Réflexe :
Importer les librairies nécessaires au calcul scientifique.

Entrée[1]:
```
1 import numpy as np
2 import math
```

TP Python 12 : Approximation d'intégrales

Correction

Partie 0 : Préliminaires

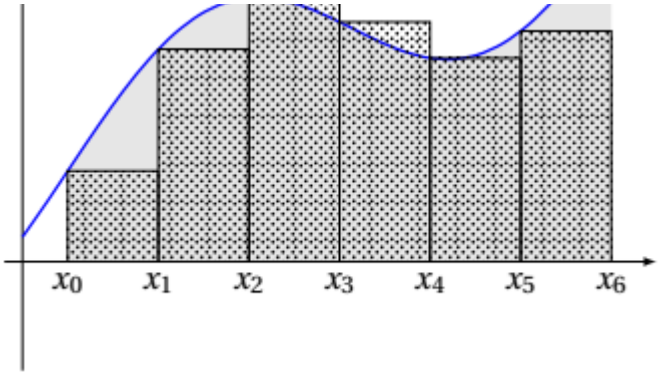
À vous de jouer
Implémenter une fonction python `f` prenant en argument un réel x et renvoyant le réel $\sqrt{1-x^2}$. Cette fonction sera utilisée à plusieurs reprises dans ce TP : il est inutile de l'implémenter de nouveau et vous pouvez tout à fait l'appeler à l'intérieur d'une autre fonction!

Entrée[2]:
```
1 def f(x):
2     return math.sqrt(1-x**2)
```

Partie I : Méthode des rectangles

Lorsqu'une fonction est continue sur un segment $I=[a,b]$, on décompose I en subdivision de longueur $\frac{b-a}{n}$:
$$\left[a, a+\frac{b-a}{n}\right], \left[a+\frac{b-a}{n}, a+2\frac{b-a}{n}\right] \dots, \left[a+k\frac{b-a}{n}, a+(k+1)\frac{b-a}{n}\right]$$

On calcule alors la somme des aires des rectangles inférieurs (ou supérieurs) :
pour l'intervalle
$$\left[a+k\frac{b-a}{n}, a+(k+1)\frac{b-a}{n}\right],$$
on prend le rectangle de hauteur $f\left(a+k\frac{b-a}{n}\right)$ (rectangle inférieur), ou $f\left(a+(k+1)\frac{b-a}{n}\right)$.



Lorsque n tend vers $+\infty$, l'aire obtenue, si la fonction est continue, tend vers l'intégrale de la fonction sur le segment.

On peut donc appliquer la méthode des rectangles, c'est-à-dire le calcul des sommes de Riemman, pour déterminer une valeur approchée de l'intégrale.

À vous de jouer

1. Implémenter la méthode des rectangles afin de calculer une valeur approchée de $\int_0^1 \sqrt{1-x^2}\mathrm{d}x$.
2. Quelle est la valeur de l'intégrale lorsque l'on subdivise l'intervalle $[0, 1]$ avec un pas de $n = 100$?

Entrée[3]:

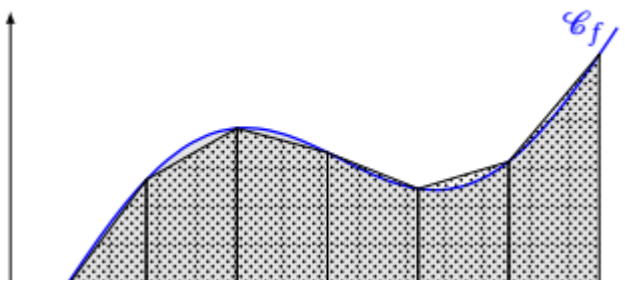
```
def rectangle(a,b,n):
    inf=0
    for i in range(n):
        inf = inf+(b-a)/n*f(a+i*(b-a)/n)
    return inf

print("La méthode des rectangles, pour n =",100,"points, donne ur
```

