.1.

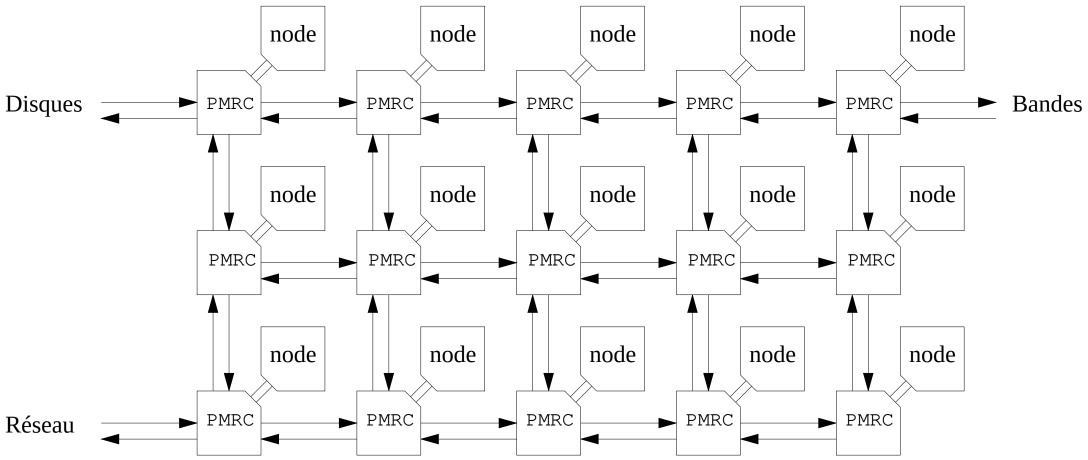

FIG. 8.5.1: Architecture de la Paragon.

8.5.2 Produit matrice-matrice

Nous présentons en figure 8.5.2 les temps d'exécution de la routine `PDGEMM` de ScaLAPACK pour un produit matrice-matrice complet et les temps d'exécution de notre routine, cela pour différentes tailles de matrices. La ligne en pointillés est le rapport entre les deux temps d'exécution. Le rapport entre un produit matrice

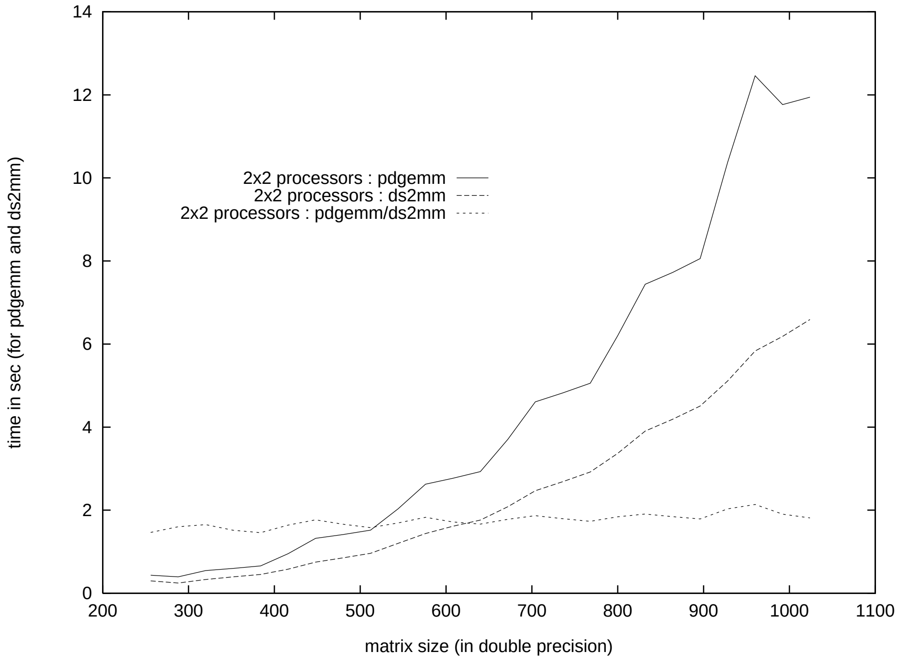

FIG. 8.5.2: Comparaison entre `PDGEMM` et `PSCMM`

matrice complet et un produit symétrique ne peut jamais atteindre deux. La routine `PDSCMM` est très efficace pour des tailles de matrices très grandes. Avec des tailles de matrices plus petites, le gain décroît. Cela est dû à la distribution par bloc cyclique des données. Chaque processeur n'a pas exactement la moitié des calculs à faire pour effectuer le produit symétrique. Le rapport reste cependant toujours supérieur à 1.5 quelle que soit la taille de la matrice.

