Institut Polytechnique des Sciences Avancées
Campus de Paris
63 bis, Boulevard de Brandebourg
94 200 Ivry-sur-Seine

Etablissement Privé d'Enseignement Supérieur
SIRET 433 695632 00011 - APE 803Z

Analyse numérique (Ma223)

TP 1 : Méthode de Gauss pour la résolution de systèmes linéaires.

Dans ce TP, nous allons utiliser le langage de programmation Python et la librairie `numpy` afin de permettre d'appliquer l'algorithme de Gauss pour la résolution de systèmes linéaires.

Léa Dupin
Aéro 2 - Classe F2

Année scolaire 2020 | 2021

Table des matières

Introduction

Dans le domaine de l'ingénierie, ou même des sciences globalement, les systèmes linéaires sont très largement utilisés. Les systèmes de plusieurs équations à plusieurs inconnues, mais aussi la plupart des schémas numériques de résolution d'équations (différentielles ou linéaires) s'expriment en termes de matrices, de taille plus ou moins grande, avec des applications en biologie, statistiques, mécanique des fluides... En bref, de nombreuses applications sont possibles dans la représentation sous forme de matrices ou d'espaces vectoriels.
C'est pour cela que la résolution de systèmes de taille parfois très élevée a dû être optimisée, notamment aux moyens d'algorithmes plus ou moins efficaces et plus ou moins rapides.

Dans notre cas, nous allons aujourd'hui étudier plusieurs méthodes, que nous avons programmées grâce au langage informatique Python, et la librairie `numpy`. Ces méthodes sont celles du pivot de Gauss, de la décomposition LU, de deux variantes (le pivot partiel et le pivot total), et finalement le solveur intégré à la librairie de Python. Chacune de ces méthodes sera explicitement contextualisée et détaillée dans la partie la concernant.

Nous allons également travailler sur les notions de temps de calcul, et de calcul de l'erreur. Dans notre cas, le temps de calcul est le temps nécessaire à l'ordinateur pour résoudre un système linéaire de taille $n * n$ avec une des méthodes étudiées. Le calcul de l'erreur sera calculé par la norme de $\|A\,X - B\|$, qui doit être au plus près de 0 pour que la méthode puisse être qualifiée de « précise ».

Nous allons donc étudier ces cinq méthodes séparément, puis analyser les résultats obtenus avant de comparer les différentes méthodes et conclure sur la plus efficace à utiliser, autant sur le critère du temps de calcul que de la précision.

Les programmes seront exécutés sur deux ordinateurs différents afin de pouvoir comparer les temps de calcul en fonction des performances de l'appareil :

ASUS VivoBook S15 S530	ASUS ZenBook Pro UX550VD
Processeur Intel® Core™ i7-8565U (8th generation) CPU @ 1.80GHz 1.99 GHz Mémoire RAM 8.00 Go	Processeur Intel® Core™ i7-7700HQ (7th generation) CPU @ 2.80 GHz 2.81 GHz Mémoire RAM 8.00 Go

Le programme informatique pour ce TP est disponible sur le site internet GitHub en suivant ce lien : https://github.com/LeaDupinAero2F2/Ma223---TP1.git

Partie 1 : Algorithme de Gauss

La méthode de Gauss est connue des mathématiciens chinois depuis au moins le Ier siècle de notre ère. Elle est référencée pour la première fois dans le livre chinois *Jiuzhang suanshu* (Les Neuf Chapitres sur l'art mathématique). En Europe, en 1810, Carl Friedrich Gauss présente sa méthode dans un livre étudiant le mouvement de l'astéroïde Pallas[1]. Dans ce livre, il décrit une méthode générale de résolution de système d'équations linéaires qui constitue l'essentiel de la méthode du pivot. En 1888, Wilhelm Jordan publie un livre de géodésie précisant comment utiliser cette méthode et adoptant une notation un peu différente. C'est grâce à ce dernier livre que cette méthode se diffusa dans tout l'Occident.[2]

Question 1 : Réduction

Dans cette fonction, on récupère la taille de la matrice fournie afin de pouvoir calculer le pivot pour chaque ligne de celle-ci, et on calcule ensuite la nouvelle ligne grâce à ce pivot. On obtient alors la matrice A réduite, sous la forme d'une matrice triangulaire supérieure.

Question 2 : Résolution

Dans cette fonction, on récupère la taille de la matrice fournie afin de trouver la matrice solution X du système $T\,X = B$, où T est la matrice réduite renvoyée par la fonction précédente.

Question 3 : Rassembler

Cette troisième fonction permet de lier les deux premières. On commence tout d'abord par créer la matrice augmentée en ajoutant B à A, puis on réduit celle-ci, et on termine par la résolution grâce au pivot de Gauss. La fonction retourne alors la matrice solution X du système $A\,X = B$, où A et B sont les deux matrices données en entrée.

Question 4 : Tester

Nous allons maintenant tester notre algorithme. Nous choisissons de mesurer le temps mis par l'ordinateur pour une matrice de taille n, avec n compris entre 0 et 600, et un pas de 1.

Taille de la matrice $(n * n)$	**Temps ordinateur 1 (en s)**	**Temps ordinateur 2 (en s)**
100	0.015625	0.015625
200	0.078125	0.09375
300	0.15625	0.203125
400	0.3125	0.34375
500	0.5625	0.59375
600	1.234375	1.0
Total	**154.734375** **(Environ 2 minutes et 35 secondes)**	**177.59375** **(Environ 2 minutes et 58 secondes)**

[1] https://fr.wikipedia.org/wiki/(2)_Pallas
[2] https://fr.wikipedia.org/wiki/%C3%89limination_de_Gauss-Jordan

Nous obtenons alors ces graphiques :

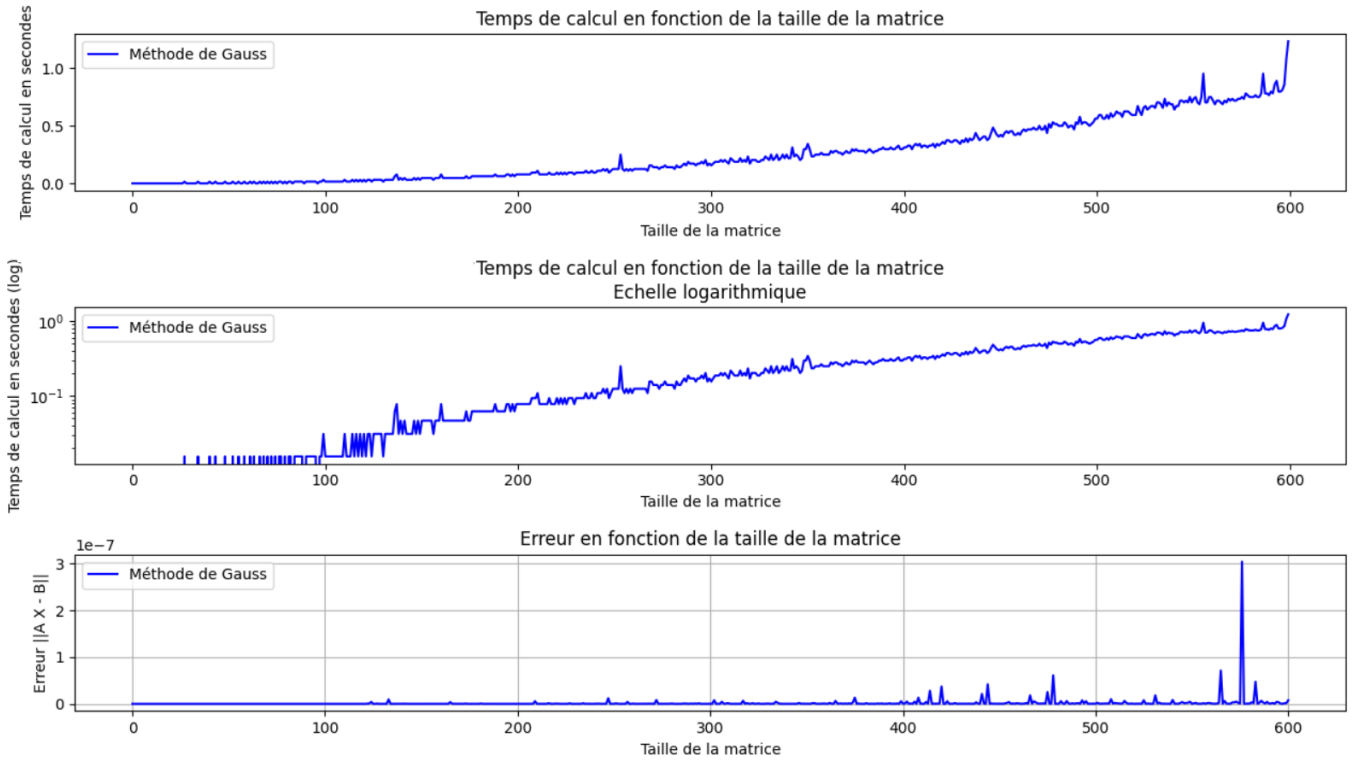

Figure 1 : Courbes obtenues pour la résolution de 600 matrices $n * n$ de taille comprise entre 1 et 600, par pas de 1
Méthode de Gauss | Ordinateur 1

