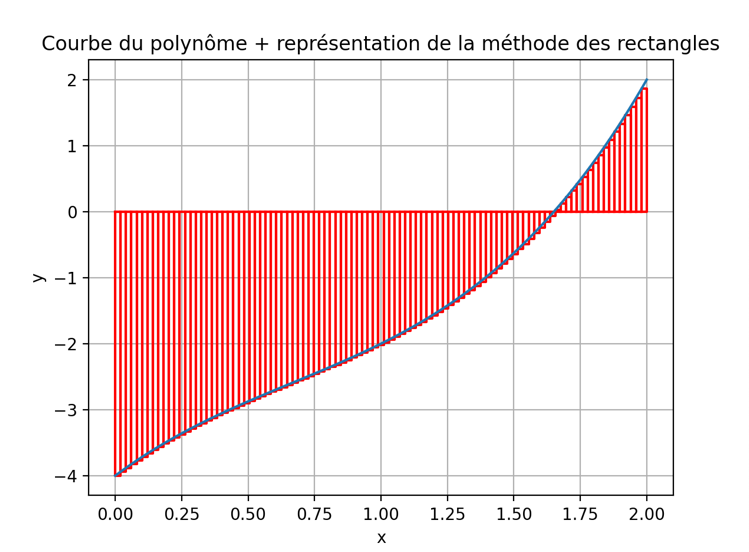
Aymen Mtar - Jhon Alejandro Rondon Gamboa - Bénédicte Turbant

**Comparaison et évaluation la performance et précision des différentes méthodes**

Pour chacune des trois méthodes (rectangles, trapèzes et Simpson) nous avons implémenté une méthode en Python basique et une méthode en Pyton NumPy. De plus, pour les méthodes trapèzes et Simpson nous avons implémenté les méthodes pré-programmée dans les paquets Python ( np.trapz et scipy.integrale.simpson ).



*Méthode des rectangles Méthode des trapèzes*

***Tableau de comparaison des différentes méthodes :***

|  |  |  |
| --- | --- | --- |
|  | Méthode des rectangles | |
|  | Python | NumPy |
| Description | Cette méthode implémente manuellement la méthode des rectangles, en divisant l’intervalle en segments et en calculant l’aire des rectangles. | Cette méthode implémente la méthode des rectangles à l’aide des tableau NumPy pour calculer l’aire des différents segments. |
| Avantages | Cette implémentation est simplement compréhensible le code. | NumPy est optimisé pour les opérations grands tableaux, ce qui le rend beaucoup plus rapide que les boucles de forme manuelle. |
| Inconvénients | Cette méthode est plus lente | Cette méthode est plus rapide car elle utilise le module NumPy |
| Runtime | C’est la méthode la plus lente car elle utilise des listes classiques et une boucle | C’est la méthode la plus rapide, elle est optimisée en utilisant les tableaux NumPy |

|  |  |  |  |
| --- | --- | --- | --- |
|  | Méthode des trapèzes | | |
|  | Python | NumPy | np.trapz |
| Description |  |  | cette méthode utilise numpy.trapz pour l'intégration, qui implémente la règle du trapèze, à des fins de comparaison et de référence. |
| Avantages |  |  | NumPy est optimisé pour les opérations grands tableaux, ce qui le rend beaucoup plus rapide que les boucles de forme manuelle. |
| Inconvénients |  |  | Il peut y avoir de légères différences de précision par rapport à une implémentation précise de Simpson car il n'utilise pas spécifiquement la règle de Simpson mais la règle du trapèze. |
| Runtime |  |  | Il est généralement plus rapide que la méthode manuelle de Simpson en raison des optimisations de NumPy. |

|  |  |  |  |
| --- | --- | --- | --- |
|  | Méthode Simpson | | |
|  | Python | NumPy | SciPy |
| Description | La méthode implémente manuellement la règle de Simpson, en divisant l'intervalle d'intégration en sous-intervalles et en appliquant la formule pour chaque paire de sous-intervalles |  | La méthode utilise scipy.integrate.simpson, qui est une implémentation optimisée de la règle Simpson fournie par SciPy. |
| Avantages | C’est une implémentation manuelle qui permet un contrôle détaillé du calcul et peut être un bon exemple pour comprendre le fonctionnement |  | NumPy est optimisé pour les opérations grands tableaux, ce qui le rend beaucoup plus rapide que les boucles de forme manuelle |
| Inconvénients | Peut-être plus lent que les implémentations optimisées dans les bibliothèques scientifiques en raison du manque d'optimisations |  | Il peut y avoir de légères différences de précision par rapport à une implémentation précise de Simpson car il n'utilise pas spécifiquement la règle de Simpson mais la règle du trapèze. |
| Runtime | C'est généralement la plus lente des trois méthodes en raison de l’implémentation de calcul manuelle. |  | Il est généralement plus rapide que la méthode manuelle de Simpson en raison des optimisations de NumPy. |

