TP 11 : Intégration

1 Méthodes des rectangles, du point milieu, des trapèzes et de Simpson

On rappelle que sur un petit intervalle $[x, x+h]$, les formules pour obtenir une approximation de $\int_x^{x+h} f(t)\,\mathrm{d}t$ sont les suivantes :

$$\mathcal{R}(f) = hf(x) \qquad \mathcal{M}(f) = hf\left(\frac{x+(x+h)}{2}\right) \qquad \mathcal{T}(f) = h\frac{f(x)+f(x+h)}{2}$$

$$\mathcal{S}(f) = \frac{h}{6}\left(f(x) + 4f\left(\frac{x+(x+h)}{2}\right) + f(x+h)\right)$$

où $\mathcal{R}(f)$, $\mathcal{M}(f)$, $\mathcal{T}(f)$ et $\mathcal{S}(f)$ sont respectivement les formules pour les méthodes des rectangles à gauche, du point milieu, des trapèzes et de Simpson. En pratique sur un « grand » intervalle $[a,b]$, on se fixe un entier n, on découpe $[a,b]$ en n petits intervalles $[a_k, a_k+h]$ (avec $h = \frac{b-a}{n}$ et $a_k = a+kh$ pour $0 \leq k \leq n-1$), sur lesquels on applique les formules précédentes.

Question 1. Écrire quatre fonctions `int_rec_g(f,a,b,n)`, `int_rec_m(f,a,b,n)`, `int_trapz(f,a,b,n)` et enfin `int_simpson(f,a,b,n)` prenant en entrée une fonction $f : [a,b] \to \mathbb{R}$, les bornes de l'intervalle, et un entier $n > 0$, et retournant l'approximation de $\int_a^b f(t)\,\mathrm{d}t$ obtenue avec la méthode des rectangles à gauche (respectivement du point milieu, des trapèzes, et de Simpson), avec l'intervalle $[a,b]$ découpé en n morceaux. *Remarque :* on pourra utiliser le fait que $\mathcal{S}(f) = \frac{2}{3}\mathcal{M}(f) + \frac{1}{3}\mathcal{T}(f)$.

Question 2. Vérifiez que la méthode du point milieu et la méthode des trapèzes sont exactes pour les fonction affines $t \mapsto at+b$, contrairement à la méthode des rectangles à gauche. Vérifiez que la méthode du point milieu et la méthode des trapèzes sont inexactes pour $t \mapsto t^2$, mais que la méthode de Simpson est exacte pour cette fonction, ainsi que pour une fonction cubique.

```
>>> print(int_rec_g(lambda x:1,0,1,10))
1.0
>>> print(int_rec_m(lambda x:1,0,1,10))
1.0
>>> print(int_rec_g(lambda x:x,0,1,10))
0.45
>>> print(int_rec_m(lambda x:x,0,1,10))
0.5
>>> print(int_rec_m(lambda x:x*x,0,1,10))
0.33249999999999996
>>> print(int_rec_m(lambda x:x*x,0,1,10))
0.33249999999999996
>>> print(int_simpson(lambda x:x*x, 0, 1, 10))
0.33333333333333326
>>> print(int_simpson(lambda x:x**3, 0, 1, 10))
0.24999999999999992
```

Question 3. *Intégrer avec Python.* Importer le module `scipy.integrate` sous le nom `sci` (c'est-à-dire qu'il faut écrire `import scipy.integrate as sci`). La fonction `quad` de ce module s'utilise comme suit : `sci.quad(f, a, b)` où f est la fonction à intégrer et a et b les bornes de l'intervalle. L'utiliser sur les fonctions précédentes. Remarque : la fonction renvoie un couple, la première composante est une estimation de $\int_a^b f$, la deuxième une estimation de l'erreur.

Question 4. *Évolution de l'erreur dans les méthodes précédentes.* On s'intéresse à la fonction $\mathcal{R} : x \mapsto \frac{1}{1+25x^2}$, dite fonction de *Runge*, qu'on veut intégrer sur l'intervalle $[0,1]$.

1. Utiliser la fonction `quad` du module `scipy.integrate` pour obtenir une approximation de $\int_0^1 \mathcal{R}(t)\,\mathrm{d}t$. Stocker la valeur donnée dans une variable `v`, on la considérera comme exacte.
2. Appliquer les quatre méthodes écrites plus haut (rectangles à gauche, point milieu, trapèzes, Simpson), pour n variant entre 10 et 500, par pas de 10. Stocker les erreurs (valeur absolue de la différence entre l'approximation obtenue et `v`) dans quatre listes. On rappelle que `abs` fournit la valeur absolue.

