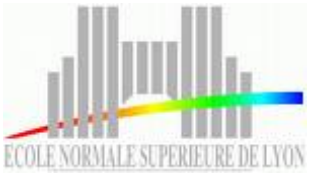
Rapport de stage de M2 : Algorithmes rapides pour la résolution de problèmes algébriques structurés

Christophe MOUILLERON

Janvier - Juin 2008



Stage effectué au Laboratoire de l'Informatique du Parallélisme dans l'équipe Arénaire.
Encadrants : Claude-Pierre Jeannerod et Gilles Villard.

Table des matières

1 Introduction

De nombreux problèmes peuvent s'exprimer en terme de calcul sur des matrices. Suivant l'origine du problème, il se peut que les matrices correspondantes soient plus ou moins particulières. Parfois, les matrices sont creuses (avec peu de coefficients non nuls). Ici nous nous intéresserons plutôt à des matrices denses mais structurées. Ces matrices apparaissent naturellement en physique, lorsqu'on a affaire à une relation de récurrence comme c'est le cas en théorie du signal [18], mais aussi pour les codes correcteurs d'erreurs [20]. Ici, nous travaillerons avec des matrices à coefficients dans un corps $\mathbb{K}$ quelconque pour couvrir à la fois $\mathbb{R}, \mathbb{C}$ et les corps finis.

Mon travail a consisté dans un premier temps à parcourir la littérature afin de lister les algorithmes existants sur les matrices structurées. Ensuite, je me suis concentré sur certains algorithmes que j'ai jugé intéressants afin de mieux les étudier et d'améliorer certains points. Après avoir introduit plus précisément les matrices denses structurées et dressé un état de l'art, je présenterai ces algorithmes ainsi que mes améliorations.

2 Contexte

2.1 Matrices denses structurées

Il existe de nombreux cas où $O(n)$ coefficients suffisent pour construire une matrice de taille $n \times n$. Commençons par voir les exemples les plus classiques.

1. Les matrices Tœplitz sont des matrices où les coefficients d'une même diagonale sont égaux. La connaissance de la première ligne et de la première colonne ($2n-1$ coefficients) permet donc de construire celle-ci en intégralité.
   Pour $n = 3$: $\begin{bmatrix} a_0 & a_{-1} & a_{-2} \\ a_1 & a_0 & a_{-1} \\ a_2 & a_1 & a_0 \end{bmatrix}$
2. Les matrices Vandermonde sont définies à partir d'un vecteur $\mathbf{x} = \begin{bmatrix} x_1 & \dots & x_n \end{bmatrix}^T$. Le coefficient (i, j) de la matrice est alors x_i^{j-1}.
   Pour $n = 3$: $\begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ 1 & x_3 & x_3^2 \end{bmatrix}$
3. Les matrices Cauchy sont définies à partir de deux vecteurs $\mathbf{x} = \begin{bmatrix} x_1 & \dots & x_n \end{bmatrix}^T$ et $\mathbf{y} = \begin{bmatrix} y_1 & \dots & y_n \end{bmatrix}^T$ tels que $\forall (i, j), x_i \neq y_j$. Le coefficient (i, j) de la matrice est alors $\dfrac{1}{x_i - y_j}$.
   Pour $n = 3$: $\begin{bmatrix} \frac{1}{x_1-y_1} & \frac{1}{x_1-y_2} & \frac{1}{x_1-y_3} \\ \frac{1}{x_2-y_1} & \frac{1}{x_2-y_2} & \frac{1}{x_2-y_3} \\ \frac{1}{x_3-y_1} & \frac{1}{x_3-y_2} & \frac{1}{x_3-y_3} \end{bmatrix}$

2.2 Opérateurs de déplacement, rang de déplacement

L'idée pour mettre en évidence une structure est d'appliquer une transformation simple à la matrice de façon à avoir un résultat de faible rang. Une telle transformation sera appelée opérateur de déplacement dans la suite.

Pour une matrice Tœplitz T, la transformation consiste à faire la différence entre d'une part T où on remonte la dernière ligne au dessus de la première et d'autre part T décalée d'un cran vers la gauche :

$$\begin{bmatrix} a_3 & a_2 & a_1 & a_0 \\ a_0 & a_{-1} & a_{-2} & a_{-3} \\ a_1 & a_0 & a_{-1} & a_{-2} \\ a_2 & a_1 & a_0 & a_{-1} \end{bmatrix} - \begin{bmatrix} a_{-1} & a_{-2} & a_{-3} & 0 \\ a_0 & a_{-1} & a_{-2} & 0 \\ a_1 & a_0 & a_{-1} & 0 \\ a_2 & a_1 & a_0 & 0 \end{bmatrix} = \begin{bmatrix} a_3 - a_{-1} & a_2 - a_{-2} & a_1 - a_{-3} & a_0 \\ 0 & 0 & 0 & a_{-3} \\ 0 & 0 & 0 & a_{-2} \\ 0 & 0 & 0 & a_{-1} \end{bmatrix}$$

On constate que le matrice obtenue est de rang au plus 2 (elle a seulement une ligne et une colonne non nulles).

Pour les matrices Cauchy, la transformation consiste à faire la différence de deux mises à l'échelle différentes de la matrice de départ :

$$\begin{bmatrix} x_1 & 0 & 0 \\ 0 & x_2 & 0 \\ 0 & 0 & x_3 \end{bmatrix} \begin{bmatrix} \frac{1}{x_1-y_1} & \frac{1}{x_1-y_2} & \frac{1}{x_1-y_3} \\ \frac{1}{x_2-y_1} & \frac{1}{x_2-y_2} & \frac{1}{x_2-y_3} \\ \frac{1}{x_3-y_1} & \frac{1}{x_3-y_2} & \frac{1}{x_3-y_3} \end{bmatrix} - \begin{bmatrix} \frac{1}{x_1-y_1} & \frac{1}{x_1-y_2} & \frac{1}{x_1-y_3} \\ \frac{1}{x_2-y_1} & \frac{1}{x_2-y_2} & \frac{1}{x_2-y_3} \\ \frac{1}{x_3-y_1} & \frac{1}{x_3-y_2} & \frac{1}{x_3-y_3} \end{bmatrix} \begin{bmatrix} y_1 & 0 & 0 \\ 0 & y_2 & 0 \\ 0 & 0 & y_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

On a cette fois une matrice de rang 1.

Dans la suite, on va considérer la famille d'opérateurs de déplacement suivante. Ici, $\mathrm{Sp}(A)$ désigne l'ensemble des valeurs propres de A :

Définition 1 *Pour $M \in \mathbb{K}^{m\times m}$ et $N \in \mathbb{K}^{n\times n}$ deux matrices telles que $\mathrm{Sp}(M) \bigcap \mathrm{Sp}(N) = \emptyset$, on définit l'opérateur de déplacement $\nabla_{M,N}$ par :*

$$\forall A \in \mathbb{K}^{m\times n}, \quad \nabla_{M,N}(A) = MA - AN \tag{1}$$

La structure d'une matrice peut alors être mesurée à l'aide de la notion de rang de déplacement, introduite par Kailath, Kung et Morf dans [13] :

Définition 2 *On appellera rang de déplacement d'une matrice A pour un opérateur de déplacement donné et on notera α le rang de l'image de cette matrice par l'opérateur :*

$$\alpha = \mathrm{rg}\,(\nabla_{M,N}(A)).$$

Étant donnée une matrice $m\times n$ de rang α, on peut toujours trouver $G \in \mathbb{K}^{m\times\alpha}$ et $H \in \mathbb{K}^{n\times\alpha}$ tels que cette matrice soit égale à GH^T. Ainsi, nous sommes en mesure de représenter une matrice dense structurée dont le rang de déplacement est α par un couple (G, H) tel que $\nabla_{M,N}(A) = GH^T$. On passe donc de nm coefficients à seulement $\alpha(m+n)$.

Voici quelques exemples :

Structure	M	N	α
Tœplitz	Z_1	Z	≤ 2
Vandermonde	$D(\mathbf{x})$	Z	≤ 1
Cauchy	$D(\mathbf{x})$	$D(\mathbf{y})$	$= 1$

où $Z_\beta = \begin{bmatrix} 0 & & & \beta \\ 1 & \ddots & & \\ & \ddots & \ddots & \\ & & 1 & 0 \end{bmatrix}$, $Z = Z_0 = \begin{bmatrix} 0 & & & \\ 1 & \ddots & & \\ & \ddots & \ddots & \\ & & 1 & 0 \end{bmatrix}$ et $D(\mathbf{x}) = \begin{bmatrix} x_1 & & & \\ & \ddots & & \\ & & \ddots & \\ & & & x_m \end{bmatrix}$.

Les blancs correspondent à des coefficients nuls. Sauf pour les cas vraiment dégénérés, les bornes ci-dessus sont atteintes.

