Generadores pseudoaleatorios

August 30, 2024

Alumno: Román Ciro Martin

En este trabajo se realiza un generador de números aleatorios por congruencia lineal con un corto período de manera introductoria, el cual denominaremos como ra(). Luego se realizará un generador con un período mayor llamado ran(). Estos generadores constan de una función lineal a la cual se le aplica una congruencia es decir:

$$x_{n+1} \equiv (a \cdot x_n + c) \pmod{M}$$

El cual generará M números pseudo-aleatorios si se cumple que: c y M son coprimos, es decir, su máximo común divisor es 1. a-1 es múltiplo de todos los factores primos (q) de M, es decir, $a \equiv 1 \pmod{q}$. Con este generador se realizarán simulaciones de experimentos y distribuciones estadísticas.

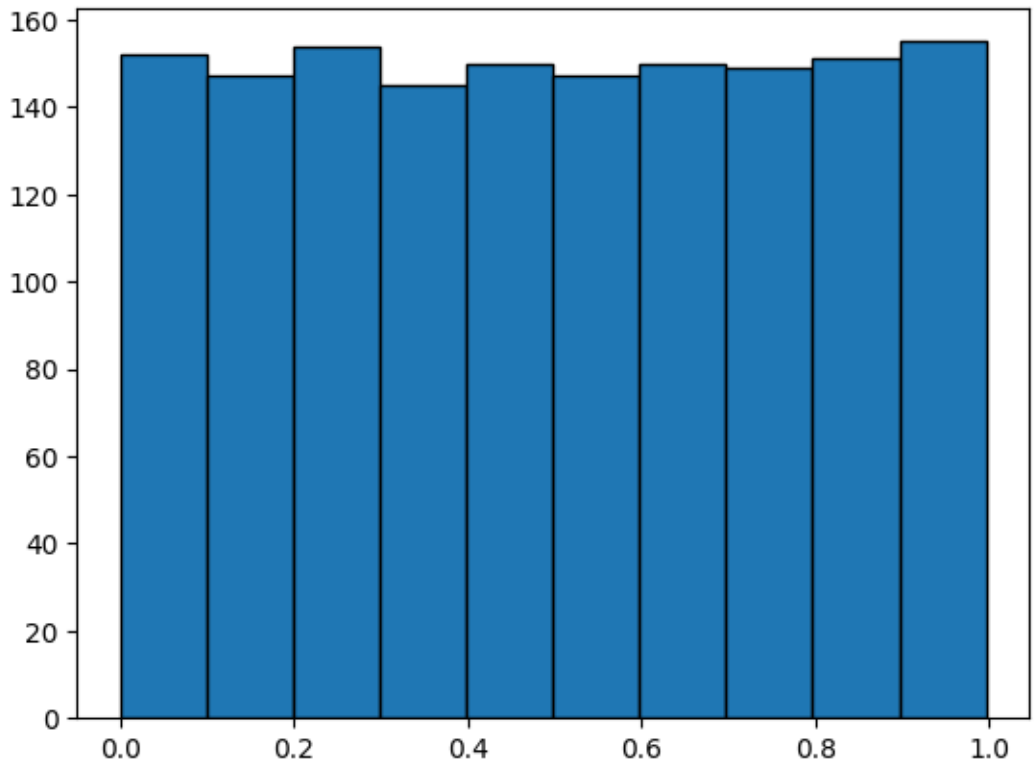
Ejercicio 16 (a)

[ ]:
```
import matplotlib.pyplot as plt
import numpy as np
```

[ ]:
```
def ra(a=57, c=1, M=256): #Defino el generador con los parámetros solicitados
    ra.semilla=((a*ra.semilla+c)%M)
    return ra.semilla/M
```

[ ]:
```
n=1500 #Cantidad de números a generar
ra.semilla=10 #Semilla solicitada
lista_de_prueba=[] #defino la lista
for i in range(n): #Bucle que genera números y los mete en la lista
    lista_de_prueba.append(ra())

print(lista_de_prueba[:5])

plt.hist(lista_de_prueba,edgecolor='black')
plt.show()
```

```
[0.23046875, 0.140625, 0.01953125, 0.1171875, 0.68359375]
```



[ ]:
```
#Testeo de período


lista=[] #Defino la lista
cuanto = 0 #Contador

#La idea es armar un bucle que genere valores indefinidamente hasta encontrar
 ↪una repetición.

while True: #Bucle
    numerito=ra()
    if numerito in lista:
        print('Período de', cuanto)
        break
    else:
        lista.append(numerito) #Añade el número a la lista
        cuanto=cuanto+1 #Conteo de números generados

print(lista[:5])
```

```
Período de 256
[0.26171875, 0.921875, 0.55078125, 0.3984375, 0.71484375]
```

Observamos que posee un período $P = 256$, demasiado pequeño para realizar un trabajo serio. Veamos sus momentos y luego comparemos con un generador mejor. Los momentos teóricos de orden k para nuestra distibución vienen dados por:

$$\int_0^1 x^k p(x)\,dx = \int_0^1 x^k\,dx = \left[\frac{x^{k+1}}{k+1}\right]_0^1 = \frac{1}{k+1}$$

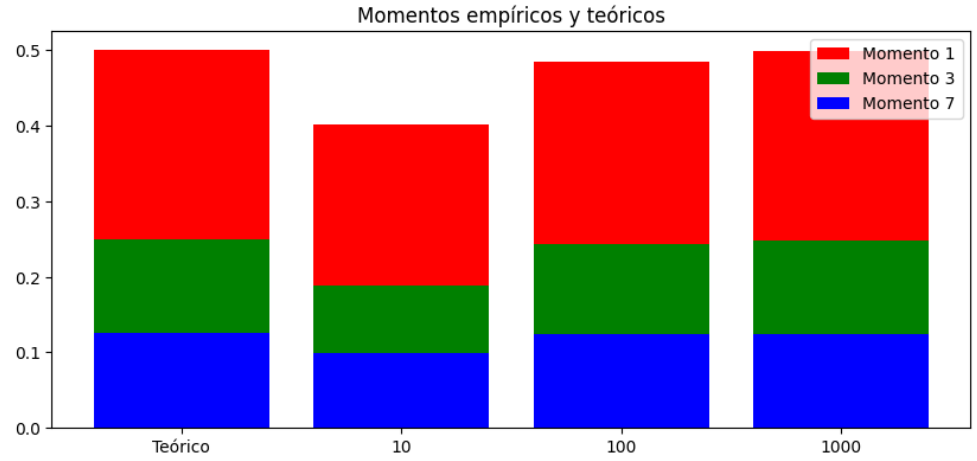
Por lo tanto tenemos que:

