Compte rendu du TP 2-3 Calcul Numérique :

« SCILAB »

Nom : DENDANI

Prénom : Lamia

N°etudiant :22109916

Dans ces TPs, nous avons fait un survol des fonctionnalités de Scilab afin de nous familiariser avec cet environnement. L'objectif est de présenter les compétences nécessaires pour démarrer avec Scilab.et les éléments de base du langage tel que la déclaration et manipulation de variables, matrices, boucles …

Dans le TP3 nous avons écrit nos premiers algorithmes tel que : multiplication de matrices ,l'élimination de Gauss la factorisation LU et pivot partiel .

Nous avons également mesurée et analysé l'erreur (erreurs avant/arrière) que peuvent produire ces algorithmes ainsi que mesuré le temps d'exécution de certain algorithme.

Les différents graphes ont été tracés à l'aide de gnuplot .

TP2 :

Exercice 6 : Prise en main de Scilab :

Dans cet exercice nous avons appris a utilisé les fonctionnalités de bases de Scilab tel que déclarer des vecteurs lignes et colonnes, générer des matrices. Nous avons ensuite effectué les opérations de bases sur ces objets. Nous avons aussi appris quelque fonction de base tel que size() qui renvoie la taille des dimension d'un objet et la fonction cond() qui permet de renvoyer le conditionnement de notre matrice/vecteur. Ce conditionnement permet de savoir à quel point notre matrice va générer de l'erreur et donc à quel point la solution peut être valide.

Vous trouverez le programme scilab sur mon dépôt github sous le nom : exo6.sce

Exercice 7 : Matrice random et problème « jouet » :

Dans cet exercice nous avons mesurer l'erreur avant avec la formule : $err = \frac{||x-\hat{x}||}{||x||}$ ainsi que l'erreur arrière : $relres = \frac{||b-A\hat{x}||}{||A||\,||\hat{x}||}$ sur le système : $Ax = b$. Nous avons obtenu les résultats suivant :

