TP : Méthodes d'intégration numérique

Pour commencer, connectez-vous au serveur L2 de la plateforme Jupyter de Lyon 1 : `https://l2-nbgrader.univ-lyon1.fr` avec vos identifiant et mot de passe de l'université.

Récupérer le notebook de ce TP en procédant comme suit :

1. Choisir *Nbgrader –> Assigment List*.
2. Dans le menu *Released, downloaded, and submitted assignments for course*, choisir MAT2092L.
3. Puis *Fetch* `TP1_2024` dans la rubrique *Released assignments*.
4. Cliquer sur le lien `TP1_2024` se trouvant à gauche dans la rubrique *Downloaded assignments*, puis juste en-dessous sur le lien vers le notebook `TP_integration_numerique`

À l'exception des cadres à compléter, rédiger vos réponses sur une feuille indépendante en reportant les numéros des questions.

À la fin de la séance de ce TP, nous vous invitons à soumettre ce notebook complété (une partie de votre code sera ensuite testée de manière automatique). Pour cela, via *Nbgrader –> Assigment List*, **cliquer sur *Submit***.

0 Échelles logarithmiques - Régression linéaire

Soit $c > 0$ et $\alpha \in \mathbb{R}$. On considère la suite $(v_n)_{n\geq 1}$ définie pour $n \geq 1$ par $v_n = \dfrac{c}{n^\alpha}$.

1) Expliciter la relation linéaire entre $\ln(n)$ et $\ln(v_n)$ pour tout $n \geq 1$.

2) Observer le premier exemple fourni et expliquer l'intérêt des échelles logarithmiques dans cet exemple.

3) Que retourne le code suivant ?

```
from scipy import stats
res = stats.linregress(log_n,log_vn)
print(res.intercept, res.slope)
```

4) Déterminer un équivalent simple de $e^2/(n + \ln n + 2)^4$ quand n tend vers l'infini.

5) Montrer qu'une suite (w_n) est équivalente à (v_n) quand n tend vers l'infini si et seulement $\ln w_n + \alpha \ln n \xrightarrow[n\to+\infty]{} \ln c$.

6) Observer le second exemple fourni et expliquer l'usage de droites de régression linéaire pour approximer les valeurs de α et de $\ln c$.

1 Méthode(s) des rectangles

Considérons une fonction réelle $f : [a, b] \to \mathbb{R}$ de classe C^1.

1.1 Formule de quadrature de la méthode des rectangles à gauche

Rappelons que la méthode des rectangles à gauche consiste à approcher l'intégrale $I = \displaystyle\int_a^b f(x)dx$ par une somme d'aires algébriques de rectangles : plus précisément, on considère une subdivision régulière de l'intervalle $[a, b]$

$$a = x_0 < x_1 < \cdots < x_n = b \quad \text{où} \quad x_k = \qquad\qquad \text{pour } k \in \Big\{ \qquad\qquad \Big\}.$$

et on approxime I par la valeur S_n où

$$S_n = \text{——} \sum_{\leq k \leq} f(x_k).$$

1) À quelle notion vue en cours correspond cette formule par ailleurs ?

Une implémentation de cette méthode est donnée par le code suivant :

```
import numpy as np

def int_approchee_rect_gauche(f,a,b,n):
    x,h=np.linspace(a,b,n+1,retstep=True)
    Sn=h * np.sum(f(x[0:n]))
    return Sn
```

2) Tester cette méthode sur la fonction sin sur $[0, \pi/2]$ et comparer les résultats à la valeur exacte de $\int_0^{\pi/2} \sin(x)dx$.

3) Tester également cette méthode sur la fonction $f : [0,1] \to \mathbb{R},\ x \mapsto \dfrac{1}{1+x^2}$ et comparer à la valeur exacte de $\int_0^1 \dfrac{1}{1+x^2}dx$

Illustration graphique de cette méthode (avec une échelle logarithmique sur l'axe des abscisses) :

```
import matplotlib.pyplot as plt

def rep_int_approchee_rect_gauche(f,a,b,nMax):
    S =[int_approchee_rect_gauche(f,a,b,n) for n in range(1,nMax+1)]
    log_n = np.log(np.arange(1,nMax+1))
    plt.plot(log_n,S,'r+')
    plt.xlabel("ln(n)")
    plt.ylabel("valeur approchée de l'intégrale")
    plt.title("Méthode des rectangles àgauche")
    plt.show()
    return S
```

4) Tester ce code sur les deux fonctions précédentes avec `nMax=50`. Que représente-t-on ? (abscisses, ordonnées ...)

