Final Project Report

Hesam Hosseini , Parsa Palizian

July 5, 2023

Contents

1 markov chain

Theoretical Question 1

.1 Gambler's Ruin

Gambler's Ruin is a classic example in the field of probability theory and Markov chains. It involves a hypothetical scenario of a gambler betting on a fair game repeatedly until either they lose all their money or reach a predetermined goal.

let's say we have a gambler who begins with an initial capital of n dollars. The gambler places bets in a fair game against an opponent, where the probability of winning each bet is denoted by p, and the probability of losing is $1 - p$.

The gambler continues to play the game until one of two events occurs:

1. The gambler reaches their predetermined goal: Let's say the gambler's goal is to accumulate a total of m dollars. If the gambler reaches this goal, they stop playing.

2. The gambler loses all their money: If at any point the gambler loses all their money and their capital reduces to 0, they are unable to continue playing.

To analyze this scenario using Markov chains, we construct a mathematical model where the different states represent the current capital of the gambler. The Markov chain has absorbing states of 0 and m, corresponding to the gambler losing all their money or reaching their goal, respectively.

The transition probabilities between states are determined by the outcomes of the game. For instance, if the gambler currently has i dollars, the transition probability from state i to state $i + 1$ is p, as there is a chance of winning the next bet and increasing the capital by 1. Similarly, the transition probability from state i to state $i - 1$ is $1 - p$, representing the probability of losing the next bet and decreasing the capital by 1.

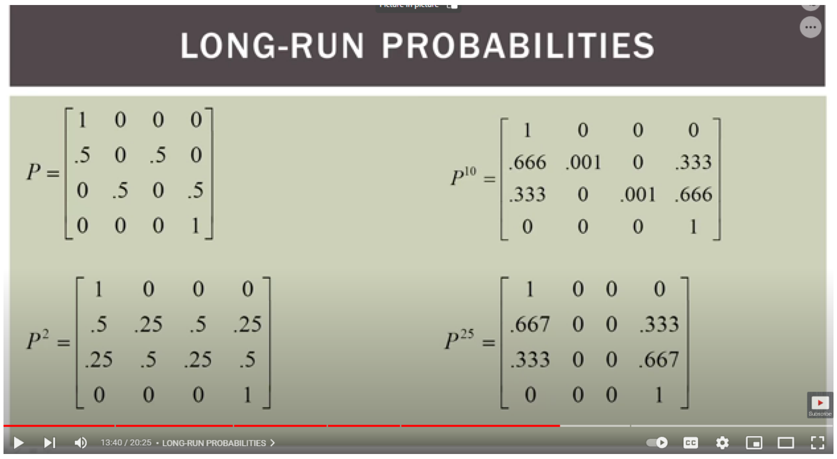
The transition matrix for Gambler's Ruin can be represented as:

$$P = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ q & 0 & p & \dots & 0 \\ 0 & q & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix}$$

In this matrix, the rows and columns represent the different capital states, and each entry represents the probability of transitioning from one state to another based on the game's outcomes.

Analyzing the properties of this Markov chain, such as the probability of reaching the absorbing states or expected time to absorption, allows us to gain insights into the gambler's chances of reaching their goal or losing all their money in the long run.

Gambler's Ruin provides a fundamental example to study the behavior of Markov chains, and it has implications beyond gambling scenarios, including studies of random walks and absorption probabilities in various fields.



LONG-RUN PROBABILITIES

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ .5 & 0 & .5 & 0 \\ 0 & .5 & 0 & .5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad P^{10} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ .666 & .001 & 0 & .333 \\ .333 & 0 & .001 & .666 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$P^2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ .5 & .25 & .5 & .25 \\ .25 & .5 & .25 & .5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad P^{25} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ .667 & 0 & 0 & .333 \\ .333 & 0 & 0 & .667 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

13:40 / 20:25 • LONG-RUN PROBABILITIES >

Figure 1. log run in gambler's ruin

.2 PageRank

PageRank is an algorithm used by search engines to determine the importance or relevance of web pages. It was developed by Larry Page and Sergey Brin, the founders of Google. The algorithm assigns a numerical ranking to each web page based on the structure of the web and the linking patterns between pages.

PageRank can be modeled as a Markov chain, where each web page represents a state in the chain, and the transitions between pages occur based on the links present on those pages.

