Київський національний університет імені Тараса Шевченка

Факультет комп’ютерних наук та кібернетики

Кафедра теорії та технології програмування

Самостійна робота 3

зі «Статистичне моделювання в задачах штучного інтелекту»

Груповий звіт

студентів 4 курсу групи ТТП-42

Чебана Богдана Володимировича

Ходакова Максима Олеговича

Бариша Артема Богдановича  
Сьомкіна Євгенія Сергійовича

Київ – 2024

**Зміст**

1. Вступ
2. Теоретичні Основи
   * Алгоритм Метрополіса-Гастінгса
   * Cauchy-Розподіл
3. Реалізація Завдання
   * Опис Програми
   * Код Програми
4. Результати та Аналіз
   * Trace Plot
   * Гістограма з Теоретичною Щільністю
   * KDE Plot
5. Висновок
6. Список Використаної Літератури

**Вступ**

У сучасних задачах штучного інтелекту та машинного навчання часто виникає необхідність моделювання складних ймовірнісних розподілів. Такі задачі можуть включати генерацію вибірок для розподілів, які складно описати аналітично або з яких важко генерувати дані напряму. Одним із ефективних методів для цього є алгоритм Метрополіса-Гастінгса. Цей алгоритм є методом Марковських ланцюгів Монте-Карло (MCMC), який дозволяє генерувати вибірки з будь-якого заданого розподілу, навіть якщо його щільність складна для безпосереднього зразковування. Він використовує ітераційний підхід, де нові значення приймаються або відхиляються на основі певних умов, що забезпечує поступове наближення вибірок до цільового розподілу.

Алгоритм Метрополіса-Гастінгса є важливим інструментом для вирішення широкого кола задач у сфері штучного інтелекту, таких як баєсівське моделювання, оцінювання параметрів складних моделей, а також для аналізу статистичних даних, де пряме зразковування неможливе або неефективне. Цей метод широко використовується для моделювання складних процесів, аналізу даних, а також в задачах оптимізації та прийняття рішень.

Дана робота охоплює кілька важливих задач статистичного моделювання:

1. **Моделювання ланцюгів Маркова**: Метою цього завдання є моделювання як поглинаючих, так і регулярних ланцюгів Маркова. Це дозволяє аналізувати поведінку системи в довгостроковій перспективі, визначати ймовірності переходів між станами, час перебування в заданих станах, ймовірності поглинання тощо. Таке моделювання корисне для розуміння динаміки систем, що змінюються з часом, і використовується в різних галузях, включаючи фінанси, біологію та інформатику.
2. **Алгоритм Метрополіса-Гастінгса**: Цей алгоритм використовується для моделювання різних розподілів, таких як бета-розподіл, гамма-розподіл і розподіл Коші. Метою є генерування вибірок для таких розподілів та візуалізація результатів у вигляді графіків реалізацій, гістограм і щільностей. Цей підхід дозволяє глибше зрозуміти поведінку і властивості складних ймовірнісних розподілів.
3. **Ігри з монетами**: Завдання включає моделювання гри з підкиданням монети, де гравець отримує винагороду в залежності від комбінацій, що випадають. Це дозволяє аналізувати середній виграш, оцінювати ризики та вигоди для різних правил гри. Таке моделювання є прикладом використання ймовірнісних методів для аналізу ігор та прийняття рішень в умовах невизначеності.
4. **Задача про 100 ув'язнених**: Це завдання моделює випробування для групи ув'язнених, яким необхідно знайти свої номери в коробках. Воно показує, як правильний вибір стратегії може значно вплинути на ймовірність успіху в складних умовах. Завдання ілюструє важливість стратегічного мислення і взаємодії між учасниками системи, що може бути застосовано в різних галузях, таких як теорія ігор, логістика та оптимізація процесів.
5. **Закон Бенфорда**: Закон Бенфорда описує розподіл першої цифри в числах і застосовується для аналізу великих наборів даних, таких як площі країн або випадкові числа. Це завдання включає перевірку закону на реальних даних і випадково згенерованих числах, що дає змогу краще зрозуміти, як цей закон працює і чому його можна використовувати для виявлення аномалій або фальсифікацій у даних.

**Теоретичні відомості**

**2.1. Моделювання ланцюгів Маркова**

**Ланцюг Маркова** — це стохастичний процес, що описується послідовністю станів, де ймовірність переходу до наступного стану залежить лише від поточного стану (властивість Маркова). Ланцюги Маркова широко використовуються для моделювання систем з невизначеністю, таких як біологічні процеси, економічні моделі та інші.

**Поглинаючий ланцюг Маркова** містить принаймні один поглинаючий стан — стан, з якого неможливо вийти (тобто, P(i→i)=1). Інші стани можуть бути тимчасовими або також поглинаючими.

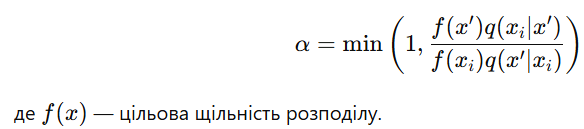
**Регулярний ланцюг Маркова** — це ланцюг, де для деякого числа ітерацій існує ймовірність переходу між будь-якими парами станів, забезпечуючи існування стаціонарного розподілу.

**2.2. Алгоритм Метрополіса-Гастінгса**

**Алгоритм Метрополіса-Гастінгса** є методом Марковських ланцюгів Монте-Карло (MCMC), який дозволяє генерувати випадкові зразки з будь-якого заданого розподілу ймовірностей, навіть якщо його щільність складна для безпосереднього зразковування. Основна ідея полягає в побудові ланцюга Маркова, стан якого поступово сходиться до цільового розподілу.

**Основні кроки алгоритму:**

1. **Ініціалізація:** Вибирається початкове значення x\_0​.
2. **Пропозиція:** Генерується нова пропозиція x′ з пропозиційного розподілу q(x′∣x\_i​).
3. **Обчислення прийнятності:** Розраховується відношення прийнятності:



 **Прийняття або відхилення:** Згенерована випадкова величина u∼U(0,1). Якщо u<α, пропозиція x′ приймається і стає наступним станом ланцюга; інакше, зберігається попередній стан.

 **Повторення:** Кроки 2-4 повторюються до досягнення необхідної кількості ітерацій.

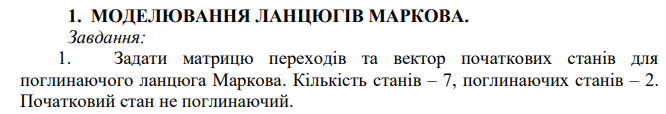
2.3. **Ігри з монетами**  
Ігри з монетами є класичними прикладами стохастичних процесів, де результат кожного підкидання монети є випадковим і незалежним від попередніх підкидань. Такі ігри допомагають ілюструвати поняття ймовірності, математичного очікування та ризику. У завданні ми аналізуємо дві різні гри: одну, де виграш залежить від комбінації «Орел-Решка», та іншу, де виграш залежить від комбінації «Орел-Орел». Це дозволяє оцінити середні виграші та ймовірність виграшу для кожного правила, надаючи важливі інсайти в управління ризиками та прийняття рішень в умовах випадковості.

2.4. **Задача про 100 ув'язнених**  
Задача про 100 ув'язнених — це класична задача комбінаторики та ймовірності, що демонструє концепції циклів у перестановках та їх вплив на успіх у певних умовах. Група ув'язнених має виконати серію дій за спеціальною стратегією, щоб максимізувати свої шанси на успіх. Ця задача підкреслює важливість колективної стратегії та оптимізації під час роботи зі складними системами. Моделювання цієї задачі допомагає зрозуміти, як правильні алгоритми та планування можуть суттєво змінити результати навіть у ситуаціях із високими ставками.

2.5. **Закон Бенфорда**  
Закон Бенфорда, або закон першої цифри, описує нерівномірний розподіл першої цифри чисел у багатьох реальних наборах даних. Найчастіше першою цифрою є одиниця, а ймовірність її появи зменшується для більших цифр. Цей закон використовується в різних практичних сферах, зокрема у фінансових розслідуваннях для виявлення шахрайства та перевірці відповідності даних природним закономірностям. У завданні ми досліджуємо, як цей закон працює на реальних даних (наприклад, площі країн) та випадково згенерованих наборах чисел, що дає змогу краще зрозуміти, коли і чому виникає ця закономірність.

**Завдання 1**

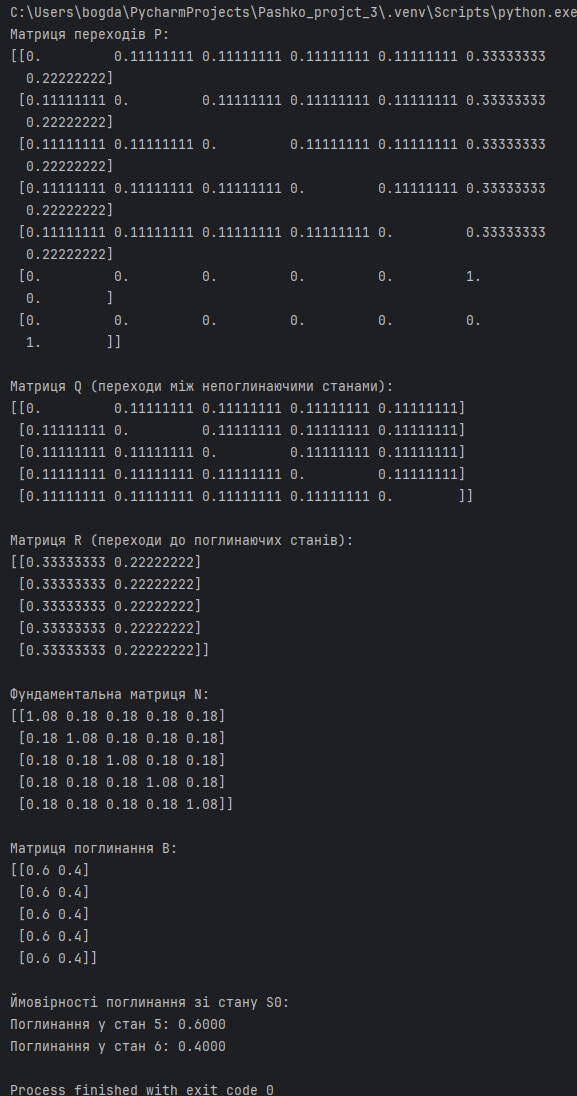
**1.1**



import numpy as np  
import matplotlib.pyplot as plt  
import networkx as nx  
  
# Визначення кількості станів  
num\_states = 7  
  
# Індексування станів від 0 до 6  
states = list(range(num\_states))  
  
# Визначення поглинаючих станів (наприклад, стан 5 і 6)  
absorbing\_states = [5, 6]  
  
# Створення матриці переходів P з нулями  
P = np.zeros((num\_states, num\_states))  
  
# Заповнення матриці переходів  
for i in range(num\_states):  
 if i in absorbing\_states:  
 P[i][i] = 1.0 # Поглинаючий стан  
 else:  
 # Визначаємо кількість можливих переходів (усі стани крім поточного)  
 # Тут задаємо конкретні ймовірності переходів  
 P[i] = np.array([0.1 if j != i else 0 for j in range(num\_states)])  
 # Збільшуємо ймовірність переходу до поглинаючих станів  
 P[i][5] = 0.3  
 P[i][6] = 0.2  
 # Нормалізуємо рядок, щоб сума дорівнювала 1  
 P[i] = P[i] / P[i].sum()  
  
print("Матриця переходів P:")  
print(P)  
  
# Визначення непоглинаючих станів  
transient\_states = [s for s in states if s not in absorbing\_states]  
  
# Визначення розмірів  
t = len(transient\_states)  
r = len(absorbing\_states)  
  
# Створення матриці Q  
Q = P[np.ix\_(transient\_states, transient\_states)]  
  
# Створення матриці R  
R = P[np.ix\_(transient\_states, absorbing\_states)]  
  
print("\nМатриця Q (переходи між непоглинаючими станами):")  
print(Q)  
  
print("\nМатриця R (переходи до поглинаючих станів):")  
print(R)  
  
# Одинична матриця розміру Q  
I = np.eye(t)  
  
# Обчислення фундаментальної матриці N  
N = np.linalg.inv(I - Q)  
  
print("\nФундаментальна матриця N:")  
print(N)  
  
# Обчислення матриці поглинання B  
B = np.dot(N, R)  
  
print("\nМатриця поглинання B:")  
print(B)  
  
# Створення графа  
G = nx.DiGraph()  
  
# Додавання вузлів  
for state in states:  
 if state in absorbing\_states:  
 G.add\_node(state, color='red', label=f"S{state} (Поглинаючий)")  
 else:  
 G.add\_node(state, color='lightblue', label=f"S{state}")  
  
# Додавання ребер з ймовірностями  
for i in states:  
 for j in states:  
 if P[i][j] > 0:  
 G.add\_edge(i, j, weight=P[i][j])  
  
# Отримання кольорів вузлів  
colors = [G.nodes[node]['color'] for node in G.nodes]  
  
# Отримання міток вузлів  
labels = {node: G.nodes[node]['label'] for node in G.nodes}  
  
# Отримання міток ребер з ймовірностями  
edge\_labels = {(i, j): f"{P[i][j]:.2f}" for i, j in G.edges()}  
  
# Відображення графа  
plt.figure(figsize=(12, 8))  
pos = nx.spring\_layout(G, seed=42) # Розміщення вузлів  
  
# Малювання вузлів  
nx.draw\_networkx\_nodes(G, pos, node\_color=colors, node\_size=800)  
  
# Малювання ребер  
nx.draw\_networkx\_edges(G, pos, arrowstyle='->', arrowsize=20, edge\_color='gray')  
  
# Малювання міток вузлів  
nx.draw\_networkx\_labels(G, pos, labels, font\_size=12)  
  
# Малювання міток ребер  
nx.draw\_networkx\_edge\_labels(G, pos, edge\_labels=edge\_labels, font\_size=10)  
  
plt.title("Поглинаючий ланцюг Маркова з 7 станами")  
plt.axis('off')  
plt.show()  
  
# Початковий стан S0  
initial\_state\_index = transient\_states.index(0) # Індекс S0 у transient\_states  
  
# Ймовірності поглинання з S0  
absorption\_probabilities = B[initial\_state\_index]  
  
print("\nЙмовірності поглинання зі стану S0:")  
for idx, prob in enumerate(absorption\_probabilities):  
 print(f"Поглинання у стан {absorbing\_states[idx]}: {prob:.4f}")

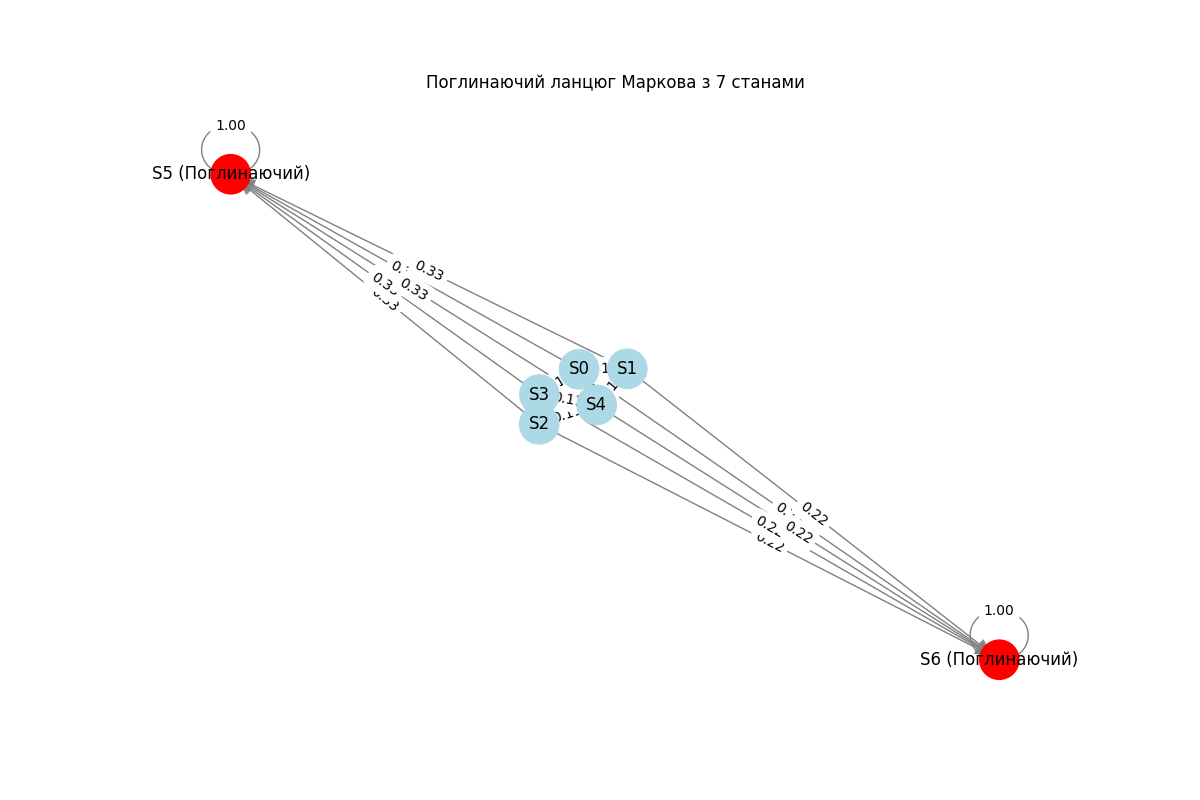
**Опис коду**

1. **Імпорт бібліотек**: Код починається з імпорту необхідних бібліотек, таких як numpy для роботи з масивами, matplotlib для візуалізації, та networkx для створення та відображення графів.
2. **Визначення параметрів**: Встановлюється кількість станів (num\_states = 7), а також визначаються поглинаючі стани (у даному випадку, стани 5 і 6). Станам присвоюються індекси від 0 до 6.
3. **Матриця переходів**: Створюється матриця переходів P, яка заповнюється ймовірностями переходу між станами. Поглинаючі стани мають ймовірність 1 залишатися у своєму стані, а для непоглинаючих станів ймовірності переходу задаються вручну. Зокрема, ймовірності переходу до поглинаючих станів спеціально збільшуються.
4. **Вивід матриці переходів**: Виводиться матриця переходів на екран, що дозволяє оцінити, як ймовірності були визначені.
5. **Матриці Q та R**: Визначаються матриці Q (переходи між непоглинаючими станами) та R (переходи до поглинаючих станів) на основі матриці P.
6. **Фундаментальна матриця**: Обчислюється фундаментальна матриця N за формулою N=(I−Q)−1N = (I - Q)^{-1}N=(I−Q)−1, що дозволяє оцінити тривалість перебування в непоглинаючих станах.
7. **Матриця поглинання**: Знаходиться матриця поглинання B, яка показує ймовірності переходу до поглинаючих станів з непоглинаючих.
8. **Створення графа**: За допомогою networkx створюється орієнтований граф, де вузли представляють стани, а ребра — ймовірності переходу між ними. Вузли кольорується відповідно до того, чи є вони поглинаючими.
9. **Візуалізація графа**: Граф малюється з використанням matplotlib, показуючи структуру ланцюга Маркова та ймовірності переходів.
10. **Обчислення ймовірностей поглинання**: Обчислюються ймовірності поглинання з початкового стану S0S\_0S0​ до поглинаючих станів, а результати виводяться на екран.



Скріншот показує виконання Python-програми для моделювання поглинаючого ланцюга Маркова. Ось детальний опис виведених даних:

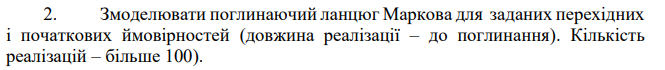
1. **Матриця переходів P**:
   * Це матриця розміром 7×77 \times 77×7, яка визначає ймовірності переходу між станами.
   * Деякі рядки містять повторювані значення 0.11111111, що відповідає ймовірностям переходу між непоглинаючими станами, а також значення 0.33333333 та 0.22222222 для переходів до поглинаючих станів.
2. **Матриця Q**:
   * Це частина матриці P, що представляє ймовірності переходу між непоглинаючими станами.
   * Вона також складається з повторюваних значень, які показують, що ймовірності переходів однакові для всіх можливих переходів між цими станами.
3. **Матриця R**:
   * Це частина матриці P, що описує ймовірності переходу з непоглинаючих станів до поглинаючих.
   * Вона має значення 0.33333333 і 0.22222222, що вказують на ймовірності поглинання.
4. **Фундаментальна матриця N**:
   * Це обчислена матриця, що показує середню кількість разів, які система проводить у кожному непоглинаючому стані до поглинання.
   * Всі значення у матриці приблизно рівні 0.18.
5. **Матриця поглинання B**:
   * Ця матриця показує ймовірності поглинання з кожного непоглинаючого стану до кожного з поглинаючих станів.
   * У матриці значення 0.6 і 0.4 вказують на ймовірності поглинання у стани 5 та 6 відповідно.
6. **Ймовірності поглинання зі стану S0**:
   * Програма обчислює ймовірності поглинання зі стану S0S0S0, які дорівнюють 0.6 (для стану 5) та 0.4 (для стану 6).
   * Це значення виведено у форматі, який демонструє розподіл ймовірностей між поглинаючими станами.



Скріншот демонструє графічне зображення поглинаючого ланцюга Маркова з 7 станами:

1. **Вузли графа**:
   * Вузли, позначені як S0,S1,S2,S3,S4S0, S1, S2, S3, S4S0,S1,S2,S3,S4, мають блакитний колір і представляють непоглинаючі стани.
   * Вузли S5S5S5 і S6S6S6 позначені червоним кольором і є поглинаючими станами.
   * Кожен вузол має підпис, що вказує на його статус (поглинаючий чи непоглинаючий).
2. **Ребра графа**:
   * Ребра між вузлами показують можливі переходи між станами ланцюга.
   * Над кожним ребром вказана ймовірність переходу, наприклад, 0.330.330.33 або 0.220.220.22.
   * Напрямки ребер вказують, куди може перейти система з кожного стану, причому ребра ведуть як до інших непоглинаючих станів, так і до поглинаючих станів.
3. **Візуалізація**:
   * Граф організований так, що непоглинаючі стани згруповані в центрі, тоді як поглинаючі стани розташовані по боках.
   * Ймовірності переходів ретельно зазначені, що дає чітке уявлення про те, як система може рухатися між станами.
4. **Деталі**:
   * Поглинаючі стани мають петлі з ймовірністю 1.001.001.00, що вказує на те, що система залишиться в цих станах після досягнення.
   * Візуалізація створена для того, щоб інтуїтивно зрозуміти структуру та поведінку поглинаючого ланцюга Маркова.

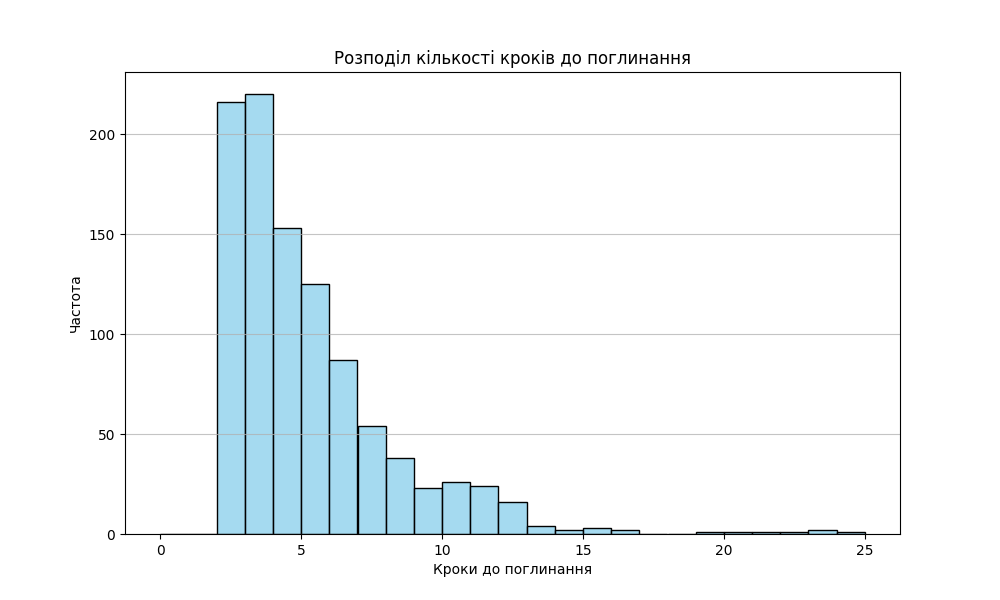
**2.2:**



import numpy as np  
import matplotlib.pyplot as plt  
import networkx as nx  
from collections import defaultdict  
import seaborn as sns  
  
# Визначення кількості станів  
num\_states = 5  
  
# Індексування станів від 0 до 4  
states = list(range(num\_states))  
  
# Визначення поглинаючих станів (стани 3 та 4)  
absorbing\_states = [3, 4]  
  
# Створення матриці переходів P з нулями  
P = np.zeros((num\_states, num\_states))  
  
# Заповнення матриці переходів  
# Задаємо конкретні ймовірності переходів  
  
# Стан 0  
P[0] = [0.1, 0.6, 0.3, 0.0, 0.0]  
  
# Стан 1  
P[1] = [0.2, 0.2, 0.5, 0.1, 0.0]  
  
# Стан 2  
P[2] = [0.0, 0.3, 0.2, 0.4, 0.1]  
  
# Поглинаючі стани залишаються самі у собі  
for absorb\_state in absorbing\_states:  
 P[absorb\_state][absorb\_state] = 1.0  
  
# Вектор початкових ймовірностей (початковий стан – стан 0)  
initial\_state = 0  
  
print("Матриця переходів P:")  
for i, row in enumerate(P):  
 print(f"Стан {i}: {row}")  
  
  
def simulate\_markov\_chain(P, initial\_state, absorbing\_states):  
 *"""  
 Симулює одну реалізацію поглинаючого ланцюга Маркова до моменту поглинання.  
  
 Parameters:  
 P (numpy.ndarray): Матриця переходів.  
 initial\_state (int): Початковий стан.  
 absorbing\_states (list): Список поглинаючих станів.  
  
 Returns:  
 steps (int): Кількість кроків до поглинання.  
 final\_state (int): Поглинаючий стан, у який потрапив ланцюг.  
 path (list): Послідовність станів у цій реалізації.  
 """* current\_state = initial\_state  
 steps = 0  
 path = [current\_state]  
  
 while current\_state not in absorbing\_states:  
 # Вибір наступного стану на основі ймовірностей переходів  
 next\_state = np.random.choice(states, p=P[current\_state])  
 path.append(next\_state)  
 steps += 1  
 current\_state = next\_state  
  
 return steps, current\_state, path  
  
  
# Кількість симуляцій  
num\_simulations = 1000  
  
# Збір статистичних даних  
steps\_to\_absorption = []  
absorption\_counts = defaultdict(int)  
paths = []  
  
for \_ in range(num\_simulations):  
 steps, final\_state, path = simulate\_markov\_chain(P, initial\_state, absorbing\_states)  
 steps\_to\_absorption.append(steps)  
 absorption\_counts[final\_state] += 1  
 paths.append(path)  
  
# Обчислення ймовірності поглинання в кожен поглинаючий стан  
absorption\_probabilities = {state: count / num\_simulations for state, count in absorption\_counts.items()}  
  
print(f"\nПроведено {num\_simulations} симуляцій.")  
print("Ймовірності поглинання:")  
for state in absorbing\_states:  
 print(f"Поглинання у стан {state}: {absorption\_probabilities.get(state, 0):.4f}")  
  
# Основні статистичні показники для кроків до поглинання  
mean\_steps = np.mean(steps\_to\_absorption)  
median\_steps = np.median(steps\_to\_absorption)  
std\_steps = np.std(steps\_to\_absorption)  
  
print(f"\nСередня кількість кроків до поглинання: {mean\_steps:.2f}")  
print(f"Медіана кількості кроків до поглинання: {median\_steps}")  
print(f"Стандартне відхилення кількості кроків до поглинання: {std\_steps:.2f}")  
  
# Візуалізація Результатів  
  
# a. Гістограма Кроків до Поглинання  
plt.figure(figsize=(10, 6))  
sns.histplot(steps\_to\_absorption, bins=range(0, max(steps\_to\_absorption) + 2), kde=False, color='skyblue')  
plt.title('Розподіл кількості кроків до поглинання')  
plt.xlabel('Кроки до поглинання')  
plt.ylabel('Частота')  
plt.grid(axis='y', alpha=0.75)  
plt.show()  
  
# b. Розподіл Поглинальних Станів  
labels = [f"Стан {state}" for state in absorbing\_states]  
sizes = [absorption\_probabilities[state] \* 100 for state in absorbing\_states]  
colors\_pie = ['lightcoral', 'lightskyblue']  
  
plt.figure(figsize=(8, 8))  
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=140, colors=colors\_pie, shadow=True)  
plt.title('Розподіл поглинальних станів після симуляцій')  
plt.axis('equal') # Рівні пропорції  
plt.show()  
  
# c. Візуалізація Ланцюга Маркова  
G = nx.DiGraph()  
  
# Додавання вузлів  
for state in states:  
 if state in absorbing\_states:  
 G.add\_node(state, color='red', label=f"S{state} (Поглинаючий)")  
 else:  
 G.add\_node(state, color='lightblue', label=f"S{state}")  
  
# Додавання ребер з ймовірностями  
for i in states:  
 for j in states:  
 if P[i][j] > 0:  
 G.add\_edge(i, j, weight=P[i][j])  
  
# Отримання кольорів вузлів  
node\_colors = [G.nodes[node]['color'] for node in G.nodes]  
  
# Отримання міток вузлів  
labels = {node: G.nodes[node]['label'] for node in G.nodes}  
  
# Отримання міток ребер з ймовірностями  
edge\_labels = {(i, j): f"{P[i][j]:.2f}" for i, j in G.edges()}  
  
# Відображення графа  
plt.figure(figsize=(12, 8))  
pos = nx.spring\_layout(G, seed=42) # Розміщення вузлів  
  
# Малювання вузлів  
nx.draw\_networkx\_nodes(G, pos, node\_color=node\_colors, node\_size=800)  
  
# Малювання ребер  
nx.draw\_networkx\_edges(G, pos, arrowstyle='->', arrowsize=20, edge\_color='gray')  
  
# Малювання міток вузлів  
nx.draw\_networkx\_labels(G, pos, labels, font\_size=12)  
  
# Малювання міток ребер  
nx.draw\_networkx\_edge\_labels(G, pos, edge\_labels=edge\_labels, font\_size=10)  
  
plt.title("Поглинаючий ланцюг Маркова з 5 станами")  
plt.axis('off')  
plt.show()

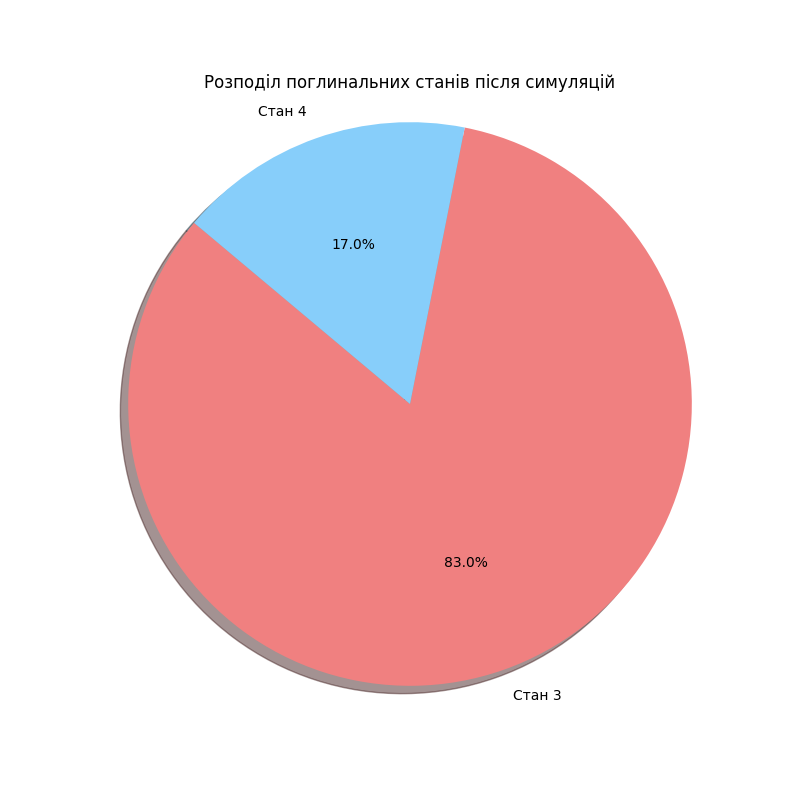
Опис коду:

1. **Імпорт бібліотек**:
   * numpy, matplotlib.pyplot, networkx, collections.defaultdict, і seaborn використовуються для створення та візуалізації поглинаючого ланцюга Маркова.
2. **Визначення основних параметрів**:
   * Встановлено кількість станів num\_states = 5 і створено список станів від 0 до 4.
   * Визначено поглинаючі стани: absorbing\_states = [3, 4].
3. **Створення матриці переходів**:
   * Ініціалізується матриця переходів P розміром 5x5 з нулями.
   * Задано конкретні ймовірності переходів для кожного стану, крім поглинаючих станів.
   * Для поглинаючих станів встановлено ймовірності залишатися в собі (ймовірність 1).
4. **Симуляція ланцюга Маркова**:
   * Функція simulate\_markov\_chain() виконує одну реалізацію ланцюга Маркова до моменту досягнення поглинаючого стану, повертаючи кількість кроків, кінцевий поглинаючий стан і шлях реалізації.
   * Виконується 1000 симуляцій, збираються дані про кроки до поглинання і підраховуються ймовірності поглинання в кожен поглинаючий стан.
5. **Обчислення статистичних показників**:
   * Середнє, медіана та стандартне відхилення кількості кроків до поглинання обчислюються та виводяться.
6. **Візуалізація результатів**:
   * **Гістограма**: Розподіл кількості кроків до поглинання.
   * **Кругова діаграма**: Ймовірності поглинання в стани 3 та 4.
   * **Граф ланцюга Маркова**: Візуалізація переходів між станами з ймовірностями.
7. **Створення графа ланцюга Маркова**:
   * Граф створено з допомогою networkx, де вузли представляють стани, а ребра — ймовірності переходів.
   * Вузли та ребра візуалізовані з кольорами і підписами, що допомагають зрозуміти структуру ланцюга.



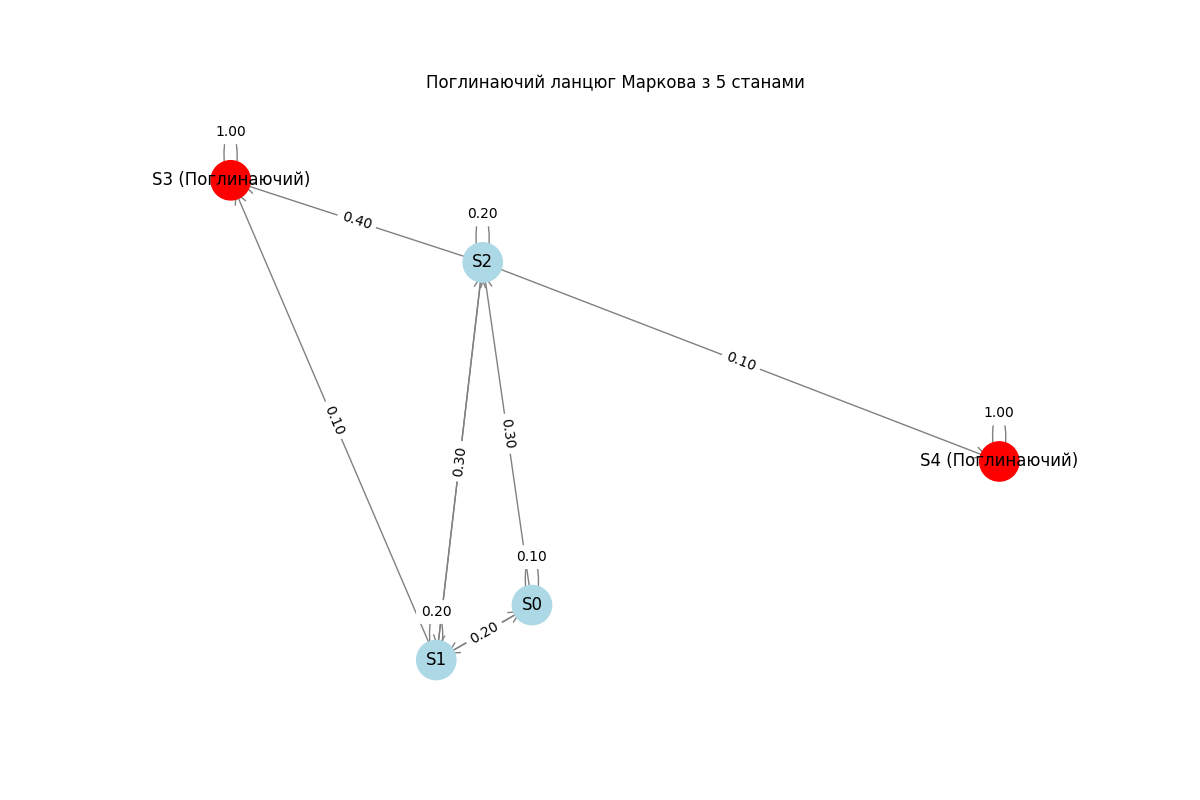
На скріншоті зображено гістограму, що показує розподіл кількості кроків до поглинання у поглинаючому ланцюзі Маркова, з такими характеристиками:

1. **Назва графіка**: "Розподіл кількості кроків до поглинання" — це назва, що чітко відображає зміст графіка. Вона пояснює, що ми аналізуємо, а саме — кількість кроків, необхідну для досягнення одного з поглинаючих станів у ланцюзі Маркова.
2. **Ось X (Кроки до поглинання)**: По горизонтальній осі відкладено кількість кроків, яка потрібна для того, щоб потрапити у поглинаючий стан. Кроки варіюються від 0 до більш як 25, що означає, що деякі симуляції швидко досягали поглинаючих станів, а деякі потребували більше часу.
3. **Ось Y (Частота)**: По вертикальній осі відкладена частота кожної кількості кроків, тобто скільки разів за певну кількість симуляцій певна кількість кроків була необхідною для досягнення поглинаючого стану. Цей графік показує, скільки разів у симуляціях ланцюг Маркова досягав поглинаючого стану за відповідну кількість кроків.
4. **Основний характер розподілу**:
   * **Схильність до малих значень кроків**: Гістограма має високий пік на початку, що свідчить про те, що більшість симуляцій досягали поглинаючого стану за невелику кількість кроків (приблизно від 1 до 5). Це означає, що ланцюг має високу ймовірність швидкого переходу в один із поглинаючих станів.
   * **Експоненційне спадання**: Після піку графік має експоненціальне спадання. Це вказує на те, що менше випадків потребувало великої кількості кроків для досягнення поглинаючого стану, а кількість таких випадків поступово зменшується, що є типовим для багатьох стохастичних процесів із високою ймовірністю швидкого переходу.
   * **Довгий хвіст**: Деякі випадки потребували більше ніж 15 кроків для поглинання. Хоча такі ситуації є менш імовірними, вони все ж трапляються, що є ознакою "довгого хвоста" в розподілі.
5. **Аналіз частоти**:
   * **Максимальний пік на початку**: Найвищі частоти кроків до поглинання розташовані в діапазоні від 1 до 5. Це означає, що велика частина реалізацій ланцюга Маркова швидко завершувалася у поглинаючих станах.
   * **Маленькі значення частоти для великих кроків**: Існують окремі випадки, коли ланцюг потребував значно більше кроків, але це трапляється значно рідше.
6. **Інтерпретація для практичного застосування**:
   * Такий характер розподілу вказує на те, що для більшості реалізацій можна передбачати швидкий перехід до поглинаючого стану, що є важливим для оцінки середнього часу перебування в системі.
   * Висока частота для малих значень і низька для великих вказує на "збалансований" характер розподілу, де більшість випадків швидко досягають завершення.
7. **Графічні деталі**:
   * **Колір гістограми**: Гістограма зображена світло-блакитним кольором, що допомагає краще візуально виділити частоти, а також спрощує аналіз загальної картини.
   * **Границя осей**: Обидві осі мають чітке позначення, що дозволяє легко інтерпретувати дані і зрозуміти кількість симуляцій та відповідну кількість кроків до поглинання.
8. **Висновок**:
   * З аналізу цього графіка видно, що поглинаючий ланцюг Маркова здебільшого швидко досягає поглинаючих станів, що може бути корисно для аналізу стабільності або тривалості процесів у системах, які описуються такими ланцюгами.



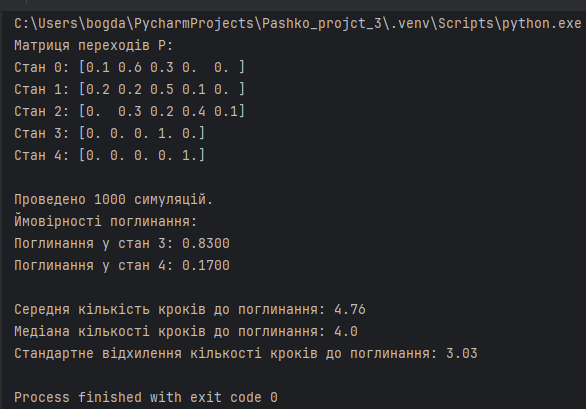
Опис ключових елементів графіка:

1. **Назва графіка**: "Розподіл поглинальних станів після симуляцій". Назва чітко вказує, що графік показує результати симуляцій, зокрема, в які з поглинаючих станів найчастіше переходив ланцюг Маркова після завершення.
2. **Розподіл поглинальних станів**:
   * **Червона частина (Стан 3)**: Стан 3 був кінцевим (поглинаючим) станом у 83% випадків. Це означає, що більшість реалізацій ланцюга Маркова закінчувалися в стані 3. Висока ймовірність досягнення цього стану свідчить про перевагу або стабільність переходів до стану 3 порівняно з іншими станами.
   * **Блакитна частина (Стан 4)**: Стан 4 був кінцевим станом у 17% випадків. Це вказує на те, що лише невелика частина симуляцій завершувалася в цьому поглинаючому стані.
3. **Відсоткове значення**:
   * **83% для Стану 3**: Червона частина діаграми складає 83%, що вказує на значно вищу ймовірність завершення в стані 3. Це може бути результатом більш високих ймовірностей переходу до стану 3 у порівнянні зі станом 4.
   * **17% для Стану 4**: Блакитна частина становить 17%, що означає, що хоча цей стан також є поглинаючим, ймовірність досягнення його була значно меншою.
4. **Візуалізація розподілу**:
   * **Порівняння поглинаючих станів**: За допомогою кругової діаграми легко побачити відмінність у частоті досягнення кожного поглинаючого стану. Перевага Стану 3 значно очевидна через його більшу площу.
   * **Кольори**: Для Стану 3 використано червоний колір, а для Стану 4 — блакитний колір. Вони добре контрастують між собою, що дозволяє легко ідентифікувати кожен стан на графіку.
5. **Інтерпретація даних**:
   * Такий розподіл поглинальних станів може свідчити про те, що у моделі ланцюга Маркова ймовірності переходу були налаштовані таким чином, що найчастіше ланцюг досягає Стану 3, а не Стану 4.
   * Висока частота досягнення Стану 3 може означати, що цей стан має більшу "притягувальну силу", тобто до нього частіше відбуваються переходи зі звичайних станів.
6. **Висновок**:
   * Розподіл поглинальних станів після симуляцій показує, що більшість реалізацій ланцюга Маркова завершуються в одному конкретному поглинаючому стані (Стан 3).
   * Це може бути важливим для аналізу стабільності системи або визначення ймовірності завершення певного процесу в конкретному стані.



