Question 1

3. A car rental company has rental offices at both Kennedy and LaGuardia airports. Assume that a car rented at one airport must be returned to one of the two airports. If the car was rented at LaGuardia the probability it will be returned there is 0.8; for Kennedy the probability is 0.7. Suppose that we start with 1/2 of the cars at each airport and that each week all of the cars are rented once. (a) What is the fraction of cars at LaGuardia ariport at the end of the first week? (b) at the end of the second? (c) in the long run?

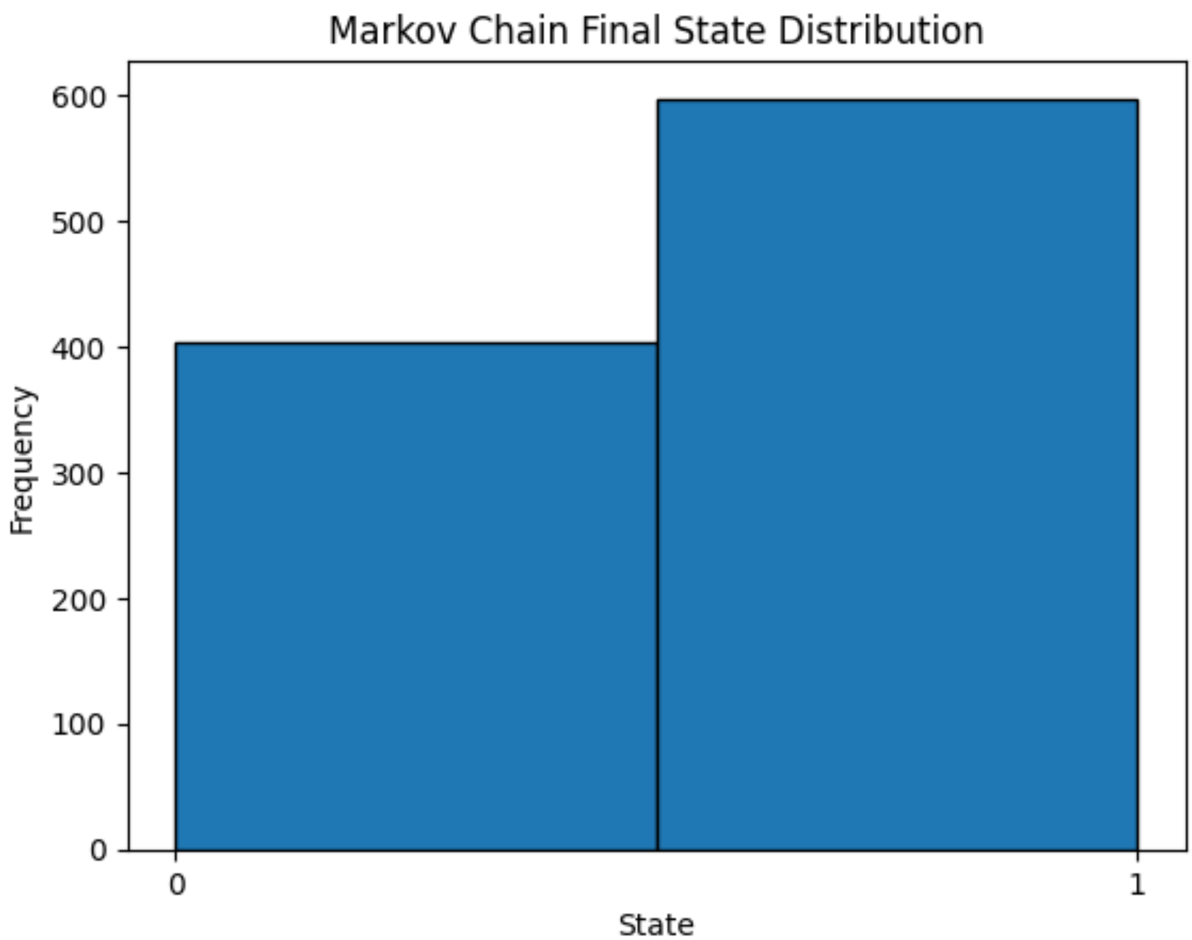
Exercise 11.1: Stationary Measures

Consider the Markov Chain about rental cars in Problem 3 of Section 6.6 of Durrett (2021). 1. Simulate trajectories of this chain and plot 25 steps of the chain.