Définition 3 *On dit qu'une matrice $A \in \mathbb{K}^{m\times n}$ est structurée pour un opérateur de déplacement si son rang de déplacement α pour cet opérateur est petit par rapport à* $\min(n, m)$.
On dira que la matrice A est :

- *Tœplitz-like si l'opérateur de déplacement est $\nabla_{Z_1,Z}$.*
- *Vandermonde-like si l'opérateur de déplacement est $\nabla_{D(\mathbf{x}),Z}$.*
- *Cauchy-like si l'opérateur de déplacement est $\nabla_{D(\mathbf{x}),D(\mathbf{y})}$.*

Remarques :

1. Les conditions sur M et N pour la définition de $\nabla_{M,N}$ sont là pour assurer l'inversibilité de l'opérateur de déplacement considéré. En effet, on veut que la donnée de l'opérateur de déplacement (M et N) et des générateurs (G et H) soit suffisante pour nous permettre de reconstruire la matrice de départ. Le problème est illustré par :

$$\nabla_{Z,Z}\Big(\begin{bmatrix} a_0 & a_{-1} & a_{-2} \\ a_1 & a_0 & a_{-1} \\ a_2 & a_1 & a_0 \end{bmatrix}\Big) = \begin{bmatrix} a_{-1} & a_{-2} & 0 \\ 0 & 0 & a_{-1} \\ 0 & 0 & a_{-2} \end{bmatrix}$$

   Ici, nous ne sommes plus en mesure de reconstruire la matrice Tœplitz de départ car a_0, a_1 et a_2 n'apparaissent plus dans le résultat de la transformation.

2. Lorsque l'opérateur $\nabla_{M,N}$ est inversible, nous disposons de formules explicites pour reconstruire la matrice de départ [21, §4]. Pour $M = D(\mathbf{x})$ et $N = D(\mathbf{y})$ par exemple (cas Cauchy-like), on a :

$$A = \sum_{i=1}^{\alpha} D(\mathbf{g_i})C(\mathbf{x},\mathbf{y})D(\mathbf{h_i}) \tag{2}$$

   où $\mathbf{g_i}$ (resp. $\mathbf{h_i}$) est la i^{e} colonne de G (resp. H) et $C(\mathbf{x},\mathbf{y}) = \Big(\ \frac{1}{x_i - y_j}\ \Big)_{1\le i,j\le n}$.

3. On voit parfois [21, §1] la définition suivante : $\Delta_{M,N}(A) = A - MAN$. Ce type d'opérateur est moins pratique à utiliser que l'opérateur $\nabla_{M,N}(A)$ d'une part parce que la condition sur M et N pour que $\Delta_{M,N}$ soit inversible est plus compliquée, et d'autre part parce que les théorèmes comme celui que nous allons voir juste après s'expriment moins facilement dans le cas de $\Delta_{M,N}$.

2.3 Intérêt des matrices structurées

Une des propriétés les plus fondamentales des matrices denses structurées est le fait que la structure est conservée par les opérations habituelles [21, §1] :

Propriété 1

- *Si A et B sont structurées pour $\nabla_{M,N}$, alors $A + B$ l'est également.*
- *Si A est structurée pour $\nabla_{M,N}$ et B est structurée pour $\nabla_{N,P}$, alors AB est structurée pour $\nabla_{M,P}$.*
- *Si A est structurée pour $\nabla_{M,N}$, alors A^T est structurée pour ∇_{N^T,M^T}.*
- *Si A est structurée pour $\nabla_{M,N}$, alors A^{-1} est structurée pour $\nabla_{N,M}$.*

Pour l'inverse d'une matrice structurée, on a même le résultat très précis suivant :

Théorème 1 *Si A est une matrice inversible et structurée pour l'opérateur $\nabla_{M,N}$ alors A^{-1} est une matrice structurée pour l'opérateur $\nabla_{N,M}$ et on a :*

$$\mathrm{rg}\,(\nabla_{M,N}(A^{-1})) = \mathrm{rg}\,(\nabla_{M,N}(A)).$$

De plus, si $\nabla_{M,N}(A) = GH^T$ et si $Y = -A^{-1}G, Z^T = H^T A^{-1}$, alors :

$$\nabla_{N,M}(A^{-1}) = YZ^T.$$

La plupart des applications pratiques nécessitent la résolution de systèmes linéaires $Ax = b$. Lorsque A est une matrice dense structurée donnée par ses générateurs (G, H), l'approche la plus couramment utilisée consiste à calculer des générateurs (Y, Z) de l'inverse à partir de G et H, puis à utiliser une formule de reconstruction comme, par exemple, (2) pour calculer le produit $A^{-1}b$. Schématiquement :

$$A \xrightarrow{\text{Compression}} (G,H) \xrightarrow[\text{les générateurs}]{\text{Travail sur}} (Y,Z) \xrightarrow{\text{Reconstruction}} A^{-1}b$$

2.4 État de l'art

La littérature sur les matrices est assez vaste. On sait depuis longtemps inverser des matrices denses de taille $n \times n$ en $O(n^3)$ opérations arithmétiques de base grâce à l'élimination de Gauss, et même en $O(n^\omega)$ avec $\omega < 2.38$ (voir [5, §21]). Pour les matrices denses structurées de rang de déplacement α, nous avons une représentation en $2\alpha n$ coefficients de la matrice contre n^2 dans le cas dense non structuré. Cela permet de résoudre les problèmes d'inversion et de résolution de systèmes linéaires en un temps nettement inférieur à $O(n^\omega)$ asymptotiquement. Précisons que nous entendons ici par inversion le fait de calculer des générateurs de l'inverse d'une matrice structurée, et non l'inverse lui-même. Il y a essentiellement deux catégories d'algorithmes connus pour les matrices denses structurées.

La première regroupe les algorithmes dits «fast», c'est à dire ceux dont le coût sera quadratique en n. Le premier algorithme de ce type a été décrit par Levinson en 1947 (voir [18]) pour résoudre en $O(n^2)$ des systèmes linéaires faisant intervenir une matrice Tœplitz. Cet algorithme a été amélioré à de nombreuses reprises par la suite. Après l'apparition de la notion de rang de déplacement en 1979 [13], plusieurs algorithmes quadratiques sont apparus pour traiter diverses classes de matrices structurées. On peut citer l'algorithme GKO [7] qui permet de calculer une décomposition PLU de matrices Cauchy-like et Vandermonde-like, ainsi que l'algorithme de [14, §1.10] pour inverser des matrices Cauchy-like et Vandermonde-like, tout deux en $O(\alpha n^2)$. Ces deux derniers algorithmes sont déterministes, puisqu'ils reposent sur la factorisation et l'élimination de Gauss avec pivotage.

La seconde catégorie est celle des algorithmes «superfast», c'est à dire quasi-linéaires en n. Ces algorithmes reposent essentiellement sur une approche récursive et sur l'utilisation du produit de polynômes via l'algorithme de FFT pour effectuer des produits (matrice structurée) $\times$ vecteur en temps quasi-linéaire (voir [8]). Le premier algorithme de ce type est apparu au début des années 1980 dans [2] et [19]. Il s'agit d'un algorithme pour inverser les matrices Tœplitz-like fortement régulières en $O^\sim(\alpha^2 n)$ (la notation $O^\sim$ est employée lorsqu'on omet les termes en log pour la complexité). Kaltofen a montré dans [15] comme enlever cette hypothèse de régularité en appliquant un préconditionnement structuré aléatoire, conduisant ainsi à un algorithme probabiliste. On trouve dans [4] et [22] des adaptations pour traiter le cas des matrices Cauchy-like. L'algorithme présenté dans [4] a le mérite de simplifier significativement l'algorithme initial. Enfin, l'étude récente [3] montre comment introduire du produit de matrices denses afin de réduire le coût pour

cette approche à $O^{\sim}(\alpha^{\omega-1}n)$, toujours sous l'hypothèse de régularité ou probabiliste. On peut trouver un résumé de cette approche ainsi que des remarques intéressantes sur la symétrisation (autre moyen de supprimer l'hypothèse de régularité si $\mathbb{K} = \mathbb{R}$ ou $\mathbb{C}$) dans les chapitres 4 et 5 de [21]. Néanmoins, si on se restreint à la classe des matrices mosaïques (Tœplitz par blocs avec des tailles de blocs variables), on peut alors inverser de manière déterministe en temps quasi-linéaire en n (voir [10],[17] et [16]) grâce à la version rapide de [6] de l'algorithme de calcul de σ-bases de Beckermann et Labahn [1].

2.5 Plan

À la section 3, nous allons voir en détail le résultat de Gohberg, Kailath et Olshevsky [7] qui a conduit à leur algorithme de factorisation pour les matrices Cauchy-like inversibles. La suite de cette section sera alors consacrée à la présentation de cet algorithme et de ses applications.

La section 4 sera consacrée à la présentation d'un nouvel algorithme pour factoriser les matrices Cauchy-like quelconques. Le coût de cet algorithme est $O(\alpha r(m+n))$ pour une matrice $m \times n$ de rang r. C'est intéressant car on a un coût sensible à la fois au rang de la matrice r et à son rang de déplacement α. Je parlerai de mon implantation de l'algorithme et je montrerai les timings que j'ai réalisés pour illustrer ce point.