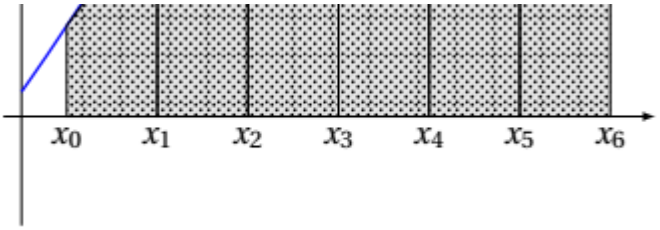
```
La méthode des rectangles, pour n = 100 points, donne une valeur ap
prochée de l'intégrale égale à  0.7901042579447618
```

Partie II : Méthode des trapèzes

On approche l'intégrale $\int_a^b f(t)\mathrm{d}t$ par les sommes :

$$\frac{b-a}{n}\sum_{k=0}^{n-1}\frac{f\left(a+k\frac{b-a}{n}\right)+f\left(a+(k+1)\frac{b-a}{n}\right)}{2} = \frac{1}{2}(S_n(f)+S'_n(f))\,.$$





On peut montrer que, si f est de classe C^2 sur $[a;b]$, alors

$$\forall n \in \mathbb{N}^*, \quad \left|\frac{1}{2}(S_n(f)+S'_n(f)) - \int_a^b f(t)\mathrm{d}t\right| \leq \frac{(b-a)^3}{12n^2} \max_{[a:b]} |f''|.$$

Ainsi, la suite converge vers l'intégrale avec une vitesse $\frac{1}{n^2}$, ce qui est plus efficace que la méthode des rectangles, pour laquelle la suite converge vers l'intégrale avec une vitesse $\frac{1}{n}$.

À vous de jouer

1. Implémenter la méthode des rectangles afin de calculer une valeur approchée de $\int_0^1 \sqrt{1-x^2}\mathrm{d}x$.
2. Quelle est la valeur de l'intégrale lorsque l'on subdivise l'intervalle $[0,1]$ avec un pas de $n = 100$?

Entrée[4]:

```
def trapeze(a,b,n):
    inf=0
    sup=0
    for i in range(n):
        inf = inf+(b-a)/n*f(a+i*(b-a)/n)
        sup = sup+(b-a)/n*f(a+(i+1)*(b-a)/n)
    return 1/2*(inf+sup)

print("La méthode des rectangles, pour n =",100,"points, donne ur
```

```
La méthode des rectangles, pour n = 100 points, donne une valeur ap
prochée de l'intégrale égale à  0.7851042579447618
```

Remarques

- Remarquons que, la valeur approchée obtenue par la méthode des trapèzes est meilleure que celle obtenue avec la méthode des rectangles. En effet ,
$$\int_0^1 \sqrt{1-x^2}\mathrm{d}x = \frac{\pi}{4} \simeq 0,7853981634.$$
- Au lieu de calculer pour un rang n, on peut utiliser les majorations vues pour obtenir une valeur approchée à une certaine précision, avec une boucle `while` .

(HP) Partie III : Une introduction à la

méthode de Monte-Carlo

methode de Monte-Carlo? https://www.f-legrand.fr/scidoc/docmml/numerique/montecarlo/integrales/integrales.html (https://www.f-legrand.fr/scidoc/docmml/numerique/montecarlo/integrales/integrales.html) citez Legrand si pompé

Réflexe :
Importer la librairie nécessaire à la simulation de variables aléatoires.

Entrée[5]:
```
# La bibliothèque numpy étant déjà importée; il ne nous reste qu
import numpy.random as rd
```

La méthode de Monte-Carlo est une méthode **probabiliste** d'approximation d'une intégrale. Elle est hors programme, mais va vous permettre d'utiliser des outils que vous avez déjà vus au cours des TP précédents.

Considérons une fonction f continue sur un segment $[a, b]$. Soit X une variable aléatoire réelle suivant une loi uniforme sur le segment $[a, b]$. Soient N échantillons $x_1, \dots x_N$ de cette variables aléatoire. L'évaluation de l'intégrale par la méthode de Monte-Carlo consiste (dans sa forme élémentaire) à calculer la somme suivante :

$$S_N = \frac{b-a}{N} \sum_{i=1}^{N} f(x_i).$$

Il s'agit de la même formule que celle de la méthode des rectangles, avec des points répartis aléatoirement dans l'intervalle $[a, b]$. Lorsque le nombre N d'échantillons de la variable aléatoire X devient très grand, la somme S_N s'approche de l'intégrale $\int_a^b f(x)\,\mathrm{d}x$.