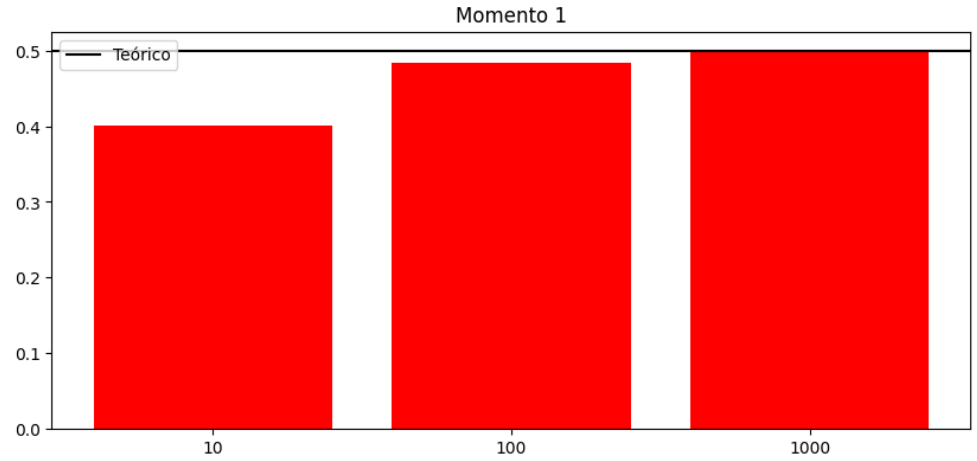
[ ]:
```
#Primero para n=10
n=10 #Cantidad de números a generar
ra.semilla=10 #Semilla solicitada
lista_pequeña=[] #defino la lista
for i in range(n): #Bucle que genera números y los mete en la lista
    lista_pequeña.append(ra())


fig, (ax1, ax2, ax3, ax4) = plt.subplots(4, 1, figsize=(10, 20))  # Creo 4␣
 ↪gráficos


lista_pequeña=np.array(lista_pequeña)

k_1 = (np.sum(lista_pequeña))/len(lista_pequeña) #Momento 1, la media.

k_3 = (np.sum(lista_pequeña**3))/len(lista_pequeña) #Momento 3

k_7 = (np.sum(lista_pequeña**7))/len(lista_pequeña) #Momento 7


teo_1=1/2
teo_3=1/4
teo_7=1/8

ax1.bar('Teórico',teo_1, color='red', label='Momento 1')
ax1.bar('Teórico',teo_3, color='green', label='Momento 3')
ax1.bar('Teórico',teo_7, color='blue', label='Momento 7')


ax1.bar('10',k_1, color='red')
ax1.bar('10', k_3, color='green')
ax1.bar('10', k_7, color='blue')


ax2.axhline(y=teo_1, color='black', label='Teórico')
ax3.axhline(y=teo_3, color='black', label='Teórico')
ax4.axhline(y=teo_7, color='black', label='Teórico')
ax2.bar('10',k_1, color='red')
```

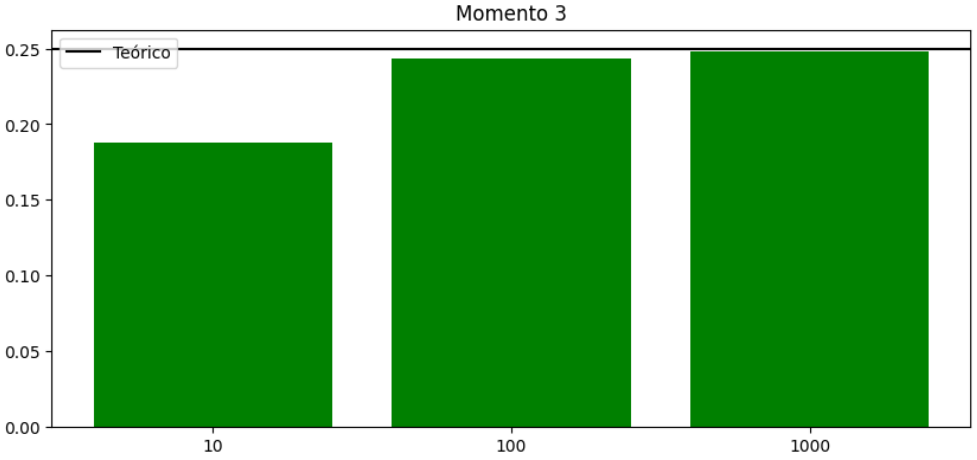
```
ax3.bar('10',k_3, color='green')
ax4.bar('10',k_7, color='blue')


print('Los momentos empíricos 1, 3 y 7 respectivamente:' , k_1 , k_3, k_7 ,␣
 ↪'para n=10 números generados')

#Ahora para n=100

n=100 #Cantidad de números a generar
ra.semilla=10 #Semilla solicitada
lista_media=[] #defino la lista
for i in range(n): #Bucle que genera números y los mete en la lista
    lista_media.append(ra())


lista_media=np.array(lista_media)

k_1 = (np.sum(lista_media))/len(lista_media) #Momento 1, la media.

k_3 = (np.sum(lista_media**3))/len(lista_media) #Momento 3

k_7 = (np.sum(lista_media**7))/len(lista_media) #Momento 7

ax1.bar('100',k_1, color='red')
ax1.bar('100', k_3, color='green')
ax1.bar('100', k_7, color='blue')

ax2.bar('100',k_1, color='red')
ax3.bar('100',k_3, color='green')
ax4.bar('100',k_7, color='blue')


print('Los momentos empíricos 1, 3 y 7 respectivamente:' , k_1 , k_3, k_7 ,␣
 ↪'para n=100 números generados')

#Finalmente para n=1000

n=1000 #Cantidad de números a generar
ra.semilla=10 #Semilla solicitada
lista_grande=[] #defino la lista
for i in range(n): #Bucle que genera números y los mete en la lista
    lista_grande.append(ra())


lista_grande=np.array(lista_grande)
```

