GG

December 13, 2021

1 Guillermo Hoyo Bravo

**

Stochastic Systems — Discrete Time Systems

###

Ejercicios — Conjunto final

###

Fecha de entrega: 13 de Diciembre de 2021

**

2 I. Gambler

En los apuntes hay el ejemplo del gambler's ruin como cadena de Markov. Hacer un gráfico del número medio de jugadas que el jugador puede hacer antes de arruinarse en función del dinero inicial. En cada jugada se juega 1 Euro, y el juego es ecuo (el jugador tiene una probabilidad 1/2 de ganar). Estimar media y varianza (usar unas 20 ejecuciones... deberían ser más que suficientes) considerando un dinero inicial de 1, . . . , 50 Euros. Indicar media y varianza. ¿Cómo varian la media y la varianza cuando aumenta el dinero inicial? Dar una estimación de la función T(e) que da el tiempo medio necesario para arruinarse en función de la cantidad inicial de dinero, e. Según esta función, ¿cuánto tarda el jugador en arruinarse si empipeza a jugar con 200 Euros?

Gamblers Chain

[2]:
```
# Imports
import numpy as np
from scipy.stats import uniform
import matplotlib.pyplot as plt
from tqdm import tqdm
```

[3]:
```
''' The class created for the generic Gamblers Chain '''
class gamblers_chain:

    '''Init Function for the gamblers_chain class, defining the chain and its␣
 ↪needed parameters'''
    def __init__(self, p, dollarzs, its):
```

```
    # Parameters
        # self.
        # p = probability of winning, going to the next state (m+1) -- (1-p) = probability of looseing, going one state back.
        # go = starting money to play until death.
        # state = total money at the moment -- current state
        # n_wins = nº of wins.
        # n_deaths = nº of looses.
        # n_rouds = number of iterations computed by the markov Chain

        self.p = p
        self.go = dollarzs
        self.state = self.go
        self.n_wins = 0
        self.n_deaths = 0
        self.n_iterations = its


    def play(self):
        # Return -- next computed state
            # State = 0 == loose, end of Game. -- ABSORPTION STATE
            # self.state.

        if(self.state == 0):
            return 0

        # Generate Uniform distribution number between (0, 1)
        u = uniform.rvs()


        # if u > p --> WIN -- state = state +1; n_wins = n_wins +1; n_rouds = n_rouds+1;
        # if u <= p --> LOOSE -- state = state -1; ....
            # The Crupier has mroe winning chances :)
        if (u > self.p):
            self.state = self.state +1
            self.n_wins = self.n_wins +1

        elif (u <= self.p):
            self.state = self.state -1
            self.n_deaths = self.n_deaths +1

        return self.state

    def mama_raised_a_soldier_not_a_bitch(self):
        # Parameters
            # p = posibility of winning -- 1 - p = loosing
```

```
            # dollarzs = initial state / initial money cuantity
            # its = number of iterations computed by the Markov Chain. -- Could␣
 ↪be done until result == 0, without a number.

        # Returns
            # i = number of iterations before money -- state gets to Absorption␣
 ↪== State 0
            # its = number of total iterations, never gets to Absorption state 0

        # Playing an it number of times, finding the time until we get poor.
        for i in range(self.n_iterations):
            moneyy = self.play()
            if (moneyy == 0):
                return i

        return its
```

[4]:
```
""" Simulation """
# Chain Initialization parameters
p = 1/2 #print("p", p)
its = 100000 #print("its", its)
dollarzs = np.arange(1,50+1) #print("dollarzs", dollarzs)

# Number of Executions
n = 20

# Saving Values Variable
results = np.zeros((50, n))

# Computing the Output
for i in tqdm(dollarzs):
    for j in range(n):
        gambol = gamblers_chain(p, i, its)
        results[i-1][j] = gambol.mama_raised_a_soldier_not_a_bitch()␣
 ↪#print("results",results[i][j])
```

```
100%|| 50/50 [07:56<00:00,  9.52s/it]
```

[ ]:
```
# Variación:
    # Media && Varainza

# Media -- Mean
media = np.mean(results, axis = 1)
print("\n media: \n", media)
# Varianza - desvest
var = np.var(results, axis = 1)
```

```
print("\n var: \n", var)
```