1.2 Erreur de l'estimation de l'intégrale

En fonction du nombre n de sous-intervalles de $[a,b]$ considérés, notons $E_n = |S_n - I|$ l'erreur commise lors de l'estimation de l'intégrale (c.à.d. la valeur absolue de la différence de la valeur de l'intégrale avec l'estimation de celle-ci par la formule de quadrature).

On rappelle que pour la méthode des rectangles à gauche, on a la majoration théorique suivante de cette erreur :

$$\forall n \geq 1,\ E_n \leq \frac{M_1(b-a)^2}{2n} \text{ avec } M_1 = \quad .$$

5) Déterminer M_1 pour chacune des deux fonctions précédentes.

On souhaite maintenant tracer l'évolution de l'erreur, ainsi que sa majoration théorique en fonction de n :

```
def erreur_integration(f,a,b,I,M1,nMax):
    S = np.array([int_approchee_rect_gauche(f,a,b,n) for n in range(1,nMax+1)])
    E = np.abs(S-I)
    majE = np.array([M1*(b-a)**2/(2*n) for n in range(1,nMax+1)])
    log_n = np.log(np.arange(1,nMax+1))
    plt.plot(log_n,np.log(E),'r+')
    plt.plot(log_n,np.log(majE),'b+')
    plt.xlabel("ln(n)")
    plt.ylabel("ln(erreur)")
    plt.legend(["erreur","majoration théorique"])
    plt.title("Méthode des rectangles àgauche")
    plt.show()
```

6) Tester ce code sur les deux fonctions précédentes avec `nMax=50`. Que représente-ton ? Pourquoi utilise-t-on des échelles logarithmiques ? (L'argument d'entrée `I` de cette fonction Python correspond à la valeur exacte de l'intégrale calculée par vos soins.)

1.3 Comparaison avec les formules de quadrature des méthodes des rectangles à droite et au milieu.

Rappeler les formules de quadrature de ces méthodes :

Rectangles à droite : $S_n =$ —— $\sum_{\leq k \leq} f(x_k)$ Rectangles au milieu : $S_n =$ —— $\sum_{\leq k \leq} f($).

Implémenter ces deux formules avec deux fonctions Python

```
def int_approchee_rect_droite(f,a,b,n):
    # compléter

def int_approchee_rect_milieu(f,a,b,n):
    # compléter
```

Illustrer sur un même graphique ces trois méthodes des rectangles par une fonction Python

```
def rep_int_approchee_rect(f,a,b,nMax):
    # compléter
```

À l'aide de la fonction Python suivante, comparer l'évolution des erreurs respectives des trois formules.

```
def erreur_integration(f,a,b,I,nMax):
    SG = np.array([int_approchee_rect_gauche(f,a,b,n) for n in range(1,nMax+1)])
    SD = np.array([int_approchee_rect_droite(f,a,b,n) for n in range(1,nMax+1)])
    SM = np.array([int_approchee_rect_milieu(f,a,b,n) for n in range(1,nMax+1)])
    EG = np.abs(SG-I)
    ED = np.abs(SD-I)
    EM = np.abs(SM-I)
    log_n = np.log(np.arange(1,nMax+1))
    plt.plot(log_n,np.log(EG),'r+')
    plt.plot(log_n,np.log(ED),'b+')
    plt.plot(log_n,np.log(EM),'g+')
    plt.ylabel("ln(erreur)")
    plt.xlabel("ln(n)")
    plt.legend(["à gauche","à droite","au milieu"])
    plt.title("Méthodes des rectangles")
    plt.show()
```

7) Qu'observe-t-on ? (Remarque : faire le lien avec les majorations théoriques des erreurs vues en cours).
Compléter la fonction Python précédente avec les lignes de code

```
print(stats.linregress(log_n,np.log(EG)).slope)
print(stats.linregress(log_n,np.log(ED)).slope)
print(stats.linregress(log_n,np.log(EM)).slope)
```

8) Qu'affichent ces commandes ? Quelles valeurs retrouvez-vous ?