Let's consider a simplified model with n web pages. We can represent the web pages as states $S_1, S_2, \ldots, S_n$. The links between web pages determine the transition probabilities in the Markov chain.

The transition matrix for this Markov chain can be represented as:

$$P = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1n} \\ p_{21} & p_{22} & \cdots & p_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ p_{n1} & p_{n2} & \cdots & p_{nn} \end{bmatrix}$$

Each entry p_{ij} represents the probability of transitioning from web page S_i to web page S_j. These probabilities are determined by the links on each page. A web page S_i with k outgoing links will have the transition probability $p_{ij} = \frac{1}{k}$ for each page S_j that it links to.

The initial probability distribution vector $\mathbf{v}_0$ represents the importance or rank of each web page at the start of the algorithm. It can be represented as:

$$\mathbf{v}_0 = \begin{bmatrix} v_{01} \\ v_{02} \\ \vdots \\ v_{0n} \end{bmatrix}$$

where v_{0i} represents the initial rank of web page S_i. Typically, the initial rank values are set uniformly or randomly.

By iteratively multiplying the transition matrix P with the probability distribution vector $\mathbf{v}_0$, we can calculate the PageRank vector $\mathbf{v}$, which represents the final importance or ranking of each web page.

$$\mathbf{v} = P \cdot \mathbf{v}_0$$

The PageRank vector $\mathbf{v}$ will converge to a steady-state vector where each entry represents the importance or rank of the corresponding web page. The higher the PageRank value, the more important or relevant the web page is considered to be.

The PageRank algorithm takes into account both the number of links pointing to a web page and the quality of those linking pages. It considers the idea that a page is more important if it is linked to by other important pages.

Theoretical Question 2

first we prove $\lambda_1 = \lambda p \quad \lambda$ is the initial distribution

$$\begin{aligned} &\lambda_1(i) = \sum_j \lambda(j)\mathbb{P}\left(X_1 = i \mid X_0 = j\right) \\ &\forall i : \lambda_1(i) = \sum_j \lambda(j)P(j,i) \\ &\Rightarrow \lambda_1 = \lambda P \end{aligned}$$

suppose the the assumption for $n = k$ is true $\lambda_k = \lambda p^k$

for $n = k+1$ we have:

$$\begin{aligned}
\lambda_{k+1}(i) &= \sum_j \lambda_k(j)\mathbb{P}\left(X_{k+1} = i | X_k = j\right) \\
&= \sum_j \lambda_k(j)p(j,i) \\
&= \sum_j \left(\lambda p^k\right)_j p(j,i) \\
&= \sum_j \sum_l \lambda(l)p^k(l,j)p(j,i) \\
&= \sum_l \lambda(l) \sum_j p^k(l,j)p(j,i) \\
&= \sum_l \lambda(l)p^{k+1}(l,i) \\
&= \sum_l \lambda(l)p^{k+1}(l,i) \\
&\Rightarrow \lambda_{k+1} = \lambda p^{k+1}
\end{aligned}$$

another method:
we have: $\lambda_k = \lambda \cdot (P^k)$

Now, to find λ_{k+1}, we can multiply λ_k by P: $\lambda_{k+1} = \lambda_k \cdot P$
$= (\lambda \cdot (P^k)) \cdot P$
$= \lambda \cdot ((P^k) \cdot P)$
$= \lambda \cdot (P^{k+1})$

Therefore, the equation $\lambda_n = \lambda \cdot (P^n)$ holds for $n = k+1$.

By mathematical induction, we have proved that $\lambda_n = \lambda \cdot (P^n)$ for all $n \geq 0$.

This means that the distribution of the Markov chain at time n can be obtained by multiplying the initial distribution λ with the transition matrix raised to the power of n.

Theoretical Question 3

first let's simplifies the question what is the probability of making a transition from state i to state j over two step We are seeking $\mathbb{P}\left(X_2 = j \mid X_0 = i\right)$.