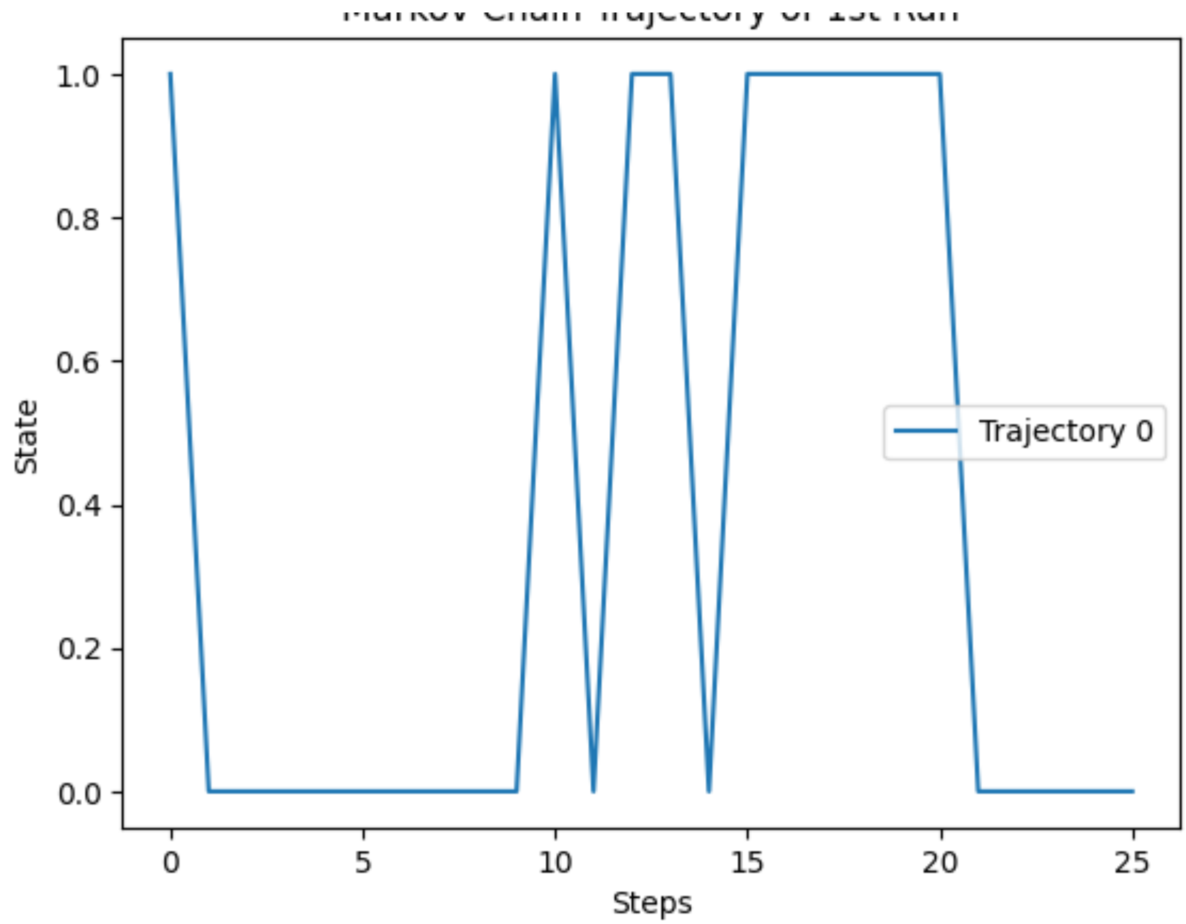
In [1]:

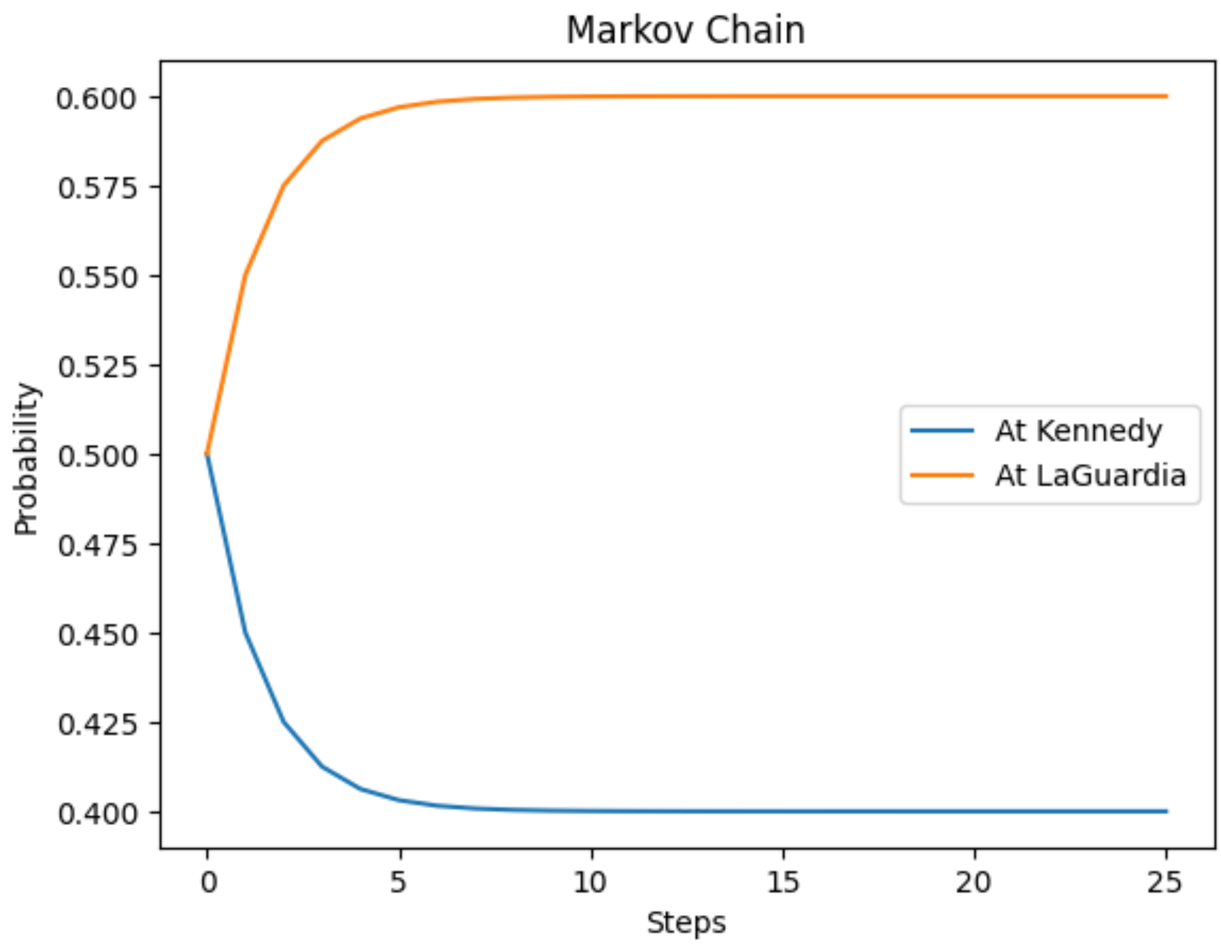
```
import numpy as np
import matplotlib.pyplot as plt

T_matrix = np.array([[0.7, 0.3],
                     [0.2, 0.8]])

initial_states_prob = np.array([0.5, 0.5])

# for plotting

steps = 25

num_trajectories = 1000

final_states = np.zeros((num_trajectories))

final_trajectories = np.zeros((num_trajectories, steps+1))


for i in range(num_trajectories):
    current_state = np.random.choice([0, 1], p=initial_states_prob)
    final_trajectories[i][0] = current_state

    for j in range(steps):
        current_state = np.random.choice([0, 1], p=T_matrix[current_st
        final_trajectories[i][j+1] = current_state

    final_states[i] = current_state

plt.hist(final_states, bins=2, edgecolor='black')
plt.xlabel('State')
plt.ylabel('Frequency')
plt.title('Markov Chain Final State Distribution')
plt.xticks([0, 1])
plt.show()

plt.plot(final_trajectories[0].T)
```

```python
plt.plot(final_trajectories[0].T)
plt.xlabel('Steps')
plt.ylabel('State')
plt.title('Markov Chain Trajectory of 1st Run')
plt.legend(['Trajectory ' + str(i) for i in range(num_trajectories)])
plt.show()


current_state = initial_states_prob
array_of_states = np.zeros((steps+1, 2))
array_of_states[0] = current_state

for i in range(steps):
    current_state = np.dot(current_state, T_matrix)
    array_of_states[i+1] = current_state

plt.plot(array_of_states)
plt.xlabel('Steps')
plt.ylabel('Probability')
plt.title('Markov Chain')
plt.legend(['At Kennedy', 'At LaGuardia'])
plt.show()

# doing some
```



