Lab Problems

Grayson Meckfessel

Problem 1

Part B

```
import numpy as np
import random

# Define the transition matrix
P = np.array([
    [0.2, 0.7, 0.1],
    [0.2, 0.5, 0.3],
    [0.2, 0.4, 0.4]
])

# Initial state (index = 0 implies X(0)=1)
current_state = 0

# Number of steps
num_steps = 10

# Simulate the Markov chain
states = [current_state + 1]
for _ in range(num_steps):
    current_state = np.random.choice([0, 1, 2], p = P[current_state])
    states.append(current_state + 1)

print(states)
```

```
[1, 2, 3, 2, 2, 3, 3, 3, 2, 2, 3]
```

The vector shows how the Markov chain changes from one state to another over 10 steps. It's a good example of how the process can move randomly, based on set chances.

Problem 2

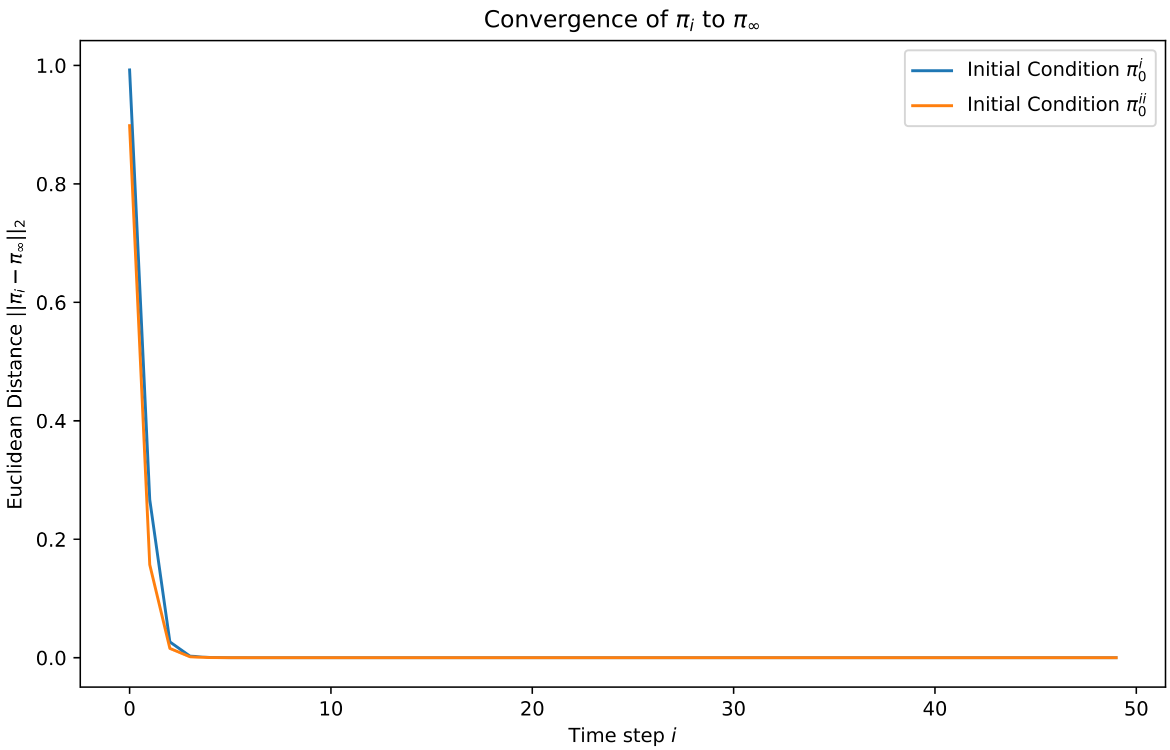
Part A

```
import numpy as np

P = np.array([[0.2, 0.7, 0.1],
              [0.2, 0.5, 0.3],
              [0.2, 0.4, 0.4]])

# P transpose
P_T = P.T

# Identity matrix
I = np.identity(3)

# Constructing the matrix
A = P_T - I

# Adding row with constraint
A_with_constraint = np.vstack([A, [1, 1, 1]])

# Creating b
b = np.array([0, 0, 0, 1])

# Solving
pi = np.linalg.lstsq(A_with_constraint, b, rcond=None)[0]
print(pi)
```

```
[0.2        0.51111111 0.28888889]
```

Part B

```python
import matplotlib.pyplot as plt
import numpy as np

# Define two initial distributions
pi_0_i = np.array([1, 0, 0])
pi_0_ii = np.array([0, 0, 1])

# Number of steps
num_steps = 50

# Store distances
distances_i = []
distances_ii = []

# Calculate the distance for each time step
for i in range(num_steps):

    pi_i_a = np.dot(pi_0_i, np.linalg.matrix_power(P, i))
    pi_i_b = np.dot(pi_0_ii, np.linalg.matrix_power(P, i))

    distance_a = np.linalg.norm(pi_i_a - pi)
    distance_b = np.linalg.norm(pi_i_b - pi)

    distances_i.append(distance_a)
    distances_ii.append(distance_b)

# Plotting
plt.figure(figsize = (10, 6))
plt.plot(range(num_steps), distances_i, label='Initial Condition $\pi_0^i$')
plt.plot(range(num_steps), distances_ii, label='Initial Condition $\pi_0^{ii}$')
plt.xlabel('Time step $i$')
plt.ylabel('Euclidean Distance $||\pi_i - \pi_{\infty}||_2$')
plt.title('Convergence of $\pi_i$ to $\pi_{\infty}$')
plt.legend()
plt.show()
```



