MINI-PROJET B :

Intégration numérique

Auteurs :

Augustin MONTREDON
Baptiste CORN

Enseignants :

Mme. Sanjosé
M. Tabiai

– Rapport de recherche –

Nous attestons que ce travail est original, qu'il est le fruit de notre travail et qu'il a été rédigé de manière autonome.

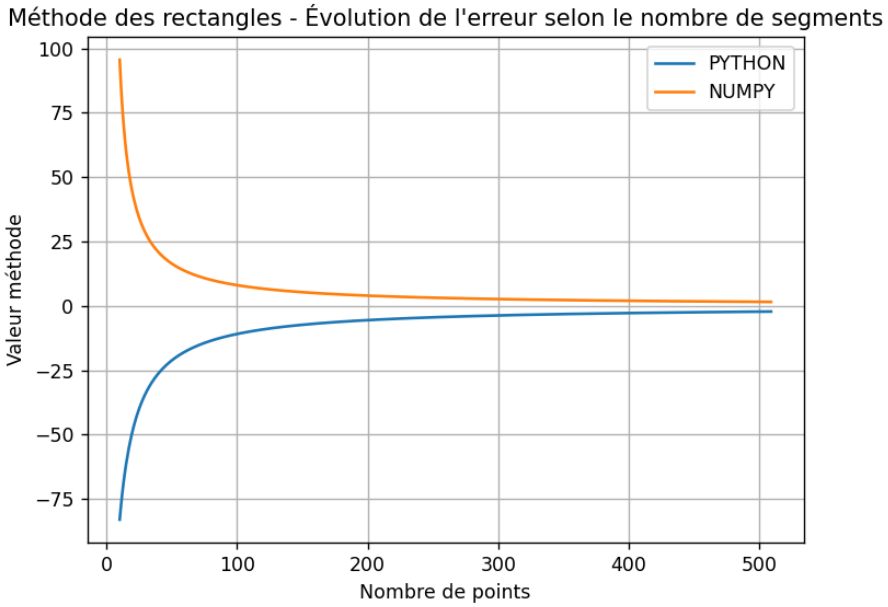
Montréal, le 11/06/2024

Introduction

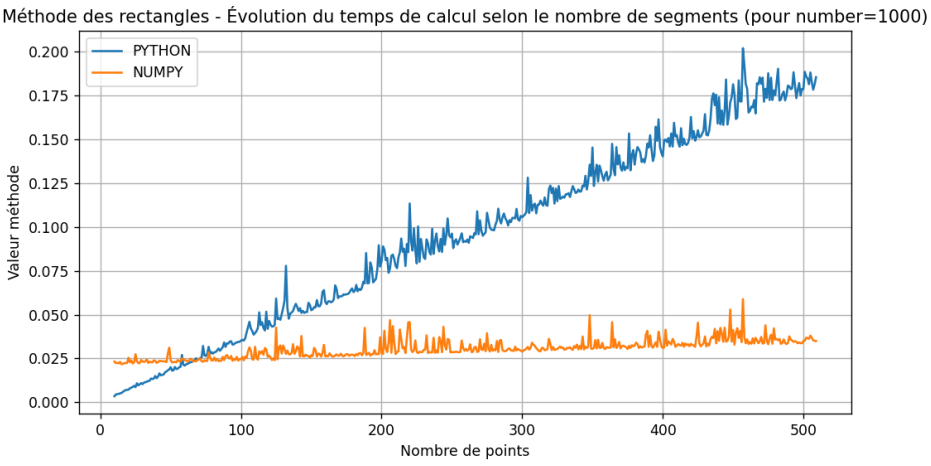
L'objectif de l'exercice est de démontrer la performance de la bibliothèque NumPy pour le calcul numérique. On va calculer l'aire sous la courbe I d'une fonction polynomiale du 3e ordre f à l'aide de plusieurs méthodes implémentées en python seulement et avec NumPy. Voici le polynôme que nous allons manipuler tout au long de l'exercice, les courbes comportent 500 points et commencent à partir de 15. Si on commence à 0, le début des courbes divergent donc nous avons choisi de commencer arbitrairement à 15 par soucis de lisibilité. (1)