Markov Chain Trajectory of 1st Run


State
1.0
0.8
0.6
0.4
0.2
0.0
Trajectory 0
0
5
10
15
20
25
Steps


Markov Chain
Probability
0.600
0.575
0.550
0.525
0.500
0.475
0.450
0.425
0.400
At Kennedy
At LaGuardia
0
5
10
15
20
25
Steps

> 2. Estimate the fraction of time a car spends at Kennedy airport empirically by the fraction of time a long trajectory describing the location of a car spends at Kennedy airport .

In [2]:
```
print('After 25 steps, the probability of being at Kennedy is', array_
print('After 25 steps, the probability of being at LaGuardia is', arra
print()
print('Therefore, the probability of being at Kennedy is approximately
```

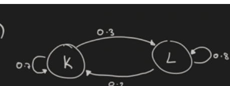
```
After 25 steps, the probability of being at Kennedy is 0.400000002980
2321
After 25 steps, the probability of being at LaGuardia is 0.5999999970
197676

Therefore, the probability of being at Kennedy is approximately 0.400
0000029802321
and the probability of being at LaGuardia is approximately 0.59999999
70197676
```

> 3. Answer the questions as writing in Durrett (2021) using paper and pencil.

3. A car rental company has rental offices at both Kennedy and LaGuardia airports. Assume that a car rented at one airport must be returned to one of the two airports. If the car was rented at LaGuardia the probability it will be returned there is 0.8; for Kennedy the probability is 0.7. Suppose that we start with 1/2 of the cars at each airport and that each

week all of the cars are rented once. (a) What is the fraction of cars at LaGuardia ariport at the end of the first week? (b) at the end of the second? (c) in the long run?

1)



$T(x,y) = P(X_{n+1} = y \mid X_n = x)$.

$\therefore \quad T = \begin{array}{c c} & \begin{array}{cc} K & L \end{array} \\ \begin{array}{c} K \\ L \end{array} & \begin{bmatrix} 0.7 & 0.3 \\ 0.2 & 0.8 \end{bmatrix} \end{array}$ initial state $= [0.5 \quad 0.5]$

a) end of first week $\Rightarrow$ 1 rental cycle

$\Rightarrow [0.5 \quad 0.5] \begin{bmatrix} 0.7 & 0.3 \\ 0.2 & 0.8 \end{bmatrix}^1$

$\Rightarrow [0.45 \quad 0.55]$ $\therefore$ 55% at La Guardia

b) end of second

$\Rightarrow [0.45 \quad 0.55] \begin{bmatrix} 0.7 & 0.3 \\ 0.2 & 0.8 \end{bmatrix} = [0.425 \quad 0.575]$

$\therefore$ 57.5% at La Guardia

c) the long run

$[0.5 \quad 0.5]\, T^n = \alpha \Rightarrow \alpha T = \alpha$

$\Rightarrow \alpha T - \alpha = 0$

$\Rightarrow \alpha(T - I) = 0$ $\qquad T - I = \begin{bmatrix} 0.7 & 0.3 \\ 0.2 & 0.8 \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

$= \begin{bmatrix} -0.3 & 0.3 \\ 0.2 & -0.2 \end{bmatrix}$

$\Rightarrow [a \quad b] \begin{bmatrix} -0.3 & 0.3 \\ 0.2 & -0.2 \end{bmatrix} = 0$

$\Rightarrow \begin{array}{l} 2b - 3a = 0 \\ 3a - 2b = 0 \end{array} \Rightarrow b = \frac{3}{2}a \Rightarrow [a \quad \frac{3}{2}a]$

normalize $\Rightarrow \frac{3}{2}a + a = 1 \Rightarrow \frac{5}{2}a = 1 \Rightarrow a = \frac{2}{5} \Rightarrow b = \frac{3}{5}$

$\therefore \alpha = [2/5 \quad 3/5] \Rightarrow \frac{3}{5}$ in La Guardia on long run

> 4. Check your answers using Python and the `numpy` library.

In [3]:
```python
T = np.array([[0.7, 0.3],
              [0.2, 0.8]])

initial_state = np.array([0.5, 0.5])

# one week probability at LaGuardia
one_week = np.dot(initial_state, T)
print('\nAfter one week, the probability of being at LaGuardia is', or

# two weeks probability at LaGuardia
two_weeks = np.dot(initial_state, np.linalg.matrix_power(T, 2))
print('\nAfter two weeks, the probability of being at LaGuardia is', t

# long run probability at LaGuardia

# solving for a, aT = a
eigenvalues, eigenvectors = np.linalg.eig(T.T)

#print("Eigenvalues:", eigenvalues, "\nEigenvectors:", eigenvectors)

relevant_eigenvalue_index = np.where(np.isclose(eigenvalues, 1))[0][0]
left_eigenvector = eigenvectors[:, relevant_eigenvalue_index].real

#print("Left Eigenvector:", left_eigenvector)

# normalizing the left eigenvector
left_eigenvector /= left_eigenvector.sum()

print("\nNormalized Left Eigenvector:", left_eigenvector)
print("The long run probability of being at LaGuardia is", left_eigenv
```

```
After one week, the probability of being at LaGuardia is 0.55

After two weeks, the probability of being at LaGuardia is 0.575000000
0000001

Normalized Left Eigenvector: [0.4 0.6]
The long run probability of being at LaGuardia is 0.6
```

> ⓘ **Exercise 11.2: Hitting Times and Probabilities**
>
> Consider problem 28 from section 6.6 of Durrett (2021).
>
> 1. Solve the problem as written using paper and pencil.

28. At a manufacturing plant, employees are classified as a recruit (R), technician (T) or supervisor (S). Writing Q for an employee who quits we model their progress through the ranks as a Markov chain with transition probability

	R	**T**	**S**	**Q**
R	.2	.6	0	.2
T	0	.55	.15	.3
S	0	0	1	0
Q	0	0	0	1

(a) What fraction of recruits eventually become a supervisor? (b) What is the expected time until a recruit quits or becomes supervisor?

2) a)

$$P = \begin{array}{c|cc|cc} & R & T & S & Q \\ R & .2 & .6 & 0 & .2 \\ T & 0 & .55 & .15 & .3 \\ \hline S & 0 & 0 & 1 & 0 \\ Q & 0 & 0 & 0 & 1 \end{array} \quad \longleftarrow \text{Absorbing states.}$$

let $h(x) = \mathbb{P}$ state $x \longrightarrow$ supervisor

$$\Rightarrow \quad h(R) = 0.2h(R) + 0.6h(T)$$
$$h(T) = 0.55h(T) + 0.15h(S)$$
$$h(S) = 1.$$

$$\therefore \quad h(T) = \frac{0.15}{0.45} = \frac{3}{9}$$

$$\Rightarrow \quad h(R) = \frac{\cancel{10}}{8} \times \frac{\cancel{6}^{\,2}}{\cancel{10}} \times \frac{3}{\cancel{9}_{\,3}} = \frac{1}{4}.$$

$$\therefore \quad \mathbb{P}(\text{Recruit} \to \text{Supervisor}) = \frac{1}{4}$$

b) expected time until a recruit quits or becomes a supervisor.

let $g(x)$ be the expected number of years to become a supervisor/quit

$$\therefore \quad g(R) = 1 + 0.2g(R) + 0.6g(T) + 0.2g(Q) \qquad g(S),\ g(Q) = 0$$
$$= 1 + 0.2g(R) + 0.6g(T).$$

$$g(T) = 1 + 0.55g(T) + 0.15g(S) + 0.3g(Q)$$
$$= 1 + 0.55g(T).$$

$$\Rightarrow \quad g(T) = \frac{1}{0.45} = \frac{20}{9}$$

$$\Rightarrow \quad 0.8g(R) = 1 + \frac{\cancel{6}^{\,2}}{\cancel{10}} \cdot \frac{\cancel{20}^{\,2}}{\cancel{9}_{\,3}} = \frac{3}{3} + \frac{4}{3}$$

$$\Rightarrow \quad g(R) = \frac{10}{8} \cdot \frac{7}{3} = \frac{35}{12}$$

is there a general matrix approach?

$$0.8g(R) - 0.6g(T) = 1$$
$$0g(R) + 0.45g(T) = 1$$
$$\Rightarrow \begin{pmatrix} 0.8 & 0.6 \\ 0 & 0.45 \end{pmatrix} \begin{pmatrix} g(R) \\ g(T) \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

looks like a subset of markov matrix, $(I - M)$?

