Corrigé du Devoir à la Maison n°1

Partie A. Méthode de Monte-Carlo

1. Dans le programme ci-dessous on choisit aléatoirement N points, et le compteur c compte combien d'entre eux sont à l'intérieur du cercle trigonométrique. Ainsi $\frac{c}{N}$ est une approximation de $\frac{\pi}{4}$, et donc $4\frac{c}{N}$ est une approximation de π.

```
from random import random

N=int(input("Nombre de points : "))

c=0    # compteur
for k in range(N):
    x=random()
    y=random()
    if x**2+y**2<1:
        c=c+1

print("Approximation obtenue :",4*c/N)
```

2. On obtient une approximation de π avec une erreur de l'ordre de $\frac{1}{\sqrt{N}}$. Par exemple pour $N = 10^6$ on obtient des valeurs comprises entre 3,14 et 3,143, c'est-à-dire une erreur de l'ordre de 10^{-3}.

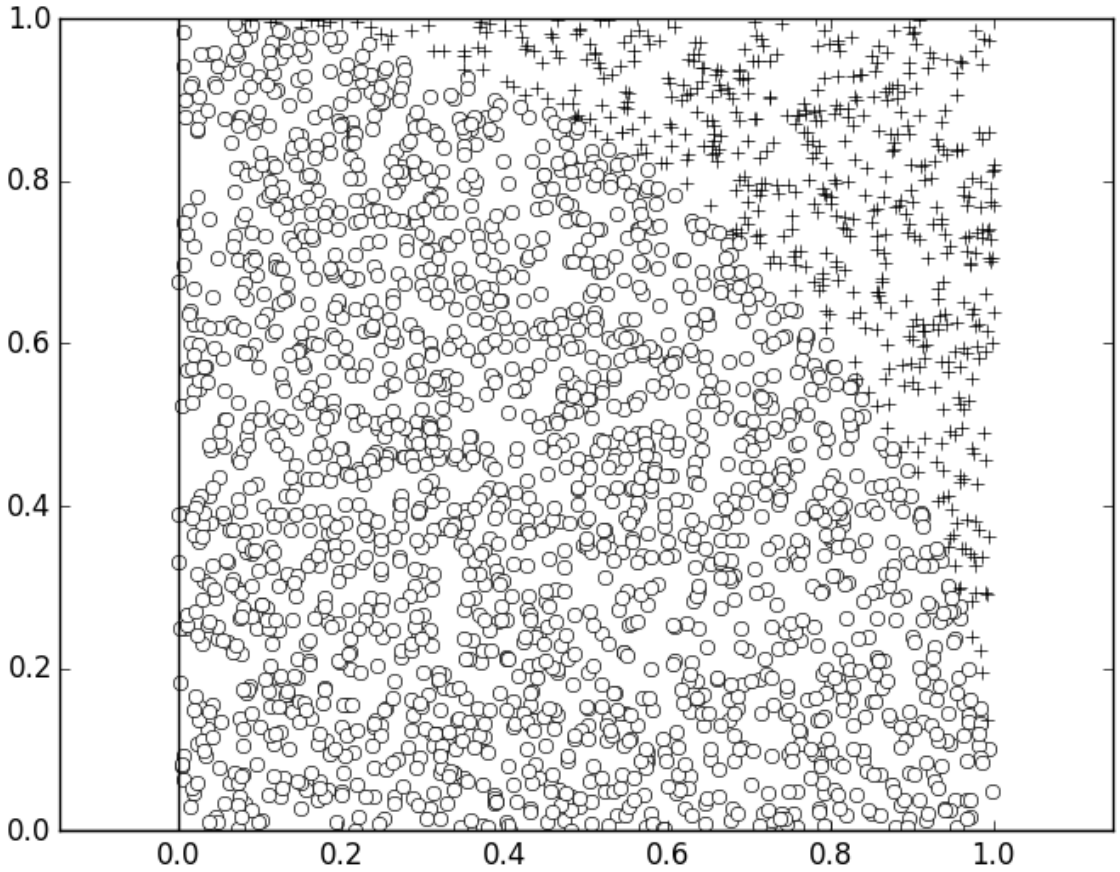
   Des valeurs comme $N = 10^8$ demandent plus de temps, il est donc délicat d'espérer une approximation à moins de 10^{-4} près.

