Décomposition LU

Minh-Phuong Tran - 51091600 - Décembre 2018

1 Introduction

L'élimination de Gauss-Jordan (Gaussian elimination) a pour objectif de transformer un système linéaire A, en une matrice triangulaire supérieure U, en lui appliquant une suite de transformations linéaires à gauche. Ces transformations linéaires peuvent être regroupées en une matrice triangulaire inférieure L^{-1} où L est également triangulaire inférieure (ces 2 propriétés peuvent être démontrées avec les 2 *strokes of luck* cfr Lecture 20 du livre Numerical Linear Algebra). Cela constitue donc la décomposition A = LU. Il est à noter que cette décomposition n'est réalisable que sur des matrices inversibles et dont la diagonale ne comporte aucun élément nul (dans le cas contraire, la décomposition LU nécessiterait le pivotage). Cet article analyse l'efficacité d'une décomposition LU sur une matrice pleine, creuse avec le format CSR (Compressed Sparse Row) avec et sans numérotation suivant l'algorithme RCMK (Reverse Cuthill-McKee).

2 LU matrice pleine

Pour commencer, la décomposition LU sur matrice pleine a été réalisée en considérant que celle-ci est hermitienne. NB : Une décomposition Cholesky aurait été possible étant donné que la matrice est hermitienne. Ainsi $A = LL^*$.

Il a été montré au cours d'Analyse Numérique que le nombre d'opérations pour cette décomposition est $\frac{2}{3}m^3$ avec m la taille de la matrice. En effet, la décomposition effectue pour chaque ligne k, n combinaisons linéaires, étant le nombre de lignes après k et vaut au plus m. La combinaison linéaire dépendant de la taille de la ligne k, nous avons donc au plus m*m*m = m^3.

Au cours des expériences menées sur la décomposition LU pleine, un premier graphique avec le temps d'exécution de la résolition du système Ax = b par décomposition LU pleine a pu être réalisée. Le calcul de la complexité de cet algorithme a été ensuite effectué avec $log_4(\frac{O(4n)}{O(n)})$. Le deuxième graphique nous montre pourtant une complexité

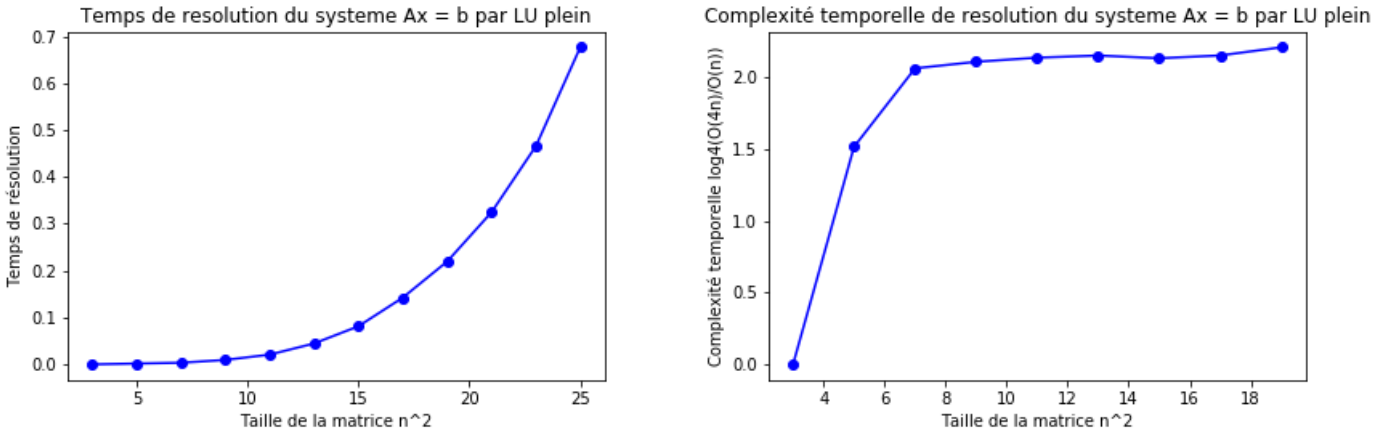

Figure 1: Décomposition LU

asymptotique aux environs de 2.5 ce qui entre en conflit avec ce qui a été annoncé précédemment. Une des hyppothèses qui pourraient être émise serait que la valeur asymptotique n'a pas encore été atteinte et qu'il aurait fallu calculer cette décomposition sur de plus grandes matrices encore afin de vérifier si cette valeur s'accroît jusqu'à atteindre 3.

3 LU creux sous format CSR

Il peut être intéressant de remarquer que de nombreux problèmes physiques se traduisent par un système linéaire creux. Dans ce cas-là, une décomposition LU pour matrice pleine n'est plus appropriée.