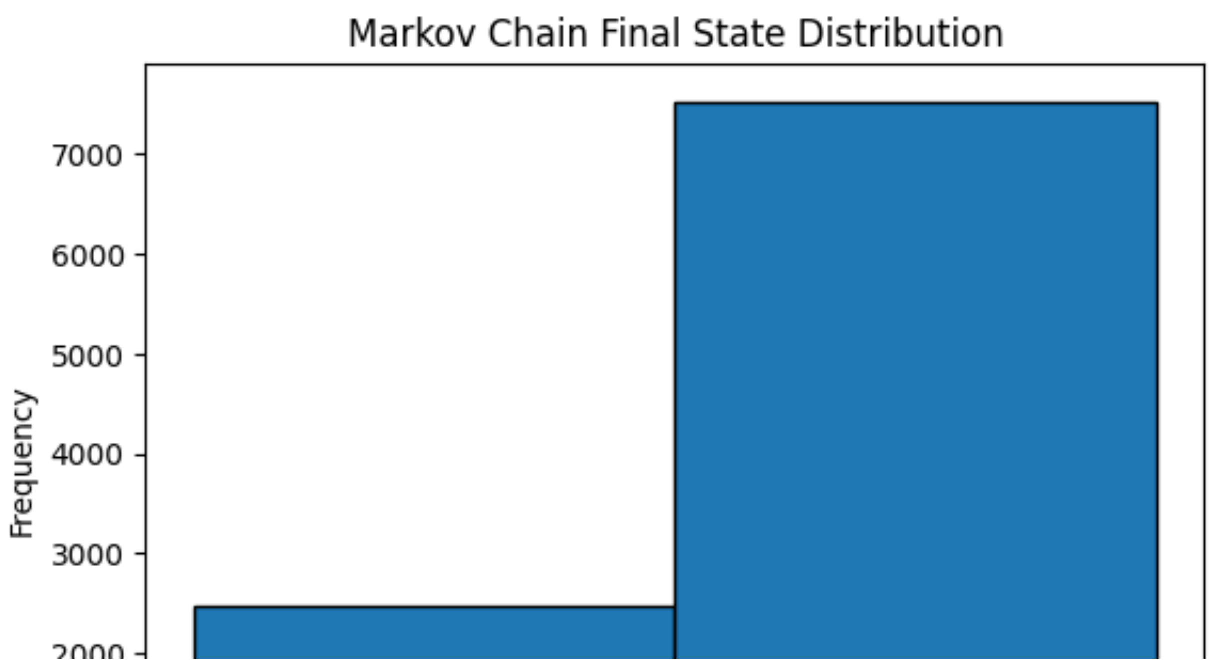
is this generally true?

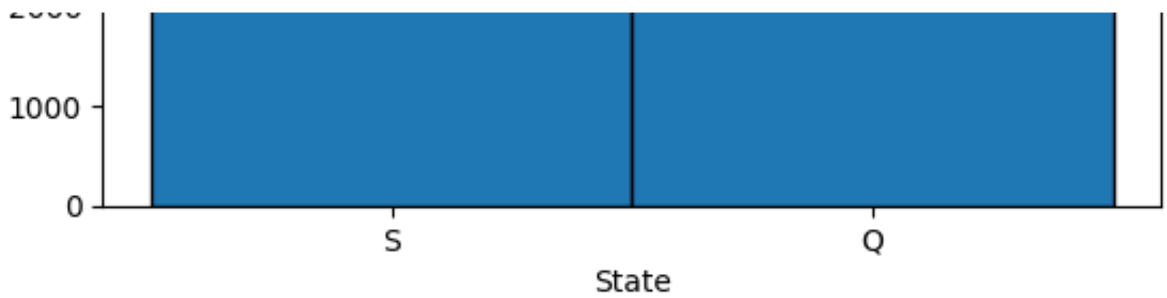
what would matrix subset generally be?

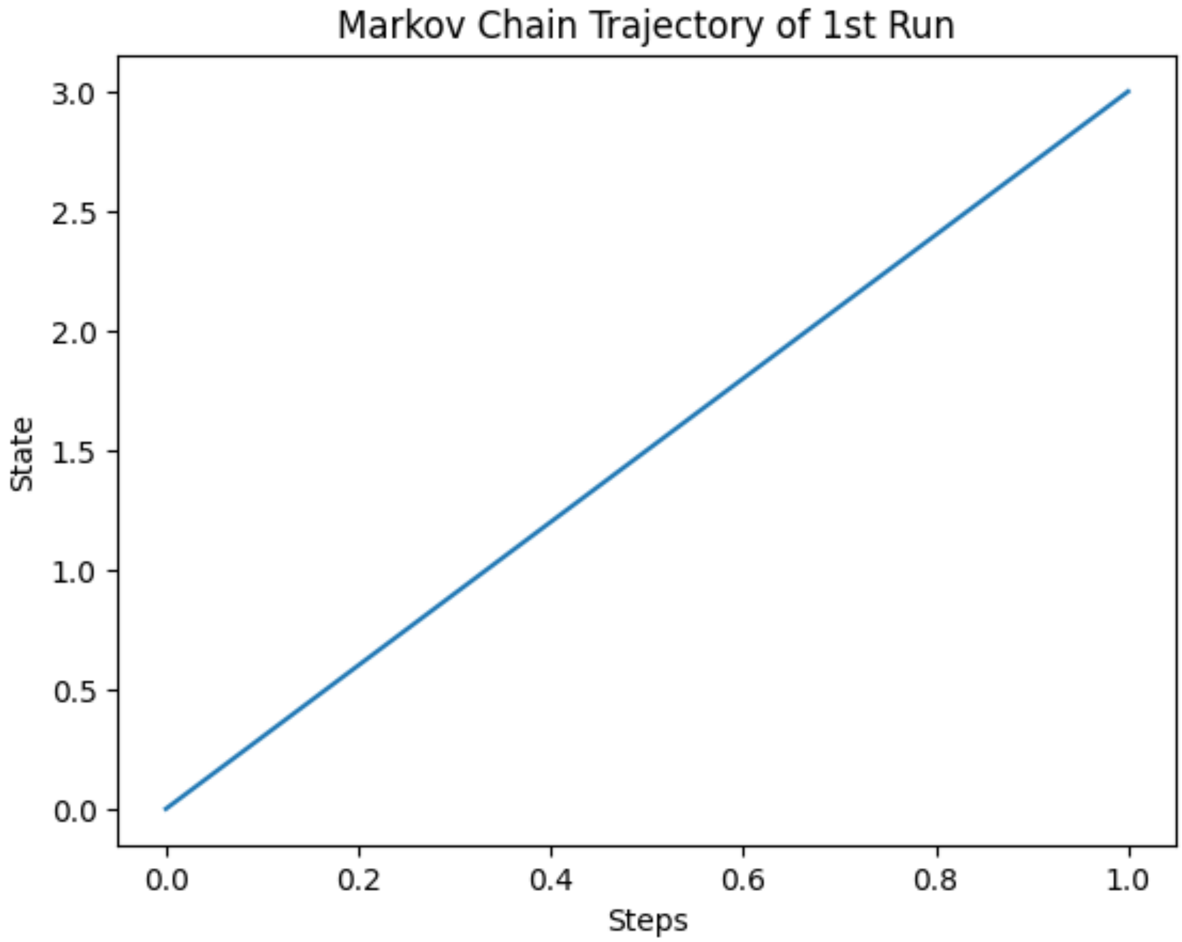
> 2. Simulate trajectories of the Markov Chain.

