Corrigé du TP Python sur la simulation

852 - M.Lalauze - M.Junier

1 Préambule

Le module random de Python fournit un certain nombre de fonctions permettant de générer des nombres dits *pseudo-aléatoires*. Il s'agit de nombres générés par des méthodes déterministes mais se comportant *comme* s'ils étaient vraiment aléatoires. Ils nous permettront ici de simuler des expériences aléatoires.

Nous n'utiliserons que deux des fonctions de ce module :

- `randint(a,b)` renvoie un entier tiré au hasard entre a et b inclus (a et b doivent être des entiers) ;
- `random()` renvoie un réel tiré *au hasard* entre 0 et 1. *Au hasard* signifie ici que les nombres renvoyés sont uniformément distribués sur $[0, 1]$: si $0 \leqslant a \leqslant b \leqslant 1$, la probabilité que le nombre renvoyé soit entre a et b vaut $b - a$.

Vous taperez donc au début de votre fichier `.py`, `from random import randint, random`.

```
from random import random, randint
```

Vous taperez également les lignes suivantes pour importer les bibliothèques graphiques.

```
#import du module matplotlib.pyplot pour les graphiques
import matplotlib.pyplot as plt
import numpy as np
```

2 Lancers de dés

2.1 Exercice 1 Lancers de dés partie 1

```
def de(nb_faces):
    return randint(1,nb_faces)

def frequence_de_equilibre(nb_faces, nb_exp):
    t = [0]*nb_faces  #t[k] contiendra le nb d'occurences de la face k +1
    for i in range(nb_exp):
        t[de(nb_faces) - 1] += 1
```



```
    return [t[k]/nb_exp for k in  range(len(t))]

"""
In [2]: frequence_de_equilibre(4, 10**4)
Out[2]: [0.2524, 0.2438, 0.2553, 0.2485]
"""

def grands_nombres():
    """Graphique du nombre moyen de lancers avant le premier 6
    pour tailles n d'échantillon"""
    import matplotlib.pyplot as plt
    import numpy as np
    plt.xlabel('Faces')
    plt.ylabel('Fréquences')
    plt.axis([0,7,0.12,0.19])
    plt.grid()
    #plt.savefig('grandsnombres_de_equilibre.png')
    plt.title('Loi faible des grands nombres, dé équilibré')
    nlancers = [10**i for i in  [3, 4, 6]]
    couleurs = ['red', 'green', 'blue']
    for k in range(3):
        freq = frequence_de_equilibre(6, nlancers[k])
        plt.plot([face for face in range(1,7)],freq,color=couleurs[k],marker='^', label=r'$%s$
    plt.legend(loc='lower center')
    plt.savefig('exo1-loi-grands-nombres.png')
    plt.show()

"""
grands_nombres()
"""
```

2.2 Exercice 2 Lancers de dés partie 2

```
def premier6():
    """retourne le nombre de lancers jusqu'à l'obtention
    du premier 6"""
    compteur = 1
    while de(6) != 6:
        compteur += 1
    return compteur

def moyenne_premier6(n):
    """nombre de lancers moyen jusqu'à l'apparition du
    premier 6 sur n parties"""
    c = 0
    for i in range(n):
        c += premier6()
```

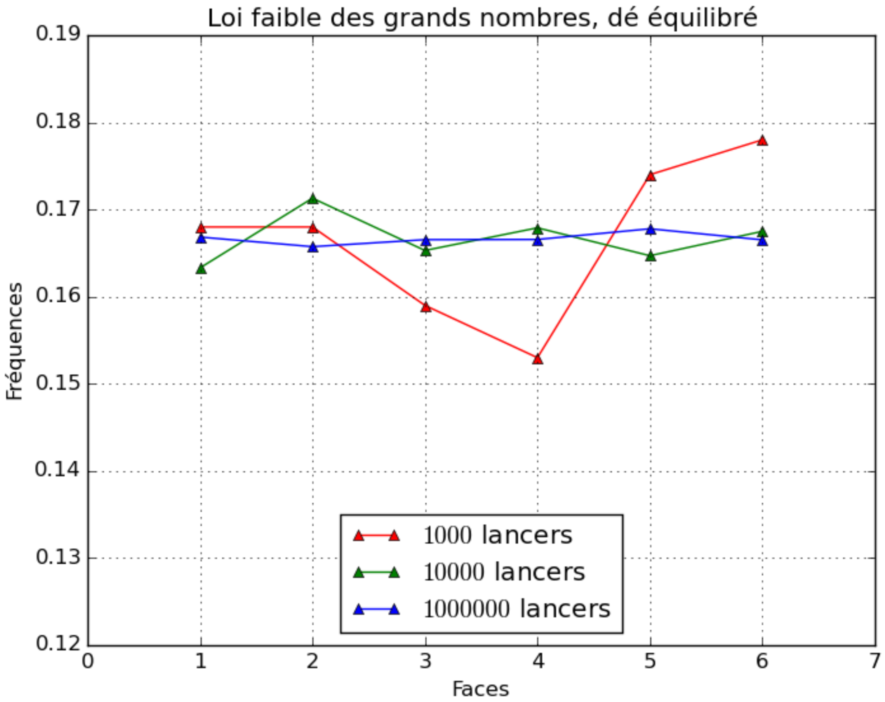


Figure 1:


```
    return c/n

def test_exo2():
    """Graphique du nombre moyen de lancers avant le premier 6
    pour tailles n d'échantillon"""
    nlancers = [10*2**i for i in range(14)]
    moypremier6 = []
    for n in nlancers:
        moypremier6.append(moyenne_premier6(n))
    plt.plot(nlancers,moypremier6,color='red',marker='^')
    plt.ylim(float(min(moypremier6))-0.2,float(max(moypremier6))+0.2)
    #échelle logarithmique sur l'axe des x
    plt.xscale('log')
    plt.title('Nombre moyens de lancers avant le premier 6')
    plt.xlabel('Nombre de parties')
    plt.ylabel('Nombre moyen de lancers')
    plt.savefig('nbmoyenlancerspremier6.png')
    plt.show()
```

