Rapport Calcul numérique

BATTISTON Ugo

December 2, 2021

Abstract

Dans ce document, vous trouverez des indications sur les algorithmes, compléxités, mais aussi des graphes de performances et des graphes sur les erreurs avants et arrières sur chaques exercices du TD2 et du TD3. Certains exercices n'ont pas pu être réalisés dans de bonnes conditions, pour cause le logiciel Scilab crash (c'est indiqué dans la partie de l'exercice lorsqu'il engendrait un plantage de l'application). Tout les codes écrit sont disponible sur un dépot github (voir annexe).

1 TP2

1.1 Exercice 6 - Prise en main de Scilab

L'exercice 6 est une prise en main du logiciel Scilab, utilisation des variables, et des fonctions.

1.2 Exercice 7 - Matrice random et problème "jouet"

L'ojectif de l'exercice 7 est de visualiser l'erreur avant et arrière que l'on obtient lors de la résolution d'équation du type $Ax = b$ tel que A $\in$ R^{n*n} inversible (det(A) $\neq$ 0), x et b $\in \mathbf{R^n}$.

Sur mon ordinateur (linux mint 20.2 cinamon), Scilab (version 6.1.0.0) crash sur l'implementation de cette exercice, je ne peux donc pas tester l'erreur. Pour exemple, le code disponible sur github (voir annexe) de l'exercice 2 crash sur ma machine.

Pour pouvoir quand même pouvoir tirer des conclusions, j'ai pris comme référence les graphiques obtenu par mes camarades. On observe que la courbe de l'erreur avant est en tout point supérieur à la courbe de l'erreur arrière mais aussi que les deux courbes suivent les mêmes variations. On observe aussi que la courbe de l'erreur avant est bornée par la courbe de l'erreur arrière * conditionnement de la matrice A.

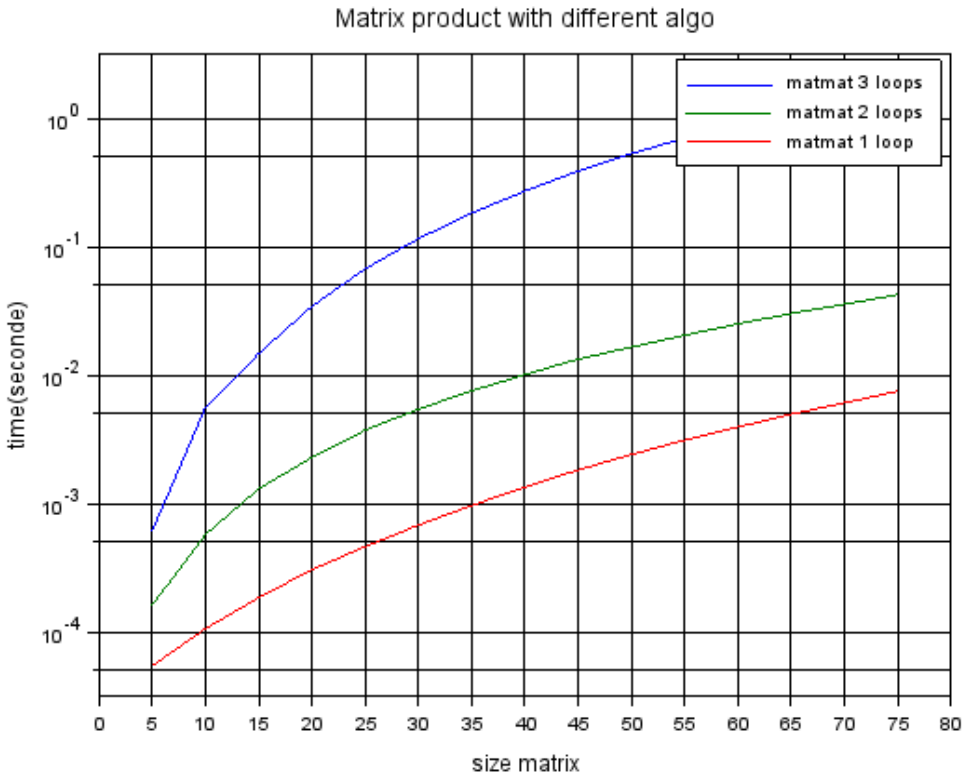
1.3 Exercice 8 - Produit Matrice-Matrice

L'exercice 8 consiste à l'implémentation de trois algorithmes de multiplication de matrice matrice avec 3 boucles, 2 boucles et 1 boucle.

