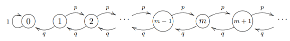
José Antonio Álvarez Ocete
77553417Q
joseantonio.alvarezo@estudiante.uam.es

Entrega final
Procesos estocásticos discretos

2021-12-12

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 en cada repetición del juego.
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 ecuo (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.
python def simulate$_gambler_step(p = 1/2) : """Simulatesasinglestepofthegambler'sruin - p : theprobabilityofwinning - returns : 1ifwon, -1iflost""" r = random.uniform(0, 1)return1ifr <= pelse - 1$
def simulate$_gamblers_ruin(initial_money, p = 1/2, max_steps = 10 * *6) : """Simulatesagambler'sruinmarkovchain. - initial_money : thestartingnode-p : theprobabilityofwinningateachstep-max_steps : themaxnumberofstepstobecomptued- returns : timetakentoruinthegambler.Ifmax_stepsisreachedwithoutruiningthegambler, returnmax_stepsinstead.""" money = initial_moneysteps = 0whilemoney > 0andsteps <= max_steps : steps+ = 1money+ = simulate_gambler_step(p)returnsteps$
def plot$_mean_ruin_time(max_initial_money = 50, n = 20, max_steps = 10**5, p = 1/2) : """Plotsthemeantimeforagambler'storuint max_initial_money : maxinitialmoneytobeplotted-n : thenumberofexecutionsperinitialmoneyvalue-max_steps : themaxnumbero p : theprobabilityofwinningateachstep""" Computetimesinitial_money_range = np.arange(1, max_initial_money)mean_times = [np.mean([simulate_gamblers_ruin(initial_money, p = p, max_steps = max_steps)for_inrange(n)])forinitial_moneyininitial_money_ar$
Plotting plt.figure(figsize=(12,6)) plt.plot(initial$_money_range, mean_times,' -o')plt.legend(['Meantimetoruin',' Maxiterations'])pl$
random.seed(123) plot$_mean_ruin_time(max_steps = 10 * *5)$

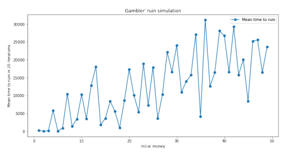


Figure 1: Single simulation of Gambler's Ruin problem