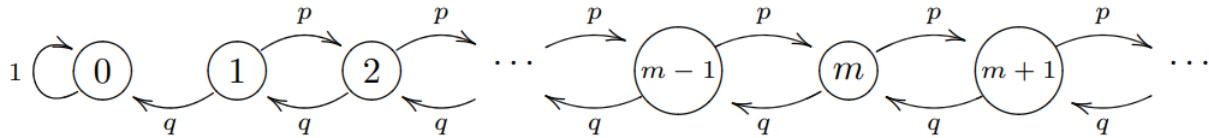


Ejercicio I.

Enunciado. En los apuntes hay el ejemplo del *gambler's ruin* como cadena de Markov:



Donde estar en el nodo i implica tener i Euros, y p y $q = 1 - p$ son las probabilidades de ganar y perder respectivamente en cada repetición del juego.

Apartado a) 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 ecua (el jugador tiene una probabilidad $p = 1/2$ de ganar).

Sabemos por el estudio teórico realizado para este problema que en el caso de $p = q = \frac{1}{2}$, el jugador convergerá a arruinarse tarde o temprano. Sin embargo, la simulación puede tomar un tiempo arbitrariamente alto de tiempo si el dinero inicial es alto. Es por ello que hemos de poner un límite al número de iteraciones máximo que ejecutaremos nuestra simulación. Consideramos que 10^5 es una cantidad aceptable de iteraciones para los bajos valores de dinero inicial que utilizaremos. Si se alcanza ese número de iteraciones podemos considerar que el jugador "se arruina en tiempo 10^5 ", que es básicamente infinito.

Implementamos la simulación de esta cadena de Markov y la ejecutamos 20 veces, computando su media. Realizar esta simulación en múltiples ocasiones y utilizar la media es imprescindible para obtener resultados representativos en procesos de Monte Carlo como este.

```
def simulate_gambler_step(p=1/2):  
    """  
        Simulates a single step of the gambler's ruin  
        - p: the probability of winning  
        - returns: 1 if won, -1 if lost  
    """  
    r = random.uniform(0, 1)  
    return 1 if r <= p else -1  
  
def simulate_gamblers_ruin(initial_money, p=1/2, max_steps=10**6):  
    """  
        Simulates a gambler's ruin markov chain.  
        - initial_money: the starting node  
        - p: the probability of winning at each step  
        - max_steps: the max number of steps to be comptued  
        - returns: time taken to ruin the gambler. If max_steps is reached  
        without ruining the gambler, return max_steps instead.  
    """  
    money = initial_money  
    steps = 0  
    while money > 0 and steps <= max_steps:  
        steps += 1  
        money += simulate_gambler_step(p)  
    return steps  
  
def plot_mean_ruin_time(max_initial_money=50, n=20, max_steps=10**5, p=1/2):  
    """
```

```

Plots the mean time for a gambler's to ruin themselves, against the initial money.
- max_initial_money: max initial money to be plotted
- n: the number of executions per initial money value
- max_steps: the max number of steps to be comptued
- p: the probability of winning at each step
"""
# Compute times
initial_money_range = np.arange(1, max_initial_money)
mean_times = [
    np.mean( [simulate_gamblers_ruin(initial_money, p=p, max_steps=max_steps)
              for _ in range(n)] )
    for initial_money in initial_money_range
]

# Plotting
plt.figure(figsize=(12,6))
plt.plot(initial_money_range, mean_times, '-o')
plt.legend(['Mean time to ruin', 'Max iterations'])
plt.xlabel('Initial money')
plt.ylabel('Mean time to ruin in {} iterations'.format(n))
plt.title('Gambler\' ruin simulation')
plt.show()

random.seed(123)
plot_mean_ruin_time(max_steps=10**5)

```

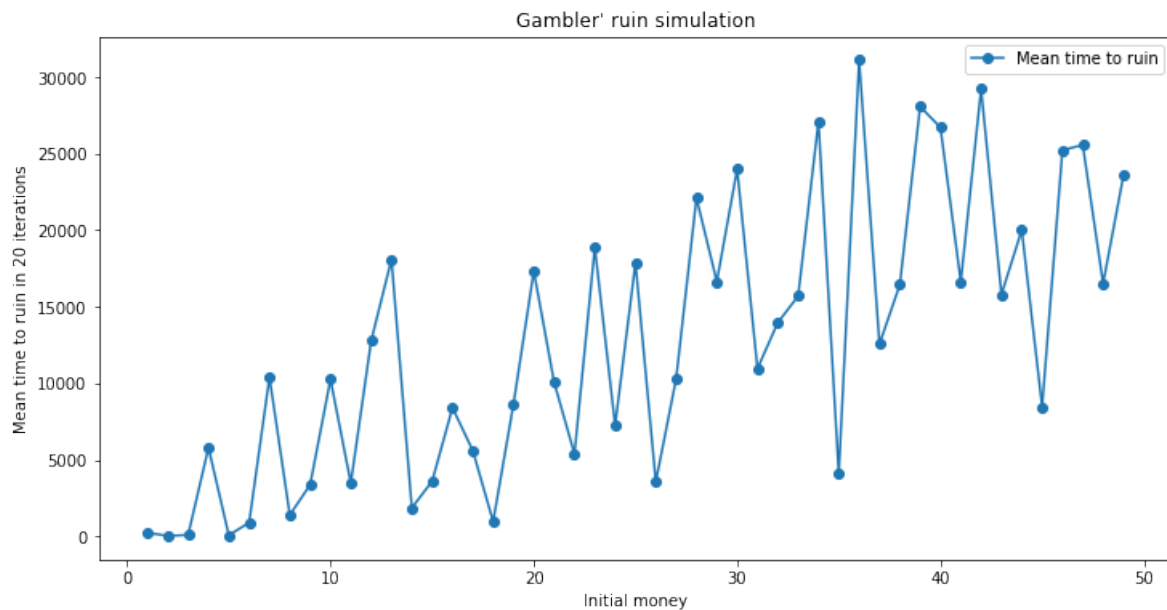


Figure 1: Mean time of 50 simulations of Gambler's Ruin process