TP

October 19, 2020

0.0.1 Marche Aléatoire

Soit $S_n = S_{n-1} + X_n$ avec $S_0 = 0$, une marche aléatoire . On s'intéresse à des marches aléatoires de N pas et à la distribution $P(S_N)$ de la position finale S_N d'une particule qui se déplace sur un axe.

1. Loi Binomiale :

1.1.1 Question 1 :

Générer quelques marches aléatoires de $N = 10000$ pas pour $p = 0.4, 0.49$ et 0.5.

1.1.2 Réponse 1 :

- On commence par définir une fonction simuler_pas qui simule la valeur d'un pas (+1 ou -1)
- Les entrées de la fonction sont : la probabilité p .

[7]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
from math import *
def simuler_pas(p):
    u=np.random.uniform(0,1)
    if (u<p):
        pas = 1
    else :
        pas = -1
    return pas
```

- On définit par la suite une fonction simuler_marche_aléatoire qui simule la marche aléatoire
- Les entrées de la fonction sont : le nombre de pas de la marche et la probabilité p
- La fonction retourne un tableau qui représente la marche

[8]:
```
def simuler_marche_aleatoire(n,p):
    s=0
    marche=[s]
    for i in range(n):
        s += simuler_pas(p)
        marche.append(s)
    return marche
```

- On génère les marches demandées
- Pour chaque valeur de p on affiche uniquement les 10 premières position et la position finale des 3 premières simulations.

- **Pour p = 0.4**

[9]:
```
s1 = [] # tableau qui comporte les marches aléatoires de pas = 10000 et p = 0.4
for i in range(50):
    m = simuler_marche_aleatoire(10000,0.4)
    s1.append(m)
for s in s1[:3]:
    print ( 'Les 10 premières position (10 premières valeurs de Sn) de la␣
 ↪marche : ',s[:10])
    print ( 'La position finale de la marche : ',s[10000])
    print ('')
```

```
Les 10 premières position (10 premières valeurs de Sn) de la marche :  [0, -1,
-2, -3, -4, -5, -6, -5, -6, -5]
La position finale de la marche :  -2086

Les 10 premières position (10 premières valeurs de Sn) de la marche :  [0, 1, 2,
1, 0, -1, -2, -3, -2, -3]
La position finale de la marche :  -1974

Les 10 premières position (10 premières valeurs de Sn) de la marche :  [0, -1,
-2, -3, -4, -3, -4, -3, -2, -3]
La position finale de la marche :  -2002
```

- **Pour p = 0.49**

[10]:
```
s2 = [] # tableau qui comporte les marches aléatoires de pas = 10000 et p = 0.49
for i in range(50):
    m = simuler_marche_aleatoire(10000,0.49)
    s2.append(m)
for s in s2[:3]:
    print ( 'Les 10 premières position (10 premières valeurs de Sn) de la␣
 ↪marche : ',s[:10])
    print ( 'La position finale de la marche : ',s[10000])
    print ('')
```

```
Les 10 premières position (10 premières valeurs de Sn) de la marche :  [0, 1, 0,
-1, 0, 1, 0, -1, 0, -1]
La position finale de la marche :  -256

Les 10 premières position (10 premières valeurs de Sn) de la marche :  [0, 1, 0,
1, 0, -1, -2, -3, -4, -3]
La position finale de la marche :  -142
```

```
Les 10 premières position (10 premières valeurs de Sn) de la marche :  [0, 1, 0,
-1, -2, -3, -2, -1, 0, -1]
La position finale de la marche :  -144
```

- **Pour p = 0.5**

[11]:
```python
s3 = [] # tableau qui comporte les marches aléatoires de pas = 10000 et p = 0.5
for i in range(50):
    m = simuler_marche_aleatoire(10000,0.5)
    s3.append(m)
for s in s3[:3]:
    print ( 'Les 10 premières position (10 premières valeurs de Sn) de la␣
 ↪marche : ',s[:10])
    print ( 'La position finale de la marche : ',s[10000])
    print ('')
```

```
Les 10 premières position (10 premières valeurs de Sn) de la marche :  [0, -1,
-2, -1, 0, 1, 2, 3, 2, 3]
La position finale de la marche :  -30

Les 10 premières position (10 premières valeurs de Sn) de la marche :  [0, 1, 2,
3, 2, 3, 2, 3, 4, 5]
La position finale de la marche :  38

Les 10 premières position (10 premières valeurs de Sn) de la marche :  [0, 1, 2,
3, 4, 3, 2, 3, 4, 5]
La position finale de la marche :  -74
```

1.2.1 Question 2 :

Tracer ces marches aléatoires (S_n en fonction de n).

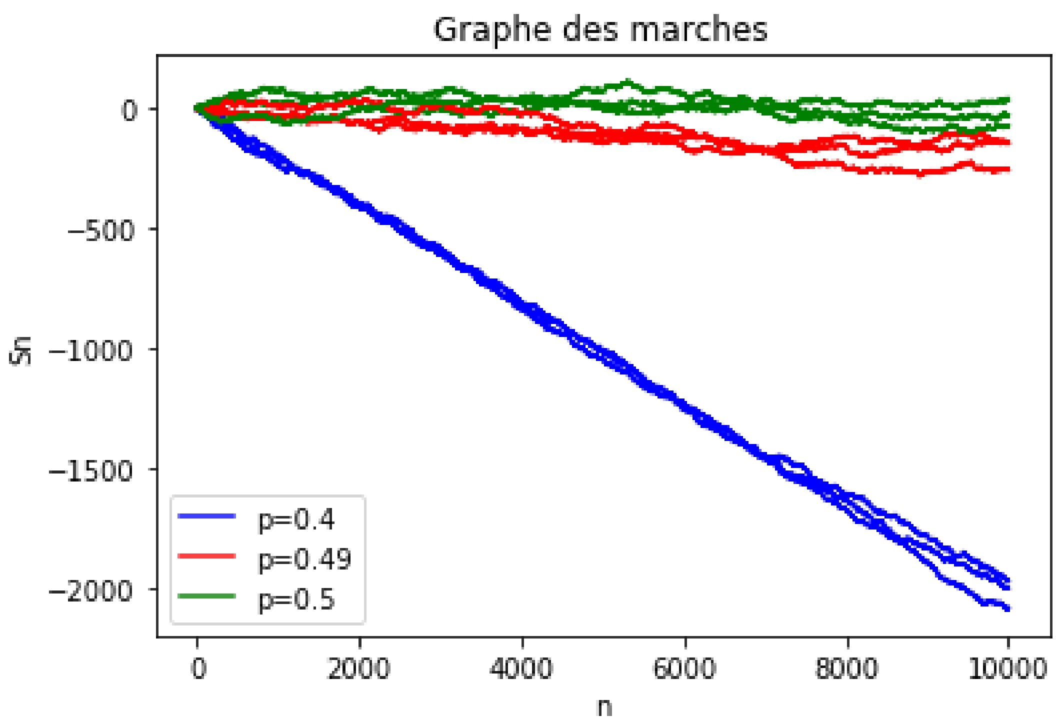
1.2.2 Réponse 2 :

- On trace 3 simulations pour chaque valeur de p
- On trace toutes les marches sur le meme graphe pour pouvoir comparer.
- Les couleurs sont comme suit :
  - $p = 0.4$ —> couleur Bleue
  - $p = 0.49$ —> couleur Rouge
  - $p = 0.5$ —> couleur Verte

[12]:
```python
X = np.linspace(0, 10000, 10001)
for i in range(3):
    if (i==0):
        plt.plot(X,s1[i],color = "blue",label="p=0.4")
        plt.plot(X,s2[i],color = "red",label="p=0.49")
        plt.plot(X,s3[i],color = "green",label="p=0.5")
```

```
    else:
        plt.plot(X,s1[i],color = "blue")
        plt.plot(X,s2[i],color = "red")
        plt.plot(X,s3[i],color = "green")
plt.title("Graphe des marches")
plt.xlabel('n')
plt.ylabel('Sn')
plt.legend()
plt.show
```

[12]: `<function matplotlib.pyplot.show(*args, **kw)>`



1.3.1 Question 3 :

Quelle est la trajectoire moyenne $< S_n >$ pour chaque valeur de p ?

1.3.2 Réponse 3 :

- On définit une fonction qui calcule la trajectoire moyenne $< S_n >$ pour un ensemble de marches aléatoires.
- Pour chaque trajectoire moyenne on affiche les 10 premières et les 10 dernières positions (valeurs de $< S_n >$)

[13]:
```
def calculer_trajectoire_moyenne (E): # E est l'ensemble des marches aleatoires
    res = []
```

```
    nb_marches = len(E) # nombre de marches
    nb_pas = len(E[0]) # nombre de pas dans les marches
    for i in range (nb_pas):
        for num_marche in range(nb_marches):
            if (num_marche==0):
                res.append(E[num_marche][i])
            else:
                res[i] += E[num_marche][i]
        res[i] = res[i]/nb_marches
    return res
```

- **Pour $p = 0.4$:**

[14]:
```
moy = calculer_trajectoire_moyenne(s1)
print('Les 10 premières composantes de < Sn > sont : ',moy[:10])
print('Les 10 dernières composantes de < Sn > sont : ',moy[-10:])
```