Je passerai alors aux algorithmes de la catégorie «superfast» à la section 5. J'y détaillerai brièvement certains points permettant d'obtenir un coût quasi-linéaire en n avant de parler de ma tentative pour résoudre certains systèmes linéaires directement à l'aide de l'algorithme de σ-bases, puis de ma redécouverte de l'algorithme d'inversion de Cardinal [4]. Nous verrons enfin ce que mon approche permet d'apporter par rapport à [4].

Enfin, j'ai mis pour référence en annexe un algorithme «fast» pour inverser les matrices Cauchy-like. Il s'agit d'une adaptation personnelle de l'algorithme de Kailath et Sayed [14] au cas des matrices Cauchy-like quelconques (l'algorithme initial traite une classe plus générale, mais avec la supposition classique que la matrice est symétrique, donc en particulier fortement régulière). C'est en passant à une version rapide de cet algorithme que j'ai retrouvé l'algorithme de Cardinal.

3 Factorisation des matrices Cauchy-like

Nous allons voir dans cette section comment exploiter la structure d'une matrice Cauchy-like. En fait, il s'agit essentiellement de faire tout le travail sur les générateurs plutôt que sur la matrice elle-même. Comme la taille des générateurs est de $2\alpha n$ coefficients contre n^2 pour la matrice, on peut ainsi gagner en complexité spatiale et temporelle.

3.1 Quelques propriétés remarquables

L'intérêt porté aux matrices Cauchy-like est justifié par quelques propriétés remarquables propres à cette structure. Bien que les applications pratiques fassent intervenir plutôt des matrices Tœplitz-like, il peut être intéressant de se ramener au cas Cauchy-like pour traiter les problèmes.

Dans la suite, on se donne une matrice $A \in \mathbb{K}^{n\times n}$ Cauchy-like, deux vecteurs $\mathbf{x}, \mathbf{y} \in \mathbb{K}^n$ et des générateurs $G, H \in \mathbb{K}^{n\times\alpha}$ de A pour l'opérateur de déplacement $\nabla_{D(\mathbf{x}),D(\mathbf{y})}$.

Il est facile de calculer un coefficient de la matrice A à partir de $\mathbf{x}, \mathbf{y}, G$ et H :

Propriété 2 *Le coefficient (i,j) de A est donné par la formule*

$$a_{i,j} = \frac{G_{i,*}\, H_{j,*}^T}{x_i - y_j} \tag{3}$$

où $G_{i,*}$ (resp. $H_{j,*}$) est la i^e ligne de G (resp. la j^e ligne de H).

Cela vient du fait que l'action de l'opérateur $\nabla_{D(\mathbf{x}),D(\mathbf{y})}$ revient à multiplier le coefficient (i,j) d'une matrice par $x_i - y_j$ (voir l'exemple à la section 2.2).

À partir de cette formule, nous sommes en mesure de calculer certaines sous-matrices de A. Cela s'avère utile quand on veut appliquer une élimination de Gauss avec pivotage sur A. En effet, on peut reconstruire tout une ligne de A afin de trouver un pivot non nul. Il reste à voir comment ramener ce pivot sur la diagonale. Pour cela, on a la propriété suivante :

Propriété 3 *Si A est une matrice Cauchy-like et P est une matrice de permutation, alors PA est encore une matrice Cauchy-like. Plus précisément, si $\nabla_{D(\mathbf{x}),D(\mathbf{y})}(A) = GH^T$, alors $\nabla_{D(P\mathbf{x}),D(\mathbf{y})}(PA) = (PG)H^T$.*

En effet, il suffit de multiplier par P à gauche dans (1) et de remarquer que $PD(\mathbf{x})P^{-1} = D(P\mathbf{x})$. En réalité, la seule chose importante pour avoir conservation de la structure lorsqu'on applique une permutation est d'avoir un opérateur de déplacement défini avec des matrices diagonales : si M est diagonale, on pourra permuter à gauche et si N est diagonale, on pourra permuter à droite.

3.2 Génération du complément de Schur

Commençons par quelques rappels sur les compléments de Schur. On se donne une matrice $A \in \mathbb{K}^{n\times n}$ que l'on découpe de la manière suivante : $A = \left[\begin{array}{c|c} A_{1,1} & A_{1,2} \\ \hline A_{2,1} & A_{2,2} \end{array}\right]$ avec $A_{1,1} \in \mathbb{K}^{k\times k}$ pour un certain $k \in \{1, \ldots, n-1\}$.

Définition 4 *Si $A_{1,1} \in \mathbb{K}^{k\times k}$ est inversible, on définit le complément de Schur de $A_{1,1}$ dans A comme étant $S = A_{2,2} - A_{2,1}(A_{1,1})^{-1}A_{1,2} \in \mathbb{K}^{(n-k)\times(n-k)}$.*

Le complément de Schur apparaît naturellement dès que l'on utilise un algorithme basé sur une élimination de Gauss par blocs. En effet, on a :

Propriété 4 *Pour $A_{1,1}$ inversible et en notant $E = \left[\begin{array}{c|c} I_k & \\ \hline -A_{2,1}A_{1,1}^{-1} & I_{n-k} \end{array}\right]$ et $F = \left[\begin{array}{c|c} I_k & -A_{1,1}^{-1}A_{1,2} \\ \hline & I_{n-k} \end{array}\right]$, on a :*

$$EA = \left[\begin{array}{c|c} A_{1,1} & A_{1,2} \\ \hline & S \end{array}\right], \quad AF = \left[\begin{array}{c|c} A_{1,1} & \\ \hline A_{2,1} & S \end{array}\right], \quad EAF = \left[\begin{array}{c|c} A_{1,1} & \\ \hline & S \end{array}\right].$$

De plus, le complément de Schur apparaît dans la formule d'inversion par blocs :

Propriété 5 *Si A et $A_{1,1}$ sont inversibles, alors S est inversible et on a l'expression suivante pour l'inverse de A :*

$$A^{-1} = F \left[\begin{array}{c|c} A_{1,1}^{-1} & \\ \hline & S^{-1} \end{array}\right] E = \left[\begin{array}{c|c} A_{1,1}^{-1} + A_{1,1}^{-1}A_{1,2}S^{-1}A_{2,1}A_{1,1}^{-1} & -A_{1,1}^{-1}A_{1,2}S^{-1} \\ \hline -S^{-1}A_{2,1}A_{1,1}^{-1} & S^{-1} \end{array}\right]. \tag{4}$$

L'inversibilité du complément de Schur dans le cas où A est inversible vient de la formule d'élimination $EA = \left[\begin{array}{c|c} A_{1,1} & A_{1,2} \\ \hline & S \end{array}\right]$. Comme A et E sont inversibles, $\left[\begin{array}{c|c} A_{1,1} & A_{1,2} \\ \hline & S \end{array}\right]$ l'est aussi. Or, comme il s'agit d'une matrice triangulaire par blocs, on en déduit que ses blocs diagonaux (et en particulier S) sont inversibles.

Au passage, on voit sur (4) que le bloc inférieur droit de A^{-1} est S^{-1}.

Le complément de Schur est très intéressant du point de vue des matrices denses structurées car il a une structure similaire à celle de la matrice dont il est issu. Plus précisément, on a le théorème suivant, qui est une version par blocs d'un lemme de Gohberg, Kailath et Olshevsky [7, Lemme 1.1] :

Théorème 2 *Si $\nabla_{M,N}(A) = GH^T$ avec $G, H \in \mathbb{K}^{n\times\alpha}$, $M = \left[\begin{array}{c|c} M_{1,1} & \\ \hline M_{2,1} & M_{2,2} \end{array}\right], M_{1,1} \in \mathbb{K}^{k\times k}$ et $N = \left[\begin{array}{c|c} N_{1,1} & N_{1,2} \\ \hline & N_{2,2} \end{array}\right], N_{1,1} \in \mathbb{K}^{k\times k}$, et si $A_{1,1} \in \mathbb{K}^{k\times k}$ est inversible pour un certain $k \in \{1, \ldots, n-1\}$, alors le complément de Schur de $A_{1,1}$ dans A est bien défini, et son rang de déplacement pour $\nabla_{M_{2,2},N_{2,2}}$ est majoré par α.*

De plus, si on définit $G', H' \in \mathbb{K}^{(n-k)\times\alpha}$ par $\left[\begin{array}{c} * \\ \hline G' \end{array}\right] = EG$ et $\left[\begin{array}{c} * \\ \hline H' \end{array}\right] = F^T H$ où E et F les matrices d'élimination de la propriété 4, alors :

$$\nabla_{M_{2,2},N_{2,2}}(S) = G'H'^T.$$

Démonstration :
En multipliant à gauche par E et à droite par F dans (1), on arrive à :

$$EMAF - EANF = EGH^TF. \tag{5}$$

Il suffit alors de remarquer que EM et AF ont les formes suivantes :

- $EM = \left[\begin{array}{c|c} * & * \\ \hline * & I_{n-k} \end{array}\right] \left[\begin{array}{c|c} * & \\ \hline * & M_{2,2} \end{array}\right] = \left[\begin{array}{c|c} * & * \\ \hline * & M_{2,2} \end{array}\right]$.
- $AF = \left[\begin{array}{c|c} A_{1,1} & \\ \hline A_{2,1} & S \end{array}\right]$ d'après la propriété 4.