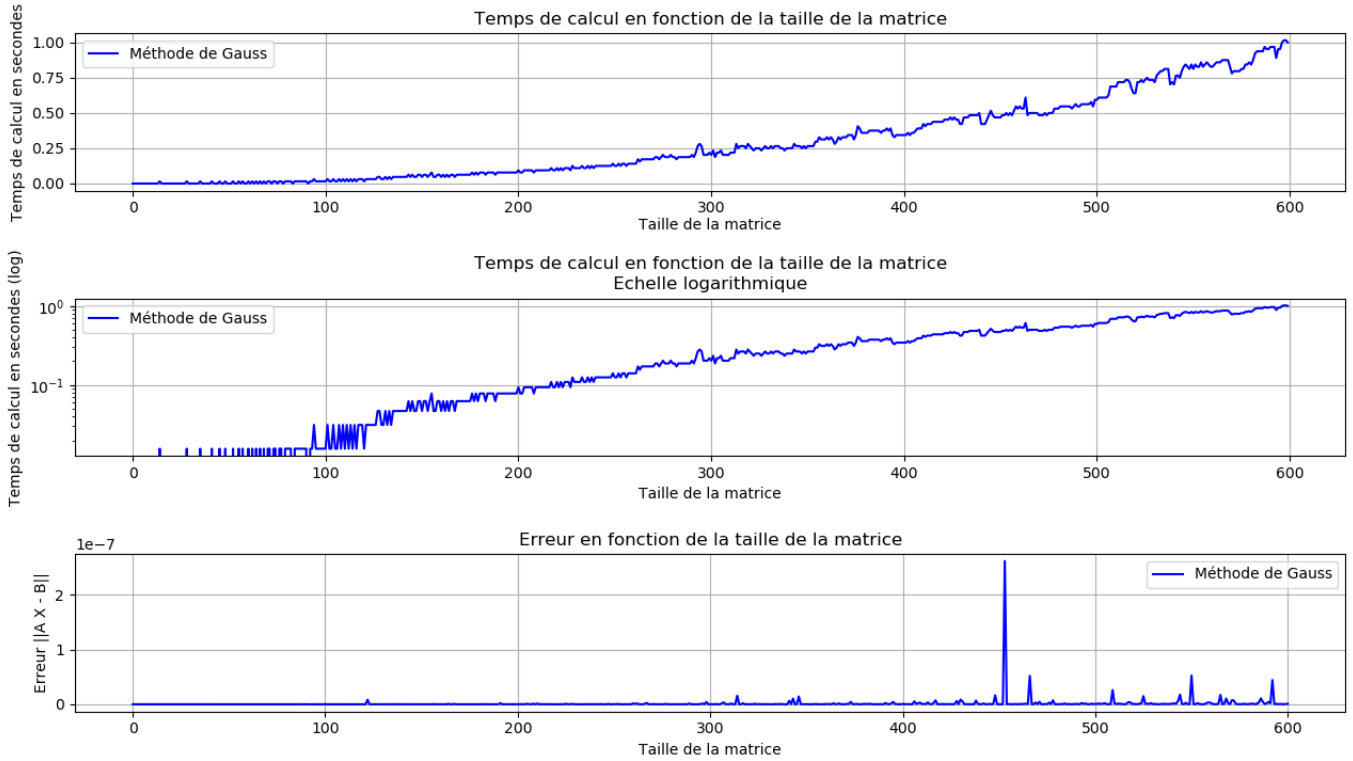

Figure 2 : Courbes obtenues pour la résolution de 600 matrices $n * n$ de taille comprise entre 1 et 600, par pas de 1
Méthode de Gauss | Ordinateur 2

On peut constater que le temps de calcul est semblable, et plus « linéaire » avec le premier ordinateur, tandis qu'avec le second il y a plus d'amplitude dans les courbes obtenues.
Les erreurs sont sensiblement les mêmes, ce qui semble normal puisque ce critère dépend plus du programme que de l'appareil l'exécutant. L'erreur est très faible : on peut voir que malgré une pique à $3 * 10^{-7}$ pour les deux appareils, la majorité des résultats présente une erreur bien plus faible.

Partie 2 : Décomposition LU

La décomposition LU est une méthode de décomposition d'une matrice comme produit d'une matrice triangulaire inférieure L (comme lower, inférieure en anglais) par une matrice triangulaire supérieure U (comme upper, supérieure). Si on pose A une matrice carrée, on dit que A admet une décomposition LU s'il existe une matrice triangulaire inférieure formée de 1 sur la diagonale, notée L, et une matrice triangulaire supérieure, notée U, qui vérifient l'égalité $A = L\,U$.[3]

Question 1 : Décomposition

Cette question nécessite la modification de la fonction de réduction de Gauss que nous avons programmé précédemment, car la matrice U que l'on obtient à la fin de cette étape est la réduction de la matrice carrée, et L est la matrice contenant tous les pivots calculés. Nous avons pris soin de vérifier que les matrices L et U obtenues vérifient l'égalité $A = L\,U$ dans le but de vérifier si notre programme est correct.

Question 2 : Résolution

Nous allons maintenant tester notre algorithme. Nous choisissons de mesurer le temps mis par l'ordinateur pour une matrice de taille n, avec n compris entre 0 et 600, et un pas de 1.

Taille de la matrice $(n * n)$	**Temps ordinateur 1 (en s)**	**Temps ordinateur 2 (en s)**
100	0.03125	0.03125
200	0.109375	0.125
300	0.234375	0.28125
400	0.421875	0.46875
500	0.734375	0.78125
600	1.0625	1.265625
Total	**201.796875** **(Environ 3 minutes et 22 secondes)**	**229.46875** **(Environ 3 minutes et 49 secondes)**

[3] https://fr.wikipedia.org/wiki/D%C3%A9composition_LU

Nous obtenons alors ces graphiques :

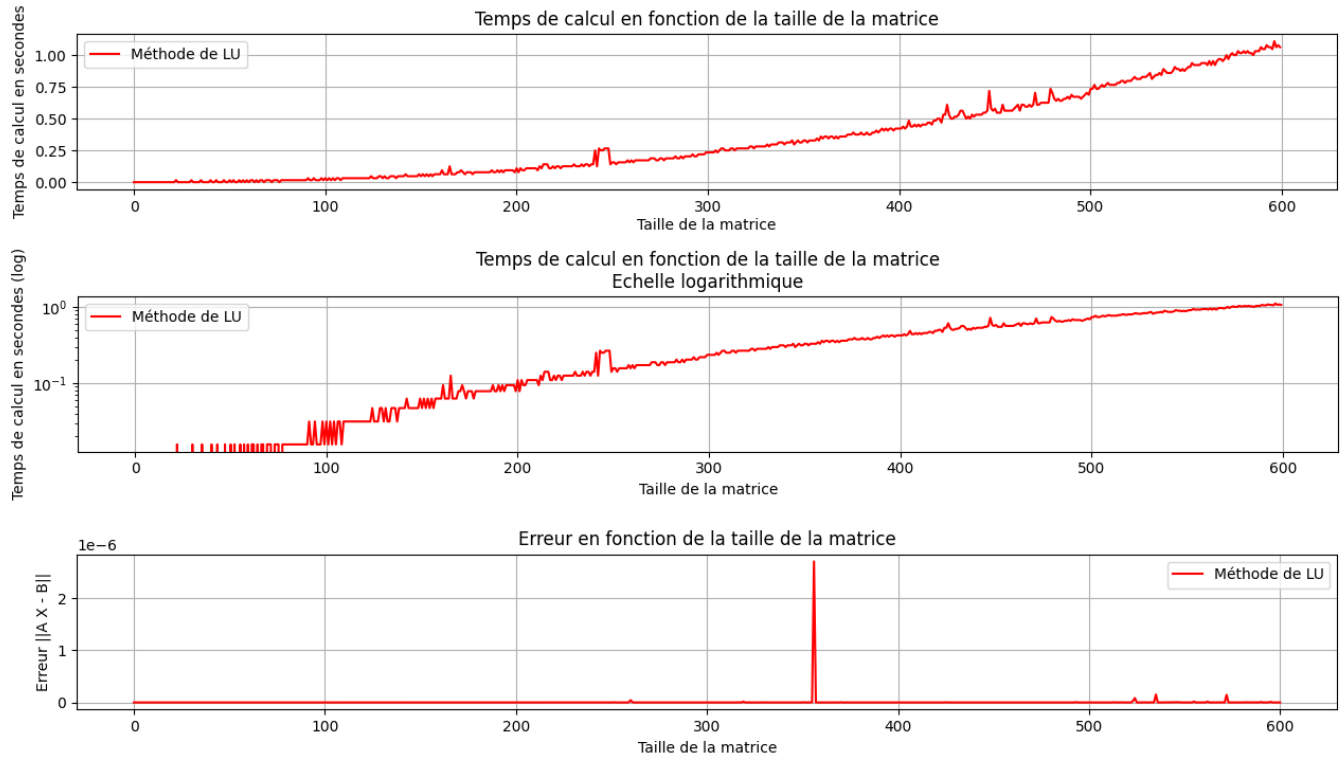

Figure 3 : Courbes obtenues pour la résolution de 600 matrices $n * n$ de taille comprise entre 1 et 600, par pas de 1 Décomposition LU| Ordinateur 1