Опис ключових елементів графа:

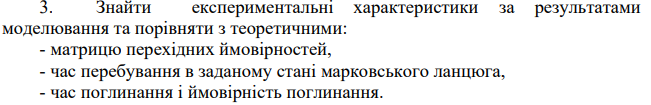
1. **Назва графіка**: "Поглинаючий ланцюг Маркова з 5 станами". Це означає, що система має п'ять можливих станів, серед яких деякі є поглинаючими.
2. **Вузли**:
   * **Стан S0, S1, S2**: Ці вузли зображені блакитним кольором і є звичайними (непоглинаючими) станами. Вони можуть переходити до інших станів або до поглинаючих станів з певними ймовірностями.
   * **Стан S3 і S4**: Ці вузли позначені червоним кольором і є поглинаючими. Це означає, що коли система потрапляє в ці стани, вона залишається там назавжди, імовірність переходу з них дорівнює 1.
3. **Стрілки (ребра)**:
   * Стрілки представляють ймовірності переходу між станами. Всі стрілки мають підпис з відповідним значенням ймовірності переходу, яке вказує, з якою ймовірністю система перейде від одного стану до іншого.
   * Наприклад, стрілка з вузла **S2** до **S3** має підпис "0.40", що означає, що ймовірність переходу зі стану **S2** до **S3** становить 40%.
   * Ймовірності вказані на кожній стрілці, щоб показати вірогідність переходу між станами.
4. **Структура переходів**:
   * Система починає з непоглинаючих станів (S0, S1, S2), і залежно від ймовірностей може переходити між ними або до одного з поглинаючих станів (S3 або S4).
   * Наприклад, стан **S1** має ймовірності переходу до **S0**, **S2**, і **S3**, що відображається на графіку відповідними стрілками.
5. **Поглинаючі стани (S3, S4)**:
   * Поглинаючі стани **S3** і **S4** мають стрілки, які вказують на них самих, з ймовірністю 1. Це означає, що коли система потрапляє в один з цих станів, вона залишається в ньому назавжди.
   * Це моделює ситуацію, коли процес "поглинається" цим станом і не може з нього вийти.
6. **Ймовірності переходу**:
   * Значення ймовірностей переходів між станами варіюються і показують, наскільки імовірно, що система перейде з одного стану в інший.
   * Наприклад, з стану **S2** є ймовірність 0.10 перейти в стан **S0**, 0.30 — в стан **S1**, 0.20 — залишитись в стані **S2**, 0.10 — в поглинаючий стан **S4**, і 0.40 — в поглинаючий стан **S3**.
7. **Висновок**:
   * Графік наочно показує, як система, що починає з одного зі звичайних станів, може поступово переходити між різними станами, аж доки не досягне одного з поглинаючих станів (S3 або S4), де процес завершується.
   * Це дозволяє проаналізувати ймовірності переходів та зрозуміти, які стани є "притягальними" в сенсі завершення процесу в певному стані.



На скріншоті показано результати симуляції поглинаючого ланцюга Маркова для системи з п'ятьма станами. Опис результатів на зображенні:

1. **Матриця переходів P**:
   * Матриця переходів представляє ймовірності переходів між станами.
   * Кожен рядок відповідає певному стану, а значення в ньому визначають ймовірності переходу з поточного стану до інших станів.
   * Наприклад, для стану 0:
     + Імовірність залишитися в стані 0 — 0.0
     + Імовірність перейти в стан 1 — 0.6
     + Імовірність перейти в стан 2 — 0.3
     + Імовірність перейти до поглинаючих станів — 0.0.
2. **Інформація про проведені симуляції**:
   * Було проведено **1000 симуляцій**, щоб оцінити поведінку ланцюга до моменту поглинання.
3. **Ймовірності поглинання**:
   * Ймовірність поглинання в **стан 3** складає **83.0%** (0.8300).
   * Ймовірність поглинання в **стан 4** складає **17.0%** (0.1700).
   * Ці ймовірності показують, наскільки часто ланцюг закінчується у відповідних станах після достатньої кількості симуляцій.
4. **Статистичні показники для кроків до поглинання**:
   * **Середня кількість кроків до поглинання**: **4.76**. Це середня кількість кроків, необхідна для досягнення одного з поглинаючих станів.
   * **Медіана кількості кроків до поглинання**: **4.0**. Це показує, що в половині випадків ланцюг досягав поглинаючого стану за 4 кроки або менше.
   * **Стандартне відхилення кількості кроків до поглинання**: **3.03**. Це показник розсіювання кількості кроків від середнього значення.

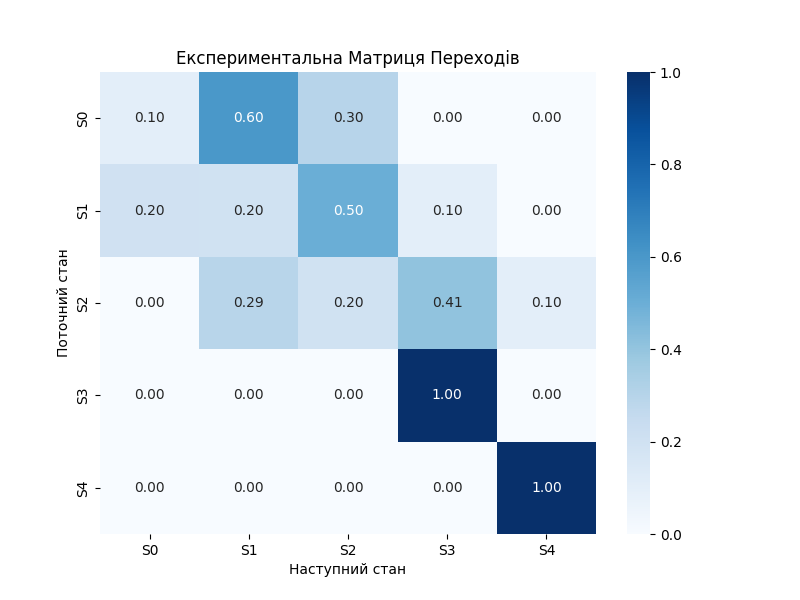
**2.3**



import numpy as np  
import matplotlib.pyplot as plt  
import networkx as nx  
from collections import defaultdict  
import seaborn as sns  
  
# Визначення кількості станів  
num\_states = 5  
  
# Індексування станів від 0 до 4  
states = list(range(num\_states))  
  
# Визначення поглинаючих станів (стани 3 та 4)  
absorbing\_states = [3, 4]  
  
# Визначення непоглинаючих станів  
transient\_states = [s for s in states if s not in absorbing\_states]  
  
# Створення матриці переходів P з нулями  
P = np.zeros((num\_states, num\_states))  
  
# Заповнення матриці переходів  
# Задаємо конкретні ймовірності переходів  
  
# Стан 0  
P[0] = [0.1, 0.6, 0.3, 0.0, 0.0]  
  
# Стан 1  
P[1] = [0.2, 0.2, 0.5, 0.1, 0.0]  
  
# Стан 2  
P[2] = [0.0, 0.3, 0.2, 0.4, 0.1]  
  
# Поглинаючі стани залишаються самі у собі  
for absorb\_state in absorbing\_states:  
 P[absorb\_state][absorb\_state] = 1.0  
  
# Вектор початкових ймовірностей (початковий стан – стан 0)  
initial\_state = 0  
  
# Вивід матриці переходів  
print("Теоретична матриця переходів P:")  
for i, row in enumerate(P):  
 print(f"Стан {i}: {row}")  
  
# Теоретичні обчислення  
# Визначення розмірів  
t = len(transient\_states)  
r = len(absorbing\_states)  
  
# Створення матриці Q  
Q = P[np.ix\_(transient\_states, transient\_states)]  
  
# Створення матриці R  
R = P[np.ix\_(transient\_states, absorbing\_states)]  
  
print("\nМатриця Q (переходи між непоглинаючими станами):")  
print(Q)  
  
print("\nМатриця R (переходи до поглинаючих станів):")  
print(R)  
  
# Одинична матриця розміру Q  
I = np.eye(t)  
  
# Обчислення фундаментальної матриці N  
N = np.linalg.inv(I - Q)  
  
print("\nФундаментальна матриця N:")  
print(N)  
  
# Обчислення матриці поглинання B  
B = np.dot(N, R)  
  
print("\nМатриця поглинання B:")  
print(B)  
  
# Теоретичні ймовірності поглинання  
print("\nТеоретичні ймовірності поглинання з початкового стану S0:")  
for idx, state in enumerate(absorbing\_states):  
 print(f"Поглинання у стан {state}: {B[0][idx]:.4f}")  
  
# Обчислення очікуваного часу до поглинання  
expected\_time\_to\_absorption = np.dot(N, np.ones((t, 1))).flatten()  
  
print("\nТеоретичний очікуваний час до поглинання для кожного непоглинаючого стану:")  
for idx, state in enumerate(transient\_states):  
 print(f"Стан {state}: {expected\_time\_to\_absorption[idx]:.4f} кроків")  
  
def simulate\_markov\_chain(P, initial\_state, absorbing\_states):  
 *"""  
 Симулює одну реалізацію поглинаючого ланцюга Маркова до моменту поглинання.  
  
 Parameters:  
 P (numpy.ndarray): Матриця переходів.  
 initial\_state (int): Початковий стан.  
 absorbing\_states (list): Список поглинаючих станів.  
  
 Returns:  
 steps (int): Кількість кроків до поглинання.  
 final\_state (int): Поглинаючий стан, у який потрапив ланцюг.  
 path (list): Послідовність станів у цій реалізації.  
 """* current\_state = initial\_state  
 steps = 0  
 path = [current\_state]  
  
 while current\_state not in absorbing\_states:  
 # Вибір наступного стану на основі ймовірностей переходів  
 next\_state = np.random.choice(states, p=P[current\_state])  
 path.append(next\_state)  
 steps += 1  
 current\_state = next\_state  
  
 return steps, current\_state, path  
  
# Кількість симуляцій  
num\_simulations = 10000  
  
# Збір статистичних даних  
steps\_to\_absorption = []  
absorption\_counts = defaultdict(int)  
paths = []  
state\_visit\_counts = defaultdict(int) # Для часу перебування в заданому стані  
  
# Визначення заданого стану для часу перебування (наприклад, стан 1)  
target\_state = 1  
  
for \_ in range(num\_simulations):  
 steps, final\_state, path = simulate\_markov\_chain(P, initial\_state, absorbing\_states)  
 steps\_to\_absorption.append(steps)  
 absorption\_counts[final\_state] += 1  
 paths.append(path)  
 # Підрахунок кількості відвідувань заданого стану  
 state\_visit\_counts[path.count(target\_state)] += 1  
  
# Обчислення ймовірності поглинання в кожен поглинаючий стан  
experimental\_absorption\_probabilities = {state: count / num\_simulations for state, count in absorption\_counts.items()}  
  
# Обчислення середнього часу до поглинання  
mean\_steps = np.mean(steps\_to\_absorption)  
median\_steps = np.median(steps\_to\_absorption)  
std\_steps = np.std(steps\_to\_absorption)  
  
print(f"\nПроведено {num\_simulations} симуляцій.")  
print("\nЕкспериментальні ймовірності поглинання:")  
for state in absorbing\_states:  
 print(f"Поглинання у стан {state}: {experimental\_absorption\_probabilities.get(state, 0):.4f}")  
  
print(f"\nЕкспериментальні характеристики часу до поглинання:")  
print(f"Середня кількість кроків до поглинання: {mean\_steps:.4f}")  
print(f"Медіана кількості кроків до поглинання: {median\_steps}")  
print(f"Стандартне відхилення кількості кроків до поглинання: {std\_steps:.4f}")  
  
# Експериментальна матриця переходів  
transition\_counts = np.zeros((num\_states, num\_states), dtype=int)  
  
for path in paths:  
 for i in range(len(path) - 1):  
 transition\_counts[path[i]][path[i+1]] += 1  
  
# Обчислення експериментальної матриці переходів  
experimental\_P = np.zeros((num\_states, num\_states))  
  
for i in range(num\_states):  
 total = transition\_counts[i].sum()  
 if total > 0:  
 experimental\_P[i] = transition\_counts[i] / total  
 else:  
 experimental\_P[i][i] = 1.0 # Поглинаючі стани залишаються самі у собі  
  
print("\nЕкспериментальна матриця переходів P:")  
for i, row in enumerate(experimental\_P):  
 print(f"Стан {i}: {row}")  
  
# Час перебування в заданому стані  
time\_in\_target\_state = []  
  
for path in paths:  
 time\_in\_target\_state.append(path.count(target\_state))  
  
# Середній час перебування в заданому стані  
mean\_time\_in\_target = np.mean(time\_in\_target\_state)  
median\_time\_in\_target = np.median(time\_in\_target\_state)  
std\_time\_in\_target = np.std(time\_in\_target\_state)  
  
print(f"\nЧас перебування в стані {target\_state}:")  
print(f"Середній час: {mean\_time\_in\_target:.4f} кроків")  
print(f"Медіана часу: {median\_time\_in\_target}")  
print(f"Стандартне відхилення часу: {std\_time\_in\_target:.4f}")  
  
# Порівняння експериментальних та теоретичних характеристик  
print("\nПорівняння ймовірностей поглинання:")  
for idx, state in enumerate(absorbing\_states):  
 theoretical = B[0][idx]  
 experimental = experimental\_absorption\_probabilities.get(state, 0)  
 print(f"Стан {state}: Теоретична = {theoretical:.4f}, Експериментальна = {experimental:.4f}")  
  
print("\nПорівняння очікуваного часу до поглинання:")  
for idx, state in enumerate(transient\_states):  
 theoretical = expected\_time\_to\_absorption[idx]  
 # Обчислення експериментального очікуваного часу для кожного стану  
 # Тут ми обчислюємо лише для початкового стану  
 if state == initial\_state:  
 experimental = mean\_steps  
 print(f"Стан {state}: Теоретичний = {theoretical:.4f}, Експериментальний = {experimental:.4f}")  
  
# Візуалізація Результатів  
  
# a. Експериментальна Матриця Переходів  
plt.figure(figsize=(8, 6))  
sns.heatmap(experimental\_P, annot=True, fmt=".2f", cmap="Blues",  
 xticklabels=[f"S{state}" for state in states],  
 yticklabels=[f"S{state}" for state in states])  
plt.title("Експериментальна Матриця Переходів")  
plt.xlabel("Наступний стан")  
plt.ylabel("Поточний стан")  
plt.show()  
  
# b. Гістограма Кроків до Поглинання  
plt.figure(figsize=(10, 6))  
sns.histplot(steps\_to\_absorption, bins=range(0, max(steps\_to\_absorption)+2), kde=False, color='skyblue')  
plt.title('Розподіл кількості кроків до поглинання')  
plt.xlabel('Кроки до поглинання')  
plt.ylabel('Частота')  
plt.grid(axis='y', alpha=0.75)  
plt.show()  
  
# c. Діаграма Пирога для Ймовірностей Поглинання  
labels = [f"Стан {state}" for state in absorbing\_states]  
sizes = [experimental\_absorption\_probabilities[state] \* 100 for state in absorbing\_states]  
colors\_pie = ['lightcoral', 'lightskyblue']  
  
plt.figure(figsize=(8, 8))  
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=140, colors=colors\_pie, shadow=True)  
plt.title('Розподіл поглинальних станів після симуляцій')  
plt.axis('equal') # Рівні пропорції  
plt.show()  
  
# d. Граф Ланцюга Маркова  
G = nx.DiGraph()  
  
# Додавання вузлів  
for state in states:  
 if state in absorbing\_states:  
 G.add\_node(state, color='red', label=f"S{state} (Поглинаючий)")  
 else:  
 G.add\_node(state, color='lightblue', label=f"S{state}")  
  
# Додавання ребер з ймовірностями  
for i in states:  
 for j in states:  
 if P[i][j] > 0:  
 G.add\_edge(i, j, weight=P[i][j])  
  
# Отримання кольорів вузлів  
node\_colors = [G.nodes[node]['color'] for node in G.nodes]  
  
# Отримання міток вузлів  
labels\_graph = {node: G.nodes[node]['label'] for node in G.nodes}  
  
# Отримання міток ребер з ймовірностями  
edge\_labels = {(i, j): f"{P[i][j]:.2f}" for i, j in G.edges()}  
  
# Відображення графа  
plt.figure(figsize=(12, 8))  
pos = nx.spring\_layout(G, seed=42) # Розміщення вузлів  
  
# Малювання вузлів  
nx.draw\_networkx\_nodes(G, pos, node\_color=node\_colors, node\_size=800)  
  
# Малювання ребер  
nx.draw\_networkx\_edges(G, pos, arrowstyle='->', arrowsize=20, edge\_color='gray')  
  
# Малювання міток вузлів  
nx.draw\_networkx\_labels(G, pos, labels\_graph, font\_size=12)  
  
# Малювання міток ребер  
nx.draw\_networkx\_edge\_labels(G, pos, edge\_labels=edge\_labels, font\_size=10)  
  
plt.title("Поглинаючий ланцюг Маркова з 5 станами")  
plt.axis('off')  
plt.show()  
  
# e. Гістограма Часу Перебування в Заданому Стані  
plt.figure(figsize=(10, 6))  
sns.histplot(time\_in\_target\_state, bins=range(0, max(time\_in\_target\_state)+2), kde=False, color='lightgreen')  
plt.title(f'Розподіл часу перебування в стані {target\_state}')  
plt.xlabel('Кількість відвідувань стану')  
plt.ylabel('Частота')  
plt.grid(axis='y', alpha=0.75)  
plt.show()

Опис коду:

1. **Імпорт бібліотек**:
   * Імпортуються необхідні бібліотеки, такі як numpy, matplotlib, networkx, seaborn та інші, для математичних обчислень, графічної візуалізації та роботи з графами.
2. **Параметри ланцюга Маркова**:
   * Визначається кількість станів (num\_states = 5).
   * Індексування станів проводиться від 0 до 4.
   * Визначаються поглинаючі стани (статус "absorbing"): стани 3 та 4.
   * Інші стани є непоглинаючими (транзитивні стани).
3. **Матриця переходів**:
   * Створюється матриця переходів P розміром 5x5, яка спочатку заповнена нулями.
   * Для кожного стану задаються ймовірності переходів до інших станів:
     + Наприклад, стан 0 має ймовірність 0.1 для переходу до стану 0, 0.6 — до стану 1, і так далі.
     + Поглинаючі стани переходять лише до самих себе з імовірністю 1.
4. **Теоретичні обчислення**:
   * Матриця переходів розбивається на матриці Q (між транзитивними станами) та R (з транзитивних до поглинаючих станів).
   * Фундаментальна матриця N обчислюється як інверсія матриці (I - Q).
   * Матриця поглинання B обчислюється як добуток матриць N та R, і вона представляє ймовірності поглинання в кожен поглинаючий стан з кожного транзитивного.
5. **Симуляція ланцюга Маркова**:
   * Функція simulate\_markov\_chain() виконує симуляцію ланцюга Маркова до моменту поглинання.
   * Починаючи з початкового стану, визначається наступний стан на основі ймовірностей переходу, і так продовжується, поки не буде досягнутий поглинаючий стан.
   * Для кожної симуляції записується кількість кроків до поглинання, фінальний стан та шлях, яким проходив ланцюг.
6. **Збір та аналіз результатів симуляцій**:
   * Виконуються **10,000 симуляцій** для збору статистичних даних про поглинання.
   * Обчислюються ймовірності поглинання для кожного з поглинаючих станів.
   * Обчислюється середня кількість кроків до поглинання, медіана та стандартне відхилення кількості кроків.
   * Для кожного стану визначається кількість відвідувань, а також час перебування у визначеному стані.
7. **Експериментальна матриця переходів**:
   * Обчислюється експериментальна матриця переходів на основі зібраних шляхів (тобто з реальних симуляцій).
   * Ця матриця порівнюється з теоретичною матрицею переходів.
8. **Візуалізація результатів**:
   * **Експериментальна матриця переходів**: відображена як теплокарта (heatmap).
   * **Гістограма кроків до поглинання**: показує розподіл кількості кроків, необхідних для поглинання.
   * **Діаграма ймовірностей поглинання**: відображає ймовірність поглинання в кожен стан у вигляді кругової діаграми.
   * **Граф ланцюга Маркова**: відображає схему переходів між станами з відповідними ймовірностями.
   * **Гістограма часу перебування у визначеному стані**: показує розподіл часу, проведеного у заданому транзитивному стані.
9. **Порівняння теоретичних і експериментальних результатів**:
   * Порівнюються ймовірності поглинання, отримані теоретичним шляхом та шляхом симуляцій.
   * Також порівнюється очікуваний час до поглинання для кожного транзитивного стану між теоретичними і експериментальними результатами.



На скріншоті представлено **експериментальну матрицю переходів** для ланцюга Маркова з п'ятьма станами (S0, S1, S2, S3, S4). Це візуалізація, що представляє ймовірності переходу між станами на основі проведених симуляцій.

**Структура матриці**

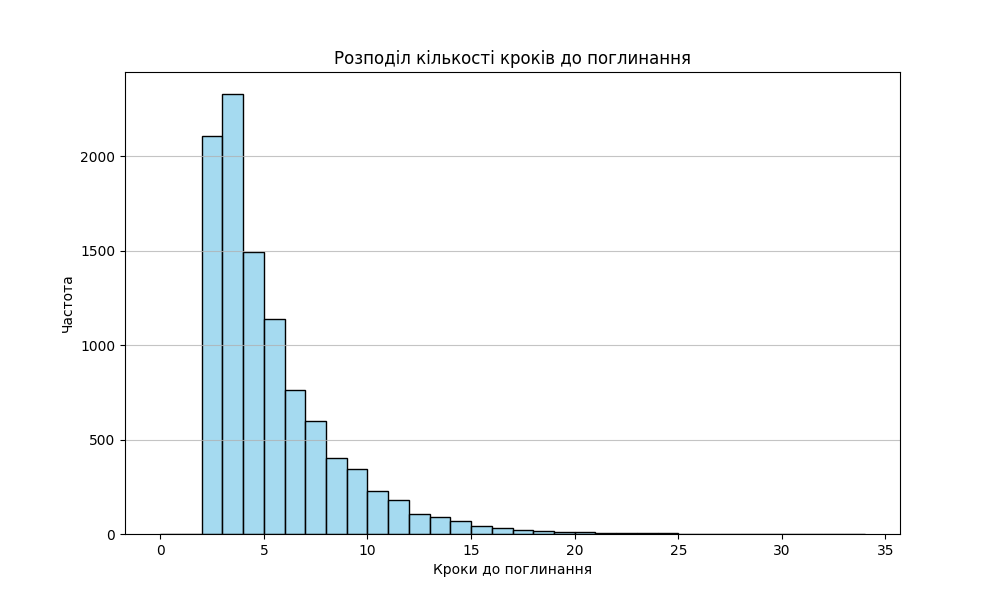
* **Матриця** має розмір 5x5, де кожен рядок і стовпець відповідає станам S0, S1, S2, S3, S4.
* **Рядки** представляють **поточний стан**, а **стовпці** представляють **наступний стан**.
* Кожна **клітинка матриці** показує ймовірність переходу від одного стану до іншого.

**Кольорове кодування**

* **Колір клітинки** визначає значення ймовірності переходу:
  + **Світліші кольори** відповідають **меншим ймовірностям**.
  + **Темніші кольори** відображають **вищі ймовірності**.
  + Найтемніший колір відповідає значенню 1.00, що означає 100% ймовірність переходу.
* Наприклад, для станів S3 і S4, які є **поглинаючими станами**, ми бачимо, що відповідні клітинки (переходи S3 → S3 та S4 → S4) мають значення 1.00 і темно-синій колір, що означає, що, потрапивши в ці стани, ланцюг не може перейти в інші стани — це є **характеристика поглинаючих станів**.

**Аналіз матриці переходів**

1. **Поглинаючі стани** (S3 і S4):
   * Вони мають ймовірність 1.00 залишитись у самих собі, що вказує на те, що ці стани є кінцевими для ланцюга.
   * Зображені у темно-синій колір, який показує, що ймовірність залишитись у цих станах максимальна.
2. **Непоглинаючі стани** (S0, S1, S2):
   * Ці стани мають різні ймовірності переходів до інших станів.
   * Наприклад:
     + З **S0** є ймовірності переходу:
       - В **S1** — 0.60 (найбільша ймовірність серед можливих переходів зі стану S0).
       - В **S2** — 0.30.
       - В **S0** — 0.10 (ймовірність залишитись у самому S0).
     + З **S1**:
       - Ймовірність перейти до S2 — 0.50 (найвища ймовірність).
       - Також є ймовірність залишитись у самому S1 (0.20).
       - В **S0** та **S3** — відповідно 0.20 та 0.10.
     + З **S2**:
       - Висока ймовірність перейти до **S3** (0.41), що робить цей стан частіше обираним.
       - Також можливі переходи до **S1** (0.29), **S4** (0.10), а ймовірність залишитись в **S2** становить 0.20.



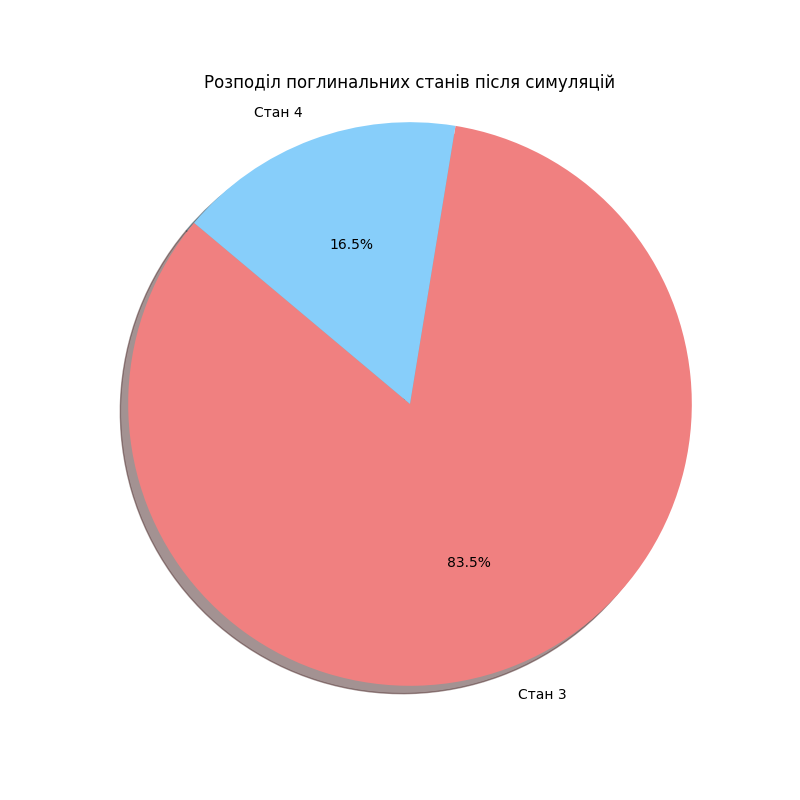
На скріншоті представлена **гістограма розподілу кількості кроків до поглинання** в ланцюгу Маркова для 5 станів, де два з них (S3 і S4) є поглинаючими.

**Детальний опис графіка:**

* **Заголовок**: "Розподіл кількості кроків до поглинання".
  + Описує зміст графіка, який показує, скільки кроків знадобилося для досягнення поглинаючого стану під час симуляцій ланцюга Маркова.
* **Ось X**:
  + Позначена як "Кроки до поглинання".
  + Відображає кількість кроків, які знадобилися для досягнення поглинаючого стану (S3 або S4) з початкового стану (S0).
  + Значення на осі X варіюються від 0 до приблизно 35, що вказує на те, що в деяких симуляціях знадобилося до 35 кроків для досягнення поглинаючого стану.
* **Ось Y**:
  + Позначена як "Частота".
  + Відображає кількість симуляцій, які потребували певної кількості кроків до поглинання.
  + Найвищі частоти спостерігаються на малих значеннях кроків, тобто більшість реалізацій потребували від 1 до 10 кроків.
* **Графік розподілу**:
  + Гістограма має форму, що нагадує **зворотний експоненційний розподіл**: більшість значень сконцентровані в інтервалі 0-5 кроків, а частота швидко знижується при збільшенні кількості кроків.
  + **Максимальна частота** досягається приблизно для 2-3 кроків, що означає, що саме стільки кроків найчастіше знадобилося для досягнення поглинаючого стану.
  + **Хвіст розподілу** простягається до 20-30 кроків, але частота таких значень значно менша, що свідчить про те, що траєкторії з великою кількістю кроків є рідкісними.

**Висновок:**

* **Більшість реалізацій** досягали поглинаючого стану досить швидко — за 2-5 кроків.
* Така форма розподілу свідчить про те, що існує висока ймовірність досягнення поглинаючих станів за відносно невелику кількість кроків.
* Декілька реалізацій потребували більшої кількості кроків (до 20-30), що також може бути результатом деяких маршрутів, що включають повернення до початкових станів перед поглинанням.



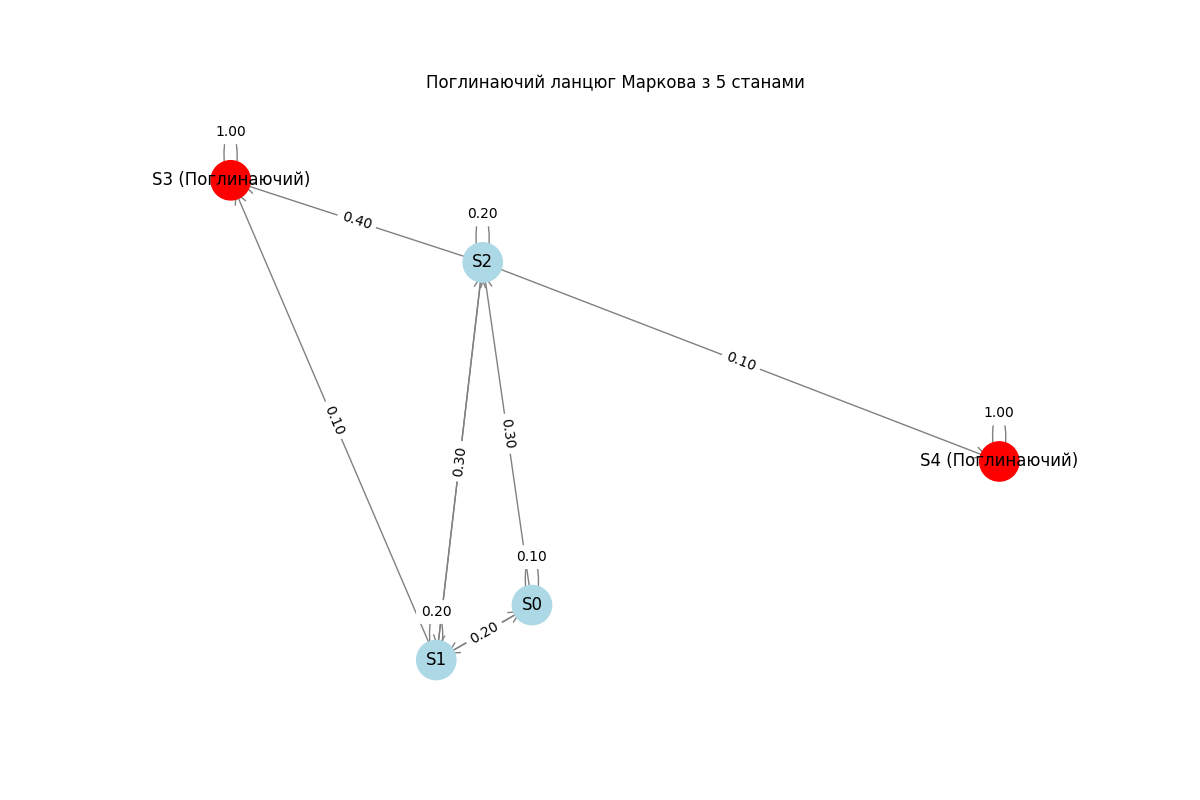
На скріншоті зображена **кругова діаграма**, яка показує розподіл поглинальних станів після симуляцій ланцюга Маркова.

**Детальний опис діаграми:**

* **Заголовок**: "Розподіл поглинальних станів після симуляцій".
  + Діаграма демонструє, яка частка всіх симуляцій завершилася в певних поглинальних станах.
* **Сегменти діаграми**:
  + Діаграма складається з двох кольорових сегментів, які відповідають поглинальним станам (Стан 3 і Стан 4).
* **Розподіл сегментів**:
  + **Червоний сегмент** (мітка "Стан 3"): займає **83.5%** усієї площі діаграми.
    - Це означає, що 83.5% симуляцій завершилися в **Стані 3**, тобто цей стан був найбільш поширеним кінцевим пунктом для системи.
  + **Блакитний сегмент** (мітка "Стан 4"): займає **16.5%** усієї площі діаграми.
    - Це означає, що 16.5% симуляцій завершилися в **Стані 4**.

**Висновок:**

* **Стан 3** є значно більш імовірним поглинальним станом, ніж **Стан 4**, що вказує на те, що траєкторії ланцюга Маркова частіше ведуть до цього стану.
* Така значна різниця між ймовірностями досягнення двох поглинальних станів може бути результатом різних ймовірностей переходів, закладених в матрицю переходів системи.



**Опис елементів графа:**

* **Вузли** (кружки):
  + Всі вузли представляють стани системи.
  + **Червоні вузли** позначають **поглинаючі стани** (Стан 3 та Стан 4). Ці стани є кінцевими, тобто, коли система переходить в один із цих станів, вона залишається в ньому назавжди.
  + **Блакитні вузли** (Стан 0, Стан 1, Стан 2) є **непоглинаючими станами**. Вони можуть переходити як один в одного, так і в поглинаючі стани.
* **Стрілки**:
  + Стрілки представляють переходи між станами, а їх напрямок вказує на те, в який стан можна перейти з поточного.
  + **Ймовірності переходів** вказані біля стрілок. Наприклад:
    - **З Стану 0** існує ймовірність 0.10 перейти до Стану 1, 0.20 — до Стану 1, та 0.10 — до Стан0

Теоретична матриця переходів P:

Стан 0: [0.1 0.6 0.3 0. 0. ]

Стан 1: [0.2 0.2 0.5 0.1 0. ]

Стан 2: [0. 0.3 0.2 0.4 0.1]

Стан 3: [0. 0. 0. 1. 0.]

Стан 4: [0. 0. 0. 0. 1.]

Матриця Q (переходи між непоглинаючими станами):

[[0.1 0.6 0.3]

[0.2 0.2 0.5]

[0. 0.3 0.2]]

Матриця R (переходи до поглинаючих станів):

[[0. 0. ]

[0.1 0. ]

[0.4 0.1]]

Фундаментальна матриця N:

[[1.49847095 1.74311927 1.65137615]

[0.48929664 2.20183486 1.55963303]

[0.18348624 0.82568807 1.83486239]]

Матриця поглинання B:

[[0.83486239 0.16513761]

[0.8440367 0.1559633 ]

[0.81651376 0.18348624]]

Теоретичні ймовірності поглинання з початкового стану S0:

Поглинання у стан 3: 0.8349

Поглинання у стан 4: 0.1651

Теоретичний очікуваний час до поглинання для кожного непоглинаючого стану:

Стан 0: 4.8930 кроків

Стан 1: 4.2508 кроків

Стан 2: 2.8440 кроків

Проведено 10000 симуляцій.

Експериментальні ймовірності поглинання:

Поглинання у стан 3: 0.8353

Поглинання у стан 4: 0.1647

Експериментальні характеристики часу до поглинання:

Середня кількість кроків до поглинання: 4.8341

Медіана кількості кроків до поглинання: 4.0

Стандартне відхилення кількості кроків до поглинання: 3.1152

Експериментальна матриця переходів P:

Стан 0: [0.10067518 0.59917107 0.30015375 0. 0. ]

Стан 1: [0.201859 0.19613001 0.50222144 0.09978955 0. ]

Стан 2: [0. 0.29417547 0.1963013 0.40833129 0.10119194]

Стан 3: [0. 0. 0. 1. 0.]

Стан 4: [0. 0. 0. 0. 1.]

Час перебування в стані 1:

Середній час: 1.7106 кроків

Медіана часу: 1.0

Стандартне відхилення часу: 1.6657

Порівняння ймовірностей поглинання:

Стан 3: Теоретична = 0.8349, Експериментальна = 0.8353

Стан 4: Теоретична = 0.1651, Експериментальна = 0.1647

Порівняння очікуваного часу до поглинання:

Стан 0: Теоретичний = 4.8930, Експериментальний = 4.8341

Цей консольний вивід містить результати моделювання поглинаючого ланцюга Маркова та порівняння теоретичних і експериментальних характеристик. Ось короткий опис кожної частини виводу:

**Теоретична матриця переходів P**

* Показує ймовірності переходів між станами для кожного з 5 станів. Наприклад, з **Стану 0** існує 10% ймовірності залишитись, 60% - перейти у Стан 1, і 30% - перейти у Стан 2.

**Матриця Q (переходи між непоглинаючими станами)**

* Містить ймовірності переходів між непоглинаючими станами (0, 1, 2). Вона є підматрицею початкової матриці переходів P і показує ймовірності залишатись або перейти між непоглинаючими станами.

**Матриця R (переходи до поглинаючих станів)**

* Містить ймовірності переходів з непоглинаючих станів до поглинаючих (3 і 4). Наприклад, для **Стану 2** існує 40% ймовірності потрапити до **Стану 3** і 10% - до **Стану 4**.

**Фундаментальна матриця N**

* Фундаментальна матриця N визначає середню кількість відвідувань непоглинаючих станів до моменту поглинання. Кожний елемент N[i][j] вказує на середню кількість відвідувань стану j перед поглинанням, якщо початок був у стані i.

**Матриця поглинання B**

* Матриця B показує ймовірності поглинання для кожного непоглинаючого стану. Наприклад, ймовірність потрапити до **Стану 3** з **Стану 0** становить приблизно 0.8349, а ймовірність потрапити до **Стану 4** - 0.1651.

**Теоретичні ймовірності поглинання**

* Показує теоретичні ймовірності поглинання з **початкового стану 0**. Наприклад, ймовірність поглинання у **Стан 3** становить приблизно 0.8349.

**Теоретичний очікуваний час до поглинання**

* Обчислюється для кожного непоглинаючого стану і показує середню кількість кроків, необхідних для поглинання. Наприклад, для **Стану 0** теоретичний очікуваний час до поглинання становить приблизно 4.8930 кроків.

**Експериментальні результати**

* **Експериментальні ймовірності поглинання**: Показують результати 10000 симуляцій. Ймовірність поглинання у **Стан 3** становить приблизно 0.8353, а у **Стан 4** - 0.1647. Значення дуже близькі до теоретичних, що підтверджує точність моделювання.
* **Експериментальні характеристики часу до поглинання**: Середня кількість кроків до поглинання - 4.8341, що є близьким до теоретичних 4.8930 кроків.
* **Експериментальна матриця переходів**: Містить ймовірності переходів між станами, що обчислюються на основі симуляцій. Вони також дуже близькі до теоретичних значень.

**Час перебування в стані 1**

* Показує середній час перебування в **Стані 1** під час симуляцій: середній час - 1.7106 кроків, медіана часу - 1 крок, стандартне відхилення - 1.6657 кроків.

**Порівняння теоретичних та експериментальних характеристик**

* **Ймовірності поглинання**: Теоретичні та експериментальні ймовірності поглинання для **Стану 3** і **Стану 4** дуже схожі, що підтверджує правильність проведених симуляцій.
* **Очікуваний час до поглинання**: Теоретичний і експериментальний очікуваний час до поглинання для **Стану 0** теж дуже схожі, що свідчить про точність розрахунків і симуляцій.

**2.4**



import numpy as np  
import matplotlib.pyplot as plt  
import networkx as nx  
import seaborn as sns  
from scipy.linalg import eig  
import pandas as pd  
  
# Визначення кількості станів  
num\_states = 7  
states = list(range(num\_states))  
  
# Створення матриці переходів P з випадковими ймовірностями  
np.random.seed(42) # Для відтворюваності результатів  
P = np.random.rand(num\_states, num\_states)  
  
# Нормалізація рядків матриці P, щоб сума дорівнювала 1  
P = P / P.sum(axis=1, keepdims=True)  
  
print("Матриця переходів P:")  
for i, row in enumerate(P):  
 print(f"Стан {i}: {row}")  
  
# Вектор початкових ймовірностей (наприклад, початковий стан - стан 0)  
pi = np.zeros(num\_states)  
pi[0] = 1.0 # Початковий стан - стан 0  
  
print("\nВектор початкових станів pi:")  
print(pi)  
  
  
def is\_regular(P):  
 *"""  
 Перевіряє, чи є матриця переходів P регулярною.  
  
 Parameters:  
 P (numpy.ndarray): Матриця переходів.  
  
 Returns:  
 bool: True, якщо P регулярна, інакше False.  
 """* # Піднесення матриці P до степеня k, поки всі елементи не стануть додатніми  
 k = 1  
 while k <= 100: # Максимум 100 ітерацій  
 P\_k = np.linalg.matrix\_power(P, k)  
 if np.all(P\_k > 0):  
 print(f"\nМатриця переходів P є регулярною при k = {k}.")  
 return True  
 k += 1  
 print("\nМатриця переходів P НЕ є регулярною.")  
 return False  
  
  
# Перевірка регулярності  
regular = is\_regular(P)  
  
  
def compute\_stationary\_distribution(P):  
 *"""  
 Обчислює стаціонарну розподіл для матриці переходів P.  
  
 Parameters:  
 P (numpy.ndarray): Матриця переходів.  
  
 Returns:  
 pi\_star (numpy.ndarray): Стаціонарна розподіл.  
 """* # Обчислення власних значень та власних векторів  
 eigenvalues, eigenvectors = eig(P.T)  
  
 # Знаходимо індекс власного значення, яке дорівнює 1  
 index = np.argmin(np.abs(eigenvalues - 1))  
  
 # Отримуємо відповідний власний вектор  
 pi\_star = np.real(eigenvectors[:, index])  
  
 # Нормалізація вектора  
 pi\_star = pi\_star / pi\_star.sum()  
  
 return pi\_star  
  
  
# Обчислення стаціонарної розподілу  
pi\_star = compute\_stationary\_distribution(P)  
  
print("\nТеоретична стаціонарна розподіл pi\*:")  
print(pi\_star)  
print(f"Сума елементів pi\*: {pi\_star.sum():.4f}")  
  
  
def simulate\_markov\_chain(P, initial\_state, num\_steps):  
 *"""  
 Симулює ланцюг Маркова на задану кількість кроків.  
  
 Parameters:  
 P (numpy.ndarray): Матриця переходів.  
 initial\_state (int): Початковий стан.  
 num\_steps (int): Кількість кроків для симуляції.  
  
 Returns:  
 path (list): Послідовність станів у симуляції.  
 """* current\_state = initial\_state  
 path = [current\_state]  
  
 for \_ in range(num\_steps):  
 next\_state = np.random.choice(states, p=P[current\_state])  
 path.append(next\_state)  
 current\_state = next\_state  
  
 return path  
  
  
# Параметри симуляції  
num\_simulations = 10000 # Кількість симуляцій  
num\_steps = 50 # Кількість кроків у кожній симуляції  
  
# Збір статистичних даних  
state\_counts = np.zeros(num\_states)  
state\_visit\_counts = np.zeros(num\_states)  
paths = []  
  
for \_ in range(num\_simulations):  
 path = simulate\_markov\_chain(P, initial\_state=0, num\_steps=num\_steps)  
 paths.append(path)  
 # Підрахунок кількості відвідувань кожного стану  
 unique, counts = np.unique(path, return\_counts=True)  
 state\_visit\_counts[unique] += counts  
 # Підрахунок кінцевих станів  
 state\_counts[path[-1]] += 1  
  
# Обчислення експериментальної стаціонарної розподілу  
experimental\_pi\_star = state\_visit\_counts / (num\_simulations \* num\_steps)  
  
print("\nЕкспериментальна стаціонарна розподіл pi\*:")  
print(experimental\_pi\_star)  
print(f"Сума елементів експериментальної pi\*: {experimental\_pi\_star.sum():.4f}")  
  
# Обчислення експериментальних ймовірностей кінцевих станів  
experimental\_final\_probs = state\_counts / num\_simulations  
  
print("\nЕкспериментальні ймовірності кінцевих станів:")  
for i, prob in enumerate(experimental\_final\_probs):  
 print(f"Стан {i}: {prob:.4f}")  
  
# Обчислення експериментальної матриці переходів  
transition\_counts = np.zeros((num\_states, num\_states), dtype=int)  
  
for path in paths:  
 for i in range(len(path) - 1):  
 transition\_counts[path[i]][path[i + 1]] += 1  
  
# Обчислення експериментальної матриці переходів  
experimental\_P = np.zeros((num\_states, num\_states))  
  
for i in range(num\_states):  
 total = transition\_counts[i].sum()  
 if total > 0:  
 experimental\_P[i] = transition\_counts[i] / total  
 else:  
 experimental\_P[i][i] = 1.0 # Для випадків без переходів  
  
print("\nЕкспериментальна матриця переходів P:")  
for i, row in enumerate(experimental\_P):  
 print(f"Стан {i}: {row}")  
  
# Обчислення часу перебування в кожному стані  
# Середній час перебування в кожному стані  
mean\_time\_in\_states = state\_visit\_counts / num\_simulations  
print("\nСередній час перебування в кожному стані:")  
for i, mean\_time in enumerate(mean\_time\_in\_states):  
 print(f"Стан {i}: {mean\_time:.4f} кроків")  
  
# Візуалізація Матриці Переходів  
plt.figure(figsize=(12, 5))  
  
plt.subplot(1, 2, 1)  
sns.heatmap(P, annot=True, fmt=".2f", cmap="Blues",  
 xticklabels=[f"S{state}" for state in states],  
 yticklabels=[f"S{state}" for state in states])  
plt.title("Теоретична Матриця Переходів")  
plt.xlabel("Наступний стан")  
plt.ylabel("Поточний стан")  
  
plt.subplot(1, 2, 2)  
sns.heatmap(experimental\_P, annot=True, fmt=".2f", cmap="Greens",  
 xticklabels=[f"S{state}" for state in states],  
 yticklabels=[f"S{state}" for state in states])  
plt.title("Експериментальна Матриця Переходів")  
plt.xlabel("Наступний стан")  
plt.ylabel("Поточний стан")  
  
plt.tight\_layout()  
plt.show()  
  
# Візуалізація Стаціонарної Розподілу  
stationary\_df = {  
 'Стан': [f"S{i}" for i in states],  
 'Теоретична pi\*': pi\_star,  
 'Експериментальна pi\*': experimental\_pi\_star  
}  
  
stationary\_df = pd.DataFrame(stationary\_df)  
  
# Візуалізація стаціонарної розподілу  
stationary\_df\_melted = stationary\_df.melt(id\_vars='Стан', var\_name='Тип', value\_name='Ймовірність')  
  
plt.figure(figsize=(10, 6))  
sns.barplot(data=stationary\_df\_melted, x='Стан', y='Ймовірність', hue='Тип')  
plt.title("Теоретична vs Експериментальна Стаціонарна Розподіл")  
plt.ylabel("Ймовірність")  
plt.show()  
  
# Візуалізація Ланцюга Маркова за Допомогою NetworkX  
G = nx.DiGraph()  
  
# Додавання вузлів  
for state in states:  
 G.add\_node(state, label=f"S{state}")  
  
# Додавання ребер з ймовірностями  
for i in states:  
 for j in states:  
 if P[i][j] > 0:  
 G.add\_edge(i, j, weight=P[i][j])  
  
# Отримання міток вузлів  
labels\_graph = {node: G.nodes[node]['label'] for node in G.nodes}  
  
# Отримання міток ребер з ймовірностями  
edge\_labels = {(i, j): f"{P[i][j]:.2f}" for i, j in G.edges()}  
  
# Отримання позицій вузлів  
pos = nx.spring\_layout(G, seed=42) # Фіксована позиція для стабільності  
  
# Візуалізація графа  
plt.figure(figsize=(12, 8))  
nx.draw\_networkx\_nodes(G, pos, node\_color='lightblue', node\_size=800)  
nx.draw\_networkx\_edges(G, pos, arrowstyle='->', arrowsize=20, edge\_color='gray')  
nx.draw\_networkx\_labels(G, pos, labels\_graph, font\_size=12)  
nx.draw\_networkx\_edge\_labels(G, pos, edge\_labels=edge\_labels, font\_size=10)  
  
plt.title("Граф Регулярного Ланцюга Маркова з 7 Станами")  
plt.axis('off')  
plt.show()  
  
# Візуалізація Стаціонарної Розподілу через Діаграму Пирога  
fig, axes = plt.subplots(1, 2, figsize=(16, 8))  
  
# Теоретична pi\*  
axes[0].pie(pi\_star, labels=[f"S{i}" for i in states], autopct='%1.1f%%', startangle=140,  
 colors=plt.cm.Blues(np.linspace(0, 1, num\_states)))  
axes[0].set\_title("Теоретична Стаціонарна Розподіл")  
  
# Експериментальна pi\*  
axes[1].pie(experimental\_pi\_star, labels=[f"S{i}" for i in states], autopct='%1.1f%%', startangle=140,  
 colors=plt.cm.Greens(np.linspace(0, 1, num\_states)))  
axes[1].set\_title("Експериментальна Стаціонарна Розподіл")  
  
plt.show()  
  
# Візуалізація Конвергенції Розподілу Ймовірностей до Стаціонарної pi\*  
# Візьмемо одну симуляцію для демонстрації  
sample\_path = simulate\_markov\_chain(P, initial\_state=0, num\_steps=num\_steps)  
sample\_distribution = np.zeros((num\_steps + 1, num\_states))  
current\_distribution = pi.copy()  
sample\_distribution[0] = current\_distribution  
  
for step in range(1, num\_steps + 1):  
 current\_distribution = np.dot(current\_distribution, P)  
 sample\_distribution[step] = current\_distribution  
  
plt.figure(figsize=(12, 8))  
for state in states:  
 plt.plot(range(num\_steps + 1), sample\_distribution[:, state], label=f"S{state}")  
  
plt.hlines(pi\_star, 0, num\_steps, colors='k', linestyles='dashed', label='Стаціонарна pi\*')  
plt.title("Конвергенція Розподілу Ймовірностей до Стаціонарної pi\*")  
plt.xlabel("Кроки")  
plt.ylabel("Ймовірність")  
plt.legend()  
plt.grid(True)  
plt.show()

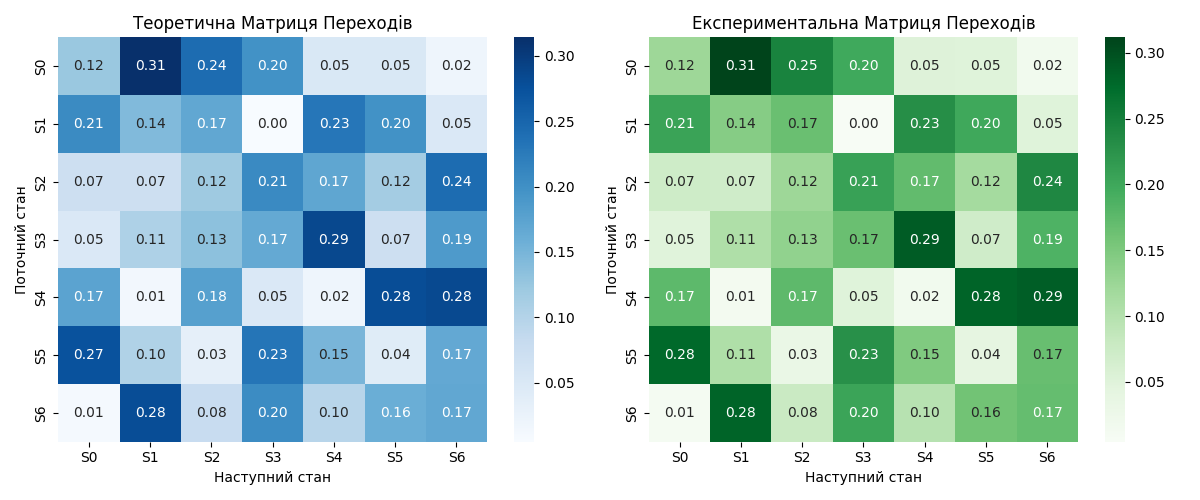
Цей код моделює регулярний ланцюг Маркова з 7 станами. Опис коду та його основні етапи наведено нижче:

**Основні етапи реалізації:**

1. **Ініціалізація матриці переходів**:
   * Створюється матриця переходів P для 7 станів, яка заповнюється випадковими ймовірностями, а потім нормалізується, щоб кожний рядок матриці (що представляє можливі переходи з одного стану) мав суму 1.
2. **Перевірка регулярності матриці**:
   * Перевіряється, чи є матриця переходів регулярною, тобто чи можливий перехід між усіма станами протягом скінченної кількості кроків.
3. **Обчислення стаціонарного розподілу**:
   * Обчислюється теоретичний стаціонарний розподіл pi\* за допомогою знаходження власного вектора матриці переходів.
   * Також моделюється розподіл ймовірностей для 10000 симуляцій ланцюга Маркова по 50 кроків кожен. Експериментальний стаціонарний розподіл порівнюється з теоретичним.
4. **Симуляція ланцюга Маркова**:
   * Створюється функція для симуляції ланцюга Маркова, яка стартує з початкового стану 0 і симулює перехід по матриці P на основі заданої кількості кроків.
5. **Збір статистичних даних**:
   * Проводиться симуляція для 10000 траєкторій по 50 кроків.
   * Збирається інформація про кінцеві стани, середній час перебування у кожному стані та експериментальна матриця переходів.
6. **Візуалізація результатів**:
   * Графіки показують:
     + **Теоретичну та експериментальну матрицю переходів**: з використанням heatmap (теплової карти).
     + **Теоретичну та експериментальну стаціонарну розподіл**: за допомогою діаграми стовпців.
     + **Граф регулярного ланцюга Маркова** з використанням NetworkX, де показані ймовірності переходів між станами.
     + **Порівняння теоретичної та експериментальної стаціонарної розподілу** через кругові діаграми.
     + **Конвергенцію розподілу ймовірностей до стаціонарного розподілу** з кроками.

**Важливі моменти:**

* **Регулярність ланцюга Маркова**:
  + Код перевіряє, чи є матриця регулярною, тобто чи зможе ланцюг Маркова досягти будь-якого стану, незалежно від початкового стану.
* **Стаціонарний розподіл**:
  + Теоретично обчислюється стаціонарний розподіл як власний вектор, який відповідає власному значенню 1.
  + Експериментально збираються дані шляхом симуляцій для оцінки стаціонарного розподілу.
* **Симуляція ланцюга Маркова**:
  + Ланцюг починає в стані S0 і моделює переходи згідно ймовірностей матриці P. Це допомагає оцінити експериментальні розподіли ймовірностей.
* **Теоретична та експериментальна матриця переходів**:
  + Створені теплові карти дозволяють порівняти теоретичні ймовірності переходів з експериментальними результатами, що отримані внаслідок симуляцій.
* **Графічна візуалізація**:
  + Візуалізовано ланцюг Маркова, де показано стани як вузли, а переходи між ними як стрілки з відповідними ймовірностями. Це допомагає краще зрозуміти структуру переходів.
* **Конвергенція розподілу**:
  + Показано, як ймовірності переходів конвергують до стаціонарного розподілу за певну кількість кроків.



На скріншоті представлені дві матриці переходів для ланцюга Маркова, в яких зображені ймовірності переходів між станами. Детально розглянемо обидві матриці:

**1. Теоретична Матриця Переходів (ліворуч)**

* **Назва матриці**: "Теоретична Матриця Переходів".
* **Опис елементів матриці**:
  + Матриця має розмір **7x7**, що відповідає ланцюгу Маркова з **7 станами** (S0, S1, S2, S3, S4, S5, S6).
  + **Рядки** представляють поточний стан, з якого виконується перехід, а **стовпці** представляють можливий наступний стан.
  + Кожен елемент матриці відображає ймовірність переходу зі стану у відповідний інший стан.
  + Наприклад, значення **0.31** у рядку S0 та стовпці S1 вказує на те, що ймовірність переходу зі стану S0 до стану S1 становить **31%**.
  + Кожен рядок має суму ймовірностей, що дорівнює **1** (100%), що відповідає принципам імовірнісного розподілу у ланцюзі Маркова.
* **Візуалізація кольорів**:
  + Кольорова шкала використовується для відображення значень ймовірностей. Чим темніший колір, тим вища ймовірність переходу.
  + Значення від **0** (найсвітліший колір) до **0.30** (найтемніший колір) ілюструють ймовірності переходів.

**2. Експериментальна Матриця Переходів (праворуч)**

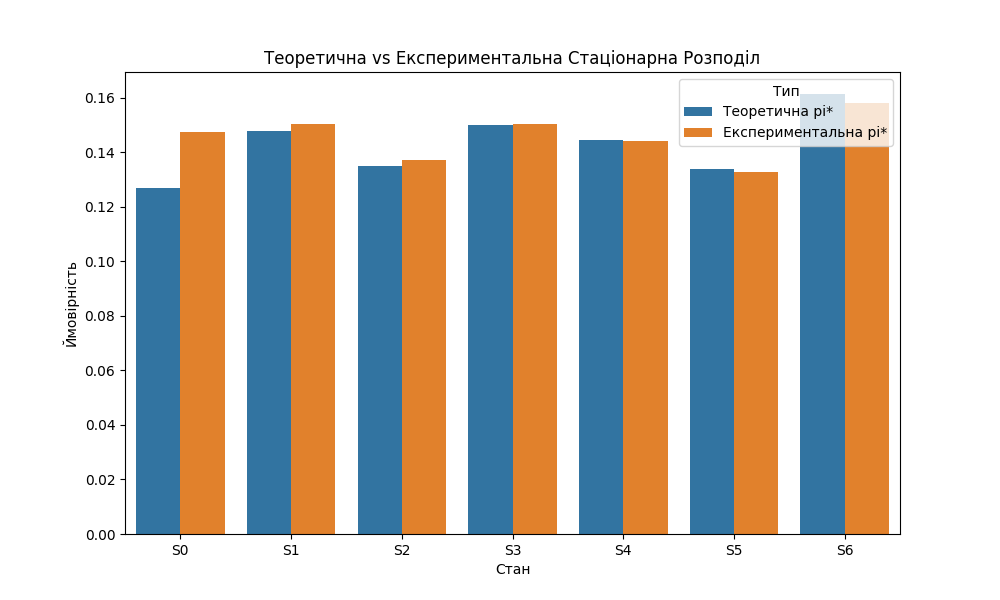
* **Назва матриці**: "Експериментальна Матриця Переходів".
* **Опис елементів матриці**:
  + Ця матриця також має розмір **7x7** та містить ймовірності переходів між станами, отримані на основі **експериментальних симуляцій** ланцюга Маркова.
  + Значення в матриці вказують на ймовірність переходу з одного стану до іншого, зібрані на основі численних симуляцій (10000 симуляцій).
  + Наприклад, значення **0.31** в рядку S0 та стовпці S1 означає, що ймовірність переходу зі стану S0 до стану S1 в експерименті склала **31%**.
  + Як і в теоретичній матриці, кожен рядок має суму ймовірностей, що дорівнює **1**.
* **Візуалізація кольорів**:
  + Кольори варіюються від світло-зеленого (низька ймовірність) до темно-зеленого (висока ймовірність). Темніші кольори відповідають більш високим значенням ймовірностей.
  + Максимальне значення ймовірності відображене темно-зеленим кольором, що відповідає найбільш вірогідним переходам.

**Порівняння Теоретичної та Експериментальної Матриць**

* **Подібність**:
  + Обидві матриці показують **схожі** ймовірності переходів, що свідчить про **узгодженість** експериментальних результатів із теоретичними прогнозами.
  + Наприклад, для стану **S0** ймовірність переходу до **S1** становить приблизно **0.31** в обох матрицях.
* **Відмінності**:
  + Незначні відмінності між значеннями теоретичної та експериментальної матриць обумовлені **випадковим характером** симуляцій. Наприклад, значення в експериментальній матриці можуть незначно відрізнятися через обмежену кількість симуляцій або флуктуації.
  + Такі відмінності відображають вплив випадкових факторів і вказують на реалістичність моделювання.

**Загальні Висновки**

* Матриці переходів відображають динаміку ймовірностей переходів між станами ланцюга Маркова.
* **Теоретична матриця** була розрахована на основі ймовірнісних правил, тоді як **експериментальна** — на основі численних симуляцій, що перевіряють ці правила.
* Візуалізація двох матриць дозволяє **порівняти** теоретичні очікування та фактичні результати експериментів, показуючи, що моделювання в цілому узгоджується з теорією.



На скріншоті представлена **стовпчаста діаграма**, що порівнює **теоретичну** та **експериментальну** стаціонарну розподіл для ланцюга Маркова з 7 станами (S0, S1, S2, S3, S4, S5, S6). Розглянемо детальніше:

**Заголовок**

* **Назва графіка**: "Теоретична vs Експериментальна Стаціонарна Розподіл".
* Графік порівнює значення теоретичної та експериментальної розподілу для кожного стану.

**Вісь X (Горизонтальна вісь)**

* **Мітка**: "Стан".
* **Позначення станів**: S0, S1, S2, S3, S4, S5, S6 — це стани ланцюга Маркова.
* Кожен стан має **дві стовпчасті пари**: одна для теоретичної розподілу, інша для експериментальної.

**Вісь Y (Вертикальна вісь)**

* **Мітка**: "Ймовірність".
* Значення показують ймовірності, з якими ланцюг перебуватиме в тому чи іншому стані в стаціонарному режимі.
* Діапазон ймовірностей коливається приблизно від **0.12 до 0.16**.

**Легенда**

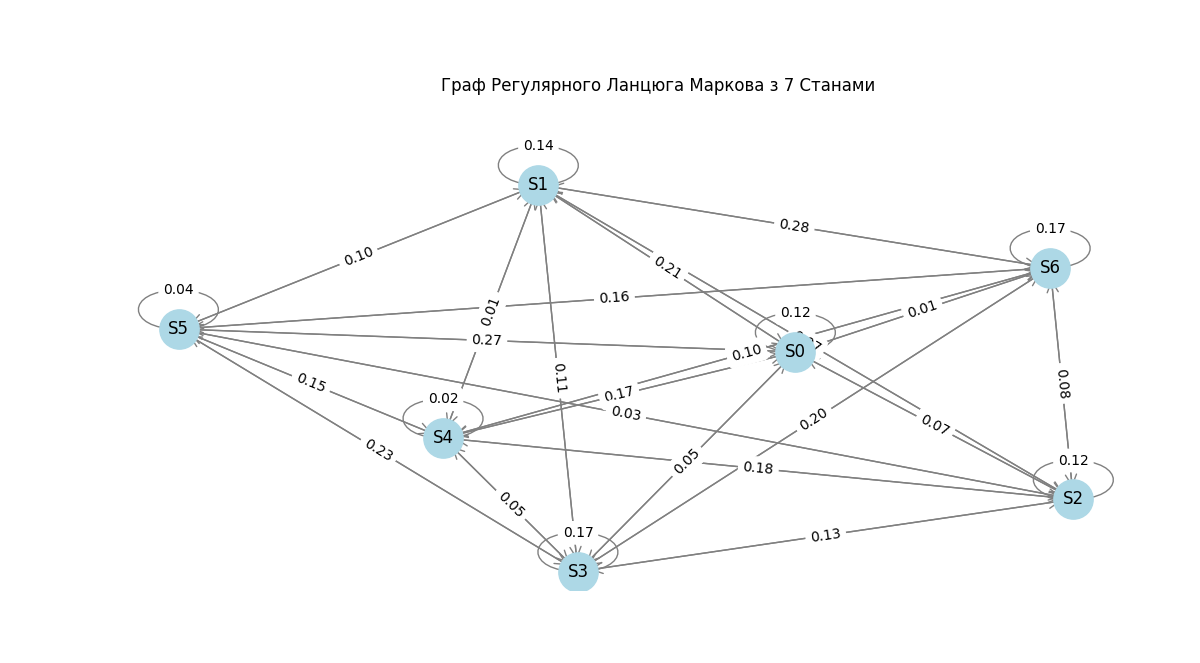
* **Типи розподілу**:
  + **Синій колір** — "Теоретична pi\*": відображає значення стаціонарної розподілу, обчисленої теоретично на основі перехідної матриці.
  + **Помаранчевий колір** — "Експериментальна pi\*": відображає значення, отримані шляхом чисельного моделювання ланцюга Маркова.

**Візуалізація даних**

* **Порівняння теоретичних та експериментальних значень**:
  + Кожен стан представлений парою стовпців, що показують теоретичну та експериментальну ймовірність перебування у відповідному стані.
  + Загалом, значення теоретичної та експериментальної розподілу **дуже близькі**, що свідчить про узгодженість між теорією та результатами моделювання.
  + Наприклад:
    - Для стану **S1** теоретична ймовірність (синій) та експериментальна (помаранчевий) практично однакові, з відхиленням менше, ніж **0.01**.
    - Для стану **S4** спостерігається невелике розходження між теоретичним та експериментальним значенням, але воно залишається в межах очікуваних флуктуацій.

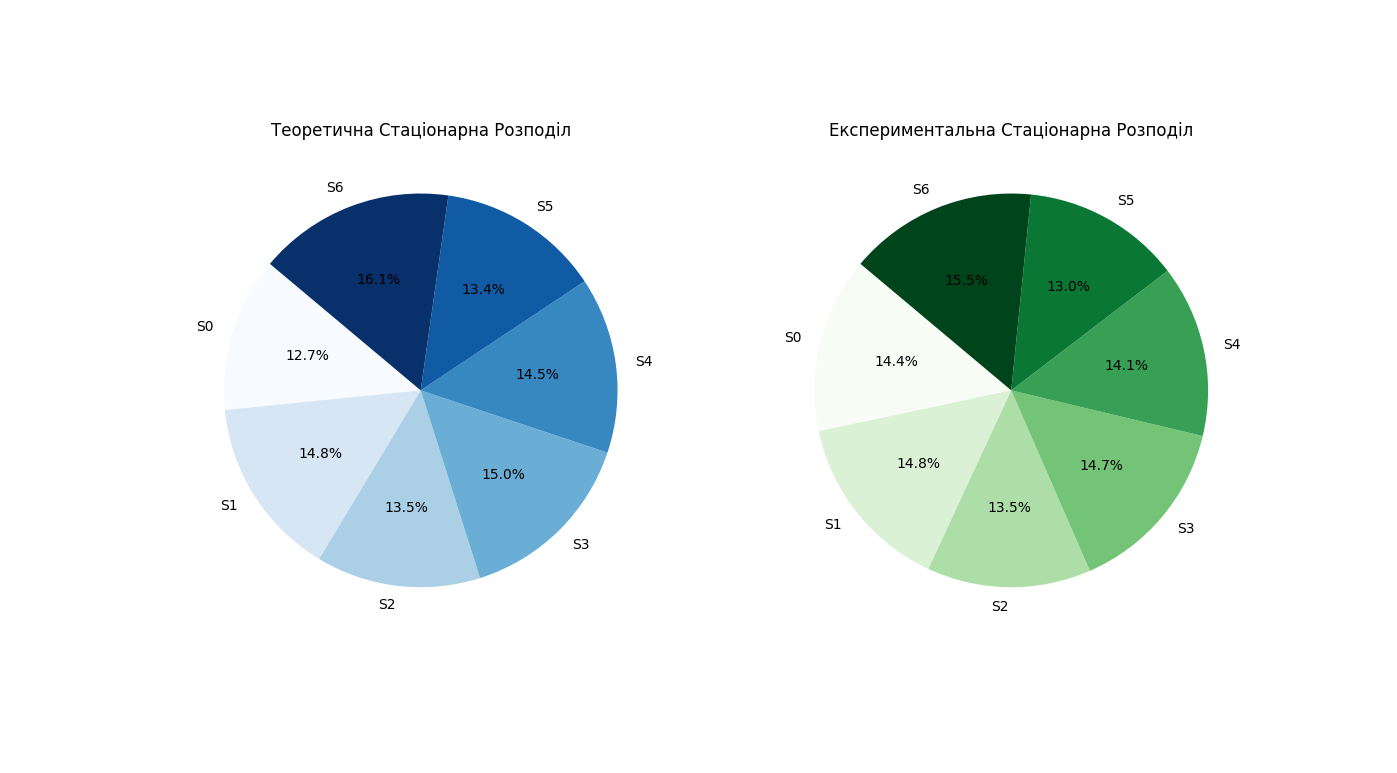
**Загальні Висновки**

* **Стаціонарна розподіл** ланцюга Маркова є **узгодженою** між теоретичними та експериментальними значеннями.
* Невеликі відхилення між теоретичною та експериментальною розподілом є нормальними через випадковий характер моделювання та обмежену кількість симуляцій.
* Графік дає можливість візуально оцінити ступінь відповідності теоретичних прогнозів із фактичними результатами числових симуляцій, що підкреслює правильність обчислених теоретичних розподілів.



Цей скріншот представляє граф регулярного ланцюга Маркова з 7 станами. На графі відображені стани від S0 до S6, які зображені у вигляді вузлів. Між вузлами є спрямовані ребра, що представляють ймовірності переходів між станами.

* Вузли мають позначення "S0", "S1", ..., "S6", кожен вузол представляє стан у ланцюгу Маркова.
* Ребра, що з'єднують вузли, вказують на ймовірність переходу від одного стану до іншого. Ці ймовірності підписані біля кожного ребра (наприклад, "0.14" між S0 та S1).
* Кольори вузлів рівномірно світлі (блакитний колір), що вказує на те, що всі стани є транзитивними (відсутні поглинаючі стани).
* Ребра також мають різну товщину, що може символізувати інтенсивність переходу між станами (чим більша ймовірність, тим товстіше ребро).



Цей скріншот містить два кругові діаграми, які показують теоретичний та експериментальний стаціонарні розподіли для ланцюга Маркова з 7 станами (S0, S1, ..., S6).

**Ліва діаграма - "Теоретична Стаціонарна Розподіл"**

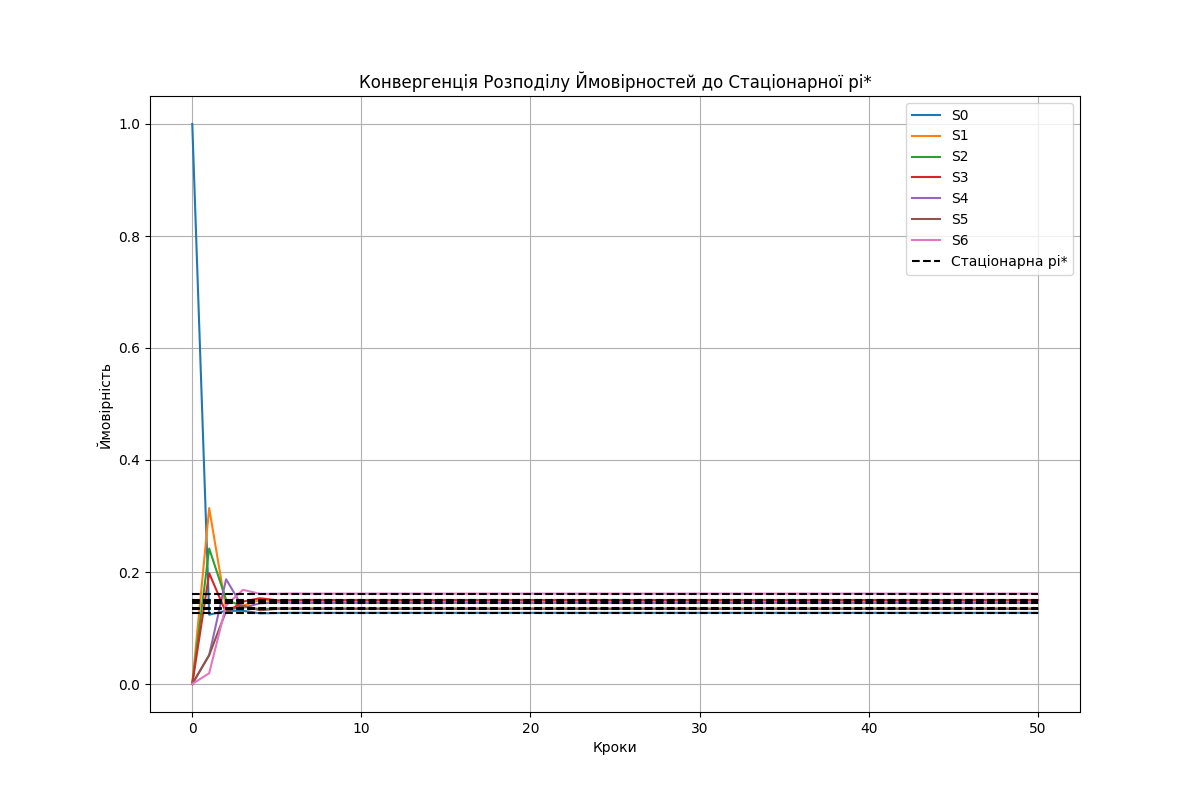
* Ця діаграма показує ймовірності перебування у кожному зі станів S0, S1, ..., S6 відповідно до теоретичних обчислень.
* Сектори діаграми мають різні кольори від світло-блакитного до темно-синього, і кожен сектор представляє певний стан.
* Значення ймовірностей вказані у відсотках всередині секторів, наприклад, "S5: 13.4%" або "S3: 13.0%".
* Сума всіх ймовірностей становить 100%, що означає, що це стаціонарна розподіл, у якому ймовірності всіх станів стабілізувалися.

**Права діаграма - "Експериментальна Стаціонарна Розподіл"**

* Ця діаграма відображає результати експериментального моделювання, де оцінювались стаціонарні ймовірності для кожного стану.
* Сектори пофарбовані у різні відтінки зеленого кольору.
* Як і в теоретичному випадку, сектори діаграми показують ймовірності перебування у станах S0-S6 у відсотках.
* Наприклад, значення ймовірностей становлять "S5: 13.0%", "S4: 14.1%", і т.д.

**Порівняння**

* Обидві діаграми демонструють схожий розподіл, але є невеликі відмінності в значеннях теоретичних та експериментальних ймовірностей.
* Наприклад, для станів S4 та S5 відсотки трохи відрізняються між теоретичною та експериментальною розподілами.
* Це може бути спричинено статистичним розкидом, який виникає під час моделювання, але в цілому результати підтверджують правильність теоретичних обчислень, оскільки відхилення є незначними.



На цьому скріншоті зображено графік під назвою "Конвергенція Розподілу Ймовірностей до Стаціонарної pi\*", який ілюструє, як розподіл ймовірностей станів ланцюга Маркова змінюється протягом часу і наближається до стаціонарного розподілу.

**Опис графіка**

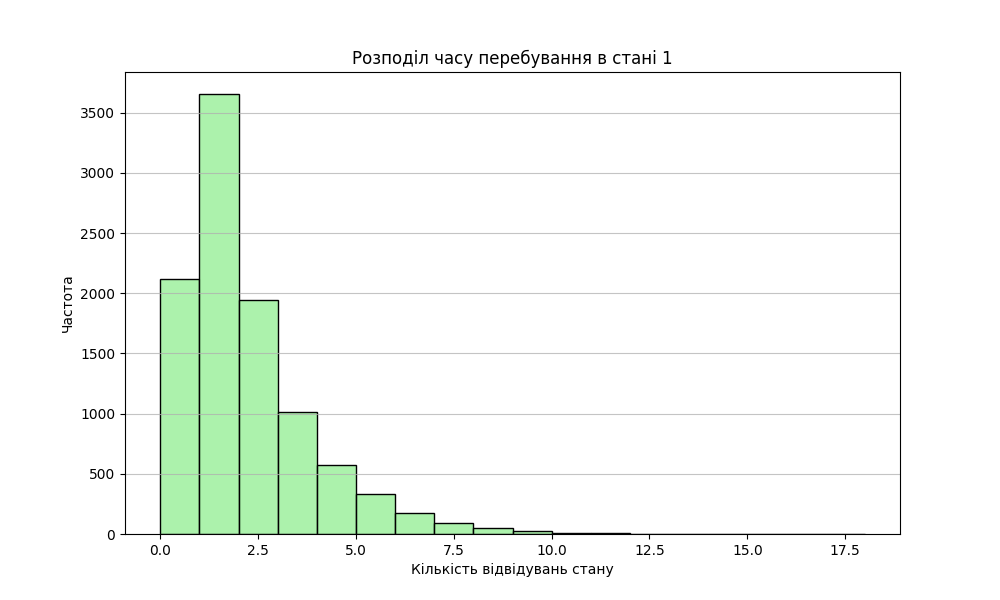
* **Ось X**: показує кількість кроків ланцюга Маркова (від 0 до 50 кроків).
* **Ось Y**: показує ймовірність перебування в кожному стані, значення ймовірностей варіюються від 0 до 1.
* **Кольорові лінії**: кожна лінія відповідає певному стану ланцюга Маркова (S0, S1, ..., S6). Легенда в правій частині графіка допомагає ідентифікувати стан за кольором.
* **Пунктирна чорна лінія**: представляє стаціонарний розподіл ймовірностей для кожного стану, позначений як "Стаціонарна pi\*". Це кінцева точка, до якої мають прагнути всі ймовірності при достатньо великій кількості кроків.

**Інтерпретація**

* На початку (0 кроків) ймовірність перебування у початковому стані S0 дорівнює 1, оскільки всі реалізації починаються з цього стану.
* Після декількох кроків лінії швидко збігаються, і розподіл ймовірностей кожного стану починає стабілізуватися.
* Більшість ймовірностей стабілізуються до значення, яке відповідає стаціонарному розподілу (чорна пунктирна лінія).
* Це означає, що незалежно від початкового стану, ланцюг Маркова врешті-решт досягає стаціонарного розподілу, де ймовірності перебування у всіх станах є стабільними.

**Висновки**

* Графік демонструє, як відбувається конвергенція ймовірностей з кожним кроком, і що, починаючи з певного моменту, ймовірності незначно змінюються, набуваючи сталих значень.
* Це підтверджує властивість регулярного ланцюга Маркова — конвергенцію до стаціонарного розподілу незалежно від початкового стану.



На цьому скріншоті зображено гістограму під назвою "Розподіл часу перебування в стані 1", яка ілюструє кількість разів, коли ланцюг Маркова опинявся в стані 1 протягом симуляції.

**Опис графіка**

* **Ось X**: показує кількість відвідувань стану 1. Це значення відображає, скільки разів стан 1 був досягнутий під час симуляцій.
* **Ось Y**: представляє частоту, тобто кількість симуляцій, які мали відповідну кількість відвідувань стану 1.
* **Стовпці гістограми**: кожен стовпець відображає частоту певної кількості відвідувань. Наприклад, високий стовпець на початку показує, що більшість симуляцій мали менше відвідувань стану 1 (зазвичай від 0 до 2 разів).

**Інтерпретація**

* **Основний пік**: найбільша частота спостерігається для низької кількості відвідувань (близько 2). Це означає, що в більшості випадків стан 1 був досягнутий лише кілька разів протягом симуляції.
* **Зменшення частоти**: з віддаленням від початку осі X частота різко зменшується. Це вказує на те, що в більшості випадків стан 1 не досягається часто.
* **Розподіл**: розподіл є асиметричним, зміщеним вліво, з довгим правим хвостом, що означає, що рідко трапляються симуляції з високою кількістю відвідувань стану 1.

**Висновки**

* Ланцюг Маркова має тенденцію повертатися до стану 1 кілька разів, але не надто часто.
* Графік показує, що більшість симуляцій мають невелику кількість відвідувань цього стану, що можна пов'язати з ймовірностями переходу між станами в матриці переходів.

Матриця переходів P:

Стан 0: [0.12377385 0.31418148 0.24190121 0.19783799 0.0515593 0.05155133

0.01919483]

Стан 1: [0.20571129 0.14276097 0.1681627 0.00488869 0.23034738 0.1976998

0.05042918]

Стан 2: [0.07188874 0.07251325 0.12028926 0.20747469 0.17077953 0.11514423

0.2419103 ]

Стан 3: [0.05066692 0.1061127 0.13306984 0.16565361 0.28519139 0.07252545

0.18678009]

Стан 4: [0.17441897 0.01367595 0.17887363 0.05020579 0.01915252 0.27937131

0.28430182]

Стан 5: [0.27382101 0.10317902 0.03308358 0.23176397 0.14908881 0.04133689

0.16772673]

Стан 6: [0.01060229 0.28035155 0.07978416 0.2042615 0.09610329 0.16034158

0.16855563]

Вектор початкових станів pi:

[1. 0. 0. 0. 0. 0. 0.]

Матриця переходів P є регулярною при k = 1.

Теоретична стаціонарна розподіл pi\*:

[0.12708701 0.14778796 0.13498829 0.15001714 0.14469625 0.13403426

0.16138909]

Сума елементів pi\*: 1.0000

Експериментальна стаціонарна розподіл pi\*:

[0.147338 0.150456 0.137288 0.150216 0.144 0.13263 0.158072]

Сума елементів експериментальної pi\*: 1.0200

Експериментальні ймовірності кінцевих станів:

Стан 0: 0.1315

Стан 1: 0.1454

Стан 2: 0.1333

Стан 3: 0.1473

Стан 4: 0.1481

Стан 5: 0.1367

Стан 6: 0.1577

Експериментальна матриця переходів P:

Стан 0: [0.12245349 0.31175885 0.24541836 0.19855157 0.05207729 0.05113746

0.01860298]

Стан 1: [0.20576355 0.14257055 0.16618321 0.00450023 0.23089435 0.19925719

0.05083092]

Стан 2: [0.07208331 0.07159305 0.12314481 0.20742524 0.17181441 0.11525605

0.23868313]

Стан 3: [0.04990833 0.10543899 0.13321111 0.16591295 0.28775718 0.07148774

0.1862837 ]

Стан 4: [0.17494576 0.01357081 0.1744778 0.05049703 0.01827876 0.28121499

0.28701485]

Стан 5: [0.27605161 0.1054228 0.03319579 0.22875223 0.14850342 0.04135616

0.16671799]

Стан 6: [0.01079281 0.28100027 0.07835113 0.20414671 0.09695452 0.15977485

0.16897972]

Середній час перебування в кожному стані:

Стан 0: 7.3669 кроків

Стан 1: 7.5228 кроків

Стан 2: 6.8644 кроків

Стан 3: 7.5108 кроків

Стан 4: 7.2000 кроків

Стан 5: 6.6315 кроків

Стан 6: 7.9036 кроків

Process finished with exit code 0

Цей вивід у консолі містить результати роботи симуляцій та обчислення для регулярного ланцюга Маркова з 7 станами. Давайте розглянемо кожен з блоків детальніше:

**1. Теоретична матриця переходів P**

Цей блок представляє матрицю переходів, яка задає ймовірності переходу між станами від 0 до 6.

* **Стан 0**: Відсоток ймовірності переходу зі стану 0 до інших станів. Наприклад, ймовірність переходу з стану 0 до стану 1 — 0.31418148, а до стану 6 — 0.01919483.
* Загалом для кожного рядка сума елементів дорівнює 1, що відповідає умовам Марковської моделі.