```
Les 10 premières composantes de < Sn > sont :  [0.0, -0.12, -0.2, -0.36, -0.64,
-0.84, -1.2, -1.64, -1.84, -2.08]
Les 10 dernières composantes de < Sn > sont :  [-1997.48, -1998.0, -1998.16,
-1998.24, -1998.48, -1998.64, -1998.88, -1999.4, -1999.36, -1999.68]
```

- **Pour $p = 0.49$:**

[15]:
```
moy = calculer_trajectoire_moyenne(s2)
print('Les 10 premières composantes de < Sn > sont : ',moy[:10])
print('Les 10 dernières composantes de < Sn > sont : ',moy[-10:])
```

```
Les 10 premières composantes de < Sn > sont :  [0.0, -0.04, -0.04, -0.04, -0.04,
0.08, -0.08, -0.16, 0.08, -0.16]
Les 10 dernières composantes de < Sn > sont :  [-192.0, -191.96, -192.28,
-192.2, -192.28, -191.96, -192.2, -192.08, -192.12, -192.2]
```

- **Pour $p = 0.5$:**

[16]:
```
moy = calculer_trajectoire_moyenne(s3)
print('Les 10 premières composantes de < Sn > sont : ',moy[:10])
print('Les 10 dernières composantes de < Sn > sont : ',moy[-10:])
```

```
Les 10 premières composantes de < Sn > sont :  [0.0, 0.12, -0.12, 0.0, 0.08,
0.12, 0.32, 0.16, 0.04, 0.04]
Les 10 dernières composantes de < Sn > sont :  [15.56, 15.72, 15.8, 15.96,
16.04, 16.12, 16.0, 15.88, 15.68, 15.72]
```

Observer l'aspect d'une marche aléatoire M symétrique ($p = 0.5$) de $N = 10000$ pas.

Soit M la marche aléatoire obtenue à partir des 1000 premiers pas de M.

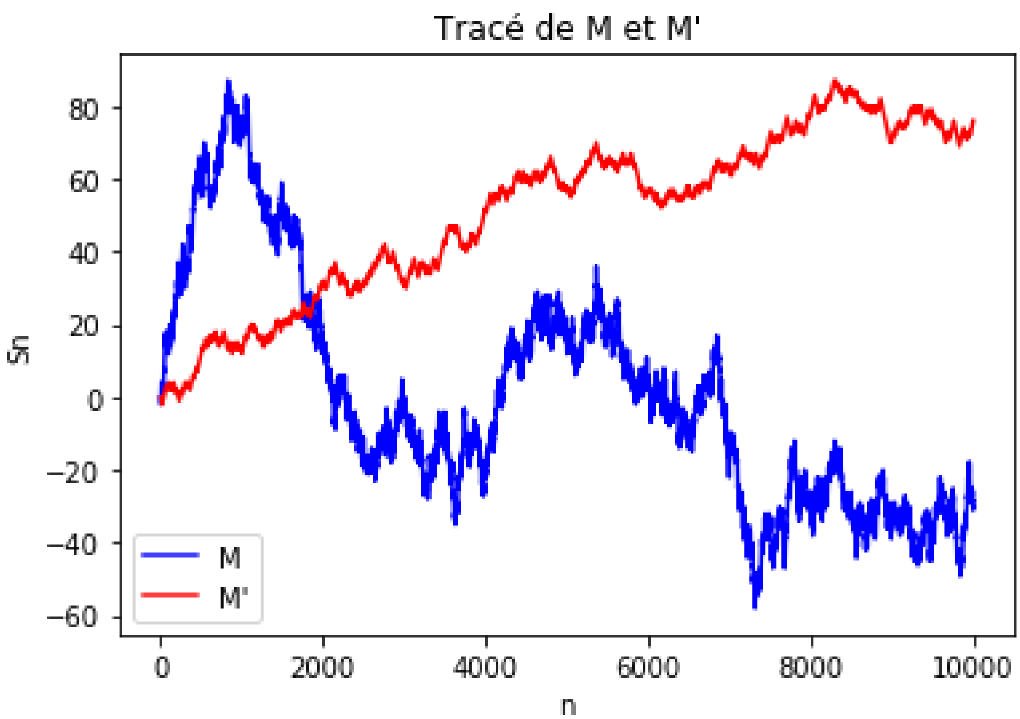
1.4.1 Question 4 :

Tracer sur un même graphe M et M en multipliant la taille des itérations (axe des abscisses) de cette dernière par 10.

1.4.2 Réponse 4 :

[17]:
```
M = s3[0]
Mprime = M[:1001]
X = np.linspace(0, 10000, 10001)
plt.plot(X,M,color = "blue",label = "M")
X = np.linspace(0, 10000, 1001)
plt.plot(X,Mprime,color = "red",label = "M'")
plt.title("Tracé de M et M'")
plt.xlabel('n')
plt.ylabel('Sn')
plt.legend()
plt.show
```

[17]: <function matplotlib.pyplot.show(*args, **kw)>



Question : Par quel facteur faut-il multiplier la taille des pas (axe des ordonnées) de M pour que les deux marches aléatoires aient le même aspect ?

Réponse : il n y a aucun facteur par lequel on multiplie la taille des pas de M' pour que les deux marches aléatoires aient le meme aspect car les pas de 1000 à 10000 sont aléatoires et donc indépendants des pas de 0 à 1000.

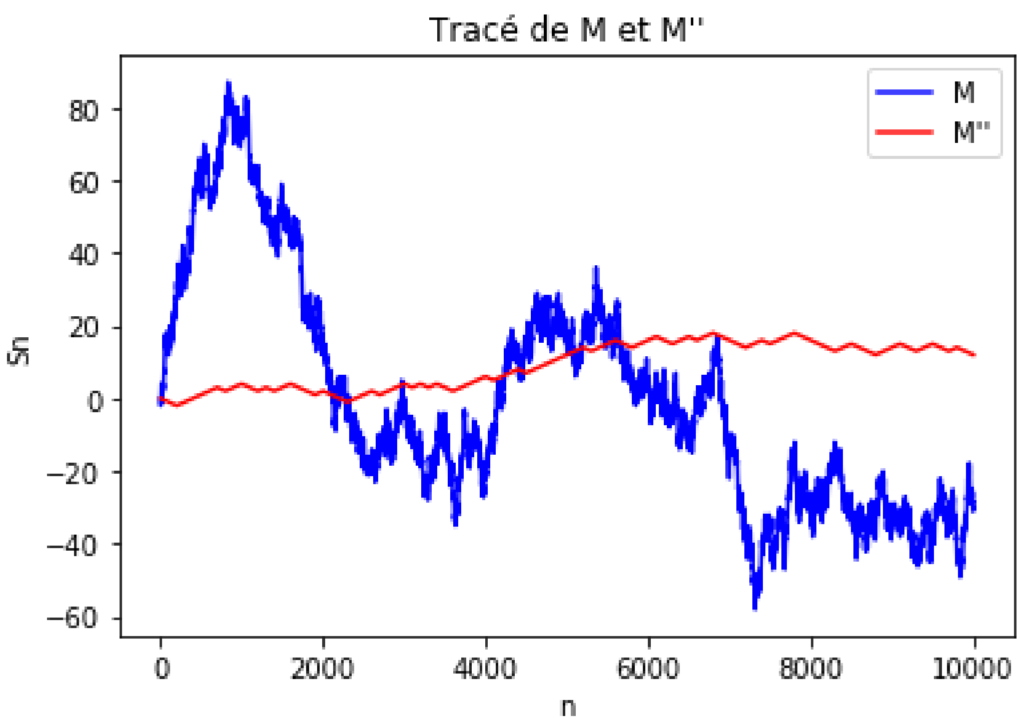
1.5.1 Question 5 :

Reprendre la même procédure avec M , la marche aléatoire obtenue à partir des 100 premiers pas de M.

1.5.2 Réponse 5 :

[18]:
```
M = s3[0]
Mprimeprime = M[:101]
X = np.linspace(0, 10000, 10001)
plt.plot(X,M,color = "blue",label = "M")
X = np.linspace(0, 10000, 101)
plt.plot(X,Mprimeprime,color = "red",label = "M''")
plt.title("Tracé de M et M''")
plt.xlabel('n')
plt.ylabel('Sn')
plt.legend()
plt.show
```

[18]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```



Question : Par quel facteur faut-il multiplier la taille des pas (axe des ordonnées) de M''pour que les deux marches aléatoires aient le même aspect ?

Réponse : il n y a aucun facteur par lequel on multiplie la taille des pas de M'' pour que les deux marches aléatoires aient le meme aspect car les pas de 100 à 10000 sont aléatoires et donc indépendants des pas de 0 à 100.

1.6.1 Question 6 :

Pour $p = 0.5$, générer $M = 10^q$ (d'abord pour $q = 1$, puis $q = 2$, 3et4) marches aléatoires de $N = 10000$ pas et calculer $< S_n >_q$ et $\sqrt{Var(S_n)_q}$ pour n N.

1.6.2 Réponse 6 :

- On définit d'abord une fonction qui calcule $\sqrt{Var(S_n)_q}$ pour un ensemble de marches aléatoires.
- Pour chaque $< S_n >_q$ et $\sqrt{Var(S_n)_q}$ on affiche les 10 premières et les 10 dernières valeurs

[19]:
```
def calculer_racine_variance (E): # E est l'ensemble des marches aleatoires
    moy = calculer_trajectoire_moyenne(E)
    res = []
    nb_marches = len(E) # nombre de marches
    nb_pas = len(E[0]) # nombre de pas dans les marches
    for i in range(nb_pas):
        for num_marche in range (nb_marches):
            if (num_marche==0):
                res.append((E[num_marche][i]-moy[i])**2)
            else:
                res[i] += ((E[num_marche][i]-moy[i])**2)
    res[i] = sqrt(res[i]/nb_marches)
    return res
```

- $q = 1$

[20]:
```
q1 = [] # tableau qui comporte les marches aléatoires de pas = 10000 et p = 0.5
for i in range(10):
    m = simuler_marche_aleatoire(10000,0.5)
    q1.append(m)
moy1 = calculer_trajectoire_moyenne(q1)
racine_var1 = calculer_racine_variance(q1)
print('Les 10 premières composantes de < Sn > sont : ',moy1[:10])
print('Les 10 dernières composantes de < Sn > sont : ',moy1[-10:])
print("Les 10 premières composantes de l'ecart type sont : ",racine_var1[:10])
print("Les 10 dernières composantes de l'ecart type sont : ",racine_var1[-10:])
```

