TD d'ITC : SIMULATION MP 2025-2026

Q_1 : On étudie la fonction x |----> x(1-x) qui est maximum en 1/2 d'où un majorant est 1/4

Q2_ On résoud 1/4/n/10**(-6)<10**(-3) et l'on trouve n=2,5.10**8.

```
from random import *
from math import *
from matplotlib.pyplot import *
from numpy import linspace

-----------------------------Bernoulli-----------------------------------------

# Q_3 ##########################################
def  ber(p):   #tirage de Bernoulli  b(p)=B(p)
   tirage=random()
   if tirage<p:
      return 1
   else:
      return 0

-----------------------------binomiale-----------------------------------------
# Q_4 ##########################################
def  bino(n,p): # S simule la somme de n épreuves de Bernoulli B(p)
  somme=0
  for i in range(n):
     somme=somme+ber(p)
  return somme


-----------------------------géométrique---------------------------------------

# Q_5 ##########################################
def geo(p):
   tirage=random()
   essai=1
   while  tirage>=p:
      tirage=u()
      essai+=1
   return essai


# Q_6 ##########################################
def geoS(p,k,nb):
   s=0
   for n in range(nb):
      tirage=geo(p)
      if tirage==k:
         s=s+1
   return [s/nb,(1-p**(k-1)*p]

# Q_7 ##########################################
def espS(p,nb):
   s=0
   for n in range(nb):
      s=s+geo(p)
   return s/nb

def esp2(p,nb): # espérance de X^2
   s=0
   for n in range(nb):
      s=s+geo(p)**2
   return s/nb

def V(p,nb):
   return esp2(p,nb)-esp(p,nb)**2

------------------------VIGNETTES ----------------------------------------
# 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 tablette achetées pour avoir les 15 images
```

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

Q_10

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

def esp2(nb): # espérance de X^2 simulée
    s=0
    for i in range(nb):
        s=s+collec()**2
    return(s/nb)

def var(nb): # variance simulée
    return(esp2(nb)-parent(nb)**2)

def vart(): # 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)

----------------------------TENNIS-----------------------------------------------------
# Q_11 ##########################################
def coup(p): # renvoie 1 avec une proba égale à p (coup=ber)
     aux=random()
     if aux<=p:
         return(1)
     else:
         return(0)

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

# Q_12 ##########################################
def f(p,nb): # renvoi statistiquement la proba. de  jeu(p)
    c=0
    for i in range(nb):
        c=c+jeu(p)
    return(c/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_13 ##########################################
def traceTennis():
   LX=linspace(0,1,500)
   LY=[f(p,5000) for p in LX]
   plot(LX,LY)
   show()

-----------------------------------SCRUTIN--------------------------------
# Q_14  ##########################################
```

```
from random import *
from turtle import *
from time import *

def depouillement():
    nbA=0
    nbB=0
    for n in range(1000):
        tirage=random()
        if tirage<0.6:
            nbA=nbA+1
        else:
            nbB=nbB+1
        if nbB>=nbA:
            return 0
    return 1 #[1,nbA,nbB]

def proba(nb):
    s=0
    for i in range(nb):
        tirage=depouillement()
        s=s+tirage
    return(s/nb)

# autre simulation du dépouillement:

def listeB(): # le liste des 400 bulletins 'B'
   tirage=400*[None]
   for i in range(400):
      alea=randint(0,999)
      while alea in tirage:
         alea=randint(0,999)
      tirage[i]=alea
   return(tirage)


def depouillementBis():
    LB=listeB()
    V=1000*['A']
    for k in LB:
        V[k]='B'
    nbA=0
    nbB=0

    for n in range(1000):
        tirage=V[n]
        if tirage=='A':
            nbA=nbA+1
        else:
            nbB=nbB+1
        if nbB>=nbA:
            return 0
    return 1 #[1,nbA,nbB]

def probaBis(nb):
    s=0
    for i in range(nb):
        tirage=depouillementbis()
        s=s+tirage
    return(s/nb)


#t0=time()
#print(proba(10010))
#print(time()-t0)


# ##########################################
#   MOUVEMENT BROWNIEN
# ##########################################

# Q_15 ET Q_16  ##########################################
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(0,0)
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"))

################################
EHRENFEST
################################
# Q_17 ET Q_18 ##########################################
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))


# ########################################
#   MONTE-CARLO
# ########################################
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import numpy as np
from random import random

# Q_19 ##########################################
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=1)  #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 orthonormés
    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=1)
    plt.title('Nuage de points répartis avec Matplotlib')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.axis('equal')
    plt.show()


# Q_20 et Q_21 ##########################################

def traceTrefle(n):
    plt.clf()
    T=np.linspace(-np.pi,np.pi,101)
    xt=np.cos(T)*(np.sin(2*T))
    yt=np.sin(T)*(np.sin(2*T))
    cerclext=np.cos(T)
    cercleyt=np.sin(T)

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

    nb=0

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

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

#close() sinon pas de dessin

################################################
VOLUME
################################################
# Q_22 ##########################################

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_23 ##########################################
def al(b): # genere un nb alea entre -b et b
    return -b+2*b*random()
```

```
# Q_24 ##########################################
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_25 ##########################################
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
```