TP2-3 Calcul Numérique :

LAPLANCHE Alexis

Introduction :

Durant les séances de TD/TP nous avons appris à utiliser Scilab. Nous avons en effet codé divers algorithme tel que la multiplication de matrice, l'élimination de Gauss et la factorisation LU. Nous avons ensuite mesurée et analysé l'erreur que peuvent produire ces algorithmes ainsi que mesuré le temps d'exécution de certain algorithme. L'explication de comment l'on a procédé ainsi que l'analyse des résultats se trouverons dans ce rapport.

Les différents graphiques ont été tracés à l'aide de gnuplot : la fonction plot de scilab générait une image vide.

TP2 :

Dans ce TP nous avons surtout appris a maitrisé l'outil Scilab :

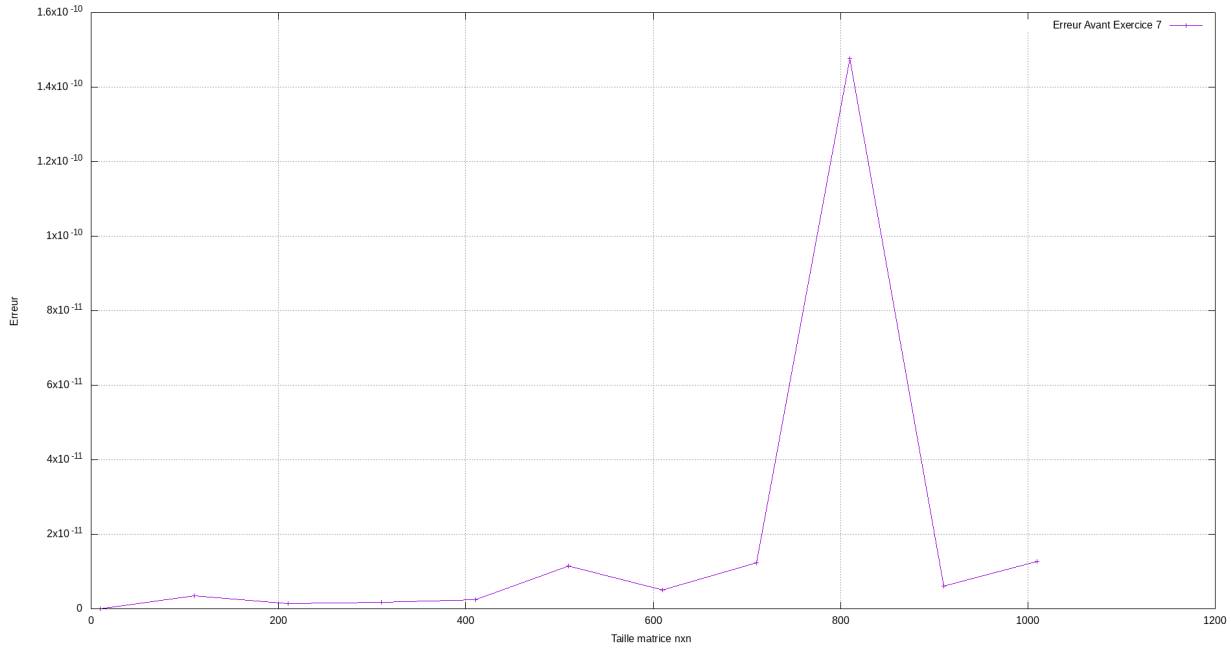
Exercice 6 : Prise en main de Scilab :

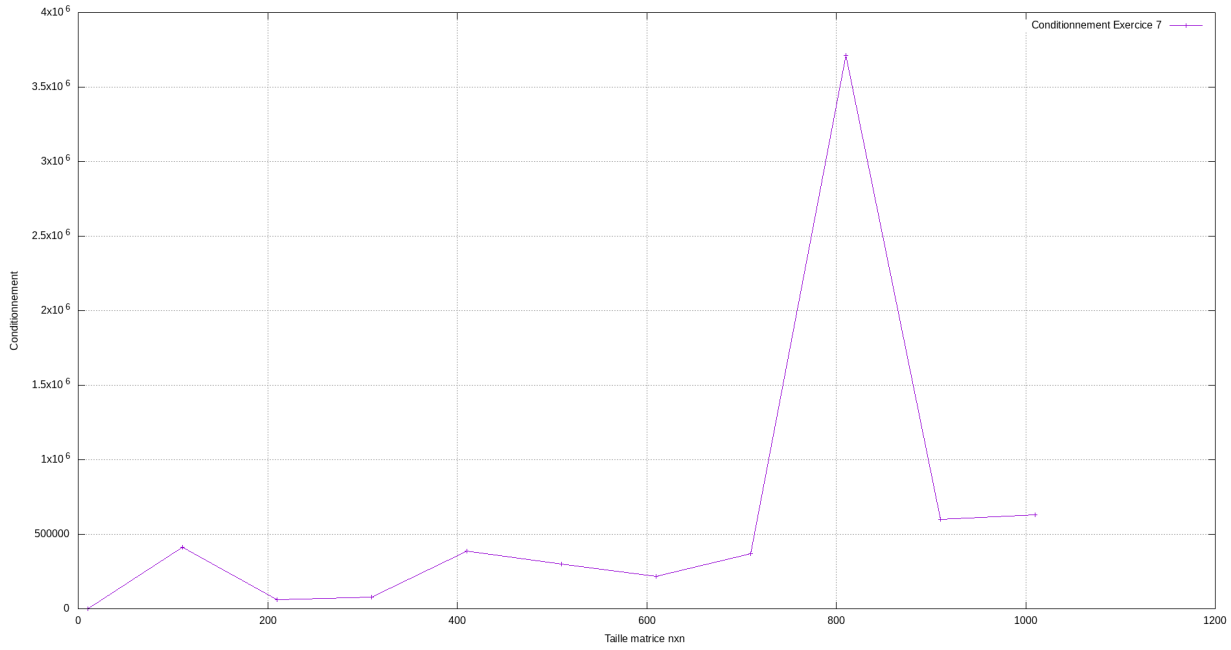
Dans cet exercice nous avons appris a utilisé les fonctionnalités de bases de Scilab tel que déclarer des vecteurs lignes et colonnes, générer des matrices. Nous avons ensuite effectué les opérations de bases sur ces objets. Nous avons aussi appris quelque fonction de base tel que size() qui renvoie 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.