$$I = \int_{-2}^{3} f(x)dx \ avec : \ f(x) = 1 + 2 * x + 3 * x^2 + 4 * x^3$$

1. Intégration par la méthode des rectangles

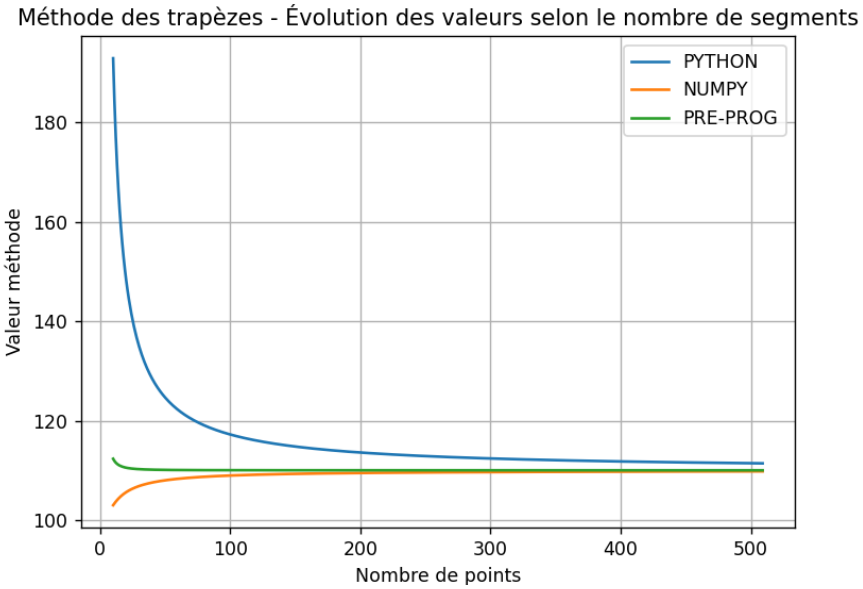


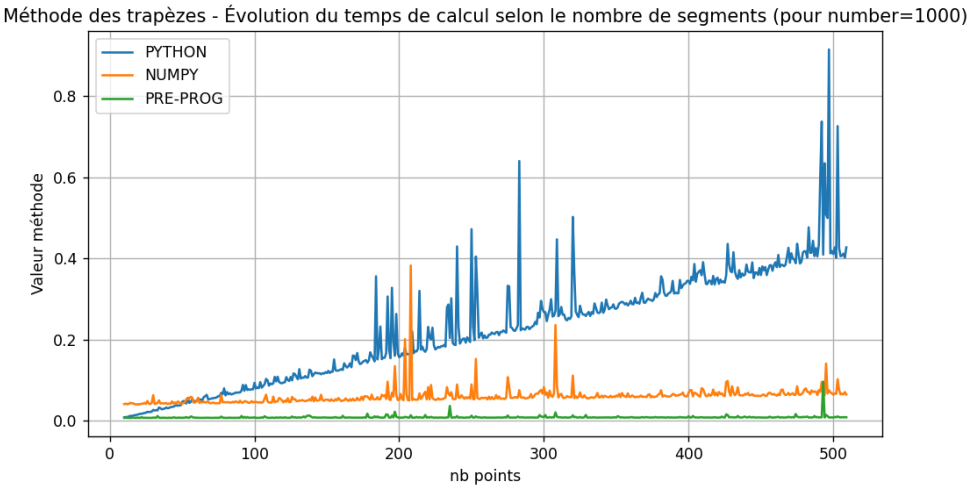
Sur ce graphique représentant l'évolution de l'erreur, les courbes convergent vers une valeur proche de 0 après 300 points. Python part d'une valeur négative car une partie de notre segment l'est aussi.



On observe que le temps de calcul reste linéaire pour Numpy car il s'agit simplement de parcourir l'objet array dont la taille est égale au nombre de points saisis. Pour Python, étant donné qu'il parcourt une liste, plus cette dernière est longue et plus le temps d'exécution augmente pour le calcul de chaque élément.