1.4 Régularité et vitesse de convergence

9) Calculer la valeur exacte de $\int_0^2 |x^2 - 2| dx$.

10) Tester les méthodes `rep_int_approchee_rect(f,a,b,nMax)` et `erreur_integration(f,a,b,I,nMax)` sur cet exemple.

11) Qu'observe-t-on ? Avez-vous une explication ?

12) Proposer une solution pour augmenter la vitesse de convergence sur cet exemple.

2 Méthode des trapèzes [Facultatif]

On considère maintenant une fonction réelle $f : [a, b] \to \mathbb{R}$ de classe C^2 et comme dans la partie précédente, une subdivision régulière $a = x_0 < x_1 < \cdots < x_n = b$ de l'intervalle $[a, b]$.

2.1 Formule de quadrature

Rappelons que la méthode des trapèzes consiste à approximer l'intégrale de f sur chacun des sous-intervalles $[x_k, x_{k+1}]$ par l'intégrale de la fonction affine dont le graphe passe par les points $(x_k, f(x_k))$ et $(x_{k+1}, f(x_{k+1}))$.

Rappeler la formule de quadrature de la méthode des trapèzes : $S_n =$

Implémenter cette formule en Python (et la tester !) :

```
def int_approchee_trap(f,a,b,n):
    # compléter
```

2.2 Erreur de l'estimation de l'intégrale

Compléter/modifier la fonction `erreur_integration` pour y inclure cette méthode.

Qu'observe-t-on sur l'ordre de grandeur de l'erreur ? **Réponse :**

Pour cette méthode, on a la majoration théorique suivante de l'erreur

$$E_n \leq M_2 \frac{(b-a)^3}{12n^2} \text{ avec } M_2 = \quad .$$

Le but de l'exercice suivant est de démontrer cette majoration (une correction de cet exercice pourra être faite en TD, voir la fiche 3). Pour montrer ce résultat, on majore les erreurs commises sur chaque sous-intervalle $[x_k, x_{k+1}]$ pour k parcourant $\{0, \ldots, n-1\}$. Fixons k : l'erreur commise sur l'intervalle $[x_k, x_{k+1}]$ s'écrit

$$\varepsilon_k = \left| \int_{x_k}^{x_{k+1}} (g(t) - f(t))dt \right|$$

où g est la fonction affine vérifiant $g(x_k) = f(x_k)$ et $g(x_{k+1}) = f(x_{k+1})$.

1) Expliciter la formule définissant g.

2) Fixons un réel $t \in]x_k, x_{k+1}[$. Vérifier qu'il existe un réel K tel que la fonction

$$h_t : [x_k, x_{k+1}] \to \mathbb{R},\ x \mapsto g(x) - f(x) + K\frac{(x-x_k)(x-x_{k+1})}{2}$$

s'annule en t.

3) En utilisant le théorème de Rolle, montrer qu'il existe $c_1 \in]x_k, t[$ et $c_2 \in]t, x_{k+1}[$ tel que $h_t'(c_1) = h_t'(c_2) = 0$.

4) En déduire qu'il existe $c_3 \in]c_1, c_2[$ tel que $h_t^{''}(c_3) = 0$.

5) Vérifier que $K = f^{''}(c_3)$ et en déduire que $g(t) - f(t) = -f^{''}(c_3)\frac{(t-x_k)(t-x_{k+1})}{2}$.

6) Vérifier alors l'inégalité

$$\varepsilon_k \leq \frac{M_2}{2} \int_{x_k}^{x_{k+1}} (t-x_k)(x_{k+1}-t)dt.$$

7) Calculer le second membre de cette inégalité (on pourra faire une intégration par partie).

8) Conclure.

3 Interpolation de Lagrange [Très facultatif]

3.1 Polynômes de Lagrange

Expliciter sous forme développée les 3 polynômes de Lagrange pour les nœuds 0, 1/2 et 1.