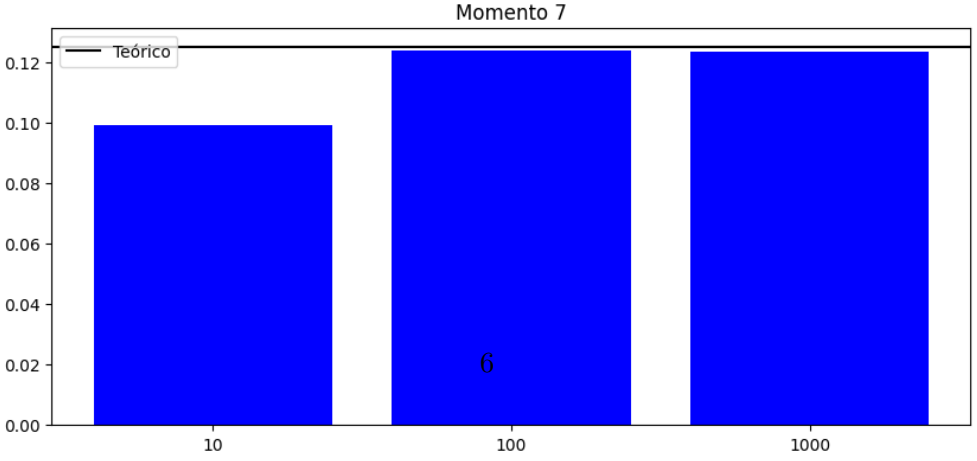
```
k_1 = (np.sum(lista_grande))/len(lista_grande) #Momento 1, la media.

k_3 = (np.sum(lista_grande**3))/len(lista_grande) #Momento 3

k_7 = (np.sum(lista_grande**7))/len(lista_grande) #Momento 7

ax1.bar('1000',k_1, color='red')
ax1.bar('1000', k_3, color='green')
ax1.bar('1000', k_7, color='blue')

ax2.bar('1000',k_1, color='red')
ax3.bar('1000',k_3, color='green')
ax4.bar('1000',k_7, color='blue')


print('Los momentos empíricos 1, 3 y 7 respectivamente:' , k_1 , k_3, k_7 ,␣
 ↪'para n=1000 números generados')


print('Los momentos teóricos por otra parte son' , teo_1, teo_3, 'y', teo_7)

ax1.legend(['Momento 1', 'Momento 3', 'Momento 7'])
ax2.legend(['Teórico'])
ax3.legend(['Teórico'])
ax4.legend(['Teórico'])
ax1.title.set_text('Momentos empíricos y teóricos')
ax2.title.set_text('Momento 1')
ax3.title.set_text('Momento 3')
ax4.title.set_text('Momento 7')

plt.show()
```

```
Los momentos empíricos 1, 3 y 7 respectivamente: 0.401171875 0.18767234683036804
0.09917423321751721 para n=10 números generados
Los momentos empíricos 1, 3 y 7 respectivamente: 0.484453125 0.243148512840271
0.12405638831464044 para n=100 números generados
Los momentos empíricos 1, 3 y 7 respectivamente: 0.498265625 0.2484909987449646
0.12350282956002501 para n=1000 números generados
Los momentos teóricos por otra parte son 0.5 0.25 y 0.125
```


Momentos empíricos y teóricos
Momento 1
Momento 3
Momento 7
0.0
0.1
0.2
0.3
0.4
0.5
Teórico
10
100
1000


Momento 1
Teórico
0.0
0.1
0.2
0.3
0.4
0.5
10
100
1000

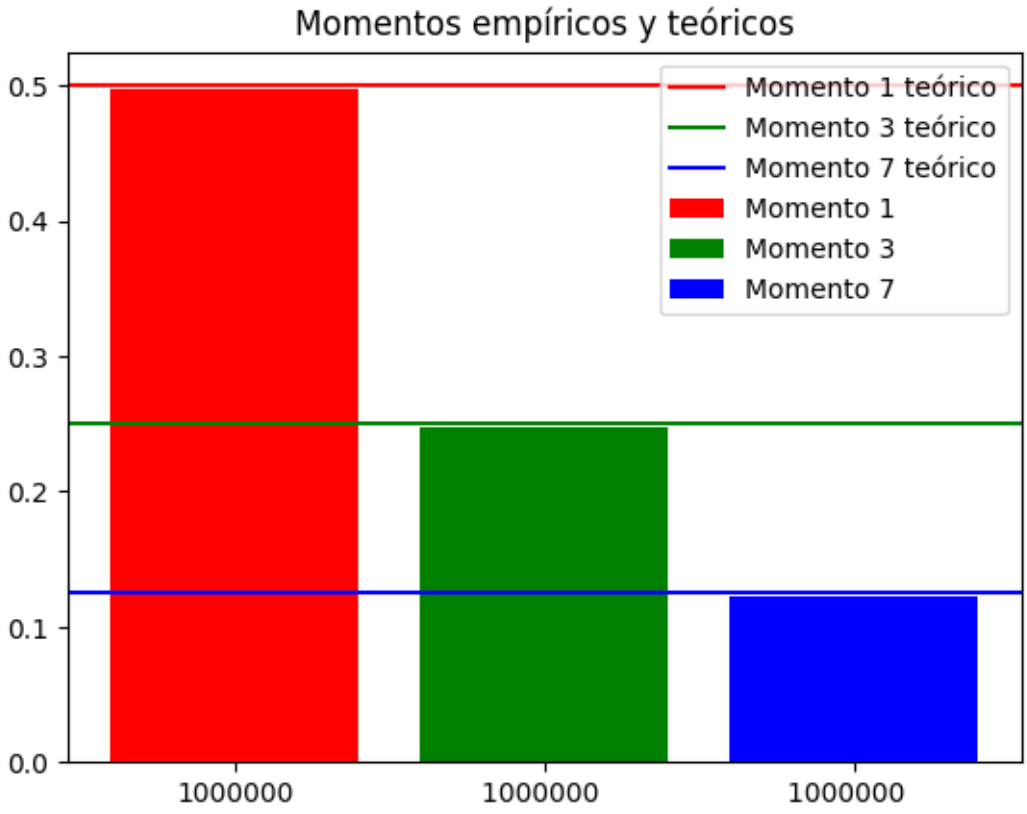

Momento 3
Teórico
0.00
0.05
0.10
0.15
0.20
0.25
10
100
1000


Momento 7
Teórico
0.00
0.02
0.04
0.06
0.08
0.10
0.12
10
100
1000

Con más números ha arrojado valores más próximos, finalmente probemos con un generador mejor, es decir, con un período mayor.