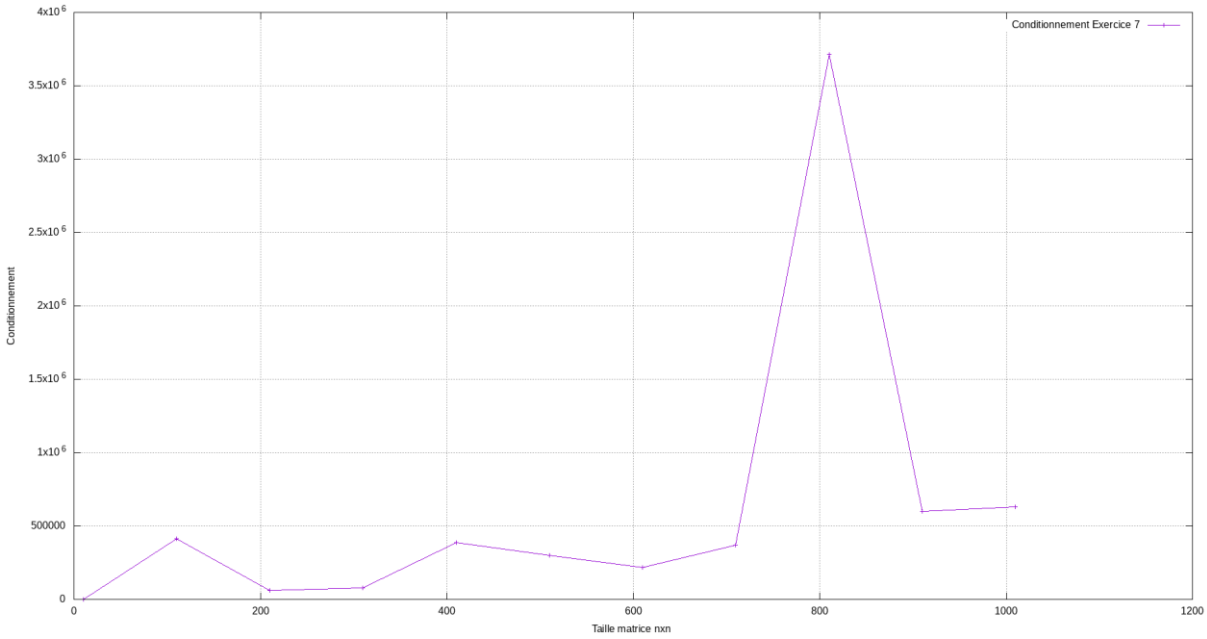

Figure 1 : Conditionnement de la matrice de taille n*n de l'exercice 07

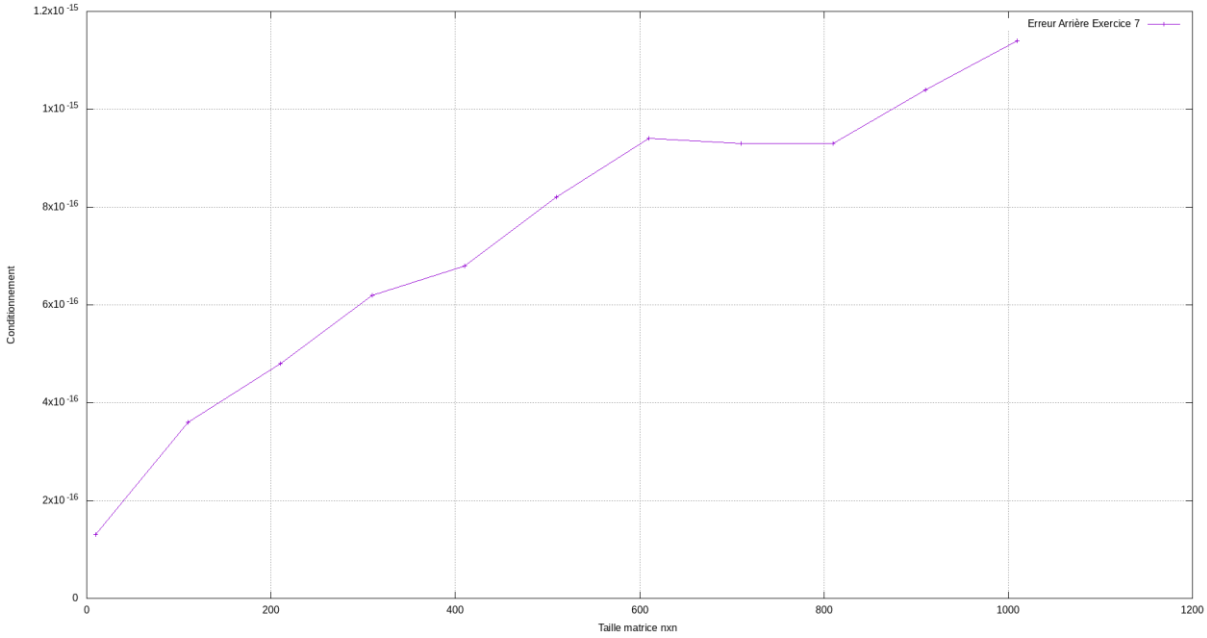

Figure 2 Erreur arrière de la matrice de taille n*n de l'exercice 07

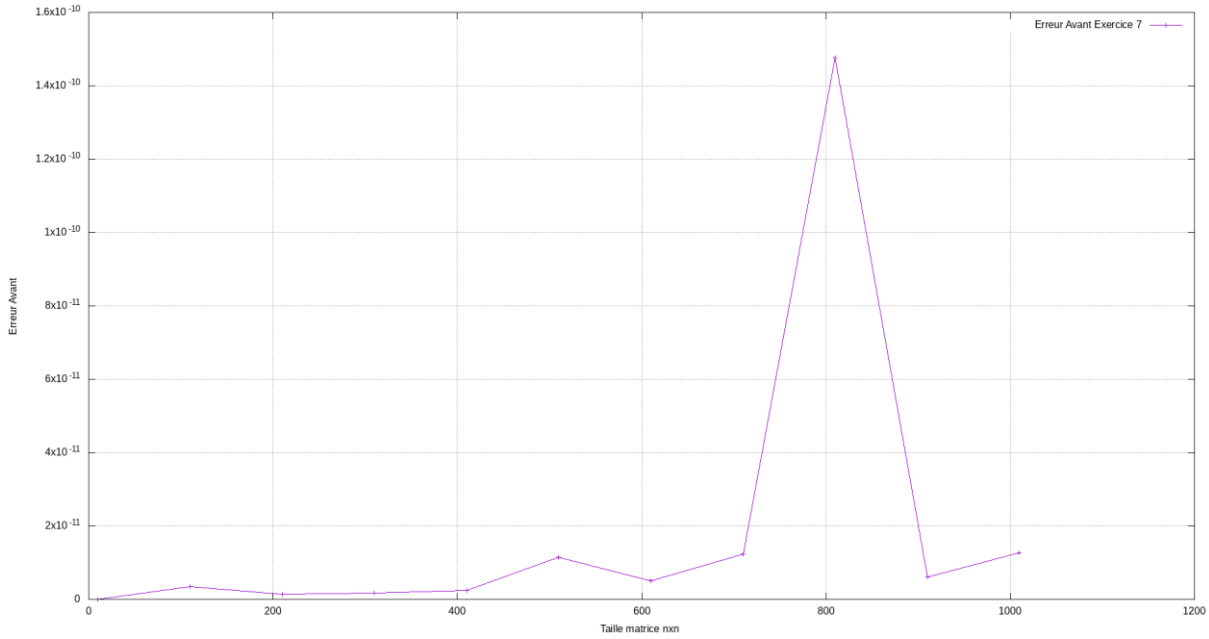

Figure 3 Erreur avant de la matrice de taille n*n de l'exercice 07

On remarque ici que l'erreur avant et le conditionnement sont liés , car conditionnement mesures la « capacité » de notre matrice à générer des erreurs de calculs , donc à quel point notre solution soit valide , autrement dit la mesure de l'erreur avant de la matrice .