Afin de travailler avec ces matrices creuses, le format CSR qui ne retient que les éléments non nuls a été implémenté. Voici ses performances : Afin d'implémenter cette méthode, des fonctions prédéfinies de la librairie

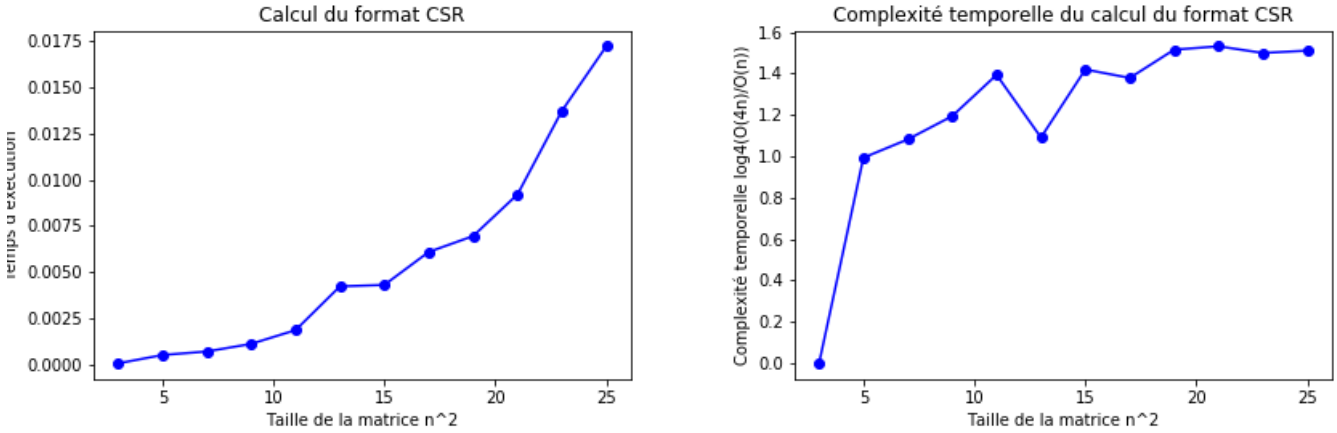

Figure 2: Format CSR

numpy ont été utilisées telles que numpy.nonzero, sa complexité peut être bornée par m^2, avec m la taille de la matrice A. En effet, il lui faut au maximum parcourir tous les éléments de la matrice de A. Ici nous trouvons bien une complexité inférieure à n^2 sur la Figure 2.

Ensuite, afin de réaliser la décomposition LU creuse, le travail avec une matrice bande a été préféré. En effet, la fonction de calcul du LU creux prend en argument la matrice A sous format CSR c'est à dire sA, iA et jA. A partir de ces éléments, il a fallu calculer la largeur de bande supérieure et inférieure pour recomposer les sA, iA et jA de la matrice bande. Travailler avec cette nouvelle matrice comporte plusieurs avantages comme le fait que la largeur de bande des matrices L et U ne seront pas plus grandes que celle de A ainsi que d'autres facilités d'implémentations. Nous obtenons donc en temps d'exécution :

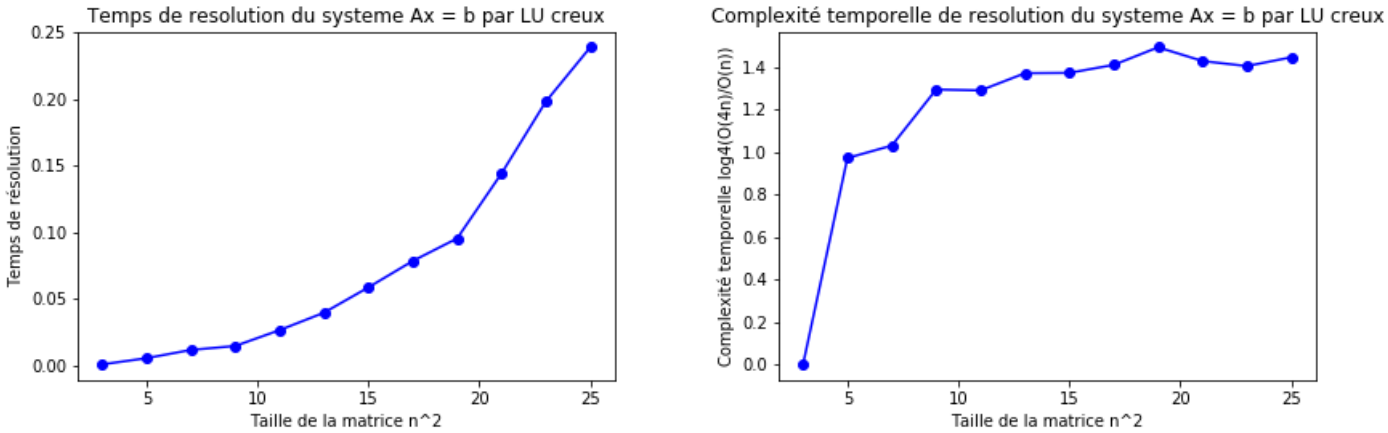

Figure 3: Décomposition LU creux- format CSR

En comparaison avec la décomposition LU pleine, nous pouvons observer un temps d'exécution nettement moins grand ainsi qu'une meilleure complexité temporelle. Cela est dû au fait que la décomposition creuse ne doit plus parcourir tous les éléments de la matrice A mais seulement ses éléments non nuls. NB : Les matrices générées pour ces séries de tests proviennent de la fonction creatDF fournie avec le script de tests sur Moodle.