[ ]:
```
#Cálculo de momentos
#Primero, normalizamos el generador:

def ran(a=1664525, c=1013904223, M=2**32): #Defino el generador con los␣
 ↪parámetros óptimos para tener un período de 2**32
    ran.current=((a*ran.current+c)%M) #Busca el atributo seed, y lo cambia
    return ran.current/M



#Generamos la lista:
listorti=[]
n=1000000 #Cantidad de números a generar
ran.current=47 #Semilla
for i in range(n): #Bucle que genera números y los mete en la lista
    listorti.append(ra())

valores = np.array(listorti)

k_1 = (np.sum(valores))/len(valores) #Momento 1, la media.

k_3 = (np.sum(valores**3))/len(valores) #Momento 3

k_7 = (np.sum(valores**7))/len(valores) #Momento 7


teo_1=1/2
teo_3=1/4
teo_7=1/8


plt.axhline(y=teo_1, color='red', label='Momento 1 teórico')
plt.axhline(y=teo_3, color='green', label='Momento 3 teórico')
plt.axhline(y=teo_7, color='blue', label='Momento 7 teórico')

plt.bar('1000000',k_1, color='red', label='Momento 1')
plt.bar('1000000 ', k_3, color='green', label='Momento 3')
plt.bar('1000000  ', k_7, color='blue', label='Momento 7')
plt.legend()
plt.title('Momentos empíricos y teóricos')
plt.show()
```

```
print('Los momentos empíricos 1, 3 y 7 respectivamente:' , k_1 , k_3, k_7)
print('Los momentos teóricos por otra parte son' , teo_1, teo_3, teo_7)
```




```
Los momentos empíricos 1, 3 y 7 respectivamente: 0.498045625 0.24804969896316528
0.12305479531339267
Los momentos teóricos por otra parte son 0.5 0.25 0.125
```

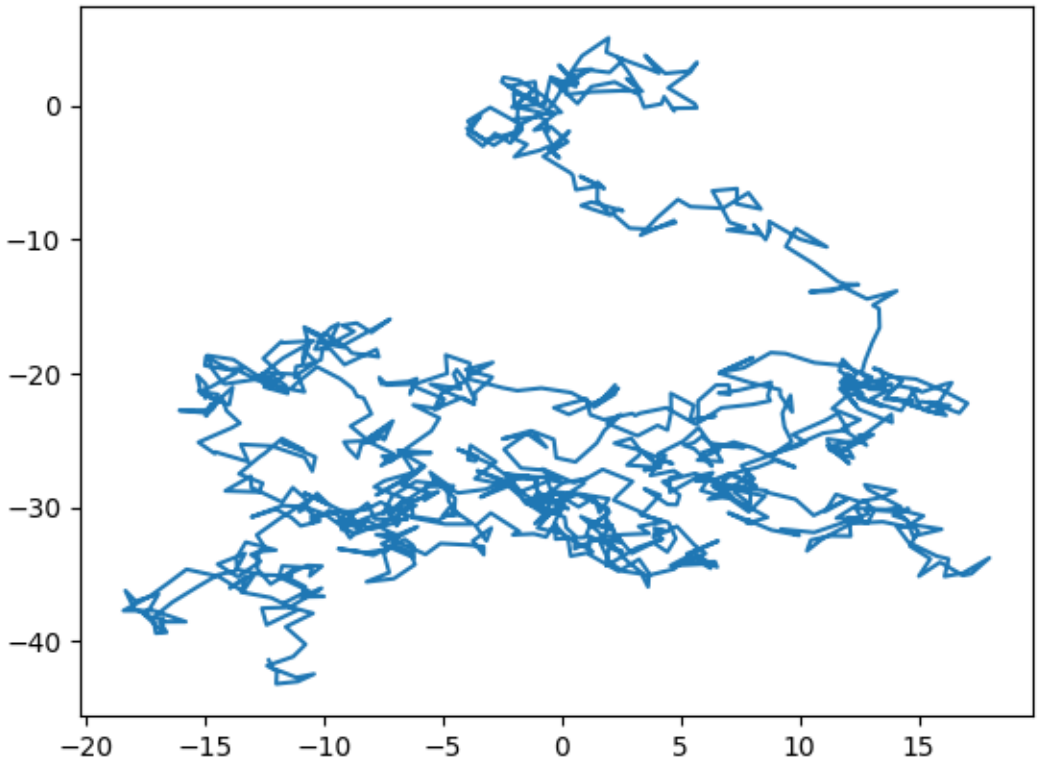
Donde observamos buenas aproximaciones a los valores teóricos.

(b)

[ ]:
```
def ran(a=1664525, c=1013904223, M=2**32): #Defino el generador con los␣
 ↪parámetros solicitados
    ran.current=((a*ran.current+c)%M) #Busca el atributo seed, y lo cambia
    return ran.current/M
```

[ ]:
```
#Si quiero valores en el intervalo [a,b] necesito una transformación lineal␣
 ↪y=(b-a)x+a en este caso particular vamos a tener a=-b:
#y=2bx-b será la transformación
ran.current=47
N=1000 #Cantidad de números a generar
b=np.sqrt(2)
```