```
Les 10 premières composantes de < Sn > sont :  [0.0, -0.4, -0.4, -0.2, -0.6,
-0.6, 0.0, 0.4, 0.6, 0.6]
Les 10 dernières composantes de < Sn > sont :  [35.2, 35.2, 35.2, 35.2, 35.8,
35.8, 36.4, 36.4, 36.4, 35.8]
Les 10 premières composantes de l'ecart type sont :  [0.0, 8.399999999999999,
6.400000000000002, 9.600000000000001, 16.4, 30.400000000000002, 32.0,
32.39999999999999, 32.4, 54.400000000000006]
Les 10 dernières composantes de l'ecart type sont :  [54475.6, 55105.6,
56067.599999999984, 56609.59999999999, 56329.6, 55587.59999999999,
56480.400000000016, 57030.4, 57712.40000000001, 76.24408173753554]
```

- $q = 2$

[21]:
```
q2 = [] # tableau qui comporte les marches aléatoires de pas = 10000 et p = 0.5
for i in range(100):
    m = simuler_marche_aleatoire(10000,0.5)
    q2.append(m)
moy2 = calculer_trajectoire_moyenne(q2)
racine_var2 = calculer_racine_variance(q2)
print('Les 10 premières composantes de < Sn > sont : ',moy2[:10])
print('Les 10 dernières composantes de < Sn > sont : ',moy2[-10:])
print("Les 10 premières composantes de l'ecart type sont : ",racine_var2[:10])
print("Les 10 dernières composantes de l'ecart type sont : ",racine_var2[-10:])
```

```
Les 10 premières composantes de < Sn > sont :  [0.0, 0.06, 0.02, 0.14, 0.14,
0.12, 0.28, 0.28, 0.42, 0.5]
Les 10 dernières composantes de < Sn > sont :  [-20.82, -20.82, -20.92, -20.84,
-20.8, -20.78, -20.88, -21.16, -21.12, -21.06]
Les 10 premières composantes de l'ecart type sont :  [0.0, 99.64000000000003,
203.96000000000006, 282.0399999999998, 362.04000000000025, 378.5599999999998,
456.1599999999994, 596.1599999999999, 706.3599999999997, 787.0]
Les 10 dernières composantes de l'ecart type sont :  [1042864.7600000004,
1040632.7600000002, 1038295.3600000001, 1040233.4400000002, 1041259.9999999998,
1040311.1599999999, 1037934.5600000002, 1041329.44, 1043238.56,
102.11579897351828]
```

- $q = 3$

[22]:
```
q3 = [] # tableau qui comporte les marches aléatoires de pas = 10000 et p = 0.5
for i in range(1000):
    m = simuler_marche_aleatoire(10000,0.5)
    q3.append(m)
moy3 = calculer_trajectoire_moyenne(q3)
racine_var3 = calculer_racine_variance(q3)
print('Les 10 premières composantes de < Sn > sont : ',moy3[:10])
print('Les 10 dernières composantes de < Sn > sont : ',moy3[-10:])
print("Les 10 premières composantes de l'ecart type sont : ",racine_var3[:10])
print("Les 10 dernières composantes de l'ecart type sont : ",racine_var3[-10:])
```