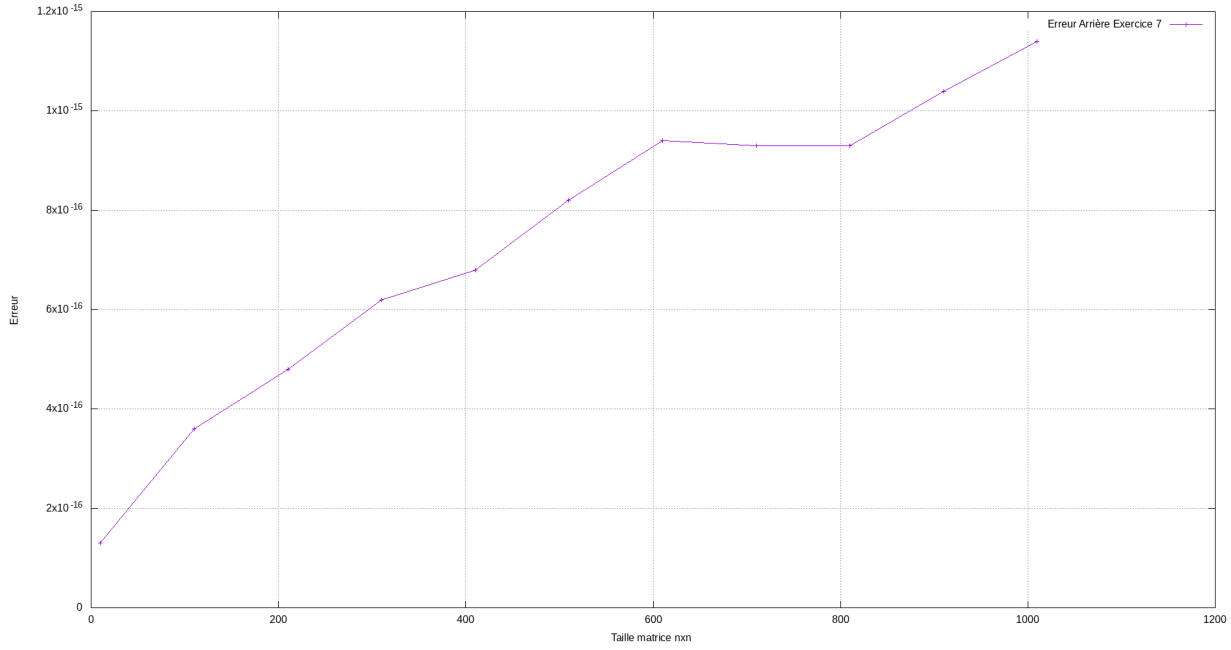
Le code source de cet exercice se trouve dans le dossier TP2-code sous le nom Exercice6.sce

Exercice 7 : Matrice random et problème « jouet » :

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


Erreur Avant Exercice 7
Erreur
Taille matrice nxn
1.6x10 -10
1.4x10 -10
1.2x10 -10
1x10 -10
8x10 -11
6x10 -11
4x10 -11
2x10 -11
0
0
200
400
600
800
1000
1200


Conditionnement Exercice 7
Conditionnement
Taille matrice nxn
4x10 6
3.5x10 6
3x10 6
2.5x10 6
2x10 6
1.5x10 6
1x10 6
500000
0
0
200
400
600
800
1000
1200


Erreur Arrière Exercice 7
Erreur
Taille matrice nxn
1.2x10 -15
1x10 -15
8x10 -16
6x10 -16
4x10 -16
2x10 -16
0
0
200
400
600
800
1000
1200