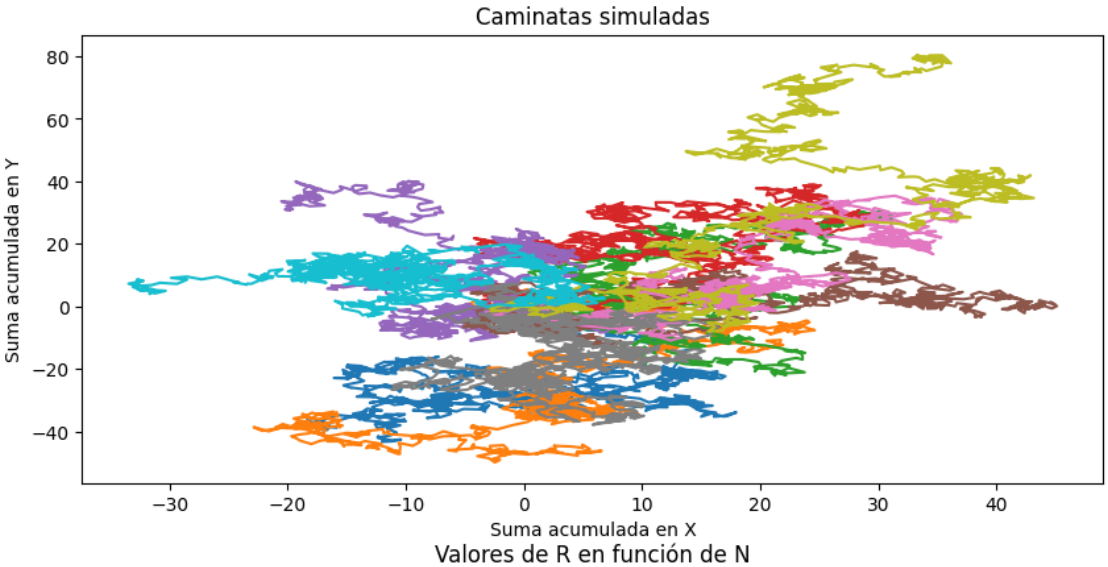
```
lista_x=np.zeros(N)
lista_y=np.zeros(N)
R_valores = np.zeros(N)

for i in range(N): #Bucle que genera números y los mete en una lista
    lista_x[i]=ran()*2*b-b
    lista_y[i]=ran()*2*b-b
plt.show()

#Defino las caminatas
caminata_x = np.cumsum(lista_x)
caminata_y = np.cumsum(lista_y)
plt.plot(caminata_x,caminata_y)
plt.show()
```

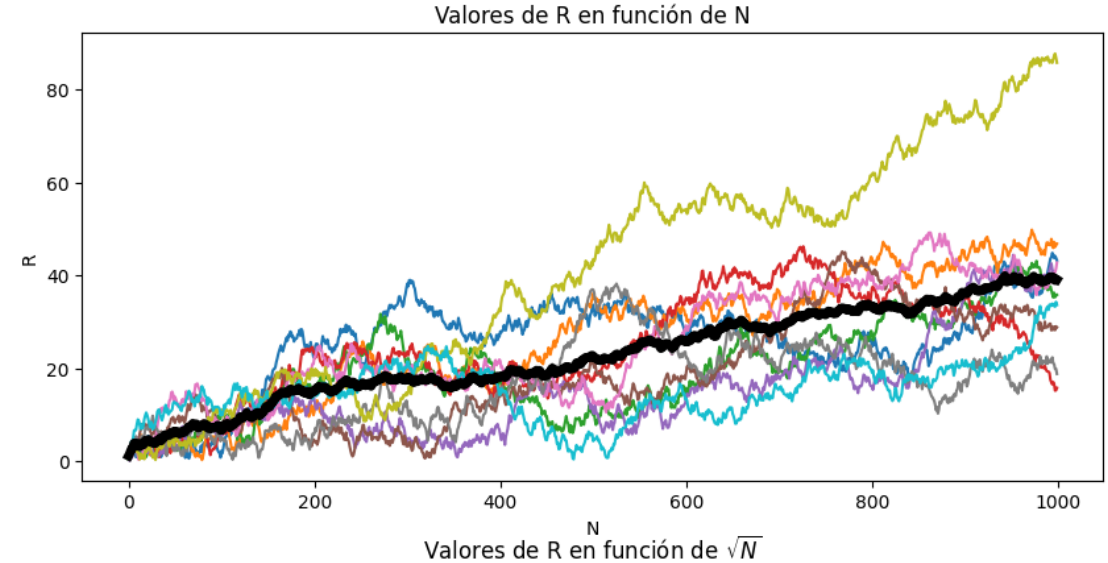


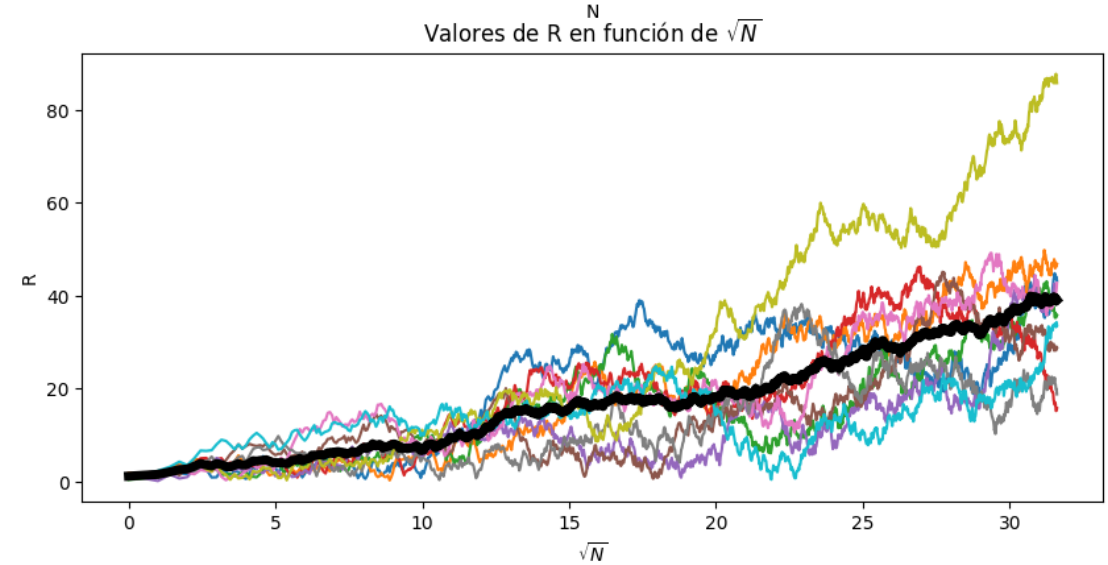
Se nos solicita 10 de estas, por lo que ahora hacemos un lazo que lo haga 10 veces en un mismo gráfico. Además calcularemos la distancia al origen para cada una de ellas.