8.5.3 Premiers Résultats

Nous avons obtenus des résultats préliminaires sur la Paragon d'Intel concernant la partie de notre code la plus coûteuse en calculs. A la Figure 8.5.4, nous avons représenté le *speed-up* et *l'efficacité*. pour une grille 2×2 de processeurs. Ce sont deux mesures importantes de la qualité d'un algorithme parallèle. Le speed-up atteint par un algorithme parallèle tournant sur p processeurs identiques, est le rapport entre le temps T_1 nécessaire à l'exécution du meilleur algorithme séquentiel et le temps T_p nécessaire à l'exécution de l'algorithme parallèle sur p

processeurs :

$$S_p = \frac{T_1}{T_p} \approx \frac{\text{Temps d'exécution sur 1 processeur}}{\text{Temps d'exécution sur } p \text{ processeurs}}.$$

L'efficacité est le rapport entre le speed-up et le nombre de processeurs :

$$E_p = \frac{S_p}{p}.$$

Nos courbes du Speed-up et de l'efficacité sont limitées à des matrices de taille 256 à cause de la grande quantité de mémoire nécessaire pour notre algorithme. Pour des tailles suffisamment grandes, notre speed-up est proche de 3.5 ce qui est relativement bon sur 4 processeurs. Ces résultats ne sont pas surprenants puisque cet algorithme utilise des modules ou routines parallèles efficaces.

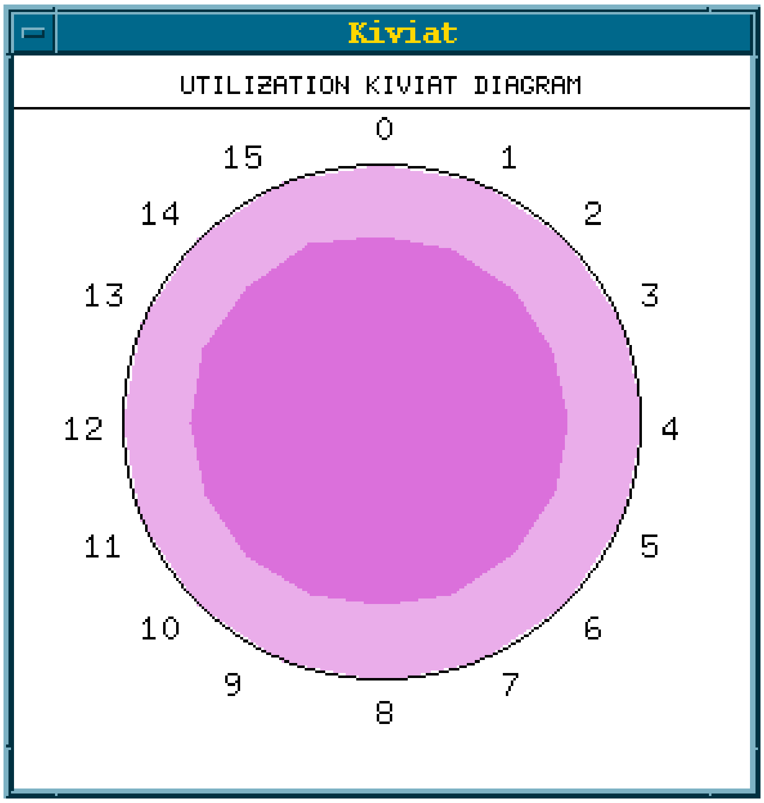

FIG. 8.5.3: Diagramme du Kiviat pour une exécution du noyau principal de Yau et Lu sur 16 noeuds de la Paragon.

