Rapport de mini-projet B

Équipe B
SIMONEAU Zacharie, SIMZ65050005
VANIER Charles-Emmanuel, VANC16059803

À remettre le 13 mars 2025 à
Ilyass Tabiai

Cours : MGA802 – Introduction à la programmation en python

1. Fonction analysée

Afin d'évaluer nos méthodes d'intégration, nous allons utiliser la fonction suivante :

$$y = -2.5 \cdot x^3 + 3 \cdot x^2 + 7 \cdot x - 5$$

Cette fonction a la solution analytique suivante :

$$y = -0.625 \cdot x^4 + x^3 + 3.5 \cdot x^2 - 5 \cdot x + c$$

Cette solution offre une valeur de 200 si on intègre de x = –5 à x = 5.

2. Intégration par la méthode des rectangles

2.1. Python régulier

Nombre de segments	Solution analytique	Intégration par rectangles	Pourcentage d'erreur	Temps d'exécution
10	200	197.5	1.25 %	0.115598
100	200	199.975	0.0125 %	0.197979
1000	200	199.99975	0.000125 %	0.399385

Nous pouvons remarquer que si l'on augmente le nombre de rectangles par 10, on obtient 100 x moins d'erreur, mais le temps d'exécution devient beaucoup plus long.

Nous pouvons observer la convergence de cette méthode d'intégration sur les graphiques suivant :

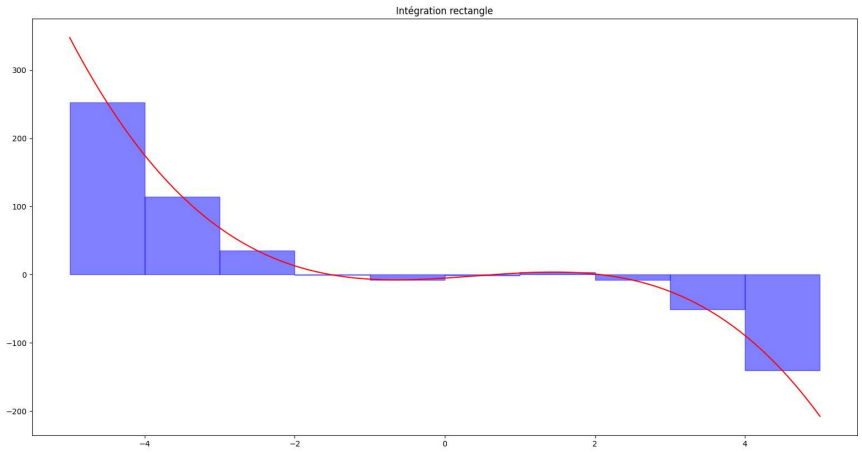

Figure 1: Intégration par la méthode des rectangles implémentés avec du python pure (n=10)

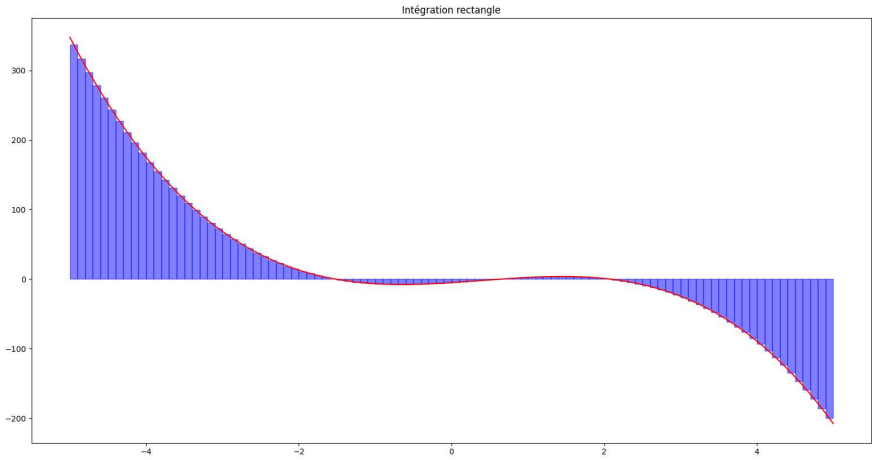

Figure 2: Intégration par la méthode des rectangles implémentés avec du python pure (n=100)

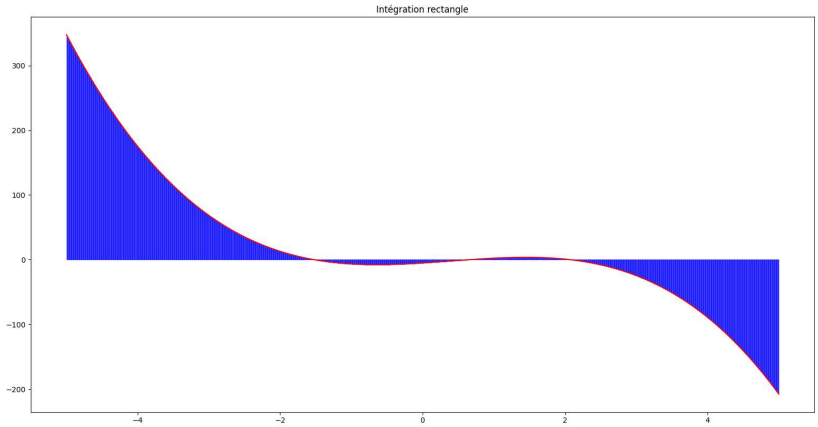

Figure 3: Intégration par la méthode des rectangles implémentés avec du python pure (n=1000)

2.2. Numpy

Nombre de segments	Solution analytique	Intégration par rectangles	Pourcentage d'erreur	Temps d'exécution
10	200	197.5	1.25 %	0.003589
100	200	199.975	0.0125 %	0.005156
1000	200	199.99975	0.000125 %	0.034144

Nous obtenons les mêmes valeurs que celles qui ont été obtenues par python régulier.