Ce qui fait que le des graphes de conditionnement et l'erreur avant ont le même comportement .
Tandis que l'erreur arrière est croissante car elle dépend de la taille de la matrice. Cela est due au nombre d'éléments à générer aléatoirement, manipuler et les stocker .

La complexité de cet algorithme étant quadratique, le temps d'exécution augmente très vite par rapport à la taille des matrices. Par conséquent , l'algorithme n'a pas été exécuter sûr des matrices a N très grand.
Vous trouverez les programmes scilab de cet exercice sur mon dépôt github (lien dans la dernière page) sous le nom : exo7.sce

Exercice 8 : produit Matrice-Matrice :

Dans cet exercice nous avons écrit un algorithme de multiplication de deux matrices.
Dans le premier temps nous avons effectué le calcul avec 3 boucles « ijk »(matmat3b.sci), puis deux boucles (matmat2b.sci)et enfin avec une seul boucle(matmat1b.sci). Nous avons ensuite mesuré et comparer le temps d'exécution des différents algorithmes. Nous avons obtenu les résultats suivant :

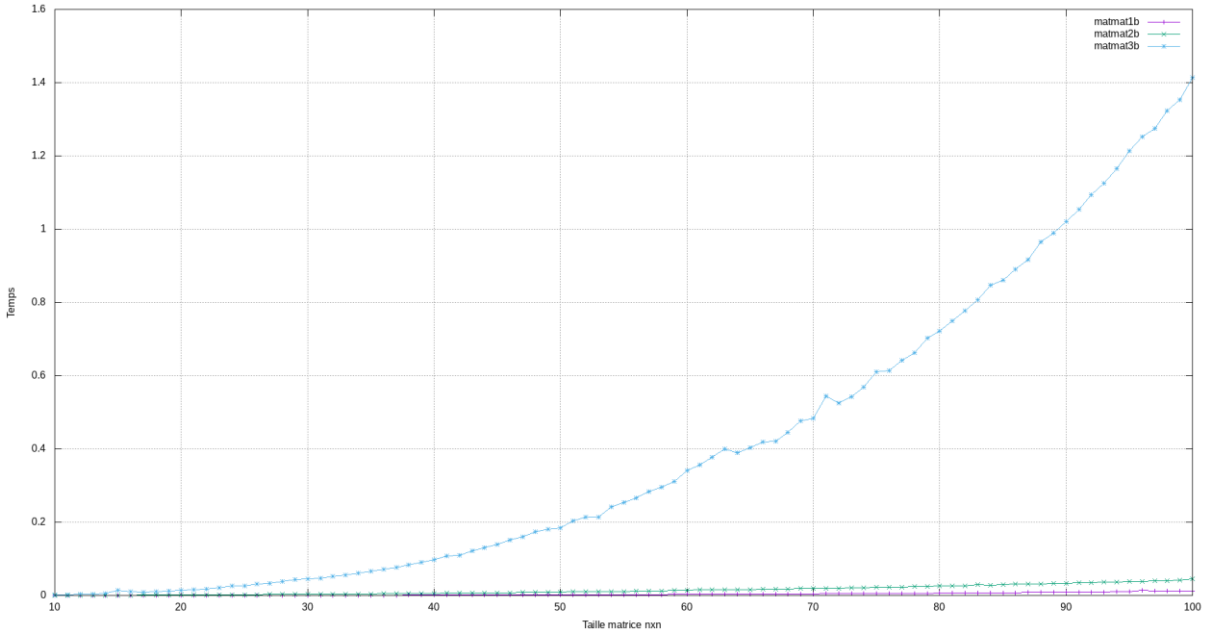

Figure 4 : temps d'exécution du programme MATRICE X MATRICE avec 1,2et 3 boucles.