Pour illustrer le bon équilibrage de la charge de travail sur chaque processeur,

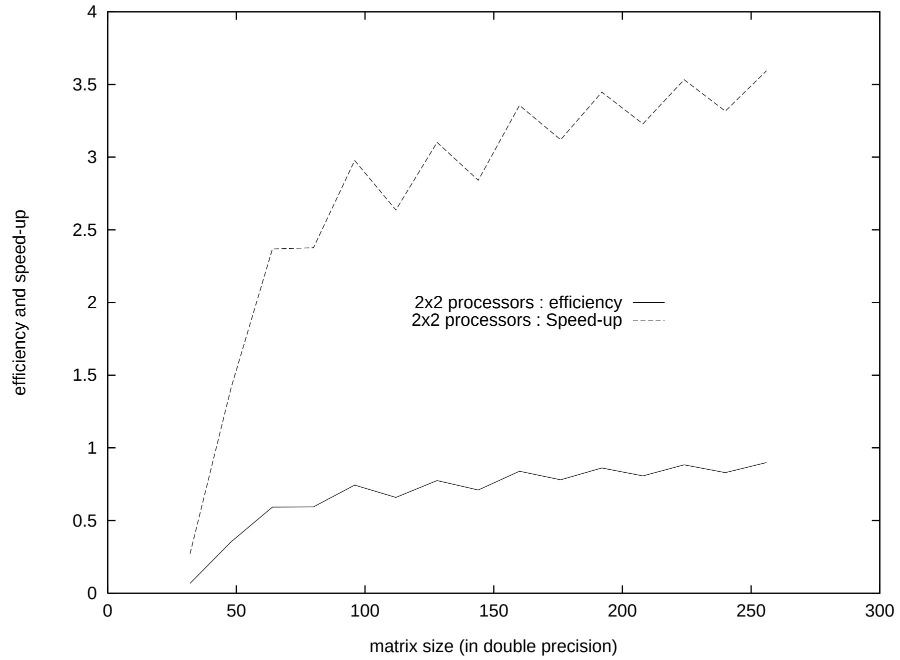

FIG. 8.5.4: Speed-up et efficacité

nous avons lancé une exécution sur 16 noeuds de la Paragon et tracé le diagramme du Kiviat à l'aide du logiciel Paragraph (voir Figure 8.5.3). Le gris foncé représente le temps d'occupation des processeurs numérotés de 0 à 15. Le gris clair correspond au temps d'attente. La partie gris foncé est en forme de disque ce qui signifie que le travail est bien réparti sur chaque processeur. Le temps d'occupation des processeurs est supérieur au temps d'attente.

8.6 Commentaires et Conclusion

Nous avons proposé une parallélisation de la méthode de Yau et Lu. Nous avons concentré nos efforts sur la partie la plus coûteuse en calculs. Le corps principal de l'algorithme de Yau et Lu repose sur une série de produits matrice-matrice. Ainsi, l'efficacité de notre code dépend des algorithmes parallèles de produits matriciels utilisés. La routine `PDSCMM` que nous avons développée pour les produits de matrices symétriques et commutantes, permet d'apporter un gain d'efficacité de 2 par rapport à la routine générale `PDGEMM`. Comme espéré, les premiers résultats obtenus sur la Paragon d'Intel pour l'exécution du corps principal de l'algorithme sont relativement bons et encourageants en ce qui concernent le speed-up et l'efficacité.

Un des rapporteurs de [36] nous a fait remarquer que puisque la complexité en calculs de l'algorithme de Yau et Lu est très supérieure aux algorithmes standards, son utilisation ne peut être justifiée que si l'algorithme se comporte bien sur machines parallèles avec un très grand nombre de processeurs. Bien sûr, les expériences faites sur une grille de 2×2 de processeurs ne peuvent servir d'arguments de réponse. Cependant, des études théoriques menées en [94] ont montré que notre algorithme est très scalable.

Nous avons proposé une stratégie de parallélisation pour la méthode de Yau et Lu, dans son ensemble. Notre code parallèle n'est pourtant pas complet. Seule le corps principal, c'est-à-dire la partie la plus coûteuse en calculs, est actuellement parallélisé. Il est inutile d'avoir un algorithme parallèle complet si cette partie n'est pas compétitive par rapport aux autres algorithmes parallèles pour résoudre le problème symétrique de valeurs propres. Lorsque nous avons commencé la parallélisation de la méthode de Yau et Lu, il n'y avait pas encore de code parallèle disponible dans les bibliothèques d'algèbre linéaire. La bibliothèque ScaLAPACK propose maintenant une version parallèle de la bissection suivie par l'itération inverse pour le calcul des vecteurs propres [24], [34]. Nous avons pu comparer les temps d'exécution de `PDSYEVX` avec ceux de notre algorithme (voir Figure 8.6.1). Nous n'avons pas exigé une grande précision concernant la construction des éléments propres par `PDSYEVX` afin d'avoir des temps d'exécution très bons. L'orthogonalité des vecteurs propres n'est pas vérifiée. Les courbes de la figure 8.6.1 montrent que le corps principal de notre algorithme est compétitif par rapport à la routine `PDSYEVX`. Ces résultats, plutôt encourageants, nous suggèrent

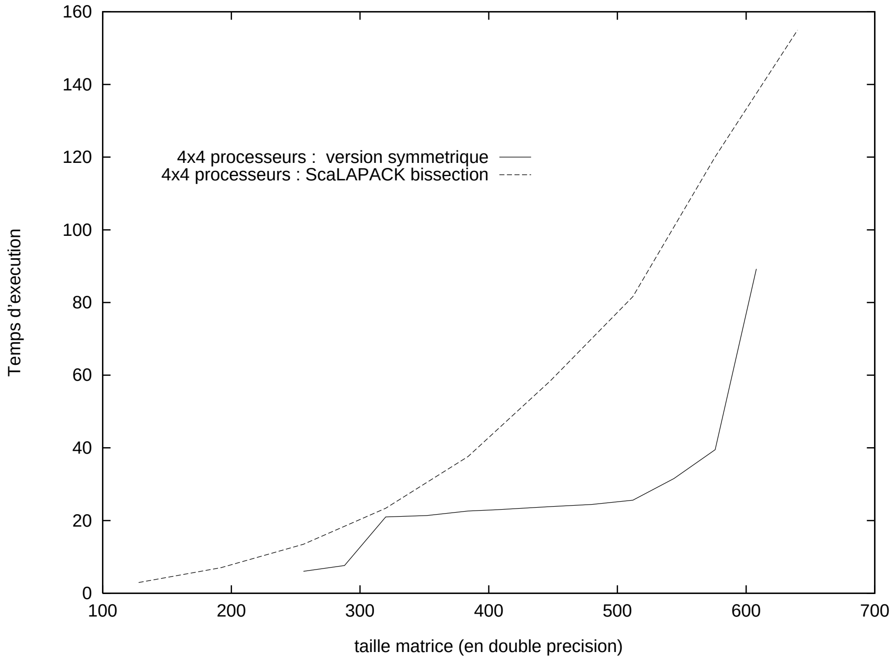

FIG. 8.6.1: Temps d'exécution de PDSYEVX et du noyau principal de Yau et Lu.

de poursuivre la parallélisation du code.

Nous craignons cependant que, dans la dernière partie de l'algorithme, une mauvaise répartition du spectre de la matrice A entraîne une mauvaise répartition de la charge de travail, une augmentation des communications et une perte de stabilité dans le cas où les processeurs partagent des vecteurs propres appartenant à un même cluster. Ce dernier problème n'est pas nouveau dans les algorithmes parallèles de recherche d'éléments propres et a été signalé par exemple dans [34].

Bibliographie

[1] Mario Ahues, Alain Largillier, and Françoise Tisseur. Stability and stopping criterion for the QR algorithm. In *Proceedings of the Householder Symposium XIII*, 1996.

[2] Mario Ahues, Alain Largillier, and Françoise Tisseur. Stopping and deflate criterion for the QR algorithm. In *Proceedings of the International Workshop on Accurate Eigensolving and Applications, Split, Croatia, July 11-17*, 1996.

[3] E. Anderson, Z. Bai, C. H. Bischof, J. W. Demmel, J. J. Dongarra, J. J. Du Croz, A. Greenbaum, S. J. Hammarling, A. McKenney, S. Ostrouchov, and D. C. Sorensen. *LAPACK Users' Guide, Release 2.0*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, second edition, 1995.

[4] Louis Auslander and Anna Tsao. On parallelizable eigensolvers. *Advances in Applied Math.*, 13 :253–261, 1992. (Preprint appeared as Technical Report SRC-TR-91-028, Supercomputing Research Center).

[5] Z. Bai and J. W. Demmel. On a direct algorithm for computing invariant subspaces with specified eigenvalues. Technical Report CS-91-139, Department of Computer Science, University of Tennessee, Knoxville, TN, USA, November 1991. LAPACK Working Note 38.

[6] Zhaojun Bai and James W. Demmel. On a Block Implementation of the

Hessenberg Multishift QR Iteration. *Int. J. High Speed Comput.*, 1 :97–112, 1989.

[7] Zhaojun Bai and James W. Demmel. Design of a parallel nonsymmetric eigenroutine toolbox, Part I. In Richard F. Sincovec, David E. Keyes, Michael R. Leuze, Linda R. Petzold, and Daniel A. Reed, editors, *Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing, Volume I*, pages 391–398. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1993.

[8] Zhaojun Bai, James W. Demmel, and Alan McKenney. On computing condition numbers for the nonsymmetric eigenproblem. *ACM Trans. Math. Software*, 19(2) :197–212–223, 1993.

[9] Jesse Barlow and James W. Demmel. Computing accurate eigensystems of scaled diagonally dominant matrices. *SIAM J. Numer. Anal.*, 27(3) :762–791, 1990.

[10] Jesse L. Barlow. Error analysis of update methods for the symmetric eigenvalue problem. *SIAM J. Matrix Anal. Appl.*, 14(2) :598–618, 1993.

[11] R. H. Barlow and D. J. Evans. A parallel organisation of the bisection algorithm. *The Computer Journal*, 22 :267–269, 1977.

[12] Steve Batterson. Convergence of the shifted QR algorithm on 3×3 normal matrices. *Numer. Math.*, 58 :341–352, 1990.

[13] Steve Batterson and David Day. Linear convergence in the shifted QR algorithm. *Math. Comp.*, 59 :141–152, 1992.

[14] Steve Batterson and Smillie. Convergence of the shifted QR algorithm on 3×3 normal matrices. *Math. Comp.*, 55 :169–178, 1990.

[15] H. J. Bernstein and M. Goldstein. Parallel implementation of bisection for the calculation of eigenvalues of tridiagonal symmetric matrices. *Computing*, 37 :85–91, 1986.

[16] C. H. Bischof. A parallel QR factorization algorithm with controlled local pivoting. *SIAM J. Sci. Stat. Comput.*, 21 :36–57, 1991.

[17] Christian Bischof, Steven Huss-Lederman, Xiaobai Sun, and Anna Tsao. The PRISM Project : Infrastructure and Algorithms for Parallel Eigensolvers. Prism working note #13, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, USA, oct 1993.

[18] Christian Bischof and Xiaobai Sun. A divide-and-conquer method for tridiagonalizing symmetric matrices with repeated eigenvalues. National Lab Technical Report ANL-MCS-P286-0192, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, USA, mar 1992. (also PRISM Working Note #1).

[19] Christian H. Bischof and Charles F. Van Loan. The WY representation for products of Householder matrices. *SIAM J. Sci. Stat. Comput.*, 8(1) :s2–s13, 1987.