[ ]:
```
cantidad = 10
N = 1000
ran.current = 47
distancia = np.zeros((cantidad,N))
```

```
fig, (ax1, ax2, ax3) = plt.subplots(3, 1, figsize=(10, 15))  # Creo 3 gráficos

for j in range(cantidad):

    b = np.sqrt(2)
    lista_x = np.zeros(N)
    lista_y = np.zeros(N)
    R_valores = np.zeros(N)

    for i in range(N):
        lista_x[i] = ran() * 2 * b - b
        lista_y[i] = ran() * 2 * b - b

    caminata_x = np.cumsum(lista_x)
    caminata_y = np.cumsum(lista_y)
    distancia[j]=np.sqrt(caminata_x**2 + caminata_y**2)

    ax1.plot(caminata_x, caminata_y)  # Gráfico de caminatas

    ax2.plot(range(N), distancia[j])  # Gráfico de R en función de N



    ax3.plot(np.sqrt(range(N)), distancia[j])  #Gráfico de R en función de np.
 ↪sqrt(N)

ax2.plot(distancia.mean(axis=0), lw=5, color='black', label='Valor de␣
 ↪expectación')

ax3.plot(np.sqrt(range(N)) ,distancia.mean(axis=0), lw=5, color='black',␣
 ↪label='Valor de expectación')

ax1.set_title('Caminatas simuladas')
ax1.set_xlabel('Suma acumulada en X')
ax1.set_ylabel('Suma acumulada en Y')

ax2.set_title('Valores de R en función de N')
ax2.set_xlabel('N')
ax2.set_ylabel('R')

ax3.set_title('Valores de R en función de $\sqrt{N}$')
ax3.set_xlabel('$\sqrt{N}$')
ax3.set_ylabel('R')

plt.show()
```


Caminatas simuladas
Suma acumulada en Y
Suma acumulada en X
80
60
40
20
0
−20
−40
−30
−20
−10
0
10
20
30
40


Valores de R en función de N
R
N
0
200
400
600
800
1000


Valores de R en función de $\sqrt{N}$
R
$\sqrt{N}$
0
5
10
15
20
25
30

Donde podemos observar el promedio como una línea negra y gruesa.

Ejercicio17

August 30, 2024

Ejercicio 17:

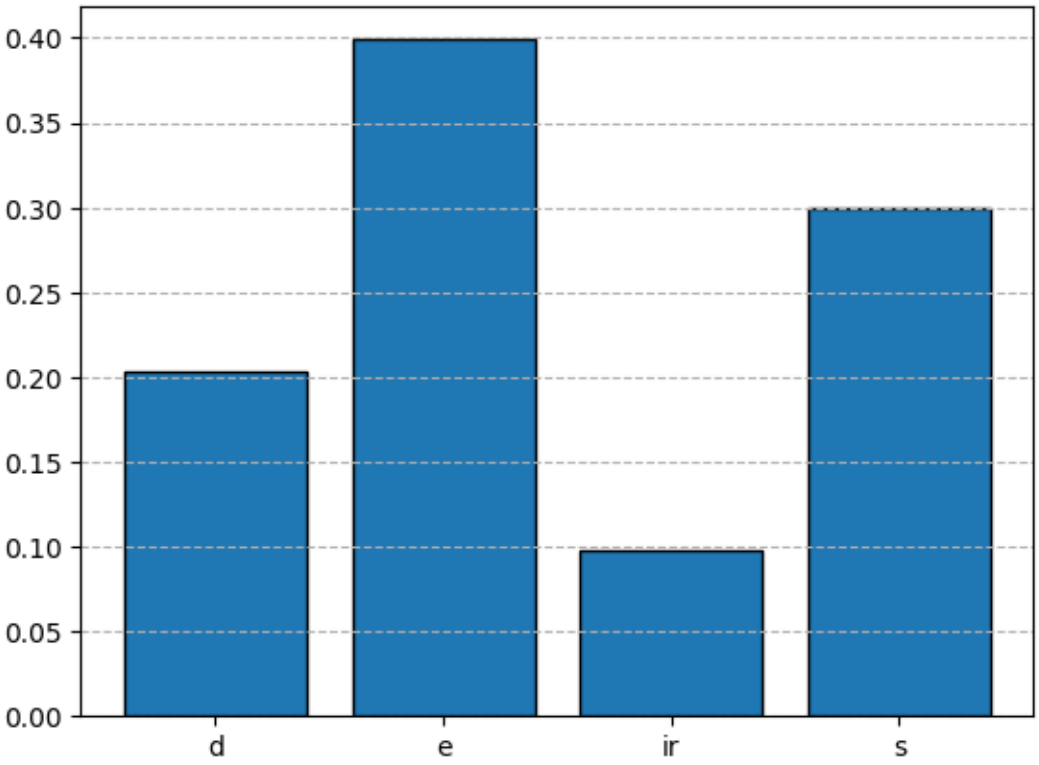
[ ]:
```
#Primero, usamos el generador del ejercicio 16 e importamos numpy y matplotlib␣
 ↪para tenerlo a mano:
import numpy as np
import matplotlib.pyplot as plt
def ran(a=1664525, c=1013904223, M=2**32): #Defino el generador con los␣
 ↪parámetros solicitados
    ran.current=((a*ran.current+c)%M) #Busca el atributo seed, y lo cambia
    return ran.current/M

ran.current = 12122002 #Semilla
```

[ ]:
```
#Planteo la muestra:
probabilidades = [0.4, 0.3, 0.2, 0.1]
tipos_gx = ['e', 's', 'd', 'ir']
prob_acumuladas = np.cumsum(probabilidades) #Calculo las probabilidades␣
 ↪acumuladas.
```