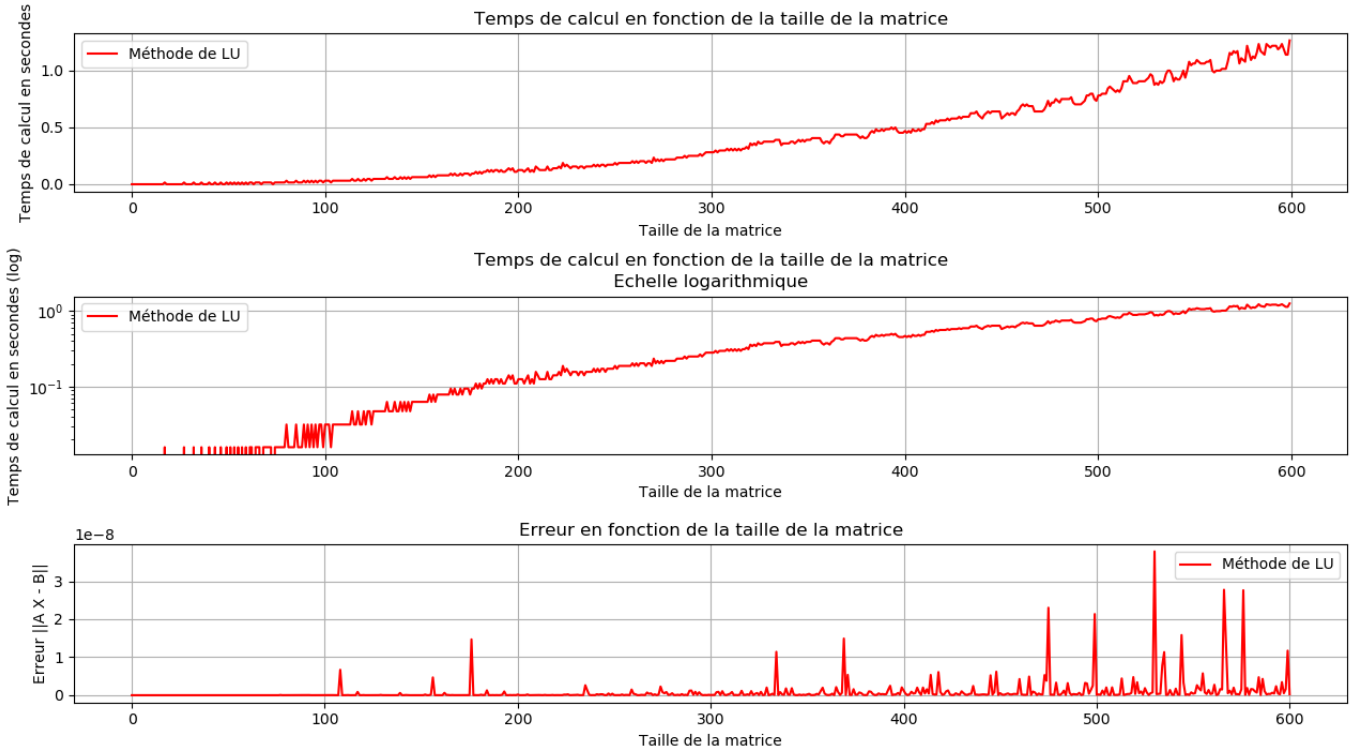

Figure 4 : Courbes obtenues pour la résolution de 600 matrices $n * n$ de taille comprise entre 1 et 600, par pas de 1 Décomposition LU| Ordinateur 2

Pour cette deuxième méthode, on peut noter que le temps de calcul semble plus grand pour le premier appareil mais plus court pour le second. Les erreurs de calculs sont sensiblement du même ordre, aux pics maximums.

Partie 3 : Variantes de l'algorithme de Gauss

Les variantes de l'algorithme de Gauss consistent à interchanger les lignes (pivot partiel) ou bien les lignes et colonnes (pivot total) de la matrice A pour pouvoir utiliser le meilleur pivot possible pour faciliter les calculs, ou bien pour éviter un pivot nul[4], puisque l'on divise les éléments par le pivot (et qu'une division par 0, ça ne marche pas très bien……).

Question 1 : Pivot partiel

Taille de la matrice $(n * n)$	Temps ordinateur 1 (en s)	Temps ordinateur 2 (en s)
100	0.03125	0.03125
200	0.078125	0.09375
300	0.171875	0.234375
400	0.359375	0.4375
500	0.703125	0.640625
600	0.953125	0.953125
Total	**179.359375** **(Environ 2 minutes et 59 secondes)**	**189.125** **(Environ 3 minutes et 09 secondes)**

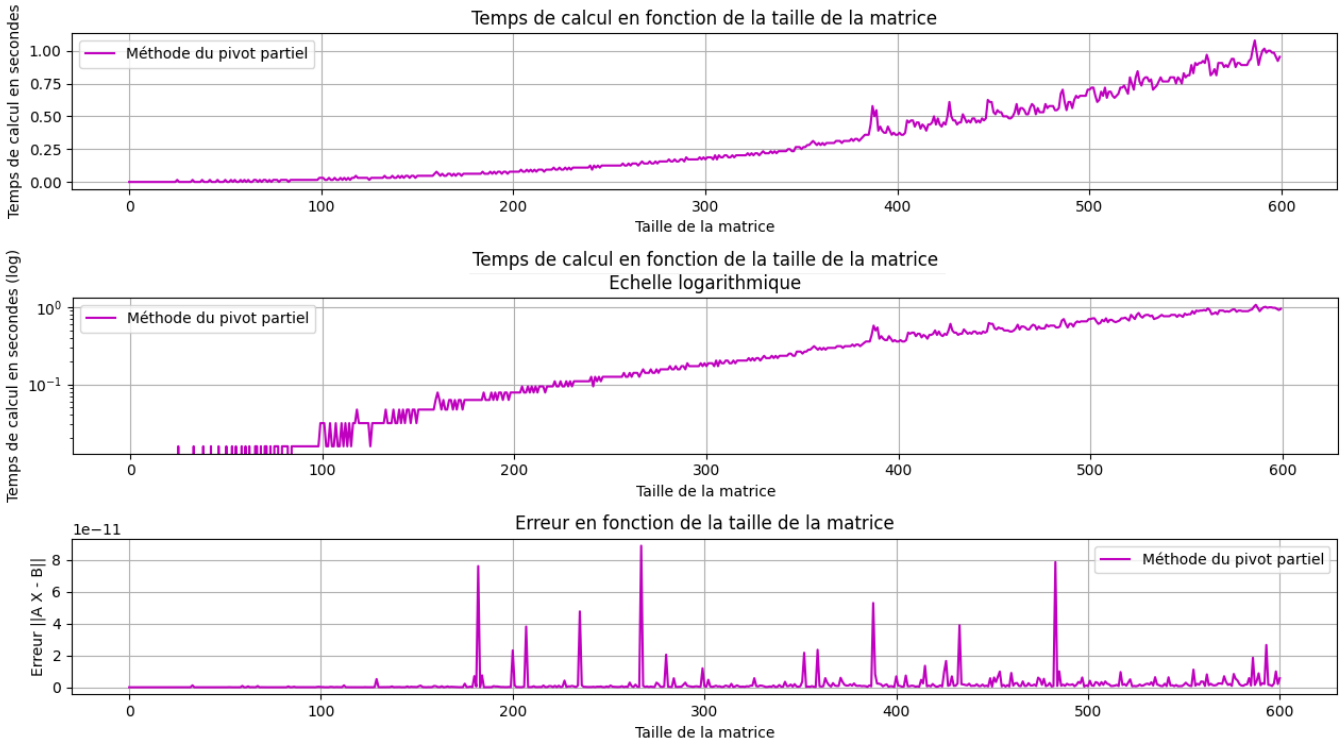

Figure 5 : Courbes obtenues pour la résolution de 600 matrices $n * n$ de taille comprise entre 1 et 600, par pas de 1
Méthode du pivot partiel | Ordinateur 1

[4] http://wwwens.aero.jussieu.fr/lefrere/master/mni/mncs/te/te1-alg-lin/gauss-jordan.pdf

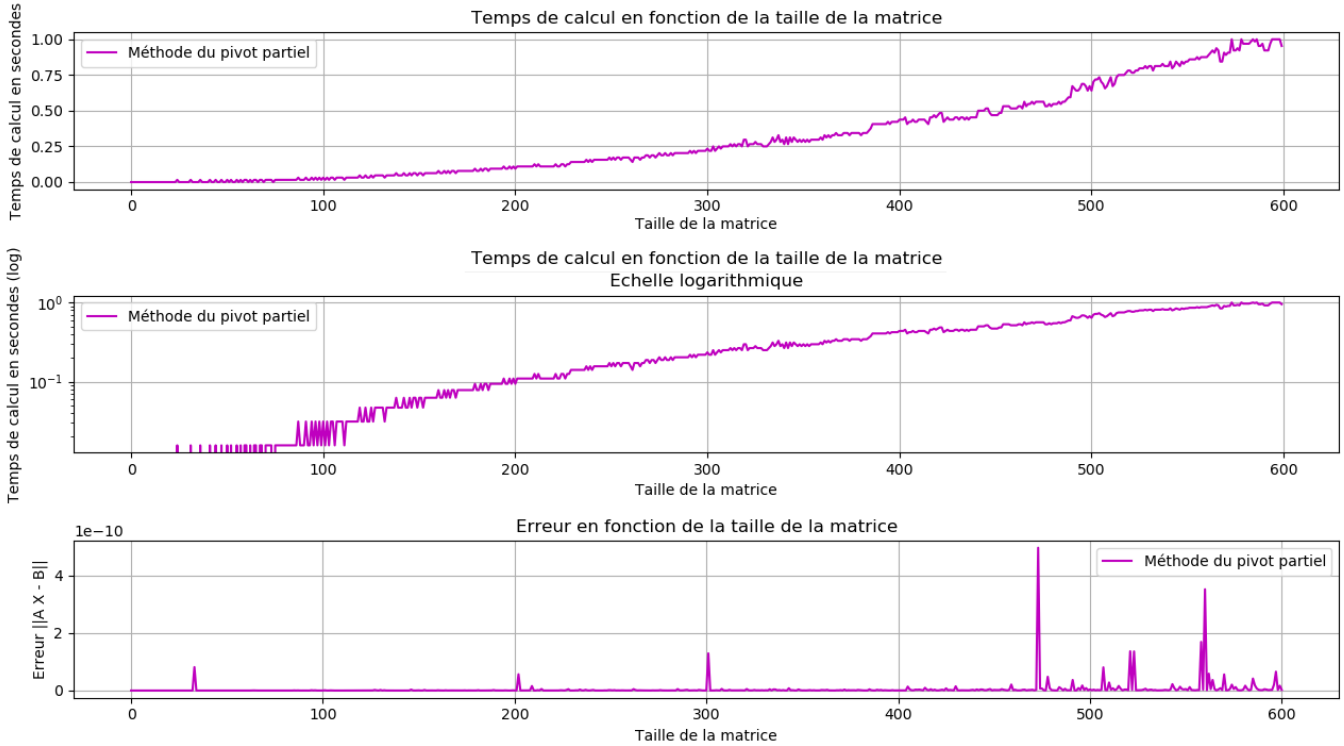

Figure 6 : Courbes obtenues pour la résolution de 600 matrices $n * n$ de taille comprise entre 1 et 600, par pas de 1
Méthode du pivot partiel | Ordinateur 2

On remarque que pour le premier appareil, le temps de calcul est très proche de la méthode de Gauss ; cependant pour l'ordinateur 2, le temps est beaucoup plus court. On peut également noter l'erreur plus faible, de l'ordre de 10^{-10} pour les deux appareils, aux pics maximums.

Question 2 : Pivot total

Taille de la matrice $(n * n)$	**Temps ordinateur 1 (en s)**	**Temps ordinateur 2 (en s)**
100	0.15625	0.171875
200	1.125	1.359375
300	3.890625	3.671875
Total	**273.125** **(Environ 4 minutes et 33 secondes)**	**290.890625** **(Environ 4 minutes et 51 secondes)**

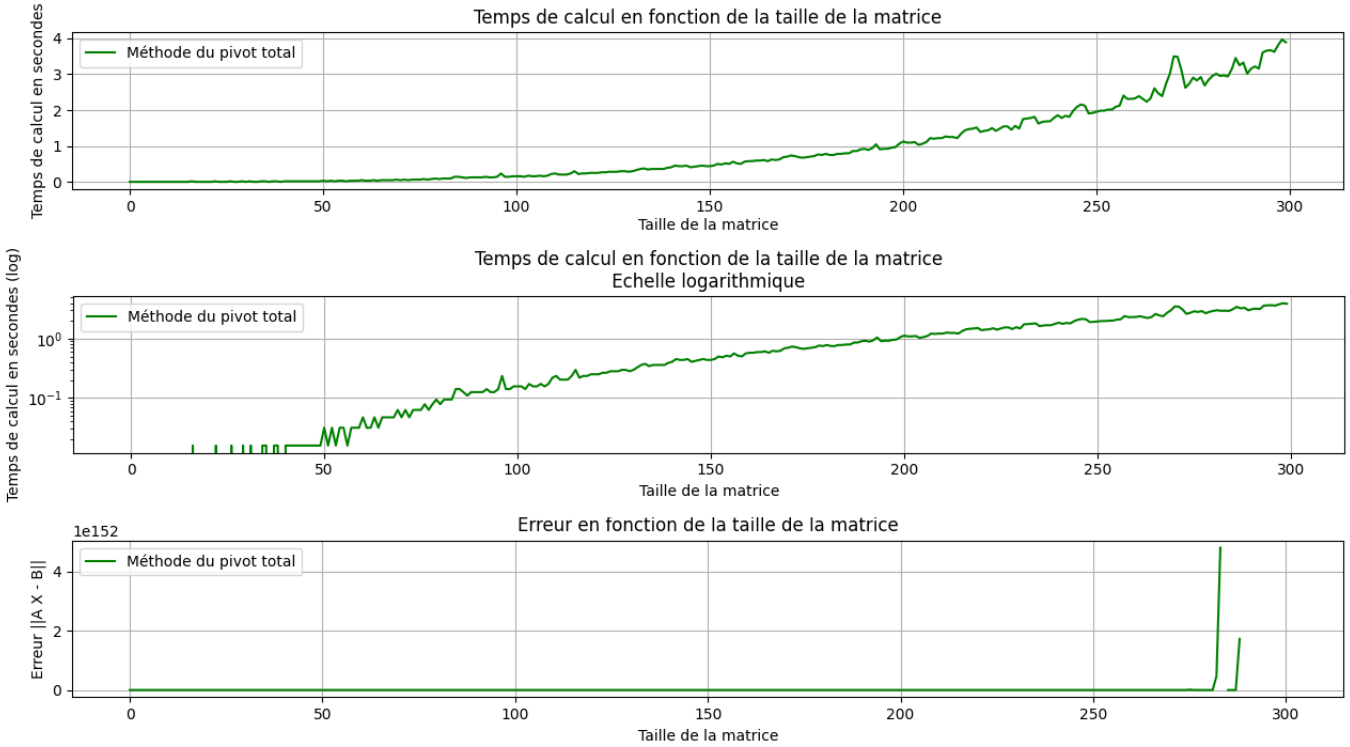

Figure 7 : Courbes obtenues pour la résolution de 600 matrices $n * n$ de taille comprise entre 1 et 600, par pas de 1
Méthode du pivot total | Ordinateur 1

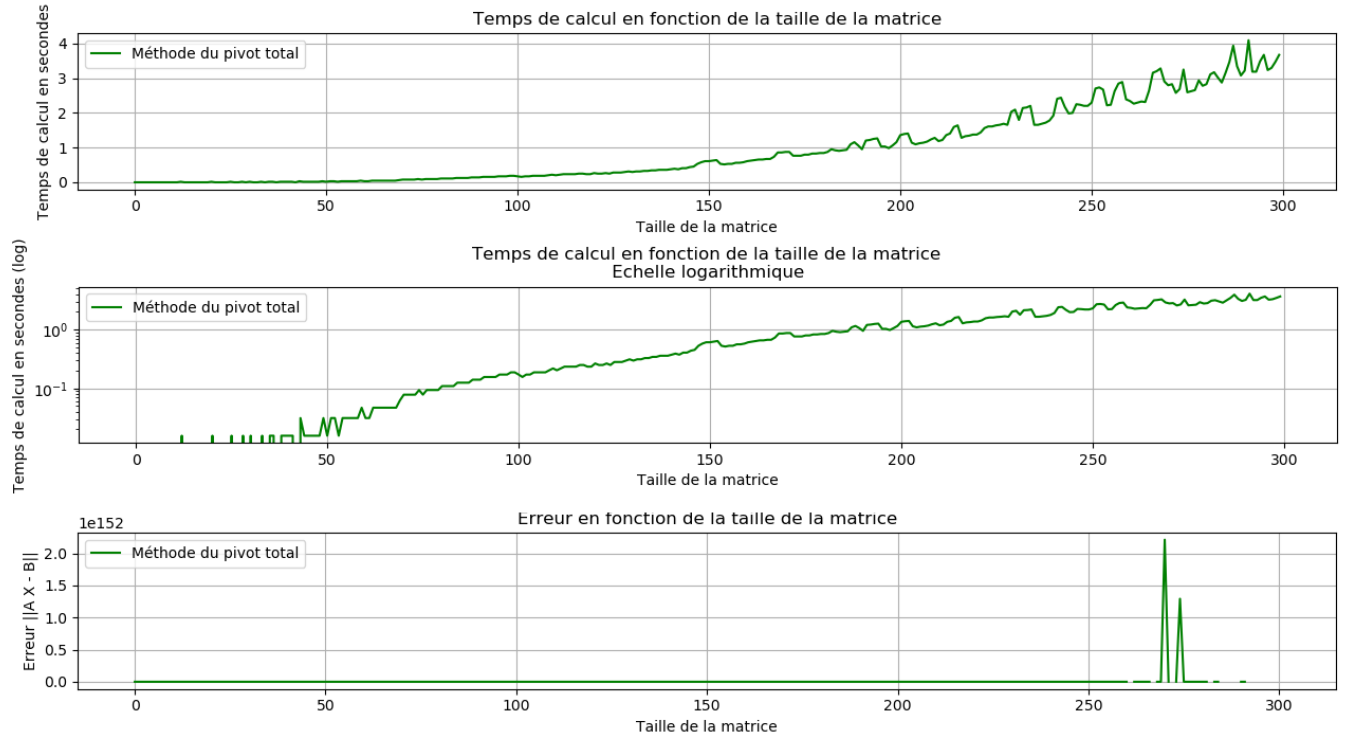

Figure 8 : Courbes obtenues pour la résolution de 600 matrices $n * n$ de taille comprise entre 1 et 600, par pas de 1
Méthode du pivot total | Ordinateur 2

Pour cette méthode, on peut dire sans trop de risque que notre programme contient sûrement une erreur, du fait des erreurs très élevées qui sont renvoyées par le programme, de l'ordre de 10^{152}… La méthode est également beaucoup plus longue, au niveau du temps de calcul, d'où le fait de n'avoir testé le programme que sur 300 matrices… Un test précédent à celui-ci nous a permis d'obtenir des résultats pour 600 matrices au terme de plus de 1h 15 de calculs. Cette donnée est donc à comparer avec les durées nécessaires aux autres méthodes pour ce même nombre de systèmes linéaires à résoudre.

Partie 4 : Solveur Python

Le solveur de la librairie `numpy` de Python permet de résoudre une matrice linéaire, ou un système scalaire linéaire. La méthode $numpy.linalg.solve(a, b)$ renvoie la solution « exacte » X de l'équation $A\ X = B$, une matrice de la même taille que la matrice B.[5]

En lançant le calcul pour 600 matrices, de taille $n * n$ allant de 1 à 600 avec un pas de 1, on obtient ces temps et graphiques :

Taille de la matrice $(n * n)$	Temps ordinateur 1 (en s)	Temps ordinateur 2 (en s)
100	0.0	0.0
200	0.0	0.0
300	0.0	0.0625
400	0.125	0.015625
500	0.015625	0.015625
600	0.015625	0.046875
Total	**12.453125**	**24.734375**

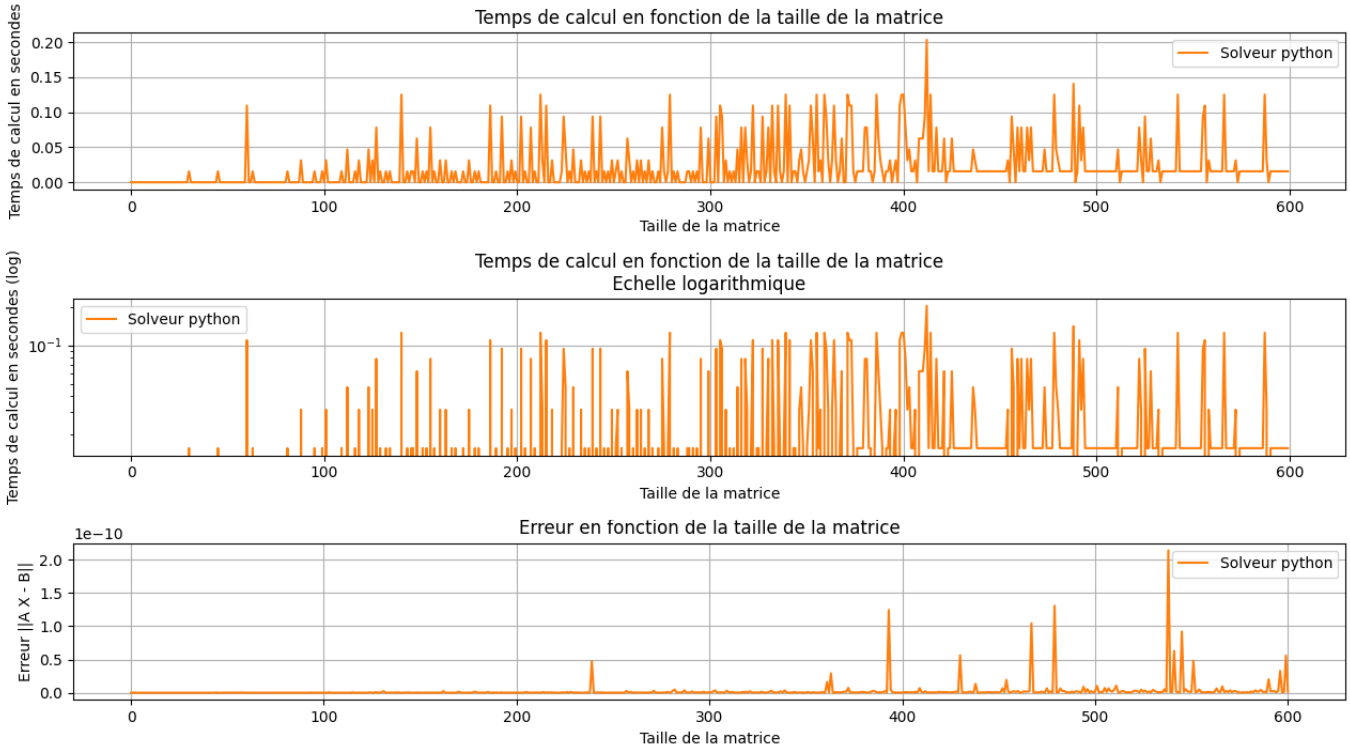

Figure 9 : Courbes obtenues pour la résolution de 600 matrices $n * n$ de taille comprise entre 1 et 600, par pas de 1
Solveur python | Ordinateur 1

[5] https://numpy.org/doc/stable/reference/generated/numpy.linalg.solve.html

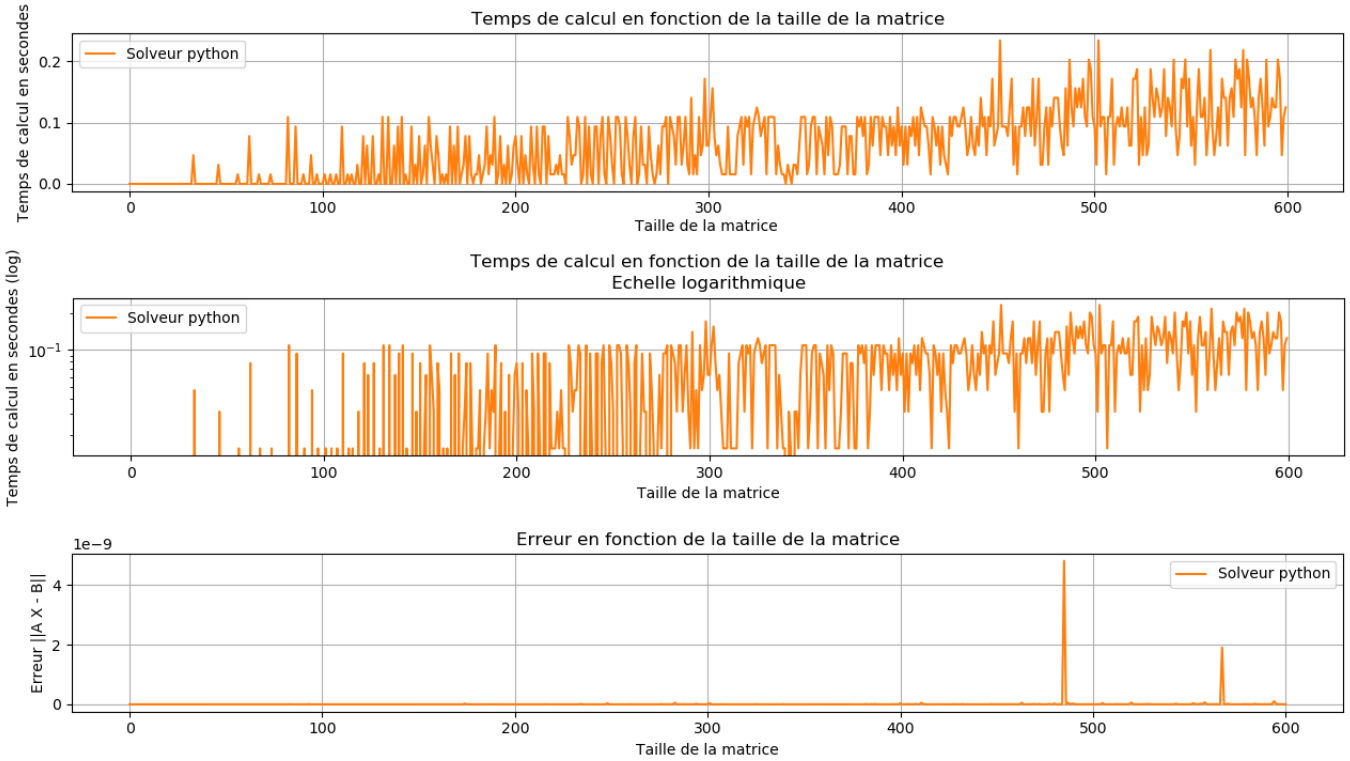

Figure 10 : Courbes obtenues pour la résolution de 600 matrices $n * n$ de taille comprise entre 1 et 600, par pas de 1
Solveur python | Ordinateur 2

On peut dire sans aucune hésitation que cette méthode est la plus rapide de toutes celles que nous essayons dans ce TP ! Que ce soit avec le premier ordinateur ou le second, la différence avec les autres méthodes est impressionnante. Rappelons que lors d'un de nos essais, nous avions pu calculer nos 600 matrices avec la méthode du pivot total au terme de plus d'une heure et 15 minutes de calcul. Ici, cela ne prend que 12 secondes…

L'erreur est également très faible, nous mesurons un pic maximal à $2.25 * 10^{-10}$ avec le premier ordinateur, et $5 * 10^{-9}$ avec le second.

Par curiosité, voici un test sur 5000 matrices avec le premier ordinateur :

Taille de la matrice $n * n$	**Temps de calcul en secondes**
500	0.109375
1000	0.21875
2000	0.8125
3000	2.0625
4000	5.578125
5000	8.46875

Analyse

Afin de pouvoir comparer nos cinq méthodes de résolution de systèmes linéaires, nous les testons sur la résolution de matrices $n * n$ allant de 1 à 500, par pas de 50.

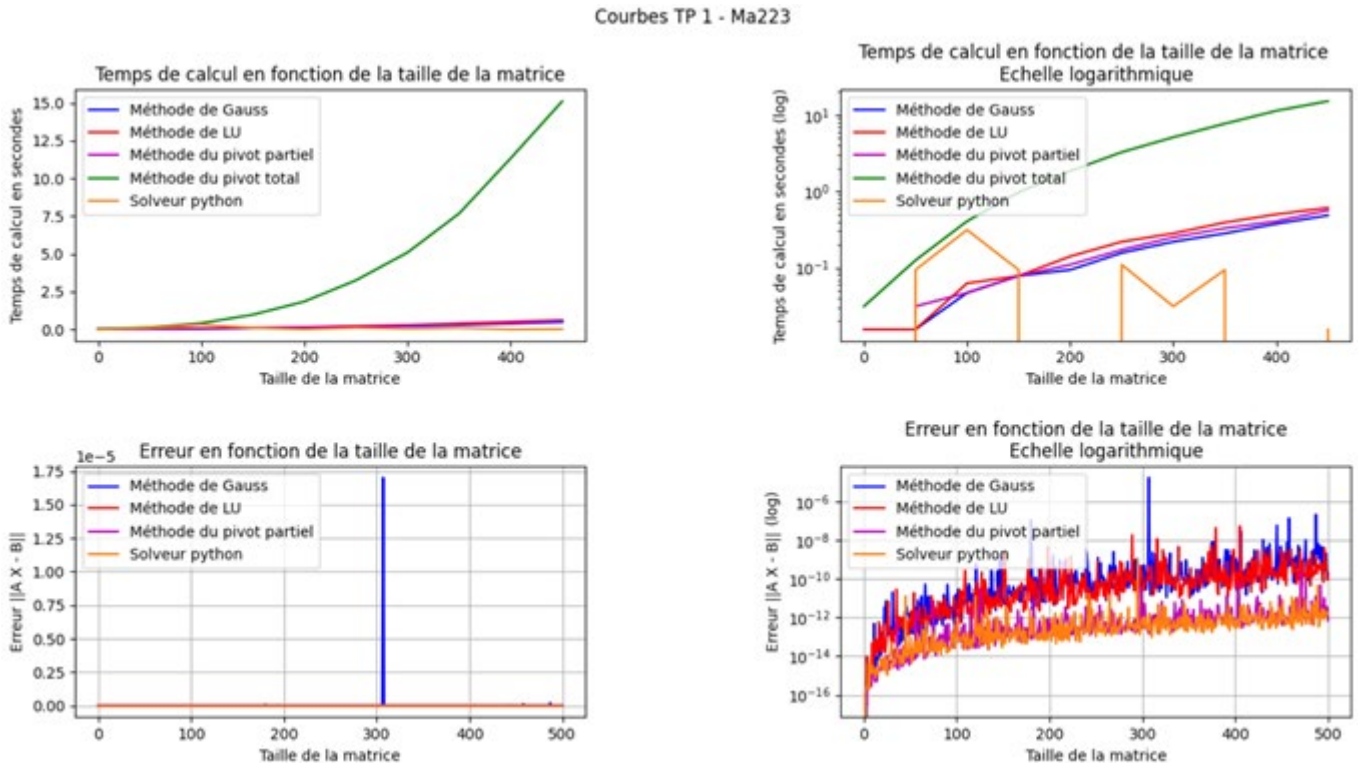

Figure 11 : Courbes obtenues pour la résolution de 500 matrices $n * n$ de taille comprise entre 1 et 500, par pas de 50
Toutes méthodes (sauf pivot total pour les erreurs) | Ordinateur 1

On peut agrandir la vue pour le graphique de l'erreur en fonction de la taille de la matrice, afin de mieux distinguer les courbes des méthodes de Gass, de la décomposition LU, du pivot partiel et du solveur Python :

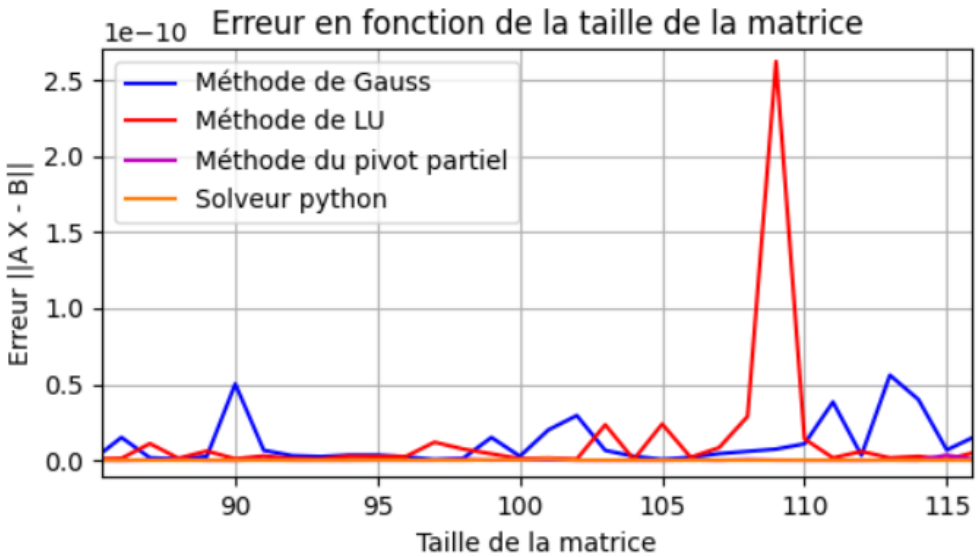

Figure 12 : Courbes des erreurs pour la résolution de 500 matrices $n * n$ de taille comprise entre 1 et 500, par pas de 50, agrandi
Toutes méthodes sauf pivot total | Ordinateur 1

La courbe violette (pivot partiel) reste toujours peu visible sur cette vue, elle est néanmoins présente, confondue dans les courbes orange et rouge. On peut la distinguer un peu sur l'extrême droite du graphique, au niveau de la graduation « 115 ».

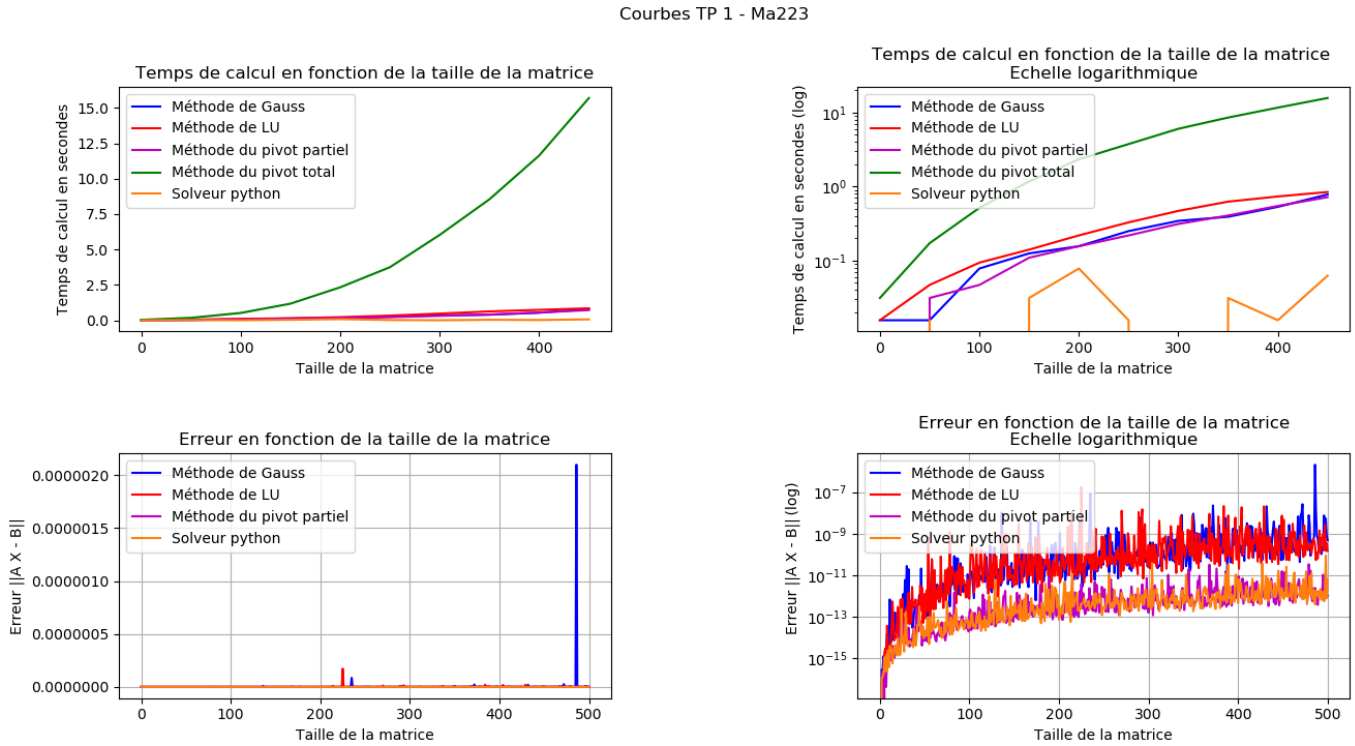

Figure 13 : Courbes obtenues pour la résolution de 500 matrices $n * n$ de taille comprise entre 1 et 500, par pas de 50
Toutes méthodes (sauf pivot total pour les erreurs) | Ordinateur 2

En analysant ces différentes courbes, on peut en tirer plusieurs remarques :

- Les temps de calculs des méthodes de Gauss, de la décomposition L U et du pivot partiel sont très proches.
- La méthode du solveur intégré à la librairie `numpy` de Python est très rapide.
- La méthode du pivot total est la plus longue, ce qui est normal puisque c'est la méthode nécessitant le plus d'opérations, et que nous avons très probablement une erreur dans notre programme, ce qui doit sûrement rallonger la durée de calcul nécessaire.
- Les méthodes du solveur Python et du pivot partiel semblent être les plus fiables face aux erreurs de calcul car ce sont les seules qui ne présentent pas de gros « pics ». A défaut de ces deux, on peut se contenter de la méthode de décomposition L U. La méthode de Gauss quant à elle est perfectible, et peut présenter des erreurs inférieures à 10^{-10} jusqu'à $1.75 * 10^{-5}$, ce qui est une amplitude très large.
- C'est cependant la méthode la plus rapide des 3 méthodes conventionnelles et fonctionnelles (Gauss, LU, pivot partiel).
- Les erreurs présentées par le solveur python sont irrégulières bien que très faibles, on peut notamment le voir grâce à l'échelle logarithmique du graphique haut droit. Cependant, ces erreurs sont les plus faibles que nous pouvons obtenir.

Voici les temps que nous avons obtenus pour les différentes méthodes :

- Ordinateur 1 :

Taille $n * n$ de la matrice	Gauss	LU	Pivot partiel	Pivot total	Solveur Python
100	0.046875	0.0625	0.046875	0.40625	0.3125
200	0.09375	0.140625	0.109375	1.84375	0.0
300	0.21875	0.28125	0.25	5.09375	0.03125
400	0.375	0.5	0.40625	11.328125	0.0
500	0.484375	0.609375	0.5625	15.109375	0.015625
Total	**1.765625**	**2.3125**	**1.984375**	**45.828125**	**0.75**

- Ordinateur 2 :

Taille $n * n$ de la matrice	Gauss	LU	Pivot partiel	Pivot total	Solveur Python
100	0.046875	0.0625	0.046875	0.515625	0.109375
200	0.140625	0.171875	0.15625	2.328125	0.0
300	0.265625	0.34375	0.265625	6.03125	0.0
400	0.4375	0.5625	0.46875	11.640625	0.0625
500	0.59375	0.71875	0.640625	15.703125	0.015625
Total	**2.15625**	**2.703125**	**2.265625**	**49.890625**	**0.265625**

Afin de mieux voir les temps de calcul, nous relançons le programme en cachant les courbes de la méthode du pivot total :

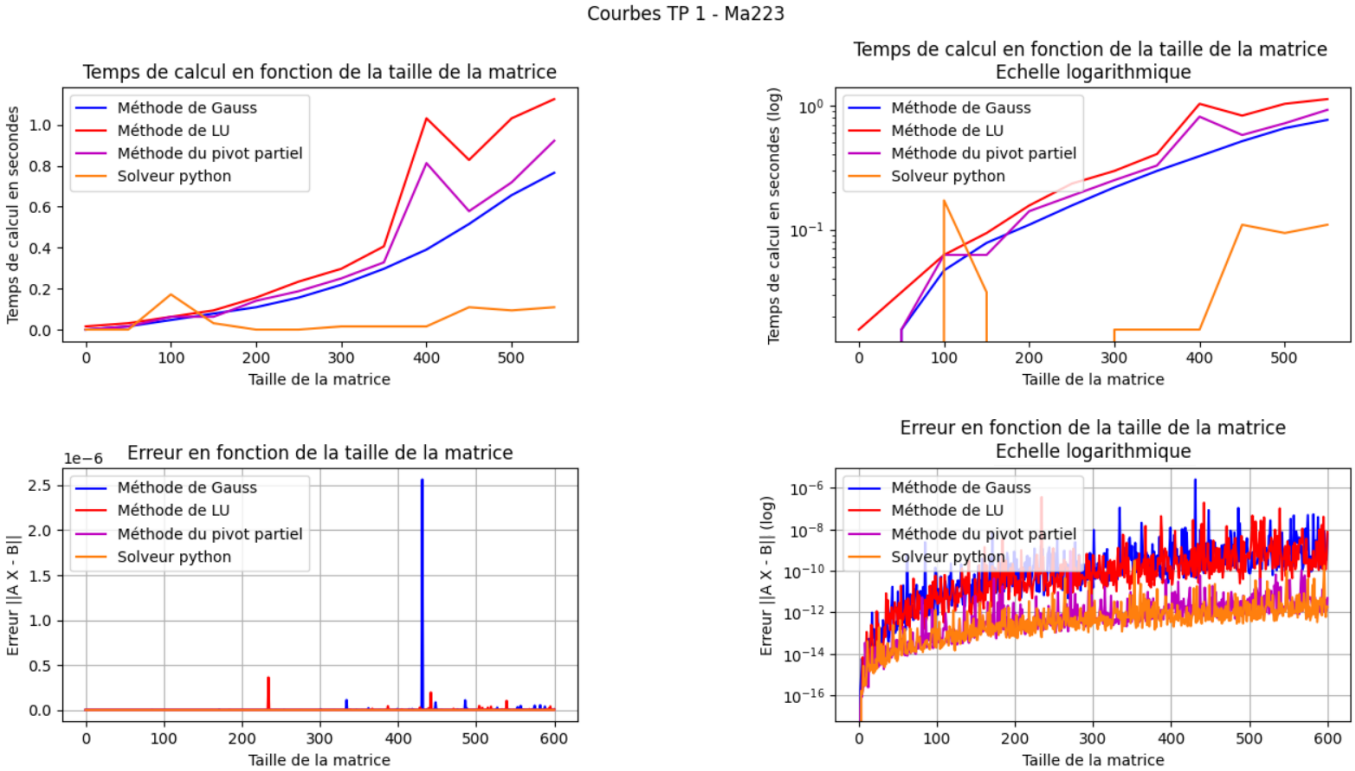

Figure 14 : Courbes obtenues pour la résolution de 500 matrices $n * n$ de taille comprise entre 1 et 500, par pas de 50
Toutes méthodes sauf pivot total | Ordinateur 1

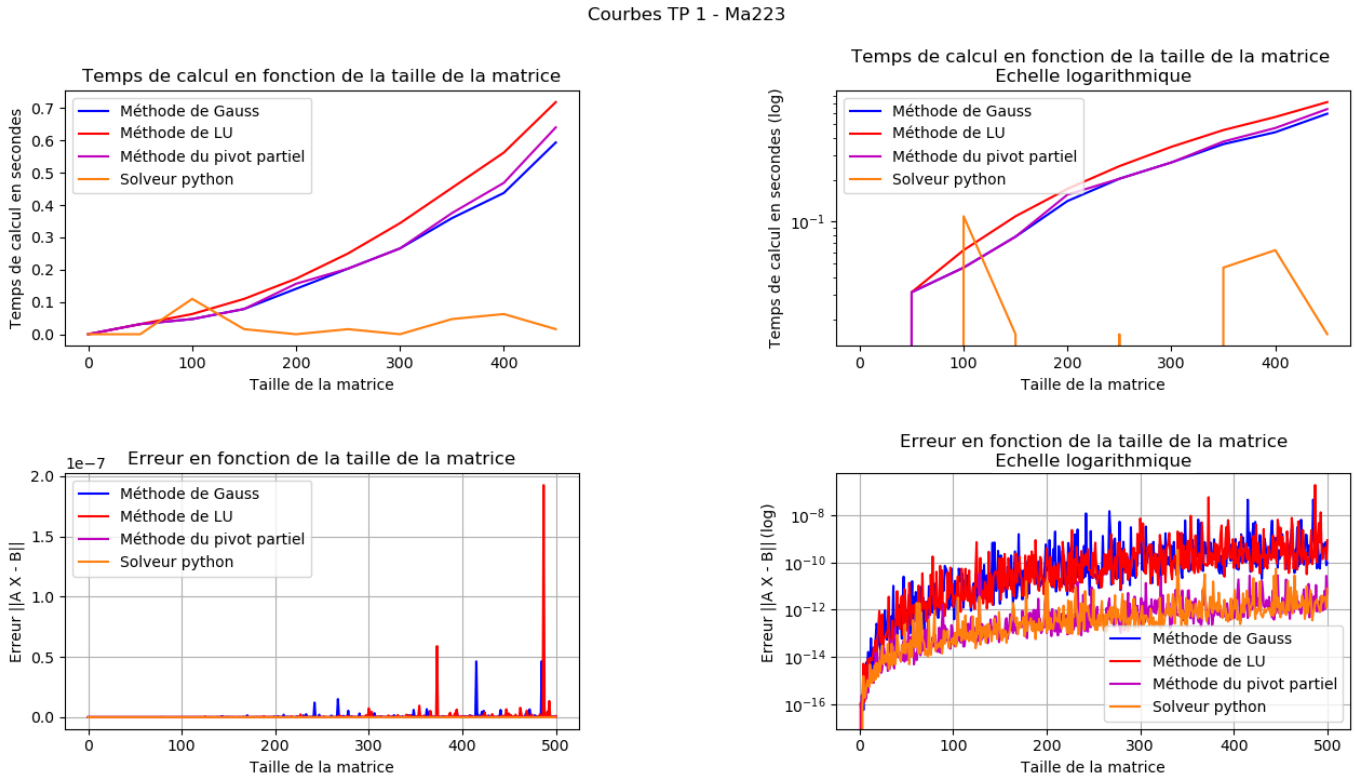

Figure 15 : Courbes obtenues pour la résolution de 500 matrices $n * n$ de taille comprise entre 1 et 500, par pas de 50
Toutes méthodes sauf pivot total | Ordinateur 2

On distingue à présent mieux les courbes de temps. On peut donc mieux expliciter l'ordre dans lequel les méthodes sont les plus efficaces :

Du plus au moins rapide	**Erreur la plus faible à la plus haute**
Solveur Python	Solveur Python
Méthode de Gauss	Méthode du pivot partiel
Méthode du pivot partiel	Méthode de décomposition L U
Méthode de décomposition L U	Méthode de Gauss

En comparant les deux appareils utilisés, malgré une haute différence de fréquence entre les deux processeurs des ordinateurs utilisés, on peut noter que les résultats sont semblables. On peut supposer que lors du passage de la 7ème à la 8ème génération d'Intel® Core™ i7, le fabriquant a amélioré la rapidité de calcul mais aussi la fiabilité : on peut voir qu'avec le premier ordinateur, les amplitudes des courbes sont moins grandes qu'avec le second ordinateur.