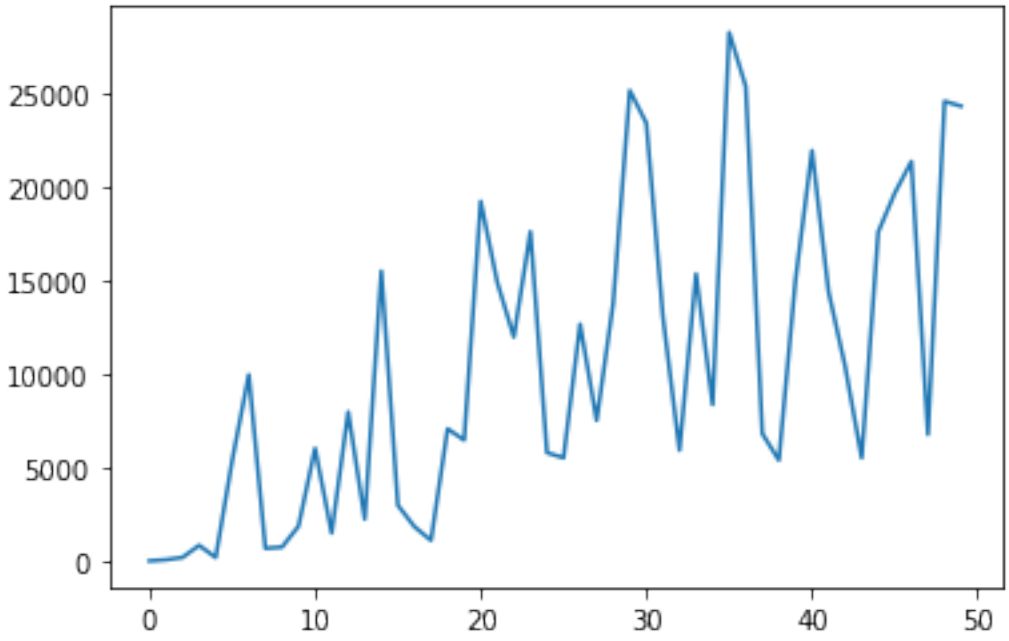
Como se puede observar en los resultados obtenidos en ambos vectores: los resultados de ambas variables aumentan según aumenta el dinero que tenemos.

Esto tiene sentido, ya que si la probabilidad de ganar o perder es de 50/50, y solo tenemos una oportunidad, es muy facil perder rapidamente. Sin emabrgo, teniendo 4 oportunidados, es muy complicado perder los 4 juegos seguidos. Por ello se puede ver que los valores son generalmente ascendente, empezando con los 10 primeros valores por debajo de 10000, y terminano con los 10 últimos por encima de este valor y con record en 35102,75. Se peuden aprecier las rachas de victorias y derrotas clarametne, como por ejemplo para la cantidad inicial de 2 la media es 2370, y sin emabrgo para 3 es 197.2.

[7]:
```
print("Media: \n\tMínimo =", min(media), "\n\tMáximo =", max(media))
plt.plot(media)
```

```
Media:
        Mínimo = 11.6
        Máximo = 28271.55
```

[7]: [<matplotlib.lines.Line2D at 0x7f1cfa8ca610>]

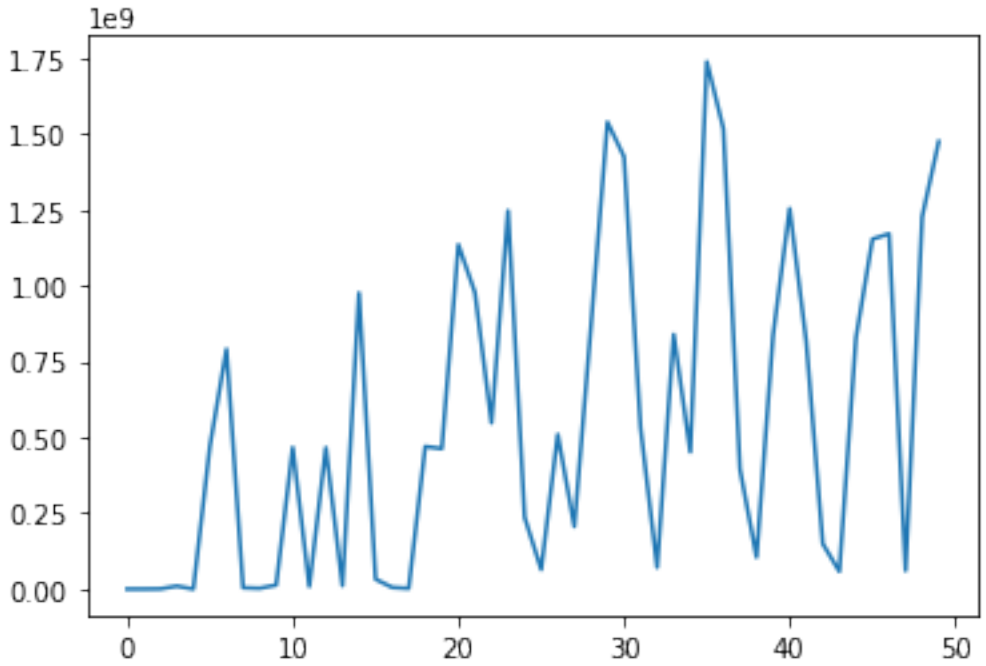


Por ello la variabilidad también es muy alta, ya que si se pueden llegar a perder 4 juegos seguiods o 20, o de infinitos, sin ser algo tan raro, para una ejecución de 1000000 de veces. De igual modo puede ocurrir una cadena de vicorias consecutiva infinita. Aunque en teoría debería de ser de 1 a 1, esto no es del todo preciso. La varianza es enorme, se ve que crece como la media, según el dinero inicial. La varianza mas pequeña encontrada es para la cantidad inicial de 4 y de magnitud 194498.96. El resultado máximo encontrado está en 1584812879.1875 empezando con el valor 42.