Le temps d'exécution du programme dépend du nombre de boucles :l'algorithme qui a le moins de boucles est plus optimal et performant .
La complexité algorithmique de ces algorithmes est de O(m*n*p) avec m dimension 1 de la première matrice, p la dimension commune des deux matrices et n la deuxième dimension de la deuxième matrice.

TP3 :

Exercice 2 : Système triangulaire :

Dans cet exercice nous avons écrit des algorithmes de résolutions par remonter et descente qui s'applique respectivement sur des matrices triangulaires supérieures et inférieurs. Nous avons également mesurer l'erreur avant, l'erreur arrière ainsi que le conditionnement de ces algorithmes.

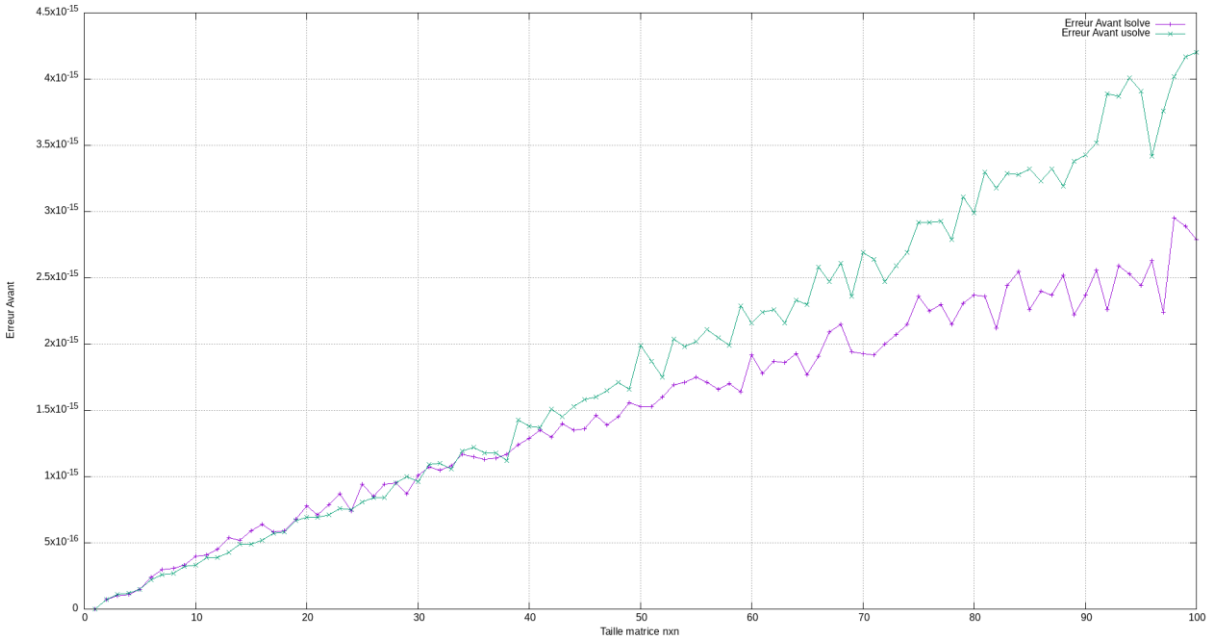

Figure 5: Erreur avant de isolve et usolve de l'exercice 02

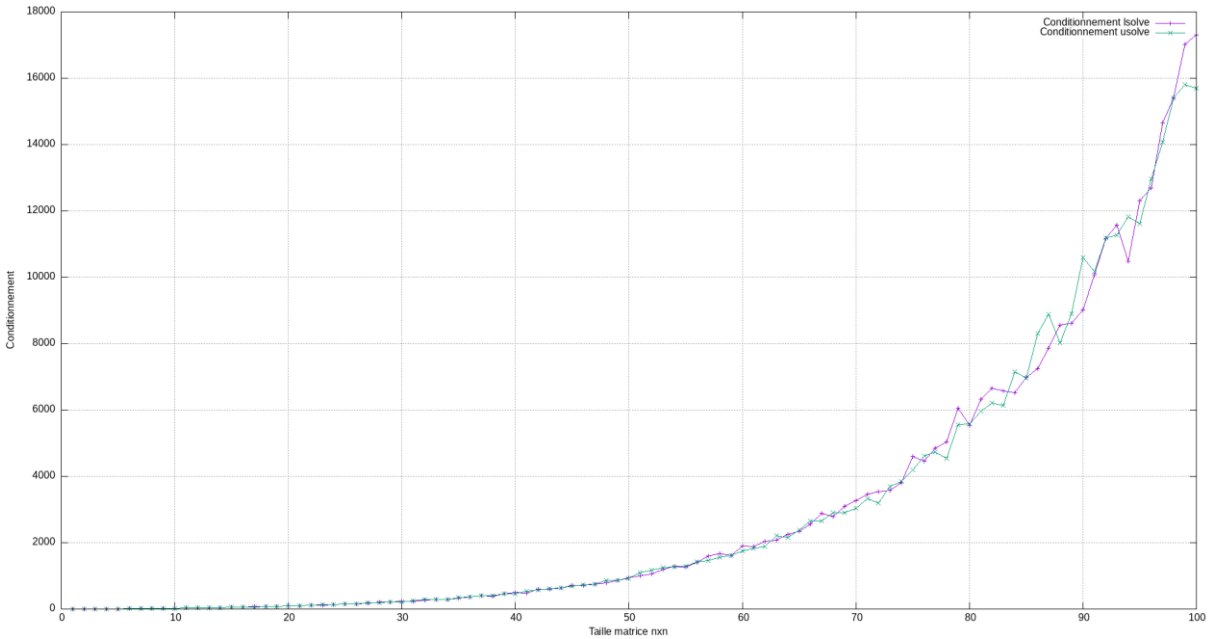

Figure 5: le conditionnement de isolve et usolve de l'exercice 02

L’erreur et le conditionnement sont croissants ,ils dépendent fortement de la taille de la matrice cela est due au nombre énorme de variable aléatoire générés avec la fonction rand().
On peut expliquer ceci car le conditionnement est associé à un problème pour mesurer de la difficulté de calcul numérique du problème. Un problème dont le conditionnement est faible autrement dit « problème *bien conditionné* », et un problème dont le conditionnement est élevé est dit « problème *mal conditionné* ».
Nous pouvons dire alors que notre problème Usolve et Isolve sont mal conditionné quans n >50 , autrement dit , si les valeurs de n sont plus grande que 50 notre problème est mal conditionné .