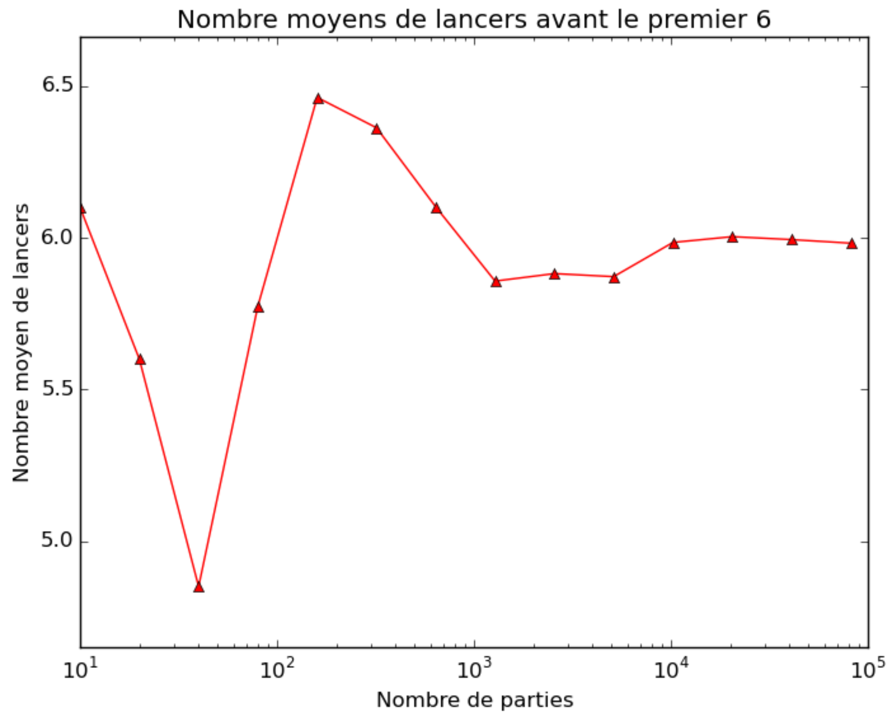


Figure 2:

3 Tirages dans une urne

3.1 Exercice 3 Tirages dans une urne avec remise

Écrire une fonction `avec_remise(nb_tirees, nb_total)` qui simule un tirage avec remise de `nb_tirees` boules dans une urne contenant `nb_total` boules, numérotées de 0 à `nb_total - 1`. On renverra la liste des numéros obtenus (dans l'ordre dans lequel on les a obtenus).

```
def avec_remise(nb_tirees, nb_total):
    """nb_tirees tirages avec remise dans une urne equiprobable contenant
    nb_total boules, equivalent de [randint(0 , nb_total -1) for _ in range(nb_tirees)]"""
    res = []
    for i in range(nb_tirees):
        res.append(randint(0 , nb_total -1))
    return res

def avec_remise2(nb_tirees, nb_total):
    """nb_tirees tirages avec remise dans une urne equiprobable contenant
    nb_total boules, equivalent de [randint(0 , nb_total -1) for _ in range(nb_tirees)]"""
    return [randint(0 , nb_total -1) for i in range(nb_tirees)]

def avec_remise_fausse(nb_tirees, nb_total):
    """Duplication du meme tirage nb_tirees fois"""
    return [randint(0 , nb_total -1)]*nb_tirees

"""
In [18]: avec_remise(10 ,20)
Out[18]: [10, 10, 5, 10, 4, 16, 18, 14, 3, 15]
"""
```

3.2 Exercice 4 Tirages dans une urne sans remise

Écrire une fonction `sans_remise(nb_tirees, nb_total)` qui simule un tirage sans remise de `nb_tirees` boules dans une urne contenant `nb_total` boules, numérotées de 0 à `nb_total - 1`. On renverra la liste des numéros obtenus, ou `"impossible"` si l'on demande de tirer plus de boules qu'il n'y en a dans l'urne.

On procédera comme suit, on tiendra à jour une liste `urne` contenant les éléments présents dans l'urne. Et pour ce faire on procèdera par slicing, pour supprimer l'élément d'indice `k` on écrit : `urne=urne[:k]+urne[k+1:]`

```
def sans_remise0(nb_tirees, nb_total):
    if nb_tirees> nb_total:
        return "Impossible"
    urne = [i for i in range(nb_total)]
    res = []
    for k in range(nb_tirees):
```



```
        print("Composition de l'urne avant le tirage", urne)
        #on choisit la position de la boule tirée
        # dans l'urne de taille nbtotal - k
        #k est le numero du tirage de 0 à nb_tirees -1
        pos = randint(0, nb_total -1 - k)
        #urne[pos] est la boule en position pos
        # res.append(urne[pos])
        #on supprime urne[pos] dans l'urne
        # urne = urne[:pos] + uren[pos+1:]
        tirage = urne.pop(pos)
        res.append(tirage)
    return urne
```

