Université d'Evry Val d'Essonne
L1 SPI - Année universitaire 2018/2019

Modélisation et Simulation - EC222

TP3 - Méthodes de Monte-Carlo

Consignes :

- pour toute question ou remarque : gaetan.hello@univ-evry.fr

1 Approximation du nombre π par une approche de Monte-Carlo

```
#=== Import des bibliotheques ============================================
import math                          #librairie de fct mathematiques
import random                        #librairie fct aleatoires
import matplotlib.pyplot as pyplot #librairie pour afficher des courbes

#=== Nettoyage ===========================================================
pyplot.close('all')

#=== Parametres numeriques ===============================================
nMC=1000 #nb de tirages de Monte-Carlo

#=== Algo de Monte Carlo =================================================
compteur=0
xIn=[] #liste vide
yIn=[]
xOut=[]
yOut=[]

for i in range(nMC):
    x=random.random()
    y=random.random()
    if (x-0.5)**2+(y-0.5)**2-0.5**2<0: #dans le cercle
        compteur+=1
        xIn.append(x) #ajoute x a la liste xIn
        yIn.append(y)
    else:
        xOut.append(x)
        yOut.append(y)

piMC=4*float(compteur)/float(nMC)
print("piReference= "+str(math.pi))
print("piMonteCarlo="+str(piMC))

#====== Affichage ========================================================
pyplot.figure(1)
plotPtsIn=pyplot.plot(xIn,yIn)
pyplot.setp(plotPtsIn,'marker','o','linestyle','none','markerfacecolor',[0,1,0],'
    markeredgecolor',[0,0,0])
plotPtsOut=pyplot.plot(xOut,yOut)
pyplot.setp(plotPtsOut,'marker','o','linestyle','none','markerfacecolor',[0,0,1],'
    markeredgecolor',[0,0,0])
pyplot.xlim(0.,1.)
pyplot.ylim(0.,1.)
pyplot.gca().set_aspect('equal')
titre='nTirages='+str(nMC)+', piMonteCarlo='+str(piMC)
pyplot.title(titre)
pyplot.savefig("test.png",bbox_inches='tight')
```

Listing 1 – Calcul de π par une approche de Monte-Carlo

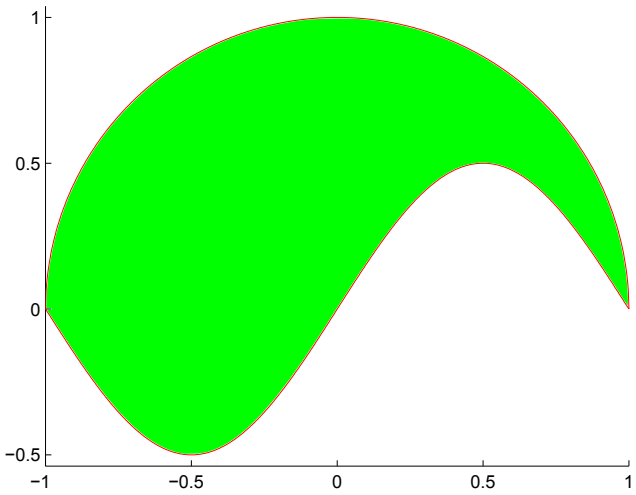
Exécuter et analyser le code précédent.

2 Calcul de l'aire et centre de gravité d'une surface complexe

On définit la surface S du plan comme :

$$S = \{(x, y) \in \mathbb{R}^2,\ x \in [-1, 1], \frac{1}{2}\sin(\pi\, x) \leq y \leq \sqrt{1 - x^2}\} \tag{1}$$

Cette surface peut ainsi avoir comme représentation graphique :



A l'aide de procédures basées sur une approche de Monte-Carlo :

2.1

Tracer les deux courbes délimitant la surface.

2.2

Déterminer une approximation de l'aire de cette surface.

2.3

Etudier l'influence du nombre de tirages de Monte-Carlo sur la variabilité de cette approximation.

2.4

Déterminer une approximation de la position du centre de gravité de cette surface. On rappellera à cette fin que le centre de gravité (x_G, y_G) d'une surface S se définit analytiquement selon :

$$\begin{aligned} x_G &= \frac{\int_S x\, dS}{\int_S dS} \\ y_G &= \frac{\int_S y\, dS}{\int_S dS} \end{aligned} \tag{2}$$

Dans le cadre d'une approche de Monte-Carlo, si les points $(x_i, y_i)_{i=1,\ldots,N}$ sont situés à l'intérieur de la surface, les coordonnées du centre de gravité peuvent s'approximer selon :

$$\begin{aligned} x_G &\simeq \frac{\sum_{i=1}^{N} x_i}{N} \\ y_G &\simeq \frac{\sum_{i=1}^{N} y_i}{N} \end{aligned} \tag{3}$$

3 Jeu de dés simple

On souhaite déterminer la probabilité que la somme de deux valeurs aléatoires fournies par deux dés à six faces vaille 8. On se propose d'employer une méthode exacte de dénombrement et une méthode aléatoire approchée de type Monte-Carlo :

```
import random

# calcul de la probabilite exacte par denombrement
compteurCasTotal=0
compteurCasVrai=0
for i in range(6):
    for j in range(6):
        compteurCasTotal+=1
        if i+j+2==8:
            compteurCasVrai+=1
probaExacte=compteurCasVrai/compteurCasTotal

# calcul approche de la probabilite par Monte-Carlo
nMC=10000
compteurMC=0
for i in range(nMC):
    d1=random.randint(1,6)
    d2=random.randint(1,6)
    if d1+d2==8:
        compteurMC+=1

probaMC=compteurMC/nMC

print(probaExacte)
print(probaMC)
```

Listing 2 – Calcul d'une probabilité par une approche de Monte-Carlo

Exécuter et analyser le code précédent.

4 Jeu de dés - Risk

Le jeu de société "Risk" repose sur des affrontements entre adversaires faisant intervenir des jets de dés à 6 faces. Le joueur attaquant (A) lance ainsi 3 dés tandis que le joueur défenseur (D) en lance 2. On note alors a_1, a_2 (avec $a_1 \geq a_2$) les deux meilleures valeurs obtenues par A et d_1, d_2 (avec $d_1 \geq d_2$) celles obtenues par D. On compare ensuite a_1 avec d_1 ainsi que a_2 avec d_2 en suivant la règle : si $a_i > d_i$ D perd 1 point, sinon A perd un point.

4.1

Ecrire sur le papier l'algorithme qui détermine le nombre de points perdus par A pour un lancer de dés.

4.2

A l'aide d'une approche de Monte-Carlo, estimer les probabilités que A perde 0,1 ou 2 point(s).

En langage python, un nombre entier entre 1 et 6 inclus peut être généré grace à la commande *random.randint(1,6)* (bibliothèque *random* à importer préalablement). Etant donné une liste d'entiers nommée *maListe* (par exemple *[5,2,4]*, les élements de celle-ci peuvent être rangés en ordre croissant en invoquant la méthode *maListe.sort()* (l'exemple précédent devenant *[2,4,5]*)

5 BONUS : Jeu de dés - Risk

Dans le contexte du jeu "Risk" précédemment décrit, déterminer la meilleure stratégie pour le défenseur si celui-ci a désormais le choix de lancer un ou deux dés. Vous pourrez par exemple définir pour chaque jets de l'attaquant s'il vaut mieux lancer un ou deux dés pour le défenseur.

Si le défenseur suit la stratégie optimale, quelles sont alors probabilités que A perde 0,1 ou 2 point(s).