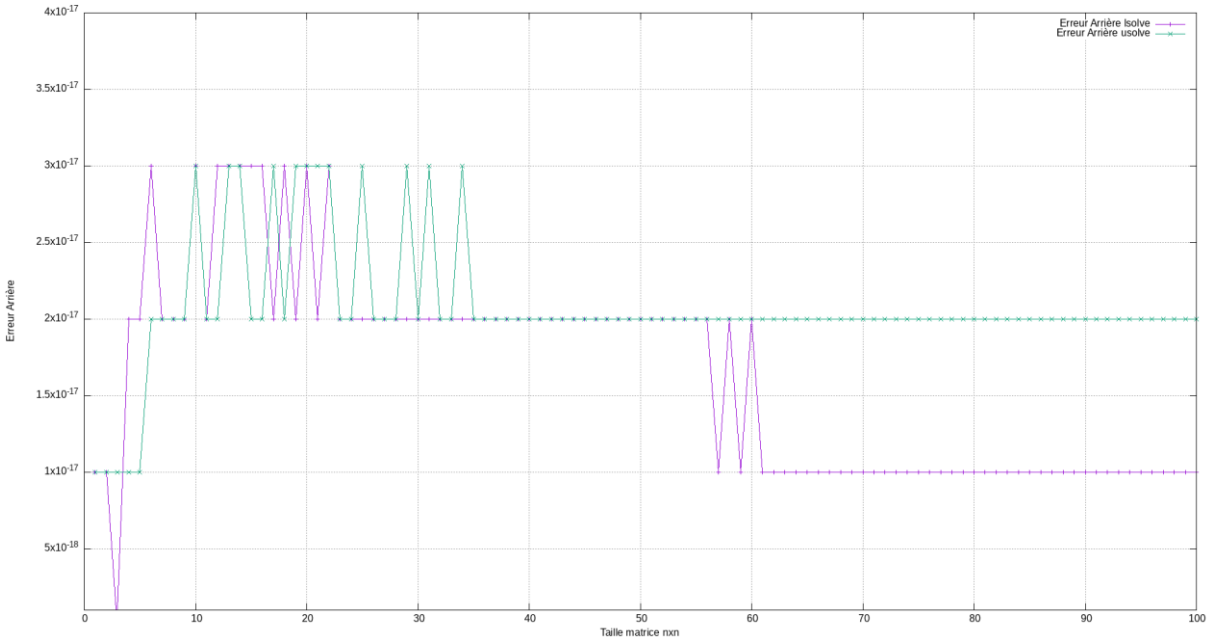

Figure 6: Erreur arrière de isolve et usolve de l’exercice 02

L’erreur arrière semble stable et reste faible $<10^{-17}$. On peut déduire que la différence entre la solution obtenue et la solution exacte des équations discrétisées reste bornée. Cette stabilité indique si l'erreur augmente ou non au cours du calcul. On peut dire qu’elle est conditionnellement stable .

Exercice 3 : Gauss

Dans cet exercice nous avons programmer l’algorithme de résolution d’un système linéaire par élimination de Gauss sans pivotage. Nous avons codé cet algorithme sous la forme « kij »

Nous avons mesuré les erreurs avant et arrière ainsi que le conditionnement en fonction de la taille de la matrice. Nous avons obtenu les résultats suivant :