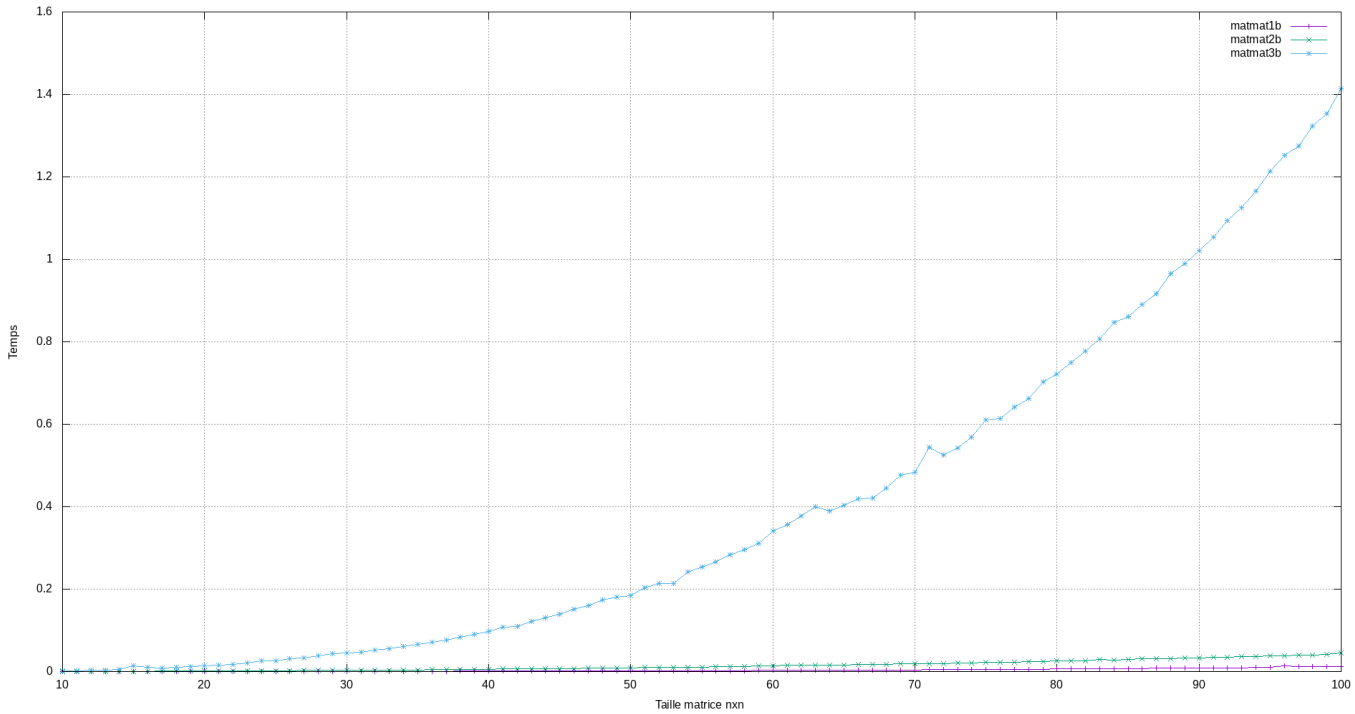
On remarque ici que l'erreur avant et le conditionnement sont liés : leurs courbes suivent une même trajectoire. On pouvait s'attendre à se résultats car comme expliqué plus tôt le conditionnement mesures la « capacité » de notre matrice à générer des erreurs de calculs. L'erreur arrière quand a elle est croissante : elle dépend de la taille de la matrice.

La complexité de cet algorithme étant quadratique, le temps d'exécution augmente très vite par rapport à la taille des matrices. C'est pour cela que l'algorithme n'a pas été exécuter sûr des matrices plus grandes que 1010.

Le code source de cet exercice se trouve dans le dossier TP2-code sous le nom Exercice7.sce

Exercice 8 : produit Matrice-Matrice :

Dans cet exercice nous avons codé un algorithme de multiplication de deux matrices entre elles. Nous avons dans un premier temps coder l'algorithme avec 3 boucles « ijk », puis deux boucles et pour finir avec une boucle. Nous avons ensuite mesuré et comparer le temps d'exécution des différents algorithmes. Nous avons obtenu les résultats suivant :



On remarque que le temps d'exécution du programme dépend du nombres de boucles : il est plus efficace d'appelé une version optimiser: 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.

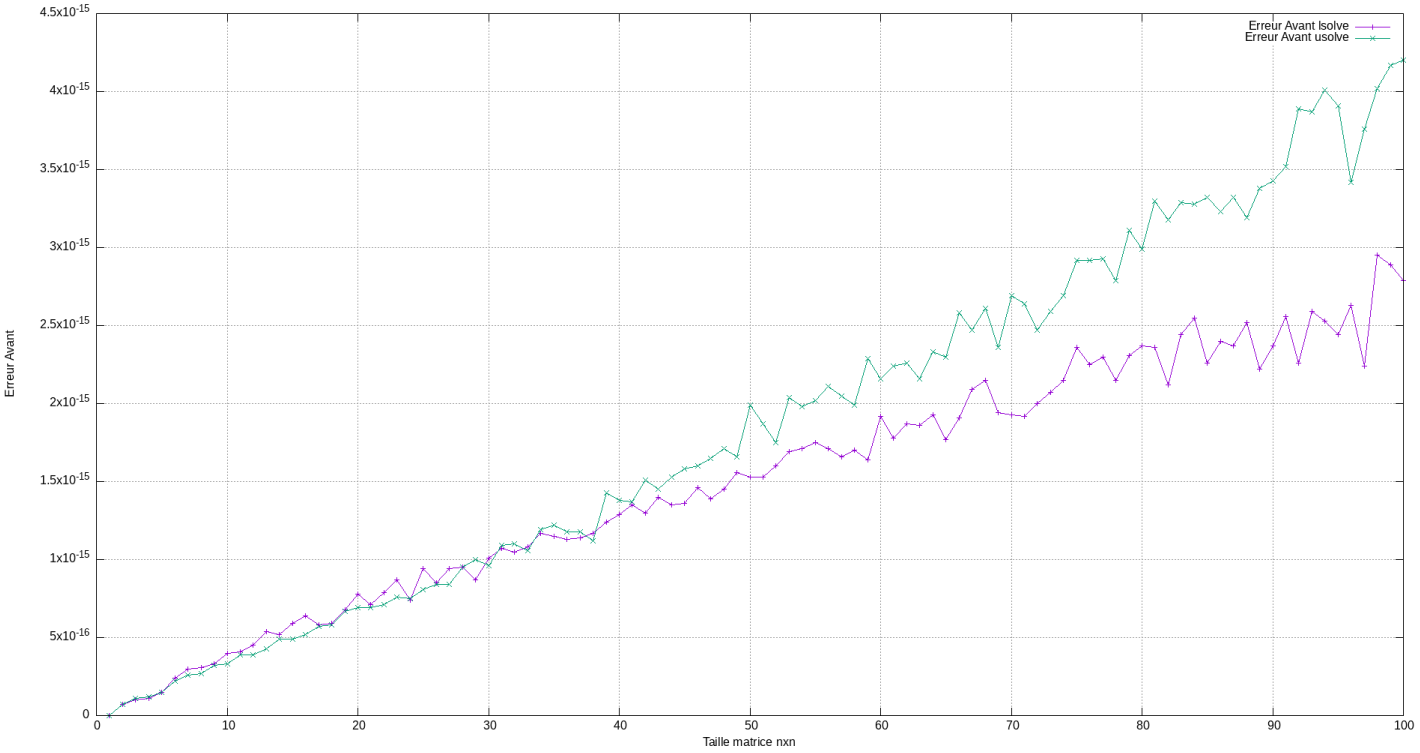
Les codes source de cet exercice se trouvent dans le dossier TP2-code sous les noms matmat3b.sci, matmat2b.sci, matmat1b.sci ainsi que mesure_temps.sci

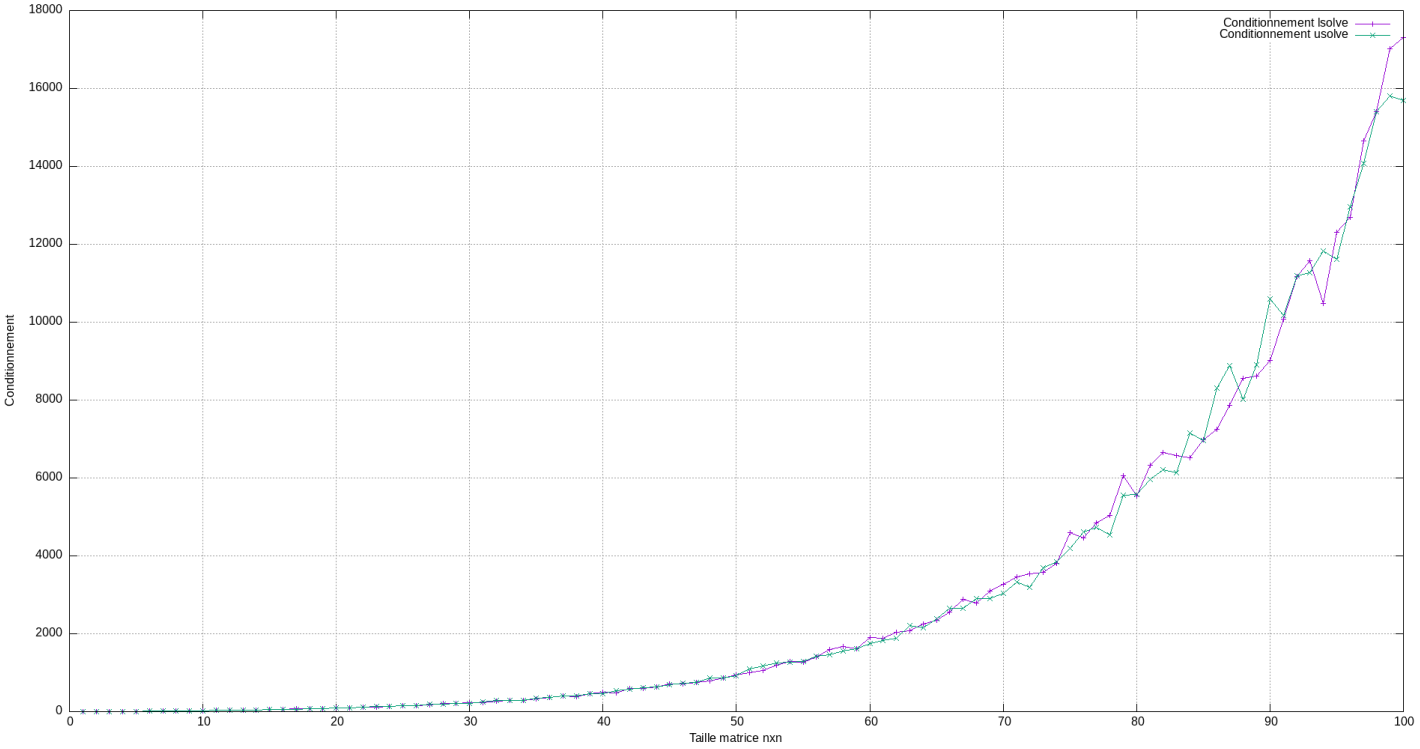
TP3 :

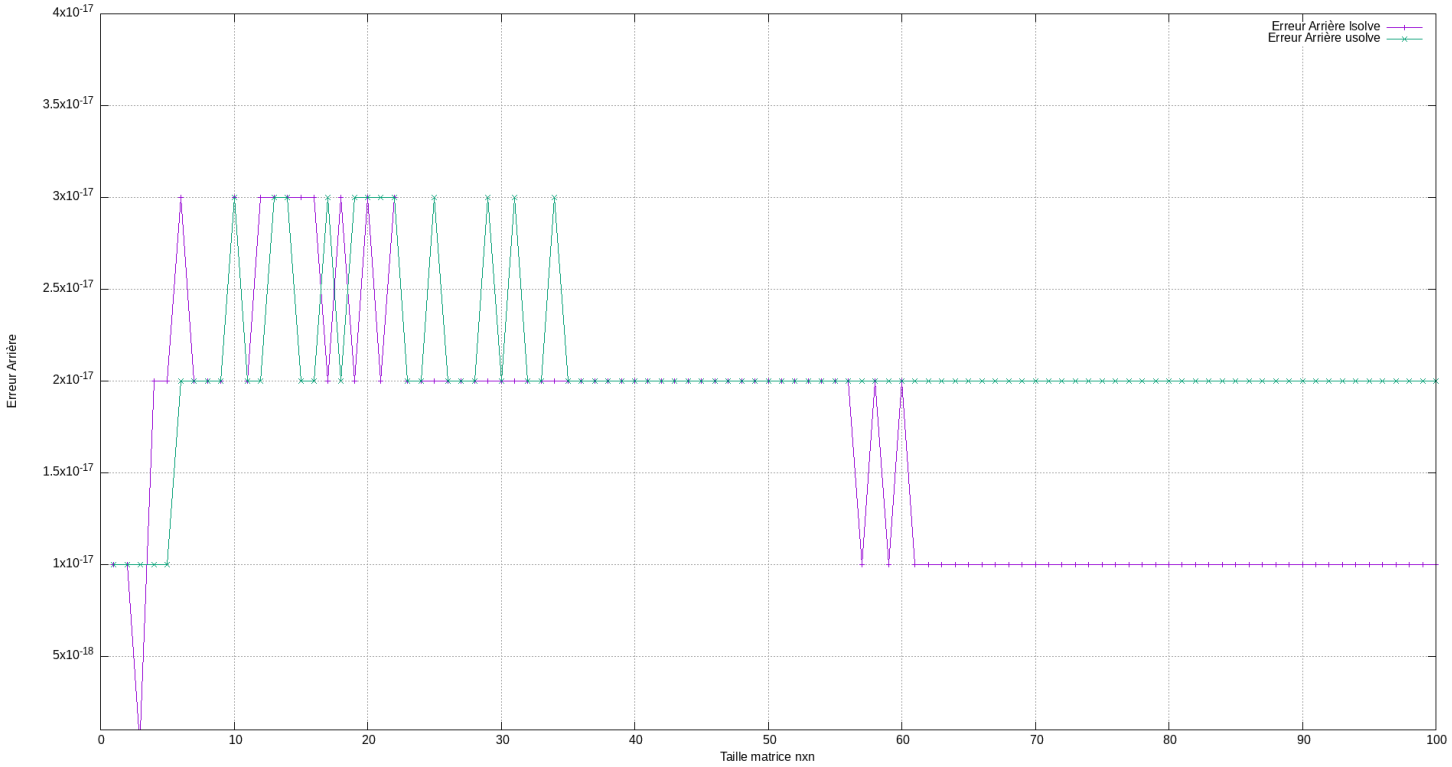
Dans ce TP nous avons commencé à coder nos premiers algorithmes plus complexes ainsi que mesuré leurs validité avec plusieurs métrique:

Exercice 2 : Système triangulaire :