In [26]:
```
T = np.array([
    #[R    T      S      Q]
    [0.2, 0.60, 0.00, 0.2], # R
    [0.0, 0.55, 0.15, 0.3], # T
    [0.0, 0.00, 1.00, 0.0], # S <- absorbing state
    [0.0, 0.00, 0.00, 1.0]  # Q <- absorbing state
])

def single_trajectory(initial_state, T):
    trajectory = []

    # current_state, R = 0, T = 1, S = 2, Q = 3
    current_state = initial_state
    trajectory.append(current_state)

    while current_state not in [2, 3]: # while not in an absorbing sta
```

```python
        # choose next state based on random choice from transition mat
        current_state = np.random.choice([0, 1, 2, 3], p=T[current_sta
        trajectory.append(current_state)

    return trajectory, trajectory[-1]

initial_state = 0 # = R
num_trajectories = 10000
final_states = np.zeros((num_trajectories))
final_trajectory_lengths = np.zeros((num_trajectories))
trajectories = []


for i in range(num_trajectories):
    trajectory, final_state = single_trajectory(initial_state, T)
    final_states[i] = final_state
    final_trajectory_lengths[i] = len(trajectory)
    trajectories.append(trajectory)


plt.hist(final_states, bins=2, edgecolor='black')
plt.xlabel('State')
plt.ylabel('Frequency')
plt.title('Markov Chain Final State Distribution')
# Add labels to the bins
plt.xticks([2.25, 2.75], ['S', 'Q'])
plt.show()

# showing the first trajectory
plt.plot(trajectories[0])
plt.xlabel('Steps')
plt.ylabel('State')
plt.title('Markov Chain Trajectory of 1st Run')
plt.show()
```







> 3. Using many simulated trajectories estimate the probabilities and average time you calculated in the first part of the question.

In [30]:
```python
print("------ Empirical Estimates of Probability and Average Length of

# probability estimates
print('\nThe probability of ending up in state S is', np.mean(final_st
print('The probability of ending up in state Q is', np.mean(final_stat

# average length of trajectories
print('\nThe average length of the trajectories is', np.mean(final_tra
```

```
------ Empirical Estimates of Probability and Average Length of Traje
ctories ------

The probability of ending up in state S is 0.2474
The probability of ending up in state Q is 0.7526

The average length of the trajectories is 2.9458
```

> **Exercise 12.1: MCMC using Walk on circle.**
>
> Consider the graph with vertices $V = \{0, 1, 2, \ldots, N-1\}$ for $N = 23$ with edges $(i, i+1)$ for all $i = 0, 1, 2, \ldots, 21$ and $(22, 0)$. (Simply nearest neighbors are connected to each other when the points are viewed on a circle). Let T be the Markov Transition matrix for this simple random walk.
>
> 1. Argue that $T(i, j) = T(j, i)$ for all i, j in V.

3)

a) edges $(i, i+1)$ for all $i = 0, 1, \ldots, 21$. & edge $(22, 0)$

$\therefore$ each row of T has elements (in location j) that represent $\mathbb{P}(X_{n+1} = j \mid X_n = \text{row})$.

and as edges $(i, i+1)$ & $(i+1, i)$ are present then matrix is symmetric

$\therefore \underline{T(i, j) = T(j, i)}$

> 2. Use the previous observation to see that the uniform distribution on V is the stationary measure for this Markov chain.

In [54]:

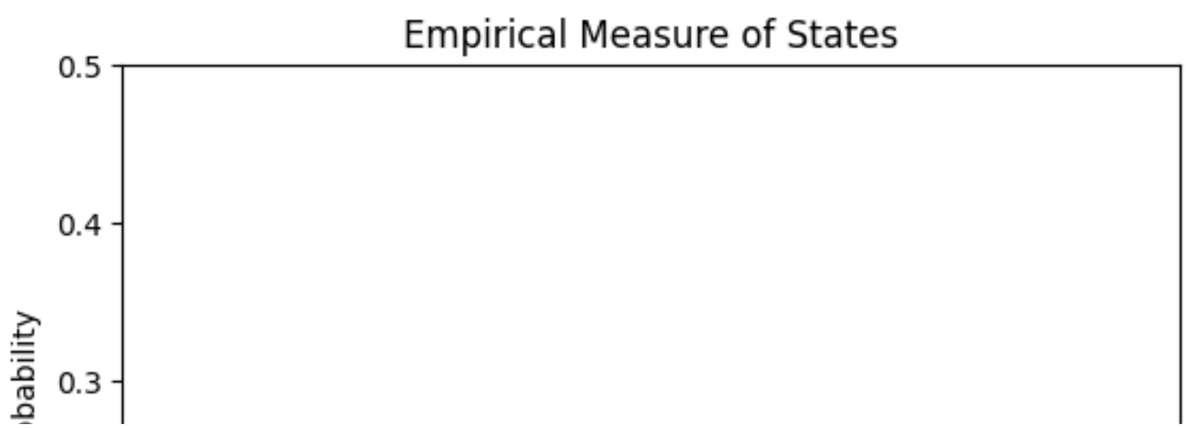
```
T = np.zeros((23, 23))
# print(T[23])
# print(T)

for i in range(22):
    T[i, i+1] = 1
    T[i+1, i] = 1

T[22, 0] = 1
T[0, 22] = 1

T = T/2

# print(T)

uniform_initial_state = np.ones(23) / 23

# checking if aT = a

aT = np.dot(uniform_initial_state, T)

print("\nUniform Initial State:", uniform_initial_state)
print("\naT:", aT)
print("\nTherefore, aT = a, where a is the uniform initial state vecto
print("\nSo the uniform initial state vector is the stationary measure
```

