# Correction TP3 : Intégration numérique

**Module :** Calcul Scientifique **Classe :** $3^{me}$ année **A.U** 2023/2024

## 1 Introduction

Le module Sympy permet de faire le calcul mathématique formel, par exemple le calcul de primitive et le calcul d'intégrale en utilisant la fonction **integrate** et la dérivée analytique d'une fonction en utilisant la fonction **diff**
* Pour utiliser les fonctions **diff** et **integrate** :

• On fait l'appel à la bibliothèque sympy en utilisant **import sympy as sp**
• On utilise les variables symboliques et on déclare par exemple **x** sous cette forme comme suit: **x=sp.symbols('x')**

* Utilisation des fonctions **diff** et **integrate**

a) Pour trouver la dérivée d'une fonction $f$, on utilise **sp.Lambda(x,sp.diff(f(x),x))**

b) Pour trouver une primitive de $f$, on utilise $\int f(x)dx =$ **sp.integrate(f(x),x)**

c) Pour calculer l'intégrale $\int_a^b f(x)dx$, on utilise **sp.integrate(f(x),(x,a,b))**

d) Pour donner la valeur numérique de l'intégrale $\displaystyle\int_a^b f(x)dx$, on utilise **sp.integrate(f(x),(x,a,b)).evalf()**

Prenant $f(x) = cos(2x)e^{-x}$. Exécuter les instructions dans le script ci-dessous qui permettent de déterminer la dérivée de $f$, l'évaluation de $f$ et sa dérivée en $\frac{\pi}{2}$, une primitive de $f$, et la valeur de $I = \displaystyle\int_{-1}^{0} f(x)dx$.

[ ]:
```
import sympy as sp                  #  appel à la bibliothèque sympy sous le nom sp
import numpy as np                  #  appel à la bibliothèque numpy sous le nom np
x=sp.symbols('x')                   #  déclarer x comme variable symbolique
f=lambda x:sp.cos(2*x)*sp.exp(-x)      #  prédéfinir la fonction f
df=sp.Lambda(x,sp.diff(f(x),x))     # la dérivée de f
V1=f(sp.pi/2); V2=df(sp.pi/2).evalf()     # Evaluer f et df au point 2
I1=sp.integrate(f(x),x)              #Calcul d'une primitive de I
I2=sp.integrate(f(x),(x,-1,0))          #Calcul de la valeur analytique de I
I=sp.integrate(f(x),(x,-1,0)).evalf() # Calcul de la valeur numérique de I
print(df,V1,V2)
print('primitive=', I1)
print('Valeur analytique =', I2,'valeur numérique =', I)
```

Soit $J = \int_{\frac{1}{e}}^{e} \frac{ln(x)}{1+x^2}\,dx$. Que peut-on conclure lorsqu'on utilise la fonction **integrate** pour calculer la valeur de J?

La fonction **integrate** ne peut pas calculer la valeur de $J$.

## 2 Application

Dans cet TP, on s'intéresse à approcher l'intégrale $I(f) = \int_a^b f(x)dx$, avec $f$ une fonction de classe $C^1$ sur $[a,b]$, par une formule de quadrature définie par:

$$Q_n(f) \quad = \frac{h}{60}\left(14\big(f(a)+f(b)\big)+h\big(f'(a)-f'(b)\big)+28\sum_{k=1}^{n-1} f(x_k)+32\sum_{k=0}^{n-1} f(m_k)\right),$$

où

- $n$ est le nombre de sous-intervalles de la subdivision uniforme de $[a,b]$,
- $h = \dfrac{b-a}{n}$ est le pas de la discrétisation de l'intervalle $[a,b]$,
- $x_k = a + kh$ , pour $k \in \{0,\cdots,n\}$, désignent les points d'intégration,
- $m_k = a + (k+\frac{1}{2})h$ , pour $k \in \{0,\cdots,n-1\}$, désignent les points milieux,
- $f'(a)$ et $f'(b)$ sont les valeurs de la dérivée de la fonction $f$ aux points $x = a$ et $x = b$ respectivement.

Question 1 :

Écrire une fonction nommée **formulecomposite(f,df,a,b,n)** prenant en entrée $f$ une fonction de classe $C^1$ sur l'intervalle $[a,b]$, $df$ la dérivée de $f$ et $n$ le nombre de sous-intervalles à considérer, et qui retourne la valeur approchée $Q_n(f)$ de $I(f)$.

[ ]:
```
def formulecomposite(f,df,a,b,n):
    h=(b-a)/n
    Q= 14*(f(a)+f(b)) + h*(df(a)-df(b))
    for k in range(1,n):
        Q+= 28*f(a+k*h)
    for k in range(0,n):
        Q+= 32*f(a+(k+0.5)*h)
    return (h/60)*Q
```

Question 2 :

On considère dans la suite la fonction $f$ définie sur $I = [\frac{1}{e}, e]$ par

$$f(x) = \frac{ln(x)}{1 + x^2}$$

Question 2 a) :

En évaluant sur $I$ la fonction $f$ en 100 points, écrire les instructions qui permettent de tracer la courbe de $f$ en bleu.

[ ]:
```
import matplotlib.pyplot as plt
f=lambda t:np.log(t)/(1+t**2)
t=np.linspace(1/np.e),np.e,100)
plt.plot(t,f(t),'b')
plt.grid(True)
plt.xlabel('axe des abscisses')
plt.ylabel('axe des ordonnées')
plt.title('la courbe de f')
```

Question 2 b)

Calculer la valeur numérique de l'erreur d'intégration $E_Q(f)$ de la formule composite $Q_{30}(f)$ $(n = 30)$ sachant que $I(f) = 0$.

[ ]:
```
x=sp.symbols('x')
f=lambda x:sp.log(x)/(1+x**2)
df=sp.Lambda(x,sp.diff(f(x),x)) # expression symbolique de  df'
a=1/sp.exp(1)
b=sp.exp(1)
Q_30=formulecomposite(f,df,a,b,30).evalf()
print('E_Q(f)=',abs(Q_30-0))
```

Question 2 c)

Compléter les instructions de la fonction nommée **nombreintervalles(f,df,a,b,epsilon)** prenant en entrée une fonction $f$ de classe $C^1$ sur $[a, b]$, $df$ la dérivée de $f$, et la tolérance *epsilon* et qui retourne le nombre de sous intervalles nécessaire $n$ associé à une tolérance *epsilon* près.

[ ]:
```
def nombreintervalles(f,df,a,b,epsilon):
    n=1
    E=np.abs(formulecomposite(f,df,a,b,n)-0)
    while E > epsilon:
        n+=1
        E=abs(formulecomposite(f,df,a,b,n)-0)
    return n
```

Question 2 d)
Ecrire un script qui permet de tracer le nombre de sous intervalles n en fonction de la tolérance $epsilon$, en considérant $epsilon \in \{10^{-3}, 10^{-5}, 10^{-7}, 10^{-9}\}$.

[ ]:
```
import matplotlib.pyplot as plt
epsilon = (1/10)**np.arange(3,10,2)
N_iter=[]
for eps in epsilon:
    N_iter.append(nombreintervalles(f,df,a,b,eps))
plt.grid(True)
plt.plot(epsilon,N_iter,'ro-')
plt.xlabel('epsilon ')
plt.xscale('log')
plt.ylabel('Nombre de sous intervalles')
plt.title('Nombre de sous intervalles en fonction de epsilon')
```

Question 2 e)
Déduire la valeur du pas de discrétisation maximal $h^*$ à considérer pour avoir une erreur d'intégration de l'intégrale $I(f)$ par la formule de quadrature $Q_n(f)$, inférieure à $10^{-7}$.

Graphiquement : pour epsilon $= 10^{-7}$, on aura $n = 15$ Ceci donne $h^* = \frac{(b-a)}{n} =$ 0.1566934924858402

Question 3 :

On s'intéresse dans la suite à étudier le degré de précision de la formule de quadrature élémentaire $Q_1(f)$, (pour $n = 1$).
Rappelons qu'une formule de quadrature est dite de degré de précision $d$, si elle est exacte pour tout polynôme $P_k(x) = x^k$, avec $0 \le k \le d$, et non exacte pour $P_{d+1}(x) = x^{d+1}$.
Autrement dit, elle est de degré de précision $d$ si l'erreur de quadrature est nulle pour les polynômes $P_k$ avec $0 \le k \le d$ et elle est non nulle pour le polynôme $P_{d+1}$.

Question 3 a)

Compléter les pointillés de la fonction nommée **erreurquadrature(a,b,n,k)** prenant en entrée $a$ et $b$ les deux extrémités de l'intervalle $[a, b]$, $n$ le nombre de sous-intervalles de discrétisation, et $k$ le degré du polynôme $P_k(x) = x^k$, et qui retourne la valeur de l'erreur de quadrature.

[ ]:
```
def erreurquadrature(a,b,n,k):
    x=sp.symbols('x')
    P=lambda x:x**k
    dP=sp.Lambda(x,sp.diff(P(x),x))
    I1=sp.integrate( P(x) ,(x,a,b) ).evalf()
    I2=formulecomposite(P,dP,a,b,n).evalf()
    Erreur=abs(I2-I1)
    return Erreur
```

Question 3 b)

Compléter la fonction nommée **degrépécision(a,b,n)** prenant en entrée $a$ et $b$ les deux extrémités de l'intervalle $[a, b]$ et $n$ le nombre de sous-intervalles de discrétisation, qui retourne le degré de précision de $Q_n(f)$.

[ ]:
```
def degréprécision(a,b,n):
    k=0
    Erreur=erreurquadrature(a,b,n,k)
    while Erreur==0:
        k=k+1
        Erreur=erreurquadrature(a,b,n,k)
    return k-1                          # c'est le degré de précision d
```

Question 3 c)

Pour cette question, on prend $a = -1$ et $b = 0$. Déduire le degré de précision de la formule de quadrature élémentaire $Q_1(f)$.

[ ]:
```
a=-1
b=0
degréprécision(a,b,1)
```

Le degré de précision de la formule vaut 5.