Use the Partition Theorem:

$$\begin{aligned}
\mathbb{P}\left(X_2=j \mid X_0=i\right) &= \mathbb{P}_i\left(X_2=j\right) \\
&= \sum_{k=1}^{N} \mathbb{P}_i\left(X_2=j \mid X_1=k\right) \mathbb{P}_i\left(X_1=k\right) \qquad \text{(Partition Thm)} \\
&= \sum_{k=1}^{N} \mathbb{P}\left(X_2=j \mid X_1=k, X_0=i\right) \mathbb{P}\left(X_1=k \mid X_0=i\right) \\
&= \sum_{k=1}^{N} \mathbb{P}\left(X_2=j \mid X_1=k\right) \mathbb{P}\left(X_1=k \mid X_0=i\right) \\
&= \sum_{k=1}^{N} p_{kj} p_{ik} \qquad \text{(by definitions)} \\
&= \sum_{k=1}^{N} p_{ik} p_{kj} \qquad \text{(rearranging)} \\
&= \left(P^2\right)_{ij} .
\end{aligned}$$

The two-step transition probabilities are therefore given by the matrix P^2 :

$$\mathbb{P}\left(X_2=j \mid X_0=i\right)=\mathbb{P}\left(X_{n+2}=j \mid X_n=i\right)=\left(P^2\right)_{ij} \qquad \text{for any } n$$

3-step transitions: We can find $\mathbb{P}\left(X_3=j \mid X_0=i\right)$ similarly, but conditioning on the state at time 2 :

$$\begin{aligned}
\mathbb{P}\left(X_3=j \mid X_0=i\right) &= \sum_{k=1}^{N} \mathbb{P}\left(X_3=j \mid X_2=k\right) \mathbb{P}\left(X_2=k \mid X_0=i\right) \\
&= \sum_{k=1}^{N} p_{kj}\left(P^2\right)_{ik} \\
&= \left(P^3\right)_{ij} .
\end{aligned}$$

General case: t-step transitions The above working extends to show that the t-step transition probabilities are given by the matrix
$P_{ij}^{(t)}=(P^{(t-1)})_{i1}P_{1j}+(P^{(t-1)})_{i2}P_{2j}+\ldots+(P^{(t-1)})_{im}P_{mj}$
P^t for any t :

$$\mathbb{P}\left(X_t=j \mid X_0=i\right)=\mathbb{P}\left(X_{n+t}=j \mid X_n=i\right)=\left(P^t\right)_{ij} \qquad \text{for any } n$$

The rows of P and P^n should each sum to 1 :

$$\sum_{j=1}^{N} p_{ij}=\sum_{j=1}^{N} \mathbb{P}\left(X_{t+1}=j \mid X_t=i\right)=\sum_{j=1}^{N} \mathbb{P}_{\{X_t=i\}}\left(X_{t+1}=j\right)=1$$

This simply states that X_{t+1} must take one of the listed values. The columns of P do not in general sum to 1 .
now let's generalize it for p^n
For the base case, consider $n = 1$. In this case, $P^1 = P$. Since P is a transition matrix, each row of P represents the transition probabilities from a specific state to all other states. By definition, the sum of transition probabilities from a state to all other states should be equal to 1. Therefore, the rows of $P^1 = P$ indeed sum to 1.

Now, assume that for some $k \geq 1$, the rows of P^k each sum to 1. We will prove that the rows of P^{k+1} also sum to 1.

Let's consider the i-th row of P^{k+1} denoted as $(P^{k+1})_{i,*}$. The entry in the j-th column of $(P^{k+1})_{i,*}$, denoted as $(P^{k+1})_{i,j}$, represents the probability of transitioning from state i to state j in $k+1$ steps.

Using matrix multiplication, the (i, j) entry of P^{k+1} is calculated as:

$$(P^{k+1})_{i,j} = (P^k)_{i,1} \cdot P_{1,j} + (P^k)_{i,2} \cdot P_{2,j} + \cdots + (P^k)_{i,n} \cdot P_{n,j}.$$

Now, let's sum the entries in the i-th row of P^{k+1}:

$$\sum_{j=1}^{n} (P^{k+1})_{i,j} = \sum_{j=1}^{n} \left[(P^k)_{i,1} \cdot P_{1,j} + (P^k)_{i,2} \cdot P_{2,j} + \cdots + (P^k)_{i,n} \cdot P_{n,j} \right].$$

By rearranging the terms and using the fact that the rows of P^k sum to 1, we have:

$$\sum_{j=1}^{n} (P^{k+1})_{i,j} = (P^k)_{i,1} \cdot \sum_{j=1}^{n} P_{1,j} + (P^k)_{i,2} \cdot \sum_{j=1}^{n} P_{2,j} + \cdots + (P^k)_{i,n} \cdot \sum_{j=1}^{n} P_{n,j}.$$