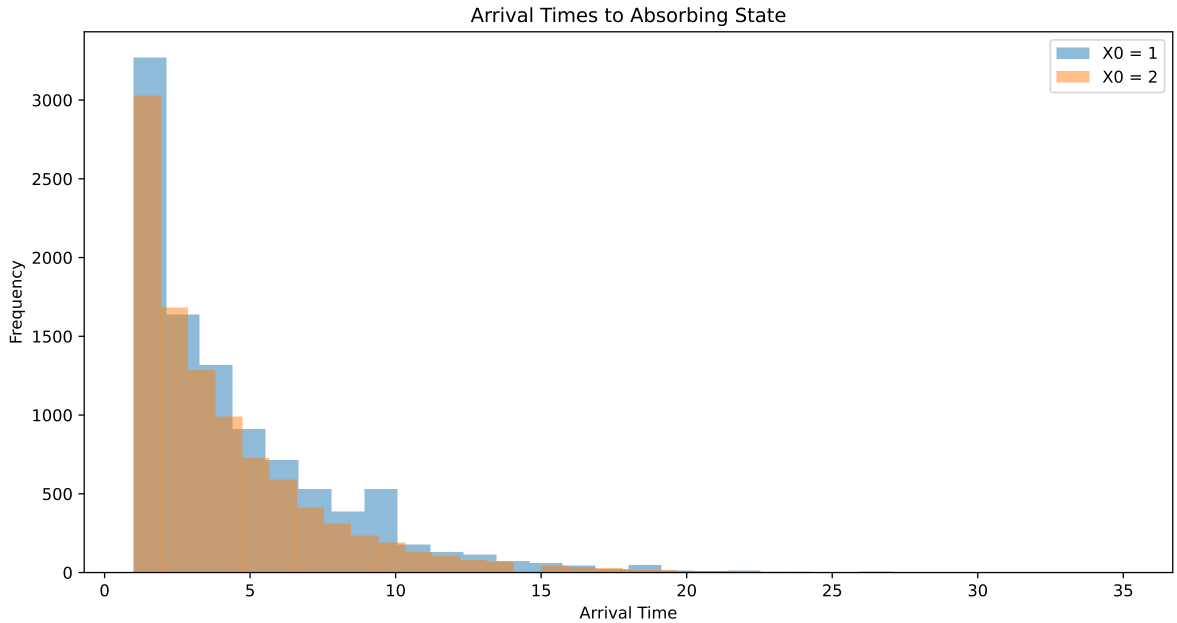
Since the graph converges to zero, this implies that π_i converges to π_∞ in the long run.

Problem 3

Part A

```
import numpy as np
import matplotlib.pyplot as plt

# Modified transition matrix
P_absorbing = np.array([
    [0.2, 0.7, 0.1],
    [0.2, 0.5, 0.3],
    [0, 0, 1]
])

def simulate_until_absorbed(start_state):
```

```
    """Simulates Markov Chain until reaches absorbing state"""

    current_state = start_state
    steps = 0
    while current_state != 2:  # index 2 is node 3
        current_state = np.random.choice([0, 1, 2], p = P_absorbing[current_state])
        steps += 1
    return steps

# Number of simulations
num_simulations = 10000

# Simulating for initial states
arrival_times_1 = [simulate_until_absorbed(0) for _ in range(num_simulations)]
arrival_times_2 = [simulate_until_absorbed(1) for _ in range(num_simulations)]

# Calculating means
mean_time_1 = np.mean(arrival_times_1)
mean_time_2 = np.mean(arrival_times_2)

# Plotting histograms
plt.figure(figsize = (12, 6))
plt.hist(arrival_times_1, bins = 30, alpha = 0.5, label = 'X0 = 1')
plt.hist(arrival_times_2, bins = 30, alpha = 0.5, label = 'X0 = 2')
plt.xlabel('Arrival Time')
plt.ylabel('Frequency')
plt.title('Arrival Times to Absorbing State')
plt.legend()
plt.show()

print(f'Mean Arrival Time for X = 1... {mean_time_1} seconds')
print(f'Mean Arrival Time for X = 2... {mean_time_2} seconds')
print(f'Overall Mean Arrival Time for both states... {0.5*(mean_time_1+mean_time_2)}')
```




```
Mean Arrival Time for X = 1... 4.6237 seconds
Mean Arrival Time for X = 2... 3.8467 seconds
Overall Mean Arrival Time for both states... 4.2352
```