TD d'ITC : SIMULATION (CORRIGÉ) MP 2024-2025

```
# TD DE SIMULATION INFORMATIQUE 2024-2025

# simulation de Bernoulli, d'une loi binimale comme somme de Bernoulli
from random import *
from math import *
from matplotlib.pyplot import *
from numpy import linspace


def  u(): # simule un nombre réel entre 0 et 1
   return(random())

def mystere_u(x,nb): # Que fait cette fonction?
    c=0
    for j in range(nb):
        tirage=u()
        if tirage<=x:
            c=c+1
    return c/nb


def trace_u(nb): # Que fait cette fonction?
   LX=linspace(-1,2,500)
   print(len(LX))
   LY=[mystere_u(p,nb) for p in LX]
   plot(LX,LY)
   show()

# Q_1 ########################################?
def  ber(p):   #tirage de Bernoulli  b(p)=B(p)
   tirage=random()
   if tirage<p:
      return 1
   else:
      return 0

# Q_2 ########################################

def  bino(n,p): # S simule la somme de n épreuve de Bernoulli B(p)
  somme=0
  for i in range(n):
     somme=somme+ber(p)
  return somme

def binoT(k,n,p):  #binomiale théorique bino(n,p)=B(n,p)
   return factorial(n)/factorial(k)/factorial(n-k)*p**k*(1-p)**(n-k)

### Vérifiaction expérimentale(début)
def listebinoT(n,p): # liste théorique des probabilités de B(n,p)
   c=(n+1)*[0]
   for k in range(n+1):
      c[k]=binoT(k,n,p)
   return c
```

```
def Verification(n,p,nb): # on simule nb tirage de S puis on compte le nb de fois
                        # où S=k et on renvoie la liste des différences entre
                        # les proba simulées et théoriques : P(S=k),k=0..n
  L=nb*[0]
  for i in range(nb):
     L[i]=bino(n,p)
  c=(n+1)*[0]
  for i in range(nb):
    for k in range(n+1):
       if L[i]==k:
          c[k]=c[k]+1
  for k in range(n+1):
      c[k]=c[k]/nb
  cc=listebinoT(n,p)
  return [c[i]-cc[i] for i in range(n+1)]

### Vérifiaction expérimentale(fin)

# Q_3 ##########################################

def geo(p):
   tirage=random()
   essai=1
   while  tirage>=p:
      tirage=u()
      essai+=1
   return essai

# Q_4 ##########################################
def probageoT(p,k):  #binomiale théorique G(p)
   return p*(1-p)**(k-1)

# Q_5 ##########################################
def probageo(p,k,nb):
   s=0
   for n in range(nb):
      tirage=geo(p)
      if tirage==k:
         s=s+1
   return s/nb

# Q_6 ##########################################
def espgeo(p,nb):
   s=0
   for n in range(nb):
      s=s+geo(p)
   return s/nb

def espgeo2(p,nb): # espérance de X^2
   s=0
   for n in range(nb):
      s=s+geo(p)**2
   return s/nb
```

```
def Vgeo(p,nb):
   return espgeo2(p,nb)-espgeo(p,nb)**2

# Q_7 ########################################
magrille=[1,2,3,4,5,6]

def loto(): # le tirage du jour
   tirage=6*[0]
   for i in range(6):
      alea=randint(1,49)
      while alea in tirage:
         alea=randint(1,49)
      tirage[i]=alea
   tirage.sort()
   return(tirage)

def jeuloto():
    c=0
    test=False
    while test==False:
        c+=1
        L=loto()
        if L==magrille:
            test=True
        if c%1000000==0:
            print(c)
    return(c/365 ,'années')


# Q_8 EXERCICE MYSTERE  ##########################################

def Xm():
    L=[randint(1,6),randint(1,6),randint(1,6)]
    c=0
    for i in range(3):
        if L[i]==2:
            c=c+1
    if c==0:
        return -1
    else:
        return c

def Ym(nb):
    s=0
    for i in range(nb):
        s=s+Xm()
    return [s/nb,'valeur théorique:',-17/216]

# Fin EXERCICE MYSTERE  ##########################################




# VIGNETTE - VIGNETTE - VIGNETTE - VIGNETTE -
```

```
# Q_9 ########################################
def collec():
    nbt=0 # nombre de tablettes
    nbj=0 # nombre d'images de joueurs
    C=15*[0] # tableau des images
    while nbj<15:

        nbt+=1 # une tablette de plus achetée
        aux=randint(0,14)
        if C[aux]==0:
            nbj+=1 # un joueur de plus dans la collec
            C[aux]=1 # on colle l'image dans son tableau

    return(nbt) # retourne le nb de tablettes achetées pour avoir les 15 images

def espcollec(nb): # espérance simulée
    s=0
    for i in range(nb):
        s=s+collec()
    return(s/nb)



def espcollecT(): # espèrance théorique
    s=0
    for i in range(1,16):
        s=s+1/i
    return(15*s) #

def espcollec2(nb): # espérance de X^2 simulée
    s=0
    for i in range(nb):
        s=s+collec()**2
    return(s/nb)

def varcollec(nb): # variance simulée
    return(espcollec2(nb)-espcollec(nb)**2)

def varcollecT(): # variance  théorique
    s=0
    for i in range(1,16):
       p=(15-i+1)/15
       q=1-p
       s=s+q/p**2
    return(s)

#i tem Même question avec la coupe du monde soit 32*26=832 joueurs.