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





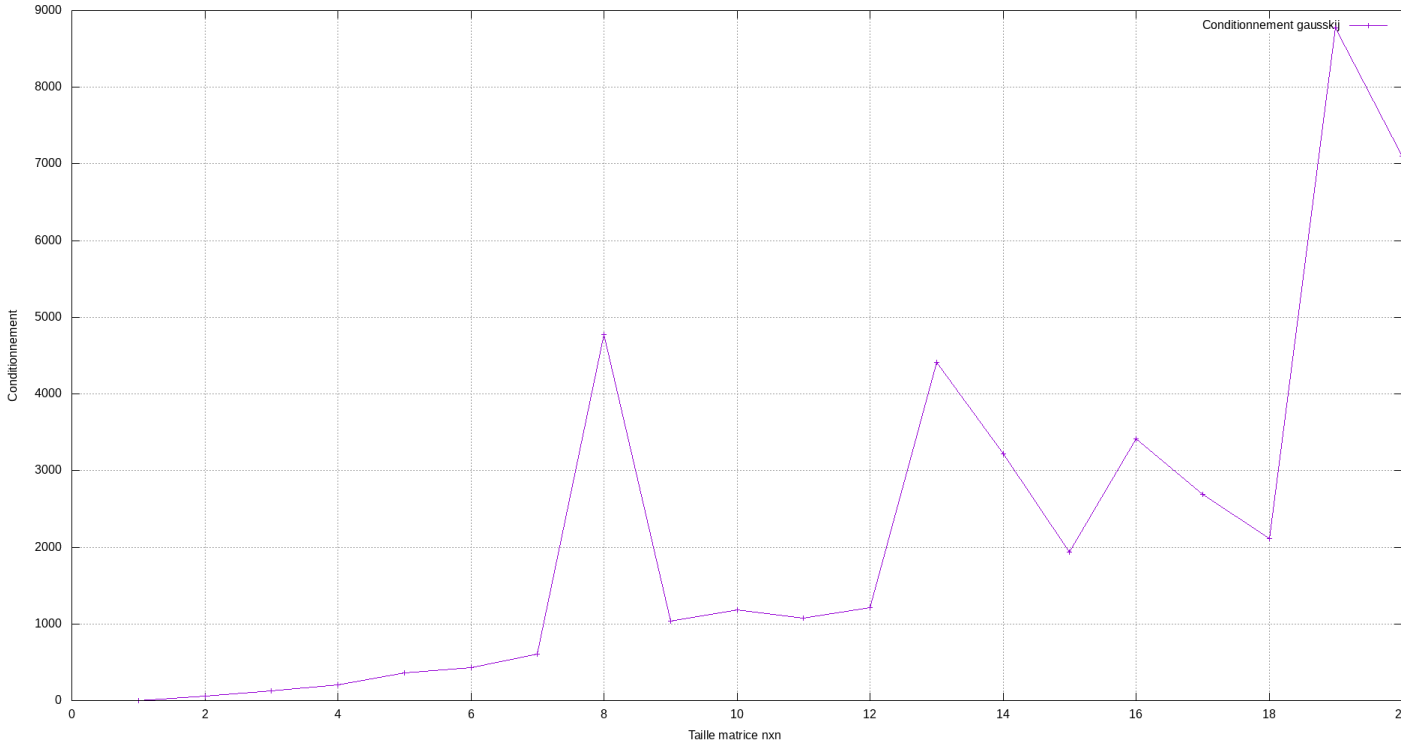


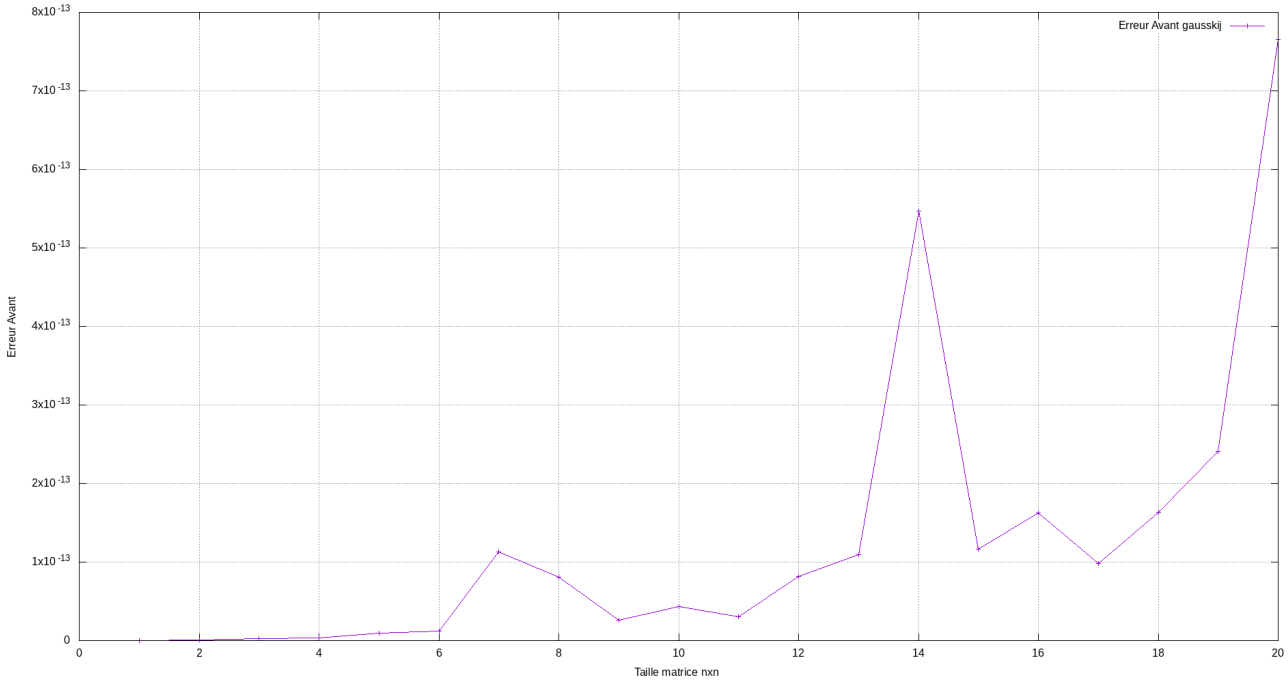
On remarque qu'encore un fois l'erreur et le conditionnement dépendent fortement de la taille de la matrice. En revanche l'erreur arrière semble stable et reste faible $<10^{-16}$. Les résultats pour l'erreur arrière ne sont pas précis : la précision des flottantes doubles précisions pour une machine 64 bit est de 10^{-16} on ne peut donc pas conclure sur l'évolution de celle-ci.