Différence entre les méthodes de liste `append` et `pop`

```
"""
In [1]: t = [851, 852, 853]

In [2]: t.append(854)

In [3]: t
Out[3]: [851, 852, 853, 854]

In [4]: t = t.append(855)

In [5]: t

In [6]: t = [851, 852, 853]

In [7]: a = t.pop(1)

In [8]: a
Out[8]: 852

In [9]: t
Out[9]: [851, 853]
"""


def sans_remise(nb_tirees, nb_total):
    if nb_tirees> nb_total:
        return "Impossible"
    urne = [i for i in range(nb_total)]
    res = []
    nb_boules = nb_total
    for k in range(nb_tirees):
        print("Composition de l'urne avant le tirage", urne)
        i = randint(0, nb_boules - 1)
        res.append(urne[i])
```

```
        urne = urne[:i] + urne[i + 1:]
        nb_boules -= 1
    return res

for k in [2, 5, 10 , 11]:
    print('Tirages de %s boules dans une urne de 10 boules sans remise'%k)
    print(sans_remise(k, 10))
    print()


"""
In [22]: (executing lines 433 to 448 of "852-correc-TPsimulation-2016-md.py")
Tirages de 2 boules dans une urne de 10 boules sans remise
Composition de l'urne avant le tirage [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
Composition de l'urne avant le tirage [1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 3]

Tirages de 5 boules dans une urne de 10 boules sans remise
Composition de l'urne avant le tirage [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
Composition de l'urne avant le tirage [1, 2, 3, 4, 5, 6, 7, 8, 9]
Composition de l'urne avant le tirage [1, 2, 3, 4, 6, 7, 8, 9]
Composition de l'urne avant le tirage [2, 3, 4, 6, 7, 8, 9]
Composition de l'urne avant le tirage [2, 3, 4, 6, 8, 9]
[0, 5, 1, 7, 9]

Tirages de 10 boules dans une urne de 10 boules sans remise
Composition de l'urne avant le tirage [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
Composition de l'urne avant le tirage [1, 2, 3, 4, 5, 6, 7, 8, 9]
Composition de l'urne avant le tirage [1, 2, 4, 5, 6, 7, 8, 9]
Composition de l'urne avant le tirage [1, 2, 4, 5, 6, 7, 8]
Composition de l'urne avant le tirage [1, 2, 4, 6, 7, 8]
Composition de l'urne avant le tirage [1, 2, 4, 6, 7]
Composition de l'urne avant le tirage [2, 4, 6, 7]
Composition de l'urne avant le tirage [2, 6, 7]
Composition de l'urne avant le tirage [6, 7]
Composition de l'urne avant le tirage [6]
[0, 3, 9, 5, 8, 1, 4, 2, 7, 6]

Tirages de 11 boules dans une urne de 10 boules sans remise
Impossible
"""

def sans_remise2(nb_tirees, nb_total):
    """La même mais avec la méthode pop au lieu du slicing"""
    if nb_tirees> nb_total:
        return "Impossible"
    urne = [i for i in range(nb_total)]
    nb_boules = nb_total
    res = []
    for k in range(nb_tirees):
```



```
        print("Composition de l'urne avant le tirage", urne)
        i = randint(0, nb_boules - 1)
        res.append(urne.pop(i))
        nb_boules -= 1
    return res
```

3.3 Exercice 5 Tirages avec remise partie 2

Écrire une fonction `urne(n)` qui simule n tirages avec remise dans une urnes contenant des boules bleues, vertes et rouges avec les proportions respectives suivantes : $\frac{1}{4}$, $\frac{1}{4}$ et $\frac{1}{2}$, et qui renvoie le nombre de boules de chaque couleur obtenues.

(*indic.* Pour simuler le tirage des boules, on pourra faire appel à la fonction `random()` et on rappelle ce qui a été dit en préambule sur cette fonction : si $0 \leqslant a \leqslant b \leqslant 1$, la probabilité que le nombre renvoyé soit entre a et b vaut $b - a$.)

```
from random import random, randint

def urne(n):
    """Tirage avec remise de n boules dans une urne contenant 3 boules
    de proportions 0.25 0.25 et 0.5.
    Retourne le tableau des nombres de tirages par catégorie"""
    boule = [0]*3
    for k in range(n):
        de = random()
        if de < 0.25:
            boule[0] += 1
        elif de < 0.5:
            boule[1] += 1
        else:
            boule[2] += 1
    return boule

"""
In [11]: [ [boule/n  for boule in urne(n)] for n in [100, 1000, 10000, 100000] ]
Out[11]:
[[0.25, 0.36, 0.39],
 [0.257, 0.217, 0.526],
 [0.2574, 0.248, 0.4946],
 [0.24992, 0.25246, 0.49762]]
"""
```

4 Méthode de Monte-Carlo

4.1 Exercice 6 : Méthode de Monte-Carlo

- **Question 1** : On choisit au hasard un point M de coordonnées (x, y) dans $[0, 1] \times [0, 1]$.

La probabilité pour qu'il appartienne au quart de disque de centre O de rayon 1 est égale au rapport entre l'aire du quart de disque $\frac{\pi}{4}$ et l'aire du carré 1. C'est donc $\frac{\pi}{4}$.

- **Question 2** :

```
def monte_carlo(n):
    """Pour n points choisis au hasard dans [0;1]x[0;1],
    retourne la proportion de points appartenant au quart
    de disque de centre O et de rayon 1"""
    compteur = 0
    for i in range(n):
        x,y = random(), random()
        if x**2+y**2<1:
            compteur += 1
    return compteur/n

def monte_carlobis(n):
    """retourne  deux listes aléatoires de taille
    n de flottants dans [0;1] et le nombre de points (x,y) dans
    le quart de disque de centre (0,0) et de rayon 1"""
    xliste,yliste = [],[]
    compteur = 0
    for i in range(n):
        x,y = random(),random()
        xliste.append(x)
        yliste.append(y)
        if x**2+y**2<1:
            compteur += 1
    return xliste,yliste,compteur

def graphe_monte_carlo(n):
    """Représente graphiquement les points appartenant
    au quart de disque pour un échantillon de taille n"""
    import matplotlib.pyplot as plt
    import numpy as np

    # Tracé des points tirés au hasard
    xlist,ylist,prop = monte_carlobis(n)
    plt.plot(xlist,ylist,color='red',marker='o',markersize=0.5,ls='')

    # Tracé du quart de cercle
    x = np.linspace(0,1,501)
    plt.plot(x,np.sqrt(1-x**2),color='blue')

    # Échelle, titre et sauvegarde
    plt.ylim(0,1)
    plt.xlim(0,1)
    plt.title(r"""Nuage de %s points aléatoires
    Fréquence dans le demi-disque : %.3f   ,  $\frac{\pi}{4} \approx$ %.3f"""%(n,prop,np.pi/4))
    plt.savefig('nuagede%s_points_montecarlo.png'%n)
```



```
    plt.show()
```