3. On peut montrer que si la fonction intégrée est suffisamment régulière (c'est le cas ici), les méthodes fournissent un résultat avec des erreurs en $O(1/n)$, $O(1/n^2)$, $O(1/n^2)$ et $O(1/n^4)$. En particulier, l'approximation tend vers la valeur de l'intégrale lorsque n tend vers l'infini, mais la convergence est d'autant plus rapide que l'ordre de la méthode est élevé. On souhaite vérifier ceci, il suffit donc de tracer le logarithme de l'erreur en fonction de $\log(n)$: on devrait obtenir une pente de coefficient directeur -1, -2 et -4 pour les trois méthodes. En utilisant `plt.loglog`, réaliser trois tracés permettant d'obtenir un tracé comme sur la figure 1 (on a aussi mis la méthode des trapèzes ici). La fonction `plt.loglog` s'utilise de la même manière que `plt.plot`, on importera comme d'habitude le module `matplotlib` sous l'alias `plt` (voir l'annexe !).

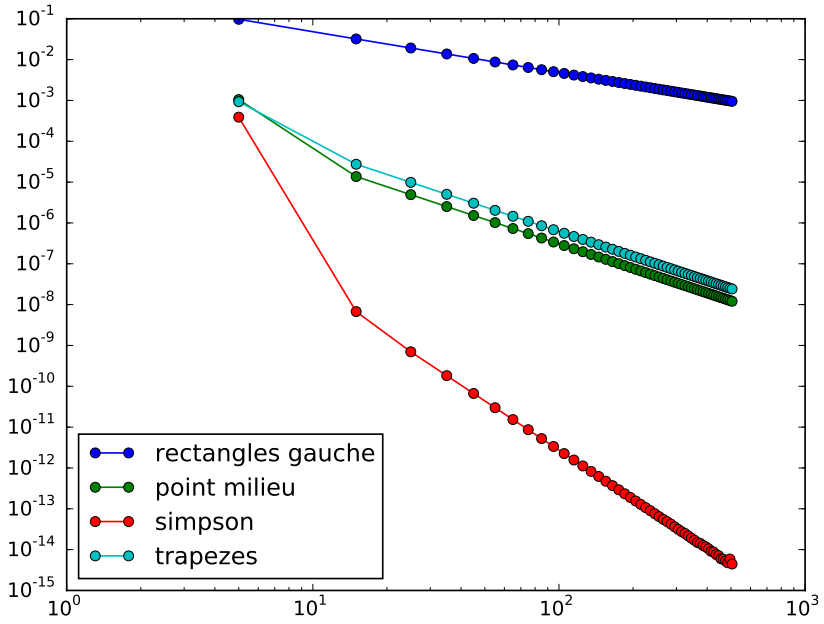

FIGURE 1: Évolution de l'erreur en fonction de n (diagramme log-log) pour les 4 méthodes vues en cours, sur la fonction de Runge intégrée sur $[0,1]$. Les méthodes des trapèzes et du point milieu ont une convergence semblable.

Question 5. *Nombre d'appels à f nécessaires pour contrôler l'erreur.* On cherche ici à approcher $\ln(2) = \int_1^2 \frac{\mathrm{d}t}{t}$. On rappelle les formules donnant une majoration de l'erreur dans le calcul de $\int_a^b f$ pour f une fonction de classe $\mathcal{C}^4$, sur l'intervalle $[a,b]$ découpé en n morceaux :

— rectangle à gauche : $\frac{(b-a)^2\|f'\|_\infty}{2n}$;

— trapèzes : $\frac{(b-a)^3\|f''\|_\infty}{12n^2}$;

— point milieu : $\frac{(b-a)^3\|f''\|_\infty}{24n^2}$;

— Simpson : $\frac{(b-a)^5\|f^{(4)}\|_\infty}{2880n^4}$;

avec ici $\|g\|_\infty = \sup\{|g(x)| \mid x \in [a,b]\}$ et $f^{(4)}$ la dérivée quatrième de f. Appliquer ces formules à $t \mapsto 1/t$ sur $[1,2]$, et déterminer pour chacune des méthodes le plus petit n tel que l'erreur soit bornée par 10^{-5}. En optimisant, on voit que le nombre d'appels à f nécessaire en fonction de n est :

— n pour la méthode des rectangles à gauche et du point milieu ;

— $n+1$ pour la méthode des trapèzes ;

— $2n+1$ pour la méthode de Simpson.

Comparer le nombre d'appels à $f = t \mapsto 1/t$ nécessaires pour le calcul de $\int_1^2 \frac{\mathrm{d}t}{t}$ avec une erreur bornée par 10^{-5}.