2. Comparaison avec la méthode des trapèzes

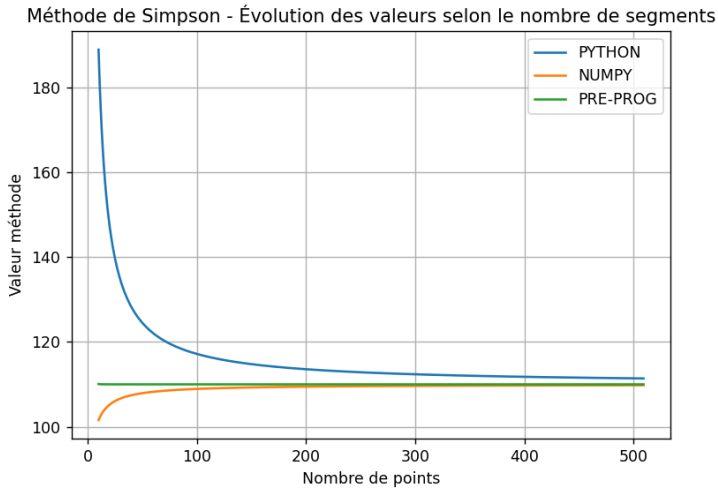


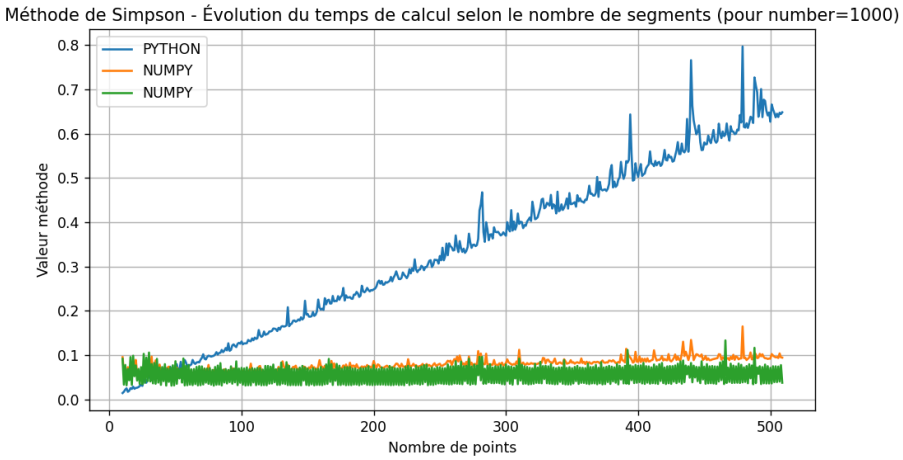


Pour la méthode des trapèzes, on observe que python prend beaucoup plus de points pour parvenir au résultat par rapport à Numpy. Python a toujours besoin d'environ 300 points pour converger tandis que Numpy est beaucoup plus efficace. La méthode préprogrammée numpy.trapz est quand à elle la plus efficace des 3 en convergent quasi-instantanément.

Les courbes du temps de calcul ont la même allure que pour la méthode des rectangles mais les valeurs sont plus élevées. Cela peut s'expliquer par l'opération plus lourde pour calculer une section en trapèze qu'une simple section rectangulaire. Numpy et la fonction préprogrammée ont un comportment similaire mais la méthode de la librairie reste plus efficace.

3. Comparaison avec la méthode de Simpson





La méthode de Simpson offre un comportement de courbe similaire à la méthode des trapèzes. Python converge beaucoup plus tard que Numpy et que la méthode de Simpson préprogrammée de la libraire Scipy. Le temps d'exécutions comporte aussi de manière similaire, Python met de plus en plus de temps à calculer chaque élément tandis que Numpy (orange) et Scipy (vert) restent linéaires. Il faut quand même préciser que la méthode préprogrammée de Scipy est plus optimisé que nos array Numpy.

Conclusion :

Au cours de cet exercice, nous avons pu observer l'évolution du temps de calcul pour différentes méthodes de programmation pour des cas d'intégration numériques. Python n'est pas optimisé pour gérer des tracés de courbes avec un nombre important de points tandis que Numpy est très efficace avec un temps d'exécution augmentant presque linéairement avec un nombre important de point. Restant ainsi presque constant pour chaque élement de l'array à calculer. Contrairement à Python qui prend de plus en plus de temps à calculer chaque élément de la liste tout en étant moins précis car demandant plus de points pour avoir une valeur qui converge.

Au niveau de la programmation, nous aurions pu améliorer un point en particulier au niveau de l'architecture du code. Créer une fonction plot() pour s'occuper de l'affichage graphique aurait été préférable par rapport à tout afficher dans le main. Cela alourdit la lecture du fichier.