[20] E. Oran Brigham. *The Fast Fourier Transform and its Applications.* Prentice-Hall, Englewood Cliffs, NJ, USA, 1988.

[21] James R. Bunch, Christopher P. Nielsen, and Danny C. Sorensen. Rank-one modification of the symmetric eigenproblem. *Numer. Math.*, 31 :31–48, 1978.

[22] H. J. Buurema. *A geometric proof of convergence for the QR method.* PhD thesis, Rijksuniversiteit Te Groningen, 1970.

[23] Françoise Chatelin. *Valeurs Propres de Matrices.* MASSON, Paris, Milan, Barcelone, 1988.

[24] Jaeyoung Choi, James Demmel, I. Dhillon, Jack J. Dongarra, Susan Ostrouchov, Antoine P. Petitet, k. Stanley, David W. Walker, and R. Clint Whaley. ScaLAPACK : A portable linear algebra library for distributed

memory computers - design issues and performances. LAPACK Working Note 95, Oak Ridge National Laboratory, Oak Ridge, TN, USA, 1995.

[25] Jaeyoung Choi, Jack J. Dongarra, Susan Ostrouchov, Antoine P. Petitet, David W. Walker, and R. Clint Whaley. The design and implementation of the ScaLAPACK LU, QR and Cholesky factorization routines. Report ORNL/TM-12470, Oak Ridge National Laboratory, Oak Ridge, TN, USA, September 1994. LAPACK Working Note 80.

[26] Jaeyoung Choi, Jack J. Dongarra, and David W. Walker. The design of a parallel, dense linear algebra software library : Reduction to Hessenberg, tridiagonal, and bidiagonal form. In Jack. J. Dongarra and Bernard Tourancheau, editors, *Environments and Tools For Parallel Scientific Computing*, pages 98–111. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1994.

[27] C.W. Cooley and J.W. Tuckey. An algorithm for the machine calculation of complex fourier series. *Math. Comput.*, 19 :297–301, 1965.

[28] J. J. M. Cuppen. A divide and conquer method for the symmetric tridiagonal eigenproblem. *Numer. Math.*, 36 :177–195, 1981.

[29] David Day. How the QR algorithm fails to converge and how to fix it. Sandia technical report 96-0913J, Sandia National Laboratory, April 1996.

[30] James W. Demmel. The componentwise distance to the nearest singular matrix. *SIAM J. Matrix Anal. Appl.*, 13(1) :10–19, 1992.

[31] James W. Demmel. A specification for floating point parallel prefix. Technical Report CS-92-167, Department of Computer Science, University of Tennessee, Knoxville, TN, USA, July 1992. LAPACK Working Note 49.

[32] James W. Demmel. Trading off parallelism and numerical stability. In Marc S. Moonen, Gene H. Golub, and Bart L. De Moor, editors, *Linear Algebra for Large Scale and Real-Time Applications*, volume 232 of *NATO*

ASI Series E, pages 49–68. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1993.

[33] James W. Demmel, Inderjit Dhillon, and Huan Ren. On correctness of parallel bisection in floating point. Computer Science Division Report UCB/CSD-94-805, University of California, Berkeley, CA, USA, March 1994.

[34] James W. Demmel and K. Stanley. The performance of finding eigenvalues and eigenvectors of dense symmetric matrices on distributed memory computers. Technical Report CS-94-254, Department of Computer Science, University of Tennessee, Knoxville, TN, USA, September 1994. LAPACK Working Note 86.

[35] James W. Demmel and Krešimir Veselić. Jacobi's method is more accurate than *QR*. *SIAM J. Matrix Anal. Appl.*, 13(4) :1204–1245, 1992.

[36] Stéphane Domas and Françoise Tisseur. Parallel implementation of a symmetric eigensolver based on the Yau and Lu method. In *Vector and Parallel Processing – VECPAR'96*, Lecture Notes in Computer Science 1215, pages 140–153. Springer-Verlag, Berlin, 1997.

[37] J. Dongarra and D. Sorensen. A fully parallel algorithm for the symmetric eigenvalue problem. *SIAM J. Sci. Stat. Comput.*, 8 :139–154, 1987.

[38] Augustin A. Dubrulle. The multishift QR algorithm : Is it worth the trouble ? Report TR G320-3558, IBM Corporation, Palo Alto Scientific Center, CA, 1992.

[39] Augustin A. Dubrulle and Gene H. Golub. A Multishift QR Iteration without Computation of the Shifts. *Numerical Algorithms*, 7 :173–181, 1994.

[40] Nahid Emad. *Contribution à la résolution de grands problèmes de valeur propre.* PhD thesis, Université Paris VI, 1989.

[41] W. Feller and George E. Forsythe. New matrix transformations for obtaining characteristic vectors. *Quart. appl. Math.*, 8 :325–331, 1951.

[42] George E. Forsythe and Cleve B. Moler. *Computer Solution of Linear Algebraic Systems*. Prentice-Hall, Englewood Cliffs, NJ, USA, 1967.