L'algorithme de mutiplication matrice matrice de taille respective ($n*p$ et $p*m$) comporte trois boucles imbriquées de taille n, m, p. La compléxité de l'algorithme est donc en O($n*m*p$). Nous pouvons généraliser la formule aux matrices carrés de taille (n) où la compléxité est en O(n$^{\mathbf{3}}$).

L'idée de l'exercice est de réécrire l'algorithme naïf, puis de le réécrire en supprimant la boucle interne pour n'avoir plus que deux boucles et d'utiliser un produit scalaire. La dernière version de l'algorithme est une version ne comportant qu'une boucle et un produit matrice vecteur.

Pour les trois algorithmes et pour differentes taille de matrice, on obtient le graphique suivant:



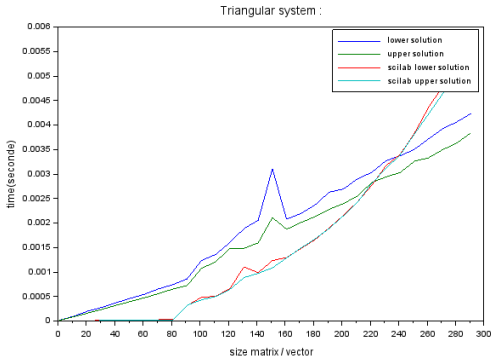
On peut observer une différence de temps notable pour les différents algorithmes dû à la différence d'implémentation d'où l'importance d'une bonne implémentation d'algorithme.

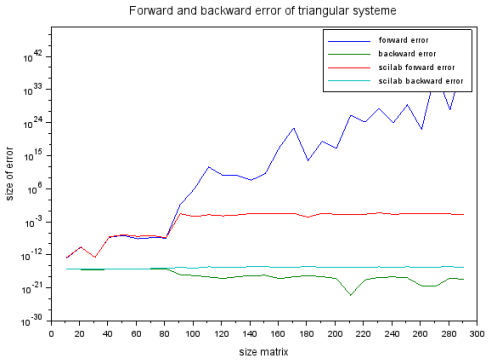
2 TP3

2.1 Exercice 2 - Système triangulaire

L'algorithme de résolution par remonté / descente de système triangulaire, permet comme son nom l'indique la résolution de systeme de la forme $Ax = b$ où A $\in$ R^{n*n} (matrice triangulaire inférieur ou supérieur) inversible (det(A) $\neq$ 0), x et b $\in \mathbf{R^n}$.

L'algorithme est en compléxité O(n$^\mathbf{2}$), une boucle itérant sur toutes les lignes de la matrice, et dans cette dernière nous retrouvons un produit scalaire entre deux vecteurs de taille allant de 1 à n.





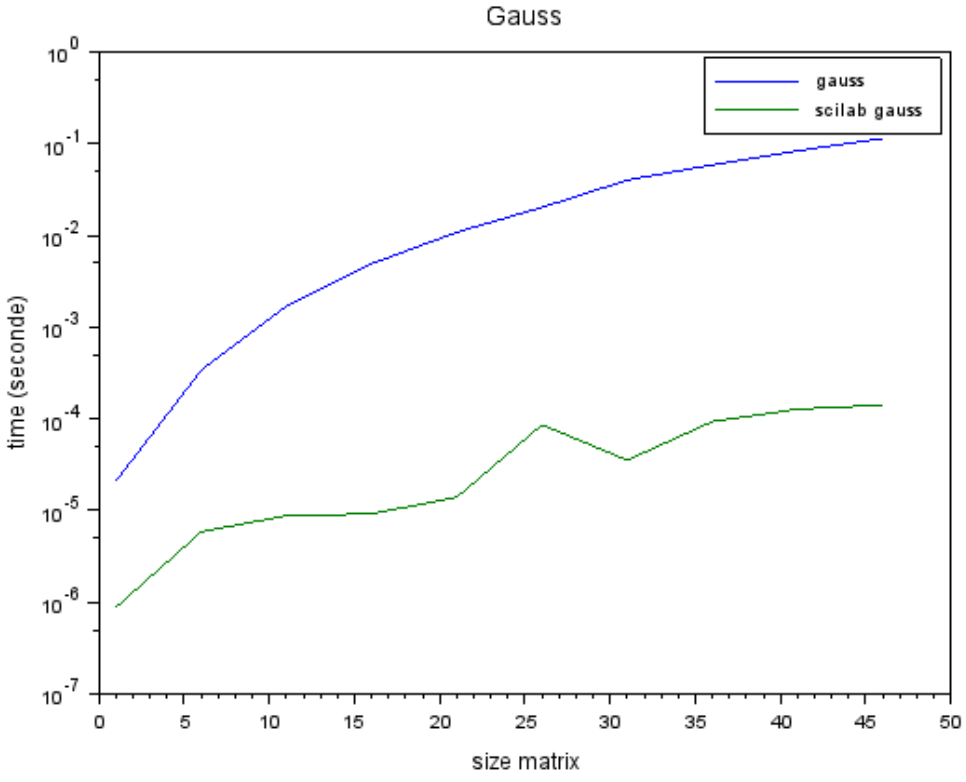
On peut remarquer que l'implémentation de l'algorithme obtient des temps inférieur à l'implémentation de Scilab (qui utilise la librarie optimisé Lapack) puis pour des matrices de tailles 220 l'algorithme de Scilab repasse devant en terme de temps d'exécution. Ceci est du à la différence d'implémentation, je suppose que pour résoudre un système triangulaire, Scilab ne sait pas qu'il est triangulaire et fait une factorisation LU, d'où un temps d'exécution plus long, mais au final sur des grosse matrice l'optimisation fait par Lapack permet quand même d'optenir des temps d'exécutions inférieurs.