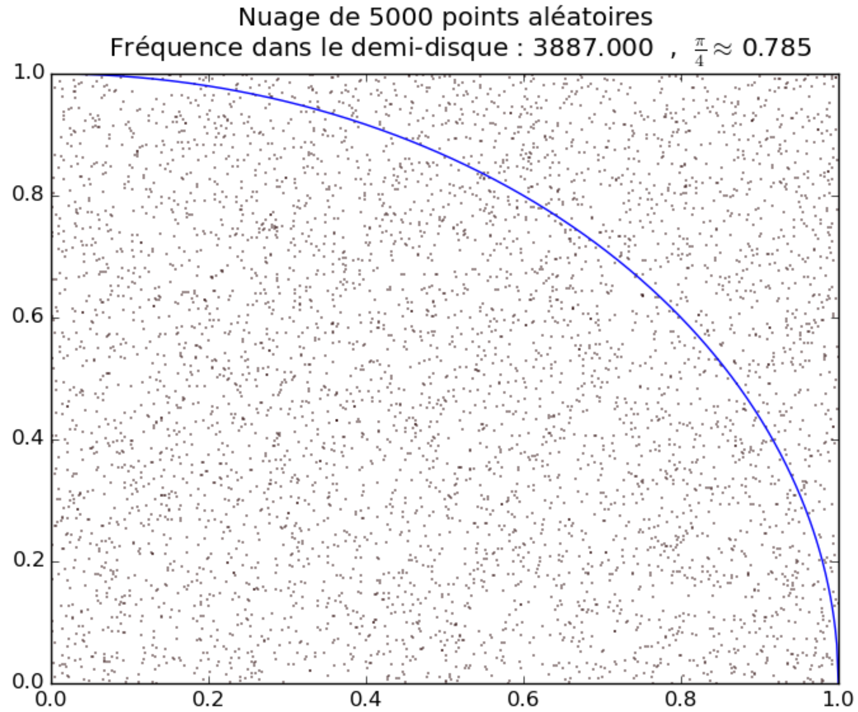


Figure 3:

5 Marche aléatoire

5.1 Exercice 7 Marche aléatoire dans le plan

- **Question1**

  Écrire une fonction `deplacement(x,y,p)` qui pour un réel $p \in]0,1[$ retourne aléatoirement :

  – 'droite' avec la probabilité $(1-p)/2$
  – 'gauche' avec la probabilité $(1-p)/2$
  – 'haut' avec la probabilité $p/2$
  – 'bas' avec la probabilité $p/2$

```
def deplacement(x,y,p):
    """retourne les nouvelles coordonnées de la particule
```

```
    après un déplacement aléatoire élémentaire avec la parametre p"""
    hasard = random()
    if hasard<p/2:
        return x,y+1
    elif hasard<p:
        return x,y-1
    elif hasard<(1+p)/2:
        return x-1,y
    return x+1,y

les_deplacements = [deplacement(0,0,1/2) for _ in range(10000)]
les_effectifs = [les_deplacements.count(d) for d in [(0,1),(0,-1),(1,0),(-1,0)]]

"""
In [5]: print(les_effectifs) #vérification de l'équiprobabilité directionnelle si p=1/2
[2461, 2500, 2505, 2534]
"""
```

- **Question 2**

  Écrire une fonction `finpromenade(x,y,B` prenant en argument les coordonnées (x, y) de la particule et qui renvoie `True` si la particule a atteint la frontière du domaine et qui rend `False` sinon.

```
#Pour les 2 fonctions proposées il est recommandé d'utiliser
#des coordonnées entières sinon les tests d'égalité risquent
#d'etre hasardeux (les nombres réels sont représentés de façon approchée)

def finpromenade(x,y,B):
    """Retourne 1 si la particule a atteint le bord du domaine
    et 0 sinon"""
    if x==B or x==-B or y==B or y==-B:
        return True
    return False

def finpromenade2(x,y,B):
    """Retourne 1 si la particule a atteint le bord du domaine
    et 0 sinon"""
    if (x**2-B**2)*(y**2-B**2)==0:
        return True
    return False
```

- **Question 3**

  Une particule est placée sur le point de coordonnées (x, y) dans l'enceinte.

  Écrire une fonction `longueur(x,y,p,B)`qui simule la promenade aléatoire de cette particule jusqu'à ce qu'elle atteigne l'une des frontières et qui calcule la longueur de cette promenade.

```
def longueur(x,y,p,B):
    """simule la promenade aléatoire d'une particule
```



```
    initialement placée en (x,y) dans le domaine [-B;B]x[-B;B]
    et retourne la longueur de la promenade (de paramètre p)"""
    length = 0
    if not(-B<=x<=B or -B<=y<=B):
        return "Le point initial n'est pas dans le domaine"
    else:
        while not(finpromenade(x,y,B)):
            x,y = deplacement(x,y,p)
            length += 1
    return length
```

- **Question 4**

  Écrire une fonction `echantillon(n,p,B)` qui place au hasard n particules dans l'enceinte et calcule la longueur moyenne de leur promenade.

```
def echantillon(n,p,B):
    """place n particules au hasard dans le domaine [-B,B]x[-B,B]
    ,calcule la longueur de leur promenade aléatoire de paramètre p
    et retourne la liste des longueurs et leur moyenne"""
    serie = []
    for i in range(n):
        serie.append(longueur(randint(-B,B),randint(-B,B),p,B))
    return serie,sum(serie)/n

#test pour un échantillon de taille n=10 avec p=0.4 et B=50
def test_exo4q4(n,p,B):
    """
    test pour un échantillon de taille n=10 avec p et B fixés
    """
    serie,moyenne = echantillon(n,p,B)
    #diagrammes sur un échantillon de taille n
    plt.subplot(121)
    plt.title("""Diagramme d'évolution
    p = %.2f et B=%s"""%(p,B))
    x = np.arange(1,n+1,1)
    plt.plot(x,serie,color='red',marker='o',markersize=0.7)
    plt.subplot(122)
    plt.title("""Diagramme en boite, moyenne=%s"""%moyenne)
    lmax = max(serie)
    plt.boxplot(serie,vert=True)
    plt.ylim(-100,lmax+100)
    plt.yticks(np.arange(-100,lmax+100,1000))
    #plt.yticks(np.arange(-100,lmax+100,100))
    plt.savefig('promenadealeatoire_echantillon_taille%s.png'%n)
    plt.subplots_adjust(wspace=0.3)
    plt.show()
```