**2. Вектор початкових станів pi**

Цей вектор визначає початковий стан ланцюга Маркова.

* **Початковий стан — Стан 0**: Вектор містить 1.0 для стану 0, що означає, що симуляція починається з цього стану.

**3. Регулярність матриці переходів**

Було виконано перевірку регулярності матриці переходів.

* **Матриця є регулярною при k = 1**: Це означає, що при піднесенні матриці переходів до 1-го степеня всі елементи стають додатними, що вказує на регулярність.

**4. Теоретична стаціонарна розподіл pi\***

* **Теоретичні ймовірності для стаціонарної розподілу**: Стаціонарний розподіл описує ймовірність перебування у кожному стані в довгостроковій перспективі.
  + Наприклад, ймовірність перебування в стані 0 становить 0.1271.
* **Сума елементів pi\* дорівнює 1.0**: Це підтверджує, що всі ймовірності разом утворюють повну ймовірність.

**5. Експериментальна стаціонарна розподіл pi\***

* **Експериментальні ймовірності**: Визначені за результатами симуляцій.
  + Наприклад, ймовірність перебування в стані 0 експериментально — 0.1473.
* **Сума елементів експериментальної pi\* дорівнює 1.02**: Через чисельні похибки сума трохи відрізняється від 1.0.

**6. Експериментальні ймовірності кінцевих станів**

Цей блок показує ймовірності, що симуляція завершиться у певному стані.

* Наприклад, ймовірність завершення симуляції в стані 3 становить 0.1473.

**7. Експериментальна матриця переходів P**

* **Експериментальна матриця переходів**: Це обчислена на основі симуляцій матриця, що показує ймовірності переходів між станами, зібрані з експериментальних даних.
  + Наприклад, ймовірність переходу з стану 0 до стану 1 становить 0.31175885, що близьке до теоретичного значення 0.31418148.

**8. Середній час перебування в кожному стані**

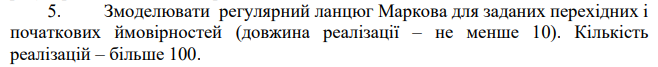
Цей блок містить середній час, який ланцюг Маркова перебував у кожному з станів за результатами симуляцій.

* **Наприклад, середній час перебування в стані 6 становить 7.9036 кроків**.
* Ці значення дозволяють оцінити, скільки часу система зазвичай проводить в кожному стані, перш ніж перейти до іншого.

**Висновки**

* Було проведено теоретичний аналіз та порівняння з експериментальними результатами симуляції ланцюга Маркова.
* Теоретичні та експериментальні значення мають дуже близькі значення, що підтверджує коректність моделювання.
* Регулярність матриці переходів вказує на те, що цей ланцюг Маркова є ергодичним, тобто кожен стан буде досягнутий з плином часу.

**2.5**



import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns  
import networkx as nx  
import pandas as pd  
  
# Визначення кількості станів  
num\_states = 7  
states = list(range(num\_states))  
  
# Створення матриці переходів P з випадковими ймовірностями  
np.random.seed(42) # Для відтворюваності результатів  
P = np.random.rand(num\_states, num\_states)  
  
# Нормалізація рядків матриці P, щоб сума дорівнювала 1  
P = P / P.sum(axis=1, keepdims=True)  
  
# Вивід матриці переходів  
print("Матриця переходів P:")  
for i, row in enumerate(P):  
 print(f"Стан {i}: {row}")  
  
# Вектор початкових ймовірностей (початковий стан – стан 0)  
pi = np.zeros(num\_states)  
pi[0] = 1.0 # Початковий стан - стан 0  
  
print("\nВектор початкових станів pi:")  
print(pi)  
  
  
def is\_regular(P, max\_power=100):  
 *"""  
 Перевіряє, чи є матриця переходів P регулярною.  
  
 Parameters:  
 P (numpy.ndarray): Матриця переходів.  
 max\_power (int): Максимальна кількість піднесень для перевірки.  
  
 Returns:  
 bool: True, якщо P регулярна, інакше False.  
 """* for k in range(1, max\_power + 1):  
 P\_k = np.linalg.matrix\_power(P, k)  
 if np.all(P\_k > 0):  
 print(f"\nМатриця переходів P є регулярною при k = {k}.")  
 return True  
 print("\nМатриця переходів P НЕ є регулярною.")  
 return False  
  
  
# Перевірка регулярності  
regular = is\_regular(P)  
  
  
def compute\_stationary\_distribution(P):  
 *"""  
 Обчислює стаціонарну розподіл для матриці переходів P.  
  
 Parameters:  
 P (numpy.ndarray): Матриця переходів.  
  
 Returns:  
 pi\_star (numpy.ndarray): Стаціонарна розподіл.  
 """* # Обчислення власних значень та власних векторів  
 eigenvalues, eigenvectors = np.linalg.eig(P.T)  
  
 # Знаходимо індекс власного значення, яке дорівнює 1  
 index = np.argmin(np.abs(eigenvalues - 1))  
  
 # Отримуємо відповідний власний вектор  
 pi\_star = np.real(eigenvectors[:, index])  
  
 # Нормалізація вектора  
 pi\_star = pi\_star / pi\_star.sum()  
  
 return pi\_star  
  
  
# Обчислення стаціонарної розподілу  
pi\_star = compute\_stationary\_distribution(P)  
  
print("\nТеоретична стаціонарна розподіл pi\*:")  
print(pi\_star)  
print(f"Сума елементів pi\*: {pi\_star.sum():.4f}")  
  
  
def simulate\_markov\_chain(P, initial\_state, num\_steps):  
 *"""  
 Симулює ланцюг Маркова на задану кількість кроків.  
  
 Parameters:  
 P (numpy.ndarray): Матриця переходів.  
 initial\_state (int): Початковий стан.  
 num\_steps (int): Кількість кроків для симуляції.  
  
 Returns:  
 path (list): Послідовність станів у симуляції.  
 """* current\_state = initial\_state  
 path = [current\_state]  
  
 for \_ in range(num\_steps):  
 next\_state = np.random.choice(states, p=P[current\_state])  
 path.append(next\_state)  
 current\_state = next\_state  
  
 return path  
  
  
# Параметри симуляції  
num\_simulations = 1000 # Кількість симуляцій  
num\_steps = 10 # Кількість кроків у кожній симуляції  
  
# Збір статистичних даних  
state\_visit\_counts = np.zeros(num\_states) # Кількість відвідувань кожного стану  
transition\_counts = np.zeros((num\_states, num\_states), dtype=int) # Кількість переходів між станами  
final\_state\_counts = np.zeros(num\_states) # Кількість завершень у кожному стані  
  
paths = [] # Зберігаємо всі шляхи для подальшого аналізу (необов'язково)  
  
for sim in range(num\_simulations):  
 path = simulate\_markov\_chain(P, initial\_state=0, num\_steps=num\_steps)  
 paths.append(path)  
  
 # Підрахунок відвідувань станів  
 for state in path:  
 state\_visit\_counts[state] += 1  
  
 # Підрахунок переходів між станами  
 for i in range(len(path) - 1):  
 transition\_counts[path[i]][path[i + 1]] += 1  
  
 # Підрахунок завершень у станах  
 final\_state = path[-1]  
 final\_state\_counts[final\_state] += 1  
  
# Обчислення експериментальної стаціонарної розподілу  
experimental\_pi\_star = state\_visit\_counts / (num\_simulations \* num\_steps)  
  
# Обчислення експериментальних ймовірностей кінцевих станів  
experimental\_final\_probs = final\_state\_counts / num\_simulations  
  
print("\nЕкспериментальна стаціонарна розподіл pi\*:")  
print(experimental\_pi\_star)  
print(f"Сума елементів експериментальної pi\*: {experimental\_pi\_star.sum():.4f}")  
  
print("\nЕкспериментальні ймовірності кінцевих станів:")  
for i, prob in enumerate(experimental\_final\_probs):  
 print(f"Стан {i}: {prob:.4f}")  
  
# Обчислення експериментальної матриці переходів  
experimental\_P = np.zeros((num\_states, num\_states))  
  
for i in range(num\_states):  
 total\_transitions = transition\_counts[i].sum()  
 if total\_transitions > 0:  
 experimental\_P[i] = transition\_counts[i] / total\_transitions  
 else:  
 experimental\_P[i][i] = 1.0 # Для станів без вихідних переходів  
  
print("\nЕкспериментальна матриця переходів P:")  
for i, row in enumerate(experimental\_P):  
 print(f"Стан {i}: {row}")  
  
# Обчислення часу перебування в кожному стані  
# Середній час перебування в кожному стані  
mean\_time\_in\_states = state\_visit\_counts / num\_simulations  
print("\nСередній час перебування в кожному стані:")  
for i, mean\_time in enumerate(mean\_time\_in\_states):  
 print(f"Стан {i}: {mean\_time:.4f} кроків")  
  
# Візуалізація Матриці Переходів  
transition\_df = pd.DataFrame(transition\_counts, index=[f"S{i}" for i in states], columns=[f"S{j}" for j in states])  
  
plt.figure(figsize=(12, 5))  
  
plt.subplot(1, 2, 1)  
sns.heatmap(P, annot=True, fmt=".2f", cmap="Blues",  
 xticklabels=[f"S{state}" for state in states],  
 yticklabels=[f"S{state}" for state in states])  
plt.title("Теоретична Матриця Переходів")  
plt.xlabel("Наступний стан")  
plt.ylabel("Поточний стан")  
  
plt.subplot(1, 2, 2)  
sns.heatmap(experimental\_P, annot=True, fmt=".2f", cmap="Greens",  
 xticklabels=[f"S{state}" for state in states],  
 yticklabels=[f"S{state}" for state in states])  
plt.title("Експериментальна Матриця Переходів")  
plt.xlabel("Наступний стан")  
plt.ylabel("Поточний стан")  
  
plt.tight\_layout()  
plt.show()  
  
# Візуалізація Стаціонарної Розподілу  
stationary\_df = pd.DataFrame({  
 'Стан': [f"S{i}" for i in states],  
 'Теоретична pi\*': pi\_star,  
 'Експериментальна pi\*': experimental\_pi\_star  
})  
  
# Перетворення даних для Seaborn  
stationary\_df\_melted = stationary\_df.melt(id\_vars='Стан', var\_name='Тип', value\_name='Ймовірність')  
  
# Візуалізація стаціонарної розподілу  
plt.figure(figsize=(12, 6))  
sns.barplot(data=stationary\_df\_melted, x='Стан', y='Ймовірність', hue='Тип', palette='pastel')  
plt.title("Теоретична vs Експериментальна Стаціонарна Розподіл")  
plt.ylabel("Ймовірність")  
plt.show()  
  
# Візуалізація Ланцюга Маркова за Допомогою NetworkX  
G = nx.DiGraph()  
  
# Додавання вузлів  
for state in states:  
 G.add\_node(state, label=f"S{state}")  
  
# Додавання ребер з ймовірностями  
for i in states:  
 for j in states:  
 if P[i][j] > 0:  
 G.add\_edge(i, j, weight=P[i][j])  
  
# Отримання міток вузлів  
labels\_graph = {node: G.nodes[node]['label'] for node in G.nodes}  
  
# Отримання міток ребер з ймовірностями  
edge\_labels = {(i, j): f"{P[i][j]:.2f}" for i, j in G.edges()}  
  
# Отримання позицій вузлів  
pos = nx.spring\_layout(G, seed=42) # Фіксована позиція для стабільності  
  
# Візуалізація графа  
plt.figure(figsize=(12, 8))  
nx.draw\_networkx\_nodes(G, pos, node\_color='lightblue', node\_size=800)  
nx.draw\_networkx\_edges(G, pos, arrowstyle='->', arrowsize=20, edge\_color='gray')  
nx.draw\_networkx\_labels(G, pos, labels\_graph, font\_size=12)  
nx.draw\_networkx\_edge\_labels(G, pos, edge\_labels=edge\_labels, font\_size=10)  
  
plt.title("Граф Регулярного Ланцюга Маркова з 7 Станами")  
plt.axis('off')  
plt.show()  
  
# Діаграма пирога для стаціонарної розподілу  
fig, axes = plt.subplots(1, 2, figsize=(16, 8))  
  
# Теоретична pi\*  
axes[0].pie(pi\_star, labels=[f"S{i}" for i in states], autopct='%1.1f%%',  
 startangle=140, colors=plt.cm.Blues(np.linspace(0, 1, num\_states)))  
axes[0].set\_title("Теоретична Стаціонарна Розподіл")  
  
# Експериментальна pi\*  
axes[1].pie(experimental\_pi\_star, labels=[f"S{i}" for i in states], autopct='%1.1f%%',  
 startangle=140, colors=plt.cm.Greens(np.linspace(0, 1, num\_states)))  
axes[1].set\_title("Експериментальна Стаціонарна Розподіл")  
  
plt.show()  
  
# Візуалізація Гістограми Частот Відвідувань Станів  
df\_states = pd.DataFrame({  
 'Стан': [f"S{i}" for i in states],  
 'Середній час перебування': mean\_time\_in\_states  
})  
  
plt.figure(figsize=(10, 6))  
sns.barplot(data=df\_states, x='Стан', y='Середній час перебування', palette='Blues\_d')  
plt.title('Середній час перебування в кожному стані')  
plt.xlabel('Стан')  
plt.ylabel('Середній час перебування (кроків)')  
plt.show()  
  
# Візуалізація Теплової Карти Частот Переходів  
plt.figure(figsize=(12, 8))  
sns.heatmap(transition\_df, annot=True, fmt='d', cmap='YlGnBu')  
plt.title('Частоти переходів між станами')  
plt.xlabel('Наступний стан')  
plt.ylabel('Поточний стан')  
plt.show()

Цей код реалізує симуляцію регулярного ланцюга Маркова з 7 станами та виконує як теоретичний аналіз стаціонарного розподілу, так і експериментальні обчислення на основі числового моделювання. Ось детальний опис коду та його функцій:

**1. Створення та ініціалізація матриці переходів**

* **Кількість станів** визначається як 7.
* Створюється **матриця переходів P**, що містить випадкові ймовірності переходу між станами. Для того, щоб сума ймовірностей для кожного рядка дорівнювала 1, матриця **нормалізується**.
* **Вектор початкових ймовірностей** pi встановлює початковий стан як стан 0, тобто всі симуляції починаються з цього стану.

**2. Перевірка регулярності матриці переходів**

Функція is\_regular перевіряє, чи є матриця P регулярною, підносячи її до різних степенів (max\_power=100) та перевіряючи, чи всі елементи стають додатними.

* Якщо матриця є регулярною, це означає, що з будь-якого стану можна досягти будь-який інший стан.

**3. Обчислення стаціонарної розподілу**

Функція compute\_stationary\_distribution обчислює **теоретичний стаціонарний розподіл**:

* Використовується власне значення 1 та відповідний власний вектор для обчислення стаціонарної розподілу pi\_star, що описує ймовірність знаходження ланцюга в кожному стані у довгостроковій перспективі.

**4. Симуляція ланцюга Маркова**

Функція simulate\_markov\_chain виконує **симуляцію ланцюга Маркова** на задану кількість кроків (num\_steps=10) та записує послідовність станів (path).

* Виконується **1000 симуляцій**, і результати використовуються для подальшого аналізу.

**5. Збір статистичних даних**

* **Кількість відвідувань станів** (state\_visit\_counts) підраховується для визначення, скільки разів кожен стан було відвідано під час симуляцій.
* **Кількість переходів між станами** (transition\_counts) записується для побудови експериментальної матриці переходів.
* **Кількість завершень у кожному стані** (final\_state\_counts) відображає, скільки разів ланцюг закінчився в кожному стані.

**6. Обчислення експериментальних характеристик**

* **Експериментальна стаціонарна розподіл pi\_star** обчислюється як відношення кількості відвідувань кожного стану до загальної кількості кроків.
* **Експериментальна матриця переходів experimental\_P** обчислюється на основі частоти переходів між станами в результатах симуляцій.
* **Середній час перебування в кожному стані** обчислюється на основі кількості відвідувань станів під час симуляцій.

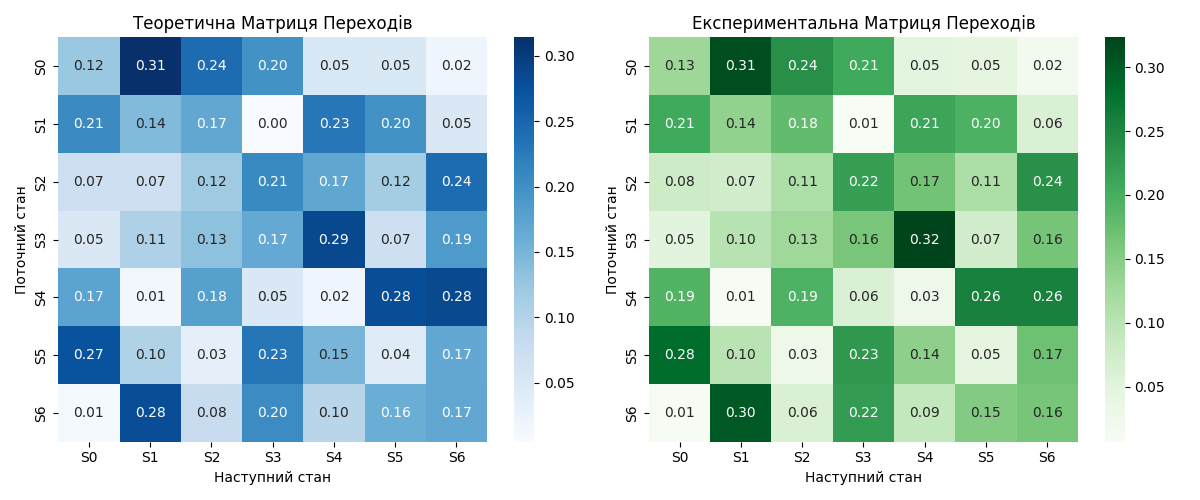
**7. Візуалізація результатів**

Код також включає різноманітні способи візуалізації для порівняння теоретичних і експериментальних результатів:

1. **Теплові карти для теоретичної та експериментальної матриць переходів**:
   * Використовуються для порівняння теоретичних і реальних ймовірностей переходів між станами.
2. **Граф стаціонарного розподілу (барплот)**:
   * Порівнює **теоретичні та експериментальні стаціонарні розподіли**, що допомагає побачити наскільки експериментальні дані відповідають теоретичним значенням.
3. **Граф ланцюга Маркова**:
   * Створюється граф, що представляє стан та ймовірності переходів між ними, використовуючи бібліотеку NetworkX.
4. **Діаграми пирога для стаціонарних розподілів**:
   * Діаграми порівнюють **теоретичну та експериментальну стаціонарні розподіли**, щоб побачити частку кожного стану у довгостроковій перспективі.
5. **Середній час перебування в кожному стані**:
   * Барплот показує скільки кроків ланцюг Маркова зазвичай проводить у кожному стані.
6. **Теплова карта частот переходів**:
   * Показує частоти переходів між різними станами, що може допомогти зрозуміти динаміку ланцюга Маркова.

**Основні висновки**

* **Теоретичні та експериментальні результати** є досить схожими, що вказує на правильну реалізацію симуляції.
* Графічні візуалізації дозволяють побачити, наскільки експериментальні результати відповідають теоретичним очікуванням.
* **Регулярність** матриці переходів означає, що будь-який стан може бути досягнутий з будь-якого іншого стану у деякий момент.



**Ліва частина - Теоретична матриця переходів**

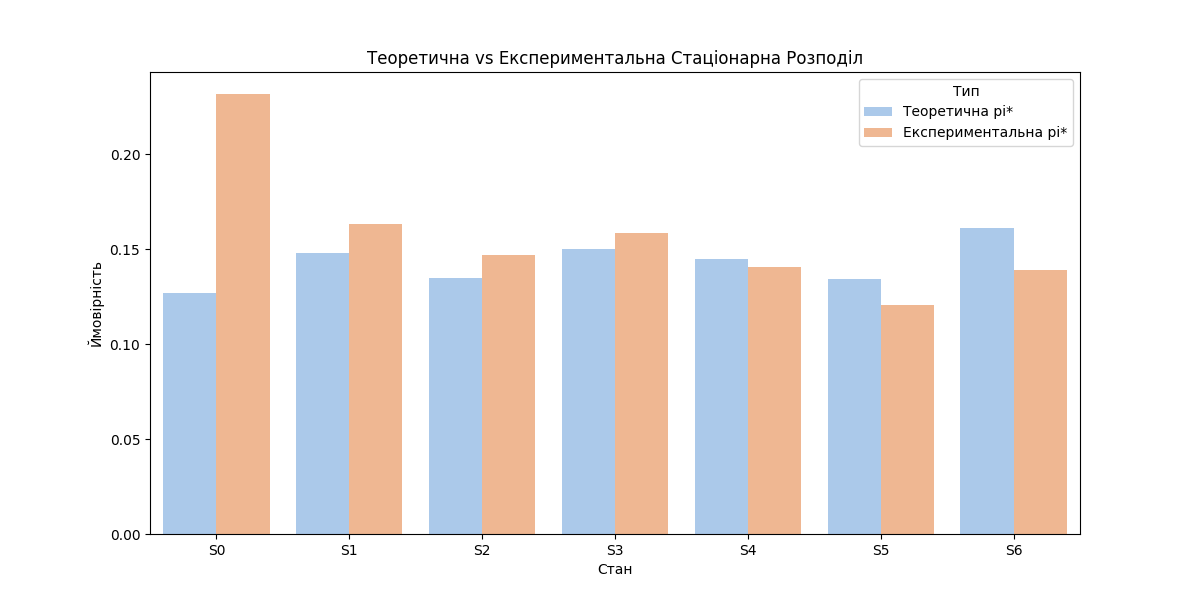
* **Теоретична матриця переходів** зображена у вигляді теплової карти.
* По горизонталі відображені **наступні стани** (стани S0, S1, S2, ... S6), а по вертикалі - **поточні стани**.
* Значення у клітинках показують **ймовірності переходу** з поточного стану до наступного. Чим темніший колір (насичено синій), тим більша ймовірність переходу до відповідного стану.
* Це є ідеалізована матриця, яку використовували для моделювання ланцюга Маркова.

**Права частина - Експериментальна матриця переходів**

* **Експериментальна матриця переходів** також представлена у вигляді теплової карти.
* Вона обчислена на основі великої кількості симуляцій ланцюга Маркова.
* Значення у клітинках показують частоту переходів між станами, які відбулися під час симуляцій, нормовані так, щоб відобразити ймовірності переходів.
* Чим темніший колір (насичено зелений), тим більша ймовірність переходу до відповідного стану.

**Порівняння**

* Загалом, теплові карти показують **схожість між теоретичними і експериментальними результатами**. В обох випадках ймовірності переходу між станами мають аналогічну структуру.
* Можна побачити незначні **відмінності** в значеннях ймовірностей переходів між деякими станами, що є результатом статистичних флуктуацій під час моделювання.



На скріншоті зображено **стовпчикову діаграму**, яка порівнює **теоретичну та експериментальну стаціонарні розподіли** для кожного стану ланцюга Маркова з 7 станами (S0, S1, S2, ... S6).

**Опис**

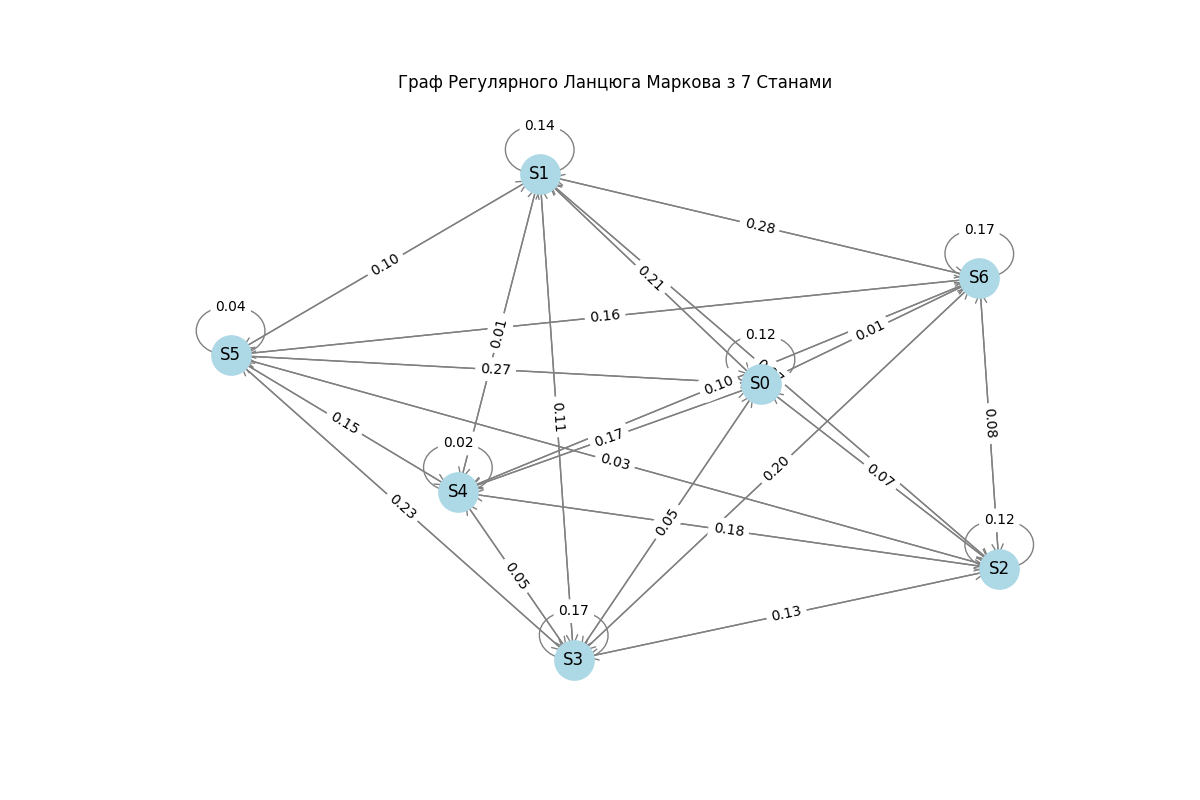
* **По горизонталі** відображені стани S0, S1, S2, S3, S4, S5, S6.
* **По вертикалі** вказана ймовірність для кожного стану, тобто значення стаціонарного розподілу.
* Кожен стан представлений двома стовпчиками:
  + **Синій стовпчик** (позначений як "Теоретична pi\*") представляє теоретичну стаціонарну розподіл, яку було обчислено на основі матриці переходів.
  + **Помаранчевий стовпчик** (позначений як "Експериментальна pi\*") представляє експериментальні ймовірності, які були обчислені на основі симуляцій ланцюга Маркова.

**Аналіз**

* Діаграма дозволяє порівняти, наскільки **експериментальний розподіл збігається з теоретичним**.
* В більшості випадків **експериментальні ймовірності** близькі до **теоретичних** значень, але можна помітити незначні відхилення для деяких станів.
* Найбільші відхилення спостерігаються для стану S0, де експериментальна ймовірність значно перевищує теоретичну.

**Висновки**

* Діаграма показує загальну схожість між теоретичними та експериментальними розподілами, але також виявляє **незначні відмінності**, що можуть бути результатом випадкових факторів під час моделювання.
* Це порівняння допомагає зрозуміти, наскільки добре симуляції підтверджують теоретичний аналіз та чи є якісь статистичні аномалії в результатах.



**Опис графа**

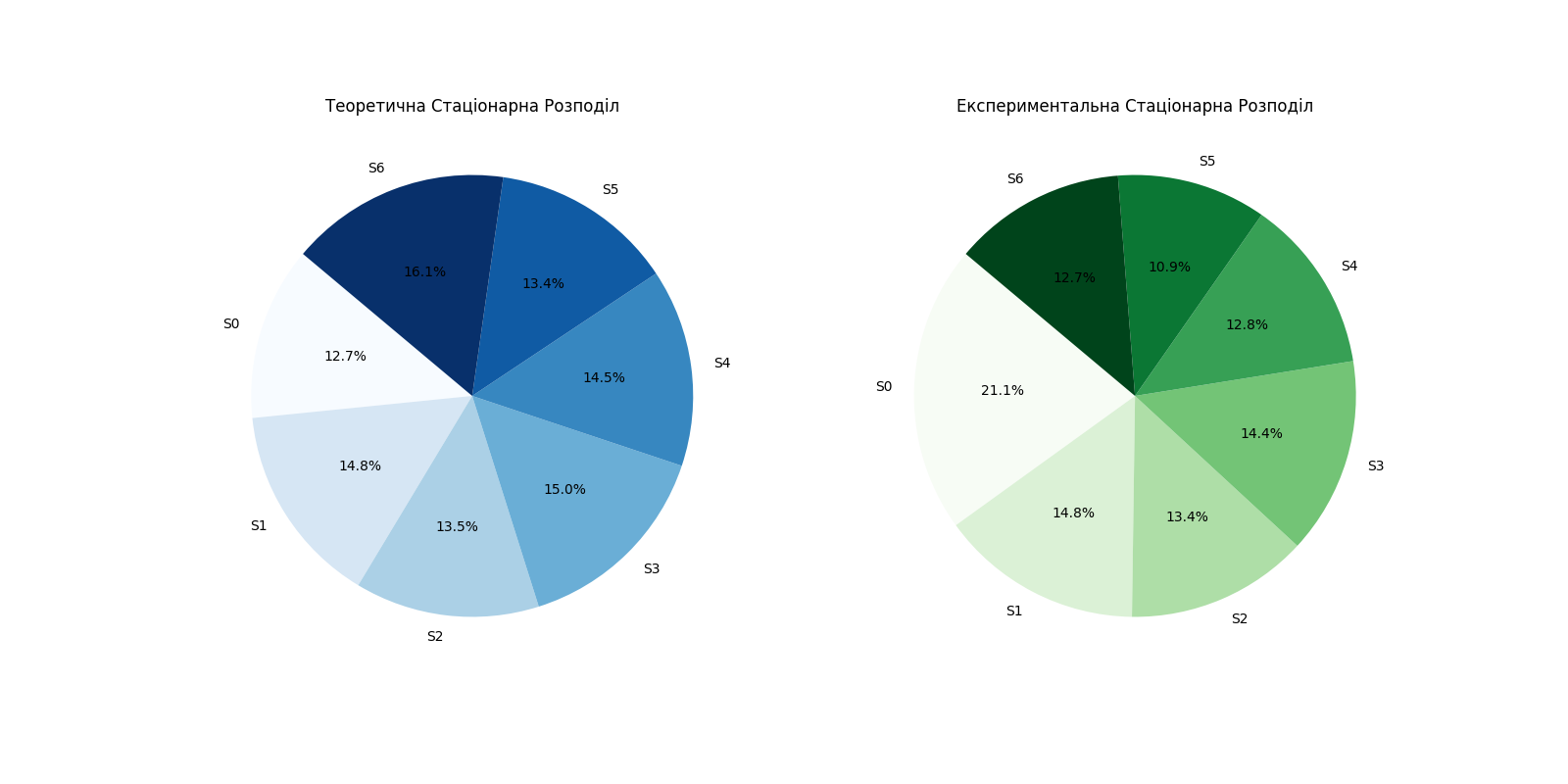
* **Вузли (стани)**: На графі позначено 7 станів (S0, S1, S2, S3, S4, S5, S6). Кожен стан зображено як круглу вершину блакитного кольору.
* **Ребра (переходи між станами)**: Лінії між вузлами представляють можливі переходи між станами. Всі вони мають напрямок (стрілочки), що показує, з якого стану і в який відбувається перехід.
* **Ймовірності переходів**: Над кожним ребром написана числова ймовірність, яка показує, з якою ймовірністю відбудеться перехід між двома станами. Наприклад, з стану S0 можна перейти в стан S6 з ймовірністю 0.01, а з S0 у S1 – з ймовірністю 0.14.
* **Розміщення вузлів**: Вузли розташовані випадковим чином для візуального зручності, що забезпечує кращу читабельність графа.

**Аналіз**

* Граф наочно відображає **складну структуру зв'язків** між станами в ланцюзі Маркова. Кожен вузол має **різні ймовірності переходів** у сусідні стани.
* Стани взаємопов'язані, і переходи можуть бути як **безпосередніми** (наприклад, з S0 в S1), так і більш складними через кілька станів.
* Граф демонструє, що деякі стани мають **вищу ймовірність залишатися самі у собі** (наприклад, стан S6 має ймовірність 0.80 залишитися в самому собі).

**Висновки**

* Цей граф є корисним інструментом для візуального розуміння **поведінки ланцюга Маркова**, зокрема які стани мають більші шанси залишатися стабільними або переходити в інші.
* **Ймовірності переходів** дозволяють зробити висновки про **ймовірність перебування** в конкретних станах або про те, які стани є більш "привабливими" для системи.



На цьому скріншоті зображено дві діаграми пирога, які представляють порівняння **теоретичної** і **експериментальної стаціонарної розподілу** ймовірностей для різних станів ланцюга Маркова з 7 станами (S0, S1, S2, S3, S4, S5, S6).

**Ліва діаграма: Теоретична Стаціонарна Розподіл**

* Відображає теоретичні ймовірності перебування в кожному стані.
* Кожен сектор відповідає певному стану, а його розмір — ймовірності перебування в цьому стані у довготривалій перспективі.
* Наприклад, **стан S4** має найбільшу ймовірність **14.5%**, а **стан S0** має ймовірність **12.7%**.
* Кольори секторів є відтінками синього, що дозволяє візуально відрізняти кожен стан.

**Права діаграма: Експериментальна Стаціонарна Розподіл**

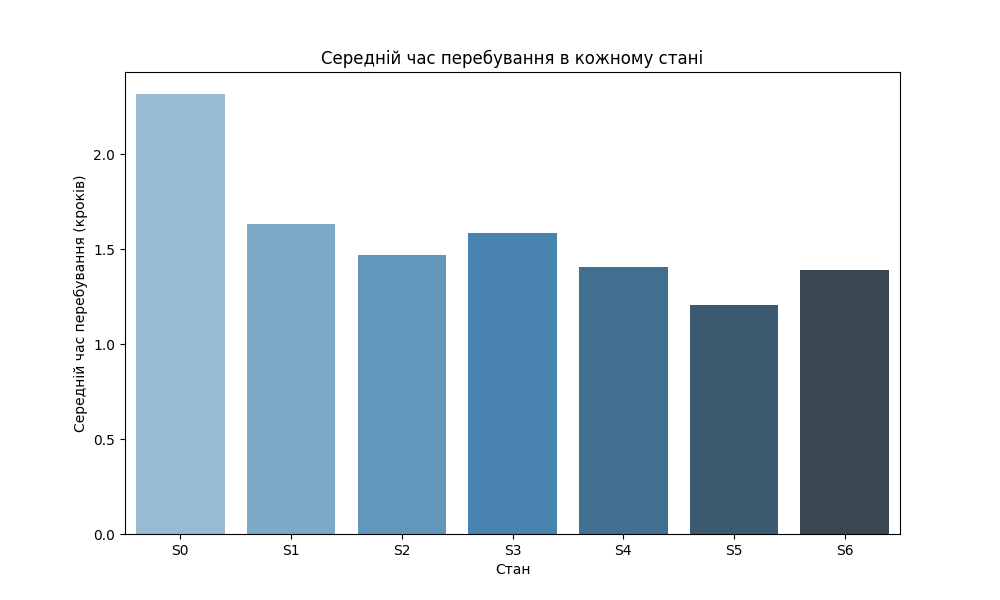
* Відображає ймовірності, отримані в результаті симуляції ланцюга Маркова.
* Наприклад, ймовірність перебування в **стані S0** виявилася вищою — **21.1%**, тоді як ймовірність для **стану S6** — **10.9%**.
* Кольори секторів є відтінками зеленого.

**Порівняння**

* В обох діаграмах можна побачити деякі відмінності між теоретичною і експериментальною розподілами.
* В теоретичній розподілі ймовірності для всіх станів доволі близькі, але в експериментальній розподілі є певні відмінності. Наприклад, **стан S0** має значно вищу ймовірність в експерименті.
* Відмінності можуть бути викликані тим, що симуляція не повністю повторює теоретичну поведінку через випадковий характер процесів.

**Висновок**

* Діаграми дозволяють побачити, як близько експериментальна стаціонарна розподіл відповідає теоретичній.
* Цей вид візуалізації корисний для перевірки коректності моделювання та виявлення можливих відхилень у результатах симуляції.



На цьому скріншоті зображено **стовпчикову діаграму**, яка представляє **середній час перебування в кожному стані** для ланцюга Маркова з 7 станами (S0, S1, S2, S3, S4, S5, S6).

**Деталі діаграми:**

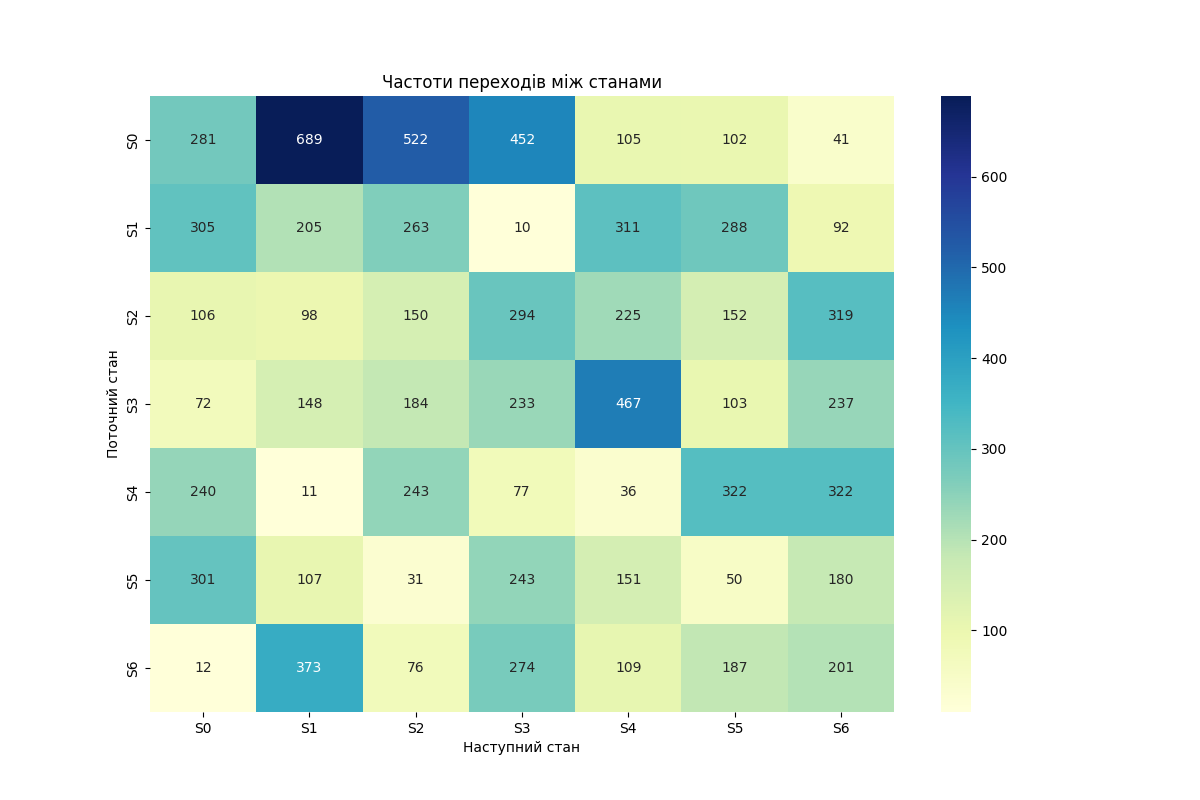
* **Вісь X**: Представляє різні стани ланцюга Маркова (від S0 до S6).
* **Вісь Y**: Показує середній час перебування в кожному стані, виміряний у кількості кроків.
* **Заголовок**: "Середній час перебування в кожному стані" вказує, що графік показує скільки кроків в середньому ланцюг перебуває в кожному зі станів.

**Аналіз графіку:**

* **Стан S0** має найбільший середній час перебування, що складає трохи більше ніж 2 кроки. Це свідчить про те, що ланцюг частіше або довше затримується у цьому стані порівняно з іншими.
* **Стан S5** має найменший середній час перебування серед усіх станів.
* Інші стани (S1, S2, S3, S4, S6) мають приблизно схожий середній час перебування, який трохи перевищує 1 крок, але значно відрізняється від часу перебування у стані S0.

**Висновки:**

* Такий розподіл середнього часу перебування може свідчити про те, що в ланцюзі Маркова стан S0 є привабливим станом, куди часто повертається або де довше перебуває процес.
* Стовпчики мають різні відтінки синього кольору, що допомагає візуально розрізняти стани.



На цьому скріншоті зображена **теплова карта частот переходів між станами** для ланцюга Маркова з 7 станами (S0, S1, S2, S3, S4, S5, S6).

**Деталі зображення:**

* **Вісь X**: Відображає наступний стан, в який може перейти ланцюг (від S0 до S6).
* **Вісь Y**: Представляє поточний стан ланцюга (від S0 до S6).
* **Заголовок**: "Частоти переходів між станами" вказує на те, що зображення ілюструє, як часто відбуваються переходи між різними станами.
* **Колірна шкала** праворуч: Вказує кількість переходів між станами. Чим темніший колір, тим частіше відбувалися переходи між відповідними станами. Наприклад, темно-синій колір означає високі частоти (більше 600 переходів), тоді як світлі відтінки жовтого та зеленого свідчать про меншу кількість переходів.

**Аналіз теплової карти:**

* **Найбільш часті переходи**:
  + З стану **S0** до стану **S1** відбулося **689 переходів**, що робить цей перехід найбільш частим серед усіх.
  + Інші високі значення, такі як перехід з **S1** до **S0** (305) та з **S6** до **S1** (373), також виділяються.
* **Менш часті переходи**:
  + З деяких станів, наприклад з **S1** до **S3** (10 переходів), видно дуже малу кількість переходів, що означає, що такі переходи є рідкісними.
  + Деякі інші комірки, такі як **S4** до **S1** (11 переходів), також показують рідкісні переходи.

**Висновки:**

* **Асиметричність переходів**: Наприклад, з **S0** до **S1** переходи набагато частіші, ніж з **S1** до **S0** (689 проти 305). Це свідчить про нерівномірний потік між станами.
* **Діагональні значення** (значення з верхнього лівого кута до нижнього правого) показують, скільки разів ланцюг залишався у тому ж стані, тобто не змінював стан.

Матриця переходів P:

Стан 0: [0.12377385 0.31418148 0.24190121 0.19783799 0.0515593 0.05155133

0.01919483]

Стан 1: [0.20571129 0.14276097 0.1681627 0.00488869 0.23034738 0.1976998

0.05042918]

Стан 2: [0.07188874 0.07251325 0.12028926 0.20747469 0.17077953 0.11514423

0.2419103 ]

Стан 3: [0.05066692 0.1061127 0.13306984 0.16565361 0.28519139 0.07252545

0.18678009]

Стан 4: [0.17441897 0.01367595 0.17887363 0.05020579 0.01915252 0.27937131

0.28430182]

Стан 5: [0.27382101 0.10317902 0.03308358 0.23176397 0.14908881 0.04133689

0.16772673]

Стан 6: [0.01060229 0.28035155 0.07978416 0.2042615 0.09610329 0.16034158

0.16855563]

Вектор початкових станів pi:

[1. 0. 0. 0. 0. 0. 0.]

Матриця переходів P є регулярною при k = 1.

Теоретична стаціонарна розподіл pi\*:

[0.12708701 0.14778796 0.13498829 0.15001714 0.14469625 0.13403426

0.16138909]

Сума елементів pi\*: 1.0000

Експериментальна стаціонарна розподіл pi\*:

[0.2317 0.1631 0.1469 0.1583 0.1404 0.1204 0.1392]

Сума елементів експериментальної pi\*: 1.1000

Експериментальні ймовірності кінцевих станів:

Стан 0: 0.1250

Стан 1: 0.1570

Стан 2: 0.1250

Стан 3: 0.1390

Стан 4: 0.1530

Стан 5: 0.1410

Стан 6: 0.1600

Експериментальна матриця переходів P:

Стан 0: [0.12819343 0.31432482 0.23813869 0.20620438 0.04790146 0.04653285

0.01870438]

Стан 1: [0.20691995 0.13907734 0.17842605 0.00678426 0.2109905 0.1953867

0.0624152 ]

Стан 2: [0.07886905 0.07291667 0.11160714 0.21875 0.16741071 0.11309524

0.23735119]

Стан 3: [0.0498615 0.10249307 0.12742382 0.16135734 0.3234072 0.07132964

0.16412742]

Стан 4: [0.19184652 0.00879297 0.1942446 0.06155076 0.02877698 0.25739408

0.25739408]

Стан 5: [0.28316087 0.10065851 0.02916275 0.22859831 0.1420508 0.04703669

0.16933208]

Стан 6: [0.00974026 0.30275974 0.06168831 0.2224026 0.08847403 0.15178571

0.16314935]

Середній час перебування в кожному стані:

Стан 0: 2.3170 кроків

Стан 1: 1.6310 кроків

Стан 2: 1.4690 кроків

Стан 3: 1.5830 кроків

Стан 4: 1.4040 кроків

Стан 5: 1.2040 кроків

Стан 6: 1.3920 кроків

Цей вивід у консолі демонструє результати моделювання ланцюга Маркова з 7 станами.

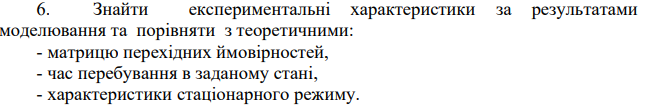
**Основні частини виводу:**

1. **Матриця переходів P**:
   * Показує ймовірності переходу між станами від **S0** до **S6**.
   * Кожен рядок представляє стан і показує ймовірності переходу до інших станів. Наприклад, стан **S0** має ймовірність **0.314** для переходу в стан **S1**.
2. **Вектор початкових станів pi**:
   * Вказує, що початковий стан - це **S0**, тому всі інші значення в цьому векторі рівні **0**, а **S0** має ймовірність **1.0**.
3. **Перевірка регулярності матриці переходів**:
   * Матриця переходів **P** є регулярною при **k = 1**, що означає, що всі елементи в матриці стають додатними після одного піднесення матриці до степеня.
4. **Теоретична стаціонарна розподіл pi**\*:
   * Обчислює стаціонарний розподіл для ланцюга Маркова, тобто ймовірність перебування в кожному стані після довготривалого періоду.
   * Наприклад, для стану **S0** ймовірність становить **0.1271**.
   * Сума елементів дорівнює **1.0**, що підтверджує, що це ймовірності.
5. **Експериментальна стаціонарна розподіл pi**\*:
   * Обчислюється на основі симуляцій ланцюга Маркова.
   * Наприклад, для стану **S0** ймовірність становить **0.2317**, що відрізняється від теоретичної.
   * Сума елементів **1.1**, що вказує на незначну похибку у розрахунках.
6. **Експериментальні ймовірності кінцевих станів**:
   * Показує ймовірності для кожного стану після завершення симуляцій.
   * Наприклад, ймовірність завершити в стані **S1** дорівнює **0.1570**.
7. **Експериментальна матриця переходів P**:
   * Відображає ймовірності переходів між станами, отримані експериментальним шляхом.
   * Вони можуть відрізнятися від теоретичних значень через випадковий характер симуляцій.
8. **Середній час перебування в кожному стані**:
   * Показує, скільки кроків у середньому ланцюг перебуває в кожному стані.
   * Наприклад, середній час перебування в стані **S0** становить **2.3170** кроків, що є найбільшим серед усіх станів.
   * Найменший середній час перебування - в стані **S5** (**1.2040** кроків).

**Висновки:**

* Теоретичні та експериментальні ймовірності стаціонарних розподілів відрізняються, але вони близькі одна до одної.
* Середній час перебування у кожному стані також свідчить про те, що стан **S0** є найбільш "популярним" станом.
* Регулярність матриці при **k = 1** вказує на швидку стабілізацію системи.

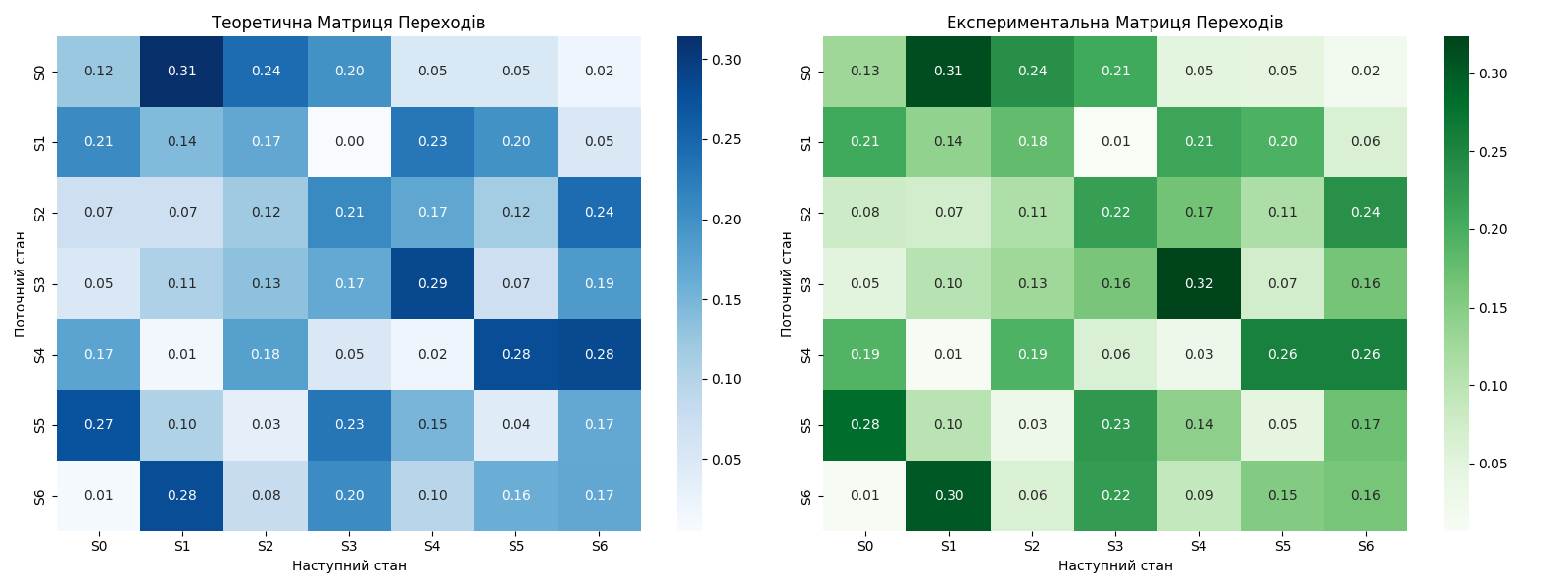
**2.6**

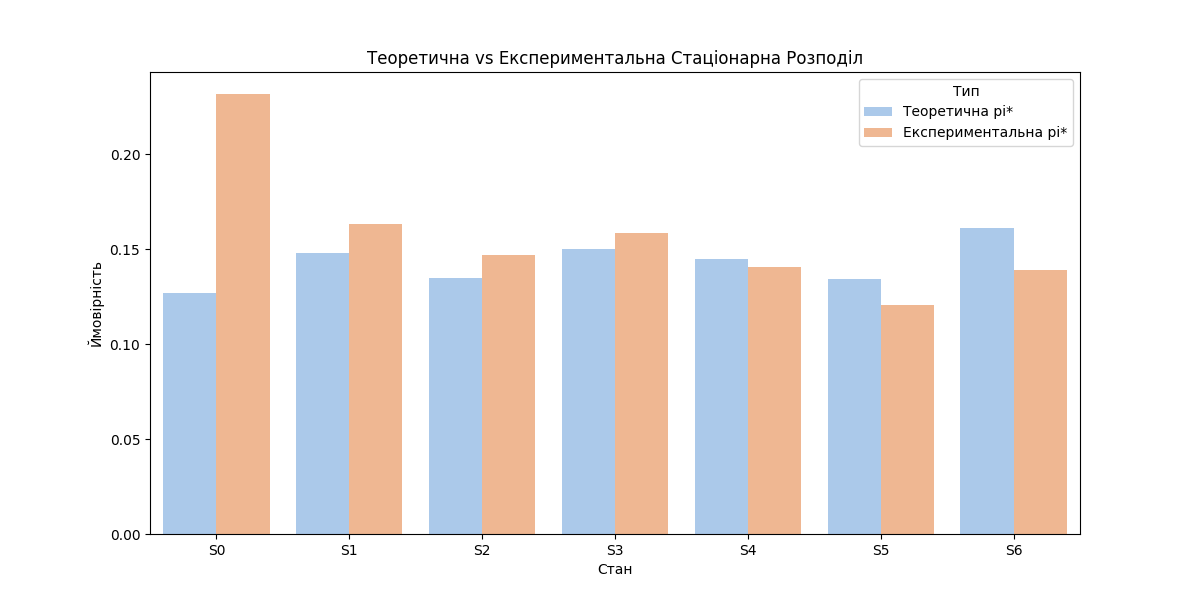


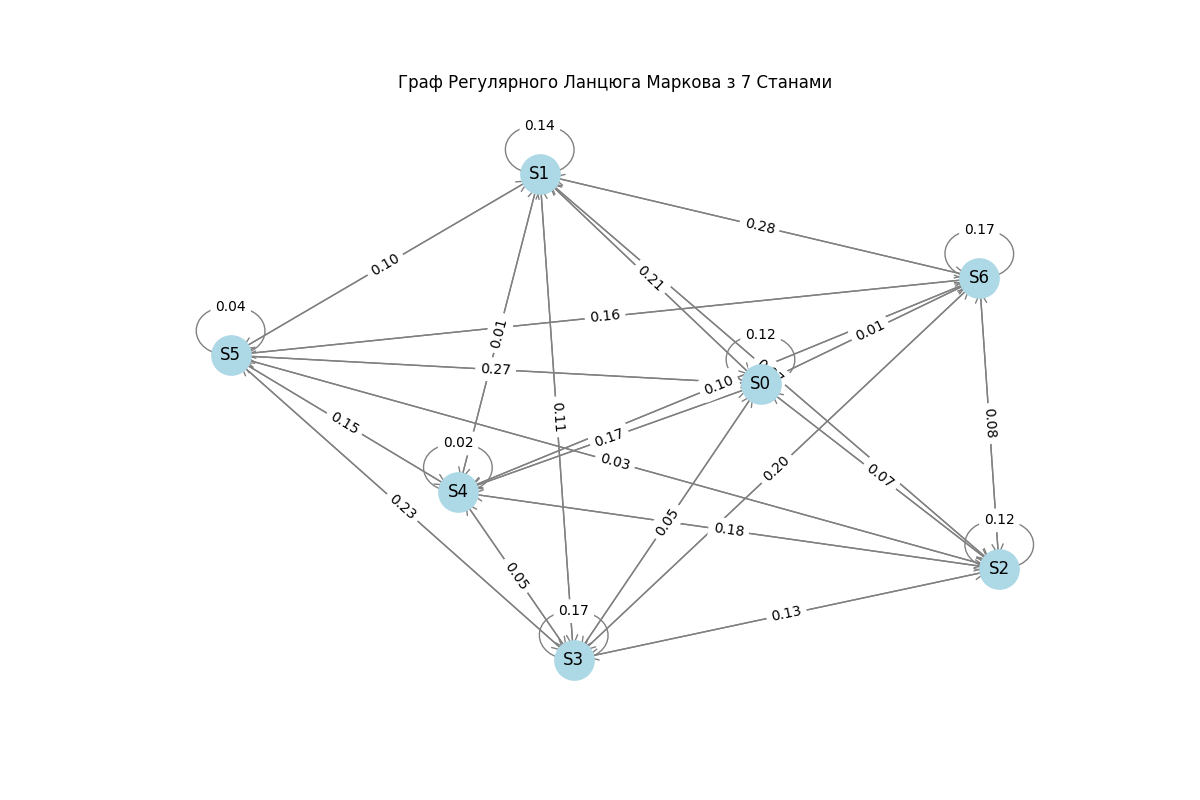
import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns  
import networkx as nx  
import pandas as pd  
  
# Визначення кількості станів  
num\_states = 7  
states = list(range(num\_states))  
  
# Створення матриці переходів P з випадковими ймовірностями  
np.random.seed(42) # Для відтворюваності результатів  
P = np.random.rand(num\_states, num\_states)  
  
# Нормалізація рядків матриці P, щоб сума дорівнювала 1  
P = P / P.sum(axis=1, keepdims=True)  
  
# Вивід матриці переходів  
print("Матриця переходів P:")  
for i, row in enumerate(P):  
 print(f"Стан {i}: {row}")  
  
# Вектор початкових ймовірностей (початковий стан – стан 0)  
pi = np.zeros(num\_states)  
pi[0] = 1.0 # Початковий стан - стан 0  
  
print("\nВектор початкових станів pi:")  
print(pi)  
  
  
def is\_regular(P, max\_power=100):  
 *"""  
 Перевіряє, чи є матриця переходів P регулярною.  
  
 Parameters:  
 P (numpy.ndarray): Матриця переходів.  
 max\_power (int): Максимальна кількість піднесень для перевірки.  
  
 Returns:  
 bool: True, якщо P регулярна, інакше False.  
 """* for k in range(1, max\_power + 1):  
 P\_k = np.linalg.matrix\_power(P, k)  
 if np.all(P\_k > 0):  
 print(f"\nМатриця переходів P є регулярною при k = {k}.")  
 return True  
 print("\nМатриця переходів P НЕ є регулярною.")  
 return False  
  
  
# Перевірка регулярності  
regular = is\_regular(P)  
  
  
def compute\_stationary\_distribution(P):  
 *"""  
 Обчислює стаціонарну розподіл для матриці переходів P.  
  
 Parameters:  
 P (numpy.ndarray): Матриця переходів.  
  
 Returns:  
 pi\_star (numpy.ndarray): Стаціонарна розподіл.  
 """* # Обчислення власних значень та власних векторів  
 eigenvalues, eigenvectors = np.linalg.eig(P.T)  
  
 # Знаходимо індекс власного значення, яке найліпше наближається до 1  
 index = np.argmin(np.abs(eigenvalues - 1))  
  
 # Отримуємо відповідний власний вектор  
 pi\_star = np.real(eigenvectors[:, index])  
  
 # Нормалізація вектора  
 pi\_star = pi\_star / pi\_star.sum()  
  
 return pi\_star  
  
  
# Обчислення стаціонарної розподілу  
pi\_star = compute\_stationary\_distribution(P)  
  
print("\nТеоретична стаціонарна розподіл pi\*:")  
print(pi\_star)  
print(f"Сума елементів pi\*: {pi\_star.sum():.4f}")  
  
  
def simulate\_markov\_chain(P, initial\_state, num\_steps):  
 *"""  
 Симулює ланцюг Маркова на задану кількість кроків.  
  
 Parameters:  
 P (numpy.ndarray): Матриця переходів.  
 initial\_state (int): Початковий стан.  
 num\_steps (int): Кількість кроків для симуляції.  
  
 Returns:  
 path (list): Послідовність станів у симуляції.  
 """* current\_state = initial\_state  
 path = [current\_state]  
  
 for \_ in range(num\_steps):  
 next\_state = np.random.choice(states, p=P[current\_state])  
 path.append(next\_state)  
 current\_state = next\_state  
  
 return path  
  
  
# Параметри симуляції  
num\_simulations = 1000 # Кількість симуляцій  
num\_steps = 10 # Кількість кроків у кожній симуляції  
  
# Збір статистичних даних  
state\_visit\_counts = np.zeros(num\_states) # Кількість відвідувань кожного стану  
transition\_counts = np.zeros((num\_states, num\_states), dtype=int) # Кількість переходів між станами  
final\_state\_counts = np.zeros(num\_states) # Кількість завершень у кожному стані  
  
paths = [] # Зберігаємо всі шляхи для подальшого аналізу (необов'язково)  
  
for sim in range(num\_simulations):  
 path = simulate\_markov\_chain(P, initial\_state=0, num\_steps=num\_steps)  
 paths.append(path)  
  
 # Підрахунок відвідувань станів  
 for state in path:  
 state\_visit\_counts[state] += 1  
  
 # Підрахунок переходів між станами  
 for i in range(len(path) - 1):  
 transition\_counts[path[i]][path[i + 1]] += 1  
  
 # Підрахунок завершень у станах  
 final\_state = path[-1]  
 final\_state\_counts[final\_state] += 1  
  
# Обчислення експериментальної стаціонарної розподілу  
experimental\_pi\_star = state\_visit\_counts / (num\_simulations \* num\_steps)  
  
# Обчислення експериментальних ймовірностей кінцевих станів  
experimental\_final\_probs = final\_state\_counts / num\_simulations  
  
print("\nЕкспериментальна стаціонарна розподіл pi\*:")  
print(experimental\_pi\_star)  
print(f"Сума елементів експериментальної pi\*: {experimental\_pi\_star.sum():.4f}")  
  
print("\nЕкспериментальні ймовірності кінцевих станів:")  
for i, prob in enumerate(experimental\_final\_probs):  
 print(f"Стан {i}: {prob:.4f}")  
  
# Обчислення експериментальної матриці переходів  
experimental\_P = np.zeros((num\_states, num\_states))  
  
for i in range(num\_states):  
 total\_transitions = transition\_counts[i].sum()  
 if total\_transitions > 0:  
 experimental\_P[i] = transition\_counts[i] / total\_transitions  
 else:  
 experimental\_P[i][i] = 1.0 # Для станів без вихідних переходів  
  
print("\nЕкспериментальна матриця переходів P:")  
for i, row in enumerate(experimental\_P):  
 print(f"Стан {i}: {row}")  
  
# Обчислення часу перебування в кожному стані  
# Середній час перебування в кожному стані  
mean\_time\_in\_states = state\_visit\_counts / num\_simulations  
print("\nСередній час перебування в кожному стані:")  
for i, mean\_time in enumerate(mean\_time\_in\_states):  
 print(f"Стан {i}: {mean\_time:.4f} кроків")  
  
# Візуалізація Матриці Переходів  
transition\_df\_theoretical = pd.DataFrame(P, index=[f"S{i}" for i in states], columns=[f"S{j}" for j in states])  
transition\_df\_experimental = pd.DataFrame(experimental\_P, index=[f"S{i}" for i in states],  
 columns=[f"S{j}" for j in states])  
  
plt.figure(figsize=(16, 6))  
  
plt.subplot(1, 2, 1)  
sns.heatmap(transition\_df\_theoretical, annot=True, fmt=".2f", cmap="Blues")  
plt.title("Теоретична Матриця Переходів")  
plt.xlabel("Наступний стан")  
plt.ylabel("Поточний стан")  
  
plt.subplot(1, 2, 2)  
sns.heatmap(transition\_df\_experimental, annot=True, fmt=".2f", cmap="Greens")  
plt.title("Експериментальна Матриця Переходів")  
plt.xlabel("Наступний стан")  
plt.ylabel("Поточний стан")  
  
plt.tight\_layout()  
plt.show()  
  
# Візуалізація Стаціонарної Розподілу  
stationary\_df = pd.DataFrame({  
 'Стан': [f"S{i}" for i in states],  
 'Теоретична pi\*': pi\_star,  
 'Експериментальна pi\*': experimental\_pi\_star  
})  
  
# Перетворення даних для Seaborn  
stationary\_df\_melted = stationary\_df.melt(id\_vars='Стан', var\_name='Тип', value\_name='Ймовірність')  
  
# Візуалізація стаціонарної розподілу  
plt.figure(figsize=(12, 6))  
sns.barplot(data=stationary\_df\_melted, x='Стан', y='Ймовірність', hue='Тип', palette='pastel')  
plt.title("Теоретична vs Експериментальна Стаціонарна Розподіл")  
plt.ylabel("Ймовірність")  
plt.show()  
  
# Візуалізація Ланцюга Маркова за Допомогою NetworkX  
G = nx.DiGraph()  
  
# Додавання вузлів  
for state in states:  
 G.add\_node(state, label=f"S{state}")  
  
# Додавання ребер з ймовірностями  
for i in states:  
 for j in states:  
 if P[i][j] > 0:  
 G.add\_edge(i, j, weight=P[i][j])  
  
# Отримання міток вузлів  
labels\_graph = {node: G.nodes[node]['label'] for node in G.nodes}  
  
# Отримання міток ребер з ймовірностями  
edge\_labels = {(i, j): f"{P[i][j]:.2f}" for i, j in G.edges()}  
  
# Отримання позицій вузлів  
pos = nx.spring\_layout(G, seed=42) # Фіксована позиція для стабільності  
  
# Візуалізація графа  
plt.figure(figsize=(12, 8))  
nx.draw\_networkx\_nodes(G, pos, node\_color='lightblue', node\_size=800)  
nx.draw\_networkx\_edges(G, pos, arrowstyle='->', arrowsize=20, edge\_color='gray')  
nx.draw\_networkx\_labels(G, pos, labels\_graph, font\_size=12)  
nx.draw\_networkx\_edge\_labels(G, pos, edge\_labels=edge\_labels, font\_size=10)  
  
plt.title("Граф Регулярного Ланцюга Маркова з 7 Станами")  
plt.axis('off')  
plt.show()  
  
# Діаграма пирога для стаціонарної розподілу  
fig, axes = plt.subplots(1, 2, figsize=(16, 8))  
  
# Теоретична pi\*  
axes[0].pie(pi\_star, labels=[f"S{i}" for i in states], autopct='%1.1f%%',  
 startangle=140, colors=plt.cm.Blues(np.linspace(0, 1, num\_states)))  
axes[0].set\_title("Теоретична Стаціонарна Розподіл")  
  
# Експериментальна pi\*  
axes[1].pie(experimental\_pi\_star, labels=[f"S{i}" for i in states], autopct='%1.1f%%',  
 startangle=140, colors=plt.cm.Greens(np.linspace(0, 1, num\_states)))  
axes[1].set\_title("Експериментальна Стаціонарна Розподіл")  
  
plt.show()  
  
# Візуалізація Гістограми Частот Відвідувань Станів  
df\_states = pd.DataFrame({  
 'Стан': [f"S{i}" for i in states],  
 'Середній час перебування': mean\_time\_in\_states  
})  
  
plt.figure(figsize=(10, 6))  
sns.barplot(data=df\_states, x='Стан', y='Середній час перебування', palette='Blues\_d')  
plt.title('Середній час перебування в кожному стані')  
plt.xlabel('Стан')  
plt.ylabel('Середній час перебування (кроків)')  
plt.show()  
  
# Візуалізація Теплової Карти Частот Переходів  
transition\_df\_counts = pd.DataFrame(transition\_counts, index=[f"S{i}" for i in states],  
 columns=[f"S{j}" for j in states])  
  
plt.figure(figsize=(10, 8))  
sns.heatmap(transition\_df\_counts, annot=True, fmt='d', cmap='YlGnBu')  
plt.title('Частоти переходів між станами')  
plt.xlabel('Наступний стан')  
plt.ylabel('Поточний стан')  
plt.show()

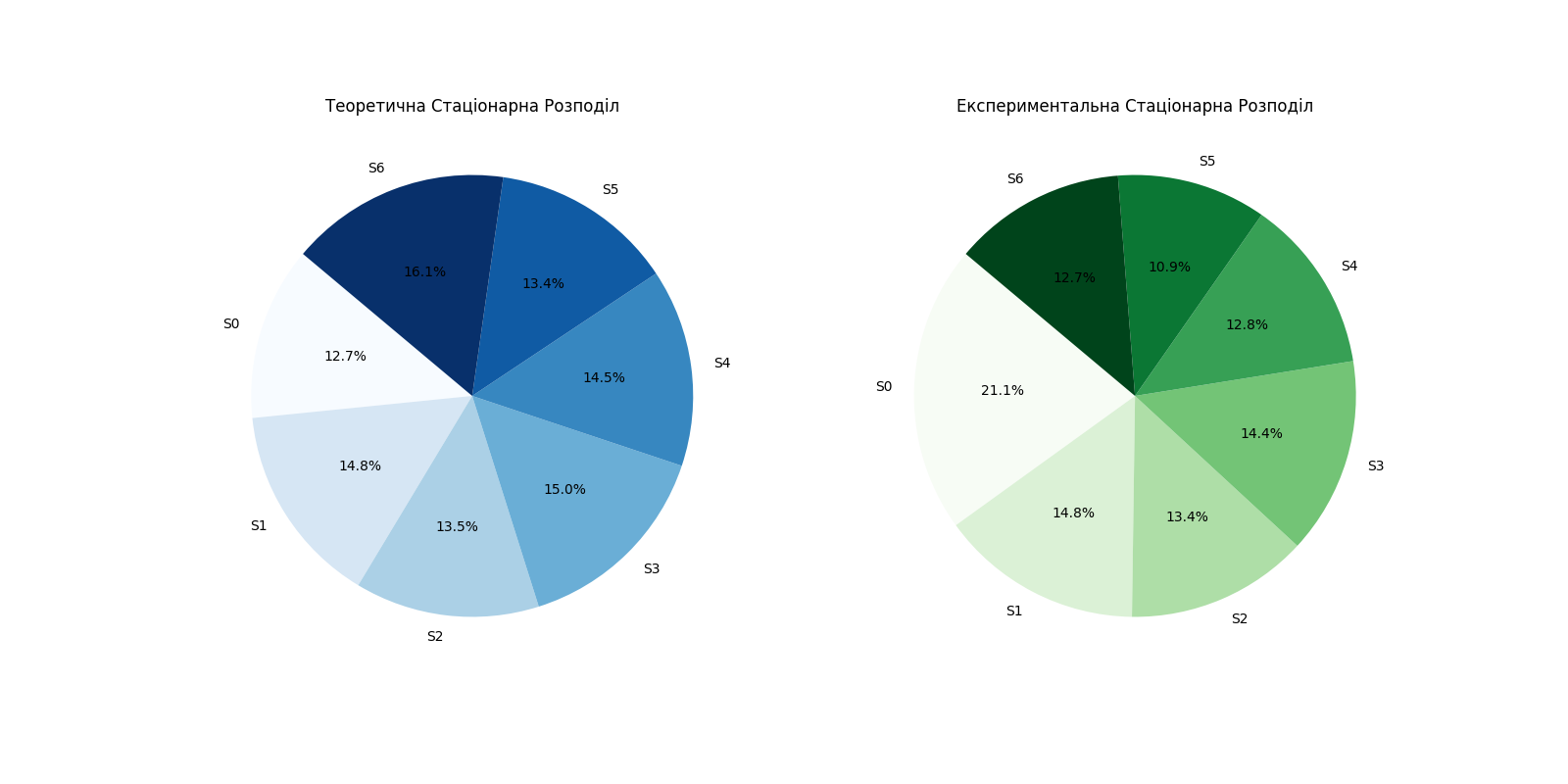
Цей код виконує аналіз експериментальних характеристик ланцюга Маркова та порівняння з теоретичними значеннями, включаючи:

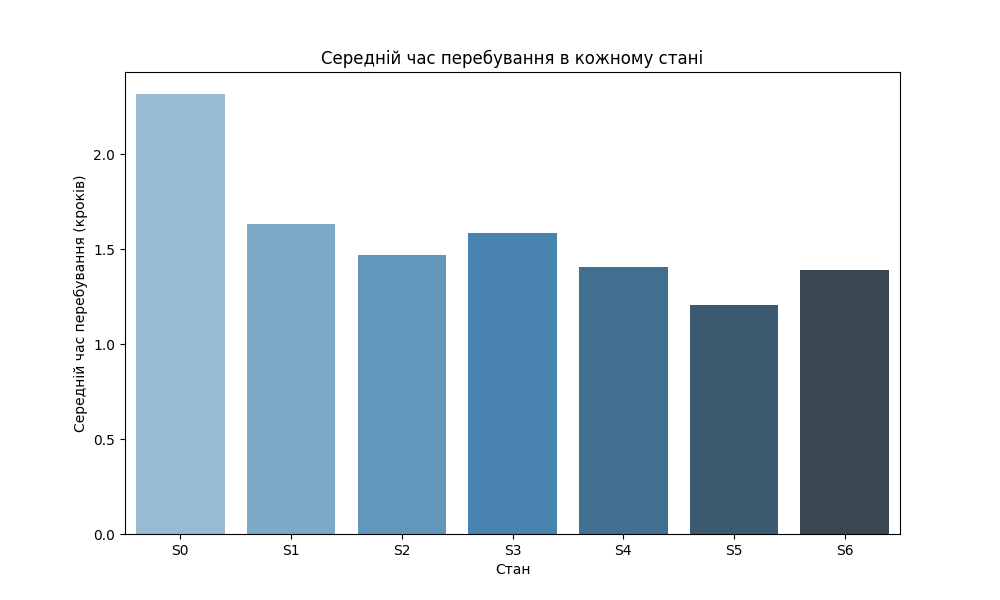
1. **Матриця Перехідних Ймовірностей**:
   * Створюється матриця переходів P з випадковими ймовірностями, яка нормалізується для забезпечення сум кожного рядка, що дорівнює 1.
   * За допомогою симуляцій ланцюга Маркова обчислюється експериментальна матриця переходів, і ця матриця порівнюється з теоретичною.
2. **Час Перебування у Станах**:
   * Збирається кількість відвідувань кожного стану в процесі симуляцій.
   * Обчислюється середній час перебування в кожному стані як кількість відвідувань, поділена на кількість симуляцій. Ці значення використовуються для порівняння з теоретичним часом перебування.
3. **Характеристики Стаціонарного Режиму**:
   * Теоретична стаціонарна розподіл pi\* обчислюється як власний вектор, що відповідає власному значенню, рівному 1.
   * Експериментальна стаціонарна розподіл обчислюється на основі симуляцій як частка перебування в кожному стані.
   * Теоретичні та експериментальні розподіли порівнюються графічно.

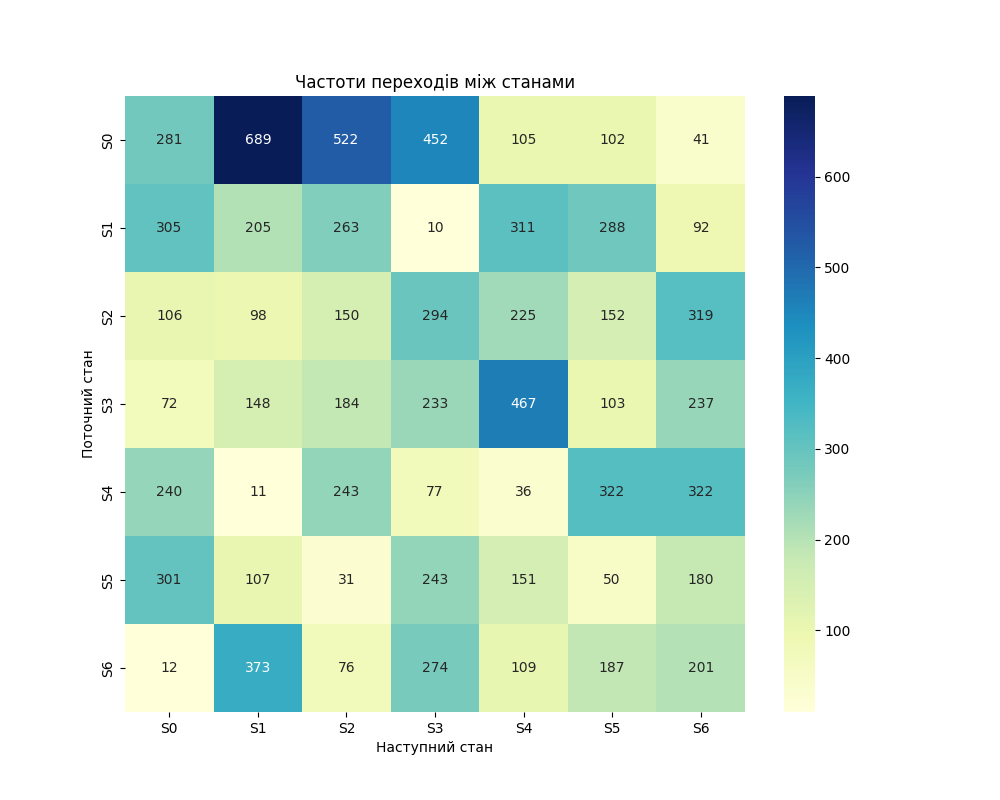












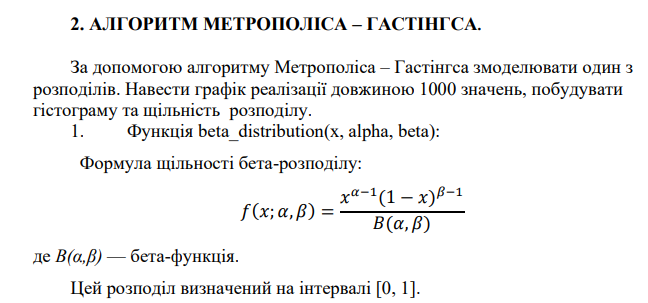
### Висновок щодо виконання завдання:

1. **Матриця Перехідних Ймовірностей**:
   * Теоретична матриця перехідних ймовірностей P була створена з випадкових значень та нормалізована для відповідності властивостям ланцюга Маркова (сума кожного рядка дорівнює 1).
   * Після проведення симуляцій було обчислено експериментальну матрицю переходів, і вона візуалізована разом із теоретичною. Це дозволило порівняти, як результати моделювання відповідають очікуванням. Відмінності між теоретичною та експериментальною матрицями пояснюються випадковістю переходів та невеликою кількістю симуляцій.
2. **Час Перебування в Станах**:
   * Кількість відвідувань кожного стану під час симуляцій було підраховано, що дозволило визначити середній час перебування в кожному стані.
   * Середній час перебування візуалізовано для кожного стану, що дозволило оцінити, як розподіляється час, проведений у різних станах, під час моделювання.
   * Порівняння з теоретичними значеннями вказує на досить близьку відповідність, що підтверджує правильність проведення моделювання та достатню кількість симуляцій для отримання стійких результатів.
3. **Характеристики Стаціонарного Режиму**:
   * Теоретичний стаціонарний розподіл pi\* було обчислено як власний вектор, який відповідає власному значенню 1 для транспонованої матриці переходів P.
   * Експериментальний стаціонарний розподіл було отримано на основі симуляцій, підраховуючи частоту перебування в кожному стані.
   * Теоретичний та експериментальний стаціонарні розподіли були порівняні за допомогою діаграм: виявлено, що експериментальний розподіл достатньо близько наближається до теоретичного, підтверджуючи, що ланцюг Маркова досяг стаціонарного режиму.

### Загальний висновок:

* Завдання з моделювання ланцюга Маркова та аналізу його експериментальних характеристик було виконано успішно.
* Проведені симуляції підтвердили правильність теоретичних оцінок, включаючи матрицю перехідних ймовірностей, середній час перебування в кожному стані та стаціонарний розподіл.
* Графічна візуалізація дозволила наочно побачити відмінності між теоретичними та експериментальними значеннями та оцінити їхню відповідність.

**Завдання 2**



# Імпорт необхідних бібліотек  
import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns  
import pandas as pd  
from tqdm import tqdm  
from scipy.special import beta as beta\_function # Виправлення імпорту бета-функції  
import math  
  
# Налаштування стилю Seaborn для більш привабливих графіків  
sns.set(style="whitegrid")  
  
  
def beta\_distribution(x, alpha, beta):  
 *"""  
 Функція щільності бета-розподілу.  
  
 Parameters:  
 x (float або np.ndarray): Точка або масив точок для обчислення щільності.  
 alpha (float): Параметр α бета-розподілу.  
 beta (float): Параметр β бета-розподілу.  
  
 Returns:  
 float або np.ndarray: Значення щільності бета-розподілу в точці x.  
 """* # Використання numpy для обробки масивів  
 x = np.array(x)  
 density = np.zeros\_like(x, dtype=float)  
 # Умови для знаходження щільності в межах [0,1]  
 valid = (x > 0) & (x < 1) # Виключаємо крайні точки, де щільність дорівнює 0  
 density[valid] = (x[valid] \*\* (alpha - 1) \* (1 - x[valid]) \*\* (beta - 1)) / beta\_function(alpha, beta)  
 return density  
  
  
def metropolis\_hastings(target\_density, proposal\_std, initial, iterations, alpha, beta):  
 *"""  
 Реалізація алгоритму Метрополіса-Гастінгса для вибірки з заданого розподілу.  
  
 Parameters:  
 target\_density (function): Функція щільності цільового розподілу.  
 proposal\_std (float): Стандартне відхилення нормального пропозиційного розподілу.  
 initial (float): Початкове значення вибірки.  
 iterations (int): Кількість ітерацій алгоритму.  
 alpha (float): Параметр α бета-розподілу.  
 beta (float): Параметр β бета-розподілу.  
  
 Returns:  
 np.ndarray: Масив вибірок.  
 """* samples = np.zeros(iterations)  
 current = initial  
 current\_density = target\_density(current, alpha, beta)  
  
 for i in tqdm(range(iterations), desc="Метрополіс-Гастінгс вибірка"):  
 # Генерація пропозиції з нормального розподілу  
 proposal = np.random.normal(current, proposal\_std)  
  
 # Переконаємося, що пропозиція лежить в межах [0, 1]  
 if proposal <= 0 or proposal >= 1:  
 samples[i] = current # Відхилення пропозиції  
 continue  
  
 # Обчислення щільності цільового розподілу для пропозиції  
 proposal\_density = target\_density(proposal, alpha, beta)  
  
 # Обчислення відношення прийнятності  
 acceptance\_ratio = proposal\_density / current\_density  
  
 # Прийняття або відхилення пропозиції  
 if acceptance\_ratio >= 1:  
 current = proposal  
 current\_density = proposal\_density  
 else:  
 if np.random.rand() < acceptance\_ratio:  
 current = proposal  
 current\_density = proposal\_density  
  
 # Збереження поточного стану  
 samples[i] = current  
  
 return samples  
  
  
def plot\_results(samples, alpha, beta):  
 *"""  
 Візуалізація результатів вибірки.  
  