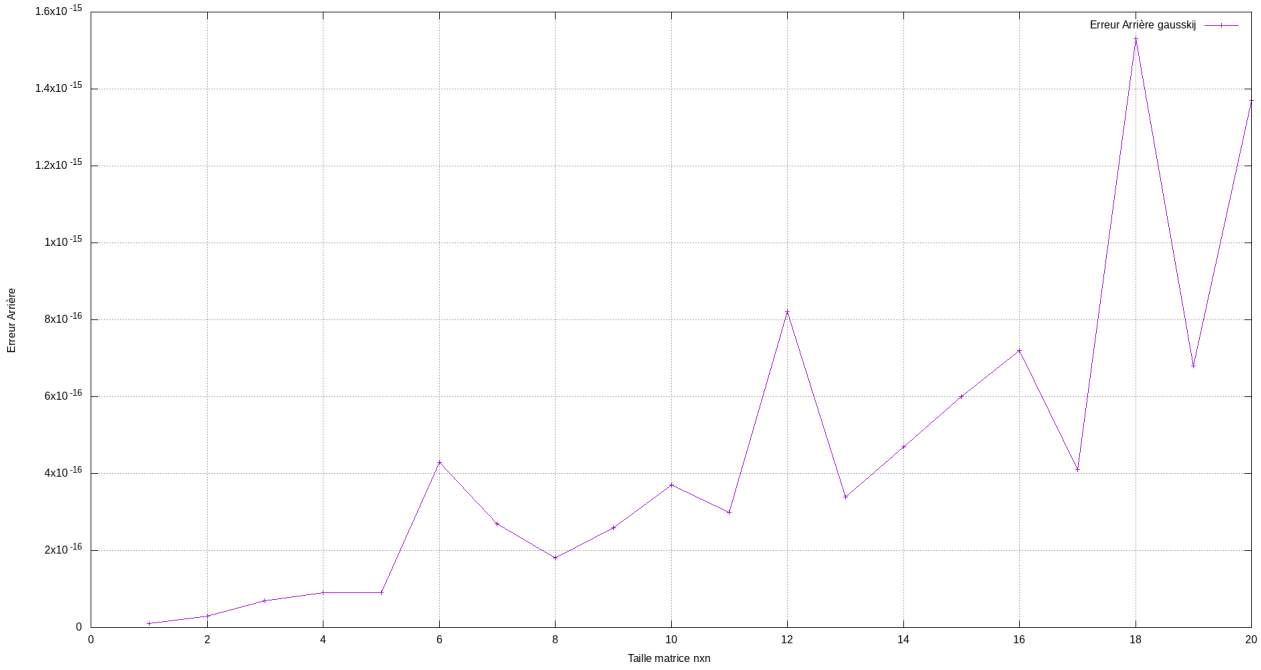
Les codes source de cet exercice se trouvent dans le dossier TP3-code sous les noms lsolve.sci, usolve.sci et test_u_l_solve.sci.