On observe sur le graphique de calcul d'erreur de la résolution de système triangulaire par remonté, que la version de Scilab obtient une erreur avant croissante jusqu'à une valeur limite (10^1 pour des matrices de taille 60) puis la courbe semble rester constante autour de cette valeur. L'erreur avant quant à elle reste constante du début à la fin de l'expérimentation autour de 10^{-16}. La version de l'algorithme écrit à la main obtient une erreur arrière qui suit la tendence de l'erreur arrière de l'algorithme de Scilab à 10^{-16} mais à partir de matrices de taille 60, l'erreur diminue pour atteindre 10^{-19} pour des matrices de taille 300. L'erreur avant suit aussi la même tendance que l'erreur avant générée par Scilab pour des tailles de matrices de 0 à 60, puis, continu de croitre pour atteindre 10^{40} pour des matrices de 300.

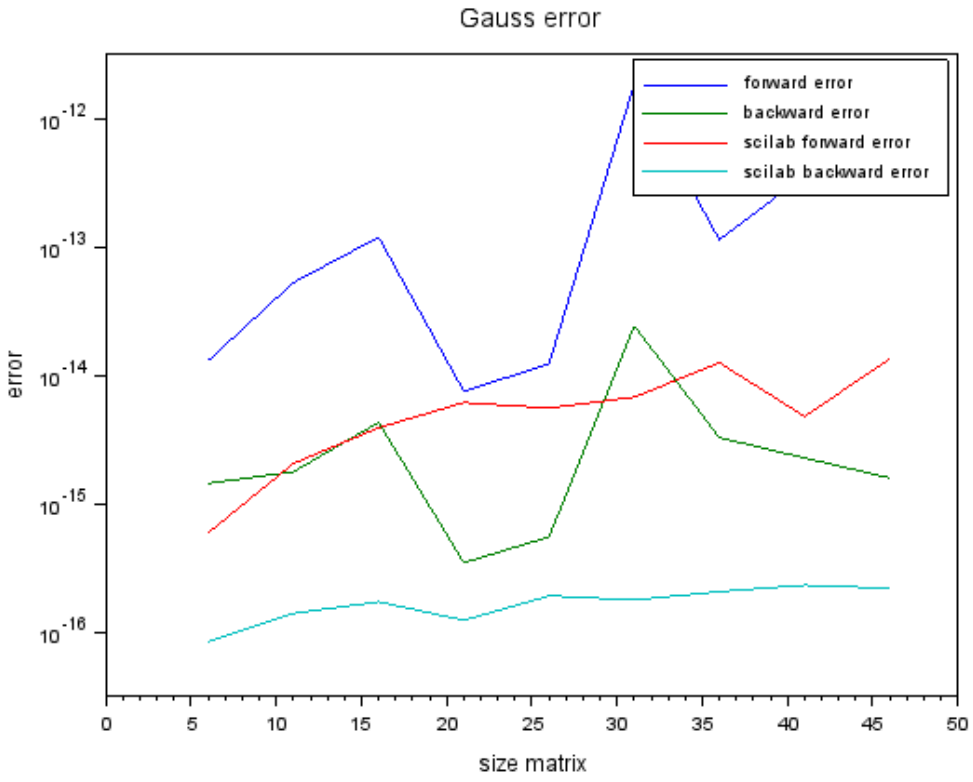
2.2 Exercice 3 - Gauss

Contrairement à l'algorithme précédent, l'algorithme de gauss permet la résolution de système de la forme $Ax = b$ où A n'est pas obligatoirement une matrice triangulaire. Dans un premier temps, l'algorithme de gauss triangularise la matrice, dans un second temps il utilise l'algorithme vu au dessus de résolution de système triangulaire pour de résoudre le nouveau système.

