Université Lyon 1 Année 2018-2019
Licence Mathématiques 2ème année
Introduction à l'analyse numérique

TP 2

Exercice 1. (*Ordre des formules d'intégration numérique*)

L'intégrale I d'une fonction f entre a et b peut être calculée de manière approchée en utilisant les formules d'intégration de Newton-Cotes. On note $y_j = a + jh$, pour $j \in \{0, \dots, N\}$, avec $h = (b-a)/N$. On considère les formules d'intégration composées suivantes :

— Rectangles à gauche : $I \approx h \sum_{0 \leq J \leq N-1} f(y_j)$.

— Rectangles à droite : $I \approx h \sum_{0 \leq j \leq N-1} f(y_{j+1})$.

— Trapèzes : $I \approx h \sum_{0 \leq j \leq N-1} \frac{f(y_j) + f(y_{j+1})}{2}$.

— Simpson : $I \approx \frac{h}{6} \sum_{0 \leq j \leq N-1} \left(f(y_j) + 4f\left(\frac{y_j + y_{j+1}}{2}\right) + f(y_{j+1}) \right)$.

Une implémentation de ces méthodes est donnée par la fonction suivante :

```
import numpy as np

def integration_approchee(f, a, b, N):
    Y, h = np.linspace(a, b, N+1, retstep=True)
    I = np.empty(4)
    I[0] = h/6 * np.sum(f(Y[0:n]) + 4*f((Y[0:n] + Y[1:n+1]) / 2) + f(Y[1:n+1]))
    I[1] = h/2 * np.sum(f(Y[0:n]) + f(Y[1:n+1]))
    I[2] = h * np.sum(f(Y[0:n]))
    I[3] = h * np.sum(f(Y[1:n+1]))
    return I
```

La fonction `integration_approchee` prend comme arguments d'entrée une fonction `f`, les bornes `a` et `b` de l'intervalle d'intégration et `N` le nombre de sous-intervalles utilisés pour calculer l'intégrale approchée par une méthode composée. En sortie, la fonction renvoie un tableau `I` contenant la valeur approchée de l'intégrale avec chacune des quatre méthodes de quadrature.

1. Écrire la fonction `integration_approchee` dans un fichier **integration.py**. À quelle méthode correspond chaque composante de I ?

RÉPONSE : I[0] :
I[1] :
I[2] :
I[3] :

2. Quel est l'ordre théorique de chacune des méthodes ?

RÉPONSE : rectangles à droite :

rectangles à gauche :

trapèzes :

Simpson :

3. On considère $a = 0$, $b = \pi/2$ et $f(x) = \sin(x)$. Définir la fonction f dans le fichier **integration.py** en utilisant la commande `def`.
4. Calculer algébriquement l'intégrale suivante :

RÉPONSE : $I = \displaystyle\int_0^{\pi/2} \sin(x)\,\mathrm{d}x =$

5. On souhaite tracer l'évolution de l'erreur de chacune des formules d'intégration approchée en fonction du nombre N de sous-intervalles $[y_j, y_{j+1}]$ considérés. La fonction suivante en est une implémentation.

```
def erreur_integration(f, a, b, valTh, Nmax):
    err = np.empty((Nmax, 4))
    for k in range(Nmax):
        I = integration_approchee(f, a, b, k+1)
        err[k, :] = np.abs(valTh - I)

    n = np.arange(1, Nmax+1)
    plt.plot(np.log(n), np.log(err[:, 0]), "r")
    plt.plot(np.log(n), np.log(err[:, 1]), "b")
    plt.plot(np.log(n), np.log(err[:, 2]), "m")
    plt.plot(np.log(n), np.log(err[:, 3]), "k")
    plt.xlabel("log(n)")
    plt.ylabel("log(err)")
    plt.title("...")
    plt.legend(["methode ...", "methode ...", "methode ...", "methode ..."])
    plt.show()

    return err
```

La fonction `erreur_integration` prend comme arguments d'entrée une fonction `f`, les bornes `a` et `b` de l'intervalle d'intégration, `valTh` la valeur algébrique de l'intégrale de f sur $[a, b]$, et `Nmax` le nombre maximal de sous-intervalles. En sortie, la fonction renvoie un tableau `err` contenant l'erreur entre la valeur théorique de l'intégrale et les valeurs approchées obtenues avec chacune des quatre méthodes de quadrature.

(a) Écrire la fonction `erreur_integration` dans le fichier **integration.py** et mettre en légende la méthode d'intégration approchée qui correspond à chaque figure.

(b) Utiliser la fonction avec `Nmax=50`. Quelles sont les quantités tracées sur la figure qui s'affiche ? Mettre un titre à la figure.

RÉPONSE :

(c) Qu'affichent les commandes suivantes ?

```
n = np.arange(1, Nmax+1)
print(np.polyfit(np.log(n), np.log(err[:, 0]), 1))
print(np.polyfit(np.log(n), np.log(err[:, 1]), 1))
print(np.polyfit(np.log(n), np.log(err[:, 2]), 1))
print(np.polyfit(np.log(n), np.log(err[:, 3]), 1))
```

RÉPONSE :

(d) Quelles quantités retrouvez-vous ?

RÉPONSE :

Exercice 2. (*Méthode de Newton*)
On rappelle qu'il s'agit d'une méthode calculant les zéros d'une fonction f en utilisant l'algorithme donné comme suit en dimension 1

$$\begin{cases} x_0 \in \mathbb{R} \text{ donné} \\ x_{k+1} = x_k - \dfrac{f(x_k)}{f'(x_k)} \text{ pour tout } k \in \mathbb{N} \end{cases}$$

Une implémentation de cette méthode est donnée ci-dessous.

```
def newton(f, df, x0, tol, nmax):
    valeurs = [x0]
    x = x0
    err = tol + 1

    niter = 0
```

```
    while err > tol and niter < nmax:
        x = x - f(x) / df(x)
        valeurs.append(x)
        err = np.abs(valeurs[niter+1] - valeurs[niter])
        niter = niter + 1

    if niter == nmax and err > tol:
        print("Arrêt, nombre maximum d'itérations atteint")

    return np.array(zero), niter
```

La fonction `newton` prend en arguments d'entrée une fonction `f`, sa dérivée `df`, un point initial `x0`, la tolérance souhaitée `tol` et le nombre maximal d'itérations souhaitées `nmax`. En sortie, cette fonction renvoie la suite des approximations $(x_k)_{k\leq niter}$ dans la variable `x` et le nombre d'itérations pris par l'algorithme pour atteindre la tolérance `tol` dans la variable `niter`.

On considère la fonction $f : \mathbb{R} \to \mathbb{R}$, $x \mapsto x + e^x + \dfrac{10}{1+x^2} - 5$ définie de la façon suivante :

```
def f(x):
    return x + np.exp(x) + 10 / (1 + x**2) - 5
```

Dans la suite, on prendra `tol=1e-8`.

1. Tracer le graphe de la fonction f sur l'intervalle $[-2, 2]$ à l'aide de la commande `plot`.
2. Calculer algébriquement la dérivée de f, et définir la fonction Python correspondante, que l'on appellera `df`.

   RÉPONSE : f'(x) =

3. Donner une valeur approchée du zéro de f à l'aide de la commande `fsolve` du paquet `scipy.optimize` (qui utilise une autre méthode que la méthode de Newton) pour $x_0 = -0.1$.

   RÉPONSE :

   Nous allons trouver une approximation semblable à l'aide de la méthode de Newton.
4. Écrire la fonction `newton` dans un fichier **newton.py**.
5. On considère $x_0 = -0.1$. À partir de combien d'itérations la méthode a-t-elle convergé ?

   RÉPONSE :

6. Tracer la suite des itérés $(x_k)_{0\leq k\leq niter}$.

7. On rappelle qu'une suite $(x_k)_{k\in\mathbb{N}}$ converge vers c au moins à l'ordre $r \geq 1$ dès lors que

$$\exists\, C_0 > 0,\, \forall k \in \mathbb{N}, \quad |x_{k+1} - c| \;\leq\; C_0 |x_k - c|^r \;\; .$$

   En déduire un graphe que l'on peut tracer pour mettre en évidence l'ordre de convergence r d'une méthode itérative.

RÉPONSE :

8. Tracer la figure mettant en évidence l'ordre de convergence de la méthode de Newton.
9. En utilisant la commande `polyfit` (cf première partie du TP), déterminer l'ordre de convergence.

RÉPONSE :

10. Que donnent les méthodes `fsolve` et `newton` si l'on prend $x_0 = 1$? Proposer une explication.

RÉPONSE :