Exercice 3 : Gauss

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





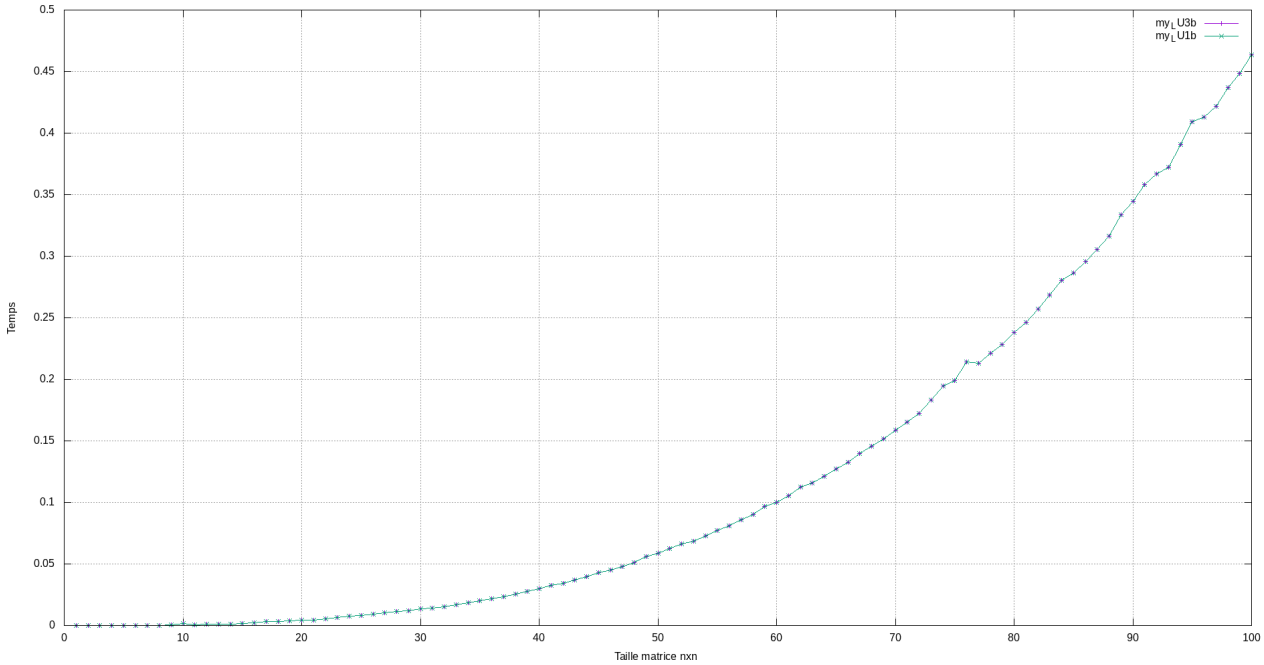


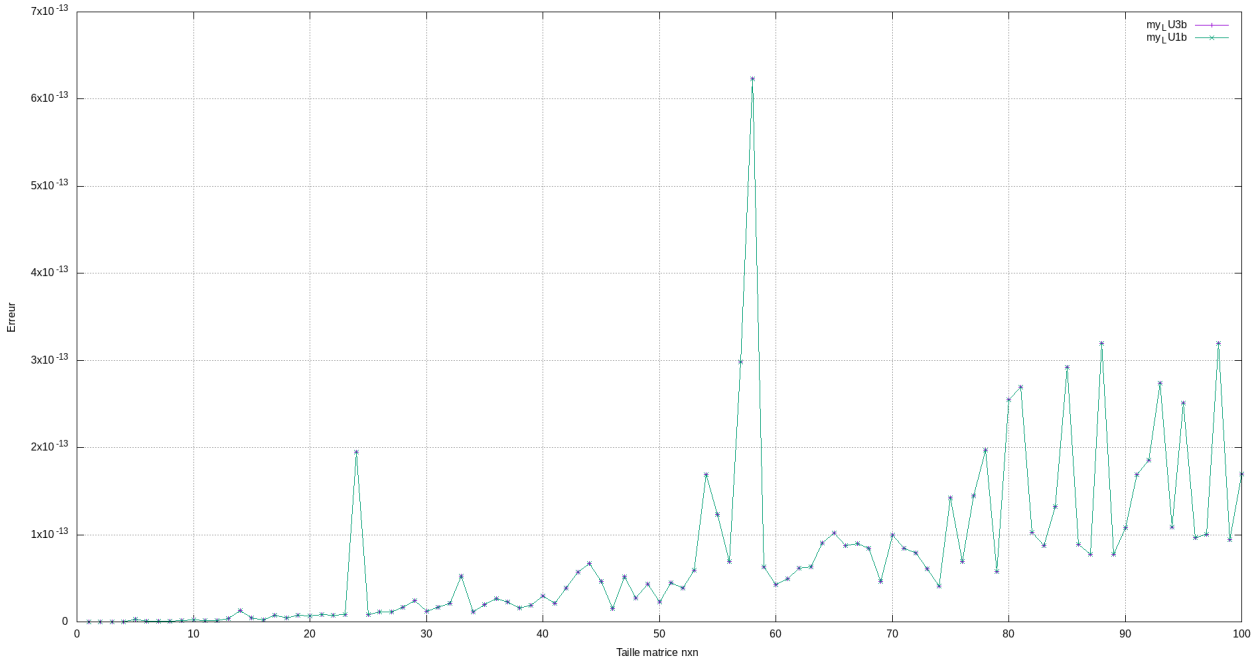
On remarque ici que plus la taille de la matrice augmente, plus l'erreur augmente. On essaie cet algorithme seulement sur de petite car la complexité de notre algorithme est en $O(2/3\ n^3)$ pour la triangularisation de la matrice et $O(1/2\ n^2)$ pour la résolution du système triangulaire. Ce qui nous fait donc une complexité de $O(2/3 * n^3 + 1/2 * n^2)$. La complexité est donc trop grande pour que cet algorithme s'exécute dans des temps raisonnables. Le temps d'exécution évolue de manière cubique lorsque l'on augmente linéairement la taille de la matrice.

Les codes source de cet exercice se trouvent dans le dossier TP3-code sous les noms gausskij3b.sci et test_gauss.sci.

Exercice 4 : LU

Dans cet exercice nous avons réalisé l'algorithme de factorisation LU à 1 et 3 boucles. Nous avons mesuré le temps d'exécution ainsi que l'erreur de la manière suivante : err = A – L*U





On remarque que les temps d'exécutions des deux algorithmes sont identiques, ce qui est surprenant. L'erreur de calcul des deux algorithmes sont identiques, on pouvait s'attendre à se résultats car les deux algorithmes sont les même : l'un est juste ne version « compressé » de l'autre. L'erreur semble légèrement croissante.

Les codes source de cet exercice se trouvent dans le dossier TP3-code sous les noms mylu3b.sci, mylu1b.sci, mylu.sci et test_lu3b.sci.

Annexe :

Dépôt github : https://github.com/Ennaox/CN-TP2-3