***Graphique de comparaison des temps d’exécution de la méthodes Simpson :***

Une image contenant texte, capture d’écran, diagramme, Tracé

Description générée automatiquement**Observations :**

Selon la théorie des avantages de chaque méthode, la précision du résultat peut varier de petites quantités à haute résolution. En ce qui concerne les temps d'exécution, on peut observer que la méthode ScyPy est la plus retardée, en cours d'investigation, cela peut être dû aux raisons suivantes :

1. Vitesse du processeur :
2. Architecture du processeur
3. Caché de la CPU
4. Nombre de cœurs
5. Système d’exploitation
6. Une image contenant texte, capture d’écran, Police, information

   Description générée automatiquementGestion de l'énergie

De plus, le calcul de l'aire d'une équation du troisième degré n'est pas une opération si complexe, le temps de chargement de la bibliothèque ScyPy est ajouté au temps mesuré par la fonction timeit.

***Une image contenant texte, capture d’écran, ligne, Tracé

Description générée automatiquementGraphique du temps de calcul en fonction du nombre de segments :***

***Observations :***

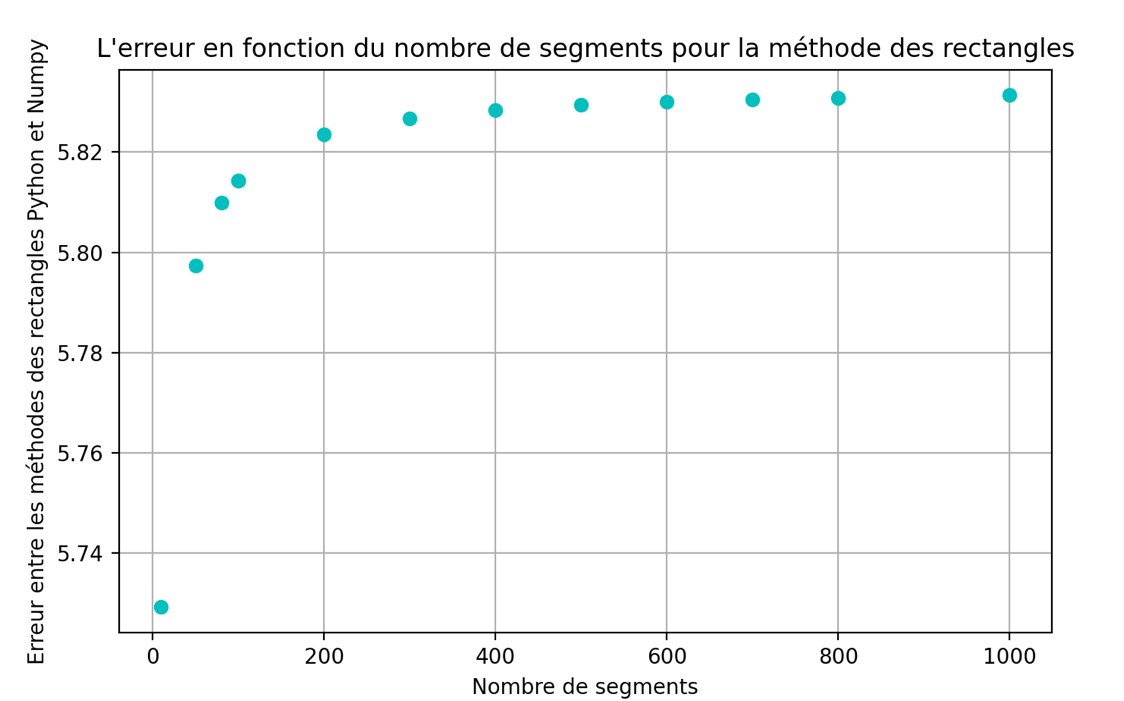
Le temps d’exécution est impacté par le nombre de segments. Plus celui-ci est grand, plus le temps d’exécution est long mais le résultat est bien plus précis.

***Graphique de l’aire sous la courbe en fonction du nombre de segments :***

Une image contenant texte, ligne, Tracé, capture d’écran

Description générée automatiquement***Observations :***

On remarque que pour chaque méthode (ici la méthode des rectangles) une convergence de l’aire sous la courbe plus le nombre de segments utilisés. En effet, plus le nombre de segments est élevé plus l’aire des rectangles est précise, réduisant l’erreur.

***Graphique de l’erreur en fonction du nombre de segments :***

***Observations :***

On remarque la aussi que plus le nombre de segments augmente plus l’erreur se stabilise.

**Références bibliographiques :**

([SciPy Documentation] (<https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.simpson.html)>)

([NumPy Documentation] (<https://numpy.org/doc/stable/reference/generated/numpy.trapz.html)>)