3. On ajoute l'importation du sous-module `matplotlib.pyplot`, l'affichage des points en bleu ou en rouge, puis les lignes finales pour l'affichage du graphique.

```
from matplotlib.pyplot import *
from random import random

N=int(input("Nombre de points : "))

c=0    # compteur
for k in range(N):
    x=random()
    y=random()
    if x**2+y**2<1:
        c=c+1
        plot([x],[y],'b+')
    else:
        plot([x],[y],'r+')
```

```
print("Approximation obtenue :",4*c/N)

# Axes :
axhline(color='k')
axvline(color='k')

# Repère orthonormé :
axis("equal")

# Fenêtre à afficher :
axis([0,1,0,1])

# Affichage :
show()
```

4. En remplaçant les croix pour les points à l'intérieur du cercle trigonométrique on obtient, pour $N = 2500$, le graphique suivant :



Partie B. La fonction arc-tangente

5. On écrit une fonction dans laquelle une variable d'accumulation S est définie, elle reçoit les $(-1)^k \frac{x^{2k+1}}{2k+1}$ pour k allant de 0 à n.

```
def Somme(n,x):
    """Calcule S_n(x)"""
    S=0
    for k in range(n+1):
        S=S+(-1)**k*x**(2*k+1)/(2*k+1)
    return S
```

6. On remplace la boule **for** par une boucle **while**. Il faut donc initialiser k à 0 et l'incrémenter à chaque boucle.

   Le monôme $\frac{x^{2k+1}}{2k+1}$ est aussi stocké dans une variable m, ce qui permet d'utiliser sa valeur dans la condition d'arrêt de la boucle : en effet on souhaite calculer les valeurs successives de S_n jusqu'à ce que ce monôme soit de valeur absolue inférieure à ε (noté `eps`).

```
def Arctan(x,eps):
    """Calcule arctan(x) à eps près."""
    S=0
    k=0
    m=x
    while abs(m)>eps:
        m=x**(2*k+1)/(2*k+1)
        S=S+(-1)**k*m
        k=k+1
    return S
```

7. On exécute la commande suivante :

```
>>> pi1=4*Arctan(1,1e-6/4)
```

   Comme la fonction `Arctan` renvoie une approximation de $\frac{\pi}{4}$ avec une erreur de l'ordre de $\frac{1}{4}10^{-6}$, alors `pi1` reçoit une approximation de π avec une erreur de l'ordre de 10^{-6}.

   On obtient 3,1415931535894743 au lieu de 3,141592653589793.

   On peut demander une précision plus fine (comme 10^{-7}) mais l'exécution de la fonction dure un certain temps.

8. Les commandes suivantes :

```
>>> x1=Arctan(1/5,1e-16/32)
>>> x2=Arctan(1/239,1e-16/8)
>>> pi2=16*x1-4*x2
```

   donnent en un temps très court l'approximation désirée : 3.141592653589793.

Partie C. Approximation rationnelle de π

9. Il suffit d'écrire une simple boucle.

```
from math import sin

N=int(input("Entrez la limite supérieure du dénominateur : "))

for p in range(1,N+1):
    print(abs(sin(p)))
```

10. On utilise la variable `sm` qui contient la valeur minimale atteinte par $|\sin(p)|$. Lorsque, durant la boucle, une valeur plus petite est obtenue, on la stocke dans la variable `sm`, puis on stocke l'entier p pour lequel cette valeur a été obtenue dans la variable `pm`, et enfin on calcule la valeur de q.

```
from math import sin,pi

N=int(input("Entrez la limite supérieure du dénominateur : "))

sm=1
for p in range(1,N+1):
    s=abs(sin(p))
    if s<sm:
        sm=s
        pm=p
        q=round(pm/pi)

print("Meilleure approximation : ",pm,"/",q,sep="")
print("Erreur :",abs(pm/q-pi))
```

11. On obtient les approximations successives suivantes :

```
3/1           Erreur : 0.14159265358979312
22/7          Erreur : 0.0012644892673496777
333/106       Erreur : 8.32196275291075e-05
355/113       Erreur : 2.667641894049666e-07
103993/33102  Erreur : 5.778906242426274e-10
```

Ainsi l'approximation $\frac{355}{113}$ est très correcte, son erreur est de l'ordre de 3×10^{-7}. Elle a été découverte par le mathématicien chinois Zu Chongzhi au V$^{\text{ème}}$ siècle après J.C.

Pour en obtenir une plus proche il faut tester les entiers jusqu'à plus de 100 000 !