$L_0 =$ $L_1 =$ $L_2 =$

Que retourne la fonction Python suivante ? (On pourra la tester avec le code `L(2)[0], L(2)[1], L(2)[2]`.)

```
from numpy.polynomial import Polynomial
def L(d):
    X = Polynomial([0,1])
    L= [np.prod([(d*X-j)/(i-j) for j in list(range(i))+list(range(i+1,d+1))]) for i in range(d+1)]
    return L
```

Réponse :

Fixons un entier $d \geq 1$ et notons L_0, L_1, …, L_d, respectivement $\tilde{L}_0$, $\tilde{L}_1$, …, $\tilde{L}_d$, les polynômes de Lagrange aux $d+1$ nœuds équi-distribués dans l'intervalle $[0,1]$, respectivement dans l'intervalle $[a,b]$.

1) Montrer que pour chaque $i \in \{0,\dots,d\}$,

$$\int_a^b \tilde{L}_i(x)dx = (b-a)\int_0^1 L_i(x)dx.$$

Pour la suite, on posera pour $i \in \{0,\dots,d\}$, $w_i = \int_0^1 L_i(x)dx$.

2) Calculer w_0, w_1 et w_2 pour $d=2$: $w_0 =$ $w_1 =$ $w_2 =$

Compléter le code suivant pour calculer numériquement les valeurs des w_k.

```
def w(d):
    Lag=L(d)
    coef=[list(Lag[i]) for i in range(d+1)] # tableau des coefficients des polynômes de Lagrange
    # compléter
    return w
```

3.2 Méthodes simples

On construit des formules de quadrature à partir des polynômes d'interpolation de Lagrange : considérons une fonction $f : [a,b] \to \mathbb{R}$ intégrable, on remplace le calcul de l'intégrale de f par celui de l'intégrale d'un polynôme interpolateur.

3) En utilisant la partie précédente, montrer que la formule de quadrature obtenue à l'aide du polynôme interpolateur de Lagrange de f aux $d+1$ nœuds équi-distribués dans l'intervalle $[a,b]$ vaut

$$(b-a)\sum_{i=0}^{d} w_i f\left(a + i\frac{(b-a)}{d}\right).$$

Expliciter cette formule pour $d = 2$:

Réponse :

Coder une fonction `int_approchee_Lag_simple(f,a,b,d)` qui retourne l'intégrale du polynôme interpolateur de Lagrange en question.

3.3 Méthodes composites

On considère à nouveau une subdivision régulière $a = x_0 < x_1 < \cdots < x_n = b$ de l'intervalle $[a, b]$ et on approxime l'intégrale de f sur chacun des sous-intervalles $[x_k, x_{k+1}]$ par l'intégrale du polynôme interpolateur.

Que retrouve-t-on pour $d = 1$?

Réponse :

Méthode de Simpson

Le cas $d = 2$ est la méthode Simpson : on approxime sur chaque sous-intervalle $[x_k, x_{k+1}]$ par l'intégrale du polynôme interpolateur quadratique aux nœuds x_k, $(x_k + x_{k+1})/2$, x_{k+1}.

Expliciter la formule de quadrature de la méthode Simpson.

Réponse :

Implémenter cette formule en Python et compléter la fonction `erreur_integration`.

Sur des fonctions suffisamment régulières, estimer l'ordre de grandeur de l'erreur.

Réponse :

Méthodes composites d'ordre supérieur

À l'aide des fonctions Python suivantes, estimer l'ordre de grandeur de l'erreur pour $d = 3, 4, 5, 6$ et des fonctions suffisamment régulières. (N'hésitez pas à diminuer la valeur de la variable `nMax`.

```
def int_approchee_Lag_composite(f,a,b,d,n):
    W=w(d)
    x,h=np.linspace(a,b,n+1,retstep=True)
    pas=h/d
    Sn = h*np.sum([np.sum([W[i]*f(x[k]+i*pas) for i in range(d+1)]) for k in range(n)])
    return Sn

def erreur_integration(f,a,b,I,d,nMax):
    SC = np.array([int_approchee_Lag_composite(f,a,b,d,n) for n in range(1,nMax+1)])
    EC = np.abs(SC-I)
    log_n = np.log(np.arange(1,nMax+1))
    plt.plot(log_n,np.log(EC),'k+')
    plt.ylabel("ln(erreur)")
    plt.xlabel("ln(n)")
    plt.title("Méthode composite d'ordre supérieur")
    plt.show()
    print(stats.linregress(log_n,np.log(EC)).slope)
```