[43] J. G. F. Francis. The QR transformation, parts I and II. *Computer Journal*, 4 :265–271, 332–345, 1961, 1962.

[44] George A. Geist. Reduction of a general matrix to tridiagonal form. *SIAM J. Matrix Anal. Appl.*, 12(2) :362–373, 1991.

[45] Marc Gengler, Stéphane Ubéda, and Frédéric Desprez. *Initiation au parallélisme :concepts, architectures et algorithmes*. MASSON, Paris, Milan, Barcelone, 1996.

[46] D. Giménez, R. van de Geijn, V. Hernández, and A. M. Vidal. Exploiting the symmetry on the Jacobi method on a mesh of processors. In *4th EUROMICRO Workshop on Parallel and Distributed Processing, Braga, Portugal*, 1996.

[47] Gene H. Golub. Some modified matrix eigenvalue problems. *SIAM Review*, 15(2) :318–334, 1973.

[48] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, MD, USA, second edition, 1989.

[49] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, MD, USA, third edition, 1996.

[50] A. Greenbaum and J. J. Dongarra. Experiments with QR/QL methods for the symmetric tridiagonal eigenproblem. Technical report, LAPACK Working Note #17, Department of Computer Science, University of Tennessee, Knoxville, 1989.

[51] Ming Gu and Stanley C. Eisenstat. A stable and efficient algorithm for

the rank-one modification of the symmetric eigenproblem. *SIAM J. Matrix Anal. Appl.*, 15(4) :1266–1276, 1994.

[52] Ming Gu and Stanley C. Eisenstat. A divide-and-conquer algorithm for the symmetric tridiagonal eigenproblem. *SIAM J. Matrix Anal. Appl.*, 16(1) :172–191, 1995.

[53] B. Hendrickson and D. Womble. The torus-wrap mapping for dense matrix calculations on massively parallel computers. *SIAM J. Sci. Comput.*, 15 :1201–1226, 1994.

[54] G. Henry, D. Watkins, and J. Dongarra. A parallel implementation of the nonsymmetric QR algorithm for distributed memory architectures. Computer Science Dept. Technical Report CS-97-352, University of Tennessee, Knoxville, TN, March 1997. (also LAPACK Working Note # 121).

[55] Greg Henry and Robert Van de Geijn. Parallelizing the QR algorithm for the unsymmetric algebraic eigenvalue problem : Myths and reality. *SIAM J. Sci. Comput.*, 17(4) :870–883, 1996.

[56] Nicholas J. Higham. *Handbook of Writing for the Mathematical Sciences.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1993.

[57] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1996.

[58] Nicholas J. Higham and Robert S. Schreiber. Fast polar decomposition of an arbitrary matrix. *SIAM J. Sci. Stat. Comput.*, 11(4) :648–655, July 1990.

[59] Alston S. Householder. Unitary triangularization of a nonsymmetric matrix. *J. Assoc. Comput. Mach.*, 5 :339–342, 1958.

[60] Alston S. Householder. *The Theory of Matrices in Numerical Analysis.* Blaisdell, New York, 1964. Reprinted by Dover, New York, 1975.

[61] H. Huang. A parallel algorithm for symmetric tridiagonal eigenvalue problems. *CAC Document No.109, Center for Advanced Computation, University of Illinois*, 1974.

[62] Steve Huss-Lederman. Large-scale matrix diagonalization in computational chemistry. *SIAM News*, 29(9), 1996.

[63] I. C. F. Ipsen and E. R. Jessup. Solving the symmetric tridiagonal eigenvalue problem on the hypercube. *SIAM J. Sci. Stat. Comput.*, 11(2) :203–29, 1990.

[64] E. R. Jessup. A case against a divide and conquer approach to the nonsymmetric eigenvalue problem. *Applied Numerical Mathematics*, 12 :403–420, 1993.

[65] E. R. Jessup and I. C. F. Ipsen. Improving the accuracy of inverse iteration. *SIAM J. Sci. Stat. Comput.*, 13(3) :550–72, 1992.

[66] W. Kerner. Large-scale complex eigenvalue problems. *J. Comp. Phys.*, 85 :1–85, 1989.

[67] V. N. Kublanovskaya. On some algorithms for the solution of the complete eigenvalue problem. *Zh. vych. mat.*, 1 :555–570, 1961.

[68] J. R. Kuttler and V. G. Sigillito. Eigenvalues of the Laplacien in two dimensions. *SIAM Review*, 26(2) :163–193, 1984.

[69] Abdelhak Lakhouaja. *Contribution à l'algorithmique numérique parallèle.* PhD thesis, Université Mohamed Premier, Faculté des Sciences, Oujda, 1996.

[70] Bruno Lang. A parallel algorithm for reducing symmetric banded matrices to tridiagonal form. *SIAM J. Matrix Anal. Appl.*, 14(6) :1320–1338, 1993.