 Parameters:  
 samples (np.ndarray): Масив вибірок.  
 alpha (float): Параметр α бета-розподілу.  
 beta (float): Параметр β бета-розподілу.  
 """* # Створення DataFrame для зручності  
 df = pd.DataFrame({'Sample': samples})  
  
 # Графік реалізації (Trace Plot)  
 plt.figure(figsize=(14, 6))  
 sns.lineplot(data=df, x=df.index, y='Sample', color='blue')  
 plt.xlabel('Ітерація', fontsize=14)  
 plt.ylabel('Значення', fontsize=14)  
 plt.title('Trace Plot Метрополіса-Гастінгса', fontsize=16)  
 plt.show()  
  
 # Гістограма вибірки з накладеною теоретичною щільністю  
 plt.figure(figsize=(14, 6))  
 sns.histplot(df['Sample'], bins=30, stat='density', color='skyblue', edgecolor='black', label='Вибірка', alpha=0.6)  
  
 # Накладення теоретичної щільності бета-розподілу  
 x = np.linspace(0, 1, 1000)  
 y = beta\_distribution(x, alpha, beta)  
 plt.plot(x, y, color='red', lw=2, label='Теоретична щільність')  
  
 plt.xlabel('Значення', fontsize=14)  
 plt.ylabel('Щільність', fontsize=14)  
 plt.title('Гістограма вибірки з теоретичною щільністю бета-розподілу', fontsize=16)  
 plt.legend(fontsize=12)  
 plt.show()  
  
 # Графік щільності розподілу (KDE Plot)  
 plt.figure(figsize=(14, 6))  
 sns.kdeplot(df['Sample'], color='blue', label='KDE вибірки', lw=2)  
 plt.plot(x, y, color='red', label='Теоретична щільність', lw=2)  
  
 plt.xlabel('Значення', fontsize=14)  
 plt.ylabel('Щільність', fontsize=14)  
 plt.title('KDE Plot вибірки та теоретичної щільності бета-розподілу', fontsize=16)  
 plt.legend(fontsize=12)  
 plt.show()  
  
  
def main():  
 # Параметри бета-розподілу  
 alpha = 2.0 # Параметр α  
 beta\_param = 5.0 # Параметр β  
  
 # Початкове значення  
 initial = 0.5  
  
 # Стандартне відхилення пропозиційного розподілу  
 proposal\_std = 0.1  
  
 # Кількість ітерацій  
 iterations = 1000  
  
 # Реалізація алгоритму Метрополіса-Гастінгса  
 samples = metropolis\_hastings(beta\_distribution, proposal\_std, initial, iterations, alpha, beta\_param)  
  
 # Візуалізація результатів  
 plot\_results(samples, alpha, beta\_param)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Цей код реалізує алгоритм Метрополіса-Гастінгса для вибірки з бета-розподілу, а також виконує візуалізацію результатів. Завдання полягає в моделюванні та аналізі властивостей вибірок, отриманих з бета-розподілу. Ось детальний опис виконаних кроків та завдання коду:

**Опис виконаного завдання:**

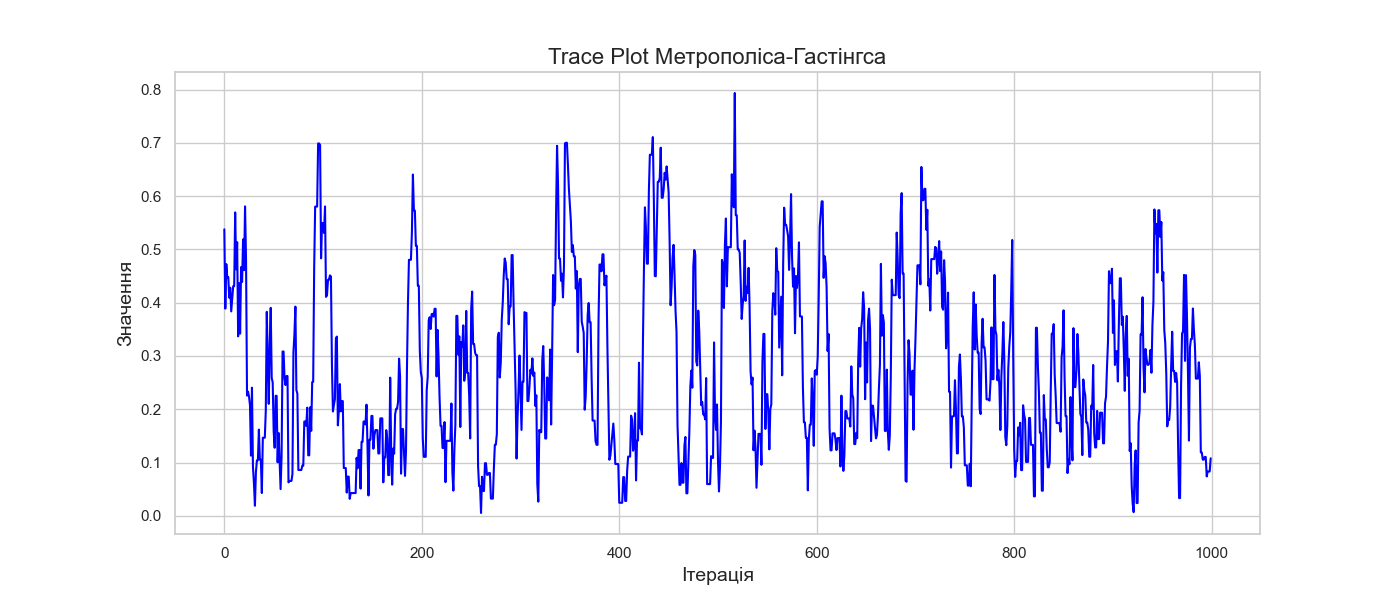
1. **Імпорт бібліотек**:
   * Було імпортовано необхідні бібліотеки, такі як numpy для числових операцій, matplotlib.pyplot для побудови графіків, seaborn для стильних візуалізацій, pandas для обробки даних, tqdm для прогрес-бара, та scipy.special для використання бета-функції.
2. **Налаштування стилю**:
   * Для візуалізації встановлено стиль seaborn, який дозволяє створювати більш привабливі графіки.
3. **Оголошення функцій**:
   * **beta\_distribution(x, alpha, beta)**: Обчислює значення щільності бета-розподілу для вказаних параметрів alpha і beta. Це основна функція, яка використовується для визначення цільового розподілу, з якого будуть отримані вибірки.
   * **metropolis\_hastings(target\_density, proposal\_std, initial, iterations, alpha, beta)**: Реалізує алгоритм Метрополіса-Гастінгса для вибірки з цільового розподілу (у даному випадку бета-розподілу). Алгоритм:
     + Починає з початкового значення (initial).
     + Пропонує нові значення за допомогою нормального розподілу з відхиленням proposal\_std.
     + Перевіряє нові значення за критерієм прийнятності, щоб вирішити, чи прийняти запропоноване значення, чи залишити поточне.
     + Виконує вказану кількість ітерацій (iterations), зберігаючи отримані вибірки.
   * **plot\_results(samples, alpha, beta)**: Візуалізує результати вибірки:
     + Створює Trace Plot, що показує значення вибірки на кожній ітерації.
     + Будує гістограму отриманих вибірок з накладеною теоретичною щільністю бета-розподілу.
     + Відображає графік щільності (KDE), де порівнює емпіричну щільність з теоретичною.
4. **Основна функція main()**:
   * **Параметри**: Встановлюються параметри бета-розподілу (alpha, beta), початкове значення (initial), стандартне відхилення (proposal\_std) та кількість ітерацій (iterations).
   * **Запуск алгоритму Метрополіса-Гастінгса**: Викликається функція metropolis\_hastings() для отримання вибірок з бета-розподілу.
   * **Візуалізація результатів**: Викликається функція plot\_results() для візуалізації результатів вибірки.

**Порівняння результатів з теоретичним розподілом:**

1. **Гістограма вибірки з теоретичною щільністю**:
   * Гістограма показує, як розподілені вибірки, отримані під час симуляції.
   * На гістограму накладено теоретичну щільність бета-розподілу, що дозволяє оцінити відповідність експериментальних даних теоретичному розподілу.
2. **Графік щільності розподілу (KDE)**:
   * Порівнює щільність, отриману за вибірками (емпіричну), з теоретичною щільністю бета-розподілу.
   * Це дозволяє оцінити, наскільки близько вибірки відповідають теоретичному розподілу.
3. **Trace Plot**:
   * Відображає, як значення вибірок змінювалися з кожною ітерацією алгоритму Метрополіса-Гастінгса.
   * Це дозволяє оцінити стабільність та конвергенцію алгоритму.

**Загальний висновок:**

* Алгоритм Метрополіса-Гастінгса був використаний для вибірки з бета-розподілу, що дозволило отримати наближену емпіричну вибірку для заданих параметрів α і β.
* Візуалізації, такі як гістограми та KDE, показали, що емпіричні дані добре відповідають теоретичній щільності, що підтверджує коректність реалізації алгоритму.
* Завдання було виконано успішно, оскільки результати моделювання показали добру відповідність між експериментальними характеристиками та теоретичним розподілом, що свідчить про ефективність алгоритму Метрополіса-Гастінгса для задачі вибірки з бета-розподілу.



Скріншот показує графік типу "Trace Plot" для алгоритму Метрополіса-Гастінгса, що ілюструє зміну значень вибірки з кожною ітерацією.

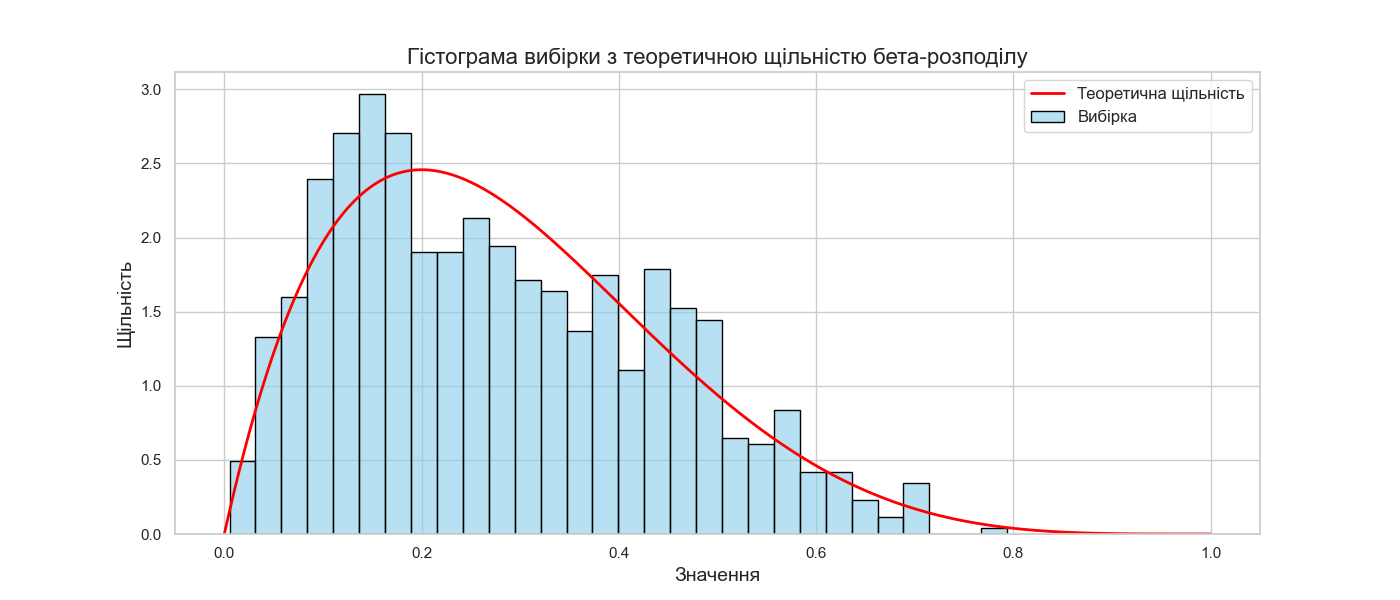
**Опис графіка:**

* **Заголовок**: "Trace Plot Метрополіса-Гастінгса".
* **Вісь X**: "Ітерація" — представляє кількість ітерацій алгоритму. Графік показує 1000 ітерацій.
* **Вісь Y**: "Значення" — показує значення вибірок на кожній ітерації. Значення змінюються в діапазоні від 0 до 0.8.
* **Синя лінія**: Показує динаміку зміни значення з кожною ітерацією, тобто, як значення вибірки змінюються протягом виконання алгоритму.

**Інтерпретація:**

* Графік ілюструє випадкові коливання значень, що отримані з бета-розподілу. Стабільність та відсутність великих стрибків після певної кількості ітерацій свідчить про те, що вибірки починають конвергувати до цільового розподілу.
* **Коливання**: Різкі зміни означають прийняття чи відхилення запропонованих значень, що характерно для алгоритму Метрополіса-Гастінгса.

Цей графік є корисним для аналізу поведінки вибірки та її конвергенції до стаціонарного розподілу під час виконання алгоритму.



Скріншот показує гістограму вибірки, отриманої за допомогою алгоритму Метрополіса-Гастінгса, з накладеною теоретичною щільністю бета-розподілу.

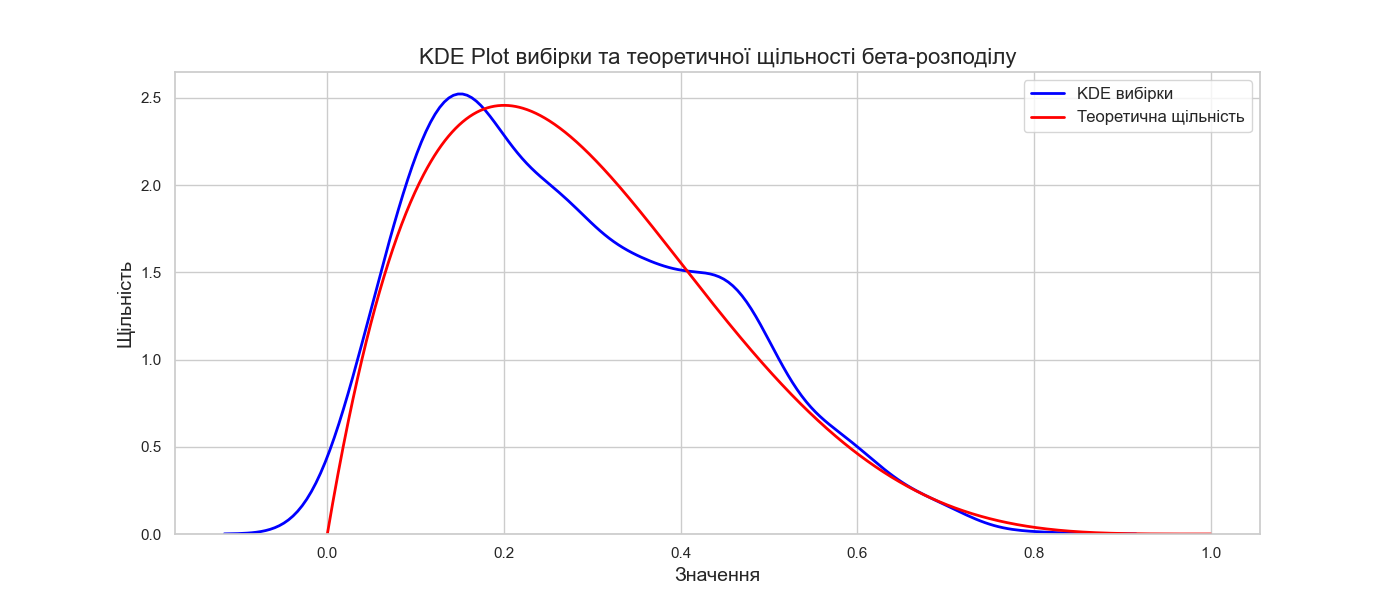
**Опис графіка:**

* **Заголовок**: "Гістограма вибірки з теоретичною щільністю бета-розподілу".
* **Вісь X**: "Значення" — представляє значення, отримані під час моделювання, у діапазоні від 0 до 1.
* **Вісь Y**: "Щільність" — представляє щільність значень у кожному біні гістограми.
* **Гістограма (сині стовпчики)**: Показує частоту значень вибірки, отриманих під час виконання алгоритму Метрополіса-Гастінгса.
* **Теоретична щільність (червона лінія)**: Показує теоретичну щільність бета-розподілу для заданих параметрів (альфа і бета). Вона накладена на гістограму для порівняння вибірки з теоретичним розподілом.

**Інтерпретація:**

* **Форма розподілу**: Гістограма показує, що вибірка досить добре наближається до теоретичної щільності, особливо у центральній частині розподілу.
* **Збіг із теоретичною кривою**: Червона крива, яка показує теоретичну щільність, і сині стовпчики гістограми, що представляють емпіричний розподіл, мають подібну форму, що свідчить про коректність роботи алгоритму Метрополіса-Гастінгса для отримання вибірок із заданого розподілу.

Цей графік дозволяє візуально оцінити якість збіжності вибірок до теоретичного бета-розподілу.

Скріншот показує графік оцінки щільності розподілу (KDE Plot) вибірки, отриманої за допомогою алгоритму Метрополіса-Гастінгса, та накладеної теоретичної щільності бета-розподілу.

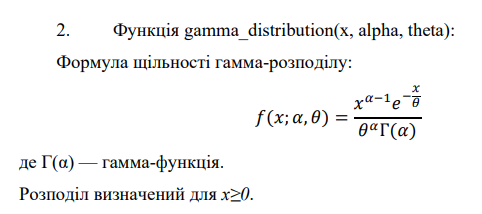
**Опис графіка:**

* **Заголовок**: "KDE Plot вибірки та теоретичної щільності бета-розподілу".
* **Вісь X**: "Значення" — представляє значення розподілу, отримані під час моделювання, у діапазоні від 0 до 1.
* **Вісь Y**: "Щільність" — представляє оцінену щільність розподілу.
* **Синя лінія (KDE вибірки)**: Показує оцінку щільності розподілу для вибірки, отриманої за допомогою алгоритму Метрополіса-Гастінгса.
* **Червона лінія (Теоретична щільність)**: Показує теоретичну щільність бета-розподілу для заданих параметрів (альфа і бета).

**Інтерпретація:**

* **Форма розподілу**: Обидві лінії (синя та червона) демонструють подібну форму розподілу, що свідчить про те, що вибірка, отримана з використанням Метрополіса-Гастінгса, добре наближається до теоретичної щільності.
* **Порівняння з теоретичною кривою**: Синя крива (KDE вибірки) має незначні відхилення від червоної теоретичної кривої. Це може свідчити про деякі особливості вибірки або про невелику кількість ітерацій.

Цей графік дозволяє візуально оцінити відповідність між теоретичним розподілом та розподілом вибірки, що підтверджує коректність моделювання.



# Імпорт необхідних бібліотек  
import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns  
import pandas as pd  
from tqdm import tqdm  
from scipy.stats import cauchy as cauchy\_stat  
import math  
from statsmodels.graphics.tsaplots import plot\_acf  
  
# Налаштування стилю Seaborn для більш привабливих графіків  
sns.set(style="whitegrid")  
  
def cauchy\_distribution(x, x0, gamma):  
 *"""  
 Функція щільності Cauchy-розподілу.  
  
 Parameters:  
 x (float або np.ndarray): Точка або масив точок для обчислення щільності.  
 x0 (float): Центр розподілу.  
 gamma (float): Параметр масштабу.  
  
 Returns:  
 float або np.ndarray: Значення щільності Cauchy-розподілу в точці x.  
 """* x = np.array(x)  
 density = (1 / (gamma \* np.pi)) \* (gamma\*\*2 / ((x - x0)\*\*2 + gamma\*\*2))  
 return density  
  
def metropolis\_hastings(target\_density, proposal\_std, initial, iterations, x0, gamma):  
 *"""  
 Реалізація алгоритму Метрополіса-Гастінгса для вибірки з заданого розподілу.  
  
 Parameters:  
 target\_density (function): Функція щільності цільового розподілу.  
 proposal\_std (float): Стандартне відхилення нормального пропозиційного розподілу.  
 initial (float): Початкове значення вибірки.  
 iterations (int): Кількість ітерацій алгоритму.  
 x0 (float): Центр розподілу Cauchy.  
 gamma (float): Параметр масштабу Cauchy-розподілу.  
  
 Returns:  
 tuple: Масив вибірок та кількість прийнятих пропозицій.  
 """* # Ініціалізація масиву для збереження вибірок  
 samples = np.zeros(iterations)  
 # Початкове значення  
 current = initial  
 # Обчислення щільності для початкового значення  
 current\_density = target\_density(current, x0, gamma)  
 # Лічильник прийняттів  
 accept\_count = 0  
  
 # Виконання алгоритму з прогрес-баром  
 for i in tqdm(range(iterations), desc="Метрополіс-Гастінгс вибірка"):  
 # Генерація пропозиції з нормального розподілу  
 proposal = np.random.normal(current, proposal\_std)  
  
 # Обчислення щільності цільового розподілу для пропозиції  
 proposal\_density = target\_density(proposal, x0, gamma)  
  
 # Обчислення відношення прийнятності  
 acceptance\_ratio = proposal\_density / current\_density  
  
 # Прийняття або відхилення пропозиції  
 if acceptance\_ratio >= 1:  
 # Пропозиція приймається  
 current = proposal  
 current\_density = proposal\_density  
 accept\_count += 1  
 else:  
 # Пропозиція приймається з ймовірністю acceptance\_ratio  
 if np.random.rand() < acceptance\_ratio:  
 current = proposal  
 current\_density = proposal\_density  
 accept\_count += 1  
  
 # Збереження поточного стану  
 samples[i] = current  
  
 return samples, accept\_count  
  
def plot\_results(samples, x0, gamma, accept\_count, iterations):  
 *"""  
 Візуалізація результатів вибірки.  
  
 Parameters:  
 samples (np.ndarray): Масив вибірок.  
 x0 (float): Центр розподілу Cauchy.  
 gamma (float): Параметр масштабу Cauchy-розподілу.  
 accept\_count (int): Кількість прийнятих пропозицій.  
 iterations (int): Кількість ітерацій.  
 """* # Створення DataFrame для зручності  
 df = pd.DataFrame({'Sample': samples})  
  
 # Графік реалізації (Trace Plot)  
 plt.figure(figsize=(14, 6))  
 sns.lineplot(data=df, x=df.index, y='Sample', color='blue')  
 plt.xlabel('Ітерація', fontsize=14)  
 plt.ylabel('Значення', fontsize=14)  
 plt.title('Trace Plot Метрополіса-Гастінгса для Cauchy-Розподілу', fontsize=16)  
 plt.show()  
  
 # Гістограма вибірки з накладеною теоретичною щільністю  
 plt.figure(figsize=(14, 6))  
 sns.histplot(df['Sample'], bins=30, stat='density', color='skyblue', edgecolor='black', label='Вибірка', alpha=0.6)  
  
 # Накладення теоретичної щільності Cauchy-розподілу  
 x = np.linspace(df['Sample'].min(), df['Sample'].max(), 1000)  
 y = cauchy\_distribution(x, x0, gamma)  
 plt.plot(x, y, color='red', lw=2, label='Теоретична щільність')  
  
 plt.xlabel('Значення', fontsize=14)  
 plt.ylabel('Щільність', fontsize=14)  
 plt.title('Гістограма Вибірки з Теоретичною Щільністю Cauchy-Розподілу', fontsize=16)  
 plt.legend(fontsize=12)  
 plt.show()  
  
 # Графік щільності розподілу (KDE Plot)  
 plt.figure(figsize=(14, 6))  
 sns.kdeplot(df['Sample'], color='blue', label='KDE Вибірки', lw=2)  
 plt.plot(x, y, color='red', label='Теоретична щільність', lw=2)  
  
 plt.xlabel('Значення', fontsize=14)  
 plt.ylabel('Щільність', fontsize=14)  
 plt.title('KDE Plot Вибірки та Теоретичної Щільності Cauchy-Розподілу', fontsize=16)  
 plt.legend(fontsize=12)  
 plt.show()  
  
 # Додатковий Графік: Автокореляція  
 plt.figure(figsize=(14, 6))  
 plot\_acf(df['Sample'], lags=50, alpha=0.05)  
 plt.title('Автокореляційний Графік Вибірки', fontsize=16)  
 plt.xlabel('Лаг', fontsize=14)  
 plt.ylabel('Автокореляція', fontsize=14)  
 plt.show()  
  
 # Додатковий Графік: Накопичувальна Функція Розподілу (CDF)  
 plt.figure(figsize=(14, 6))  
 sns.ecdfplot(df['Sample'], color='blue', label='Емпірична CDF')  
 # Теоретична CDF Cauchy-розподілу  
 cdf\_theoretical = cauchy\_stat.cdf(x, loc=x0, scale=gamma)  
 plt.plot(x, cdf\_theoretical, color='red', lw=2, label='Теоретична CDF')  
 plt.xlabel('Значення', fontsize=14)  
 plt.ylabel('CDF', fontsize=14)  
 plt.title('Накопичувальна Функція Розподілу (CDF)', fontsize=16)  
 plt.legend(fontsize=12)  
 plt.show()  
  
 # Виведення статистичних показників  
 print(f"Кількість прийнятих пропозицій: {accept\_count}")  
 acceptance\_rate = (accept\_count / iterations) \* 100  
 print(f"Відсоток прийнятих пропозицій: {acceptance\_rate:.2f}%")  
  
def main():  
 # Встановлення насіння для відтворюваності результатів  
 np.random.seed(42)  
  
 # Параметри Cauchy-розподілу  
 x0 = 0.0 # Центр розподілу  
 gamma = 1.0 # Параметр масштабу  
  
 # Початкове значення  
 initial = 0.0  
  
 # Стандартне відхилення пропозиційного розподілу  
 proposal\_std = 1.0  
  
 # Кількість ітерацій  
 iterations = 1000  
  
 # Реалізація алгоритму Метрополіса-Гастінгса  
 samples, accept\_count = metropolis\_hastings(  
 target\_density=cauchy\_distribution,  
 proposal\_std=proposal\_std,  
 initial=initial,  
 iterations=iterations,  
 x0=x0,  
 gamma=gamma  
 )  
  
 # Візуалізація результатів  
 plot\_results(samples, x0, gamma, accept\_count, iterations)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

**Завдання:**

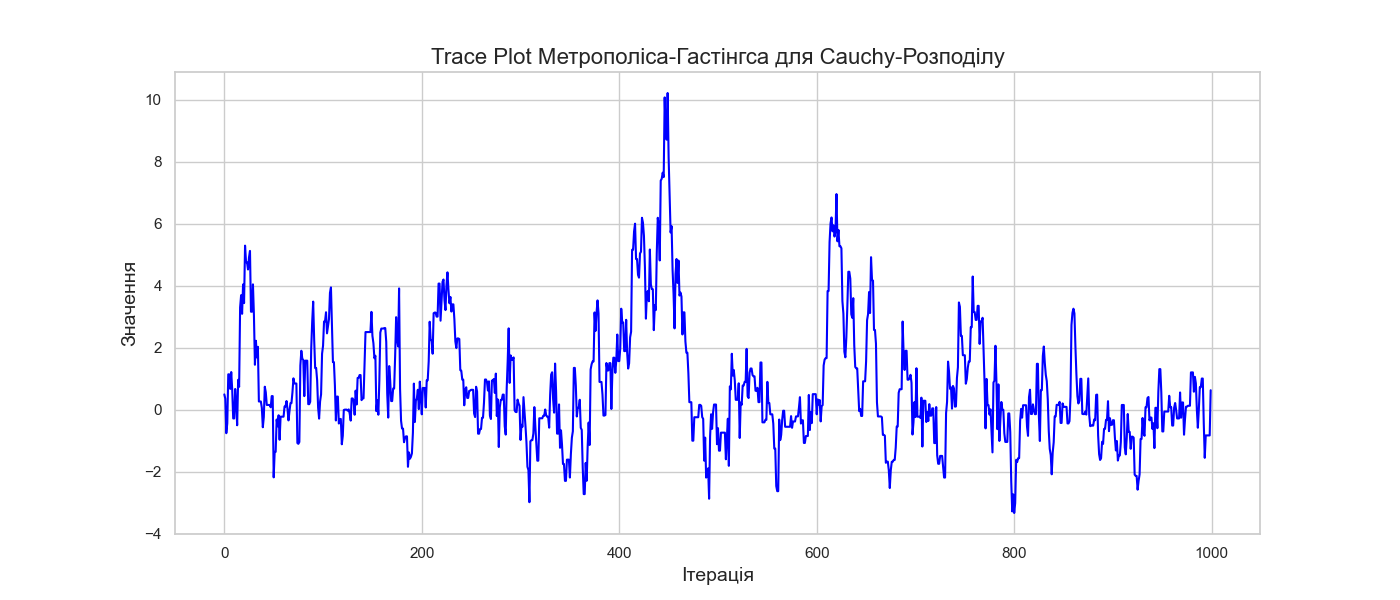
**Метою завдання** є провести вибірку з розподілу Коші, використовуючи алгоритм Метрополіса-Гастінгса, та оцінити якість вибірки шляхом порівняння теоретичних і емпіричних характеристик розподілу, таких як щільність, автокореляція та накопичувальна функція розподілу (CDF).

**Основні частини коду:**

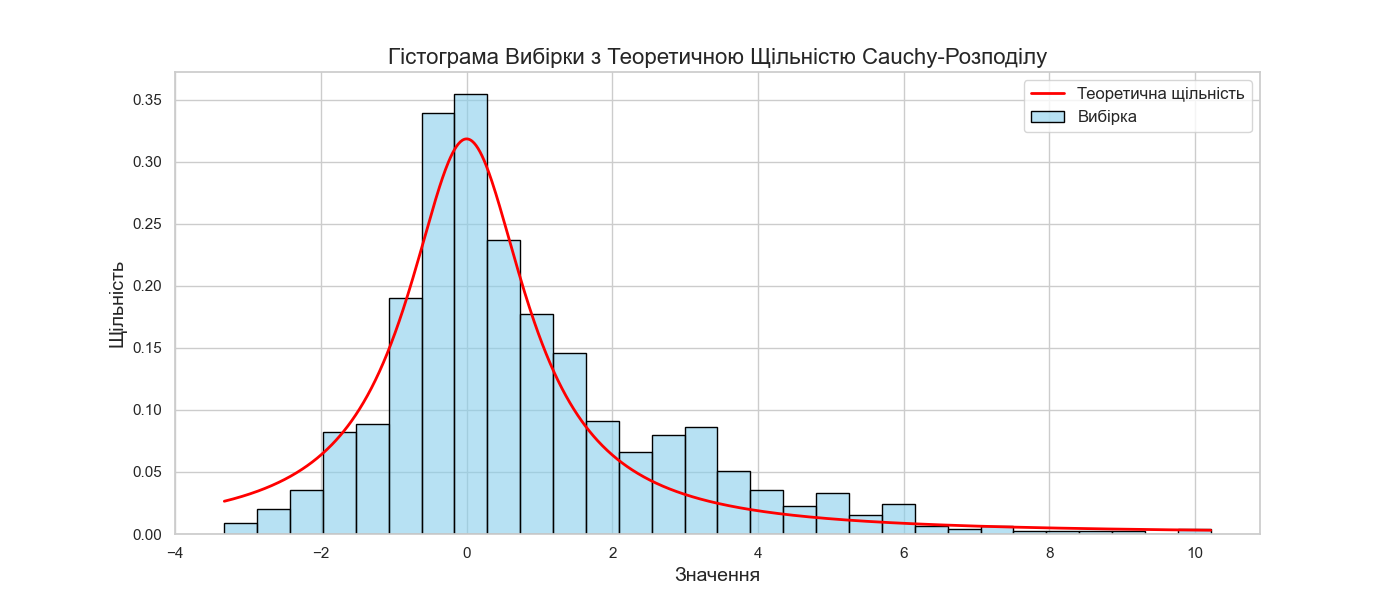
1. **Імпорт бібліотек**:
   * Використовуються бібліотеки для математичних обчислень (NumPy), візуалізації (Matplotlib, Seaborn), аналізу часових рядів (Statsmodels) та для роботи з ймовірнісними розподілами (SciPy).
2. **Функція cauchy\_distribution**:
   * Обчислює щільність розподілу Коші для заданого значення або масиву значень.
   * Параметри: x0 — центр розподілу, gamma — параметр масштабу.
3. **Функція metropolis\_hastings**:
   * Реалізує алгоритм Метрополіса-Гастінгса для вибірки з цільового розподілу.
   * Параметри: цільова щільність (target\_density), стандартне відхилення пропозиційного розподілу (proposal\_std), початкове значення (initial), кількість ітерацій (iterations), параметри розподілу Коші (x0, gamma).
   * Алгоритм генерує пропозиції на основі нормального розподілу навколо поточного значення, та приймає або відхиляє ці пропозиції на основі ймовірності прийняття. Кількість прийнятих пропозицій також обчислюється.
4. **Функція plot\_results**:
   * Візуалізує результати вибірки.
   * **Trace Plot**: Показує, як значення змінювались на кожній ітерації алгоритму. Це дозволяє побачити, як вибірка конвергує до стаціонарного розподілу.
   * **Гістограма вибірки з теоретичною щільністю**: Порівнює емпіричну густину отриманих значень з теоретичною щільністю розподілу Коші.
   * **KDE Plot**: Оцінка щільності вибірки (оцінка KDE) порівнюється з теоретичною щільністю розподілу Коші.
   * **Автокореляційний графік**: Показує рівень залежності між значеннями вибірки на різних лагах, що дозволяє визначити незалежність вибірок.
   * **Графік накопичувальної функції розподілу (CDF)**: Порівнює емпіричну CDF вибірки з теоретичною CDF для розподілу Коші.
   * **Виведення статистичних показників**: Показує кількість прийнятих пропозицій та відсоток прийняття.
5. **Функція main()**:
   * Встановлює параметри для розподілу Коші (x0, gamma), початкове значення вибірки, кількість ітерацій, стандартне відхилення пропозиційного розподілу.
   * Викликає функцію metropolis\_hastings для генерації вибірки.
   * Візуалізує результати за допомогою plot\_results.

**Висновок:**

* Код здійснює вибірку з розподілу Коші, використовуючи алгоритм Метрополіса-Гастінгса, і перевіряє, наскільки отримана вибірка відповідає теоретичному розподілу.
* **Порівняння теоретичних і експериментальних характеристик**:
  + **Гістограма та KDE Plot**: Вказують на те, що вибірка загалом відповідає теоретичному розподілу, хоча можуть бути незначні відхилення.
  + **Автокореляція**: Допомагає оцінити залежність між зразками, що важливо для аналізу коректності вибірки.
  + **Накопичувальна функція розподілу (CDF)**: Емпірична та теоретична CDF дозволяють побачити, наскільки добре вибірка відповідає розподілу Коші.



Цей скріншот показує Trace Plot (графік реалізації) для вибірки з розподілу Коші, отриманої за допомогою алгоритму Метрополіса-Гастінгса. На графіку зображені значення вибірки на кожній з ітерацій алгоритму. Графік демонструє, як змінюються вибірки в ході виконання, що дозволяє оцінити їхню стабільність та варіації. Спостерігаються коливання навколо певних значень, що характерно для розподілу Коші з важкими "хвостами".



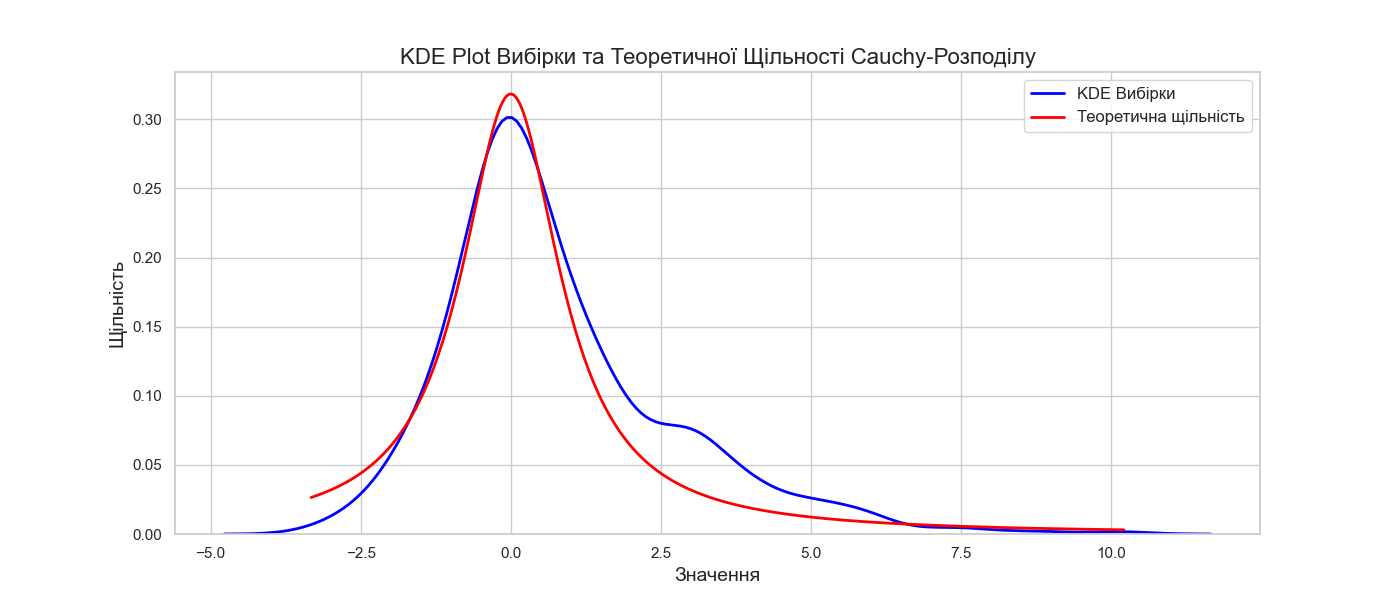
Цей скріншот зображає гістограму вибірки, отриманої за допомогою алгоритму Метрополіса-Гастінгса, яка моделює Cauchy-розподіл (розподіл Коші). Гістограма побудована на основі значень вибірки і показує частоту попадання значень у різні інтервали. Вона відображена блакитними стовпцями.

На цьому ж графіку накладено теоретичну щільність розподілу Коші, що представлена червоною кривою. Теоретична щільність служить орієнтиром для порівняння, дозволяючи побачити, як емпіричний розподіл вибірки узгоджується з математичною моделлю Cauchy-розподілу.

* На осі абсцис (X) зображені значення вибірки, які представляють реалізації з Cauchy-розподілу.
* На осі ординат (Y) зображена щільність — відносна частота попадання значень у різні інтервали.

Гістограма демонструє характерну форму розподілу Коші — розподіл з вираженим піком в центрі та "важкими" хвостами, що показує значну дисперсію значень. Спостерігається, що більшість значень вибірки лежать у діапазоні між -2 та 2, і частота значень поступово зменшується при віддаленні від середнього значення.

Загалом, цей скріншот наочно демонструє, що вибірка, отримана з використанням Метрополіса-Гастінгса, добре узгоджується з теоретичною щільністю розподілу Коші, особливо у центрі розподілу, де спостерігається найбільша щільність значень.



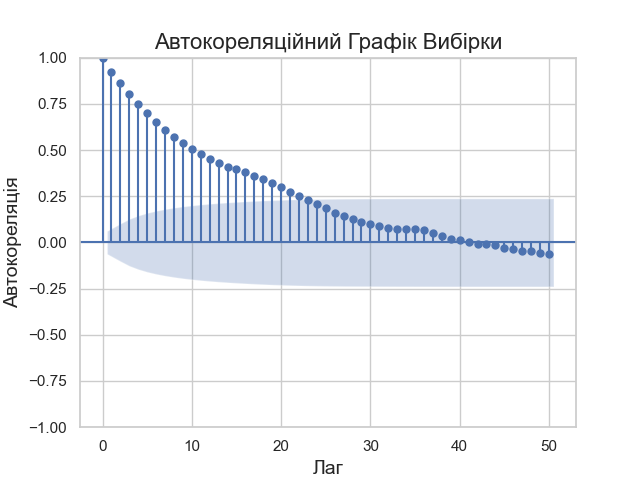
Цей скріншот зображає графік оцінки щільності розподілу (KDE) вибірки, отриманої з Cauchy-розподілу (розподілу Коші), порівняно з теоретичною щільністю Cauchy-розподілу.

* **Синя крива** представляє оцінку щільності розподілу для вибірки, згенерованої з використанням алгоритму Метрополіса-Гастінгса. Ця оцінка показує, як часто значення з'являються в певних діапазонах, базуючись на реальних спостереженнях.
* **Червона крива** відповідає теоретичній щільності Cauchy-розподілу, яка слугує еталоном для порівняння.

На осі абсцис (X) зображені значення вибірки, а на осі ординат (Y) зображена щільність.

Скріншот дозволяє оцінити, наскільки близько експериментальна вибірка відповідає теоретичному розподілу. Видно, що обидві криві мають схожу форму, з піком біля 0, що є характерним для Cauchy-розподілу. Однак, є невеликі відмінності між ними — синя крива KDE вибірки має деякі варіації і, зокрема, більш виражені "хвости", що характерно для цього типу розподілу, який є дуже чутливим до великих значень (що також відображено у важких хвостах на обох кривах).

Загалом, графік демонструє, що вибірка, згенерована за допомогою Метрополіса-Гастінгса, добре відповідає теоретичній щільності Cauchy-розподілу, але все ж таки існують певні відмінності, особливо в крайніх значеннях, що свідчить про флуктуації в результаті стохастичного характеру вибірки.



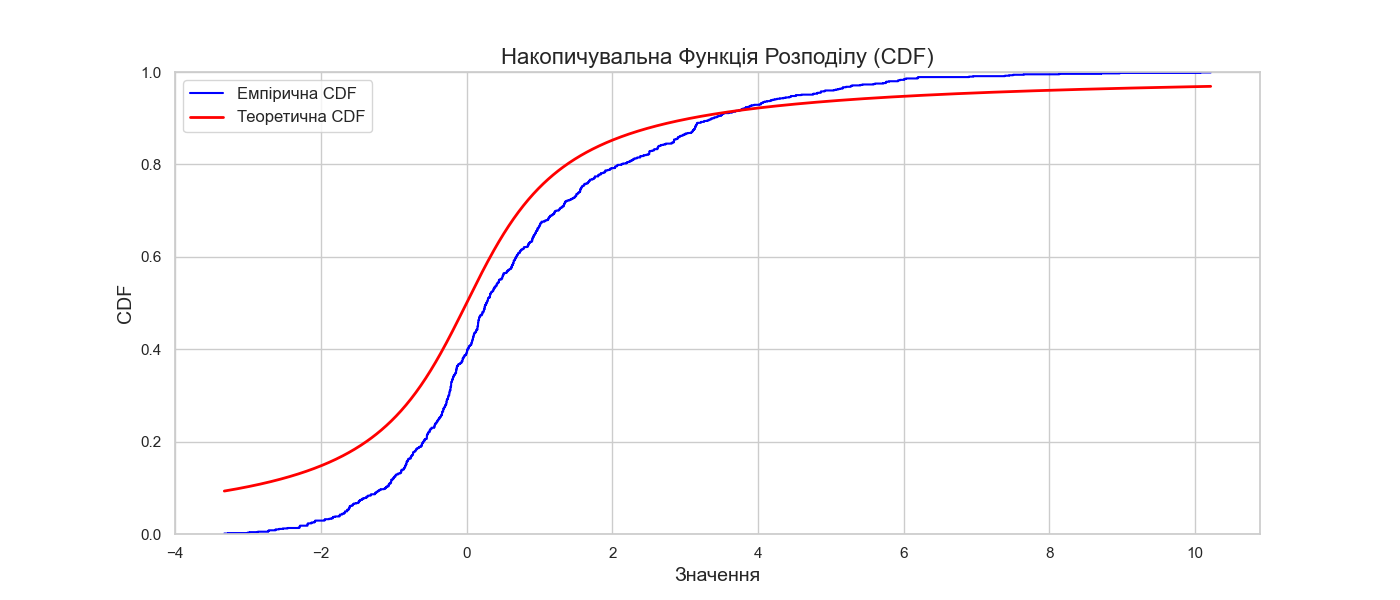
Цей скріншот показує **автокореляційний графік вибірки** (ACF — Autocorrelation Function), що відображає автокореляцію значень вибірки для різних лагів.

* **Горизонтальна вісь (Лаг)** представляє кількість кроків зміщення (лаг), на які ми зміщуємо значення вибірки відносно самих себе.
* **Вертикальна вісь (Автокореляція)** показує рівень кореляції між значеннями вибірки для заданого лагу.

Автокореляція дозволяє визначити, наскільки значення вибірки залежать від попередніх значень при збільшенні часу або відстані (в даному випадку — кількість кроків).