# il faut en moyenne 6074.971924 images!!!

# Q_10 et Q_11 : BANACH ######################################
def Xb(N):
    poche1=N
    poche2=N
    while poche1>=0 and poche2>=0:
        tir=random()
        if tir<1/2: #on choisit la poche1
```

```
            if poche1==0:
                return poche2
            else:
                poche1=poche1-1
        else:
            if poche2==0:
                return poche1
            else:
                poche2=poche2-1


def espXb(N,nb):
    s=0
    for i in range(nb):
        s=s+Xb(N)
    return [s/nb,espT(N)]



def probaXb(N,k,nb):
    c=0
    for i in range(nb):
        if Xb(N)==k:
            c=c+1
    return [c/nb,loi(N,k)]

def bino(n,p):
    if p==0:
        return 1
    else:
        return bino(n-1,p-1)*n//p

def loi(N,k): #Valeur exacte de P(X=k)
    return bino(2*N-k,N)/2**(2*N-k)

def espT(N): #Valeur exacte
    return  (2*N+1)*bino(2*N,N)/4**N-1 #(exercice difficile)

# Fin  BANACH ###############################################

from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import numpy as np
from random import random

# TENNIS - TENNIS - TENNIS - TENNIS - TENNIS -
# Q_12 ######################################
def coup(p): # renvoie 'A' avec une proba égale à p
     aux=random()
     if aux<=p:
         return('A')
     else:
         return('B')
# Q_13 ######################################
def test(n,p): # test statistiquement la fonction coup()
    nbA=0;nbB=0
    for i in range(n):
```

```
        aux = coup(p)
        if aux=='A':
            nbA+=1
        else:
            nbB+=1
    return(nbA,nbB,nbA/n)

# Q_14 ########################################
def jeu(p): #renvoie 'A' si 'A' gagne le jeu et 'B' si 'B' gagne
    points_A=0;points_B=0
    for i in range(6):
        aux=coup(p)
        if aux=='A':
            points_A +=1
        else:
            points_B +=1
        if points_A==4:
            return('A')
        if points_B==4:
            return('B')
    while abs(points_B-points_A)<=1:
        aux=coup(p)
        if aux=='A':
            points_A +=1
        else:
            points_B +=1
     #   print(points_A,points_B)
    #print('jeu')
    if points_A>points_B:
        return('A')
    else:
        return('B')

# Q_15 ########################################
def f(p,nb): # renvoi statistiquement la proba. de  jeu(p)
    #n=5000
    nbA=0;nbB=0
    for i in range(nb):
        aux = jeu(p)
        if aux=='A':
            nbA+=1
        else:
            nbB+=1
    return(nbA/nb)

def fT(p): # valeur de la probabilité exacte de jeu(p)
   return (15*p**4-34*p**5+28*p**6-8*p**7)/(1-2*p+2*p**2)

# Q_16 ########################################
def traceTennis():
   LX=linspace(0,1,500)
   LY=[f(p,5000) for p in LX]
   plot(LX,LY)
   show()
# Fin Q_16 ####################################
##############################################
```

```
# Q_17 et Q_18 MOUVEMENT BROWNIEN
##########################################
from turtle import *
from random import *
reset()
def anglealea(): # simule un angle aléatoire entre 0 et 360
    return(randint(0,360))
up()
goto(0,-200)
down()
circle(200)
up()
goto(0,100)
down()
ht()
L=[]
d=0
nb=0
speed('fastest')
while d<200:
    forward(10)
    left(anglealea())
    d=distance(0,0)
    nb=nb+1
up()
goto(0,-220)
write('le nombre de sauts vaut : '+str(nb),font=("Arial", 18,"normal"))
mainloop()
# Fin  MOUVEMENT BROWNIEN
##########################################

##########################################
# EHRENFEST
##########################################

# Q_19 & Q_20 ############################

from math import *
from random import *
import numpy as np
import matplotlib.pyplot as pp


def nombredeboules(a):
    s=0
    for i in range(len(a)):
        s=s+a[i]
    return s


def transfert(a,b):
    N=len(a)
    i=randint(0,N-1) # on choisit un n° entre 0 et N-1
    if a[i]==1: #on échange la case i des urnes A et B
        a[i]=0
        b[i]=1
    else:
```

```
        a[i]=1
        b[i]=0
    return(a,b)


def nbA(N,t):
    a=N*[1] # on mets toutes les boules dans A
    b=N*[0] # B est vide
    for i in range(t):
        transfert(a,b)
    return(a,b)




def TracerNbA(N,t):
    a=N*[1] # on mets toutes les boules dans A
    b=N*[0] # B est vide
    x=t*[0]
    y=t*[0]
    for i in range(t):
        x[i]=i
        y[i]=nombredeboules(a)
        transfert(a,b)


    pp.plot(x,y)
    pp.show()
    #return(x,y)

#for k in range(25):
 #   print(nbA(7,k))

##########################################
# Fin EHRENFEST
##########################################


##########################################
# MONTE-CARLO
##########################################

# Q_21 ###################################################
def nuage(n):
    plt.clf()
    x=n*[0]
    y=n*[0]
    for i in range(n):
        r=random()
        t=random()*2*np.pi
        x[i]=r*np.cos(t)
        y[i]=r*np.sin(t)
    #MONTE-CARLO
    plt.scatter(x,y,s=5)  #s donne la grosseur des points
    plt.title('Nuage de points avec Matplotlib')
    plt.xlabel('x')
```

```
    plt.ylabel('y')
    plt.axis('equal') # pour avoir des axes orthonorme?s
    plt.show()

def nuagebis(n):
    plt.clf()
    x=n*[0]
    y=n*[0]
    for i in range(n):
        r=np.sqrt(random()) # pour une répartition unfiforme
        t=random()*2*np.pi
        x[i] = r*np.cos(t)
        y[i] = r*np.sin(t)
    plt.scatter(x,y,s=10)
    plt.title('Nuage de points répartis avec Matplotlib')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.axis('equal')
    plt.show()


# Q_22 et 23 ##########################################
def traceCardioide(n):
    plt.clf()
    T=np.linspace(0,2*np.pi,101)
    xt=np.cos(T)*(1+np.cos(T))
    yt=np.sin(T)*(1+np.cos(T))
    cerclext=np.cos(T)*(2.5)
    cercleyt=np.sin(T)*(2.5)

    x=n*[0]
    y=n*[0]

    nb=0

    for i in range(n):
        r=random()*2.5**2
        t=random()*2*np.pi
        x[i]=np.sqrt(r)*np.cos(t)
        y[i]=np.sqrt(r)*np.sin(t)
        if np.sqrt(r)<1+np.cos(t):
            nb+=1
    plt.scatter(x,y,s=1) #s donne la grosseur des points
    plt.title('cardioide')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.axis('equal')
    plt.plot(xt,yt,linewidth=4,color='red')
    plt.plot(cerclext,cercleyt,linewidth=1,color='g')

    plt.show()
    print('Valeur approchée (MonteCarlo) avec ',n,'points : ',
    nb/n*np.pi*2.5**2)
    print('Valeur exacte (avec les intégrales doubles) : ',3*np.pi/2)

    #close() sinon pas de dessin???
```

```
# Q_24 : VOLUME ##########################################

def trace(): #trace l'hyperboloide à une nappe
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    u = np.linspace(0, 2 * np.pi, 100)
    v = np.linspace(-1,1, 100)
    x = np.outer(np.cos(u), np.cosh(v))
    y = np.outer(np.sin(u), np.cosh(v))
    z = np.outer(np.ones(np.size(u)), np.sinh(v))
    ax.plot_surface(x, y, z,  rstride=4, cstride=4, color='g')
    plt.show()

cube=4*4*2;

# Q_25 ##########################################
def al(b): # genere un nb alea entre -b et b
    return -b+2*b*random()


# Q_26 ##########################################
def montecarlo(n):
    nbdedans=0;nbdehors=0;
    for i in range(n):
        x=al(2);y=al(2);z=al(1);
        if (x**2+y**2-z**2)<=1 and z**2<=1:
            nbdedans=nbdedans+1;
        else:
            nbdehors=nbdehors+1;
    return(cube*nbdedans/n,nbdedans,nbdehors)

# Q_27 ##########################################
def vol(nb,max):
    s=0
    for i in range(nb):
        s=s+montecarlo(max)[0]
    return s/nb

# Valeur exacte avec les intégrales triples : 8*Pi/3

##########################################
# Fin MONTE-CARLO
##########################################
```