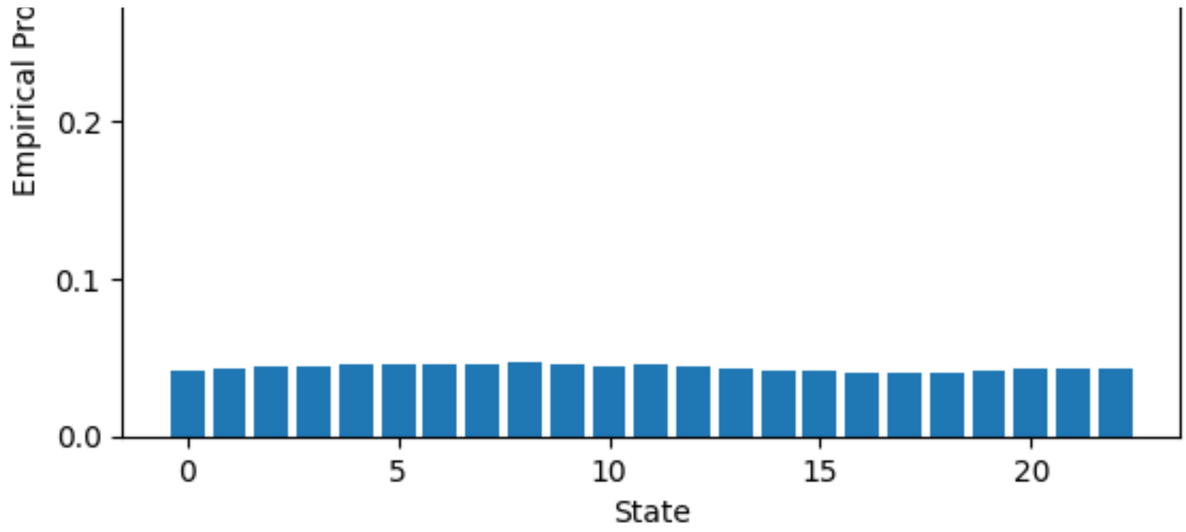
```
Uniform Initial State: [0.04347826 0.04347826 0.04347826 0.04347826 0
.04347826 0.04347826
 0.04347826 0.04347826 0.04347826 0.04347826 0.04347826 0.04347826
 0.04347826 0.04347826 0.04347826 0.04347826 0.04347826 0.04347826
 0.04347826 0.04347826 0.04347826 0.04347826 0.04347826]

aT: [0.04347826 0.04347826 0.04347826 0.04347826 0.04347826 0.0434782
6
 0.04347826 0.04347826 0.04347826 0.04347826 0.04347826 0.04347826
 0.04347826 0.04347826 0.04347826 0.04347826 0.04347826 0.04347826
 0.04347826 0.04347826 0.04347826 0.04347826 0.04347826]

Therefore, aT = a, where a is the uniform initial state vector.

So the uniform initial state vector is the stationary measure for thi
s Markov chain.
```

> 3. Simulate a long trajectory $\{X_n : n = 1, \ldots, L\}$ of the Simple random walk with Transition Matrix T. Check that the empirical measure m defined by
>
> $$m_L(k) = \frac{1}{L} \sum_{j=1}^{L} \mathbf{1}_k(X_j)$$
>
> is close to the uniform distribution on V. (That is $m(k) \approx \frac{1}{N}$ for all k).

In [67]:

```
# simulating a long trajectory

current_state = np.random.choice(range(23), p=uniform_initial_state)

num_steps = 100000
trajectory = np.zeros((num_steps+1))
trajectory[0] = current_state

empirical_measure = np.zeros(23)

for i in range(num_steps):
    current_state = np.random.choice(range(23), p=T[current_state])
    trajectory[i+1] = current_state
    empirical_measure[current_state] += 1

empirical_measure /= num_steps

# Plotting the bar chart
plt.bar(range(len(empirical_measure)), empirical_measure)
plt.xlabel('State')
plt.ylim(0, 0.5)
plt.ylabel('Empirical Probability')
plt.title('Empirical Measure of States')
plt.show()

print("\nTherefore, the empirical measure of the states is approximate

# plt.plot(trajectory)
# plt.xlabel('Steps')
# plt.ylabel('State')
# plt.title('Markov Chain Trajectory')
# plt.show()
```






```
Therefore, the empirical measure of the states is approximately unifo
rm.
```

> 4. To measure the distance from the uniform distribution plot the Total Variation distance between m_L and the uniform distribution given by
>
> $$\|m_L - \text{Uniform}(V)\|_{TV} = \frac{1}{2}\sum_{k\in V}|m(k) - \frac{1}{N}|$$
>
> as a function of L

In [68]:

```python
uniform_initial_state = np.ones(23) / 23

total_variation = 0.5 * np.sum(np.abs(empirical_measure - uniform_init

print("\nThe total variation distance between the empirical measure ar
```

```
The total variation distance between the empirical measure and the un
iform initial state is 0.0188591304347826
```

> 5. Let π be a distribution on V with $\pi(k) \propto e^{-H(k)}$ where $H(k) = \sin(2\pi\frac{k}{N})$. Use the Metropolis-Hastings algorithm to sample π using the simple random walk discussed as the proposal. Simulate and plot a reasonably long trajectory of this Metropolis-Hastings algorithm.