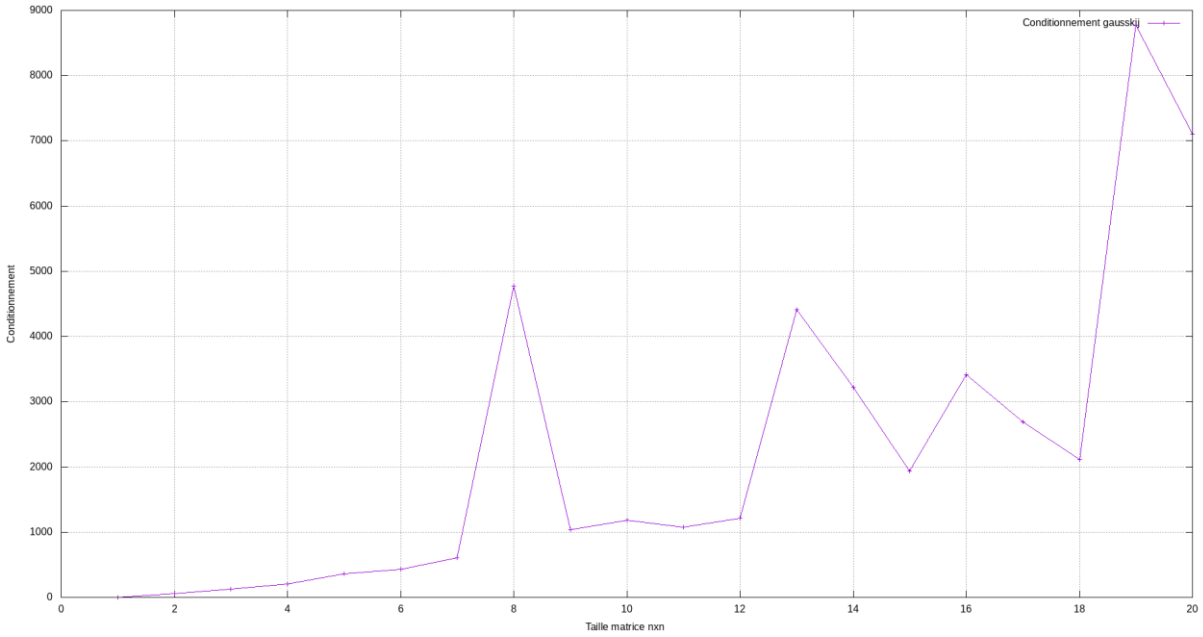

Figure 7: le conditionnement de gauss <kij> de l'exercice 03

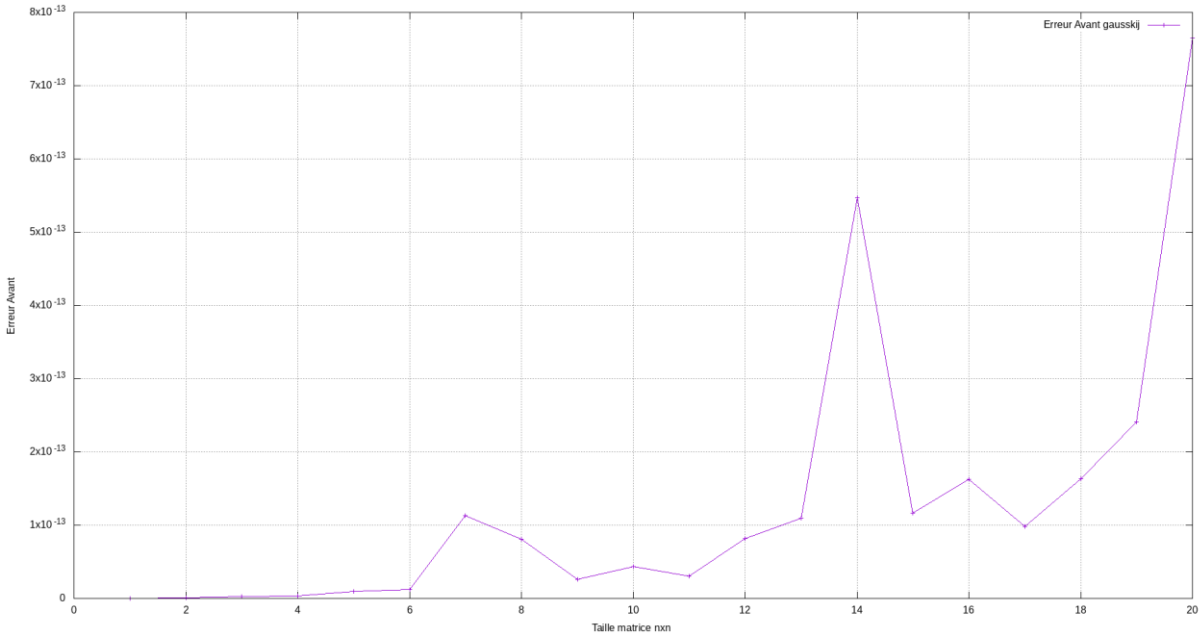

Figure 8: l'erreur avant de gauss <kij> de l'exercice 03

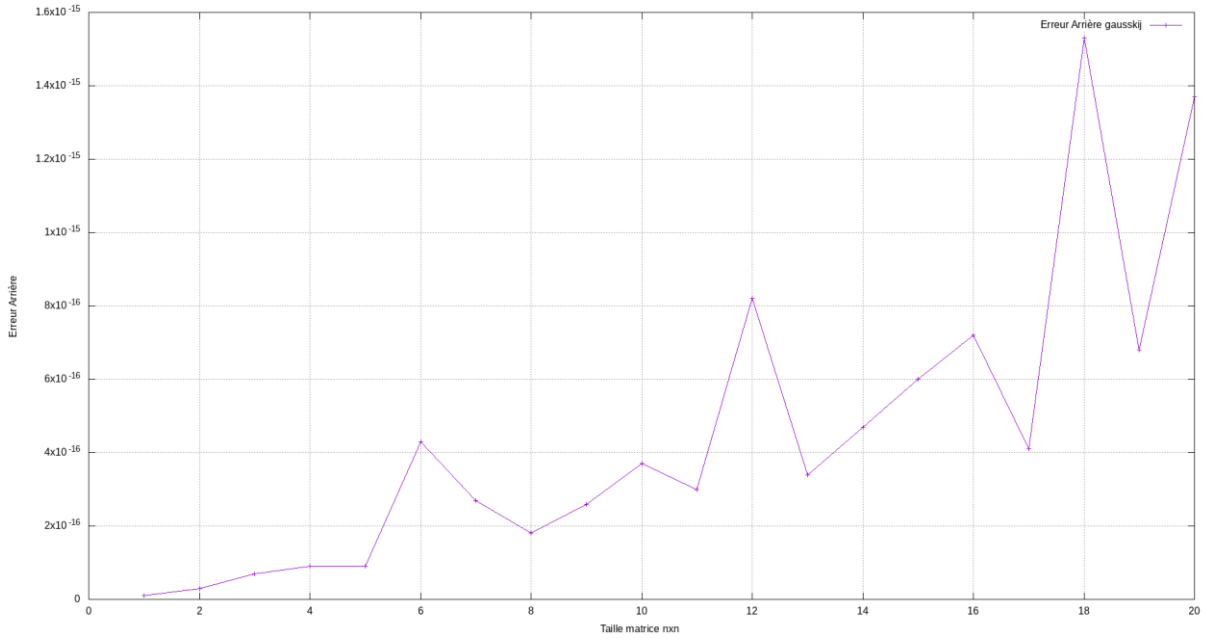

Figure 9: l'erreur arrière de gauss <kij> de l'exercice 03

Nous remarquons le même comportement de l'erreur avant et arrière ainsi que le conditionnement avec les exercices précédents.

Exercice 4 :

Dans cet exercice nous avons programmer un algorithme qui calcul la factorisation LU avec 1 , 2 et 3 boucle , ainsi qu'une version optimisé , nous avons également calculer le pivot partiel (vous trouverez les programmes sous les noms : ***mylu1b.sci, mylu2b.sci,mylu3b.sci ,lu_optim.sci ,pivot_partiel.sci*** ainsi que les test ***: test_lu.sci , test_lu3b.sci , test_pivot_partiel.sci***)

Pivot partiel :

Dans cette partie de l'exercice , nous avons programmer un algorithme du pivot partiel .
Nous avons calculer le temps déxecution de la fonction mylu[A].

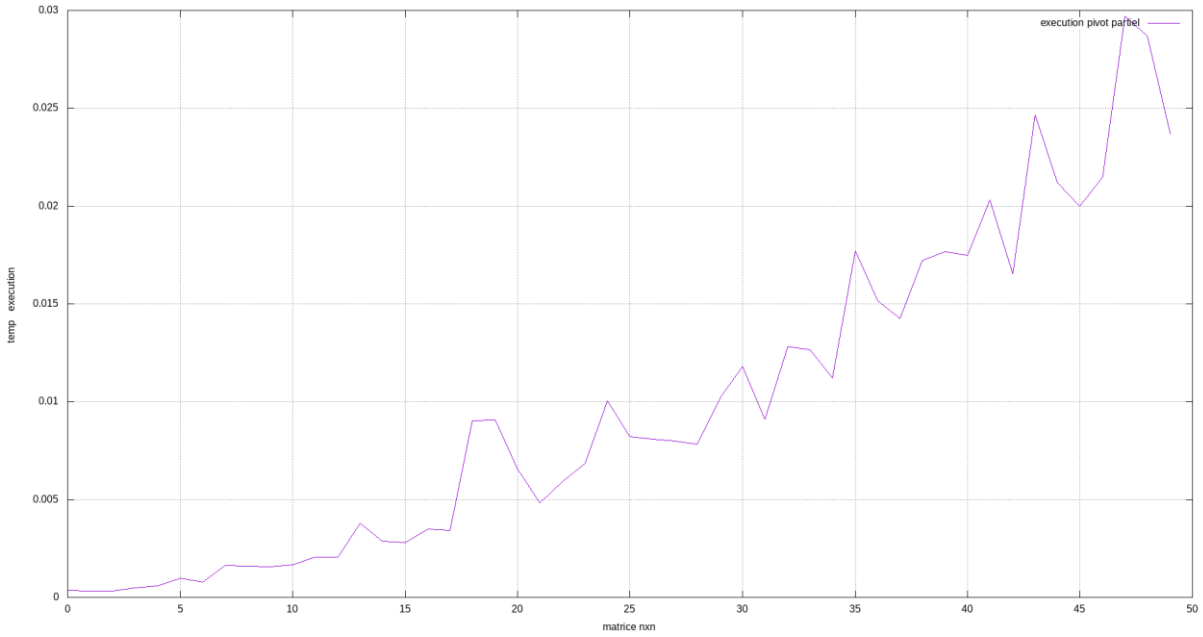

Figure 9 : temps d'exécution du Pivot partiel de l'exercice 03

Le temps d'exécution du programme est croissant , il dépend fortement de la taille de la matrice Cette méthode dite du pivot partiel à une complexité en **$O(n^2)$.**

Vous trouveretous les programmes du tp2 ainsi que le tp3 sur mon dépôt github :

Dépôt GITHUB : https://github.com/lamialami1997/SCILAB-TP-2-3