D'autres tests ont été également réalisés sur le fichier ccore.py, mais avec problème, la décomposition LU pleine reste meilleure que creuse ce que nous ne savons malheureusement pas expliquer...

4 LU creux avec RCMK

Enfin, lorsque l'on travaille avec des matrices creuses, il peut être intéressant de renuméroter les colonnes et les lignes de la matrice A afin d'obtenir une matrice avec laquelle la densité de la décomposition LU est nettement moindre. Cette numérotation est réalisable avec l'algorithme Reverse Cuthill-McKee qui est fort semblable à l'algorithme BFS. En effet, le graphe ci-dessous décrit le rapport entre la densité d'une décomposition LU creuse avec RCMK et sans en fonction de la taille de la matrice.

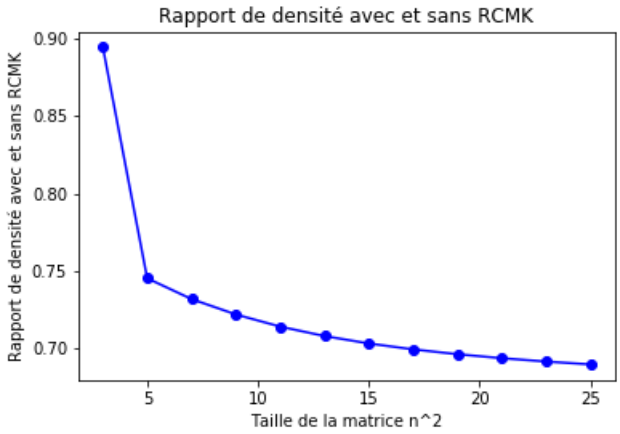

Figure 4: Rapport de densité de la décomposition LU avec RCMK

Ce qui est observable est que l'algorithme RCMK permet de réduire la densité de la décomposition LU jusqu'à 70 % de sa densité sans renumérotation ce qui représente une optimisation importante en terme de mémoire.

A partir de cette renumérotation, il a donc pu être possible de calculer la décomposition LU et résoudre le système Ax = b. Ses performances sont présentées sur les graphes suivants.

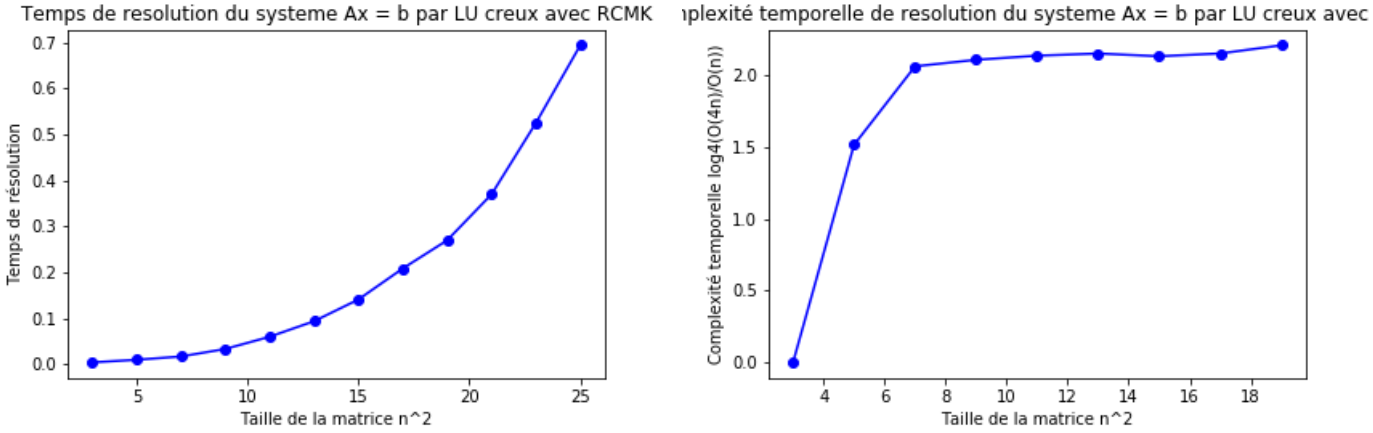

Figure 5: Décomposition LU creux- format CSR avec RCMK

Dans ce dernier cas, le temps de résolution ainsi que la complexité temporelle est inférieure à celui avec la décomposition LU pleine mais reste supérieure à celle sans RCMK. Cela pourrait être expliquée par une implémentation pas assez optimisée qui aurait donc pu être améliorée.

Cependant lors de tests avec le fichier ccore.py, le temps d'exécution de résolution avec RCMK était nettement meilleur que sans.

5 Conclusion

Au terme de cette analyse, il est important de retenir qu'en présence de problèmes creux, il est préférable de travailler avec un format CSR qui en plus d'optimiser l'utilisation de la mémoire optimise également le temps de calcul. L'utilisation de RCMK a fait ses preuves en matière de densité de la décomposition LU. Néanmoins il reste nécessaire d'optimiser son implémentation actuelle afin que son temps d'exécution devienne moindre.