```
Les 10 premières composantes de < Sn > sont :  [0.0, -0.03, 0.0, -0.008, -0.024,
-0.054, -0.042, 0.002, 0.02, 0.032]
Les 10 dernières composantes de < Sn > sont :  [0.39, 0.38, 0.328, 0.42, 0.398,
0.44, 0.454, 0.452, 0.464, 0.496]
Les 10 premières composantes de l'ecart type sont :  [0.0, 999.1000000000026,
2072.0, 3279.9360000000165, 4303.423999999946, 5405.084000000009,
6490.235999999997, 7759.995999999987, 8351.599999999959, 9686.97600000008]
Les 10 dernières composantes de l'ecart type sont :  [10137463.89999999,
10131879.600000001, 10135852.416000005, 10127607.600000007, 10127257.596000014,
10130398.399999995, 10135537.883999983, 10143019.696000014, 10146832.703999996,
100.71692004822224]
```

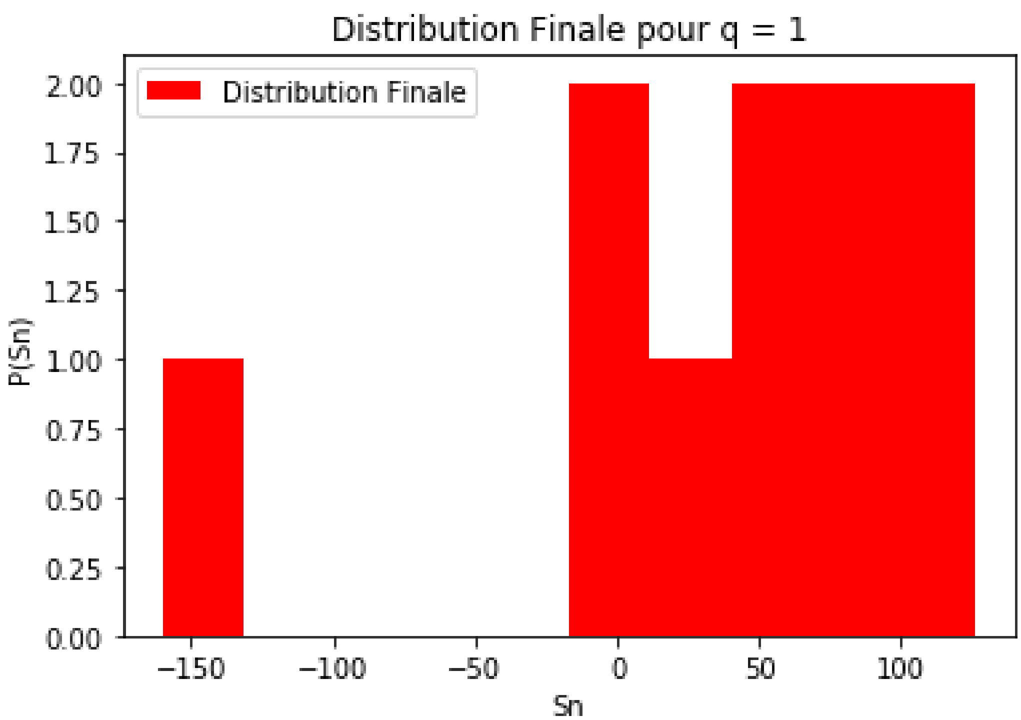
1.7.1 Question 7 :

Calculer la distribution (ie les histogrammes) $P(S_N)_q$ de la dernière position S_N .

1.7.2 Réponse 7 :

[23]:
```
pos_finale1 = []
for i in range(10):
    pos_finale1.append(q1[i][10000])
plt.hist(pos_finale1,color="red",label = "Distribution Finale")
plt.title("Distribution Finale pour q = 1")
plt.xlabel('Sn')
plt.ylabel('P(Sn)')
plt.legend()
plt.show
```

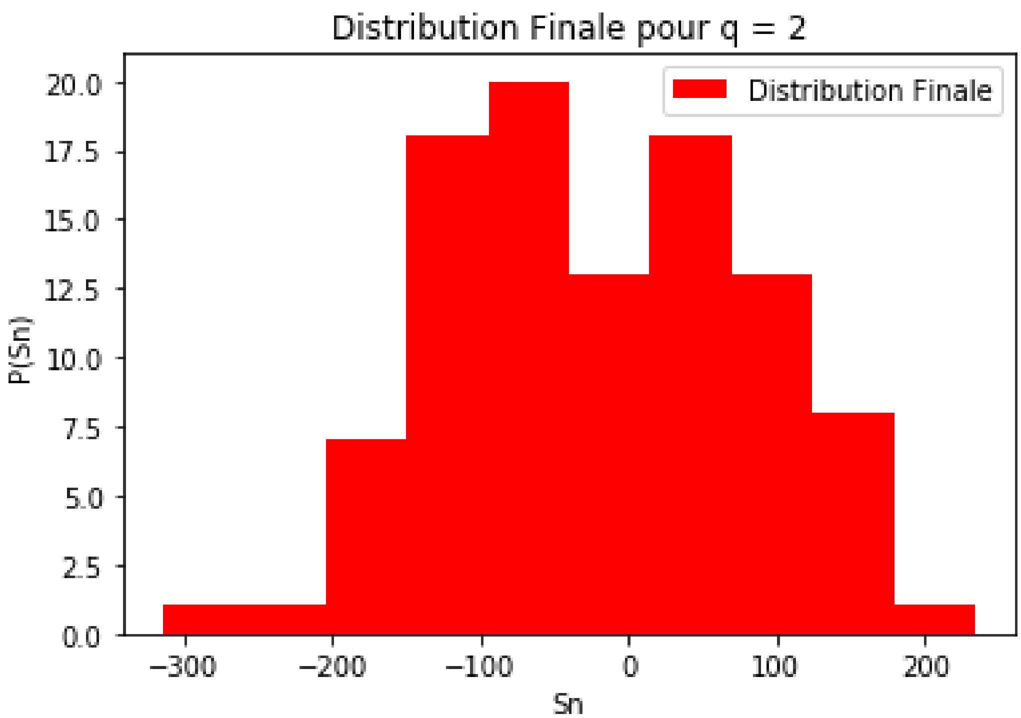
[23]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```



[24]:
```
pos_finale2 = []
for i in range(100):
    pos_finale2.append(q2[i][10000])
plt.hist(pos_finale2,color="red",label = "Distribution Finale")
plt.title("Distribution Finale pour q = 2")
plt.xlabel('Sn')
plt.ylabel('P(Sn)')
plt.legend()
```

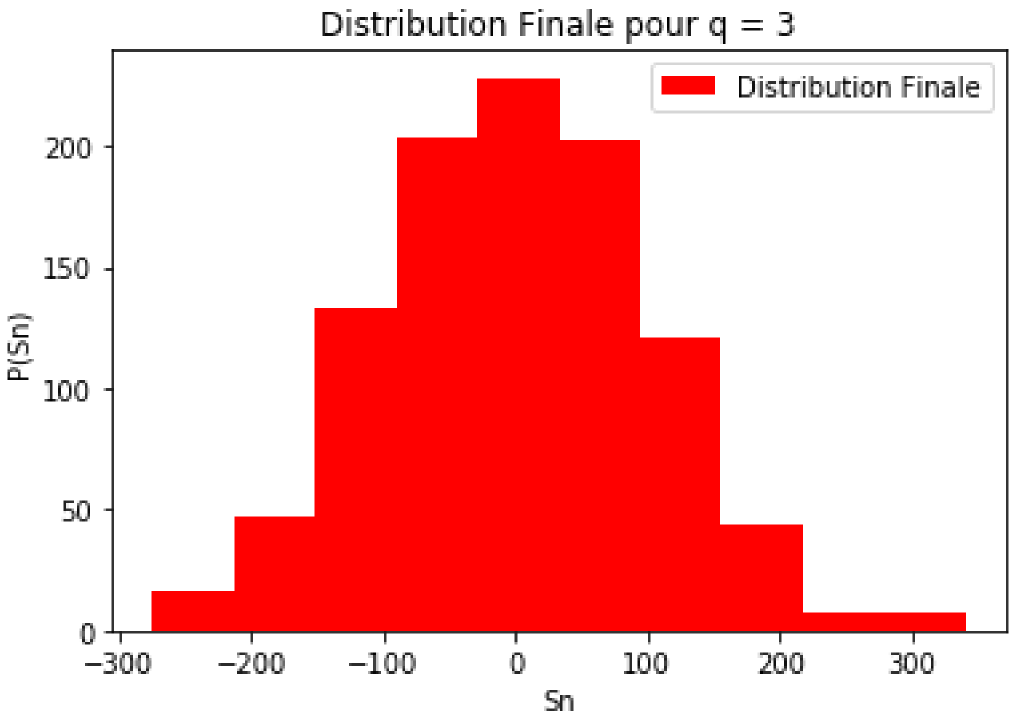
```
plt.show
```

[24]: <function matplotlib.pyplot.show(*args, **kw)>



[25]:
```
pos_finale3 = []
for i in range(1000):
    pos_finale1.append(q3[i][10000])
plt.hist(pos_finale1,color="red",label = "Distribution Finale")
plt.title("Distribution Finale pour q = 3")
plt.xlabel('Sn')
plt.ylabel('P(Sn)')
plt.legend()
plt.show
```

[25]: <function matplotlib.pyplot.show(*args, **kw)>



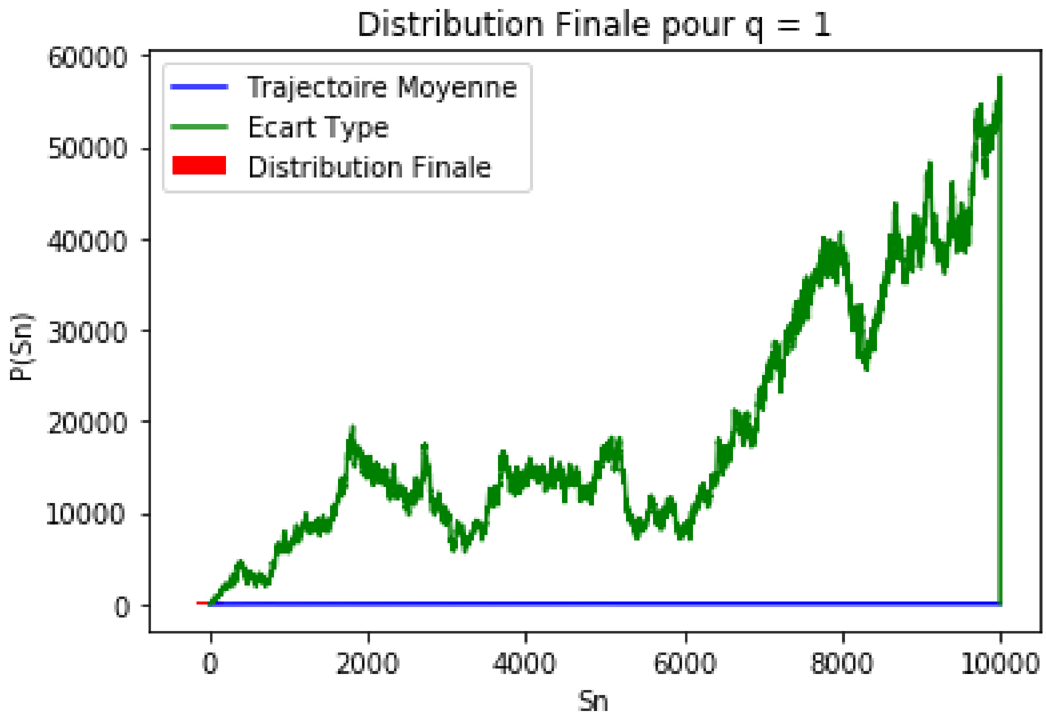
Question : Tracer sur un même graphe $< S_n >_q$ et $\sqrt{Var(S_n)_q}$ d'une part et $P(S_N)_q$ d'autre part.

Réponse :

- $q = 1$

[26]:
```
plt.hist(pos_finale1,color="red",label = "Distribution Finale")
plt.plot(moy1,color = "blue",label = "Trajectoire Moyenne")
plt.plot(racine_var1,color = "green",label = "Ecart Type")
plt.title("Distribution Finale pour q = 1")
plt.xlabel('Sn')
plt.ylabel('P(Sn)')
plt.legend()
plt.show
```

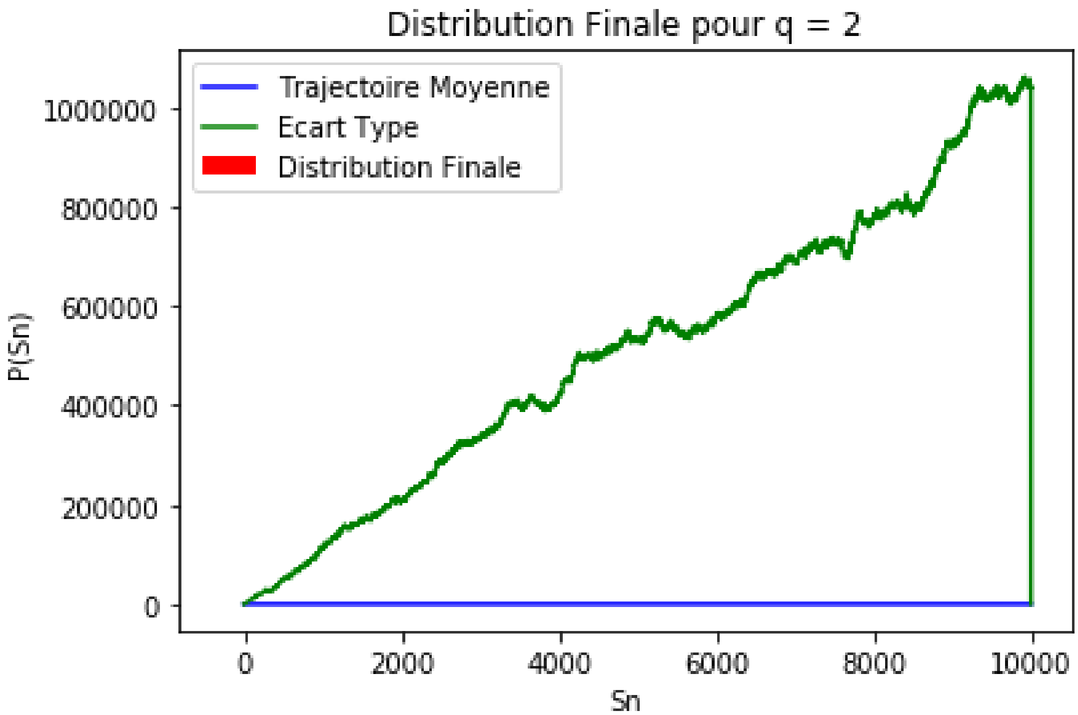
[26]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```



- $q = 2$

[27]:
```
plt.hist(pos_finale2,color="red",label = "Distribution Finale")
plt.plot(moy2,color = "blue",label = "Trajectoire Moyenne")
plt.plot(racine_var2,color = "green",label = "Ecart Type")
plt.title("Distribution Finale pour q = 2")
plt.xlabel('Sn')
plt.ylabel('P(Sn)')
plt.legend()
plt.show
```

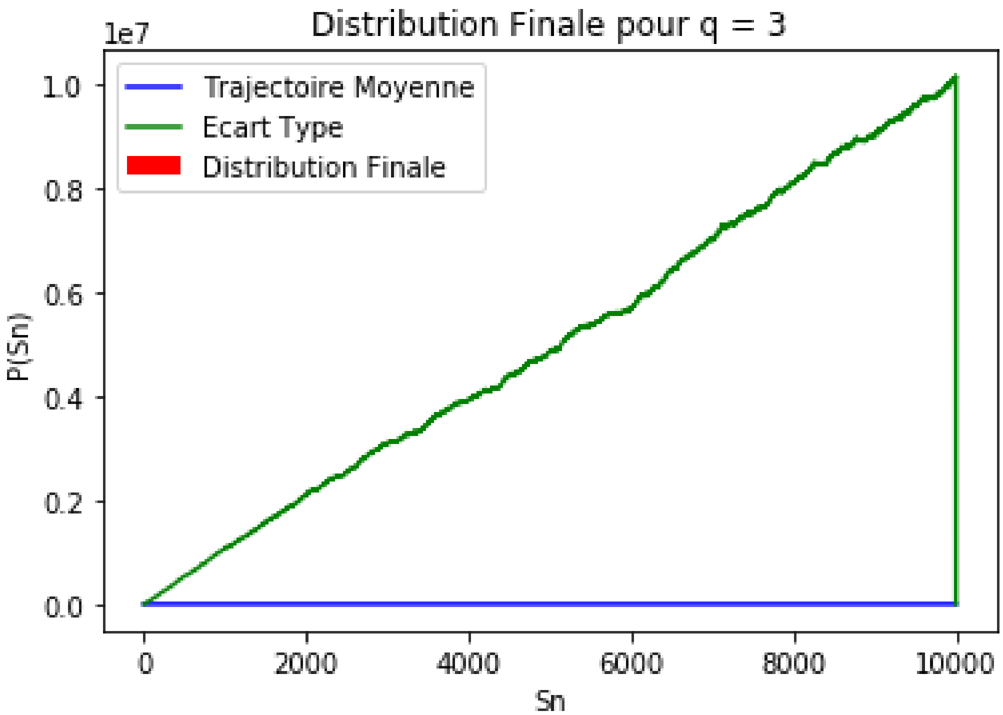
[27]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```



- $q = 3$

[28]:
```
plt.hist(pos_finale3,color="red",label = "Distribution Finale")
plt.plot(moy3,color = "blue",label = "Trajectoire Moyenne")
plt.plot(racine_var3,color = "green",label = "Ecart Type")
plt.title("Distribution Finale pour q = 3")
plt.xlabel('Sn')
plt.ylabel('P(Sn)')
plt.legend()
plt.show
```

[28]: <function matplotlib.pyplot.show(*args, **kw)>



1.8.1 Question 8 :

Quelle est la fonction qui ajuste cette distribution dans la limite M 1 ? Pourquoi ?

1.8.2 Réponse 8 :

la fonction qui ajuste cette distribution dans la limite M 1 est la distribution gaussienne, car on remarque que le nuage des positions finales est en gros centré sur la marche initiale pour M 1

Justification : Car sur le régime asymptotique: la loi binomiale se comporte asymptotiquement comme une distribution gaussienne.

2. Loi Uniforme :

Les pas de la marche aléatoire sont distribués sur une loi uniforme centréeentre -0.5 et 0.5.

2.1.1 Question 1 :

Générer quelques marches aléatoires de $N = 10000$ pas .

2.1.2 Réponse 1 :

- On commence par définir une fonction simuler_marche_aléatoire_uniforme qui simule la marche aléatoire.
- Les entrées de la fonction sont : le nombre de pas.
- La fonction retourne un tableau qui représente la marche.

[29]:
```
def simuler_marche_aleatoire_uniforme(n):
    s=0
    marche=[s]
    for i in range(n):
        s += np.random.uniform(-0.5,0.5)
        marche.append(s)
    return marche
```

- On génère quelques marches.
- On affiche uniquement les 10 premières position et la position finale des 3 premières simulations.

[30]:
```
u1 = [] # tableau qui comporte les marches aléatoires de pas = 10000
for i in range(50):
    m = simuler_marche_aleatoire_uniforme(10000)
    u1.append(m)
for u in u1[:3]:
    print ( 'Les 10 premières position (10 premières valeurs de Sn) de la␣
 ↪marche : ',u[:10])
    print ( 'La position finale de la marche : ',u[10000])
    print ('')
```

```
Les 10 premières position (10 premières valeurs de Sn) de la marche :  [0,
0.004944880424876508, 0.26545393675466744, -0.20452814922677742,
-0.5907827844456929, -0.5088369440554877, -0.7241133475863385,
-1.1484112010155605, -1.5918497096127013, -1.2214931696494422]
La position finale de la marche :  3.6397495959795982

Les 10 premières position (10 premières valeurs de Sn) de la marche :  [0,
-0.3470971129987249, -0.36750950134962823, -0.24766518585374264,
-0.14344914908557382, -0.16774015419135768, -0.18336345379005659,
-0.22713904603475632, -0.49264408528259895, -0.9538350299807841]
La position finale de la marche :  29.048930201127085

Les 10 premières position (10 premières valeurs de Sn) de la marche :  [0,
0.14133863462400498, 0.09442120344025828, -0.3354460251196413,
-0.5063572766705146, -0.7574060240010998, -0.4768404294356694,
-0.8059348150838457, -1.286182202644885, -0.8799696137835238]
La position finale de la marche :  26.225605906217645
```

2.2.1 Question 2 :

Tracer une marche aléatoire (S_n en fonction de n). et la comparer à une marche aléatoire binomiale symétrique.

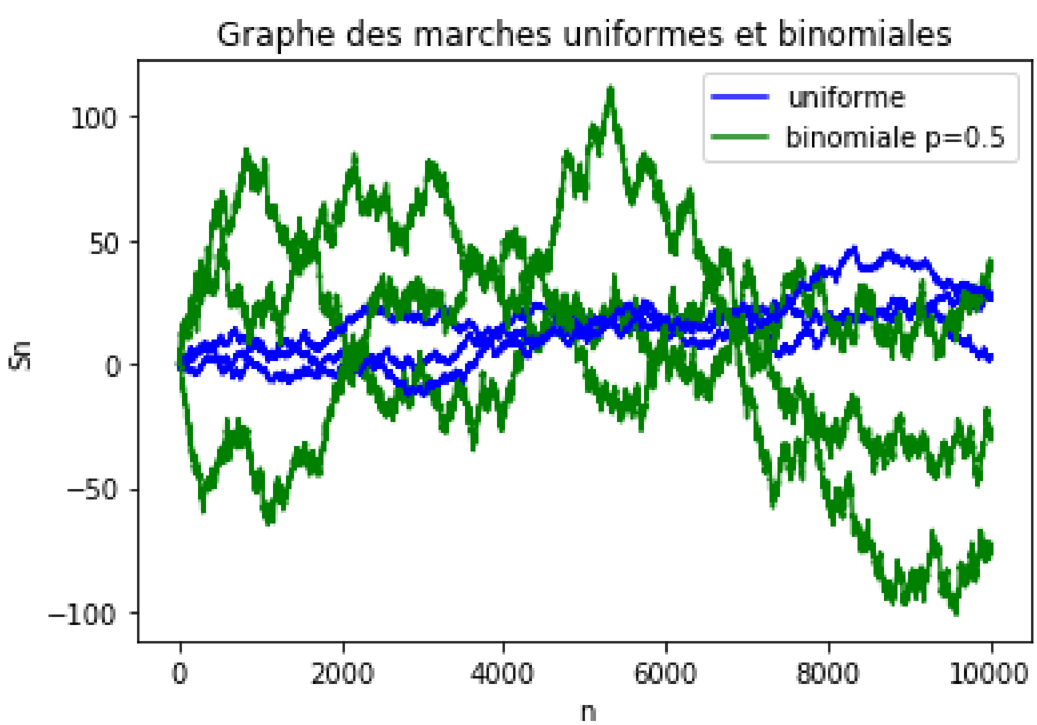
2.2.2 Réponse 2 :

- On trace 3 simulations de marche aléatoire uniforme
- On trace 3 simulations de marche aléatoire binomiale symétrique ($p = 0.5$)

- On trace toutes les marches sur le meme graphe pour pouvoir comparer.
- Les couleurs sont comme suit :
    - Uniforme —> couleur Bleue
    - binomiale avec $p = 0.5$ —> couleur Verte

[31]:
```
X = np.linspace(0, 10000, 10001)
for i in range(3):
    if (i==0):
        plt.plot(X,u1[i],color = "blue",label="uniforme")
        plt.plot(X,s3[i],color = "green",label="binomiale p=0.5")
    else:
        plt.plot(X,u1[i],color = "blue")
        plt.plot(X,s3[i],color = "green")
plt.title("Graphe des marches uniformes et binomiales")
plt.xlabel('n')
plt.ylabel('Sn')
plt.legend()
plt.show
```

[31]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```



Comparaison : * On remarque que les deux marches se comportent presque de la meme manière .

Différence : * La différence est que la marche aléatoire avec loi uniforme reste plus proche du point du départ que la loi binomiale, ceci revient à la taille du pas (entre -0.5 et 0.5) contrairement a la loi binomiale qui à un pas de soit 1 soit -1.

2.6.1 Question 6 :

Générer $M = 10^q$ (d'abord pour $q = 1$, puis $q = 2, 3et4$) marches aléatoires de $N = 10000$ pas et calculer $< S_n >_q$ et $\sqrt{Var(S_n)_q}$ pour n N.

2.6.2 Réponse 6 :

- $q = 1$

[32]:
```
r1 = [] # tableau qui comporte les marches aléatoires uniformes de pas = 10000
for i in range(10):
    m = simuler_marche_aleatoire_uniforme(10000)
    r1.append(m)
moy_u1 = calculer_trajectoire_moyenne(r1)
racine_var_u1 = calculer_racine_variance(r1)
print('Les 10 premières composantes de < Sn > sont : ',moy_u1[:10])
print('Les 10 dernières composantes de < Sn > sont : ',moy_u1[-10:])
print("Les 10 premières composantes de l'ecart type sont : ",racine_var_u1[:10])
print("Les 10 dernières composantes de l'ecart type sont : ",racine_var_u1[-10:
↪])
```

```
Les 10 premières composantes de < Sn > sont :  [0.0, -0.14834404830086362,
0.01318127130498733, -0.0736029798201475, -0.08737800212324709,
-0.21062241907833795, -0.1415075710729142, 0.026032390944068517,
-0.09377656446047992, -0.0674805771312382]
Les 10 dernières composantes de < Sn > sont :  [-5.752118171059247,
-5.634509692078727, -5.629712121953136, -5.495231707249486, -5.574228038317068,
-5.554986390607685, -5.533758600063884, -5.50031832702985, -5.456771308826383,
-5.418795927955883]
Les 10 premières composantes de l'ecart type sont :  [0.0, 0.5376278720744099,
0.6971351014446144, 1.7808289340220256, 2.8884894579740026, 2.3480435283263095,
2.9088861626918003, 5.080775975637345, 5.813455874706382, 5.031393301148903]
Les 10 dernières composantes de l'ecart type sont :  [17787.308142396832,
17859.862900061296, 17772.869996414676, 17769.09299222907, 17808.18275012156,
17861.74824701963, 17891.840818184202, 17985.850837394926, 18108.40448703909,
42.460295707542315]
```

- $q = 2$

[33]:
```
r2 = [] # tableau qui comporte les marches aléatoires uniformes de pas = 10000
for i in range(100):
    m = simuler_marche_aleatoire_uniforme(10000)
    r2.append(m)
moy_u2 = calculer_trajectoire_moyenne(r2)
racine_var_u2 = calculer_racine_variance(r2)
print('Les 10 premières composantes de < Sn > sont : ',moy_u2[:10])
```

```
print('Les 10 dernières composantes de < Sn > sont : ',moy_u2[-10:])
print("Les 10 premières composantes de l'ecart type sont : ",racine_var_u2[:10])
print("Les 10 dernières composantes de l'ecart type sont : ",racine_var_u2[-10:
↪])
```

```
Les 10 premières composantes de < Sn > sont :  [0.0, -0.02762591719791565,
-0.02556442392919921, -0.02953090161896088, -0.006001074146966785,
-0.0034269751183879883, 0.053709859922989676, 0.04494726177829704,
0.012156165063223817, -0.07299626960383424]
Les 10 dernières composantes de < Sn > sont :  [-1.913227242642683,
-1.9115258056168072, -1.9147708136805044, -1.9196038114252598,
-1.9224263393594154, -1.94071467368769, -1.952519245705711, -1.9400351861889156,
-1.9371256134668107, -1.9090715918281043]
Les 10 premières composantes de l'ecart type sont :  [0.0, 8.840422428304782,
18.88831817739317, 31.54881615124305, 35.26070892898455, 45.46452314383323,
54.7244001170986, 65.35348236966531, 77.69362938879324, 85.33288692701889]
Les 10 dernières composantes de l'ecart type sont :  [96149.1205512452,
96183.44300851543, 96306.98245592478, 96368.98620221954, 96090.97385634344,
96190.01865094915, 95880.99021592844, 95311.86849163173, 95166.88725326164,
30.884755924575526]
```

- $q = 3$

[34]:
```
r3 = [] # tableau qui comporte les marches aléatoires uniformes de pas = 10000
for i in range(1000):
    m = simuler_marche_aleatoire_uniforme(10000)
    r3.append(m)
moy_u3 = calculer_trajectoire_moyenne(r3)
racine_var_u3 = calculer_racine_variance(r3)
print('Les 10 premières composantes de < Sn > sont : ',moy_u3[:10])
print('Les 10 dernières composantes de < Sn > sont : ',moy_u3[-10:])
print("Les 10 premières composantes de l'ecart type sont : ",racine_var_u3[:10])
print("Les 10 dernières composantes de l'ecart type sont : ",racine_var_u3[-10:
↪])
```