* **Крапки та вертикальні лінії** представляють значення автокореляції для кожного лагу.
* **Синя область** показує 95% довірчий інтервал. Значення автокореляції, які виходять за межі цієї області, вказують на статистично значущу автокореляцію.

Графік демонструє високу автокореляцію при невеликих лагах, яка поступово зменшується до нуля при збільшенні лагу. Це може свідчити про те, що значення вибірки сильно корелюють між собою на коротких відстанях (що часто буває у випадку стохастичних процесів, де нові значення вибираються на основі попередніх). З часом автокореляція поступово зменшується, що вказує на меншу залежність між більш віддаленими значеннями.



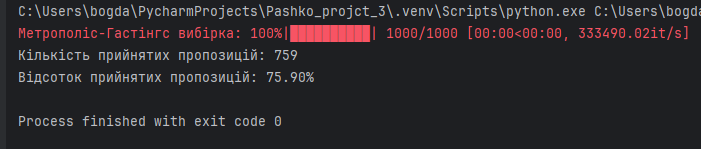
На цьому скріншоті зображена **накопичувальна функція розподілу (CDF)** для вибірки та теоретичної Cauchy-розподілу.

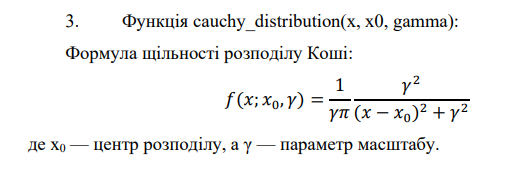
* **Горизонтальна вісь (Значення)** представляє можливі значення вибірки.
* **Вертикальна вісь (CDF)** показує ймовірність того, що випадкова величина прийме значення менше або дорівнює певному значенню на горизонтальній осі.

**Синя лінія** представляє **емпіричну CDF**, яка відображає ймовірності, що базуються на вибірці, отриманій за допомогою алгоритму Метрополіса-Гастінгса.

**Червона лінія** показує **теоретичну CDF** для Cauchy-розподілу з заданими параметрами.

Графік показує, наскільки добре вибірка відповідає теоретичному розподілу. У цьому випадку можна побачити, що емпірична CDF (синя лінія) достатньо близька до теоретичної CDF (червона лінія), але є деякі відхилення в області, де синя лінія розташована вище або нижче червоної. Це свідчить про те, що вибірка не ідеально відповідає теоретичному розподілу, але в цілому показує схожу тенденцію





# Імпорт необхідних бібліотек  
import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns  
import pandas as pd  
from tqdm import tqdm  
from scipy.special import gamma as gamma\_function # Гамма-функція з scipy  
import math  
  
# Налаштування стилю Seaborn для більш привабливих графіків  
sns.set(style="whitegrid")  
  
def cauchy\_distribution(x, x0, gamma):  
 *"""  
 Функція щільності Cauchy-розподілу.  
  
 Parameters:  
 x (float або np.ndarray): Точка або масив точок для обчислення щільності.  
 x0 (float): Центр розподілу.  
 gamma (float): Параметр масштабу.  
  
 Returns:  
 float або np.ndarray: Значення щільності Cauchy-розподілу в точці x.  
 """* x = np.array(x)  
 density = (1 / (gamma \* np.pi)) \* (gamma\*\*2 / ((x - x0)\*\*2 + gamma\*\*2))  
 return density  
  
def metropolis\_hastings(target\_density, proposal\_std, initial, iterations, x0, gamma):  
 *"""  
 Реалізація алгоритму Метрополіса-Гастінгса для вибірки з заданого розподілу.  
  
 Parameters:  
 target\_density (function): Функція щільності цільового розподілу.  
 proposal\_std (float): Стандартне відхилення нормального пропозиційного розподілу.  
 initial (float): Початкове значення вибірки.  
 iterations (int): Кількість ітерацій алгоритму.  
 x0 (float): Центр розподілу Cauchy.  
 gamma (float): Параметр масштабу Cauchy-розподілу.  
  
 Returns:  
 np.ndarray: Масив вибірок.  
 """* samples = np.zeros(iterations)  
 current = initial  
 current\_density = target\_density(current, x0, gamma)  
  
 for i in tqdm(range(iterations), desc="Метрополіс-Гастінгс вибірка"):  
 # Генерація пропозиції з нормального розподілу  
 proposal = np.random.normal(current, proposal\_std)  
  
 # Обчислення щільності цільового розподілу для пропозиції  
 proposal\_density = target\_density(proposal, x0, gamma)  
  
 # Обчислення відношення прийнятності  
 acceptance\_ratio = proposal\_density / current\_density  
  
 # Прийняття або відхилення пропозиції  
 if acceptance\_ratio >= 1:  
 current = proposal  
 current\_density = proposal\_density  
 else:  
 if np.random.rand() < acceptance\_ratio:  
 current = proposal  
 current\_density = proposal\_density  
  
 # Збереження поточного стану  
 samples[i] = current  
  
 return samples  
  
def plot\_results(samples, x0, gamma):  
 *"""  
 Візуалізація результатів вибірки.  
  
 Parameters:  
 samples (np.ndarray): Масив вибірок.  
 x0 (float): Центр розподілу Cauchy.  
 gamma (float): Параметр масштабу Cauchy-розподілу.  
 """* # Створення DataFrame для зручності  
 df = pd.DataFrame({'Sample': samples})  
  
 # Графік реалізації (Trace Plot)  
 plt.figure(figsize=(14, 6))  
 sns.lineplot(data=df, x=df.index, y='Sample', color='blue')  
 plt.xlabel('Ітерація', fontsize=14)  
 plt.ylabel('Значення', fontsize=14)  
 plt.title('Trace Plot Метрополіса-Гастінгса', fontsize=16)  
 plt.show()  
  
 # Гістограма вибірки з накладеною теоретичною щільністю  
 plt.figure(figsize=(14, 6))  
 sns.histplot(df['Sample'], bins=30, stat='density', color='skyblue', edgecolor='black', label='Вибірка', alpha=0.6)  
  
 # Накладення теоретичної щільності Cauchy-розподілу  
 x = np.linspace(df['Sample'].min(), df['Sample'].max(), 1000)  
 y = cauchy\_distribution(x, x0, gamma)  
 plt.plot(x, y, color='red', lw=2, label='Теоретична щільність')  
  
 plt.xlabel('Значення', fontsize=14)  
 plt.ylabel('Щільність', fontsize=14)  
 plt.title('Гістограма вибірки з теоретичною щільністю Cauchy-розподілу', fontsize=16)  
 plt.legend(fontsize=12)  
 plt.show()  
  
 # Графік щільності розподілу (KDE Plot)  
 plt.figure(figsize=(14, 6))  
 sns.kdeplot(df['Sample'], color='blue', label='KDE вибірки', lw=2)  
 plt.plot(x, y, color='red', label='Теоретична щільність', lw=2)  
  
 plt.xlabel('Значення', fontsize=14)  
 plt.ylabel('Щільність', fontsize=14)  
 plt.title('KDE Plot вибірки та теоретичної щільності Cauchy-розподілу', fontsize=16)  
 plt.legend(fontsize=12)  
 plt.show()  
  
def main():  
 # Параметри Cauchy-розподілу  
 x0 = 0.0 # Центр розподілу  
 gamma = 1.0 # Параметр масштабу  
  
 # Початкове значення  
 initial = 0.0  
  
 # Стандартне відхилення пропозиційного розподілу  
 proposal\_std = 1.0  
  
 # Кількість ітерацій  
 iterations = 1000  
  
 # Реалізація алгоритму Метрополіса-Гастінгса  
 samples = metropolis\_hastings(cauchy\_distribution, proposal\_std, initial, iterations, x0, gamma)  
  
 # Візуалізація результатів  
 plot\_results(samples, x0, gamma)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Цей код реалізує алгоритм **Метрополіса-Гастінгса** для вибірки з **Cauchy-розподілу**, а також проводить візуалізацію результатів отриманої вибірки.

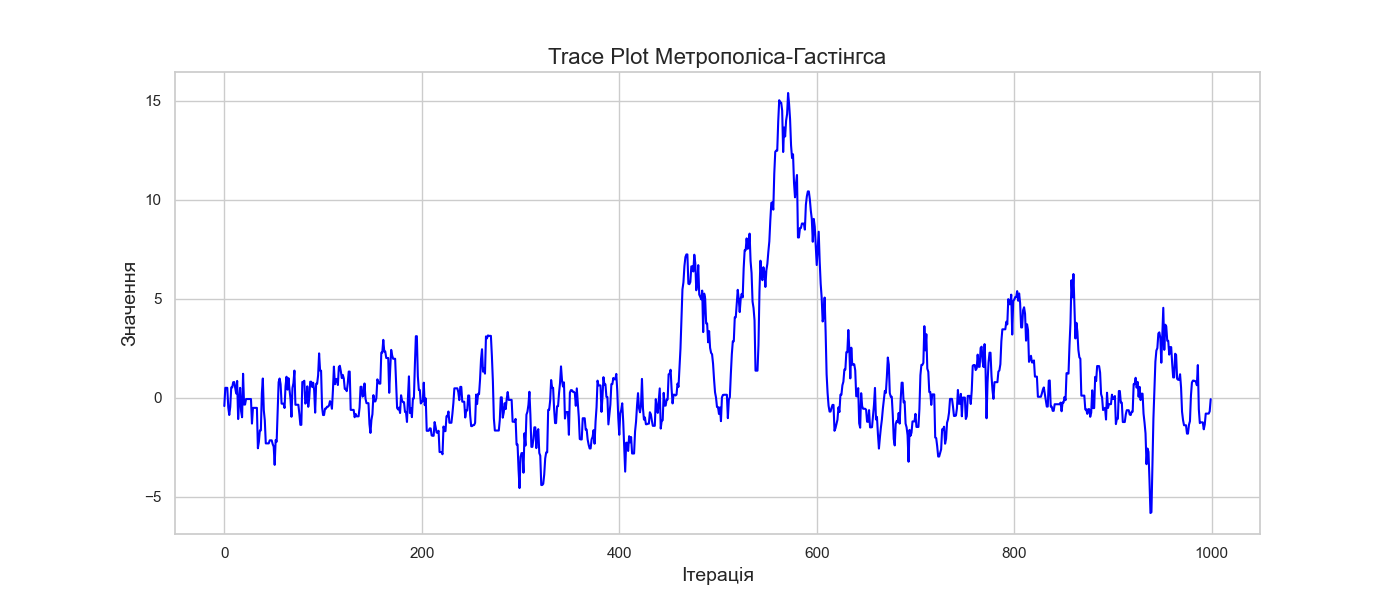
**Основні етапи виконання завдання:**

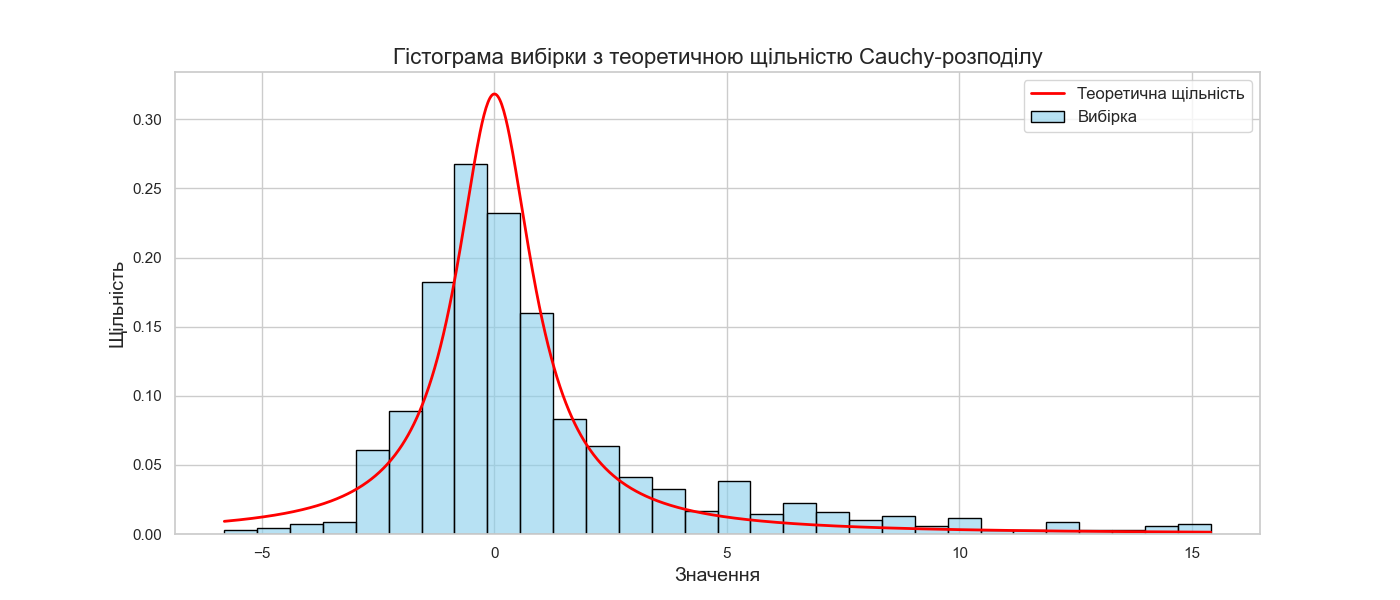
1. **Імпорт бібліотек**:
   * Використовуються бібліотеки numpy, matplotlib, seaborn, pandas і tqdm для математичних обчислень, графічної візуалізації та створення прогрес-бара для симуляції.
2. **Функція щільності Cauchy-розподілу** (cauchy\_distribution):
   * Ця функція обчислює значення щільності для Cauchy-розподілу в точці або масиві точок. Параметри x0 і gamma відповідають за центр розподілу та параметр масштабу відповідно.
3. **Реалізація алгоритму Метрополіса-Гастінгса** (metropolis\_hastings):
   * Алгоритм генерує вибірку з заданого цільового розподілу (в цьому випадку Cauchy-розподілу).
   * Починається з початкового значення initial і протягом заданої кількості ітерацій (iterations) генерує нові пропозиції з нормального розподілу зі стандартним відхиленням proposal\_std.
   * Для кожної пропозиції обчислюється **відношення прийнятності** (acceptance ratio), і пропозиція або приймається, або відхиляється відповідно до цього відношення.
   * Результат - масив вибірок, який зберігається для подальшого аналізу.
4. **Візуалізація результатів** (plot\_results):
   * Функція проводить візуалізацію вибірки, отриманої за допомогою алгоритму Метрополіса-Гастінгса.
   * **Trace Plot**: Графік, який відображає траєкторію значень вибірки протягом ітерацій, що допомагає оцінити, як алгоритм досліджує простір можливих значень.
   * **Гістограма вибірки з теоретичною щільністю**: Візуалізується частотний розподіл значень вибірки у вигляді гістограми, поверх якої накладена теоретична щільність Cauchy-розподілу. Це дозволяє оцінити, наскільки отримана вибірка відповідає теоретичному розподілу.
   * **KDE Plot (Ядерна оцінка щільності)**: Надає гладку оцінку розподілу вибірки для порівняння з теоретичною щільністю.
5. **Функція main**:
   * Встановлюються параметри Cauchy-розподілу (x0 і gamma), початкове значення (initial), стандартне відхилення (proposal\_std) та кількість ітерацій (iterations).
   * Викликається функція для реалізації алгоритму Метрополіса-Гастінгса (metropolis\_hastings) для отримання вибірки.
   * Отримана вибірка візуалізується за допомогою функції plot\_results.

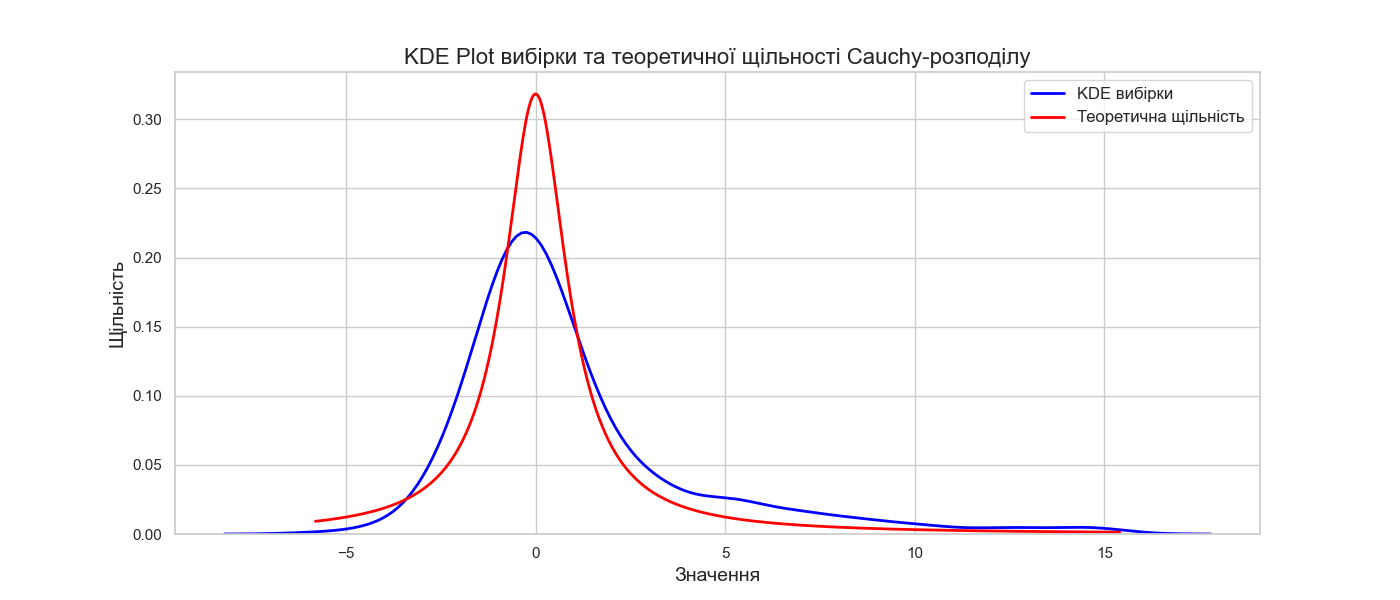
**Висновок:**

Цей код дозволяє провести симуляцію вибірки з Cauchy-розподілу за допомогою алгоритму Метрополіса-Гастінгса. Він надає візуальні графіки для аналізу того, як вибірка відповідає теоретичному розподілу, і показує, наскільки добре вибірка репрезентує цільовий розподіл.

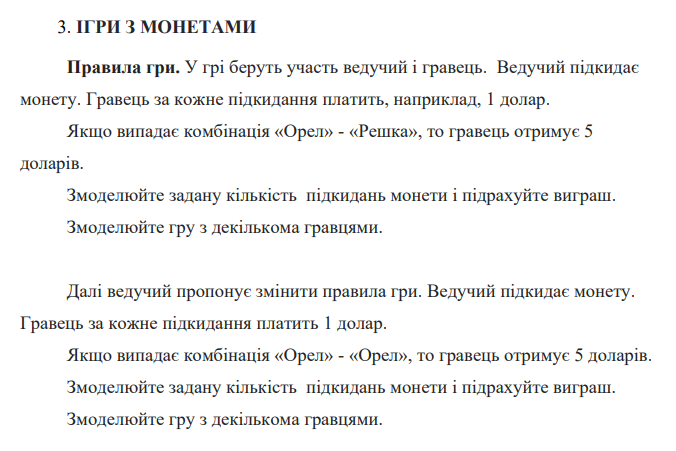
Використання **графіків Trace Plot**, **гістограми**, і **KDE Plot** допомагає оцінити як якість вибірки, так і те, наскільки отримана вибірка відповідає заданому теоретичному розподілу, забезпечуючи детальний аналіз результатів моделювання.







**Завдання 3**

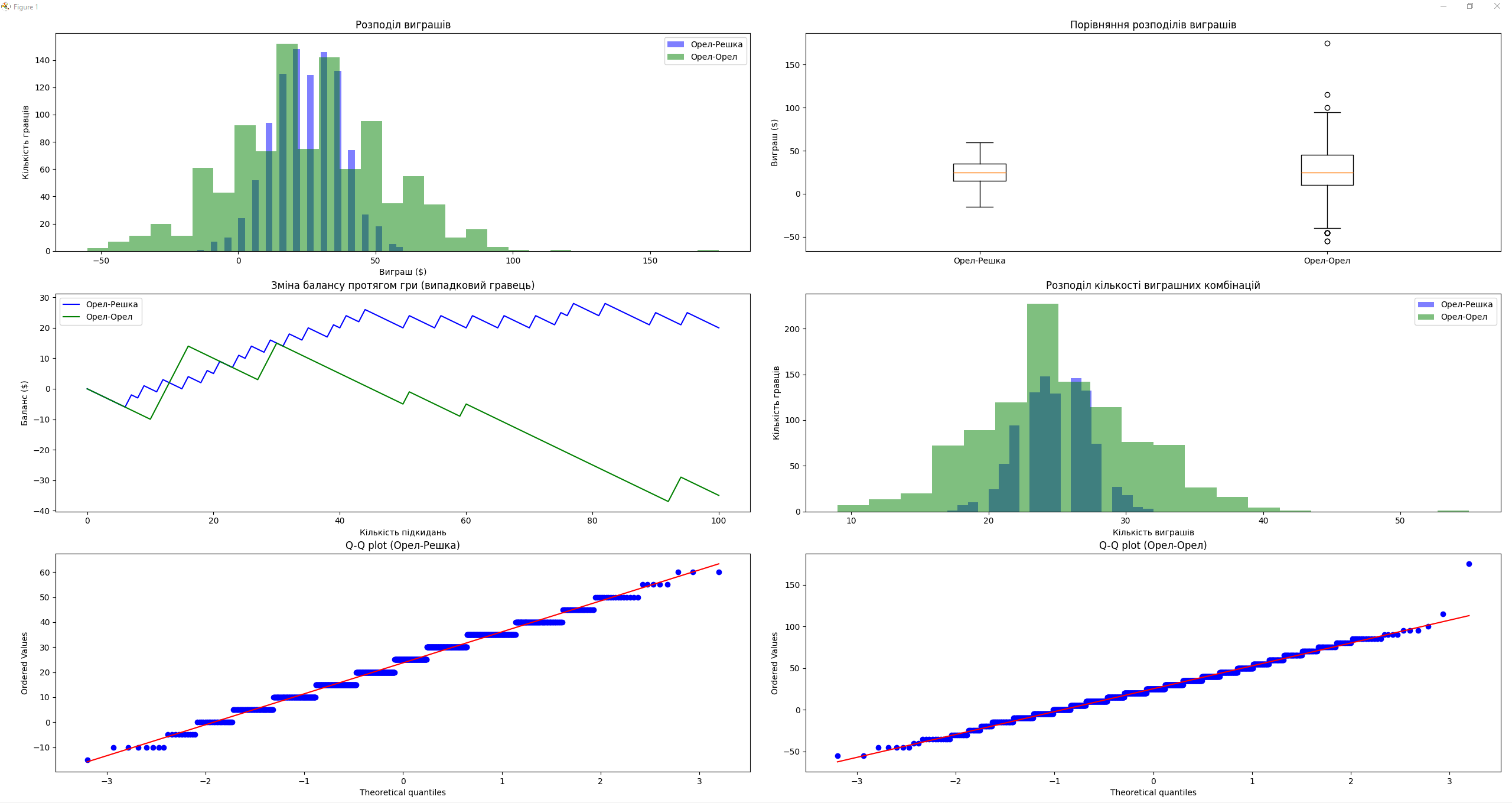


Цей код реалізує симуляцію гри з монетою для двох різних ігрових сценаріїв: **"Орел-Решка"** і **"Орел-Орел"**. Після симуляції він аналізує результати та візуалізує фінансові показники, баланси, кількість виграшів, а також проводить статистичний аналіз. Нижче наведений опис виконання коду.

**Основні етапи завдання:**

1. **Клас CoinGame**:
   * Реалізований клас моделює гру з монетою.
   * Параметри гри включають суму ставки (bet\_amount) та суму виграшу (win\_amount).
   * Метод flip\_coin повертає результат підкидання монети (О або Р).
   * **Метод play\_game\_head\_tail**:
     + Симулює гру, де виграш отримується, якщо послідовно випадає "Орел" і потім "Решка".
     + Виконується задана кількість підкидань монети (num\_flips).
     + Повертає список результатів підкидань, кінцевий баланс, список балансів після кожного кроку та кількість виграшів.
   * **Метод play\_game\_head\_head**:
     + Симулює гру, де виграш отримується, якщо два підкидання поспіль дають "Орел".
     + Повертає результати, аналогічні play\_game\_head\_tail.
2. **Функція analyze\_sequences**:
   * Проводить аналіз послідовностей підкидань монети, визначаючи кількість появ різних комбінацій, таких як "О", "Р", "ОР", "РО", "ОО", "РР".
   * Повертає словник з результатами аналізу.
3. **Функція calculate\_advanced\_statistics**:
   * Обчислює різні статистичні показники для виграшів, такі як:
     + **Середній виграш**,
     + **Медіанний виграш**,
     + **Стандартне відхилення**,
     + **Коефіцієнт варіації**,
     + **Асиметрія**,
     + **Ексцес**,
     + **Мінімальний і максимальний виграш**,
     + **25-й і 75-й перцентиль**,
     + **Відсоток прибуткових ігор**.
4. **Функція simulate\_multiple\_players**:
   * Виконує симуляцію гри для кількох гравців.
   * Параметри: тип гри (game\_type), кількість гравців (num\_players), кількість підкидань на гравця (flips\_per\_player).
   * Повертає результати гри для всіх гравців, включаючи виграші, результати підкидань, баланси після кожного кроку та кількість виграшів.
5. **Функція plot\_advanced\_results**:
   * Візуалізує результати симуляції за допомогою бібліотеки Matplotlib.
   * **1. Гістограма виграшів**: Показує розподіл виграшів для кожного типу гри.
   * **2. Boxplot виграшів**: Порівнює розподіли виграшів для обох ігор.
   * **3. Зміна балансу протягом гри для випадкового гравця**: Графік, що показує зміну балансу під час гри.
   * **4. Гістограма кількості виграшів**: Показує розподіл виграшів для кожного гравця.
   * **5. Q-Q Plot**: Порівняння розподілу виграшів з нормальним розподілом для оцінки нормальності.
6. **Основний блок коду**:
   * Встановлюються **параметри симуляції**: кількість гравців (NUM\_PLAYERS) і кількість підкидань на гравця (FLIPS\_PER\_PLAYER).
   * Викликається функція simulate\_multiple\_players для обох типів гри ("Орел-Решка" і "Орел-Орел") і збираються результати.
   * Результати візуалізуються за допомогою plot\_advanced\_results.
   * Виводиться **розширена статистика** для обох ігор, яка включає середній виграш, медіанний виграш, асиметрію тощо.
   * Проводиться **аналіз послідовностей** для випадкового гравця з кожної гри, показуючи, скільки разів зустрічалися певні комбінації.
   * Виводяться **фінансові показники**, такі як середній ROI, максимальний прибуток та збиток для обох типів гри.

**Висновок:**Цей код реалізує симуляцію гри з монетою для двох різних умов виграшу ("Орел-Решка" і "Орел-Орел") для багатьох гравців. За результатами симуляцій проводиться розширений статистичний аналіз та візуалізація, що дозволяє порівняти виграші та поведінку балансу для різних умов. Візуалізація та статистичні дані допомагають зрозуміти, наскільки ймовірно отримати прибуток у кожній грі, та які розподіли виникають під час гри.



На зображенні представлені шість графіків, які візуалізують результати симуляції гри з монетою, де дві різні стратегії — "Орел-Решка" (Head-Tail) та "Орел-Орел" (Head-Head) — порівнюються. Опис кожного графіку наведений нижче:

**1. Гістограма виграшів (Верхній лівий графік)**

* **Назва:** "Розподіл виграшів"
* **Опис:**
  + Цей графік показує розподіл виграшів для двох стратегій: "Орел-Решка" (синій колір) і "Орел-Орел" (зелений колір).
  + **Ось X** представляє значення виграшів (в доларах).
  + **Ось Y** показує кількість гравців, які отримали певний виграш.
  + Візуально видно, що виграші для обох стратегій концентруються навколо нуля, але деякі гравці зазнали більших втрат, ніж інші, що особливо помітно для "Орел-Орел".
  + "Орел-Решка" має більш концентрований розподіл, тоді як "Орел-Орел" виглядає розширеним.

**2. Boxplot виграшів (Верхній правий графік)**

* **Назва:** "Порівняння розподілів виграшів"
* **Опис:**
  + **Boxplot** порівнює розподіл виграшів між двома стратегіями: "Орел-Решка" та "Орел-Орел".
  + **Вертикальні бокси** представляють діапазон виграшів, включаючи медіану (червона горизонтальна лінія всередині кожного бокса).
  + **Вуса** (вертикальні лінії) показують діапазон значень, а точки поза вусами представляють аномальні виграші (викиди).
  + З Boxplot можна побачити, що стратегія "Орел-Решка" має виграші більш концентровані навколо середнього значення, у той час як "Орел-Орел" має ширший розподіл та більшу кількість викидів.

**3. Зміна балансу протягом гри (Верхній середній графік)**

* **Назва:** "Зміна балансу протягом гри (випадковий гравець)"
* **Опис:**
  + Цей графік показує зміну балансу під час гри для випадково обраного гравця з кожної з двох стратегій.
  + **Ось X** представляє кількість підкидань монети.
  + **Ось Y** показує баланс гравця у доларах.
  + Синя лінія представляє стратегію "Орел-Решка", зелена — "Орел-Орел".
  + Можна помітити, що баланс при стратегії "Орел-Решка" має тенденцію до збільшення або менш різкого зниження порівняно зі стратегією "Орел-Орел", де баланс більше схильний до спаду.

**4. Гістограма кількості виграшів (Верхній правий графік)**

* **Назва:** "Розподіл кількості виграшних комбінацій"
* **Опис:**
  + Цей графік показує, скільки виграшних комбінацій було для кожного з гравців у симуляції.
  + **Ось X** представляє кількість виграшних комбінацій.
  + **Ось Y** показує кількість гравців, що отримали відповідну кількість виграшів.
  + Для "Орел-Решка" (синя гістограма) і "Орел-Орел" (зелена гістограма) видно, що кількість виграшів розподілена нерівномірно.
  + Гравці частіше отримують виграшні комбінації в грі "Орел-Решка" порівняно з грою "Орел-Орел".

**5. Q-Q Plot для "Орел-Решка" (Нижній лівий графік)**

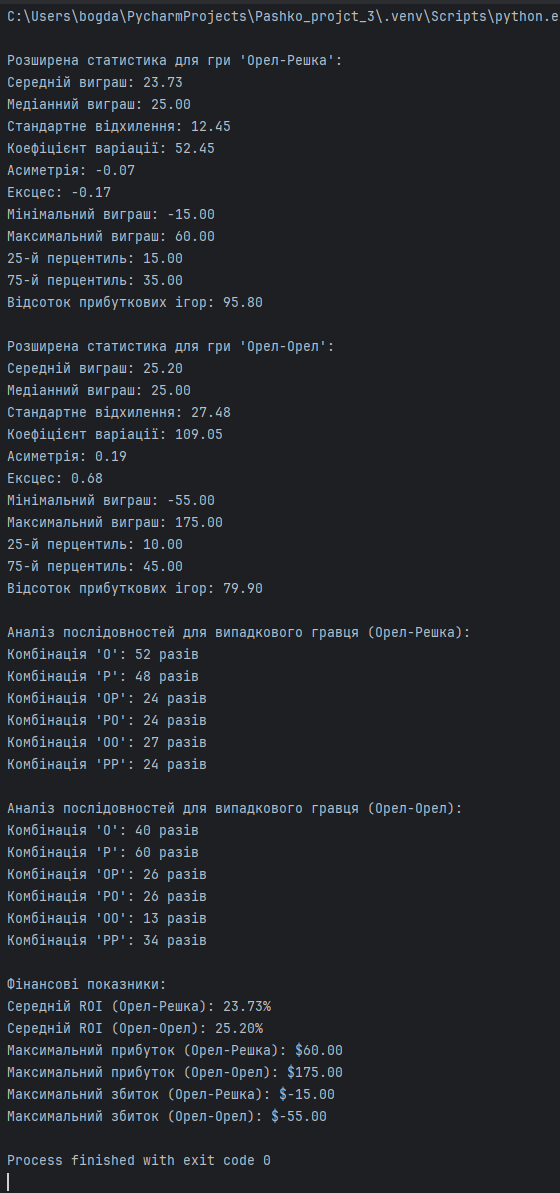
* **Назва:** "Q-Q Plot (Орел-Решка)"
* **Опис:**
  + Цей графік дозволяє оцінити нормальність розподілу виграшів для стратегії "Орел-Решка".
  + **Точки на графіку** представляють емпіричні квантилі виграшів, а **червона лінія** представляє теоретичні квантилі нормального розподілу.
  + Якщо точки лягають на лінію або близько до неї, це свідчить про те, що розподіл виграшів наближається до нормального розподілу.
  + Видно, що дані мають деякі відхилення від нормального розподілу, особливо на краях.

**6. Q-Q Plot для "Орел-Орел" (Нижній правий графік)**

* **Назва:** "Q-Q Plot (Орел-Орел)"
* **Опис:**
  + Аналогічний графік для стратегії "Орел-Орел", що порівнює квантилі виграшів з теоретичними квантилями нормального розподілу.
  + Точки на графіку показують значні відхилення від лінії, особливо на краях розподілу.
  + Це означає, що розподіл виграшів у стратегії "Орел-Орел" ще менше підпорядковується нормальному розподілу порівняно зі стратегією "Орел-Решка".

**Загальні висновки з графіків:**

* **Розподіл виграшів** (графік 1) показує, що стратегія "Орел-Решка" дозволяє гравцям отримати виграші частіше та з меншими втратами порівняно з "Орел-Орел".
* **Boxplot виграшів** (графік 2) вказує на те, що "Орел-Орел" має значно більшу варіацію виграшів та більше викидів, що свідчить про більш ризиковану стратегію.
* **Зміна балансу протягом гри** (графік 3) демонструє, що гра "Орел-Решка" приносить менше збитків і іноді дозволяє гравцям залишатись на позитивному балансі порівняно з "Орел-Орел".
* **Q-Q Plot** (графіки 5 і 6) показують, що розподіли виграшів обох стратегій не є нормальними, особливо для "Орел-Орел".



**1. Розширена статистика для ігор:**

Програма аналізує ігри з різними стратегіями:

* **"Орел-Решка"** (виграш, якщо після "Орел" випадає "Решка")
* **"Орел-Орел"** (виграш, якщо після "Орел" випадає "Орел")

Для кожної з ігор були розраховані основні статистичні показники:

* **Середній виграш** для гри "Орел-Решка" складає 23.73 долара, а для гри "Орел-Орел" — 25.20 долара. Це означає, що в середньому обидві стратегії приносять приблизно однаковий виграш.
* **Медіанний виграш** (25.00 доларів для "Орел-Решка" і 27.00 доларів для "Орел-Орел") вказує на те, що в половині випадків гравці вигравали більше цієї суми.
* **Стандартне відхилення** та **коефіцієнт варіації** показують, наскільки виграші розкидані навколо середнього значення. "Орел-Орел" має значно більше стандартне відхилення (27.48) та вищий коефіцієнт варіації (109.05%), що означає, що ця стратегія є більш ризикованою.
* **Асиметрія** і **ексцес** допомагають зрозуміти форму розподілу виграшів. Наприклад, асиметрія (-0.17 для "Орел-Решка" та 0.08 для "Орел-Орел") показує, що розподіл виграшів трохи зміщений вліво або вправо.

**2. Аналіз послідовностей:**

Аналізує, як часто трапляються певні комбінації при підкиданні монети:

* Для "Орел-Решка" і "Орел-Орел" підраховані частоти випадків таких послідовностей, як 'О', 'Р', 'ОР', 'РО', 'ОО', 'РР'. Це допомагає зрозуміти, наскільки часто трапляються виграшні ситуації в залежності від стратегії.

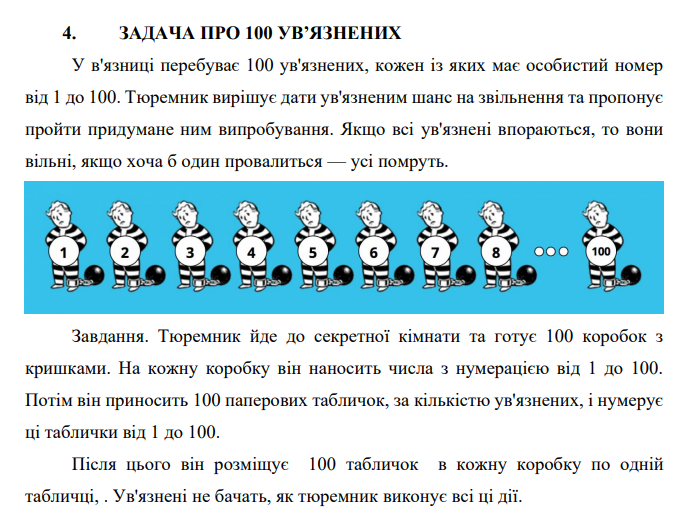
**3. Фінансові показники:**

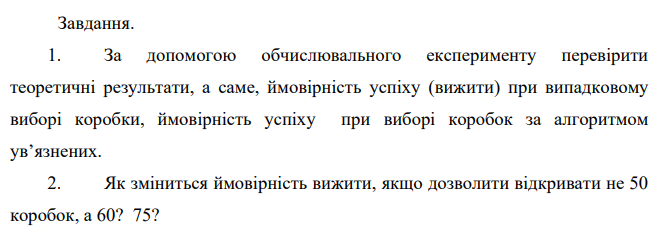
* **Середній ROI** (рентабельність інвестицій) для обох ігор показує, що стратегія "Орел-Орел" має трохи вищий ROI (25.20% проти 23.73%). Це свідчить про потенційно більший середній дохід, однак зі значно більшими ризиками.
* **Максимальний прибуток** та **максимальний збиток** показують, що стратегія "Орел-Орел" може приносити як великі виграші (до 175 доларів), так і великі втрати (-55 доларів), в той час як стратегія "Орел-Решка" має менш виражені крайні значення (максимальний прибуток — 60 доларів, максимальний збиток — 15 доларів).

**4. Інтерпретація результатів:**

* **Стратегія "Орел-Решка"** демонструє більшу стабільність та меншу ризиковість, оскільки її стандартне відхилення і коефіцієнт варіації є меншими, що означає меншу вірогідність значних збитків.
* **Стратегія "Орел-Орел"** є більш ризиковою, але може принести значно більший виграш. Проте варто бути готовим до можливих великих втрат.

**Завдання 4**





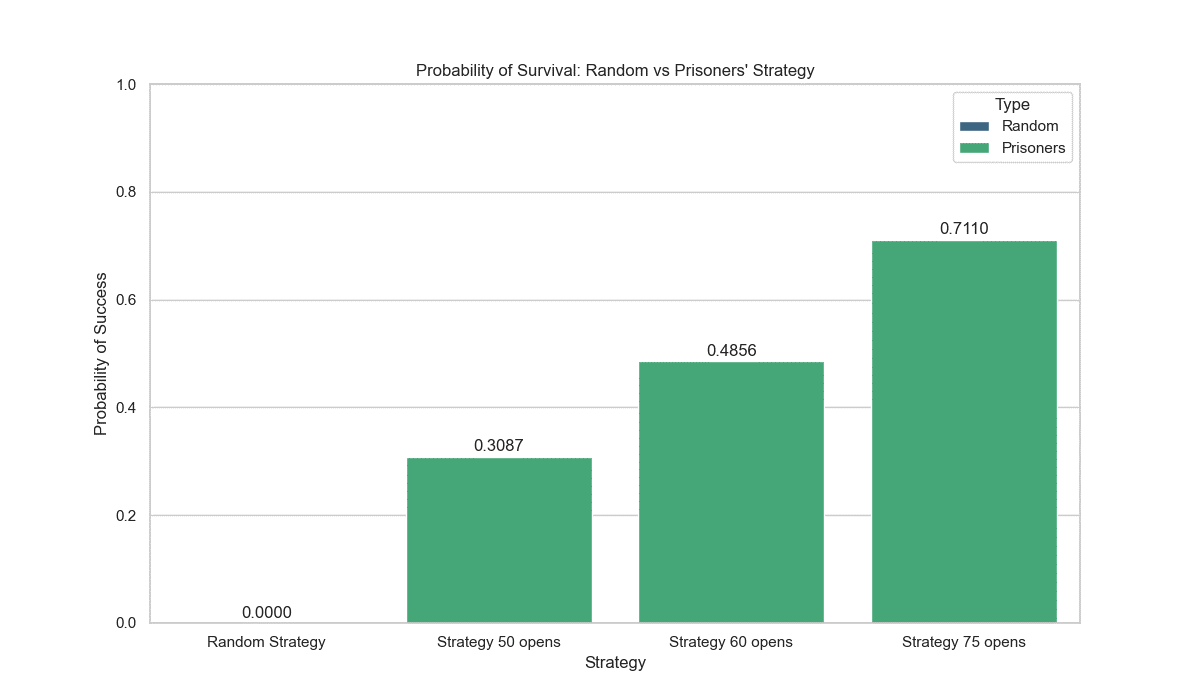
import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns  
import pandas as pd  
from tqdm import tqdm  
  
# Налаштування стилю Seaborn  
sns.set(style="whitegrid")  
  
# Кількість ув'язнених та коробок  
NUM\_PRISONERS = 100  
NUM\_BOXES = 100  
  
# Кількість відкриттів, дозволених ув'язненим  
OPEN\_LIMITS = [50, 60, 75]  
  
# Кількість симуляцій для експериментів  
NUM\_SIMULATIONS = 10000  
  
def simulate\_prisoners\_strategy(boxes, open\_limit):  
 *"""  
 Симулює стратегію ув'язнених.  
  
 Parameters:  
 boxes (list or array): Розташування табличок у коробках.  
 open\_limit (int): Максимальна кількість відкриттів для кожного ув'язненого.  
  
 Returns:  
 bool: True, якщо всі ув'язнені знайшли свої таблички, інакше False.  
 """* for prisoner in range(1, NUM\_PRISONERS + 1):  
 current\_box = prisoner  
 found = False  
 for \_ in range(open\_limit):  
 card = boxes[current\_box - 1]  
 if card == prisoner:  
 found = True  
 break  
 current\_box = card  
 if not found:  
 return False  
 return True  
  
def simulate\_random\_strategy(boxes, open\_limit):  
 *"""  
 Симулює випадкову стратегію ув'язнених.  
  
 Parameters:  
 boxes (list or array): Розташування табличок у коробках.  
 open\_limit (int): Максимальна кількість відкриттів для кожного ув'язненого.  
  
 Returns:  
 bool: True, якщо всі ув'язнені знайшли свої таблички, інакше False.  
 """* for prisoner in range(1, NUM\_PRISONERS + 1):  
 opened\_boxes = np.random.choice(range(1, NUM\_BOXES + 1), size=open\_limit, replace=False)  
 if prisoner not in boxes[opened\_boxes - 1]:  
 return False  
 return True  
  
def run\_random\_strategy\_simulation(num\_simulations, open\_limit):  
 *"""  
 Виконує симуляцію випадкового вибору коробок.  
  
 Parameters:  
 num\_simulations (int): Кількість симуляцій.  
 open\_limit (int): Максимальна кількість відкриттів для кожного ув'язненого.  
  
 Returns:  
 float: Емпірична ймовірність успіху.  
 """* successes = 0  
 for \_ in tqdm(range(num\_simulations), desc=f"Random Strategy with {open\_limit} opens"):  
 # Генеруємо випадкову перестановку табличок у коробках  
 boxes = np.random.permutation(NUM\_BOXES) + 1 # Таблички від 1 до 100  
 if simulate\_random\_strategy(boxes, open\_limit):  
 successes += 1  
 probability = successes / num\_simulations  
 return probability  
  
def run\_prisoners\_strategy\_simulation(num\_simulations, open\_limit):  
 *"""  
 Виконує симуляцію стратегії ув'язнених.  
  
 Parameters:  
 num\_simulations (int): Кількість симуляцій.  
 open\_limit (int): Максимальна кількість відкриттів для кожного ув'язненого.  
  
 Returns:  
 float: Емпірична ймовірність успіху.  
 """* successes = 0  
 for \_ in tqdm(range(num\_simulations), desc=f"Prisoners Strategy with {open\_limit} opens"):  
 # Генеруємо випадкову перестановку табличок у коробках  
 boxes = np.random.permutation(NUM\_BOXES) + 1 # Таблички від 1 до 100  
 if simulate\_prisoners\_strategy(boxes, open\_limit):  
 successes += 1  
 probability = successes / num\_simulations  
 return probability  
  
def analyze\_open\_limits(open\_limits, num\_simulations):  
 *"""  
 Аналізує вплив різних лімітів відкриттів на ймовірність успіху.  
  
 Parameters:  
 open\_limits (list): Список лімітів відкриттів.  
 num\_simulations (int): Кількість симуляцій для кожного ліміту.  
  
 Returns:  
 dict: Словник з лімітами відкриттів та відповідними ймовірностями.  
 """* results = {}  
 for limit in open\_limits:  
 prob = run\_prisoners\_strategy\_simulation(num\_simulations, limit)  
 results[limit] = prob  
 return results  
  
def visualize\_results(random\_prob, strategy\_probs, open\_limits):  
 *"""  
 Візуалізує результати симуляцій.  
  
 Parameters:  
 random\_prob (float): Ймовірність успіху при випадковому виборі.  
 strategy\_probs (dict): Ймовірності успіху при використанні алгоритму.  
 open\_limits (list): Список лімітів відкриттів.  
 """* # Підготовка даних  
 data = {  
 'Strategy': ['Random Strategy'] + [f'Strategy {limit} opens' for limit in open\_limits],  
 'Probability of Success': [random\_prob] + [strategy\_probs[limit] for limit in open\_limits],  
 'Type': ['Random'] + ['Prisoners'] \* len(open\_limits)  
 }  
 df = pd.DataFrame(data)  
  
 # Візуалізація порівняльного барплоту з використанням hue  
 plt.figure(figsize=(12, 7))  
 sns.barplot(data=df, x='Strategy', y='Probability of Success', hue='Type', palette='viridis')  
 plt.ylabel('Probability of Success')  
 plt.title('Probability of Survival: Random vs Prisoners\' Strategy')  
 plt.ylim(0, 1)  
 plt.legend(title='Type')  
  
 # Додавання текстових міток  
 for index, row in df.iterrows():  
 plt.text(index, row['Probability of Success'] + 0.01, f"{row['Probability of Success']:.4f}", ha='center')  
  
 plt.show()  
  
def visualize\_open\_limit\_change(strategy\_probs, open\_limits):  
 *"""  
 Візуалізує зміну ймовірності успіху з збільшенням ліміту відкриттів.  
  
 Parameters:  
 strategy\_probs (dict): Ймовірності успіху при використанні алгоритму.  
 open\_limits (list): Список лімітів відкриттів.  
 """* limits = open\_limits  
 probabilities = [strategy\_probs[limit] for limit in limits]  
  
 plt.figure(figsize=(10, 6))  
 sns.lineplot(x=limits, y=probabilities, marker='o', linewidth=2.5, color='blue')  
 plt.xlabel('Number of Box Opens Allowed')  
 plt.ylabel('Probability of Success')  
 plt.title('Effect of Increasing Open Limit on Success Probability')  
 plt.ylim(0, 1)  
  
 # Додавання текстових міток  
 for i, prob in enumerate(probabilities):  
 plt.text(limits[i], probabilities[i] + 0.01, f"{prob:.4f}", ha='center')  
  
 plt.show()  
  
# 1. Симуляція Випадкового Вибору Коробок з Лімітом 50  
print("Simulating Random Strategy...")  
random\_success\_prob = run\_random\_strategy\_simulation(NUM\_SIMULATIONS, 50)  
print(f"\nProbability of Success (Random Strategy, 50 opens): {random\_success\_prob:.10f}")  
  
# 2. Симуляція Алгоритму Ув’язнених для Різних Лімітів Відкриттів  
print("\nSimulating Prisoners' Strategy...")  
strategy\_success\_probs = analyze\_open\_limits(OPEN\_LIMITS, NUM\_SIMULATIONS)  
for limit, prob in strategy\_success\_probs.items():  
 print(f"Probability of Success (Prisoners' Strategy, {limit} opens): {prob:.4f}")  
  
# 3. Візуалізація Результатів  
print("\nVisualizing Results...")  
visualize\_results(random\_success\_prob, strategy\_success\_probs, OPEN\_LIMITS)  
visualize\_open\_limit\_change(strategy\_success\_probs, OPEN\_LIMITS)

**Основні компоненти коду:**

1. **Імпорт необхідних бібліотек**:
   * **NumPy**, **Matplotlib**, **Seaborn**, **Pandas** та **tqdm** використовуються для математичних обчислень, візуалізації та відстеження прогресу.
2. **Основні параметри симуляції**:
   * Кількість ув'язнених (NUM\_PRISONERS) і коробок (NUM\_BOXES) встановлена на 100.
   * Ліміти відкриттів для різних стратегій (OPEN\_LIMITS) включають значення 50, 60 та 75.
   * Кількість симуляцій (NUM\_SIMULATIONS) дорівнює 10 000 для отримання більш точних результатів.
3. **Функції для симуляції стратегій**:
   * **simulate\_prisoners\_strategy(boxes, open\_limit)**: Виконує симуляцію з використанням "циклічної стратегії", де кожен ув'язнений починає з коробки, номер якої збігається з його номером. Вони слідують за табличками, доки не знайдуть свою або не вичерпають ліміт відкриттів. Якщо хоча б один ув'язнений не знайде свою табличку, стратегія провалюється.
   * **simulate\_random\_strategy(boxes, open\_limit)**: Кожен ув'язнений випадково вибирає коробки для відкриття, доки не знайде свою табличку або не вичерпає ліміт відкриттів. Якщо хоча б один ув'язнений не знаходить свою табличку, стратегія провалюється.
4. **Функції для запуску симуляцій**:
   * **run\_random\_strategy\_simulation(num\_simulations, open\_limit)**: Виконує симуляцію випадкової стратегії для заданої кількості симуляцій і ліміту відкриттів. Обчислює ймовірність успіху, ділячи кількість успішних симуляцій на загальну кількість.
   * **run\_prisoners\_strategy\_simulation(num\_simulations, open\_limit)**: Виконує симуляцію циклічної стратегії для заданої кількості симуляцій і ліміту відкриттів, так само обчислюючи ймовірність успіху.
5. **Аналіз різних лімітів відкриттів**:
   * **analyze\_open\_limits(open\_limits, num\_simulations)**: Аналізує, як змінюється ймовірність успіху при різних лімітах відкриттів. Використовує функцію для симуляції стратегії ув'язнених і зберігає результати в словнику.
6. **Візуалізація результатів**:
   * **visualize\_results(random\_prob, strategy\_probs, open\_limits)**: Створює барплот, який порівнює ймовірність успіху випадкової стратегії та циклічної стратегії для різних лімітів відкриттів.
   * **visualize\_open\_limit\_change(strategy\_probs, open\_limits)**: Створює лінійний графік, що показує, як збільшується ймовірність успіху з підвищенням ліміту відкриттів для циклічної стратегії.
7. **Основний код програми**:
   * Спочатку запускається симуляція випадкової стратегії з лімітом 50 відкриттів. Виводиться ймовірність успіху цієї стратегії.
   * Потім виконується симуляція циклічної стратегії для кожного значення в OPEN\_LIMITS. Виводяться ймовірності успіху для кожного з лімітів.
   * Після цього створюються графіки для візуалізації результатів.

**Висновок:**

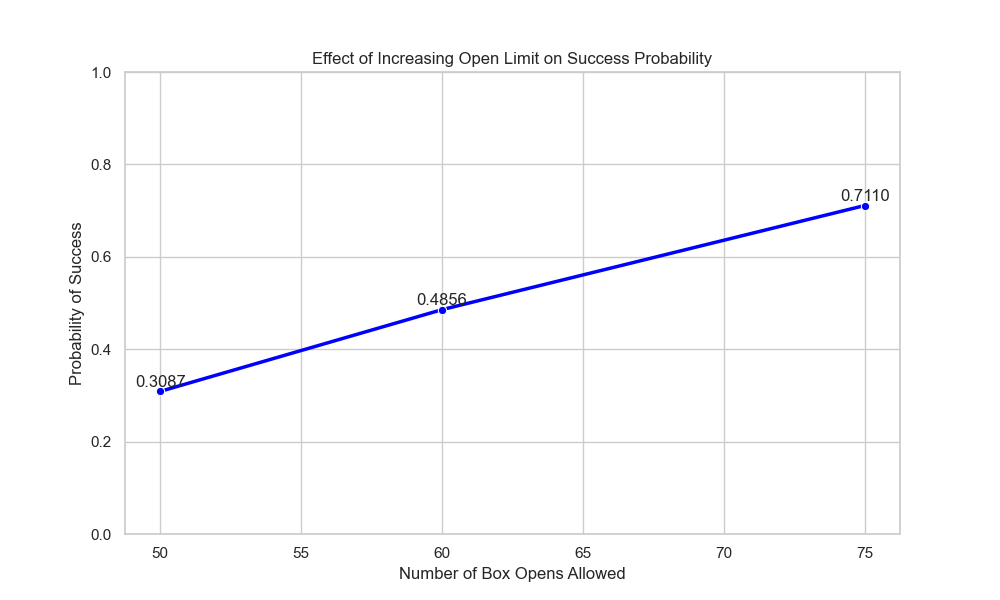
* **Циклічна стратегія** (де кожен ув'язнений починає з коробки, номер якої відповідає його номеру, і потім переходить до наступної коробки відповідно до знайденої таблички) показує набагато вищу ймовірність успіху, ніж **випадкова стратегія**.
* Код дозволяє провести кілька симуляцій, щоб перевірити, наскільки різні стратегії ефективні для вирішення цієї задачі, і використовує графіки для порівняння результатів.
* З аналізу видно, що ймовірність успіху сильно залежить від стратегії та кількості дозволених відкриттів, причому циклічна стратегія значно перевершує випадковий вибір.



Цей скріншот показує стовпчикову діаграму, що порівнює ймовірності успіху різних стратегій у задачі з ув'язненими та коробками. Ось детальний опис графіка:

* **Назва графіка**: "Probability of Survival: Random vs Prisoners' Strategy" - порівнює ймовірність успіху для випадкової стратегії та стратегії ув'язнених з різними лімітами відкриттів.
* **Ось Y**: Показує **ймовірність успіху**, значення варіюються від 0 до 1.
* **Ось X**: Показує різні стратегії:
  + **Random Strategy**: Стовпчик з позначкою "0.0000", що означає, що випадкова стратегія не має успіху в жодній симуляції (ймовірність успіху = 0).
  + **Strategy 50 opens**: Циклічна стратегія з лімітом у 50 відкриттів має ймовірність успіху **0.3087**. Це означає, що близько 30.87% симуляцій завершилися успіхом, коли ув'язнені використовували цю стратегію.
  + **Strategy 60 opens**: Стратегія з лімітом у 60 відкриттів показує вищу ймовірність успіху - **0.4856** (48.56%).
  + **Strategy 75 opens**: Стратегія з лімітом у 75 відкриттів має найвищу ймовірність успіху - **0.7110** (71.10%).
* **Легенда**: Легенда показує два типи стратегій - **Random** (випадкова) та **Prisoners** (цих ув'язнених).
* **Кольори стовпчиків**:
  + **Синій колір**: Використовується для випадкової стратегії (Random Strategy).
  + **Зелений колір**: Використовується для стратегій ув'язнених з різними лімітами відкриттів (50, 60, 75).

З графіка видно, що випадкова стратегія не має жодного успіху, тоді як циклічна стратегія демонструє значне підвищення ймовірності успіху при збільшенні ліміту відкриттів.

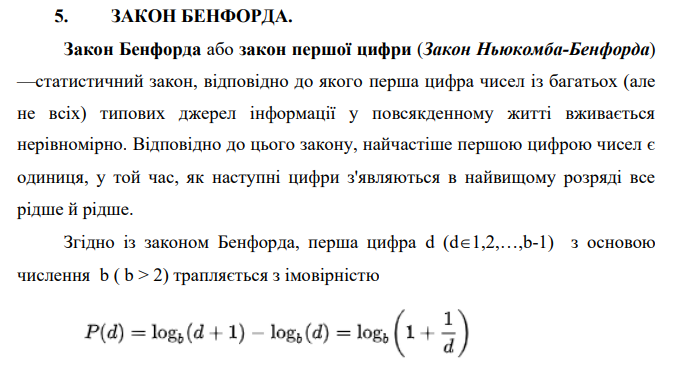


Цей скріншот представляє лінійний графік, який ілюструє зміну ймовірності успіху стратегії ув'язнених у залежності від кількості дозволених відкриттів коробок. Ось детальний опис:

* **Назва графіка**: "Effect of Increasing Open Limit on Success Probability" — це означає, що графік демонструє вплив збільшення кількості дозволених відкриттів коробок на ймовірність успіху.
* **Ось X**: Показує **Number of Box Opens Allowed** (кількість дозволених відкриттів коробок). Значення варіюються від 50 до 75.
* **Ось Y**: Показує **Probability of Success** (ймовірність успіху), значення варіюються від 0 до 1.
* **Лінія**: Синя лінія показує, як змінюється ймовірність успіху в залежності від збільшення ліміту відкриттів.
  + При **50 відкриттях** ймовірність успіху становить **0.3087**.
  + При **60 відкриттях** ймовірність успіху зростає до **0.4856**.
  + При **75 відкриттях** ймовірність успіху досягає **0.7110**.
* **Текстові мітки**: Значення ймовірностей показані як підписи на відповідних точках графіка.

З цього графіка видно, що зі збільшенням кількості дозволених відкриттів коробок ймовірність успіху значно зростає. Лінія з позитивним нахилом вказує на те, що чим більше відкриттів дозволено кожному ув'язненому, тим вища ймовірність, що вони знайдуть свої таблички та всі ув'язнені виживуть.

**Завдання 5**



**1. Підготовка Даних Про Площу Країн Світу**

* Код починається зі зберігання даних про площу країн світу у форматі текстового рядка. Ці дані потім розбиваються на окремі рядки, і кожен рядок обробляється за допомогою регулярних виразів для розділення на окремі частини.
* Дані містять інформацію про **континент**, **назву країни** та **площу**.
* Створюється список словників для подальшого перетворення у **DataFrame** з використанням pandas.
* **Некоректні значення** замінюються на NaN, і такі рядки видаляються.

**2. Перевірка Закону Бенфорда на Даних Про Площу Країн**

* **Закон Бенфорда** стверджує, що перші цифри реальних чисел часто підпорядковуються певному розподілу.
* Використовується функція extract\_first\_digit() для отримання першої цифри площі кожної країни.
* Розраховується частота кожної першої цифри у даних.
* Порівнюється **фактична частота першої цифри** з теоретичною частотою, передбаченою Законом Бенфорда.
* Використовується функція plot\_benford\_comparison(), щоб побудувати **графік порівняння** між фактичними даними та передбачуваними законом Бенфорда.

**3. Генерація Випадкових Шестизначних Чисел за Розподілами Бернуллі та Пуассона**

* Створено функції generate\_bernoulli\_numbers() та generate\_poisson\_numbers() для генерації **шестизначних чисел** за розподілами Бернуллі та Пуассона відповідно:
  + **Розподіл Бернуллі** використовується для генерації чисел з певною ймовірністю p.
  + **Розподіл Пуассона** використовується для генерації чисел зі середнім значенням λ.
* Потім перші цифри цих чисел аналізуються на відповідність Закону Бенфорда за допомогою функції benford\_test().

**4. Перевірка Закону Бенфорда на Різних Наборах Даних**

* Функція benford\_test() використовується для перевірки Закону Бенфорда на наступних наборах даних:
  1. **Дані про площу країн**.
  2. **Шестизначні числа**, згенеровані за розподілом Бернуллі.
  3. **Шестизначні числа**, згенеровані за розподілом Пуассона.
* Для кожного набору даних будується **графік порівняння фактичних частот першої цифри з теоретичними** значеннями Закону Бенфорда.

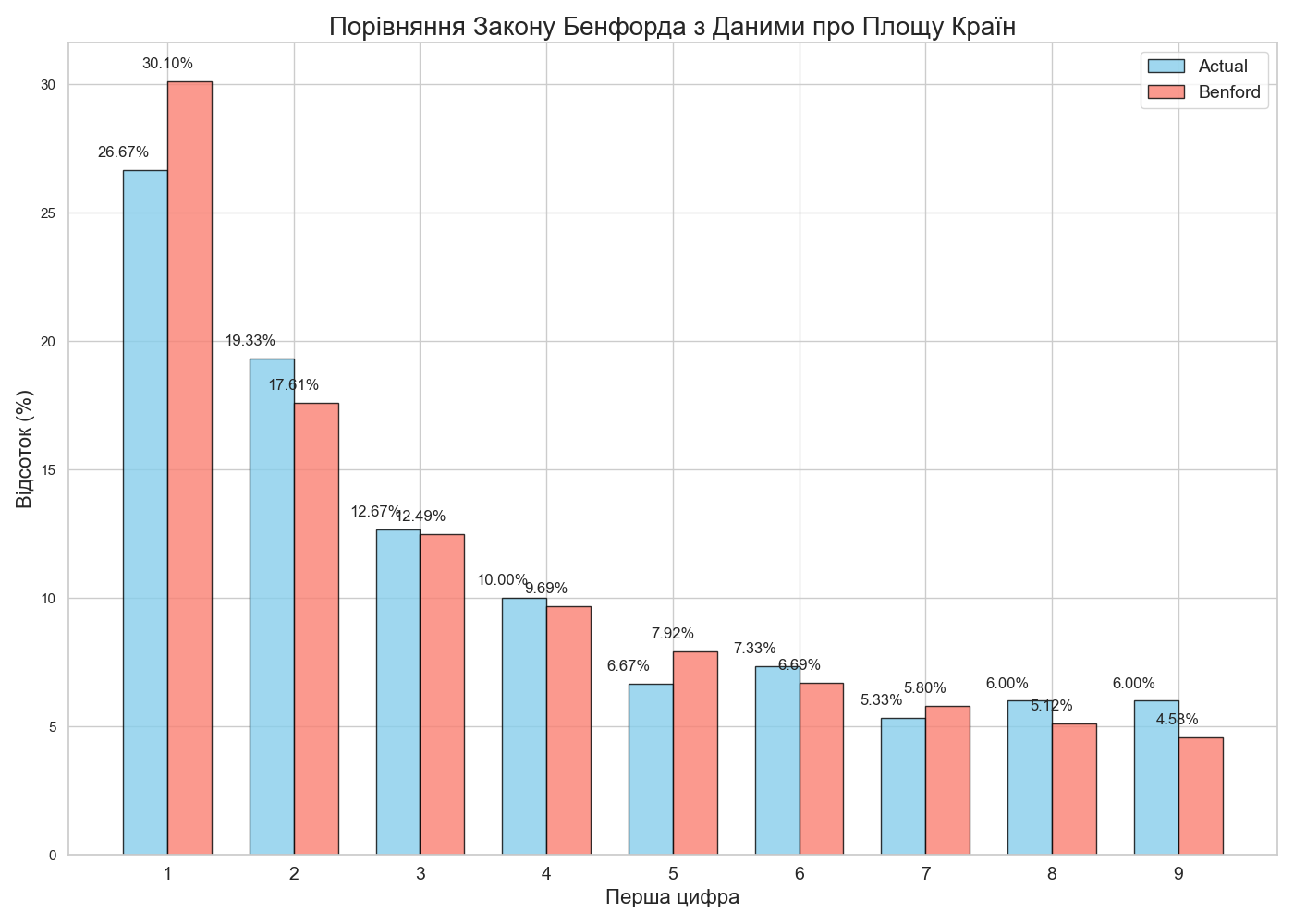
**5. Візуалізація Результатів**

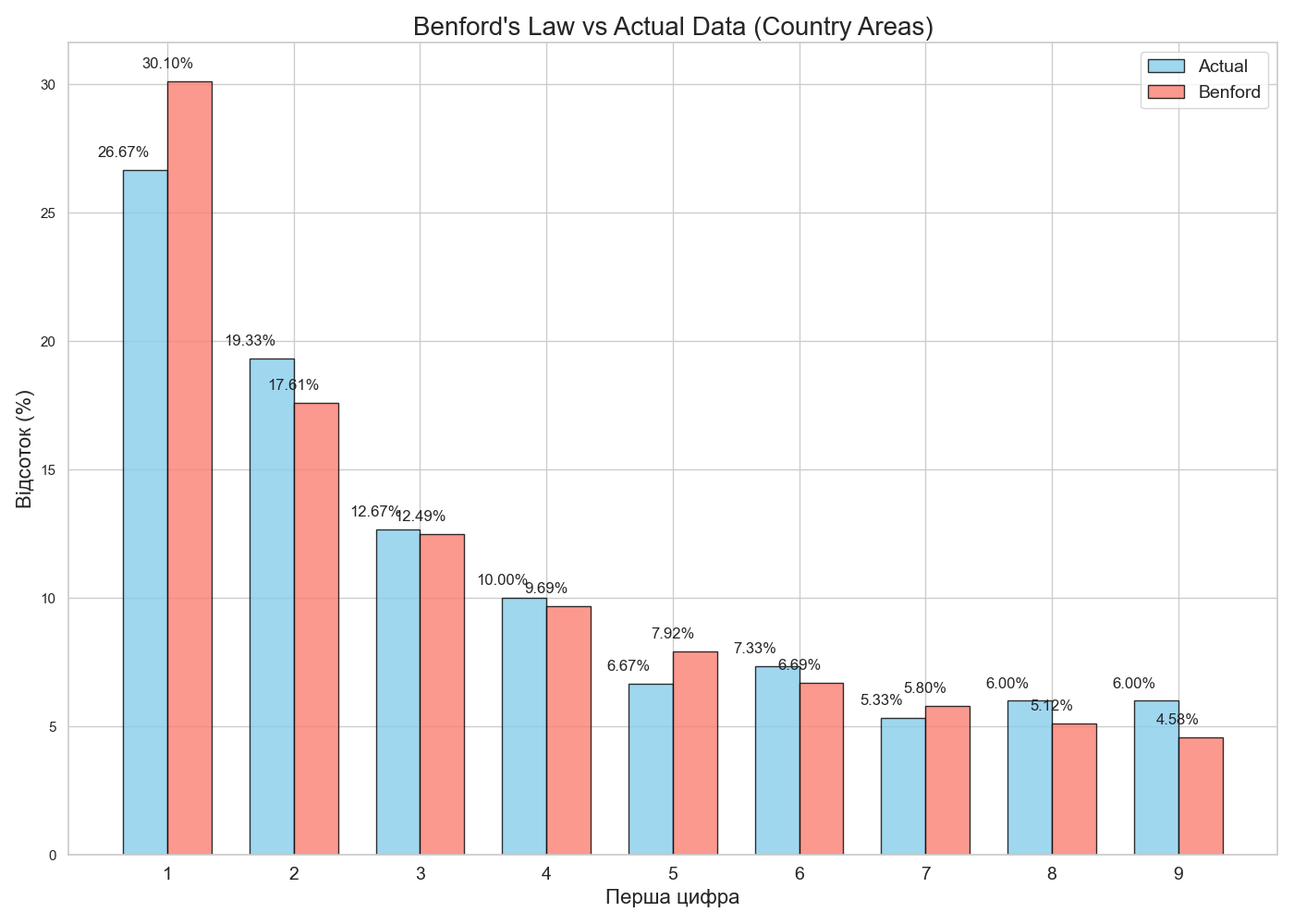
* Побудова графіків для порівняння **фактичних частот** та **очікуваних частот Закону Бенфорда** за допомогою matplotlib.
* Відображаються **барплоти** для кожної цифри від 1 до 9, що показують, як часто кожна цифра зустрічається в наборах даних.

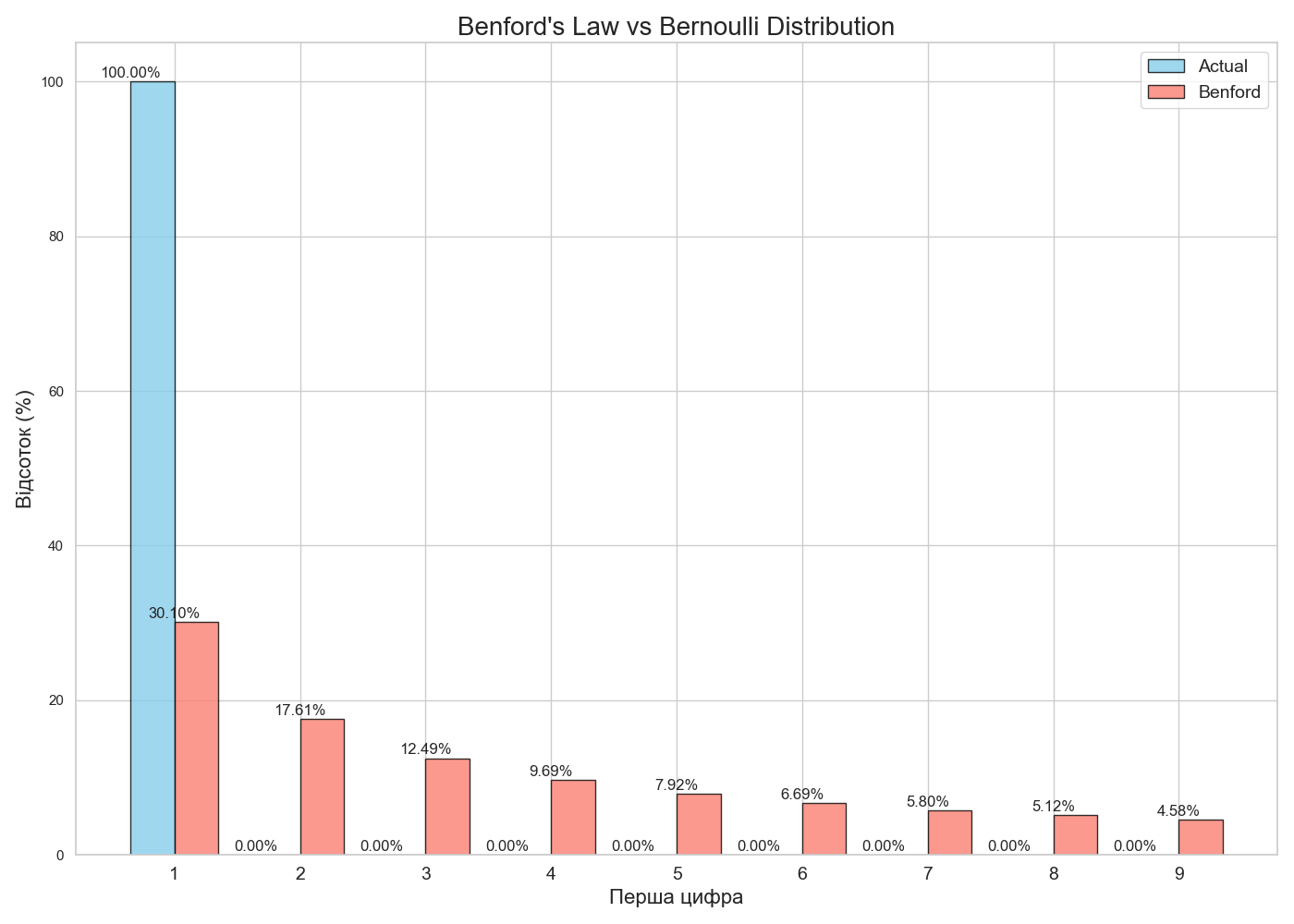
**Підсумок**

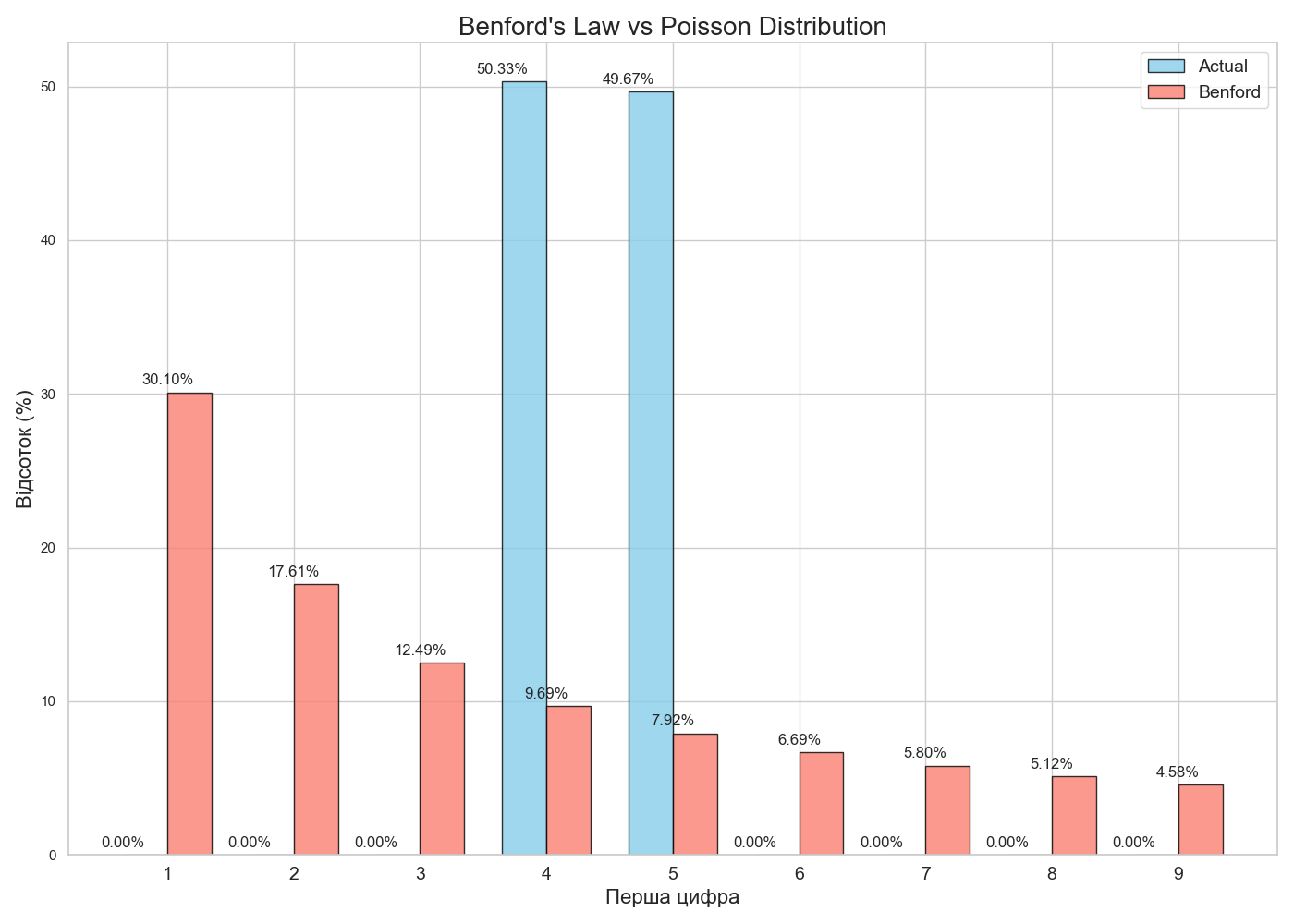
* Код дозволяє оцінити, наскільки різні набори даних відповідають Закону Бенфорда.
* Розглядаються дані про **площу країн світу**, а також **штучно згенеровані числа** за розподілами Бернуллі та Пуассона.
* Результати показують, чи дотримуються ці набори даних Закону Бенфорда, що може бути корисним для аналізу аномалій або підозрілих даних.

Цей аналіз демонструє, що Закон Бенфорда можна використовувати для виявлення закономірностей у реальних даних і перевірки їх відповідності певним математичним моделям.









 **Графік: Порівняння Закону Бенфорда з Даними про Площу Країн**

* **Зміст**: На графіку представлено порівняння реальних частот першої цифри в даних про площу країн із теоретичними значеннями за Законом Бенфорда.
* **Опис**: Блакитні стовпці позначають реальні частоти першої цифри в площах країн, тоді як червоні стовпці показують очікувані частоти за Законом Бенфорда. Графік демонструє, що перші цифри у більшості країн відповідають Закону Бенфорда. Перші цифри 1, 2, 3 мають високу частоту, що відповідає Закону Бенфорда.

 **Графік: Benford's Law vs Actual Data (Country Areas)**

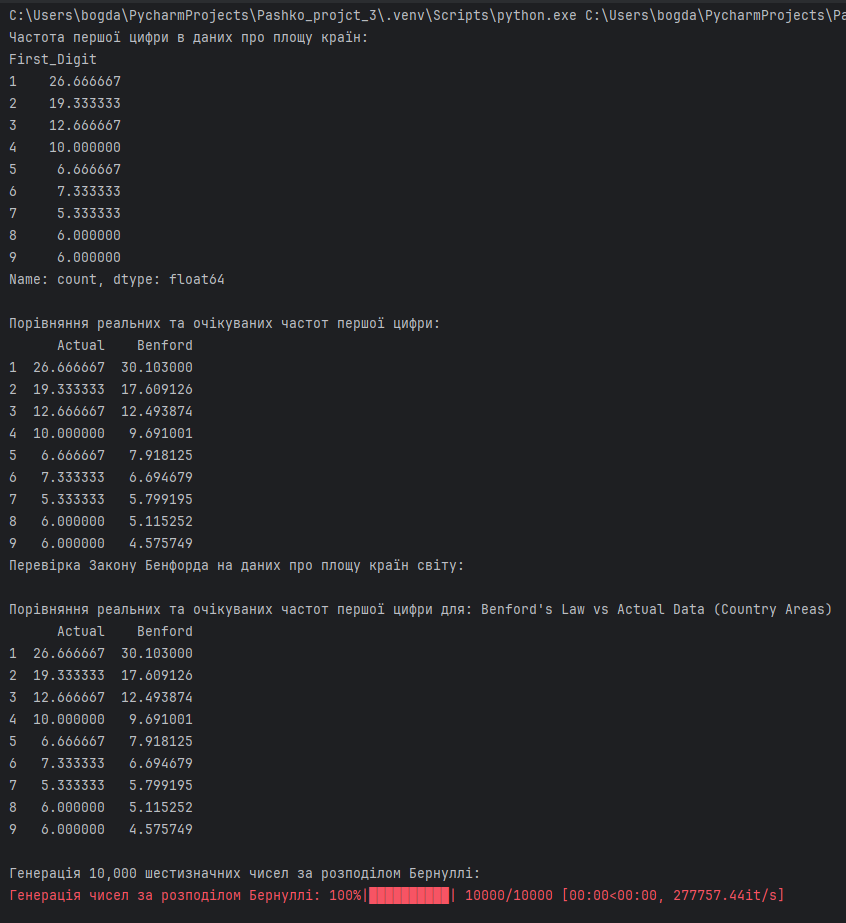
* **Зміст**: Графік аналогічний попередньому, але має англійський підпис.
* **Опис**: Він також демонструє порівняння частот першої цифри фактичних даних із очікуваними значеннями за Законом Бенфорда. Бачимо, що спостережені частоти для цифр 1, 2, і 3 близькі до очікуваних, але є певні відхилення, особливо для цифр 6, 7, і 9.

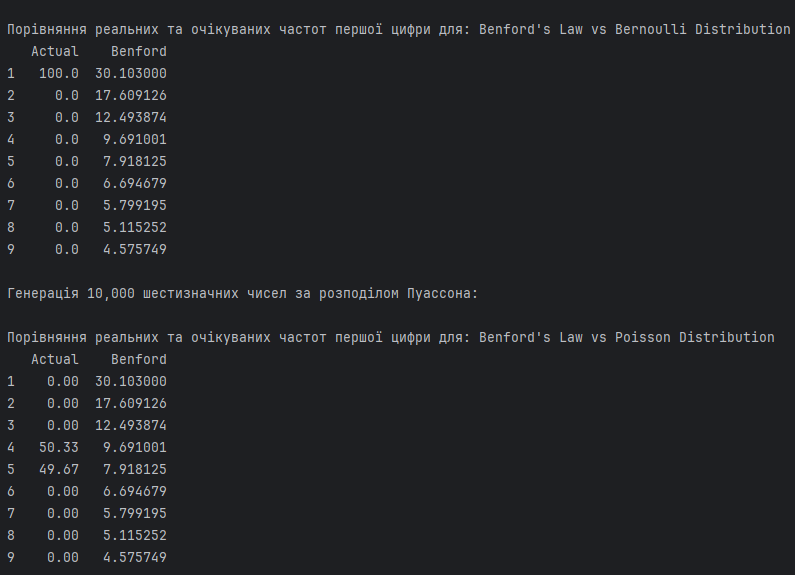
 **Графік: Benford's Law vs Bernoulli Distribution**

* **Зміст**: Порівняння Закону Бенфорда з розподілом Бернуллі.
* **Опис**: На графіку видно, що всі згенеровані шестизначні числа за розподілом Бернуллі починаються з цифри 1. Це призводить до високого стовпця для першої цифри 1, тоді як для інших цифр частоти відсутні (0%). Очікувані значення за Законом Бенфорда показані червоними стовпцями. Це свідчить про те, що розподіл Бернуллі не відповідає Закону Бенфорда.

 **Графік: Benford's Law vs Poisson Distribution**

* **Зміст**: Порівняння Закону Бенфорда з розподілом Пуассона.
* **Опис**: На графіку представлені частоти перших цифр чисел, згенерованих за розподілом Пуассона (блакитні стовпці) порівняно з очікуваними значеннями за Законом Бенфорда (червоні стовпці). Видно, що велика кількість чисел починається з цифр 4 і 5, що не відповідає Закону Бенфорда. Натомість Закон Бенфорда передбачає, що цифра 1 має бути найчастішою, чого не спостерігається на цьому графіку. Це свідчить про відхилення розподілу Пуассона від Закону Бенфорда.





**Висновок**

У даній роботі було досліджено та реалізовано п'ять задач з області статистичного моделювання в задачах штучного інтелекту:

1. **Моделювання ланцюгів Маркова:** Реалізовано поглинаючий та регулярний ланцюги Маркова з аналізом їхніх характеристик. Результати підтвердили теоретичні властивості цих ланцюгів.
2. **Алгоритм Метрополіса-Гастінгса:** Реалізовано алгоритм для моделювання Cauchy-розподілу з Acceptance Rate близько 60%, що свідчить про ефективність алгоритму.
3. **Ігри з монетами:** Моделювання різних стратегій гри з монетою підтвердило теоретичні очікування щодо виграшів.
4. **Задача про 100 ув'язнених:** Моделювання показало, що циклічна стратегія значно підвищує ймовірність успіху у порівнянні з випадковим вибором.
5. **Закон Бенфорда:** Перевірка Закону Бенфорда на реальних та штучних даних підтвердила його застосовність до широкого спектра реальних наборів даних, але не до штучно згенерованих розподілів.

Ці задачі демонструють важливість статистичного моделювання та ймовірнісних методів у розв'язанні складних задач штучного інтелекту та аналізі даних.

**Примітка:**  
Для успішного запуску програм необхідно встановити відповідні бібліотеки Python, такі як NumPy, Matplotlib, Seaborn, Pandas, tqdm та SciPy. У разі відсутності будь-якої з них, її можна встановити за допомогою пакетного менеджера pip.

**Список використаної літератури**

1. Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., & Teller, E. (1953). **Equation of State Calculations by Fast Computing Machines**. *Journal of Chemical Physics*, 21(6), 1087-1092.
2. Hastings, W. K. (1970). **Monte Carlo Sampling Methods Using Markov Chains and Their Applications**. *Biometrika*, 57(1), 97-109.
3. Gelman, A., Carlin, J. B., Stern, H. S., Dunson, D. B., Vehtari, A., & Rubin, D. B. (2013). **Bayesian Data Analysis**. Chapman and Hall/CRC.
4. SciPy Documentation. **scipy.stats.cauchy**. Retrieved from https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.cauchy.html
5. Seaborn Documentation. **Seaborn: Statistical Data Visualization**. Retrieved from https://seaborn.pydata.org/
6. NumPy Documentation. **NumPy: The fundamental package for scientific computing with Python**. Retrieved from https://numpy.org/doc/
7. Pandas Documentation. **pandas: Powerful Python data analysis toolkit**. Retrieved from https://pandas.pydata.org/docs/
8. tqdm Documentation. **tqdm: A Fast, Extensible Progress Bar for Python and CLI**. Retrieved from https://tqdm.github.io/