Donc $(EM)(AF) = \left[\begin{array}{c|c} * & * \\ \hline * & M_{2,2}S \end{array}\right]$. De même, $(EA)(NF) = \left[\begin{array}{c|c} * & * \\ \hline * & SN_{2,2} \end{array}\right]$.
Finalement, en considérant le bloc inférieur droit dans (5), on obtient $M_{2,2}S - SN_{2,2} = G'H'^T$. □

Remarques :

1. Pour $k = 1$, on retrouve [7, Lemme 1.1].
2. Si M et N sont triangulaires (et pas simplement triangulaires par blocs), alors $M_{2,2}$ et $N_{2,2}$ sont encore triangulaires et on peut donc appliquer ce théorème récursivement sur le complément de Schur S.

3. Parmi les cas où M et N sont triangulaires, on a les deux cas suivants importants en pratique :
   - les matrices Cauchy-like où M et N sont diagonales.
   - les matrices Vandermonde-like où M est diagonale et $N = Z^T$.

3.3 Factorisation PLU

Rappelons quelques définitions :

Définition 5 *On dira d'une matrice triangulaire qu'elle est unitaire si tous ses coefficients diagonaux sont égaux à* 1.

Définition 6 *Si A est une matrice inversible, alors on peut trouver un triplet (P, L, U) tel que $A = PLU$ avec P une matrice de permutation, L triangulaire inférieure unitaire et U triangulaire supérieure inversible (donc avec des coefficients non nuls sur la diagonale). Une telle factorisation s'appelle une factorisation PLU.*

Remarques :

1. Une factorisation PLU peut être obtenue en faisant une élimination de Gauss où on fait des échanges de lignes pour assurer que les pivots soient non nuls. Si on procède à une élimination de Gauss avec des échanges entre colonnes seulement, on obtiendra à la place une factorisation LUP.
2. La factorisation PLU n'est pas unique comme le montre l'exemple suivant :
$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 3 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 0 & -2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1/3 & 1 \end{bmatrix} \begin{bmatrix} 3 & 4 \\ 0 & 2/3 \end{bmatrix}$$
3. Toute matrice inversible admet une factorisation PLU. En revanche, seule une partie des matrices admettent une factorisation LU (PLU avec $P = I_n$) : ce sont les matrices fortement régulières. En fait, les matrices fortement régulières sont les matrices inversibles pour lesquelles nous n'avons pas besoin de faire d'échange lors de l'élimination de Gauss (les pivots rencontrés sont déjà tous non nuls). Nous verrons dans la suite que les algorithmes récursifs rapides ne fonctionnent qui si la matrice en entrée est fortement régulière, alors que l'algorithme présenté ici fonctionne pour toute matrice inversible.

L'algorithme présenté ici, extrait de [7], a pour but de calculer une factorisation PLU d'une matrice Cauchy-like étant donnés l'opérateur de déplacement (couple de vecteurs $(\mathbf{x}, \mathbf{y})$) et des générateurs $G, H \in \mathbb{K}^{n \times \alpha}$. L'idée est de progresser de 1 en 1 le long de la diagonale principale, de pivoter si nécessaire afin d'avoir un coefficient non nul comme pivot, et d'éliminer progressivement. On peut alors continuer à travailler sur le complément de Schur. Les matrices L et U sont obtenues en «juxtaposant» les matrices d'élimination utilisées à chaque étape. La notation $\mathbf{x}_{i:j}$ désigne le vecteur $\begin{bmatrix} x_i & \dots & x_j \end{bmatrix}^T$.

La correction de l'algorithme 1 vient du théorème 2 d'une part, et du principe de pivotage abordé dans la section précédente d'autre part. Au niveau de la complexité, le coût (en nombre d'opérations arithmétiques) est dominé à la fois par le calcul de L et U et la mise à jour de G et H. On arrive à un total de $4\alpha n^2 + O(\alpha n + n^2)$ opérations arithmétiques.

Le surcoût par rapport à l'élimination de Gauss (qui est en au plus $\frac{2}{3}n^3$ opérations arithmétiques) quand α devient proche de n est dû à trois choses :

Algorithme 1 : GKO (issu de [7])

Entrée : générateurs $G, H \in \mathbb{K}^{n\times\alpha}$ d'une matrice A Cauchy-like et vecteurs $\mathbf{x}, \mathbf{y} \in \mathbb{K}^n$ définissant l'opérateur de déplacement

Sortie : une factorisation PLU de la matrice A si elle est inversible, et un message d'erreur sinon

début

$P \leftarrow I_n$

$L \leftarrow$ matrice nulle de taille $n \times n$

$U \leftarrow$ matrice nulle de taille $n \times n$

pour k *allant de* 1 *à* n **faire**

$L_{k:n,k} \leftarrow \left[\frac{G_{k,*}\,H_{k,*}^T}{x_k - y_k} \quad \dots \quad \frac{G_{j,*}\,H_{k,*}^T}{x_j - y_k} \quad \dots \quad \frac{G_{n,*}\,H_{k,*}^T}{x_n - y_k} \right]^T$

$q \leftarrow \min\{j, L_{j,k} \neq 0\}$

si q *non défini* **alors**

retourner une erreur : matrice non inversible

si $q > k$ **alors**

échanger les lignes k et q de G

échanger les coefficients k et q de $\mathbf{x}$

échanger les lignes k et q de L

échanger les lignes k et q de P

$U_{k:n,k} \leftarrow \left[\frac{G_{k,*}\,H_{k,*}^T}{x_k - y_k} \quad \dots \quad \frac{G_{k,*}\,H_{j,*}^T}{x_k - y_j} \quad \dots \quad \frac{G_{k,*}\,H_{n,*}^T}{x_k - y_n} \right]^T$

$L_{k:n,k} \leftarrow \frac{1}{U_{k,k}} L_{k:n,k}$

$G_{k+1:n,1:\alpha} \leftarrow G_{k+1:n,1:\alpha} - L_{j+1:n,k} G_{k,1:\alpha}$

$H_{k+1:n,1:\alpha} \leftarrow H_{k+1:n,1:\alpha} - \frac{1}{U_{k,k}} U_{k,j+1:n} H_{k,1:\alpha}$

fin

1. On a deux générateurs de taille $\alpha \times n$ (G et H) pour représenter M de taille $n \times n$. Si $\alpha \approx n$, l'entrée est donc environ deux fois plus grosse.

2. Il faut reconstruire L et U dans cet algorithme alors qu'il suffit de lire les coefficients de la matrice courante dans l'algorithme de Gauss, ce qui conduit à un surcoût d'environ $2 \times 2\alpha k$ opérations arithmétiques à chaque étape.

3. Les générateurs sont de largeur α à chaque étape alors que la largeur de la matrice courante dans l'algorithme de Gauss diminue de 1 à chaque étape. Ainsi, on se retrouve avec $\sum_k \alpha k$ contre $\sum_k k^2$.

Pour résumer :

	Gauss	GKO
taille de l'entrée	n^2	$2 \times \alpha n$
nombre d'étapes	n	n
taille des données à l'étape k	k^2	$2 \times \alpha k$
coût arithmétique pour obtenir $L_{k+1:n,k}$ et $U_{k,k:n}$		$4\alpha k$
coût arithmétique total à l'étape k	$2k^2$	$2 \times 2\alpha k + 4\alpha k$
coût arithmétique total	$\frac{2}{3}n^3$	$4\alpha n^2$

3.4 Applications

Cet algorithme a été introduit par Gohberg, Kailath et Olshevsky dans [7] pour résoudre numériquement des systèmes linéaires faisant intervenir des matrices Tœplitz-like. L'intérêt de passer par Cauchy vient de la possibilité de pivoter. Grâce à ce pivotage, non seulement on supprime l'hypothèse de régularité puisqu'on a la garantie d'avoir un pivot non nul à chaque étape, mais on peut aller plus loin en choisissant comme pivot l'élément le plus grand en valeur absolue de la colonne, ce qui augmente considérablement la stabilité numérique (voir la dernière partie de [7]).

Il est possible de calculer des générateurs de l'inverse d'une matrice Cauchy-like en passant par la factorisation PLU. En effet, comme indiqué dans la section 8 de [9], il est possible d'obtenir des générateurs pour l'inverse via les équations $-G = AY = PLUY$ et $H = A^T Z = U^T L^T P^T Z$ (conséquence du théorème 1). Comme on a déjà calculé P, L et U, il reste juste à résoudre 4α systèmes triangulaires (G et H sont de largeur α). L'ennui avec cette approche est le coût en mémoire. En effet, comme il faut stocker L et U, nous avons besoin d'une mémoire de l'ordre de n^2. Il est possible de réduire le coût mémoire à $O(\alpha n)$ en utilisant l'approche astucieuse de Kailath et Sayed [14, §1.10]. J'ai réécrit leur algorithme (présenté dans le cas de matrices symétriques pour lesquelles le théorème 2 s'applique). On pourra voir en annexe ma version pour les matrices Cauchy-like générale (pas forcément fortement régulières), qui a servi de base à l'algorithme présenté au paragraphe 5.2.

Enfin, Olshevsky et Shokrollahi font remarquer dans [20] que si la matrice passée en entrée de l'algorithme GKO n'est pas inversible, il est possible de renvoyer un vecteur du noyau de cette matrice plutôt qu'une simple erreur. Si la matrice est de rang r, on se retrouve dans ce cas de figure au plus tard à l'étape r. L'algorithme de recherche d'un vecteur du noyau a donc un coût en $O(\alpha r n)$. Dans la section suivante, nous allons voir comment étendre cette remarque pour obtenir une factorisation complète d'une matrice dense structurée de rang r en le même coût.

4 Factorisation LSP pour les matrices Cauchy-like

La factorisation LSP est une généralisation de la factorisation LUP permettant de traiter les matrices non inversibles. Cette notion a été introduite par Ibarra, Moran et Hui dans [11] :

Définition 7 *Une matrice $A \in \mathbb{K}^{m\times n}$ peut toujours se mettre sous la forme LSP avec $L \in \mathbb{K}^{m\times m}$ triangulaire inférieure unitaire, $S \in \mathbb{K}^{m\times n}$ semi-triangulaire supérieure (si on enlève les lignes de 0 alors S est triangulaire supérieure avec des coefficients non nuls sur la diagonale) et $P \in \mathbb{K}^{n\times n}$ une matrice de permutation.*

L'avantage de cette décomposition est qu'elle révèle le rang de la matrice, ainsi que son profil de rang (l'emplacement des pivots). Avec $m = 4, n = 5, r = 2$, cela ressemble à :

$$A = LSP = \begin{bmatrix} 1 & & & \\ . & 1 & & \\ . & . & 1 & \\ . & . & . & 1 \end{bmatrix} \begin{bmatrix} \times & . & . & . & . \\ & & & & \\ & \times & . & . & . \\ & & & & \end{bmatrix} \begin{bmatrix} & & 1 & & \\ 1 & & & & \\ & 1 & & & \\ & & & & 1 \\ & & & 1 & \end{bmatrix}$$

où $\times$ désigne des coefficients forcément non nuls et . des coefficients quelconques.

4.1 Algorithme pour calculer une factorisation LSP

Algorithme 2 : LSP pour les matrices Cauchy-like

Entrée : matrice A de taille $m \times n$ et de rang r, donnée par ses générateurs G et H pour l'opérateur de déplacement défini par les vecteurs $\mathbf{x}$ et $\mathbf{y}$

Sortie : matrices L, S et P telles que $A = LSP$, P permutation, L triangulaire inférieure unitaire, S quasi-triangulaire supérieure

début

1 $L \leftarrow I_m$
2 $S \leftarrow$ matrice nulle de taille $m \times n$
3 $P \leftarrow I_n$
4 $r \leftarrow 0$
5 modification de G et H afin de mettre G sous forme échelonnée
6 **tant que** $A \neq 0$ **faire**
7 $r \leftarrow r + 1$
8 $i, j \leftarrow$ indices du premier coefficient non nul de GH^T
9 échanger les lignes r et j de P
10 échanger les colonnes r et j de S
11 échanger les colonnes r et j de H
12 $d \leftarrow \frac{G_{i,1:\alpha} H_{r,1:\alpha}^T}{x_i - y_r}$
13 $L_{i+1..m,i} \leftarrow \begin{bmatrix} \frac{G_{i+1,1:\alpha} H_{r,1..\alpha}^T}{d(x_{i+1} - y_r)} & \cdots & \frac{G_{m,1:\alpha} H_{r,1:\alpha}^T}{d(x_m - y_r)} \end{bmatrix}^T$
14 $S_{i,r..n} \leftarrow \begin{bmatrix} \frac{G_{i,1:\alpha} H_{r,1:\alpha}^T}{x_i - y_r} & \cdots & \frac{G_{i,1:\alpha} H_{n,1:\alpha}^T}{x_i - y_n} \end{bmatrix}$
15 $G_{i+1:m,1:\alpha} \leftarrow G_{i+1:m,1:\alpha} - L_{i+1:m,k} G_{i,1:\alpha}$
16 $H_{r+1:n,1:\alpha} \leftarrow H_{r+1:n,1:\alpha} - \frac{1}{d} S_{i,r+1:n}^T H_{r,1:\alpha}$
17 modification de G et H pour remettre G sous forme échelonnée
18 Renvoyer L, S, P

fin

L'idée de cet algorithme est de généraliser l'algorithme GKO vu précédemment afin de renvoyer une factorisation LSP. La différence intervient lors de la recherche d'un élément non nul pour pivoter. Alors qu'on limite la recherche à une seule colonne dans GKO (la colonne étant non nulle si la matrice est inversible), on cherche ici le premier coefficient non nul de la matrice (en la parcourant ligne par ligne).

La difficulté vient du fait que la matrice n'est donnée que via un couple de générateurs. Dans le cas de GKO, nous pouvions nous permettre de reconstruire une ligne et une colonne à chaque étape. Ici, nous voulons un coup sensible au rang r de la matrice. Nous ne pouvons donc pas reconstruire les n lignes une à une pour voir si elles sont nulles ou pas. Heureusement, en imposant une forme échelonnée pour G, nous disposons d'un critère simple pour trouver la première ligne non nulle de la matrice A à la simple vue de G et H.

Cette forme échelonnée (ligne 5 dans l'algorithme 2) est en fait le résultat de l'élimination de Gauss en colonnes appliquée à G. Un exemple de forme échelonnée est donné à la figure 1. Pour l'avoir au début de l'algorithme, on calcule E telle que GE soit sous forme échelonnée et on change les générateurs (G, H) en $(GE, H(E^{-1})^T)$, pour un coût total en $O(\alpha^2(m+n))$. Mal-

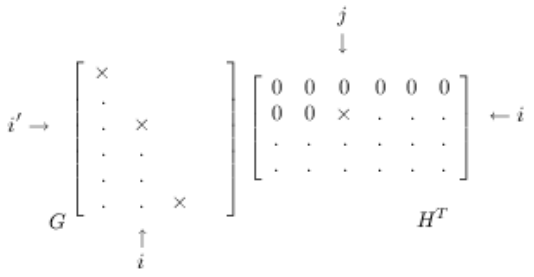


FIG. 1 – Recherche d'un pivot grâce à un parcours des générateurs

heureusement, la structure est légèrement perturbée lors du passage au complément de Schur (lignes 15 et 16), ce qui nécessite de refaire une élimination de Gauss en colonnes à chaque étape de l'algorithme (ligne 17). Toutefois, le coût descend à $O(\alpha(m+n))$ car il y a très peu de coefficients à éliminer (au plus α) puisqu'on part d'une matrice sous forme quasi-échelonnée (échelonnée si on enlève la première colonne).

Supposons que G soit effectivement sous forme échelonnée. La recherche du premier coefficient non nul de A se fait alors de la manière suivante :

1. recherche du premier coefficient non nul de H^T. On note (i,j) l'indice de ce coefficient s'il existe (sinon, $H=0$ donc $\nabla(A)=0$ donc $A=0$) ;
2. recherche du i^{e} échelon dans G. On note i' l'indice de la ligne correspondant à cet échelon s'il existe (sinon, $G = \left[\begin{array}{c|c} G' & 0 \end{array}\right]$, $H^T = \left[\begin{array}{c} 0 \\ \hline H'^T \end{array}\right]$ donc $\nabla(A)=0$ donc $A=0$) ;
3. renvoyer (i',j).

Ce procédé est illustré à la figure 1.

Pour ce qui est du coût de l'algorithme :
- le prétraitement (ligne 5) est en $O(\alpha^2(m+n))$.
- chaque passage dans la boucle est en $O(\alpha(m+n))$: le coût est majoré à la fois par la recherche de (i,j) (ligne 8), le calcul de la nouvelle colonne de L (ligne 13), le calcul de la nouvelle ligne de S (ligne 14), la mise à jour des générateurs pour passer au complément de Schur (lignes 15-16) et la mise sous forme echelonnée de G (ligne 18).
- on fait autant d'étapes que de lignes non nulles rencontrées au cours de l'élimination de Gauss, c'est à dire r.

Au final, cela donne un coût en $O(\alpha^2(m+n))+O(\alpha r(m+n))$. Or, on a l'inégalité suivante :

$$\alpha = \operatorname{rg}(D(\mathbf{x})A - AD(\mathbf{y})) \leq \operatorname{rg}(D(\mathbf{x})A) + \operatorname{rg}(AD(\mathbf{y})) \leq 2\operatorname{rg}(A) = 2r.$$

Ainsi, le coût de l'algorithme 2 est de $O(\alpha r(m+n))$ opérations arithmétiques.

4.2 Applications

Comme mentionné précédemment, le principal intérêt de la décomposition LSP vient de la possibilité de lire le profil de rang de la matrice directement sur S. En effet, le profil de rang est donné par les lignes non nulles de S.