```
Les 10 premières composantes de < Sn > sont :  [0.0, 0.005492433387563581,
-0.0034465202356714613, 0.0006055897445611897, -0.002359098976667556,
-0.0061762947584605285, -0.001558244070855136, 0.007637552240044854,
0.0256773178837503, 0.028566219948382408]
Les 10 dernières composantes de < Sn > sont :  [-0.26925578561971103,
-0.2773457465298071, -0.27753574408917464, -0.26668023323029005,
-0.25751072015070686, -0.2456968935616908, -0.23742182051208696,
-0.25120721448881245, -0.25176242799787796, -0.2510420761488508]
Les 10 premières composantes de l'ecart type sont :  [0.0, 83.74939107179269,
169.2691726141211, 255.5823815681401, 335.598907637026, 418.667776601802,
517.7723258578532, 555.5139471016241, 650.8762593735329, 716.366969273086]
Les 10 dernières composantes de l'ecart type sont :  [830421.8028149178,
830612.3825403355, 831138.5104215442, 830971.1867843167, 831107.1336349878,
831397.9459072703, 831753.8995868341, 832046.3783042023, 833234.5626977326,
```

28.863711326763465]

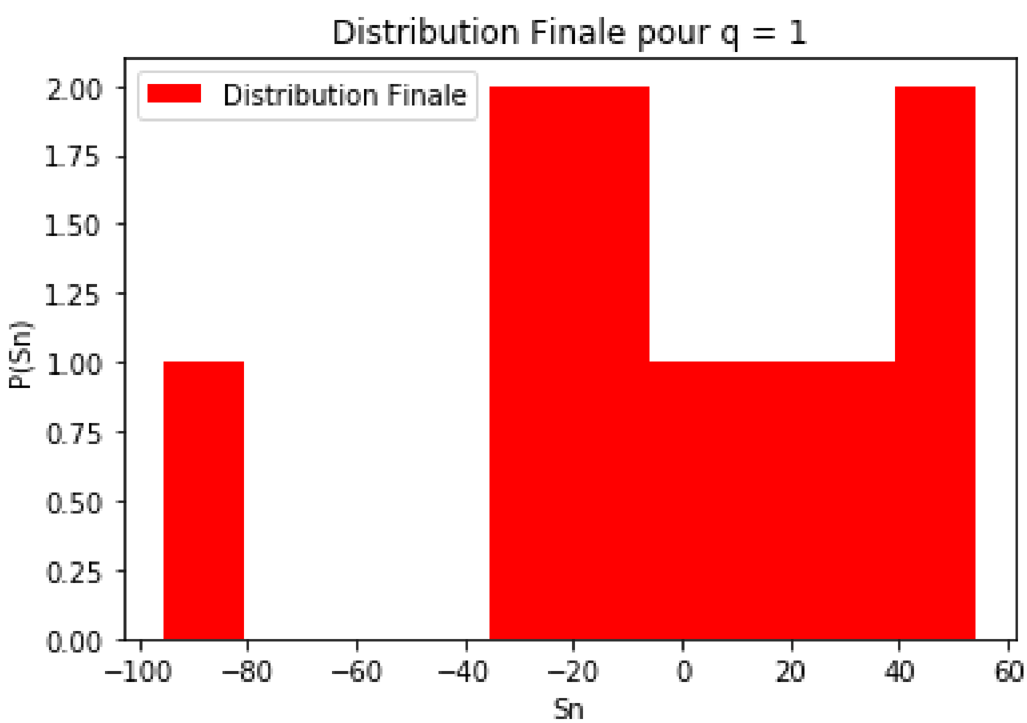
2.7.1 Question 7 :

Calculer la distribution (ie les histogrammes) $P(S_N)_q$ de la dernière position S_N .

2.7.2 Réponse 7 :

[35]:
```
pos_finale_uniforme_1 = []
for i in range(10):
    pos_finale_uniforme_1.append(r1[i][10000])
plt.hist(pos_finale_uniforme_1,color="red",label = "Distribution Finale")
plt.title("Distribution Finale pour q = 1")
plt.xlabel('Sn')
plt.ylabel('P(Sn)')
plt.legend()
plt.show
```

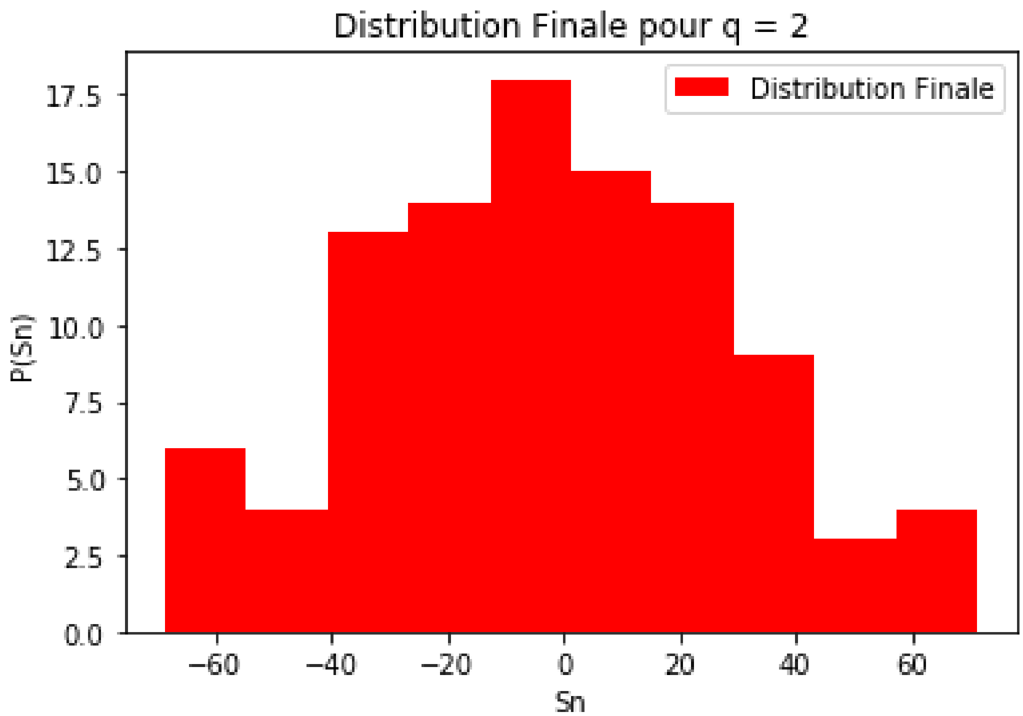
[35]: <function matplotlib.pyplot.show(*args, **kw)>



[36]:
```
pos_finale_uniforme_2 = []
for i in range(100):
    pos_finale_uniforme_2.append(r2[i][10000])
plt.hist(pos_finale_uniforme_2,color="red",label = "Distribution Finale")
plt.title("Distribution Finale pour q = 2")
```

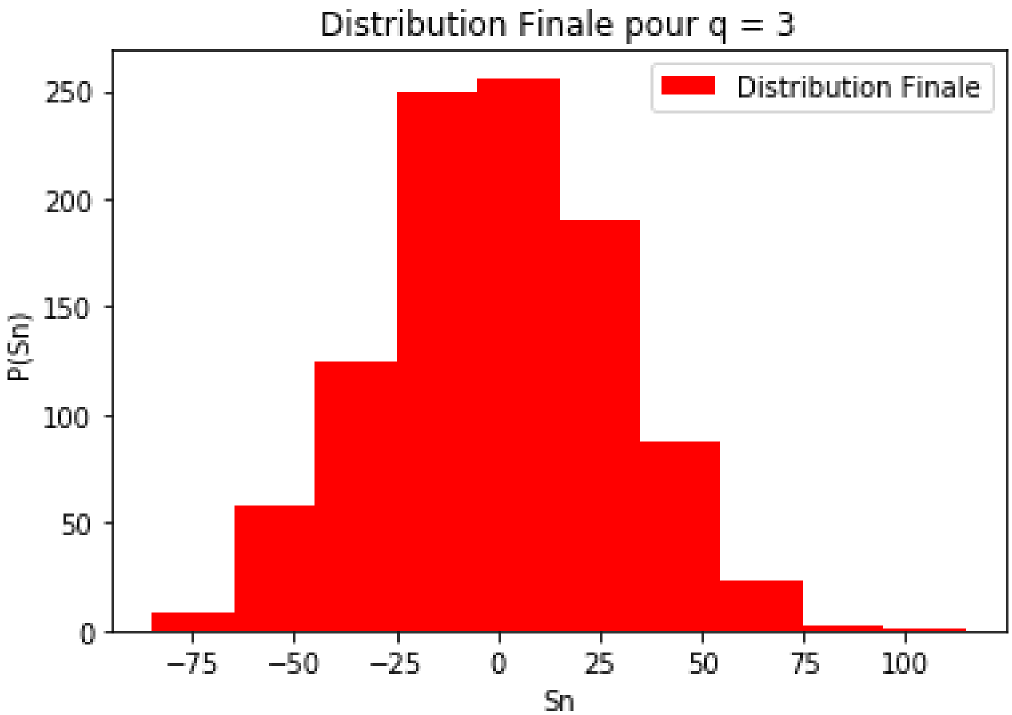
```
plt.xlabel('Sn')
plt.ylabel('P(Sn)')
plt.legend()
plt.show
```

[36]: <function matplotlib.pyplot.show(*args, **kw)>



[37]:
```
pos_finale_uniforme_3 = []
for i in range(1000):
    pos_finale_uniforme_3.append(r3[i][10000])