Le temps d'exécution, cependant, est beaucoup plus rapide qu'avec l'implémentation dans le python seulement.

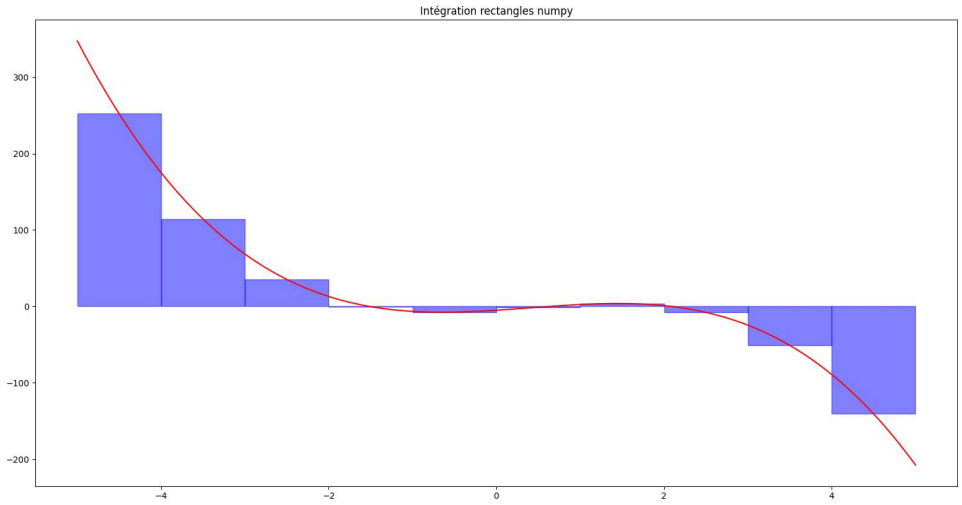

Figure 4: Intégration par la méthode des rectangles implémentés avec le module numpy (n=10)

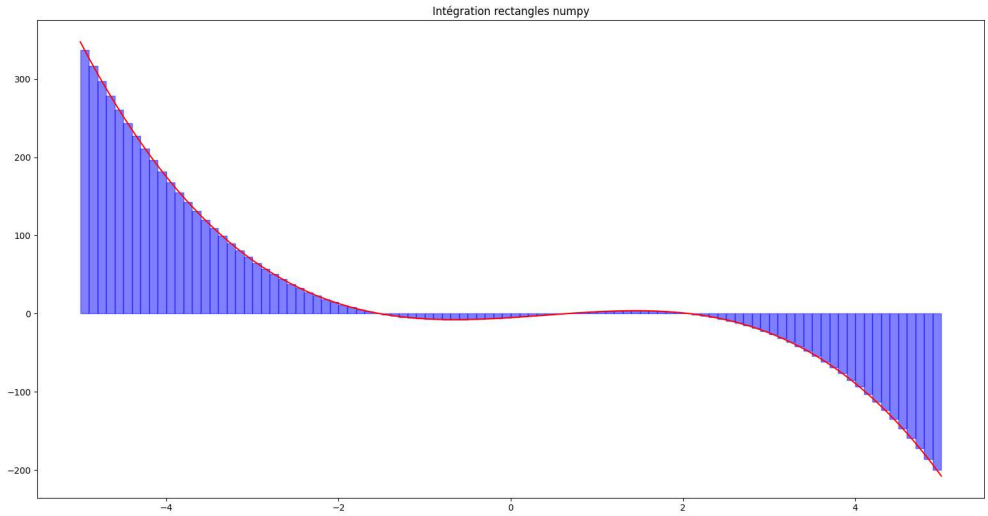

Figure 5: Intégration par la méthode des rectangles implémentés avec le module numpy (n=100)

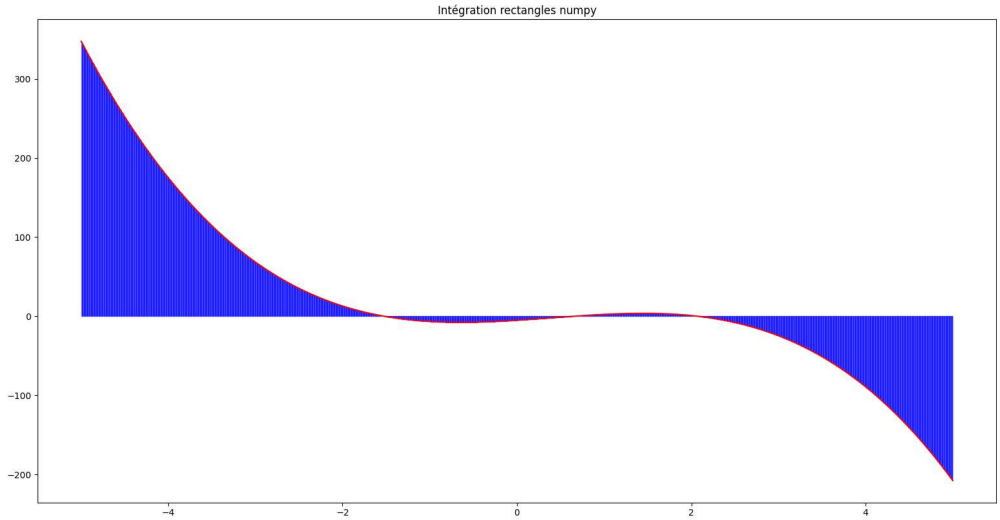

Figure 6: Intégration par la méthode des rectangles implémentés avec le module numpy (n=1000)

3. Intégration par la méthode des trapèzes

3.1. Python régulier

Nombre de segments	Solution analytique	Intégration par trapèzes	Pourcentage d'erreur	Temps d'exécution
10	200	205	2.5 %	0.002356
100	200	200.05	0.025 %	0.032153
1000	200	200.0005	0.00025 %	0.304699

Nous pouvons remarquer que si l'on augmente le nombre de rectangles par 10, on obtient encore une fois 100 x moins d'erreur. Contrairement à ce que l'on pourrait penser, la méthode des trapèzes est moins précise que la méthode des rectangles pour ce cas-ci. De plus, les temps d'exécution sont semblables.

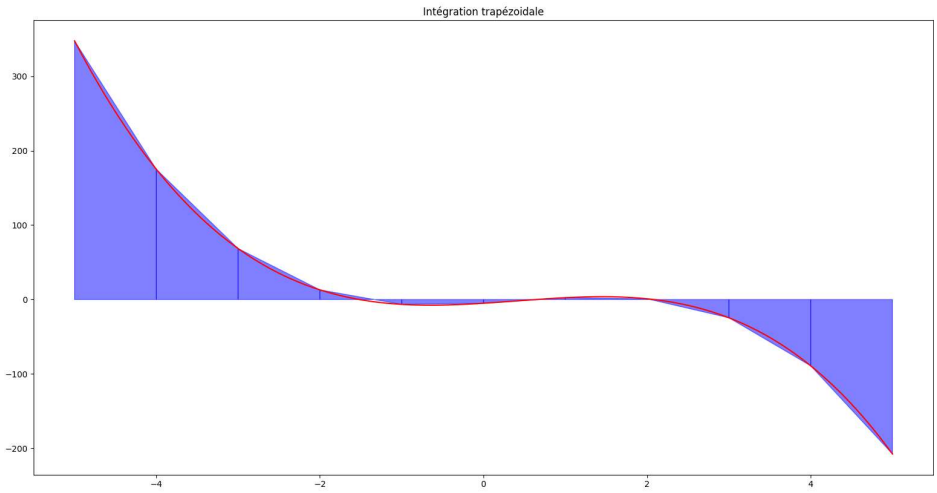

Figure 7: Intégration par la méthode des trapèzes implémentés avec du python pure (n=10)

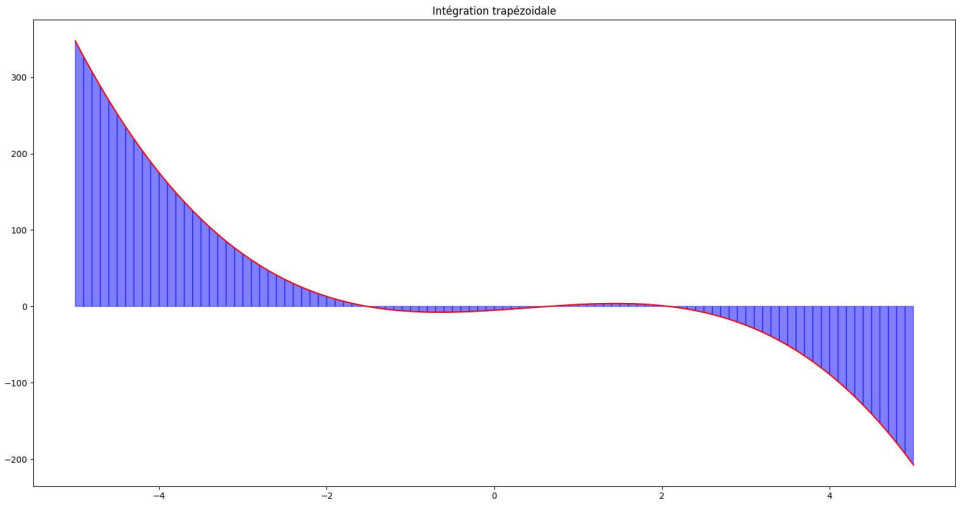

Figure 8: Intégration par la méthode des trapèzes implémentés avec du python pure (n=100)

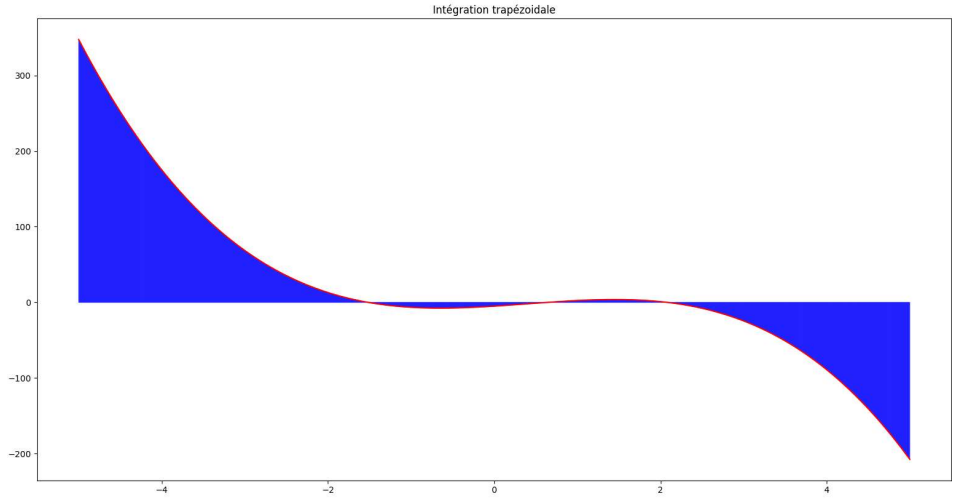

Figure 9: Intégration par la méthode des trapèzes implémentés avec du python pure (n=1000)

3.2. Numpy

Nombre de segments	Solution analytique	Intégration par trapèzes	Pourcentage d'erreur	Temps d'exécution
10	200	205	2.5 %	0.005443
100	200	200.05	0.025 %	0.009902
1000	200	200.0005	0.00025 %	0.045022

Nous obtenons les mêmes valeurs que celles qui ont été obtenues par python régulier. Le temps d'exécution, cependant, est beaucoup plus rapide.

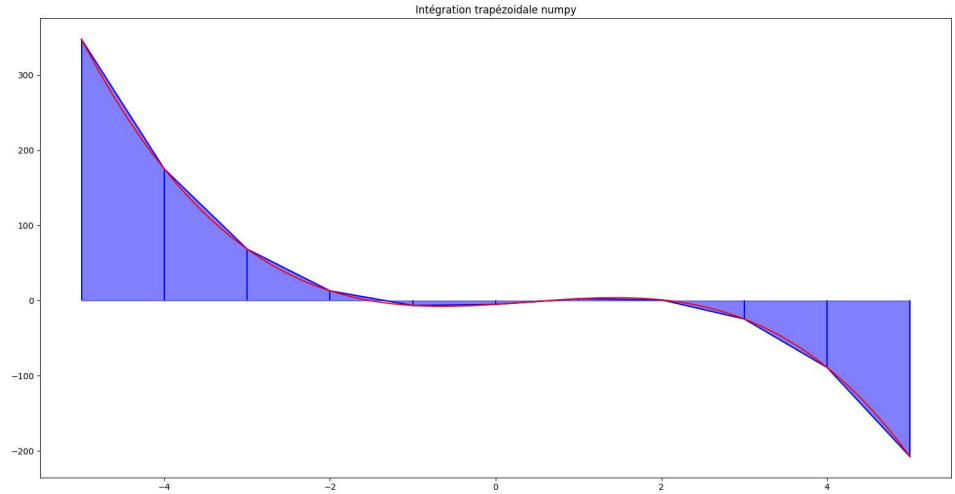

Figure 10: Intégration par la méthode des trapèzes implémentés avec le module numpy (n=10)

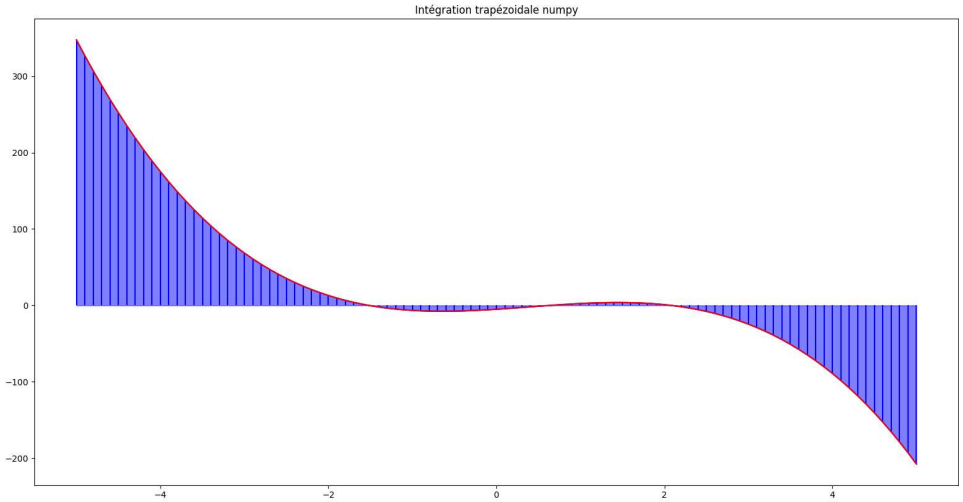

Figure 11: Intégration par la méthode des trapèzes implémentés avec le module numpy (n=100)

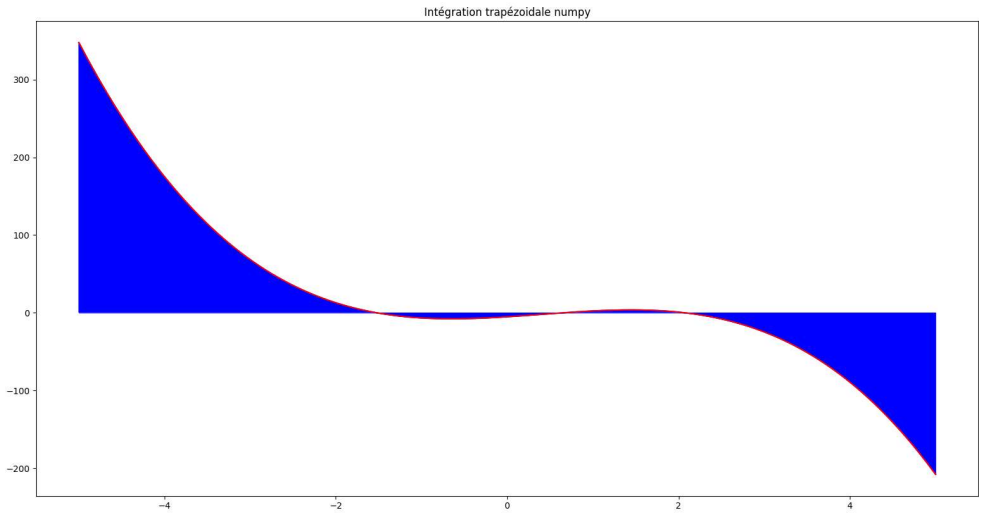

Figure 12: Intégration par la méthode des trapèzes implémentés avec le module numpy (n=1000)

3.3. Méthode préprogrammée

Nombre de segments	Solution analytique	Intégration par trapèzes	Pourcentage d'erreur	Temps d'exécution
10	200	205	2.5 %	0.093744
100	200	200.05	0.025 %	0.133550
1000	200	200.0005	0.00025 %	0.123812

Nous obtenons les mêmes valeurs que celles qui ont été obtenues par python régulier. Le temps d'exécution est plus lent que la méthode numpy, cependant, il ne varie presque pas en augmentant le nombre de segments, ce qui laisse estimer qu'avec un nombre de segment très élvée nous pourrions avoir une erreur encore plus petite et garder une certaine rapidité d'exécution.

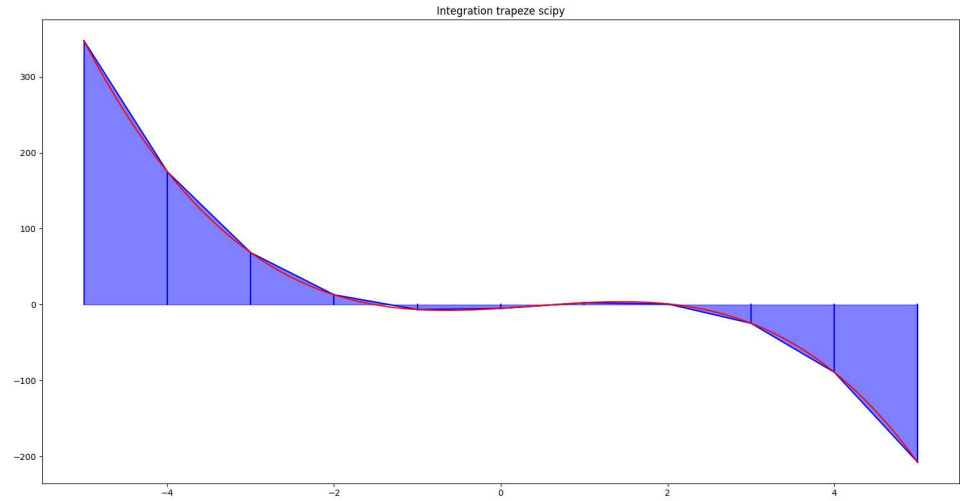

Figure 13: Intégration par la méthode des trapèzes implémentés avec le module scipy (n=10)

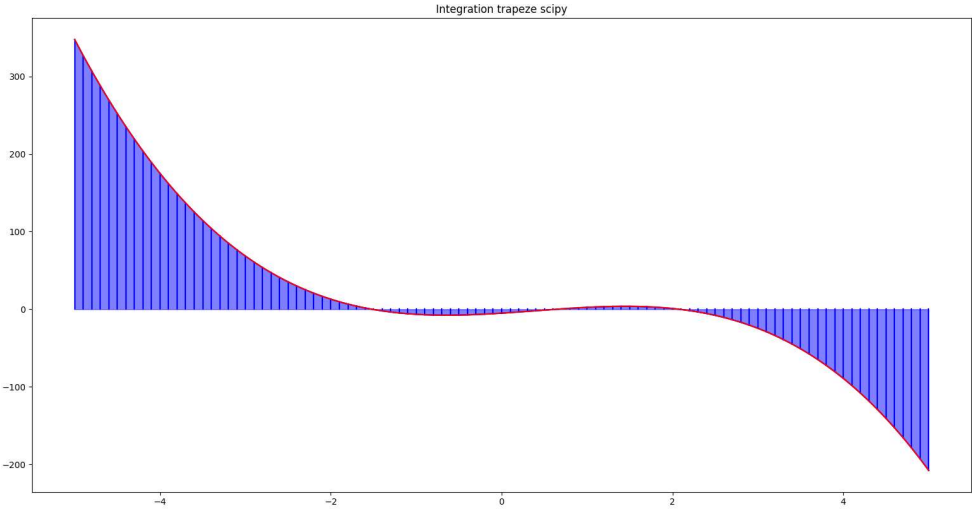

Figure 14: Intégration par la méthode des trapèzes implémentés avec le module scipy (n=100)

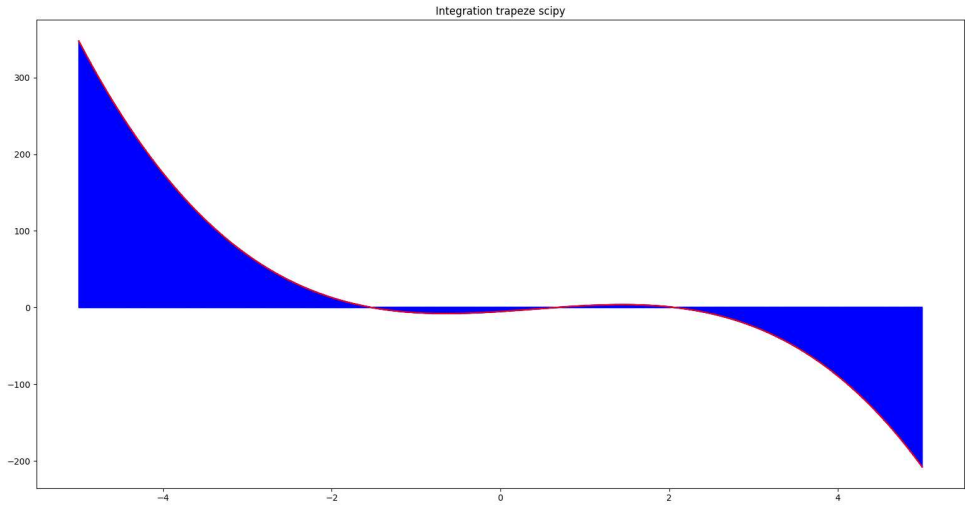

Figure 15: Intégration par la méthode des trapèzes implémentés avec le module scipy (n=1000)