Les autres applications classiques de la LSP sont listées à la fin de [12]. Grâce à cette décomposition, on peut :

1. rendre la matrice A de départ fortement régulière en $O(mn)$ (et même $O(\alpha(m+n))$ si on se contente des générateurs). En notant Q la permutation qui fait remonter dans le même ordre toutes les lignes non nulles de S, on a en effet que QAP^T est fortement régulière.
2. résoudre des systèmes linéaires, c'est-à-dire trouver une solution ou prouver qu'il n'y en a pas, en $O(r(m+n))$.

4.3 Implantation et timings pour l'algorithme de LSP

Dans cette section, je présente des courbes obtenues à partir d'une implantation en Maple de l'algorithme de factorisation LSP que nous venons de voir. Les tests ont été effectués sur une machine munie d'un processeur Intel Core 2 Duo à 1,83Ghz, sous Debian, avec la version 10 de Maple.

À la figure 2, on voit le temps mis par le code en fonction de n. Comme les générateurs sont pris aléatoirement, le rang r vaut environ n, donc on trouve un temps quadratique en n.

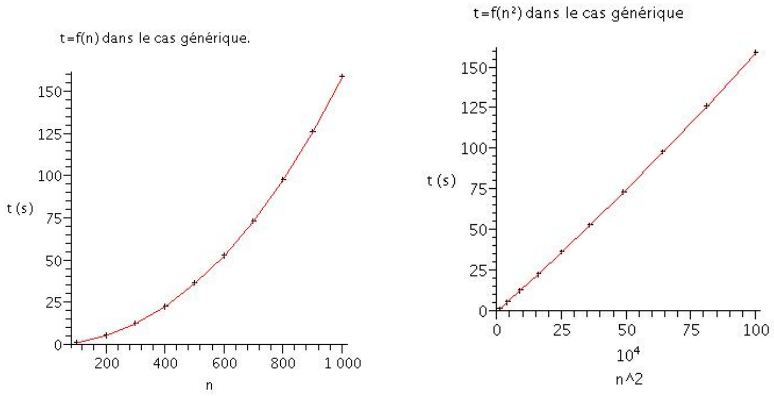


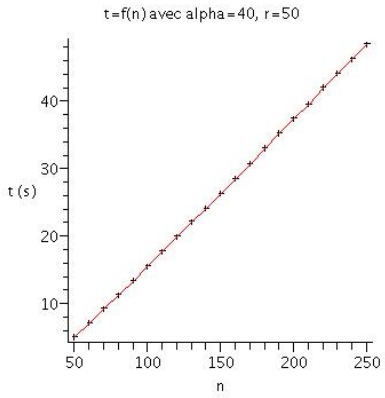
FIG. 2 – Temps en secondes en fonction de n (à α fixé)

Si on fixe le rang r, la courbe obtenue est cette fois linéaire en n, comme le montre la figure 3(a). À n et r fixés, on trouve bien un temps linéaire par rapport à α dans le corps de la boucle (courbe rouge) et un temps quadratique en α pour le prétraitement (ligne 5 de l'algorithme 2). La figure 3(b) montre ce qu'on obtient en prenant $n = r = 300$ et en faisant varier α. On constate que le prétraitement finit par coûter plus cher que le corps de l'algorithme. Toutefois, cela ne se produit que pour des valeurs de α proches de n, et donc inintéressantes en pratique.

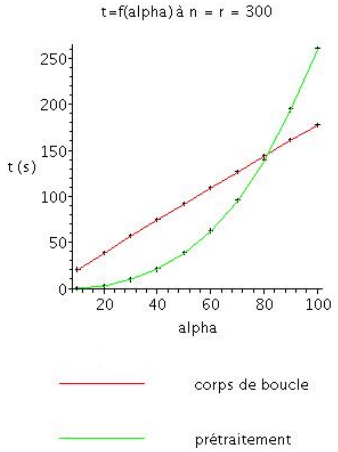
Enfin, nous pouvons voir ce que l'on obtient lorsqu'on fixe n et α et que l'on fait varier r à la figure 4 ($n = 200$ et $\alpha = 10$). On constate avec la courbe de gauche que le temps est bien linéaire en r si la supposition r petit par rapport à n est valide. En revanche, si r devient proche de n, les dernières étapes (dont le coût dépend de $n - r$) deviennent de moins en moins coûteuse, ce qui explique la courbe obtenue à droite.

5 Approches à base d'arithmétique polynomiale

Dans cette partie, nous allons voir deux approches faisant intervenir des polynômes pour résoudre des problèmes avec des matrices denses structurées. Le passage à du calcul sur les polynômes est motivé par l'algorithme de FFT qui permet de multiplier deux polynômes univariés de degré n en $M(n)$ opérations de base, avec $M(n) = O(n \log(n) \log\log(n))$ (voir [5, §8]).

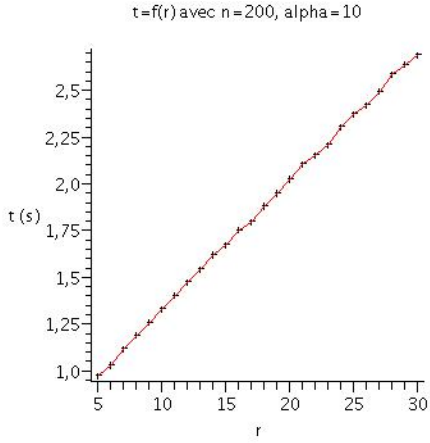


(a) Temps en secondes en fonction de n (à α et r fixés)



(b) Temps en secondes en fonction de α (à n et r fixés)

FIG. 3 – Temps en secondes à r fixé



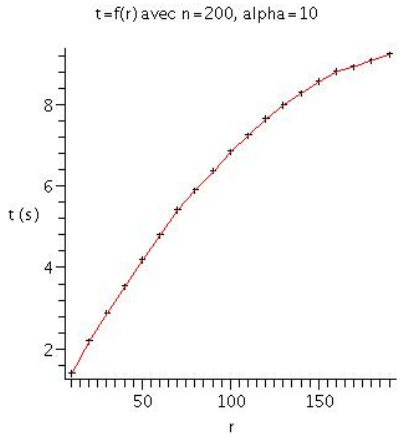


FIG. 4 – Temps en secondes en fonction de r (à n et α fixés)

5.1 Utilisation de l'algorithme de calcul de σ-bases

Dans [17] et [16], le cas de l'inversion des matrices mosaïques (Tœplitz par blocs avec des blocs de tailles variables) est traité de la manière suivante :

1. on associe un polynôme à chaque bloc,
2. on résout quatre problèmes d'approximation de matrices polynomiales à l'aide de l'algorithme de calcul de σ-bases,
3. on transforme les polynômes en sortie pour obtenir des générateurs de l'inverse.

D'une part, l'introduction de polynômes est assez naturelle dans la mesure où un produit (matrice Tœplitz) $\times$ vecteur peut être effectué à l'aide d'un produit de deux polynômes. D'autre part, les problèmes d'approximation de matrices polynomiales sont, dans ce cadre particulier, un moyen de résoudre les systèmes $AY = -G$ et $A^T Z = H$, ce qui permet d'obtenir les générateurs pour l'inverse vu au théorème 1.

Le coût de cette approche est dominée par l'appel à l'algorithme de calcul de σ-bases. Si les blocs sont de taille régulière, on atteint un coût en $O^{\sim}(\alpha^{\omega-1}n)$. En revanche, dans les cas

de blocs très rectangulaires (largeur petite par rapport à la longueur), on arrive a un coût en $O^{\sim}(\alpha^{\omega} n)$.

En m'inspirant de cette approche, j'ai essayé de traiter la résolution de système dans le cas Tœplitz-like de manière directe. L'idée était d'introduire l'approximation polynomiale au niveau de la formule de reconstruction pour les matrices Tœplitz-like. Malheureusement, le coût atteint était en $O^{\sim}(\alpha^{\omega} n)$, ce qui est nettenement moins bien que le coût en $O^{\sim}(\alpha^{\omega-1} n)$ obtenu dans [3] avec un algorithme probabiliste.

5.2 Version rapide d'un algorithme d'inversion

En s'inspirant des algorithmes rapides que nous venons de voir, on arrive à une nouvelle version de l'algorithme d'inversion pour les matrices Cauchy-like. Cet algorithme mélange à la fois le découpage en deux comme dans [2] et [19], et les formules de l'algorithme de [14] permettant de travailler en place sur les générateurs (voir aussi en annexe). En fait, cette version correspond à l'approche de Cardinal [4] pour les matrices Cauchy-like mais la présentation faite ici est plus précise et couvre le cas où l'opérateur de déplacement est $\nabla_{M,N}$ avec M et N^T triangulaires inférieures.

L'inconvénient, c'est que nous sommes obligés de supposer que la matrice en entrée est fortement régulière ou d'appliquer les principes de randomisation de [15]. Le vrai gain par rapport à [2] et [19] vient essentiellement de la disparition de la phase de compression, ce qui rend l'algorithme plus simple à écrire, à analyser et à implanter.