À vous de jouer

1. A l'aide de la fonction `random` , comment pourriez-vous simuler un nombre **réel** x appartenant à un segment $[a, b[$?

→ Cliquez-ici pour la réponse ←

À vous de jouer

2. A l'aide de question précédente, écrire une fonction `echantillons` prenant en argument un entier naturel non nul N et deux réels a et b (avec $a < b$), et renvoyant la liste contenant N réalisation d'une variable aléatoire suivant une loi uniforme sur le segment $[a, b]$.

Entrée[6]:

```
def echantillons(N,a,b):
    liste =[]
    for k in range(N):
        liste.append(a+(b-a)*rd.random())

    return liste
```

À vous de jouer

3. On considère toujours la fonction $f : x \longmapsto \sqrt{1-x^2}$ sur l'intervalle $[0, 1]$ (ici donc $a = 0$ et $b = 1$!). Écrire une fonction `MonteCarlo` qui étant donné le nombre $N \in \mathbb{N}^*$ renvoie la somme S_N mentionnée ci-dessus.

   Exécuter cette fonction pour $N = 100$, $N = 1000$ et $N = 10000$.

Entrée[9]:

```
def MonteCarlo(N):
    S = 0
    X = echantillons(N,0,1)
    for k in range(len(X)):
        S = S + f(X[k])

    S = (1/N)*S

    return S

print("La somme S_N vaut",MonteCarlo(100),"lorsque N =",100)
print("La somme S_N vaut",MonteCarlo(1000),"lorsque N =",1000)
print("La somme S_N vaut",MonteCarlo(10000),"lorsque N =",10000)
```

```
La somme S_N vaut 0.7398458607866456 lorsque N = 100
La somme S_N vaut 0.7847108952031101 lorsque N = 1000
La somme S_N vaut 0.7865084210007547 lorsque N = 10000
```

À vous de jouer

4. Que doit modifier dans la fonction `MonteCarlo` précédente si cette fois-ci il est question de déterminer l'intégrale d'une autre fonction g définie sur un segment $[a, b]$ quelconque?

→ Cliquez-ici pour la réponse ←

À vous de jouer

5. On cherche à tester l'efficacité de la méthode de Monte-Carlo pour le calcul de l'intégrale $\int_0^1 \sqrt{1-x^2}\mathrm{d}x = \frac{\pi}{4} \simeq 0,7853981634$. Exécutez le programme suivant : que fait-il? Commentez chacune des lignes du programme à l'aide de la commade dièse #.

Entrée[25]:

```
# On fixe epsilon positif
eps = 1e-3
# On va vouloir déterminer combien de valeurs aléatoires $N$ pris
# pour approcher la valeur pi/4 à epsilon près.

N=1
val_approx = MonteCarlo(N)
val_exacte = np.pi/4

#Tant qu'en valeur absoblue, la différence est plus grande que ep
while np.absolute(val_approx - val_exacte)>=eps :
    # on incrémente le nb d'échantillon de +1
    N = N+1
    # et on met à jour la valeur approchée de l'intégrale
    val_approx = MonteCarlo(N)

# Une fois qu'on est sorti de la boucle, on affiche :
print("Il faut ",N," valeurs prises aléatoirement dans [0,1]$ pou
print("La valeur approchée de l'intégrale de f sur [0,1] vaut",va
```

```
Il faut  27  valeurs prises aléatoirement dans [0,1]$ pour approche
r à  0.001 près l'intégrale de f.

La valeur approchée de l'intégrale de f sur [0,1] vaut 0.7849795322
378308
```

À vous de jouer

5. Pourquoi, lorsque l'on exécute le programme précédent, le nombre N varie? Que proposeriez-vous afin de déterminer rigoureusement le nombre de points $(x_i)_{i=1}^n$ de $[0,1]$ nécessaire à ce que l'intégrale approchée soit proche $\frac{\pi}{4}$ à 0.001 près?