4. Intégration par la méthode de Simpson

4.1. Python régulier

Nombre de segments	Solution analytique	Intégration par Simpson	Pourcentage d'erreur	Temps d'exécution
10	200	200	0 %	0.002764
100	200	~ 200	2.7 e -13 %	0.039216
1000	200	~ 200	1.27 e -11 %	0.396128

L'erreur offerte par cette méthode est trop faible pour être mesurée. En effet, celle-ci semble augmenter avec le nombre de segments, mais elle est bien trop faible pour être causée par la méthode elle-même. Ainsi, nous pensons que cette erreur est causée par l'imprécision des variables. Le temps d'exécution est très long pour cette méthode et ce fait ressentir rapidement en augmentant le nombre de segment.

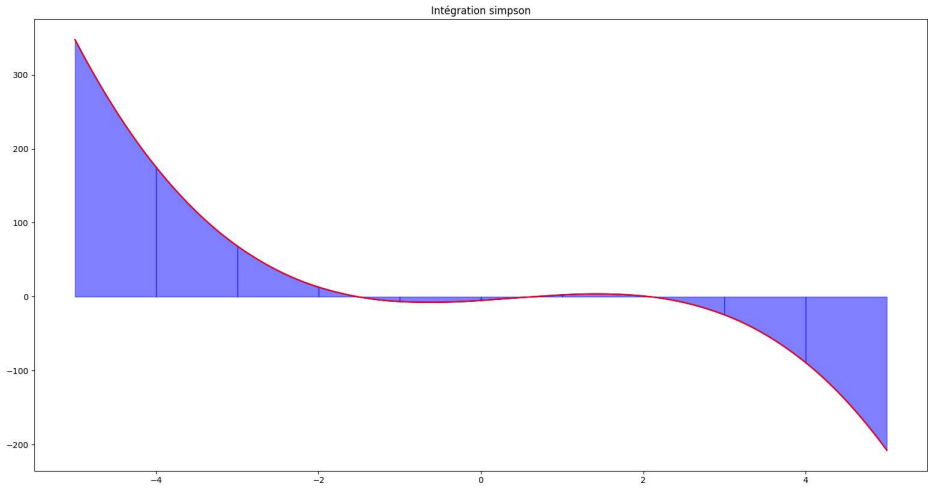

Figure 16: Intégration par la méthode de simpson implémentés avec du python pure (n=10)

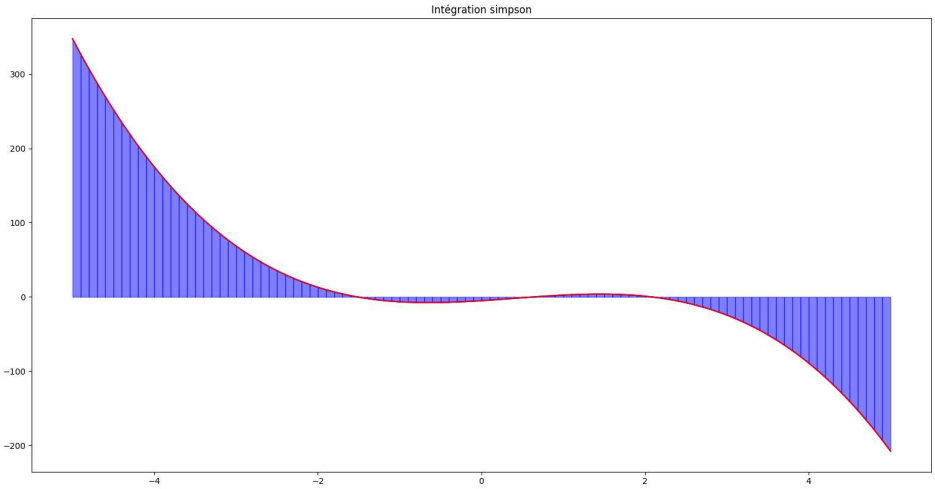

Figure 17: Intégration par la méthode de simpson implémentés avec du python pure (n=100)

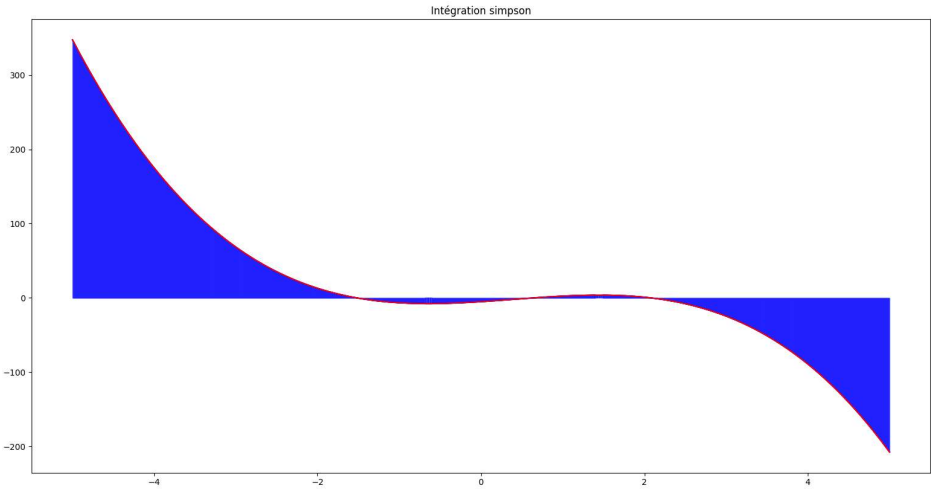

Figure 18: : Intégration par la méthode de simpson implémentés avec du python pure (n=1000)

4.2. Numpy

Nombre de segments	Solution analytique	Intégration par Simpson	Pourcentage d'erreur	Temps d'exécution
10	200	200	0 %	0.007803
100	200	~ 200	3.61 e -12 %	0.013007
1000	200	~ 200	2.2 e -11 %	0.060442

Nous pouvons observer le même phénomène en termes d'erreur. Encore une fois, nous estimons que l'erreur est causée par l'accumulation de l'imprécision des variables. Le temps d'exécution, cependant, est excessivement rapide, et ce, même avec un nombre de segment élevé.