Dans l'algorithme 3, $\bar{x}$ (resp. $\bar{X}$) désignera la partie haute de x (resp. X), c'est-à-dire les $\lfloor \frac{n}{2} \rfloor$ premiers coefficients de x (resp. premières lignes de X). De même, $\underline{x}$ (resp. $\underline{X}$) désignera l'autre partie de x (resp. X).

La correction de l'algorithme 3 vient des deux points suivants :

1. (G_S, Y_S) est bien un générateur pour le complément de Schur de $A_{1,1}$ dans A.
2. on a $Y = -A^{-1}G$ et $Z = (A^{-1})^T H$

Démonstration : La démonstration se fait par récurrence sur $n \in \mathbb{N}^*$:

- Pour $n = 1$, il n'y a pas de complément de Schur, et $A = \left[\begin{array}{c} d \end{array}\right]$. On a donc bien $Y = -\frac{1}{d}G = -A^{-1}G$ et $Z = \frac{1}{d}H = (A^{-1})^T H$.
- Pour $n > 1$, le premier appel récursif nous donne $Y_{1,1} = -A_{1,1}^{-1}\bar{G}$ et $Z_{1,1} = (A_{1,1}^{-1})^T \bar{H}$ par hypothèse de récurrence.
  Grâce au théorème 2, on trouve que :

$$\left[\begin{array}{c} * \\ \hline G_S \end{array}\right] = \underbrace{\left[\begin{array}{c|c} I_k & \\ \hline -A_{2,1}A_{1,1}^{-1} & I_{n-k} \end{array}\right]}_{E} \left[\begin{array}{c} \bar{G} \\ \hline \underline{G} \end{array}\right] \quad \text{et} \quad \left[\begin{array}{c} * \\ \hline H_S \end{array}\right] = \underbrace{\left[\begin{array}{c|c} I_k & \\ \hline -A_{1,2}^T(A_{1,1}^{-1})^T & I_{n-k} \end{array}\right]}_{F^T} \left[\begin{array}{c} \bar{H} \\ \hline \underline{H} \end{array}\right].$$

  D'où $G_S = \underline{G} - A_{2,1}A_{1,1}^{-1}\bar{G} = \underline{G} + A_{2,1}Y_{1,1}$ et $H_S = \underline{H} - A_{1,2}^T(A_{1,1}^{-1})^T\bar{H} = \underline{H} - A_{1,2}^T Z_{1,1}$, ce qui montre le premier point. On peut alors en déduire par hypothèse de récurrence que le deuxième appel récursif nous donne $Y_S = -S^{-1}G_S$ et $Z_S = (S^{-1})^T H_S$. Il nous reste alors à vérifier que Y et Z vérifient bien les équations voulues. En effet :

$$AY = \left[\begin{array}{cc} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{array}\right] \left[\begin{array}{c} Y_{1,1} - A_{1,1}^{-1}A_{1,2}Y_S \\ \hline Y_S \end{array}\right] = \left[\begin{array}{c} A_{1,1}Y_{1,1} - A_{1,2}Y_S + A_{1,2}Y_S \\ \hline A_{2,1}Y_{1,1} - A_{2,1}A_{1,1}^{-1}A_{1,2}Y_S + A_{2,2}Y_S \end{array}\right]$$

Algorithme 3 : Calcul de générateurs de l'inverse d'une matrice Cauchy-like $n \times n$ (supposée fortement régulière)

Entrée : Générateurs $(G, H) \in (\mathbb{K}^{n\times\alpha})^2$ pour A et vecteurs $\mathbf{x}, \mathbf{y}$
Sortie : $Y = -A^{-1}G$ et $Z = H(A^{-1})^T$

début
1 **si** *n=1* **alors**
2 $d \leftarrow \frac{GH^T}{x_1 - y_1}$
3 $Y \leftarrow -\frac{1}{d}G$
4 $Z \leftarrow \frac{1}{d}H$
5 **sinon**
// appel récursif sur le quart supérieur gauche
6 $Y_{1,1}, Z_{1,1} \leftarrow$ résultat de l'appel récursif sur $(\bar{G}, \bar{H}, \bar{x}, \bar{y})$
// calcul des générateurs pour le complément de Schur
7 $G_S \leftarrow \underline{G} + A_{2,1}Y_{1,1}$
8 $H_S \leftarrow \underline{H} - A_{1,2}^T Z_{1,1}$
// appel récursif sur le complément de Schur
9 $Y_S, Z_S \leftarrow$ résultat de l'appel récursif sur $(G_S, H_S, \underline{x}, \underline{y})$
// construction des générateurs de l'inverse
10 $Y \leftarrow \begin{bmatrix} Y_{1,1} - A_{1,1}^{-1}A_{1,2}Y_S \\ \hline Y_S \end{bmatrix}$
11 $Z \leftarrow \begin{bmatrix} Z_{1,1} - A_{1,1}^{-T}A_{2,1}^T Z_S \\ \hline Z_S \end{bmatrix}$
12 Retourner (Y, Z)
fin

$$= \begin{bmatrix} A_{1,1}Y_{1,1} \\ A_{2,1}Y_{1,1} + SY_S \end{bmatrix} = \begin{bmatrix} A_{1,1}Y_{1,1} \\ A_{2,1}Y_{1,1} - G_S \end{bmatrix} = - \begin{bmatrix} \bar{G} \\ \underline{G} \end{bmatrix}$$

$$A^T Z = \begin{bmatrix} A_{1,1}^T & A_{2,1}^T \\ A_{1,2}^T & A_{2,2}^T \end{bmatrix} \begin{bmatrix} Z_{1,1} - (A_{1,1}^{-1})^T A_{2,1}^T Z_S \\ Z_S \end{bmatrix} = \begin{bmatrix} A_{1,1}^T Z_{1,1} - A_{2,1}^T Z_S + A_{2,1}Z_S \\ A_{1,2}^T Z_{1,1} - A_{1,2}^T (A_{1,1}^{-1})^T A_{2,1}^T Z_S + A_{2,2}^T Z_S \end{bmatrix}$$

$$= \begin{bmatrix} A_{1,1}^T Z_{1,1} \\ A_{1,2}^T Z_{1,1} + S^T Z_S \end{bmatrix} = \begin{bmatrix} A_{1,1}^T Z_{1,1} \\ A_{1,2}^T Z_{1,1} + H_S \end{bmatrix} = \begin{bmatrix} \bar{H} \\ \underline{H} \end{bmatrix}$$

□

Remarques :

1. Comme dans le théorème 2, la seule condition sur M et N dont nous avons besoin pour assurer la correction de l'algorithme est que M et N^T soient triangulaires inférieures.
2. L'intuition derrière les formules pour Y et Z peut être vue en remarquant que $Y = F \begin{bmatrix} Y_{1,1} \\ \hline Y_S \end{bmatrix}$ et $Z^T = \begin{bmatrix} Z_{1,1}^T \\ \hline Z_S^T \end{bmatrix} E$ d'une part, et en se rappelant que $EAF = \left[\begin{array}{c|c} A_{1,1} & \\ \hline & S \end{array}\right]$ d'autre part. En effet, on a alors $A^{-1} = F \left[\begin{array}{c|c} A_{1,1}^{-1} & \\ \hline & S^{-1} \end{array}\right] E$ et on sait générer $A_{1,1}^{-1}$ et S^{-1}. On se retrouve donc dans la situation inverse de celle du théorème 2.

Pour ce qui est de la complexité, il s'agit avant tout de voir comment on va effectuer les calculs du type AV où A est une matrice Cauchy-like $n \times n$ donnée par ses générateurs, et V est une matrice $n \times \alpha$ (un paquet de α vecteurs de taille n). Pour cela, nous avons la propriété suivante :

Propriété 6 *Si $A \in \mathbb{K}^{n\times n}$ est une matrice Cauchy-like donnée par des générateurs $G, H \in \mathbb{K}^{n\times\alpha}$ pour l'opérateur de déplacement $\nabla_{D(\mathbf{x}),D(\mathbf{y})}$, et si v est un vecteur de taille n, alors on sait calculer le produit Av en $O(\alpha M(n)\log(n))$.*

Pour montrer ce résultat, il suffit de voir que l'on peut faire un produit (matrice Cauchy) $\times$ vecteur en $O(M(n)\log(n))$ grâce à de l'interpolation/évaluation polynomiale (voir [8] par exemple) et d'utiliser la formule de reconstruction (2).

Dans notre cas, on peut se contenter d'appliquer α fois la méthode précédente pour faire les produits de matrices intervenant dans l'algorithme. Ainsi, le coup d'un tel produit sera en $O(\alpha^2 M(n)\log(n))$, ce qui conduit à l'équation de récurrence pour le coût total de l'algorithme suivante :

$$C(1) = O(\alpha) \text{ et, si } n > 1,\ C(n) \leq C(\lfloor \frac{n}{2} \rfloor + C(\lceil \frac{n}{2} \rceil) + O(\alpha M(n)\log(n))$$

On en déduit que $C(n) = O(\alpha^2 M(n)\log^2(n)) \in O^{\sim}(\alpha^2 n)$.