- **Question 5**

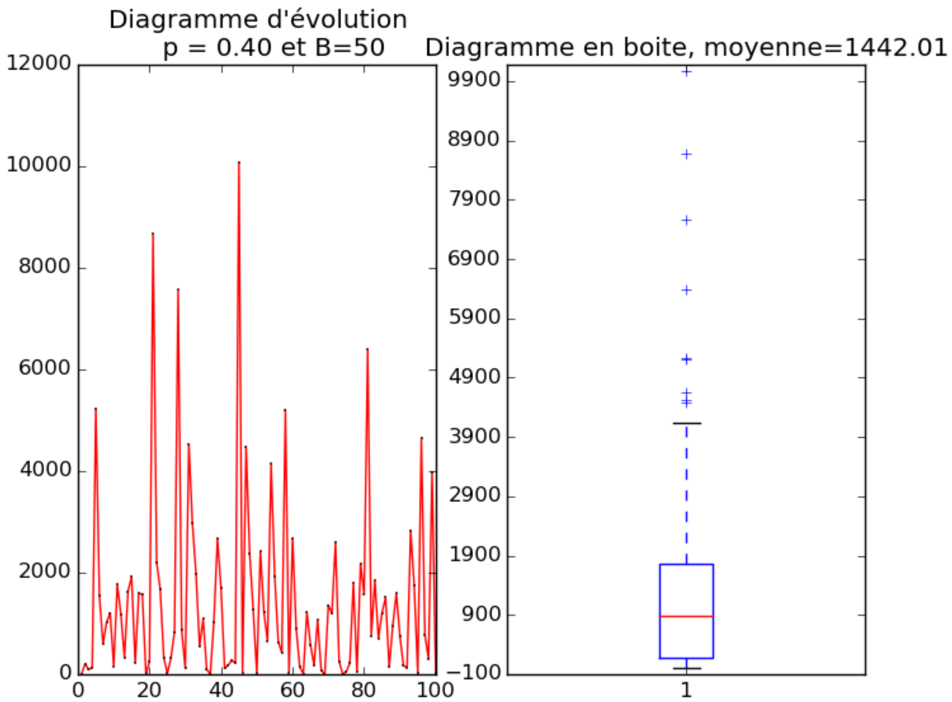


Figure 4:

Copier/Coller le bout de code permettant de représenter graphiquement le chemin suivi par une particule.

```
def trace_chemin(p,B):
    """Trace la promenade aléatoire de paramètre p
    d'une particule dans le domaine [-B,B]x[-B,B]"""
    import matplotlib.pyplot as plt
    #choix aléatoire du point de départ
    x,y = 0,0
    if not(-B<=x<=B or -B<=y<=B):
        return "Le point initial n'est pas dans le domaine"
    else:
        #tableau des abscisses et ordonnées successives
        tabx,taby = [],[]
        while not(finpromenade(x,y,B)):
            x,y = deplacement(x,y,p)
            tabx.append(x)
            taby.append(y)
        plt.title(r"Promenade aléatoire de paramètre %s" + "\n"%p +r"dans le domaine [-%s;%s]"%
        plt.scatter(tabx, taby, c=range(len(tabx)))
        plt.colorbar()
        plt.xlim(-B,B)
        plt.ylim(-B,B)
        plt.savefig('promenade-p=%s.png'%str(p).replace('.',','))
        #plt.show()
        plt.clf()
```

6 Quelques promenades

```
n, B = 10, 40
parametres = np.linspace(0, 1, 11)
for p in parametres:
    trace_chemin(p,B)
```

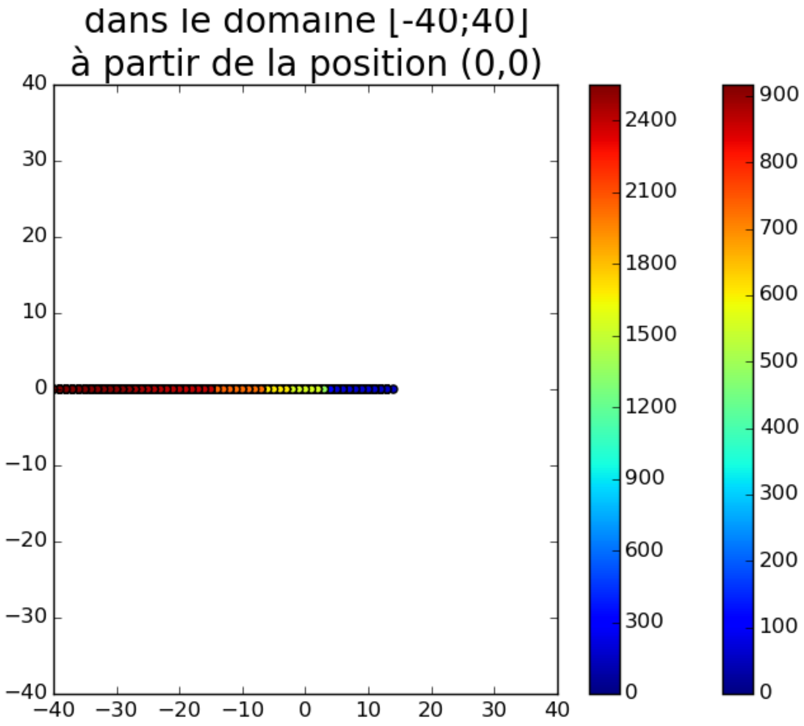


Figure 5:

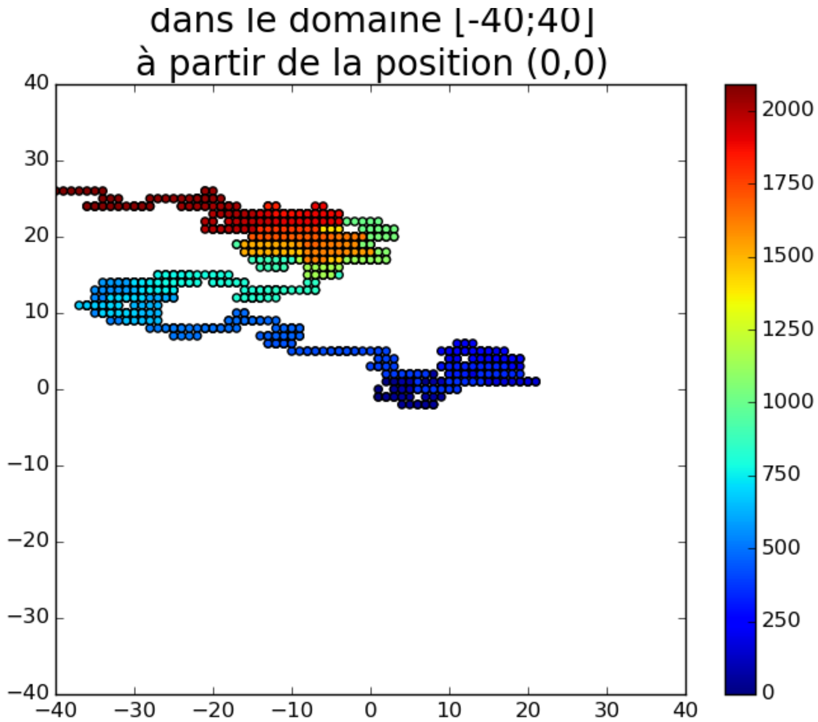


Figure 6:

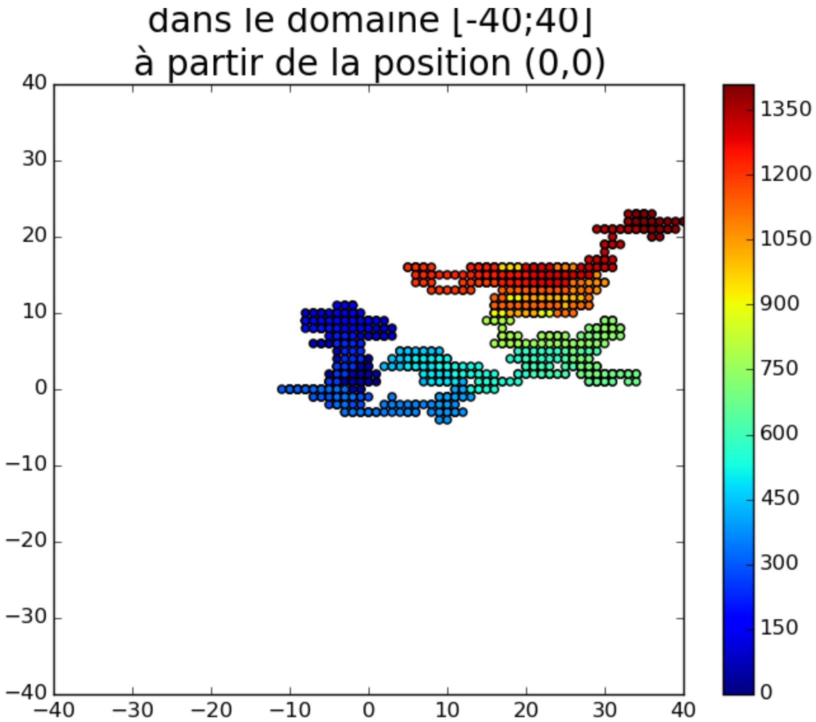


Figure 7:

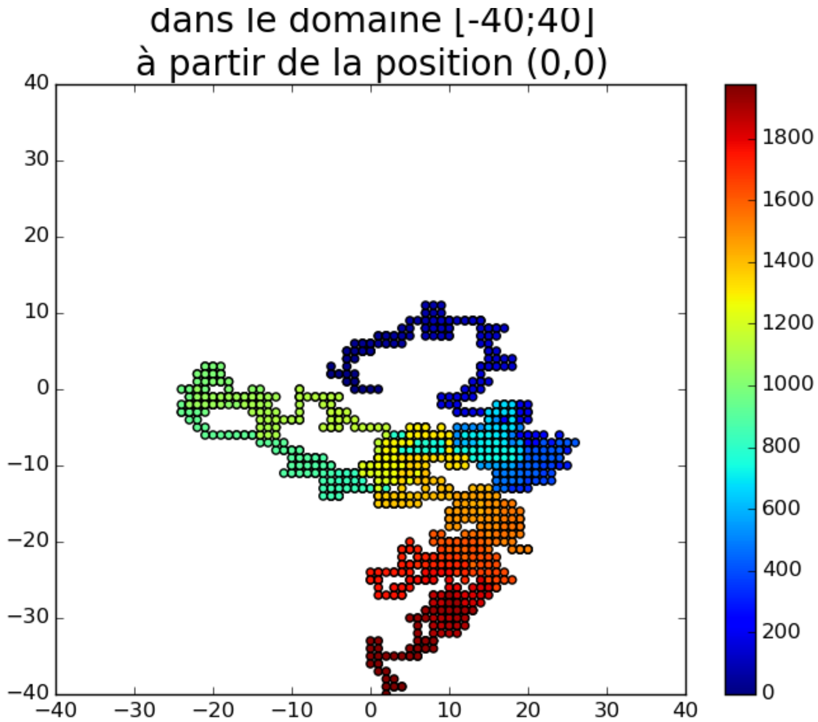


Figure 8:

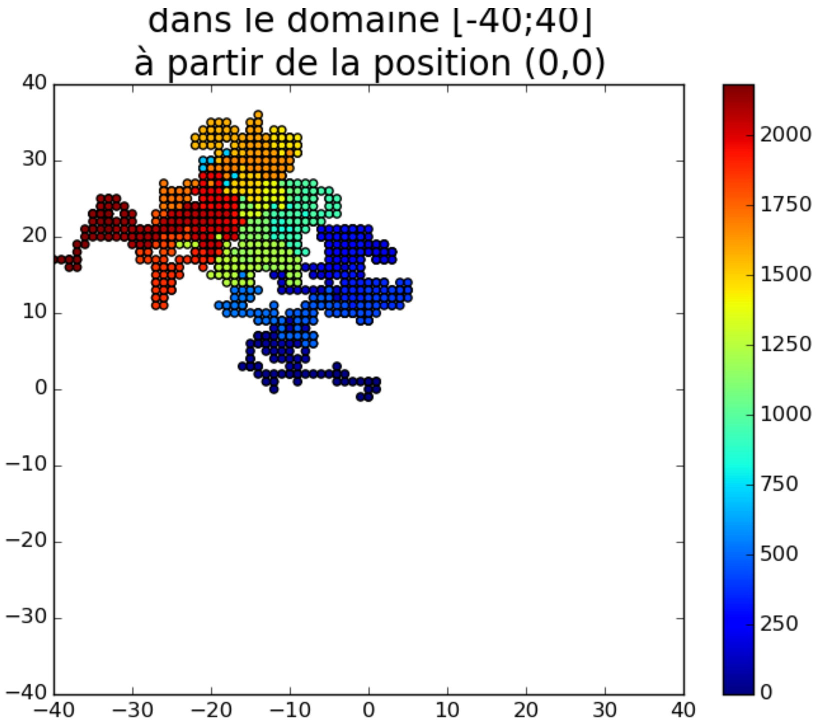


Figure 9:

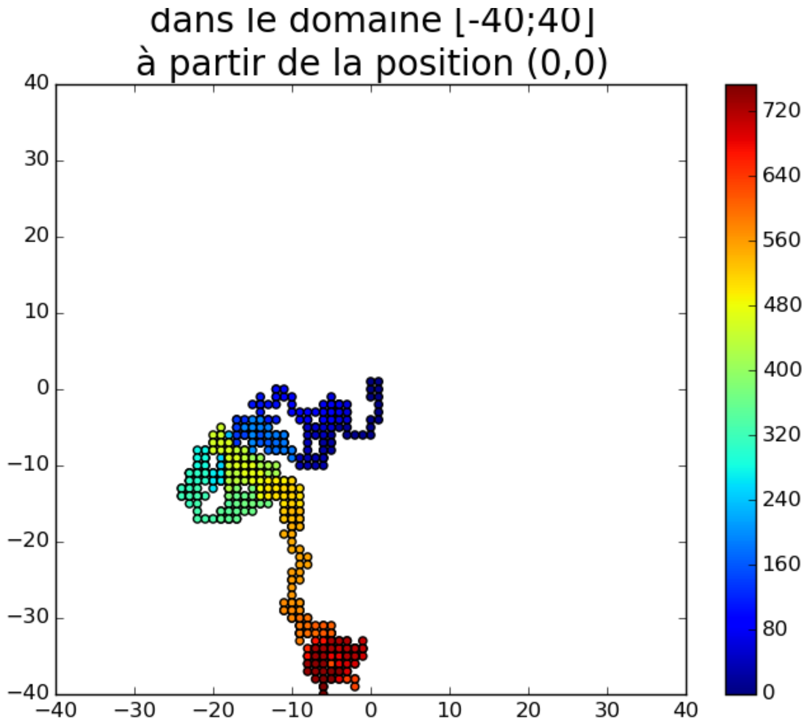


Figure 10:

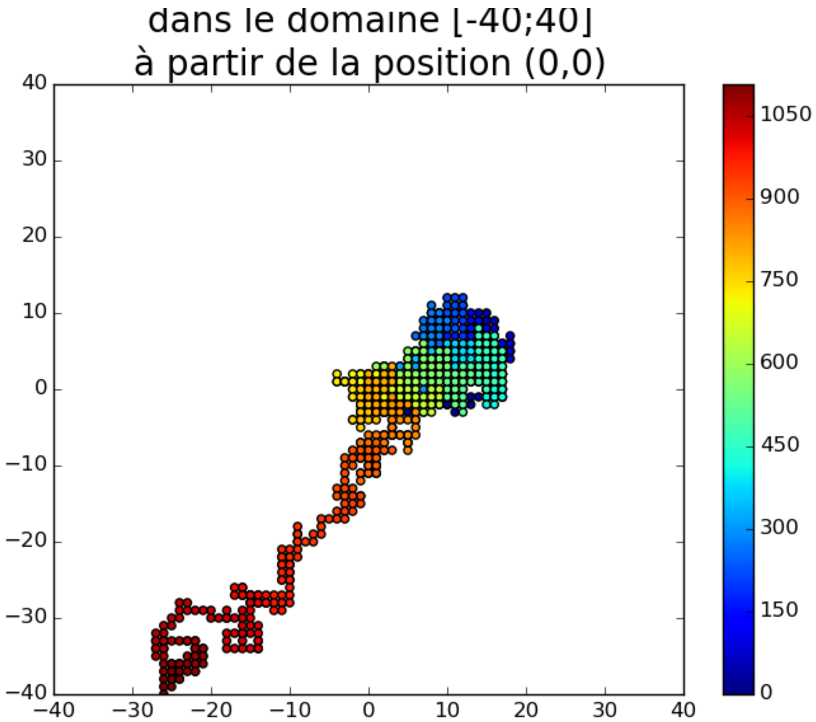


Figure 11:

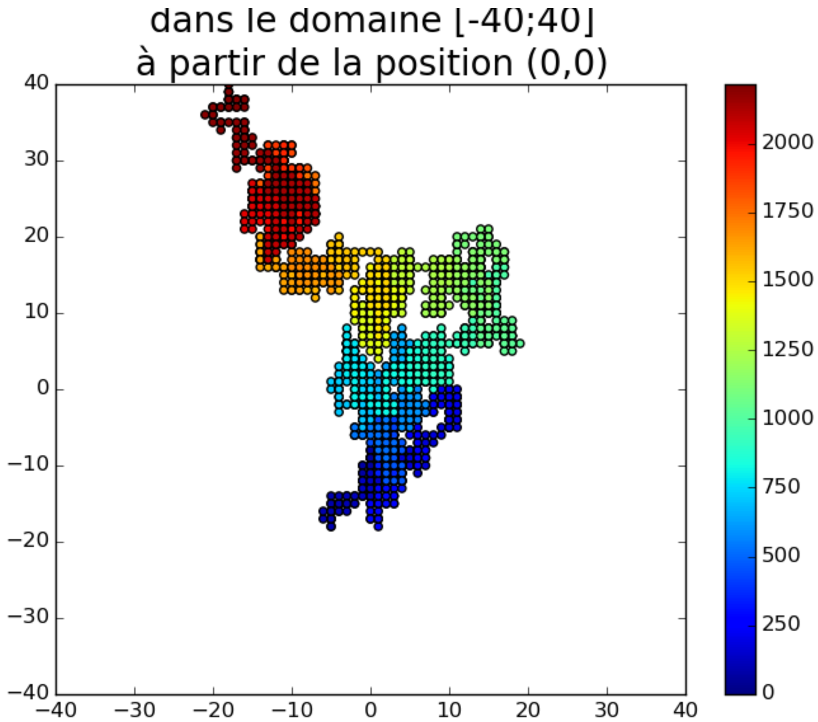


Figure 12:

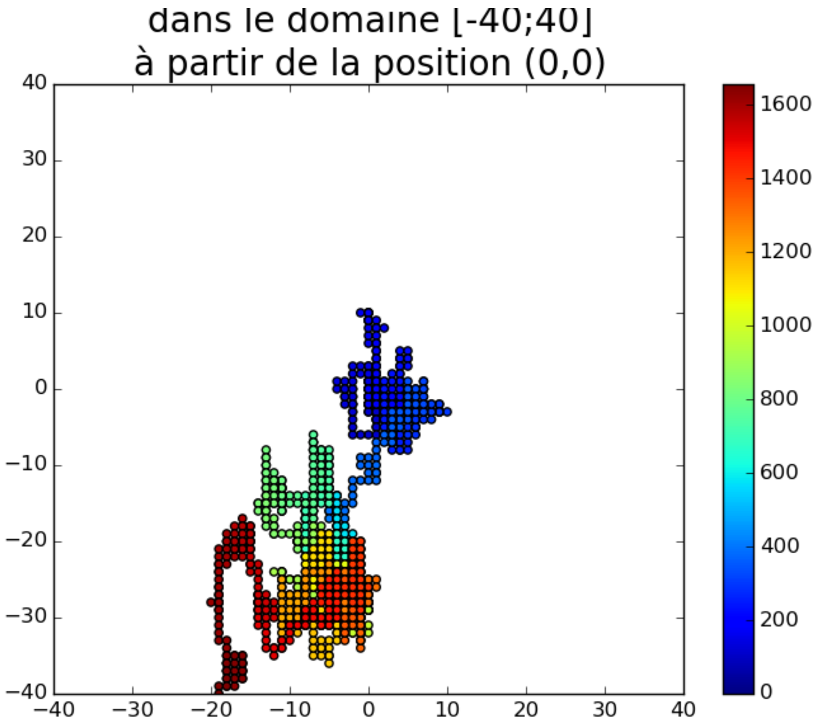


Figure 13:

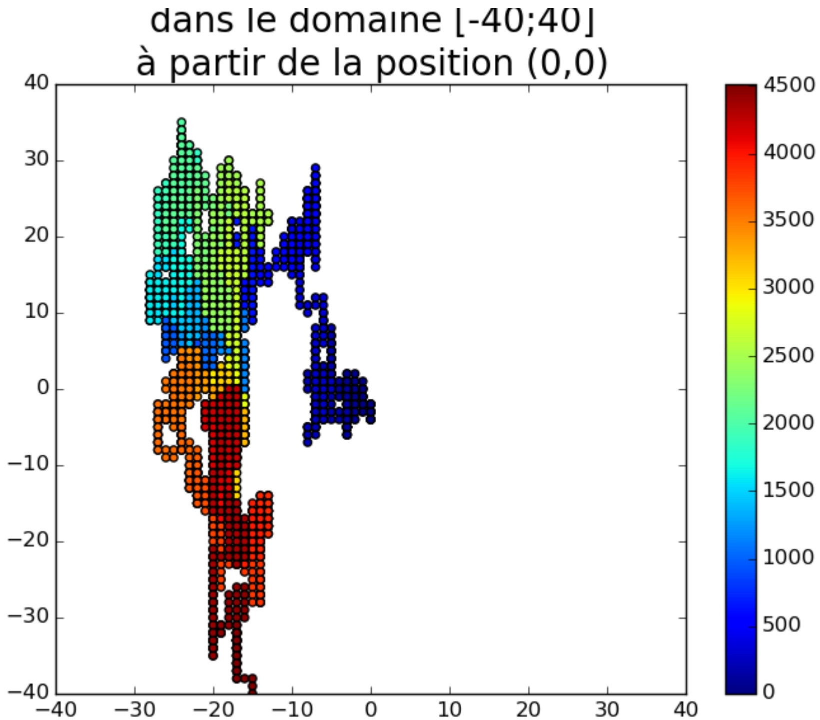
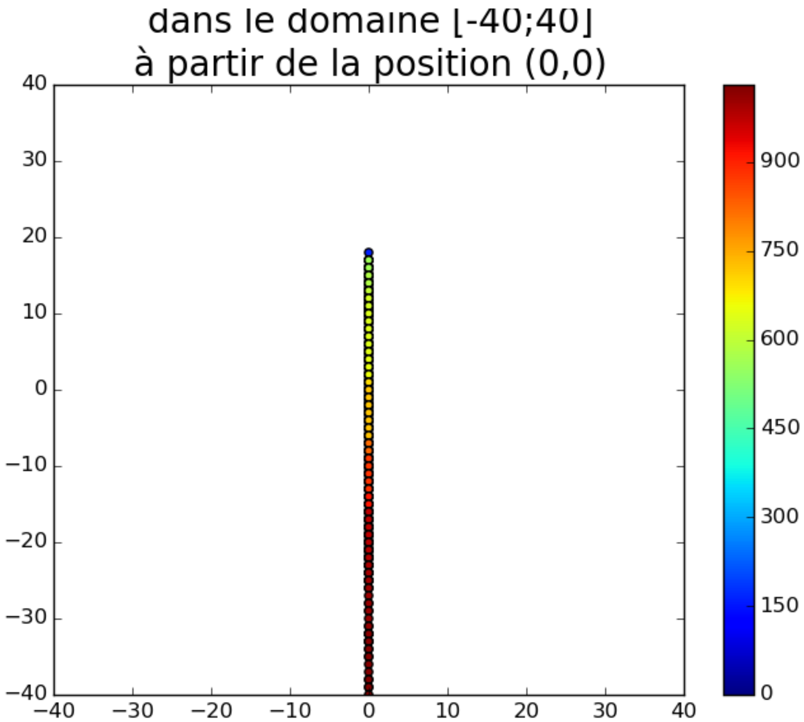


Figure 14:


dans le domaine [-40;40]
à partir de la position (0,0)
40
30
20
10
0
−10
−20
−30
−40
−40
−30
−20
−10
0
10
20
30
40
900
750
600
450
300
150
0

“”"

~