Since each row of P sums to 1, we have $\sum_{j=1}^{n} P_{1,j} = \sum_{j=1}^{n} P_{2,j} = \cdots = \sum_{j=1}^{n} P_{n,j} = 1$. Therefore, we can simplify the expression as:

$$\sum_{j=1}^{n} (P^{k+1})_{i,j} = (P^k)_{i,1} + (P^k)_{i,2} + \cdots + (P^k)_{i,n} = 1.$$

Thus, we have shown that for any $k \geq 1$, if the rows of P^k each sum to 1, then the rows of P^{k+1} also sum to 1.

there exist at least one eigenvalue for p^n equal to 1
a stochastic matrix is a square matrix where each column represents a probability distribution. In the case of a Markov chain, the transition matrix $\mathbf{P^n}$ is a stochastic matrix.

By definition, a stochastic matrix satisfies the property that each column sums up to 1. This property reflects that the probabilities of transitioning from a state to all other states must sum up to 1.

Since $\mathbf{P}$ and $\mathbf{P^n}$ are a stochastic matrix, the sum of the elements in each column is 1. so if we prove this theorem for $\mathbf{P}$ it is also true for $\mathbf{P^n}$
This property implies that for any vector $\mathbf{v}$ with all entries equal to 1, $\mathbf{Pv}$ will also have all entries equal to 1, since each column of $\mathbf{P}$ sums up to 1.

Now, let's consider the equation $\mathbf{Pv} = \lambda\mathbf{v}$, where $\mathbf{v}$ is a non-zero vector and λ is an eigenvalue of $\mathbf{P}$.

If we choose $\mathbf{v}$ as the vector with all entries equal to 1, then the equation becomes $\mathbf{P1} = \lambda\mathbf{1}$.

Since each column of $\mathbf{P}$ sums up to 1, the left-hand side of the equation is equal to the vector $\mathbf{1}$, which means that $\mathbf{1}$ is an eigenvector of $\mathbf{P}$.

Therefore, we have found an eigenvector $\mathbf{1}$ corresponding to the eigenvalue $\lambda = 1$.

This proves that there exists at least one eigenvalue of 1 for the transition matrix $\mathbf{P}$ in a Markov chain.

It's important to note that in a Markov chain, the eigenvalue $\lambda = 1$ corresponds to the stationary distribution or the long-term equilibrium distribution of the chain.

Theoretical Question 4

We know that (Chapman-Kolmogorov equation.)

$$P^{(m+n)}(i,j) = \sum_k P^{(m)}(i,k)P^{(n)}(k,j).$$

We call such a sequence a markov chain of order k. so we need to prove that any markov chain in order k can convert to a markov chain with order 1. If we only show that we can convert it to a order k-1 then based on induction we are done.
To show that a Markov chain of order k can be represented as a Markov chain of order 1, we can define a new state space that includes all possible combinations of the previous k-1 states and the current state
The transition probabilities between the new states can be calculated using the transition probabilities of the original chain.
Prove that the new chain is a Markov chain of order 1: The new chain satisfies the memoryless property of a Markov chain because the probability of moving to a particular state depends only on the current state and not on any previous states beyond the most recent k-1 states
Therefore, the new chain is a Markov chain of order 1.

Let's define a random variable $Z_n = (X_n, X_{n-1}, \dots, X_{n-K+1})$ to represent the memory of the process at time n, we have:

$$Z_{n+1} = (X_{n+1}, X_n, X_{n-1}, \dots, X_{n-K+1})$$

We know that X_{n+1} only depends on Z_n and by with having Z_n we have all $(X_{n+1}, X_n, X_{n-1}, \dots, X_{n-K+1})$ so we can say Z_n only depends on Z_n so it's a markov chain of order 1.

Now, we can rewrite the property given in the question as:

$$P[X_{n+1}|Z_n] \Rightarrow P[Z_{n+1}|Z_n].$$

This implies that the conditional probability of the next state, given the memory Z_n, is equal to the conditional probability of the next state, given only the current state Z_n.

2 Telecommunication classes

Theoretical Question 5

To prove that j $\rightarrow$ i if and only if there exists a $0 \le n$ and the states $s_0, s_1, \dots, s_n$ exist, where $i = s_0$ and $j = s_n$, such that:

$$p_{s_0 s_1} p_{s_1 s_2} p_{s_2 s_3} \cdots p_{s_{n-1} s_n} = \prod_{i=1}^{n} p_{s_{i-1} s_i} > 0,$$

we will split the proof into two parts: the forward implication and the backward implication.