6 Conclusion

En résumé, nous avons vu différents algorithmes sur les matrices structurées. L'algorithme 1 (GKO) permet d'obtenir en $O(\alpha n^2)$ une factorisation PLU pour les matrices inversibles et structurées pour l'opérateur $\nabla_{M,N}$ avec M et N^T triangulaires inférieurs. Si on se restreint à M et N diagonales, il est alors possible de traiter le cas non inversible grâce à mon algorithme de factoristaion LSP (algorithme 2).

Le problème avec les deux algorithmes réside dans la taille de la sortie ($O(n^2)$ coefficients dans le cas inversible) qui met fin à tout espoir d'accélération de l'algorithme pour obtenir un temps quasi-linéaire en n. Toutefois, si on se concentre sur le problème de l'inversion au lieu de celui de la factorisation, on peut alors réduire le coût mémoire à $O(\alpha n)$ (algorithme 4), ce qui permet de considérer une version rapide (algorithme 3).

L'algorithme 3 auquel nous sommes finalement arrivé permet donc d'inverser certaines matrices denses structurées pour un coût en $O^{\sim}(\alpha^2 n)$. Bien que cet algorithme soit déjà cité par Cardinal dans [4], notre approche est intéressante car elle conduit à une preuve simple de la correction de l'algorithme, qui s'applique non seulement au cas Cauchy-like comme dans l'article de Cardinal, mais aussi à d'autres cas dont notamment les matrices Vandermonde-like utilisées pour les codes correcteurs d'erreurs.

Cet algorithme est intéressant aussi parce qu'il relie deux pans de la littérature : les articles basés sur GKO ([7],[14]) et les articles avec une approche divisé pour régner ([2],[19],[15],[22],[3]). Il soulève aussi quelques interrogations. La question de l'introduction du pivotage dans les algorithmes quasi-linéaires en n semble difficile. L'algorithme de calcul de σ-bases semble aussi être une alternative intéressante mais son étude actuelle conduit seulement à des coûts en $O^{\sim}(\alpha^{\omega} n)$ au lieu de $O^{\sim}(\alpha^{\omega-1} n)$.

Références

[1] B. Beckermann and G. Labahn. A uniform approach for the fast computation of matrix-type Padé approximants. *SIAM J. Matrix Anal. Appl.*, 15(3) : 804–823, 1994.

[2] R. R. Bitmead and B. D. O. Anderson. Asymptotically fast solution of Toeplitz and related systems of linear equations. *LAA*, 34 : 103–116, 1980.

[3] A. Bostan, C.P. Jeannerod, and É. Schost. Solving Toeplitz- and Vandermonde-like linear systems with large displacement rank. In *ISSAC '07*, pages 33–40, NY, USA, 2007. ACM.

[4] J. P. Cardinal. A divide and conquer method to solve Cauchy-like systems. *Comptes Rendus de l'Académie des Sciences - Series I - Mathematics*, 328(11) : 1089–1093, 1999.

[5] J. von zur Gathen and J. Gerhard. *Modern computer algebra, second edition.* Cambridge University Press, New York, NY, USA, 2003.

[6] P. Giorgi, C.P. Jeannerod, and G. Villard. On the complexity of polynomial matrix computations. In *ISSAC '03, Philadelphia, Pennsylvania, USA*, pages 135–142. ACM, 2003.

[7] I. Gohberg, T. Kailath, and V. Olshevsky. Fast Gaussian elimination with partial pivoting for matrices with displacement structure. *Math. of Computation*, 64(212) : 1557–1576, 1995.

[8] I. Gohberg and V. Olshevsky. Complexity of multiplication with vectors for structured matrices. *Linear Algebra Appl.*, 202 : 163–192, 1994.

[9] I. Gohberg and V. Olshevsky. The Fast Generalized Parker-Traub Algorithm for Inversion of Vandermonde and Related Matrices. 1997.

[10] G. Heinig and A. Tewodros. On the inverses of Hankel and Toeplitz mosaic matrices. In *Seminar on Analysis of Operator Equation and Numerical Analysis*, pages 53–65, 1988.

[11] O. H. Ibarra, S.h Moran, and R. Hui. A Generalization of the Fast LUP Matrix Decomposition Algorithm and Applications. *J. Algorithms*, 3(1) : 45–56, 1982.

[12] C.P. Jeannerod. LSP matrix decomposition revisited, 2006. LIP research report RR2006-28.

[13] T. Kailath, S. Y. Kung, and M. Morf. Displacement ranks of matrices and linear equations. *J. Math. Anal. Appl.*, 68 : 395–407, 1979.

[14] T. Kailath and A. H. Sayed, editors. *Fast reliable algorithms for matrices with structure.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1999.

[15] E. Kaltofen. Asymptotically fast solution of Toeplitz-like singular linear systems. In *ISSAC '94*, pages 297–304. ACM, 1994.

[16] G. Labahn, B. Bechermann, and S. Cabay. Inversion of Mosaic Hankel Matrices via Matrix Polynomial Systems. *Linear Algebra and its Applications*, (221) : 253–280, 1995.

[17] G. Labahn, D. K. Choi, and S. Cabay. The Inverses of Block Hankel and Block Toeplitz Matrices. *SIAM J. of Computing*, (19) : 98–123, 1990.

[18] N. Levinson. The Wiener RMS error criterion in filter design and prediction. *J. Math. Phys.*, 25 : 261–278, 1947.

[19] M. Morf. Doubling algorithms for Toeplitz and related equations. *IEEE Conference on Acoustics, Speech, and Signal Processing*, pages 954–959, 1980.

[20] V. Olshevsky and M. Amin Shokrollahi. A displacement approach to decoding algebraic codes. In *Contemporary mathematics : theory and applications*, pages 265–292, Boston, MA, USA, 2003. AMS.

[21] V. Pan. *Structured matrices and polynomials : unified superfast algorithms.* Springer-Verlag New York, Inc., NY, USA, 2001.

[22] V. Pan and A. Zheng. Superfast algorithms for Cauchy-like matrix computations and extensions. *Linear Algebra and its Applications*, 310 : 83–108, 2001.

Annexe

Algorithme 4 : Calcul de générateurs de l'inverse d'une matrice Cauchy-like $n \times n$ (supposée inversible)

Entrée : Générateurs $G, H \in \mathbb{K}^{n \times \alpha}$ pour A et vecteurs $\mathbf{x}, \mathbf{y}$
Sortie : $Y = -A^{-1}G$ et $Z = (A^{-1})^T H$

début

$P \leftarrow I_n$

$Y \leftarrow$ matrice nulle de taille $n \times \alpha$

$Z \leftarrow$ matrice nulle de taille $n \times \alpha$

$\ell \leftarrow$ vecteur nul de taille n

$u \leftarrow$ vecteur nul de taille n

pour k *allant de* 1 *à* n **faire**

$$\ell \leftarrow \begin{bmatrix} 0 & \dots & 0 & \frac{g_k h_k^T}{x_k - y_k} & \dots & \frac{g_j h_k^T}{x_j - y_k} & \dots & \frac{g_n h_k^T}{x_n - y_k} \end{bmatrix}^T$$

$q \leftarrow \min\{j, \ell_j \neq 0\}$

si $q > k$ **alors**

// On doit pivoter

$g_k, g_q \leftarrow g_q, g_k$

$x_k, x_q \leftarrow x_q, x_k$

$\ell_k, \ell_q \leftarrow \ell_q, \ell_k$

$p_k, p_q \leftarrow p_q, p_k$

$d \leftarrow \ell_k$

$$u \leftarrow \begin{bmatrix} 0 & \dots & 0 & \frac{g_k h_k^T}{x_k - y_k} & \dots & \frac{g_k h_j^T}{x_k - y_j} & \dots & \frac{g_k h_n^T}{x_k - y_n} \end{bmatrix}^T$$

// mise à jour des générateurs pour l'inverse de A

$$Y_{1..k-1,*} \leftarrow Y_{1..k-1,*} - \frac{1}{d} \begin{bmatrix} y_1 - y_k & & \\ & \ddots & \\ & & y_{k-1} - y_k \end{bmatrix}^{-1} Y_{1..k-1,*}\ h_k^T\ g_k$$

$$Z_{1..k-1,*} \leftarrow Z_{1..k-1,*} + \frac{1}{d} \begin{bmatrix} x_1 - x_k & & \\ & \ddots & \\ & & x_{k-1} - x_k \end{bmatrix}^{-1} Z_{1..k-1,*}\ g_k^T\ h_k$$

$$y_k \leftarrow -\frac{1}{d} g_k$$

$$z_k \leftarrow \frac{1}{d} h_k$$

// mise à jour des générateurs pour A

$$G_{k+1..n,*} \leftarrow G_{k+1..n,*} - \frac{1}{d}\ \ell[k+1..n]\ g_k$$

$$H_{k+1..n,*} \leftarrow H_{k+1..n,*} - \frac{1}{d}\ u[k+1..n]\ h_k$$

Retourner $(Y, P^T Z)$

fin