Mais qu'est-ce que la fréquence d'un processeur ? Lorsqu'on évoque la fréquence du processeur ou du CPU (Le CPU, *central processing unit*, est la traduction anglaise de UCT, *unité centrale de traitement*, c'est-à-dire le processeur), on parle de la vitesse du chargement des programmes et de leur bonne exécution. Le processeur d'un appareil est très sollicité lorsque l'utilisateur ouvre des programmes et ordonne des tâches.
Exprimée en gigahertz (GHz) à présent, la fréquence du processeur désigne le nombre d'opérations effectuées en une seconde par le processeur. Une horloge lui définit sa cadence. La fréquence d'horloge détermine le nombre de programmes, ou plutôt de cycles, que le processeur est capable d'exécuter chaque seconde. À l'intérieur de chaque cycle, on assiste à l'ouverture et à la fermeture de milliards de transistors.
Lorsque les processeurs étaient tous mono-cœurs (un seul cœur), la fréquence d'horloge était l'unique critère qui permettait de connaître la vitesse d'exécution du CPU. De nos jours, vitesse d'horloge n'est pas forcément synonyme d'exécution d'un plus grand nombre de cycles, notamment car les CPU ne sont plus mono-cœurs, mais

également car les processeurs avancés bénéficient d'une architecture de haute qualité qui leur permet de traiter les données plus efficacement que leurs prédécesseurs. [6]

Contrairement à ce que l'on pourrait donc penser, un ordinateur avec une fréquence plus élevée (ici l'ordinateur n°2) ne sera pas forcément plus rapide qu'un ordinateur donc la fréquence du CPU est plus basse (ordinateur n°1). Nous en avons la démonstration dans ce TP.

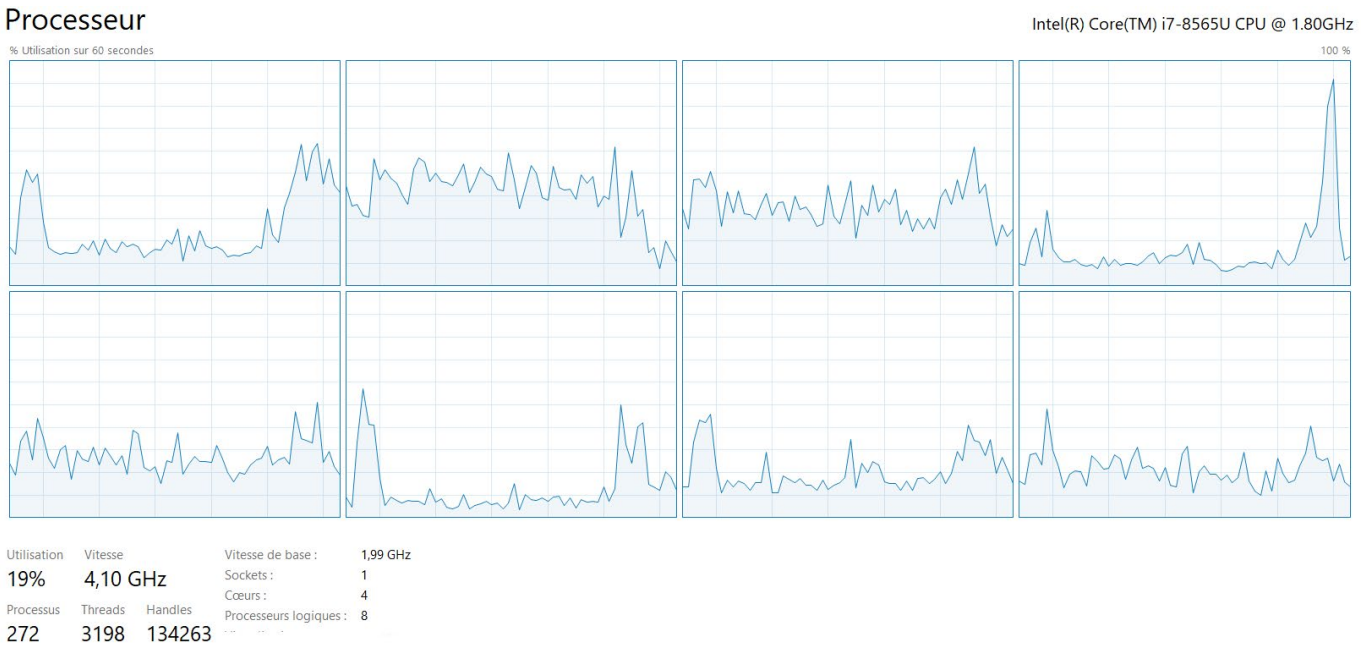

Figure 16 : Performances du processeur du premier ordinateur lors du lancement du programme permettant d'obtenir les figures 11 et 12

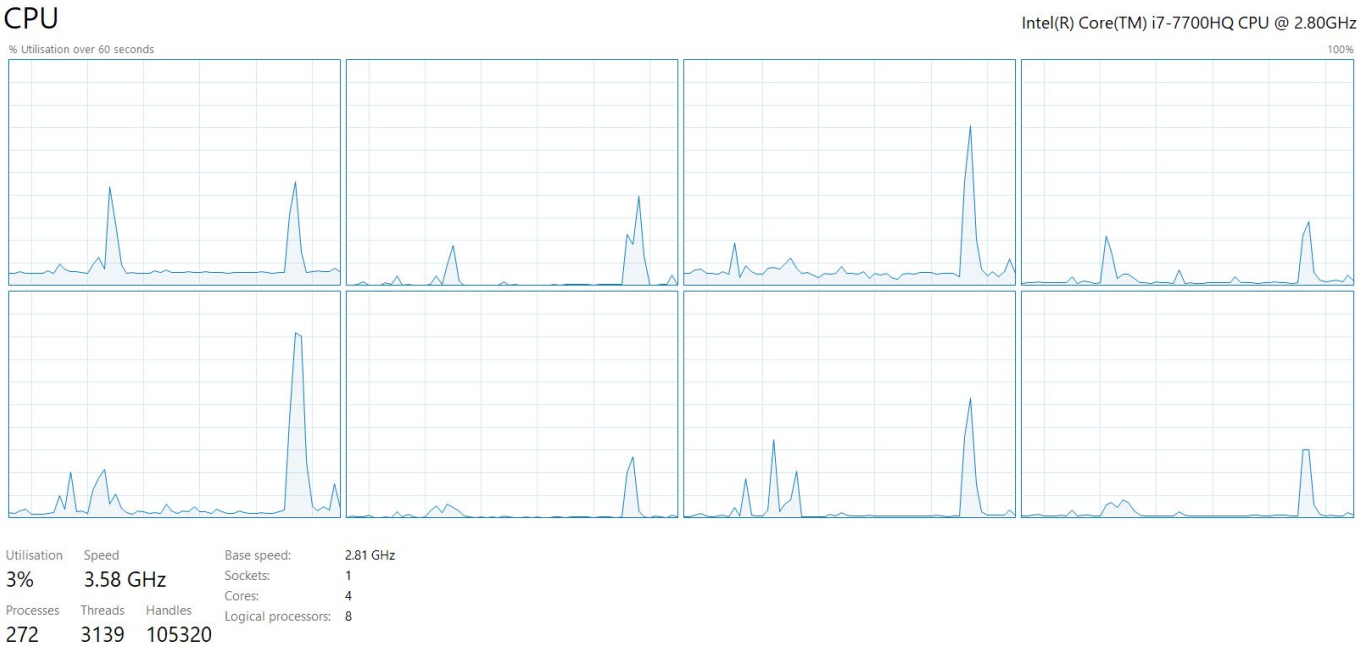

Figure 17 : Performances du processeur du second ordinateur lors du lancement du programme permettant d'obtenir la figure 13

Les deux ordinateurs utilisés possèdent chacun 4 cœurs et 8 processeurs logiques, mais pas à la même fréquence. On peut supposer que le processeur de 8ème génération a une fréquence plus basse peut-être car son architecture interne est de meilleure qualité, et que cela lui permet de pouvoir traiter les données plus efficacement que son prédécesseur de la 7ème génération.

[6] https://www.futura-sciences.com/tech/definitions/informatique-frequence-processeur-1895/

Sur le site de référence de comparaison de processeur *UserBenchmark*, on peut y lire que bien que la vitesse effective (*effective speed*) soit meilleure pour le second processeur, le score overclocké est meilleur pour le premier et c'est ceci qui fait la différence.[7]

[7] Plus de détails ici : https://cpu.userbenchmark.com/Compare/Intel-Core-i7-8565U-vs-Intel-Core-i7-7700HQ/m591977vsm211019

Conclusion

Si vous devez absolument éviter que votre fusée explose, mais que vous avez l'éternité pour faire les calculs, alors vous pouvez utiliser le solveur (et en plus ça sera rapide), ou à défaut la méthode du pivot partiel.

Si vous devez éviter qu'une bombe explose mais que vous n'avez pas besoin d'un résultat frôlant la perfection, il vaut mieux utiliser le solveur ou bien la méthode de Gauss.

Pour un bon compromis entre les deux critères, on préférera au maximum la méthode du solveur de la librairie `numpy` de Python, ou si c'est impossible la méthode du pivot partiel.