[ ]:
```
# Genero una lista aleatoria y las meto en cada tipo en función de si es menor␣
 ↪a la probabilidad acumulada.
n = 10000  # Cantidad de galaxias a generar
gx = []

for i in range(n):
    r = ran() #Genero un número aleatorio
    if r < prob_acumuladas[0]: #Si es menor a 0.4, es elíptica.
        gx.append('e')
    elif r < prob_acumuladas[1]: #Si es menor a 0.4+0.3=0.7 pero mayor a 0.4,␣
 ↪es espiral.
        gx.append('s')
    elif r < prob_acumuladas[2]: #Si es menor a 0.7+0.2=0.9 pero mayor a 0.7,␣
 ↪es irregular.
        gx.append('d')
    else:
        gx.append('ir') #Si es mayor a 0.9, es irregular.
```

[ ]:
```
categ , frecuencia = np.unique(gx, return_counts=True) #Cuento cuántas galaxias␣
↪de cada tipo tengo.

prob = frecuencia/n #Calculo la probabilidad de cada tipo de galaxia.
```

[ ]:
```
plt.bar(categ,prob,edgecolor='black') #Grafico las probabilidades empíricas.
plt.grid(True, axis='y', linestyle='--') #Agregamos grilla
```



Donde hemos reproducido la distribución dada de galaxias Enanas(d), Elípticas (e), Irregulares (ir) y Espirales (s)

Ejercicio18

August 30, 2024

Ejercicio 18:

(a) Tenemos 2 dados con 6 resultados posibles, los cuales sumaremos. Por lo tanto, nuestra variable aleatoria x = dado1 + dado2 vive en un espacio muestral [2,12] de los números naturales, es decir, es discreta.

(b) Todos los resultados en cada dado son equiprobables, luego, veamos individualmente cada valor considerando que el valor que obtenemos de cada dado individualmente es un evento independiente (es decir, un dado no condiciona al otro):

P(x=2) = P_1(1)*P_2(1) = (1/6)(1/6) = 1/36 ; Siendo P_1(1) y P_2(1) las probabilidades de los dados 1 y 2 de obtener como resultado 1.

P(x=3) = P_1(1)*P_2(2) + P_1(2)*P_2(1) = 2/36

P(x=4) = P_1(2)*P_2(2) + P_1(1)*P_2(3) + P_2(1)*P_1(3) = 3/36

P(x=5) = 4/36

P(x=6) = 5/36

P(x=7) = 6/36

P(x=8) = 5/36

P(x=9) = 4/36

P(x=10) = 3/36

P(x=11) = 2/36

P(x=12) = 1/36

Por lo tanto, la distrubución será discreta, simétrica y centrada en x=7.

(c) Ahora la idea es, como en el ejercicio 17, definir la suma acumulada de una distribución aleatoria uniforme e ir separando los valores por categorías según la distribución teórica.

[ ]:
```
#Primero, usamos el generador del ejercicio 16 e importamos numpy y matplotlib␣
 ↪para tenerlo a mano:
import numpy as np
import matplotlib.pyplot as plt
def ran(a=1664525, c=1013904223, M=2**32): #Defino el generador con los␣
 ↪parámetros solicitados
    ran.current=((a*ran.current+c)%M) #Busca el atributo seed, y lo cambia
```

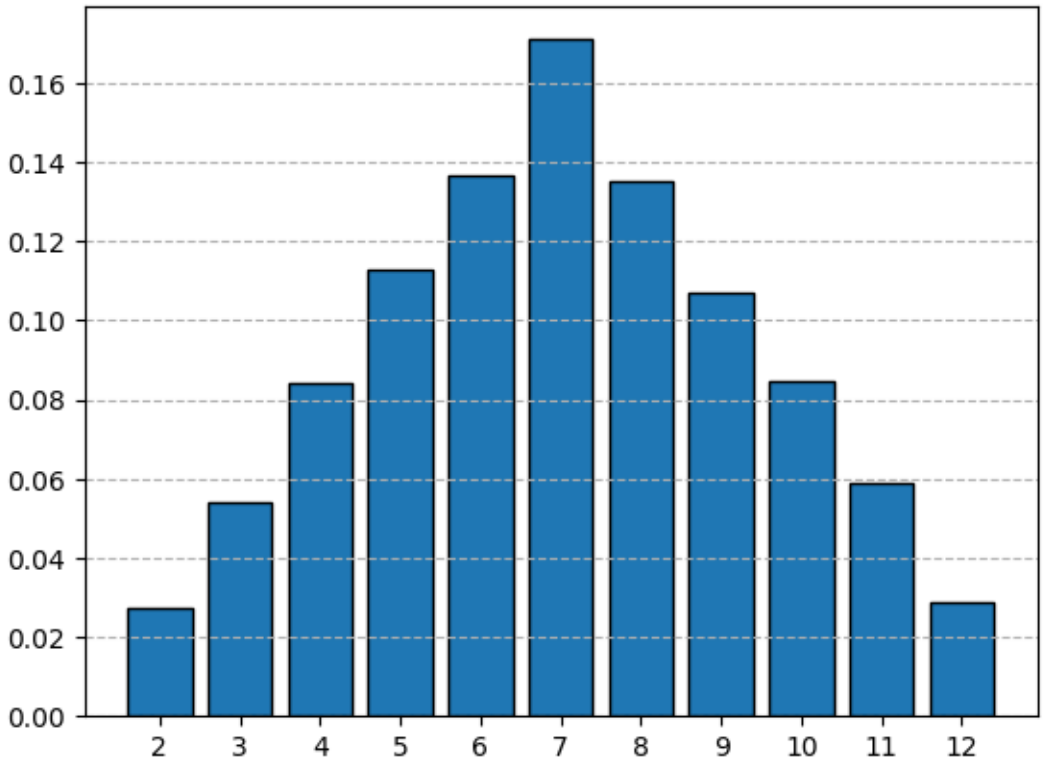
```
    return ran.current/M

ran.current = 45
```

[ ]:
```
#Planteo la muestra:
probabilidades = [1/36 , 2/36 , 3/36 , 4/36 , 5/36 , 6/36 , 5/36 , 4/36 , 3/36␣
 ↪, 2/36 , 1/36] #Defino las probabilidades
Sumas_posibles = [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12] #Defino las sumas posibles
prob_acumuladas = np.cumsum(probabilidades) #Calculo las probabilidades␣
 ↪acumuladas.
```