L'algorithme de triangularisation est en compléxité $O(\frac{2}{3}n^3)$ (une boucle déranlant toutes les colonnes avec dedans une boucle itérant les lignes et encore dedans une boucle pour la soustraction de deux sous ligne), et l'algorithme de résolution de système linéaire est en $O(n^2)$ (voir section 2.1). L'algorithme de gauss est au final en $O(\frac{2}{3}n^3 + n^2)$.



Tout comme l'exercice 7 du TD2, Scilab a des soucis de crash lorsque la taille des matrices dépassent 50x50. Toutes les mesures sont donc prisent sur des matrices 50x50 au maximum (résultats sûrement non fiables, pas assez de tailles différentes pour en tirer des conclusions...). On remarque tout de même que l'implémentation de Scilab obtient de bien meilleurs résultats que la version faite à la main (facteur de 10^2 en moins).



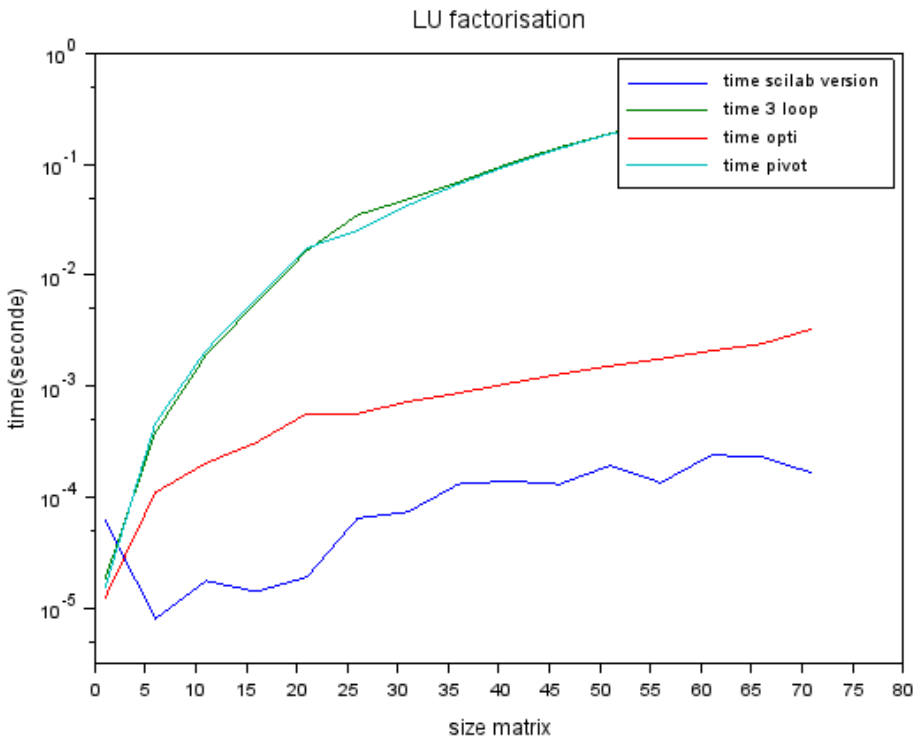
Comme dit à l'exercice 7 du TD2, l'erreur avant borne l'erreur arrière pour les deux implémentations de l'algorithme. De plus nous pouvons très bien remarquer que les variations des courbes des erreurs avants et arrières ont les mêmes variations.