[71] Alain Largillier and Françoise Tisseur. Theorie et pratique de la méthode QR en son $40^{ème}$ anniversaire. le shift : mythe ou réalité. Research Report N° 202, URA CNRS 740 Lyon St-Etienne, April 1995.

[72] Pierre-Jean Laurent. *Approximation et Optimisation*, volume 13. Hermann, Paris, 1972.

[73] Colette Lebaud. L'algorithme double Q.R avec ≪ shift ≫. *Numer. Math.*, 16 :163–180, 1970.

[74] Steven Lederman, Anna Tsao, and Thomas Turnbull. A parallelizable eigensolver for real diagonalizable matrices with real eigenvalues. Prism working note #20, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, USA, 1994. (to appear in SIAM Journal of Scientific Computing).

[75] T. Li, H. Zhang, and X. Sun. Parallel homotopy algorithm for the symmetric tridiagonal eigenvalue problem. *SIAM J. Sci. Stat. Comput.*, 12(3) :469–487, 1991.

[76] T.-Y. Li, Z. Zeng, and L. Cong. Solving eigenvalue problems of real nonsymmetric matrices with real homotopies. *SIAM J. Numer. Anal.*, 29(1) :229–248, 1992.

[77] C. Lin and L.Snyder. A matrix product algorithm and its comparative performance on hypercubes. In *Scalable High Performance Computing Conference SHPCC92*, pages 190–193. IEEE Computer Society, 1992.

[78] Nurit Lipkin, Nimrod Moiseyev, and Claude Leforestier. A three-dimensional study of NeICI predissociation resonances by the complex scaled discret variable xrepresentation method. *J. Chem. Phys.*, 98(3) :1888–1901, 1993.

[79] Richard J. Littlefield and Kristyn J. Maschhoff. Investigating the performance of parallel eigensolvers for large processor counts. *Theor. Chim. Acta*, 84 :457–473, 1993.

[80] S. Lo, Bernard Philippe, and A. Sameh. A multiprocessor algorithm for the

symmetric tridiagonal eigenvalue problem. *SIAM J. Sci. Stat. Comput.*, 8 :s155–s165, 1987.

[81] R. S. Martin, G. Peters, and J. H. Wilkinson. The QR algorithm for real Hessenberg matrices. *Numer. Math.*, 14 :219–231, 1970.

[82] Roy Mathias. Accurate eigensystem computations by Jacobi methods. *SIAM J. Matrix Anal. Appl.*, 16(3) :977–1003, 1995.

[83] Roy Mathias. The instability of parallel prefix matrix multiplication. *SIAM J. Sci. Comput.*, 16(4) :956–973, 1995.

[84] G. S. Miminis and C. C. Paige. Implicit shifting in the QR and related algorithms. *SIAM J. Matrix Anal. Appl.*, 12(2) :385–400, 1991.

[85] Beresford N. Parlett. Global convergence of the basic QR algorithm on Hessenberg matrices. *Math. Comp.*, 22 :803–817, 1968.

[86] Beresford N. Parlett. *The Symmetric Eigenvalue Problem.* Prentice-Hall, Englewood Cliffs, NJ, USA, 1980.

[87] Beresford N. Parlett. *Private communication*, 1996.

[88] Beresford N. Parlett and Jian Le. Forward instability of tridiagonal QR. *SIAM J. Matrix Anal. Appl.*, 14(1) :279–316, 1993.

[89] Beresford N. Parlett and W. G. Poole. A geometric theory for the QR and LU and power iterations. *SIAM J. Numer. Anal.*, 10(2) :389–412, 1973.

[90] M. S. Paterson and L. J. Stockmeyer. On the number of nonscalar multiplications necessary to evaluate polynomials. *SIAM Journal on Computing*, 2 :60–66, 1973.

[91] G. Peters and J. H. Wilkinson. Eigenvectors of real and complex matrices by LR and QR triangularization. *Numerische Mathematik*, 16 :181–204, 1970.

[92] Bernard Philippe. An algorithm to improve nearly orthonormal sets of vectors on a vector processor. *SIAM J. Alg. Disc. Meth.*, 8 :396–403, 1987.

[93] Makan Pourzandi. *Etude de l'impact des recouvrements calculs/communications sur des algorithmes parallèles de calculs matriciels*. PhD thesis, Laboratoire de l'Informatique du parallélisme, Lyon, France, 1995.

[94] Makan Pourzandi and Françoise Tisseur. Parallèlisation d'une nouvelle méthode de recherche de valeurs propres pour des matrices réelles symétriques. Report TR94-37, LIP, ENS Lyon, 1994.

[95] Makan Pourzandi and Bernard Tourancheau. A parallel study of Jacobi-like eigenvalue solution. In *First International Meeting on Vector and Parallel Processing*, 1993.

[96] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in FORTRAN : The Art of Scientific Computing*. Cambridge University Press, second edition, 1992.

[97] Oscar Rojo and Ricardo L. Soto. A decreasing sequence of eigenvalue localization regions. *Linear Algebra and Appl.*, 196 :71–84, 1994.