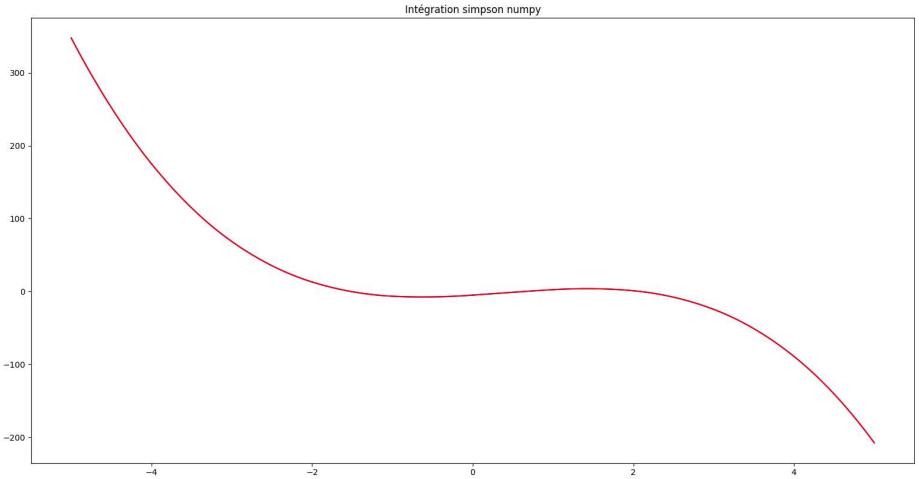

Figure 19 : Intégration par la méthode de simpson implémentés avec le module numpy (n=10)

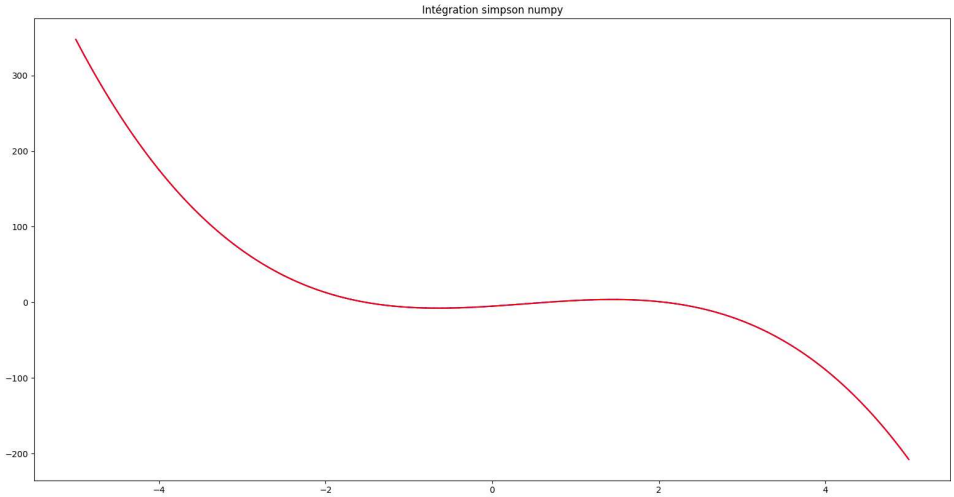

Figure 20: Intégration par la méthode de simpson implémentés avec le module numpy (n=100)

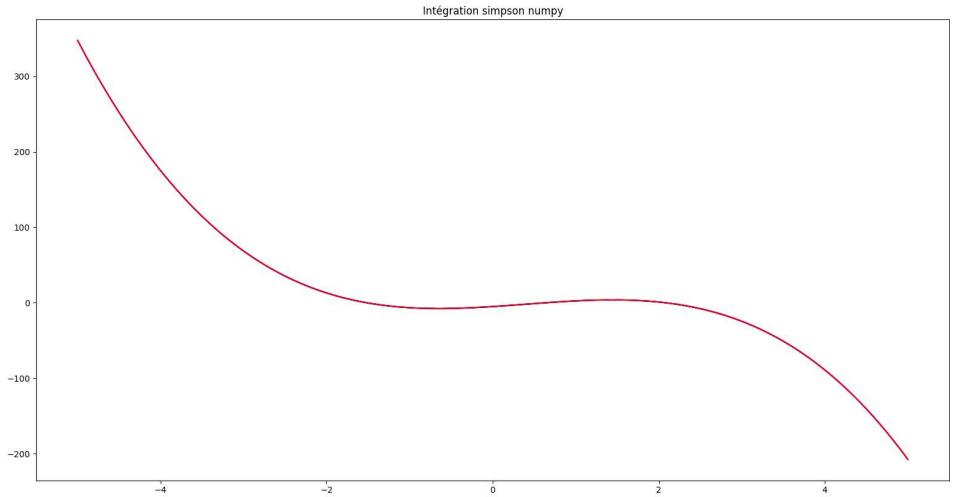

Figure 21: Intégration par la méthode de simpson implémentés avec le module numpy (n=1000)

4.3. Méthode préprogrammée

Nombre de segments	Solution analytique	Intégration par Simpson	Pourcentage d'erreur	Temps d'exécution
10	200	200	0 %	0.011266
100	200	~ 200	3.69 e -12 %	0.014952
1000	200	~ 200	2.21 e -11 %	0.022585

Nous pouvons observer le même phénomène en termes d'erreur. Encore une fois, nous estimons que l'erreur est causée par l'accumulation de l'imprécision des variables. Le temps d'exécution est très rapide et ne change presque pas en augmentant le nombre de segment.