2 Intégrale d'une fonction issue d'une série de mesures

Voir l'annexe pour un rappel concernant l'ouverture de fichiers. On se donne un fichier représentant une série de mesures (à récupérer sur le site web). Les lignes du fichier sont de la forme `x;y` où `x` et `y` sont des flottants. On suppose que ces lignes sont le résultat de mesures, c'est à dire qu'il existe une fonction f telle que $y_i = f(x_i)$ pour chaque ligne $x_i; y_i$. Les abscisses x_i sont supposées ordonnées de façon croissante, par contre elles ne sont pas supposées ordonnées régulièrement : la distance $x_{i+1} - x_i$ n'est pas constante. En supposant que le fichier a ℓ lignes ($x_0; y_0$ jusqu'à $x_{\ell-1}; y_{\ell-1}$), on souhaite approcher $\int_{x_0}^{x_{\ell-1}} f(t)\,\mathrm{d}t$, en utilisant la méthode des trapèzes, c'est à dire :

$$\int_{x_0}^{x_{\ell-1}} f(t)\,\mathrm{d}t \simeq \sum_{k=0}^{\ell-2}(x_{k+1}-x_k)\frac{f(x_k)+f(x_{k+1})}{2} = \sum_{k=0}^{\ell-2}(x_{k+1}-x_k)\frac{y_k+y_{k+1}}{2}$$

Question 6. Écrire une fonction `integrale(fichier)` permettant d'automatiser le procédé, la fonction `integrale` prenant en entrée un fichier ouvert en lecture. On pourra par exemple utiliser `readlines`, `split` pour scinder une chaîne de caractères autour d'un élément (par exemple, `chaine.split(';')` fournit une liste correspondant à la séparation de la chaîne de caractères `chaine` autour de `;`), et on n'oubliera pas `float` permettant de convertir une chaîne de caractères en flottant.

J'obtiens ceci :

```
>>> f1=open("fichier1.txt",'r') ; integrale(f1)
464.5123984199802
>>> f2=open("fichier2.txt",'r') ; integrale(f2)
0.20764416913028605
```

Attention : pour ouvrir le fichier il faut préciser le chemin absolu, de la forme `"U:/..."` au lycée.

3 Une autre approche du calcul d'intégrale : la méthode de Monte-Carlo

Cette méthode de calcul approché est basée sur les probabilités : on importera le module `random` de Python. On utilisera la fonction `random()` qui donne un nombre réel compris entre 0 et 1, en suivant une loi uniforme.
On se donne une fonction $f : [a,b] \to \mathbb{R}$. On se donne n points $(x_0, \dots, x_{n-1})$ choisis de manière aléatoire dans $[a,b]$ (en suivant une loi uniforme). Alors $\frac{b-a}{n}\sum_{i=0}^{n-1} f(x_i)$ est une approximation de $\int_a^b f$, avec un intervalle de confiance à 95% de largeur $\alpha/\sqrt{n}$.

Question 7. *1.* Écrire une fonction `point(a,b)` renvoyant un réel aléatoire de $[a,b]$.

2. En déduire une implémentation `monte_carlo(f,a,b,n)` de la méthode.

3. Comparer la précision de cette méthode avec les précédentes, en faisant plusieurs essais.

Remarque : cette méthode n'est pas très précise. Néanmoins elle se généralise très bien en dimension supérieure, et est la seule réellement applicable en grandes dimensions.

Annexe : importation de modules

On peut importer les modules utiles pour ce TP comme suit :

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.integrate as sci
from math import *
from random import random
```

- Tracé de graphe : `plt.plot(X, Y)` avec `X` et `Y` contenant les listes d'abscisses et d'ordonnées de la courbe à tracer. `plt.loglog(X,Y)` fonctionne de la même manière. Pour rajouter une étiquette sur une courbe : `plt.plot(X, Y, label="etiquette")`. Pour tracer plusieurs courbes sur un même graphique, il suffit de tracer avec `plot` toutes les courbes, avant d'afficher. Pour rajouter la légende : `plt.legend()`. Pour afficher : `plt.show()`.

- Lecture de fichier et manipulation de chaînes : `f=open('nom_du_fichier', 'r')` pour ouvrir un fichier en lecture. Le plus simple est de parcourir le fichier avec `for ligne in fichier`, la variable `ligne` prend successivement comme valeur toutes les lignes du fichier (attention, le saut de ligne `\n` est présent à la fin de chaque ligne). Pour manipuler les lignes, utiliser des extractions de portions, `s.split(c)` permettant de séparer une chaîne de caractères `s` autour d'un caractère `c` (comme `';'` par exemple) : le résultat renvoyé est une liste contenant des morceaux de `s`. Utiliser `float` ou `int` pour convertir une chaîne de caractères en flottant ou entier.
- Autre possibilité pour manipuler un fichier ouvert en lecture : `f.readlines()` renvoie une liste contenant toutes les lignes du fichier.
- `sci.quad(f,a,b)` renvoie un couple (v, e) où $v \simeq \int_a^b f$ et e est une majoration de l'erreur.