[98] Heinz Rutishauser. *Solution of Eigenvalue Problems with the LR-Transformation*, volume 49, pages 47–81. Nat. Bur. Standards Appl. Math. Ser., 1958.

[99] Victor Ryaboy, Nimrod Moiseyev, V.A. Mandelshtam, and H. S. Taylor. Resonance positions and widths by complex scaling and modified stabilization methods : van der Waals complex NeICI. *J. Chem. Phys.*, 101(7) :5677–5682, 1994.

[100] Ahmed H. Sameh and David J. Kuck. A parallel QR algorithm for symmetric tridiagonal matrices. *IEEE Trans. Comput.*, c-26(2) :147–153, 1977.

[101] D. Schmitt, B. Steffen, and T. Weiland. 2D and 3D computations of lossy eigenvalue problems. *IEEE Trans. Magnetics*, 30(5) :3578–3581, 1994.

[102] D. Schmitt and T. Weiland. 2D and 3D computations of eigenvalue problems. *IEEE Trans. Magnetics*, 28(2) :1793–1796, 1992.

[103] Ivan Slapnicar. *Accurate Symmetric Eigenreduction by a Jacobi Method.* PhD thesis, Fernuniversität Hagen, Germany, 1992.

[104] B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow, Y. Ikebe, V. C. Klema, and C. B. Moler. *Matrix Eigensystem Routines - EISPACK Guide*, volume 6 of *Lecture Notes in Computer Sciences.* Springer-Verlag, Berlin, 1976.

[105] J. Speiser and H. Whitehouse. Parallel processing algorithms and architectures for real time signal processing. In IEEE, editor, *Proc. SPIE Real Time Signal Processing IV, SPIE*, pages 2–9, 1981.

[106] G. W. Stewart. *Introduction to Matrix Computations.* Academic Press, New York, 1973.

[107] G. W. Stewart. Algorithm 506 HQR3 and EXCHANG : Fortran subroutine for calculating and ordering the eigenvalues of a real upper Hessenberg matrix [f2]. *ACM Transactions on Mathematical Software*, 2 :275–280, 1976.

[108] G. W. Stewart. A parallel implementation of the QR algorithm. *Parallel Computing*, 5 :187–196, 1987.

[109] Robert van de Geijn. Deferred shifting schemes for parallel QR methods. *SIAM J. Matrix Anal. Appl.*, 14(1) :180–194, 1993.

[110] Robert van de Geijn and Duncan G. Hudson. Efficient parallel implementation of the nonsymmetric QR algorithm. In *Proceedings of DMCC5 the Sixth SIAM Conference on Parallel Processing for Scientific Computing, Volume I*, pages 391–398, 1989.

[111] C. F. Van Loan. The sensitivity of the matrix exponential. *SIAM J. Numer. Anal.*, 14 :971–981, 1977.

[112] D. S. Watkins and L. Elsner. Chasing algorithms for the eigenvalue problem. *SIAM J. Matrix Anal. Appl.*, 12(2) :374–384, 1991.

[113] D. S. Watkins and L. Elsner. Convergence of algorithms of decomposition type for the eigenvalue problem. *Linear Algebra and Appl.*, 143 :19–47, 1991.

[114] David S. Watkins. Understanding the QR algorithm. *SIAM Review*, 24 :427–440, 1982.

[115] David S. Watkins. Bidirectional chasing algorithms for the eigenvalue problem. *SIAM J. Matrix Anal. Appl.*, 14(1) :166–179, 1993.

[116] David S. Watkins. Shifting strategies for the parallel QR algorithm. *SIAM J. Sci. Stat. Comput.*, 15(4) :953–958, July 1994.

[117] David S. Watkins. Forward stability and transmission of shifts in the QR algorithm. *SIAM J. Matrix Anal. Appl.*, 16(2) :469–487, April 1995. with J. B. Haag.

[118] David S. Watkins. The transmission of shifts and shift blurring in the QR algorithm. *Linear Algebra and Appl.*, 241–243 :877–896, 1996.

[119] J. H. Wilkinson. Error analysis of eigenvalue techniques based on orthogonal transformations. *J. Soc. Indust. Appl. Math.*, 10(1) :162–195, 1962.

[120] J. H. Wilkinson. *Rounding Errors in Algebraic Processes.* Notes on Applied Science No. 32, Her Majesty's Stationery Office, London, 1963. Also published by Prentice-Hall, Englewood Cliffs, NJ, USA. Reprinted by Dover, New York, 1994.

[121] J. H. Wilkinson. *The Algebraic Eigenvalue Problem.* Oxford University Press, 1965.

[122] J. H. Wilkinson. Error analysis of transformations based on the use of matrices of the form $I - 2ww^H$. In Louis B. Rall, editor, *Error In Digital Computation*, volume 2, pages 77–101. Wiley, New York, 1965.

[123] Shing-Tung Yau and Ya Yan Lu. Reducing the symmetric matrix eigenvalue problem to matrix multiplications. *SIAM J. Sci. Comput.*, 14(1) :121–136, 1993.