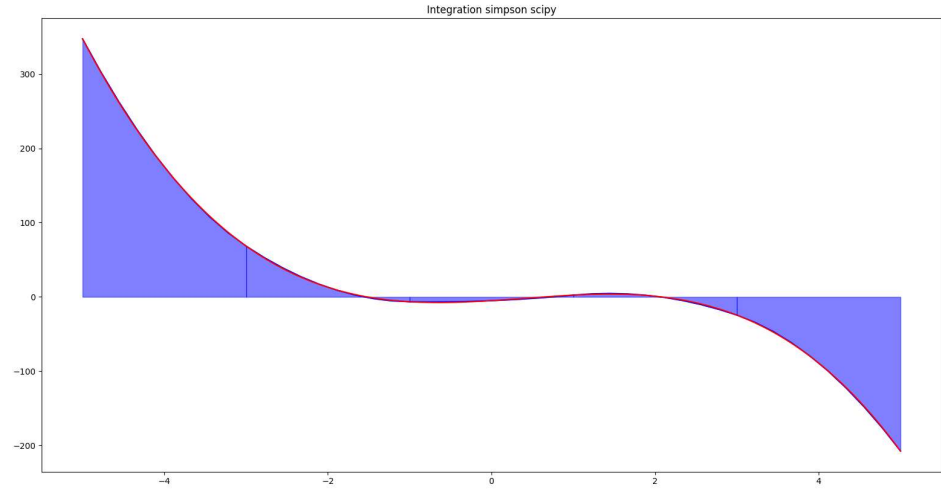

Figure 22: Intégration par la méthode de simpson implémentés avec le module scipy (n=10)

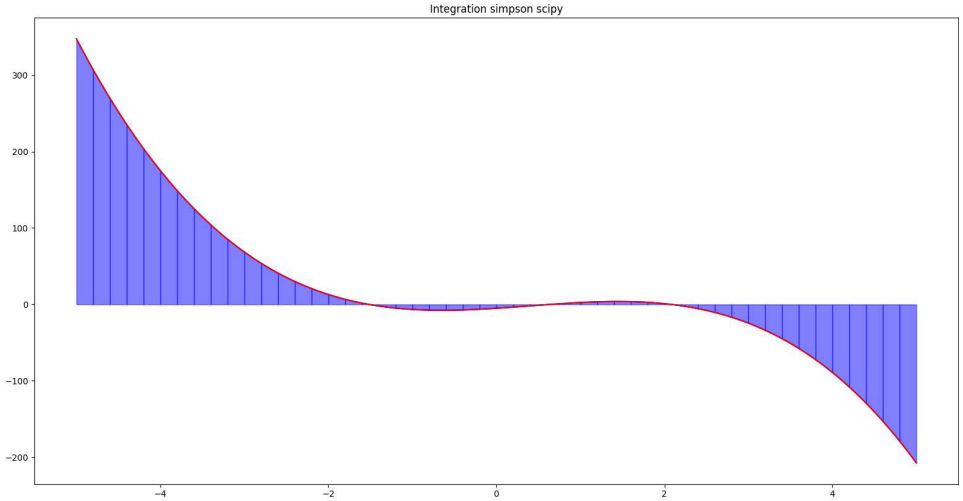

Figure 23: Intégration par la méthode de simpson implémentés avec le module scipy (n=100)

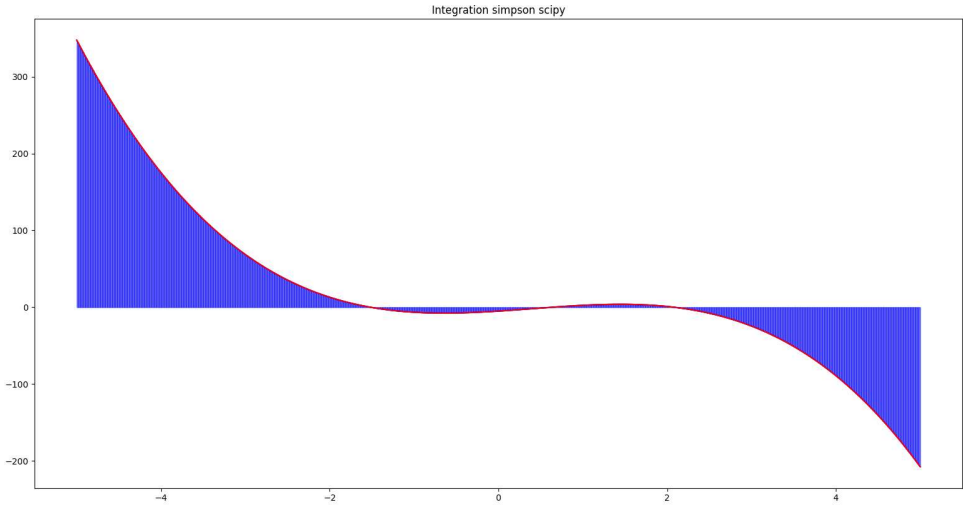

Figure 24: Intégration par la méthode de simpson implémentés avec le module scipy (n=1000)