[ ]:
```
# Genero una lista aleatoria y las meto en cada tipo en función de si es menor␣
 ↪a la probabilidad acumulada.
n = 10000  # Cantidad de galaxias a generar
tiradas_sum = []

for i in range(n):
    r = ran() #Genero un número aleatorio
    if r < prob_acumuladas[0]:
        tiradas_sum.append(2)
    elif r < prob_acumuladas[1]:
        tiradas_sum.append(3)
    elif r < prob_acumuladas[2]:
        tiradas_sum.append(4)
    elif r < prob_acumuladas[3]:
        tiradas_sum.append(5)
    elif r < prob_acumuladas[4]:
        tiradas_sum.append(6)
    elif r < prob_acumuladas[5]:
        tiradas_sum.append(7)
    elif r < prob_acumuladas[6]:
        tiradas_sum.append(8)
    elif r < prob_acumuladas[7]:
        tiradas_sum.append(9)
    elif r < prob_acumuladas[8]:
        tiradas_sum.append(10)
    elif r < prob_acumuladas[9]:
        tiradas_sum.append(11)
    else:
        tiradas_sum.append(12)
```

[ ]:
```
categ , frecuencia = np.unique(tiradas_sum, return_counts=True) #Calculo la␣
 ↪frecuencia de cada suma

prob = frecuencia/n #Calculo la probabilidad de cada suma
```

```
[ ]: plt.bar(categ,prob,edgecolor='black') #Grafico las probabilidades empíricas.
     plt.xticks(Sumas_posibles)
     plt.grid(True, axis='y', linestyle='--') #Agregamos grilla
     plt.show()
```

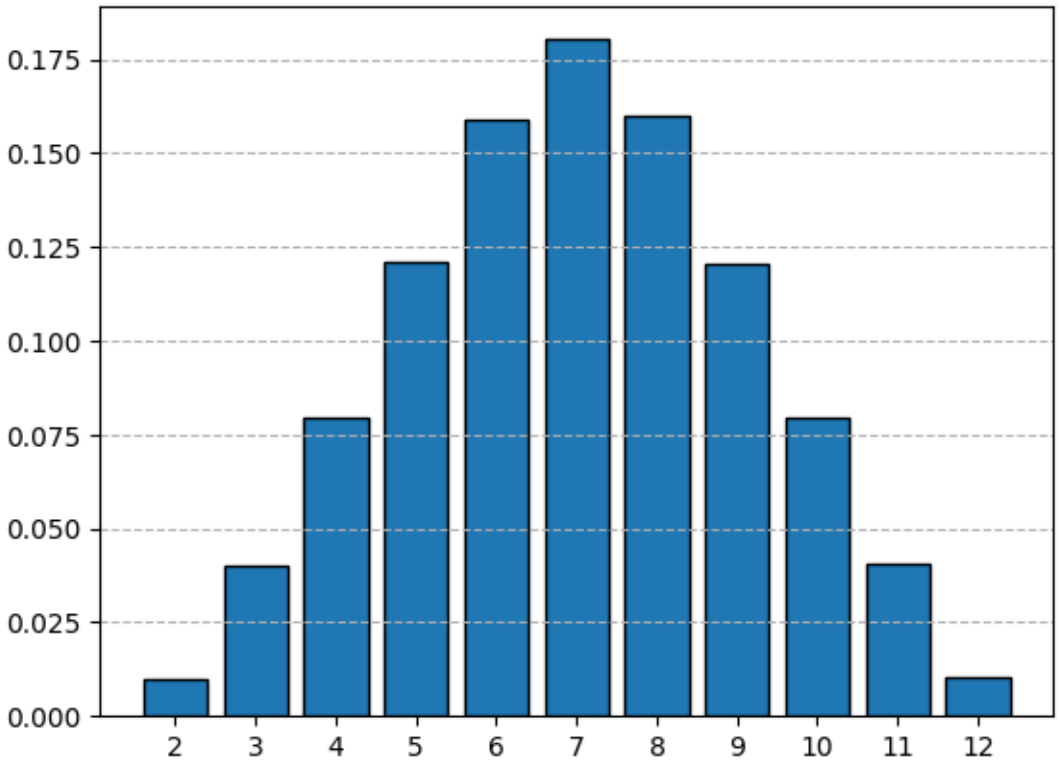


(d) Seguimos usando el generador ran() para ahora simular el experimento:

```
[ ]: #Creemos nuestros dados aleatorios:

     #Generamos una lista de números entre 0 y 1:
     n = 100000
     dado1 = []
     for i in range(n): #Bucle que genera números y los mete en la lista
         dado1.append(round(ran()*5+1)) #Los transformo al intervalo y convierto en␣
      ↪enteros, luego los meto a la lista


     dado1 = np.array(dado1) #La convertimos en array para operar más cómodos
     print(dado1[:5])

     #Repetimos para un 2° dado:
```

```
n = 100000
dado2 = []
for i in range(n): #Bucle que genera números y los mete en la lista
    dado2.append(round(ran()*5+1)) #Los transformo al intervalo y convierto en
 ↪enteros, luego los meto a la lista


dado2 = np.array(dado2) #La convertimos en array para operar más cómodos
print(dado2[:5])

experimento = dado1 + dado2

print(experimento[:5])

categ , frecuencia = np.unique(experimento, return_counts=True)
prob = frecuencia/n #Calculo la probabilidad.

#Lo graficamos

plt.xticks(np.arange(2,13)) #Los ticks del eje x, para escalearlo bien
plt.bar(categ,prob,edgecolor='black') #Grafico las probabilidades empíricas.
plt.grid(True, axis='y', linestyle='--') #Agregamos grilla

plt.show()
```

```
[5 5 5 4 4]
[2 6 3 4 4]
[ 7 11  8  8  8]
```



Podemos observar que la distribución empírica cumple con lo predicho por la teórica.

Conclusión:

En el trabajo hemos creado un generador de números pseudo-aleatorios de congruencia lineal con el cual hemos simulado experimentos y distribuciones teóricas. Observamos entonces como al aumentar la cantidad de números aleatorios las distribuciones y los experimentos se aproximan a los que nos brinda la teoría.