> 6. Use a long trajectory of the Markov chain constructed using the Metropolis-Hastings algorithm above to calculate long time probability of being in each vertex. Show that as expected this probably corresponds to $\pi(k)$.

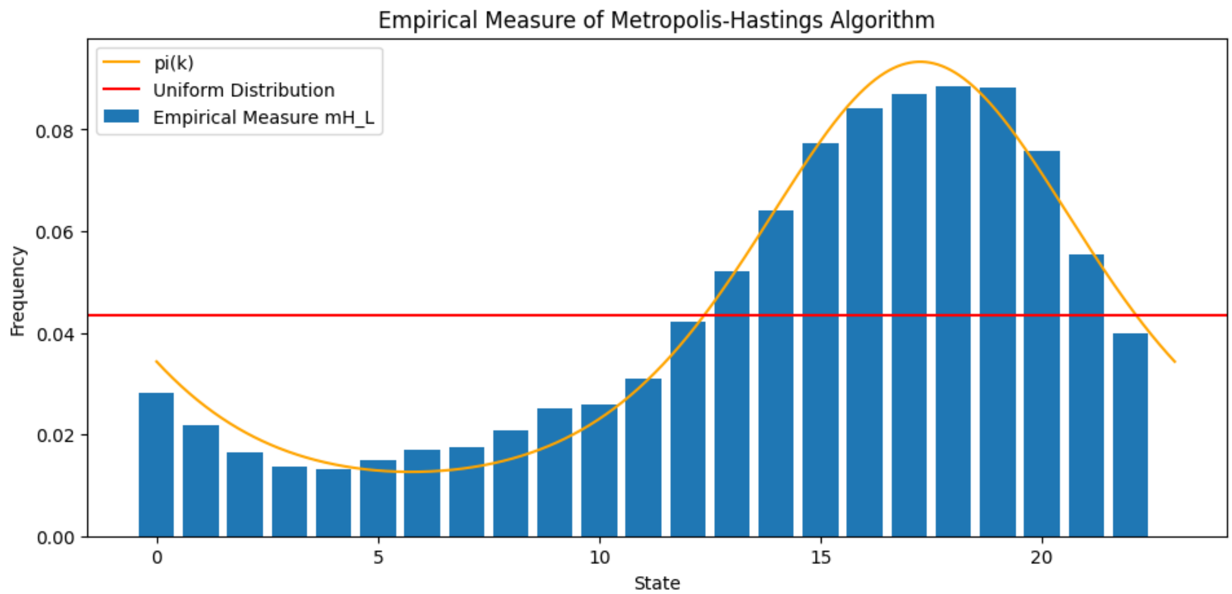
In [17]:

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

N = 23  # Number of vertices

# define transition matrix
T = np.zeros((23, 23))
# print(T[23])
# print(T)
```

```
for i in range(22):
    T[i, i+1] = 1
    T[i+1, i] = 1

T[22, 0] = 1
T[0, 22] = 1

T = T/2

L = 10000  # Length of the Markov chain

# Metropolis-Hastings
# defining H(k)
H = lambda k: np.sin(2 * np.pi * k / N)
# defining pi(k)
pi = lambda k: np.exp(-H(k))
Z = np.sum([pi(k) for k in range(N)])  # summing all values of pi(k) f
pi_normalized = lambda k: pi(k) / Z # normalising pi(k)

# Simulate the Metropolis-Hastings algorithm
Y = np.zeros(L, dtype=int)
Y[0] = np.random.choice(range(N))  # Start from a random state
for i in range(1, L):
    current_state = Y[i-1]
    # Propose a new state using the simple random walk transition matr
    proposed_state = np.random.choice(range(N), p=T[current_state])
    # Calculate the acceptance probability
    alpha = min(1, pi_normalized(proposed_state) / pi_normalized(curre
    # Accept or reject the new state
    Y[i] = proposed_state if np.random.rand() < alpha else current_sta

# Calculate the empirical measure for the Metropolis-Hastings algorith
mH_L = np.array([np.mean(Y == k) for k in range(N)])

# Plotting
plt.figure(figsize=(25, 5))

print(mH_L)

# Plot for Metropolis-Hastings algorithm
plt.subplot(1, 2, 2)

# plot pi(k) as a continuous function
k = np.linspace(0, N, 1000)
plt.plot(k, pi_normalized(k), color = "orange", label='pi(k)')

plt.bar(range(N), mH_L, label='Empirical Measure mH_L')
plt.axhline(1/N, color='red', linestyle='-', label='Uniform Distributi
plt.title('Empirical Measure of Metropolis-Hastings Algorithm')
plt.xlabel('State')
plt.ylabel('Frequency')
plt.legend()
```

```
plt.show()
```

```
[0.0283 0.0217 0.0165 0.0137 0.0131 0.0149 0.0171 0.0175 0.0207 0.025
2
 0.026  0.031  0.0423 0.052  0.064  0.0772 0.0841 0.0869 0.0886 0.088
2
 0.0758 0.0554 0.0398]
```



> 7. Defining the m_L, empirical distribution after L steps, as before. Plot the Total Variation distribution as a function of L where
>
> $$\|m_L - \pi\|_{TV} = \frac{1}{2}\sum_{k\in V}|m(k) - \pi(k)|$$
>
> Make sure you properly normalize π by setting
>
> $$\pi(k) = \frac{1}{Z}e^{-H(k)}$$
>
> where the normalizing constant Z is chosen so $\sum_k \pi(k) = 1$.

In [20]:
```
# calculating the total variation distance

N = 23

total_variation = 0.5 * np.sum( np.abs(mH_L - pi_normalized(np.arange(

print("The total variation distance between the empirical measure and
```

```
The total variation distance between the empirical measure and the ta
rget distribution is 0.04001645893750866
```