2.3 Exercice 4 - TP LU

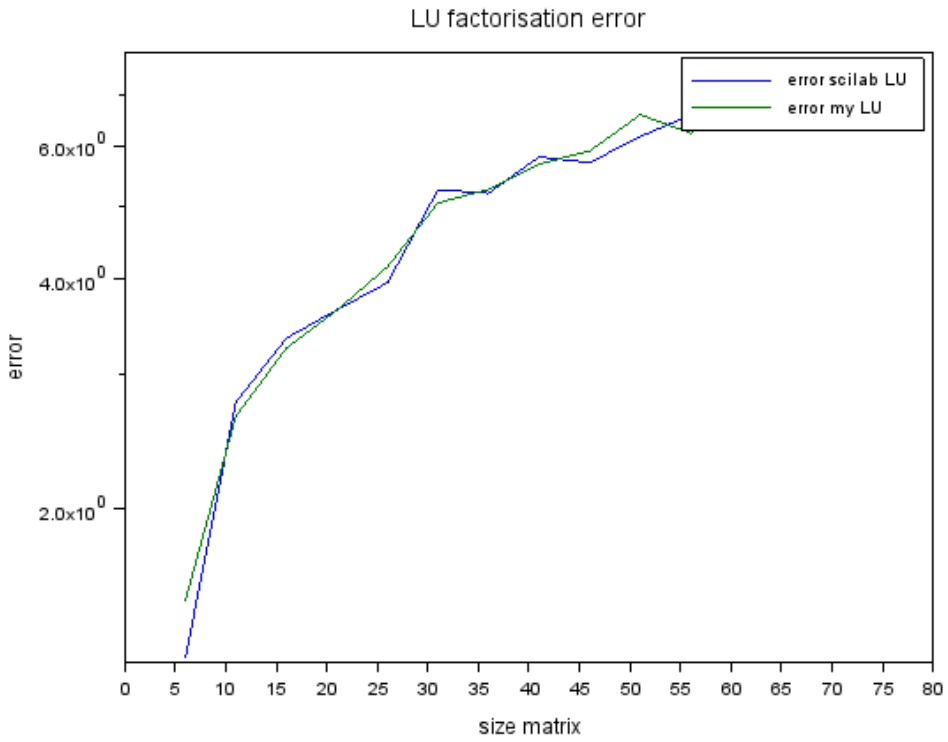
L'algorithme de factorisation LU permet la résolution de système linéaire de la forme Ax = b tel que A $\in \mathrm{R}^{n*n}$ inversible (det(A) $\neq 0$), x et b $\in \mathrm{R}^n$. Il permet l'écriture d'une matrice A sous la forme d'une multiplication de deux matrices triangulaires, inférieur avec des 1 sur la diagonal L et supérieur U où $A = L * U$. La factorisation LU ne dépend pas de b, donc une fois factorisé, une monté sur U ou une redescente sur L permet de résoudre le système pour tout b. Contrairement à l'algorithme de gauss qui dépend de b.

La compléxité de l'algorithme de factorisation LU est en $\mathrm{O}(\frac{2}{3}n^3)$ (une boucle itérant sur $n-1$ valeurs, avec dedans une boucle faisant à l'itération i $i-1$ divisions donc au total $\frac{n(n-1)}{2}$ opérations. Pour la deuxiéme boucle, elle itère sur toutes les valeurs de la sous matrice $(i:n, i:n)$ matrices et fait une soustraction et une multiplication donc $2*(i-1)^2$ opération donc $\frac{2n^3}{3} - n^2 + \frac{2n}{3}$. On obtient donc une compléxité en $\mathrm{o}(\frac{2n^3}{3} - n^2 + \frac{2n}{3} + \frac{n(n-1)}{2})$ et en $\mathrm{O}(\frac{2n^3}{3})$.

Une autre version de l'algorithme existe, c'est la factorisation LU avec pivot partiel de la forme $P * A = L * U$ avec A $\in \mathrm{R}^{n*n}$ inversible, P $\in \mathrm{R}^{n*n}$ (matrice de permutation), L $\in \mathrm{R}^{n*n}$ unit lower triangular et U $\in \mathrm{R}^{n*n}$ matrice triangulaire supérieur. Il permet de résoudre des soucis lorsque les valeurs sont trop proches de 0. Le pivot partiel à lui seul a pour compléxité $\mathrm{O}(n^2)$ et se situe dans la première boucle. On obtient donc une compléxité de $\mathrm{O}(\frac{2}{3}n^3 + n^3)$ donc en $\mathrm{O}(\frac{5}{3}n^3)$.



Nous pouvons voir que la version 3 boucles optimisées est plus rapide que la verison 3 boucles simple. Nous remarquons aussi que la version avec le pivot est aussi rapide que la version 3 boucles classique dû au pivot qui rajoute de la compléxité à l'algorithme. Et en dernier la version Scilab basé sur Lapack qui obtient les temps d'exécutions les plus rapides.



Les erreurs obtenues entre la versions Scilab et version de factorisation LU avec pivot partiel réécrite correspondent.

Annexe

Liens du dépot gitbub : https://github.com/johnkyky/Calcul-Numerique