[8]:
```
print("Varianza: \n\tMínimo =", min(var), "\n\tMáximo =", max(var))
plt.plot(var)
```

```
Varianza:
        Mínimo = 1468.64
        Máximo = 1738277943.7475
```

[8]: [<matplotlib.lines.Line2D at 0x7f1cfa7c3730>]



[7]:
```
# Función T(e) para e = 200
    # T() == Tiempo que tarda en arruinarse el jugador segun e
    # e = Dinero inicial
```

[23]:
```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.linear_model import LinearRegression
```

[25]:
```
# Jugada con 200 euros iniciales y con 100000 iteraciones máximas totales
for j in tqdm(range(20)):
        gambol = gamblers_chain(p, 200, its)
        results[i-1][j] = gambol.mama_raised_a_soldier_not_a_bitch()␣
 ↪#print("results",results[i][j])
```

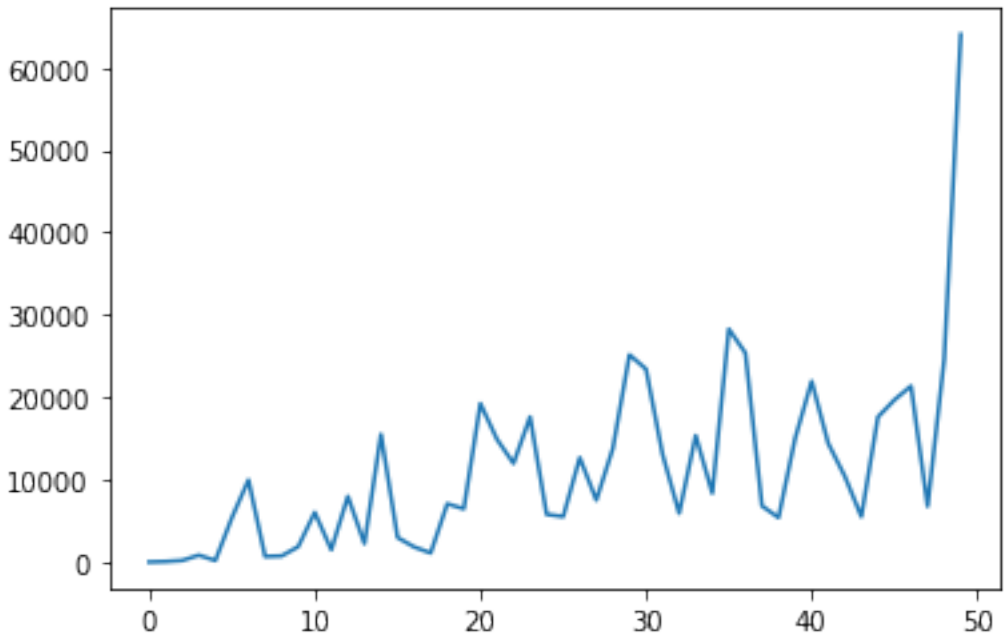
```
100%|| 20/20 [00:55<00:00,  2.78s/it]
```

[31]:
```
# Print the media results for 200 $ as starting money - state.
print(np.mean(results, axis = 1))

# Plot Media Results
plt.plot(np.mean(results, axis = 1))
```

```
[1.160000e+01 7.190000e+01 1.964000e+02 8.414000e+02 1.925000e+02
 5.401650e+03 9.968000e+03 6.834000e+02 7.447000e+02 1.867200e+03
 6.027800e+03 1.499900e+03 7.978900e+03 2.245200e+03 1.551290e+04
 2.986600e+03 1.836500e+03 1.108200e+03 7.074500e+03 6.469450e+03
 1.924010e+04 1.490870e+04 1.198380e+04 1.761875e+04 5.798000e+03
 5.521500e+03 1.267110e+04 7.522300e+03 1.380840e+04 2.516515e+04
 2.341810e+04 1.307275e+04 5.925700e+03 1.535880e+04 8.365200e+03
 2.827155e+04 2.538150e+04 6.818300e+03 5.397000e+03 1.500090e+04
 2.195220e+04 1.440915e+04 1.042060e+04 5.517100e+03 1.760450e+04
 1.968675e+04 2.136940e+04 6.768900e+03 2.458580e+04 6.404385e+04]
```

[31]:
```
[<matplotlib.lines.Line2D at 0x7f1ce8058ca0>]
```



Se puede ver como teóricamente el valor tiende a infinito clarisimamente.

[26]:
```
x_train = np.array(results)
regr = LinearRegression()
regr.fit(x_train)

pred = regr.predict(x_train)
```

```
plt.scatter(results)
plt.plot(x_train, pred, color='red')

plt.show()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/tmp/ipykernel_6344/1969733932.py in <module>
      1 x_train = np.array(results)
      2 regr = LinearRegression()
----> 3 regr.fit(x_train)
      4
      5 pred = regr.predict(x_train)

TypeError: fit() missing 1 required positional argument: 'y'
```

[ ]: