SPA Assignment 03 Report

Vishal Kumar Maurya 2022580

Q1

Approach

- **Let's understand transition matrix:**
  - Each row of P represents a transition from one state to another with their associated probabilities.
  - **For example**, the first row read off that if the current state were r:
    - There's 50 percent chances of sticking in r.
    - The possibility of getting w 30%
    - That is, with probability 20% it will become s.
    - No room to maneuver towards e.
- **Simulation Method**:
  - The Markov chain simulation starts from X_0=r.
  - It uses the current state at every step to produce the next state according to the transition probability from ***P***.
  - The subsequent states can be calculated using random sampling if the probabilities are there.
- **Implementation Standards**:
  - The simulation was done in Python using random and matplotlib library.
  - To get the next state, **random.choices()** was used, which accepts weighted probabilities.
  - A time sequence of states is recorded over a period of N=100 steps.
- **Visualization**:
  - A **line graph** is drawn to depict the trend of states over the 100 moves.
  - Figure below it is a bar graph showing the frequency distribution of states over all trajectories.

Results

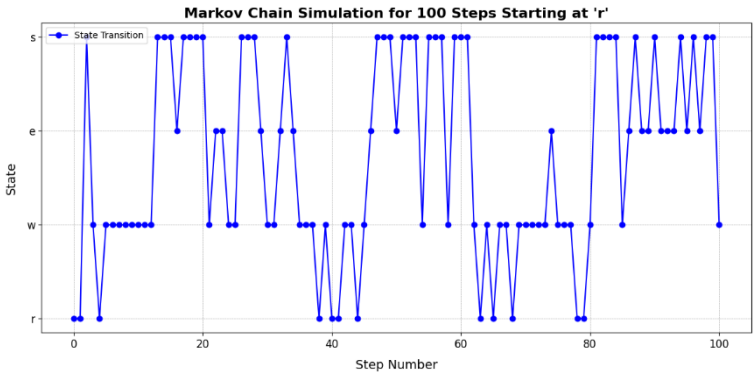
- **State Orders**:
  - The following sequence of states were generated by the Markov chain through 100 iterations.
    r -> w -> w -> r -> s -> s -> e -> s -> s -> w -> ...
  - In ***P***, randomness in transitions is specified by the sequence based on probabilities.
- **Frequency Distribution**:
  - After 100 steps, the frequency of each state was approximately(can differ if ran again):
    - **r (read)**: 12
    - **w (write)**: 38
    - **e (email)**: 17
    - **s (surf)**: 34
  - More frequent occurrences of ***w*** and ***s*** represents a higher likelihood of self-loops of these states in the Transition Table Matrix (TTM).
- **Visuals**:
  - The **Line Graph** depicted the process of changes in the states along time axis, making it clear how the process evolves over time.
  - The **bar chart** summarizes the frequency of occurrence of each state and gave an implication that ***w*** and **s** were most visited states.

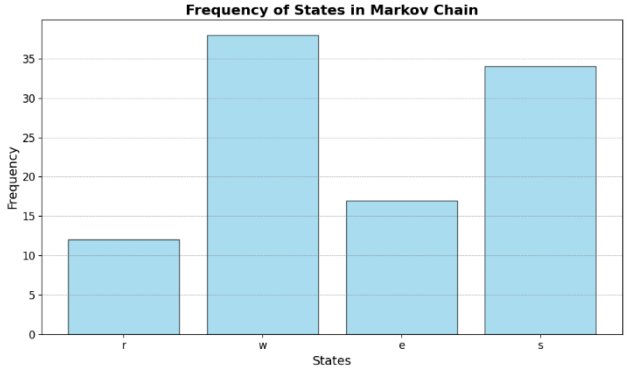
Analysis

- **Convergence**:
  - Over an enormous sequence of steps it is supposed that the Markov chain will converge and represents the stationary distribution, although this was not calculated here..
  - High frequencies with ***w*** and ***s*** denote the existence of high self-loop and high transition probabilities.
- **Practical Applications**:
  - This model can make some real-life applications, for example, the way people alternate between different activities when writing a paper. Surfing of the Internet ***s*** and writing ***w*** are very common activities which are in line with observed frequencies. However, this may differ at another run of code.
- **Randomness**:
  - The intrinsic randomness inherent in the simulation ensures that no two simulations produced trajectories that are alike yet yield comparable trends.

Conclusion

The simulation of the Markov chain yielded rich results concerning task switching behaviors throughout the paper writing process. All the results agree with these presented in the transition probabilities of ***P***. The graphical representation underlines the temporal transitions and the frequency distribution, thus aiding better comprehension of the dynamics of the system.





Q2

Approach

$$P = \begin{bmatrix} 0.7 & 0.2 & Null \\ 0.4 & Null & 0.0 \\ 0.0 & 1.0 & 0.0 \end{bmatrix}$$

- **Completing the Transition Matrix**
  - Each row of **P** must sum to 1. Missing probabilities were calculated as:
    - Row 0: 1−(0.7+0.2)=0.1
    - Row 1: 1−(0.4+0.0)= 0.6
    - Row 2: Fully defined hence, no missing values.

The completed Matrix is:

$$P = \begin{bmatrix} 0.7 & 0.2 & 0.1 \\ 0.4 & 0.6 & 0.0 \\ 0.0 & 1.0 & 0.0 \end{bmatrix}$$

- **Computing Matrix Powers**
  - Using matrix multiplication, we computed:
    - $\mathbf{P^{10}}$: Transition matrix after 10 steps.
    - $\mathbf{P^{20}}$: Transition matrix after 20 steps.
    - $\mathbf{P^{50}}$: Transition matrix after 50 steps.
- **Visualizing Convergence of all values**
  - The value of a_{00} (top-left element of $\mathbf{P^n}$) was extracted at each iteration (from 1 to 50).
  - The values at columns become approximately equal as we proceed the iteration.
  - Results were plotted to visualize the convergence behavior.

Key Results

$$P = \begin{bmatrix} 0.700000 & 0.200000 & 0.100000 \\ 0.400000 & 0.600000 & 0.000000 \\ 0.000000 & 1.000000 & 0.000000 \end{bmatrix}$$

$$P^2 = \begin{bmatrix} 0.570000 & 0.360000 & 0.070000 \\ 0.520000 & 0.440000 & 0.040000 \\ 0.400000 & 0.600000 & 0.000000 \end{bmatrix}$$

$$P^3 = \begin{bmatrix} 0.543000 & 0.400000 & 0.057000 \\ 0.540000 & 0.408000 & 0.052000 \\ 0.520000 & 0.440000 & 0.040000 \end{bmatrix}$$

$$P^5 = \begin{bmatrix} 0.540310 & 0.405680 & 0.054010 \\ 0.540760 & 0.405120 & 0.054120 \\ 0.54120 & 0.404800 & 0.054000 \end{bmatrix}$$

$$P^{10} = \begin{bmatrix} 0.540540 & 0.405405 & 0.054054 \\ 0.540540 & 0.405405 & 0.054054 \\ 0.540540 & 0.405405 & 0.054053 \end{bmatrix}$$

$$P^{20} = \begin{bmatrix} 0.540540 & 0.405405 & 0.054054 \\ 0.540540 & 0.405405 & 0.054054 \\ 0.540540 & 0.405405 & 0.054054 \end{bmatrix}$$

$$P^{35} = \begin{bmatrix} 0.540540 & 0.405405 & 0.054054 \\ 0.540540 & 0.405405 & 0.054054 \\ 0.540540 & 0.405405 & 0.054054 \end{bmatrix}$$

$$P^{50} = \begin{bmatrix} 0.540540 & 0.405405 & 0.054054 \\ 0.540540 & 0.405405 & 0.054054 \\ 0.540540 & 0.405405 & 0.054054 \end{bmatrix}$$

- As $\mathbf{n \to \infty}$, the columns of $\mathbf{P^n}$ become **identical**, indicating that the Markov chain has reached its **steady-state**:

$$\begin{bmatrix} 0.540540 & 0.405405 & 0.054054 \\ 0.540540 & 0.405405 & 0.054054 \\ 0.540540 & 0.405405 & 0.054054 \end{bmatrix}$$

Facts

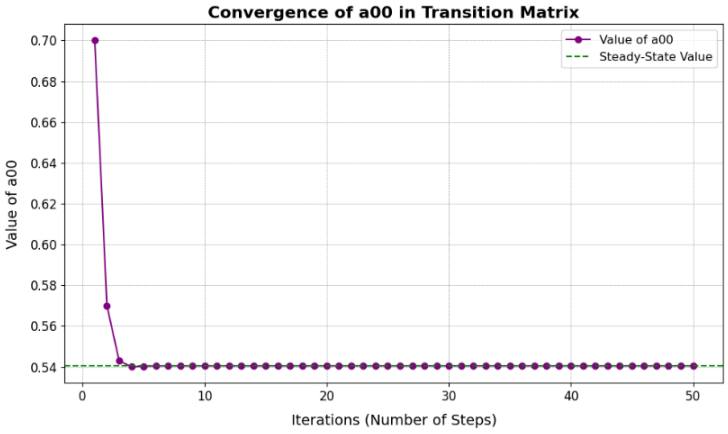
- **Irreducibility**: Every state must be reachable from every other state, either directly or through a sequence of transitions. The TPM here has the sequence of steps to move from one state to other therefor the MC is irreducible.
- **Aperiodicity**: The system must not be stuck in cycles with a fixed periodicity. Since the TPM depicts self-loops for every state therefore, its aperiodic.
- **Hence,** the MC is **Ergodic**.
- **I**t shows steady state distribution with

$$\pi = \pi\, \rho$$
$$where\ \pi = [0.540540 \quad 0.405405 \quad 0.054054]$$

This follows:

- The system is in state 0 approximately **54.05%** of the time.
- It is in state 1 around **40.54%** of the time.
- It is in state 2 for only **5.41%** of the time.

- **Limiting Behavior**: For any initial state **X0**, the probability distribution after many steps will approach this steady-state distribution. This is evident from the convergence of the rows of $\mathbf{P^n}$ (for large **n**) to the same steady-state probabilities.
- **Uniqueness**: Given that the transition matrix **P** is **irreducible** (every state is accessible from every other state) and **aperiodic** (states do not have fixed periodic cycles), the steady-state distribution is unique and guaranteed to exist.



Q-1 Python Code

```
import random
import matplotlib.pyplot as plt

P = [
#     r    w    e    s
    [0.5, 0.3, 0.0, 0.2],  # r
    [0.2, 0.5, 0.1, 0.2],  # w
    [0.1, 0.3, 0.3, 0.3],  # e
    [0.0, 0.2, 0.3, 0.5],  # s
]

# States of the Markov Chain
states = ["r", "w", "e", "s"]


def simulate_markov_chain(start_state, N):
    current_state = start_state  # Initial state X0
    state_history = [current_state]
    for _ in range(N):
        current_index = states.index(current_state)
        transition_probabilities = P[current_index]
        next_state = random.choices(states,
weights=transition_probabilities, k=1)[0]
        state_history.append(next_state)
        current_state = next_state
    return state_history


N = 100  # No. of steps
initial_state = "r"  # Starting state X0
trajectory = simulate_markov_chain(initial_state, N)

# Map states to numerical values for plotting
state_to_num = {state: i for i, state in
enumerate(states)}
num_trajectory = [state_to_num[state] for state in
trajectory]
# Plot the trajectory
plt.figure(figsize=(12, 6))
plt.plot(
    range(len(trajectory)),
    num_trajectory,
    marker="o",
    linestyle="-",
    color="b",
    label="State Transition",
)
plt.title(
    f"Markov Chain Simulation for {N} Steps Starting
at '{initial_state}'",
    fontsize=16,
    fontweight="bold",
)
plt.xlabel("Step Number", fontsize=14, labelpad=10)
plt.ylabel("State", fontsize=14, labelpad=10)
plt.yticks(ticks=list(state_to_num.values()),
labels=states, fontsize=12)
plt.xticks(fontsize=12)
plt.grid(color="gray", linestyle="--", linewidth=0.5,
alpha=0.7)
plt.legend(fontsize=12)
plt.tight_layout()
plt.show()
# Plot the bar chart
state_frequencies = {state: trajectory.count(state)
for state in states}
plt.figure(figsize=(10, 6))
plt.bar(
    state_frequencies.keys(),
    state_frequencies.values(),
    color="skyblue",
    edgecolor="black",
    alpha=0.7,
)
plt.title("Frequency of States in Markov Chain",
fontsize=16, fontweight="bold")
plt.xlabel("States", fontsize=14)
plt.ylabel("Frequency", fontsize=14)
plt.grid(axis="y", color="gray", linestyle="--",
linewidth=0.5, alpha=0.7)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.tight_layout()
plt.show()
```

Q-2 Python Code

```python
import matplotlib.pyplot as plt
# JetBrainsMono Nerd Font Mono,'JetBrainsMono Nerd Font',JetBrainsMonoNL NFP
def complete_transition_matrix(matrix):
    for i in range(len(matrix)):
        row = matrix[i]
        row_sum = 0
        missing_index = -1
        for j in range(len(row)):
            if row[j] is None:
                if missing_index == -1:
                    missing_index = j
                else:
                    raise ValueError(
                        "More than one missing value in a row is not supported."
                    )
            else:
                row_sum += row[j]
        if missing_index != -1:
            matrix[i][missing_index] = 1 - row_sum
    return matrix

# i am assuming that there is only square matrices for matrix multiplication
def input_matrix():
    n = int(input("Enter the size of the matrix (n x n): "))
    matrix = []
    print(
        "\nInstructions:- \nEnter the matrix row by row. Use 'none' (without quotes) for missing values."
    )
    for i in range(n):
        row = input(
            f"Enter row {i + 1} (space-separated, e.g., '0.8 0.1 None'): "
        ).split()
        row = [None if value.lower() == "none" else float(value) for value in row]
        matrix.append(row)
    return matrix

# Note/ I have Added two matrices you should comment any one of it to use the other one.

P_partial = [
    [0.7, 0.2, None],  # Row 0, missing value
    [0.4, None, 0.0],  # Row 1, missing value
    [0.0, 1.0, 0.0],  # Row 2, no missing values
]

# P_partial = [
#     [0.5, None, 0.0, 0.2],  # Row 0, missing value
#     [None, 0.5, 0.1, 0.2],  # Row 1, missing value
#     [None, 0.3, 0.3, 0.0],  # Row 3, missing values
#     [0.0, 0.2, 0.3, None],  # Row 4, missing values
# ]

# Main code
print("Enter 0 if you want to use the Question's Matrix.")
print("Enter 1 if you want to Enter the Matrix values.\n")
taken_input = int(input("Enter Input Value: "))
if taken_input == 0:
    completed_matrix = complete_transition_matrix(P_partial)
elif taken_input == 1:
    print("Manual Transition Matrix Input")
    P_partial = input_matrix()
    completed_matrix = complete_transition_matrix(P_partial)

print("\nCompleted Transition Matrix:")
for row in completed_matrix:
    print(row)

# Part 02 Initialised
print("Part 02 Initialised")

def multiply_matrices(A, B):
    n = len(A)
    result = [[0 for _ in range(n)] for _ in range(n)]
    for i in range(n):
        for j in range(n):
            for k in range(n):
                result[i][j] += A[i][k] * B[k][j]
    return result

def compute_matrix_power(P, power):
    n = len(P)
    result = [[1 if i == j else 0 for j in range(n)] for i in range(n)]
    for _ in range(power):
        result = multiply_matrices(result, P)
    return result
```

```
# Storing a00 values to observe convergence at each
step from 0 to 50
iterations = list(range(1, 51))
a00_values = []

for step in iterations:
    P_power = compute_matrix_power(P_partial, step)
    if step == 10:
        print("\nP^10 (Transition Matrix after 10
steps):")
        for row in P_power:
            print(row)
    elif step == 20:
        print("\nValue of a00 in P^10:",
P_power[0][0])
        print("\nP^20 (Transition Matrix after 20
steps):")
        for row in P_power:
            print(row)
        print("\nValue of a00 in P^20:",
P_power[0][0])
    elif step == 50:
        print("\nP^50 (Transition Matrix after 50
steps):")
        for row in P_power:
            print(row)
        print("\nValue of a00 in P^50:",
P_power[0][0])
    a00_values.append(P_power[0][0])  # First row,
first column value

# Plotting the values of a00
plt.figure(figsize=(10, 6))
plt.plot(
    iterations,
    a00_values,
    marker="o",
    linestyle="-",
    color="purple",
    label="Value of a00",
)
plt.title("Convergence of a00 in Transition Matrix",
fontsize=16, fontweight="bold")
plt.xlabel("Iterations (Number of Steps)",
fontsize=14, labelpad=10)
plt.ylabel("Value of a00", fontsize=14, labelpad=10)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.axhline(y=a00_values[-1], color="green",
linestyle="--", label="Steady-State Value")
plt.grid(color="gray", linestyle="--", linewidth=0.5
alpha=0.7)
plt.legend(fontsize=11.5)
plt.tight_layout()
plt.show()
```