plt.hist(pos_finale_uniforme_3,color="red",label = "Distribution Finale")
plt.title("Distribution Finale pour q = 3")
plt.xlabel('Sn')
plt.ylabel('P(Sn)')
plt.legend()
plt.show
```

[37]: <function matplotlib.pyplot.show(*args, **kw)>



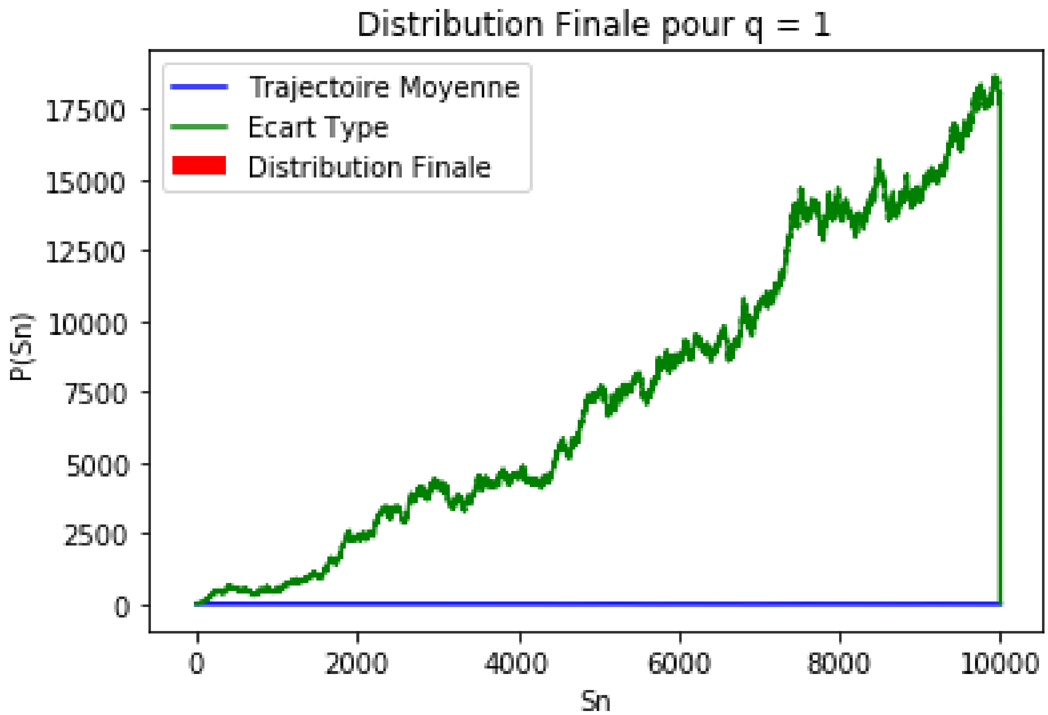
Question : Tracer sur un même graphe $< S_n >_q$ et $\sqrt{Var(S_n)_q}$ d'une part et $P(S_N)_q$ d'autre part.

Réponse :

- $q = 1$

[38]:
```
plt.hist(pos_finale_uniforme_1,color="red",label = "Distribution Finale")
plt.plot(moy_u1,color = "blue",label = "Trajectoire Moyenne")
plt.plot(racine_var_u1,color = "green",label = "Ecart Type")
plt.title("Distribution Finale pour q = 1")
plt.xlabel('Sn')
plt.ylabel('P(Sn)')
plt.legend()
plt.show
```

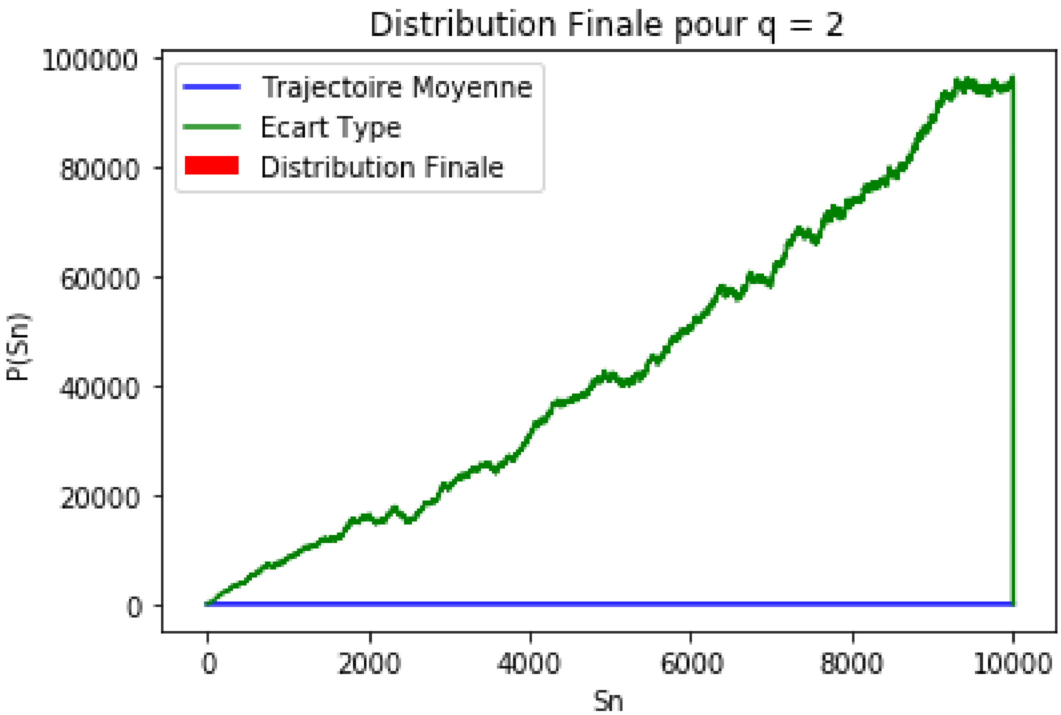
[38]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```



- $q = 2$

[39]:
```
plt.hist(pos_finale_uniforme_2,color="red",label = "Distribution Finale")
plt.plot(moy_u2,color = "blue",label = "Trajectoire Moyenne")
plt.plot(racine_var_u2,color = "green",label = "Ecart Type")
plt.title("Distribution Finale pour q = 2")
plt.xlabel('Sn')
plt.ylabel('P(Sn)')
plt.legend()
plt.show
```

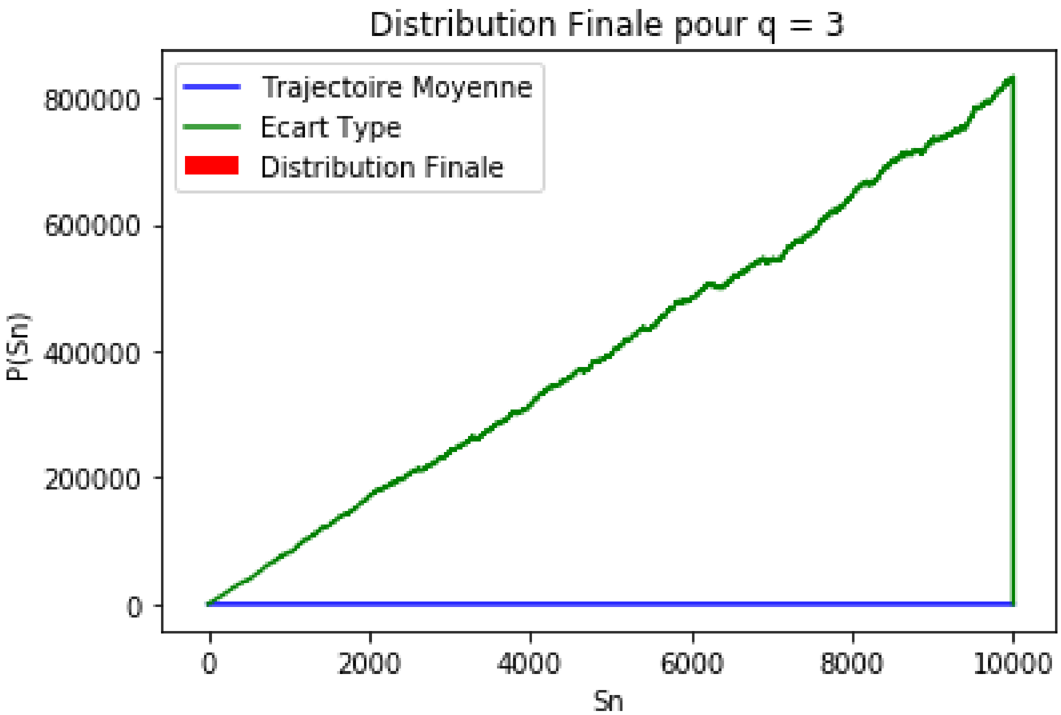
[39]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```



- $q = 3$

[40]:
```
plt.hist(pos_finale_uniforme_3,color="red",label = "Distribution Finale")
plt.plot(moy_u3,color = "blue",label = "Trajectoire Moyenne")
plt.plot(racine_var_u3,color = "green",label = "Ecart Type")
plt.title("Distribution Finale pour q = 3")
plt.xlabel('Sn')
plt.ylabel('P(Sn)')
plt.legend()
plt.show
```

[40]: <function matplotlib.pyplot.show(*args, **kw)>



2.8.1 Question 8 :

Quelle est la fonction qui ajuste cette distribution dans la limite M 1 ? Pourquoi ?

1.8.2 Réponse 8 :

la fonction qui ajuste cette distribution dans la limite M 1 est la distribution gaussienne, car on remarque que le nuage des positions finales est en gros centré sur la marche initiale pour M 1

Justification : Car sur le régime asymptotique: la loi uniforme se comporte asymptotiquement comme une distribution gaussienne.