Forward Implication: Assume that j $\rightarrow$ i. This means that there exists a sequence of states $s_0, s_1, s_2, \dots, s_n$ such that $s_0 = i$ and $s_n = j$, and each transition $s_{k-1} \rightarrow s_k$ has a non-zero transition probability $p_{s_{k-1} s_k} > 0$.

If we take n as the number of transitions from s_0 to s_n and set s_k to be the state at step k, we can rewrite the product of transition probabilities as:

$$p_{s_0 s_1} p_{s_1 s_2} p_{s_2 s_3} \cdots p_{s_{n-1} s_n} = \prod_{i=1}^{n} p_{s_{i-1} s_i}.$$

Since each transition probability $p_{s_{k-1} s_k}$ in the sequence is non-zero, it follows that the product of these probabilities is also greater than 0:

$$\prod_{i=1}^{n} p_{s_{i-1} s_i} > 0.$$

Backward Implication: Assume that there exists a $0 \leq n$ and the states $s_0, s_1, \ldots, s_n$ exist such that $i = s_0$ and $j = s_n$, and the product of transition probabilities $\prod_{i=1}^{n} p_{s_{i-1}s_i} > 0$. We want to show that j → i.

From the assumption, we have that $p_{s_0s_1}p_{s_1s_2}p_{s_2s_3}\cdots p_{s_{n-1}s_n} > 0$. Since each transition probability is non-zero, it implies that $p_{s_0s_1} > 0$, $p_{s_1s_2} > 0$, and so on, up to $p_{s_{n-1}s_n} > 0$. This implies that there exists a sequence of states $s_0, s_1, s_2, \ldots, s_n$ with $s_0 = i$ and $s_n = j$, such that each transition $s_{k-1} \to s_k$ has a non-zero probability.

Theoretical Question 6

To show that being in relation is an equivalence relation, we need to prove that it satisfies the following three properties: reflexivity, symmetry, and transitivity.

1. Reflexivity: For any state x in set χ, x is related to itself. This property is satisfied since every element x in χ is connected to itself as there must be a transition probability from x to x.

2. Symmetry: For any states x and y in set χ, if x is related to y, then y is related to x. This property is satisfied because if there is a transition probability from x to y, there must also be a transition probability from y to x.

3. Transitivity: For any states x, y, and z in set χ, if x is related to y and y is related to z, then x is related to z. This property is satisfied because if there is a transition probability from x to y and a transition probability from y to z, there must be a corresponding transition probability from x to z.

another way to say this:
Since $x \to y$, we have $p_{xy}(n) > 0$ for some n, and since $y \to z$, we also have $p_{yz}(m) > 0$ for some m. Then, by the Chapman-Kolmogorov equations, we have

$$p_{xz}(n+m) = \sum_{l \in \chi} p_{xl}(n)p_{lz}(m) \geq p_{xy}(n)p_{yz}(m) > 0,$$

from just picking out the $l = y$ term in the sum. So $x \to z$ too. The same argument with z and x swapped gives $z \to x$ also, so $x \leftrightarrow z$.

Since the relation satisfies reflexivity, symmetry, and transitivity, it is an equivalence relation.

Now, let's consider the equivalence classes formed by this equivalence relation. Each equivalence class will consist of states that are mutually connected. In other words, all states within an equivalence class can reach one another through a sequence of transitions.

These equivalence classes are referred to as communication classes in the context of the Markov chain. Within each communication class, the states are strongly connected, meaning there are non-zero probabilities of transitioning between any two states within the class.

Therefore, χ, the set of all states, can be divided into a number of communication classes, where each class consists of states that are mutually connected.

practical Question 1

all libraries needed for practical question

```python
from markovchain import MarkovChain
import sympy as sp
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```python

P = np.array([
     [0.5, 0 , 0 , 0 , 0.5],
     [0 , 0.5,0 , 0.5 , 0],
     [0 , 0,1 , 0 , 0],
     [0 , 0.25,0.25 , 0.25 , 0.25],
     [0.5, 0 , 0 , 0 , 0.5]
     ]
                )
states = ['A', 'B' , 'C' , 'D' , 'E']
mc = MarkovChain(P, states)
mc.draw()
```

output:

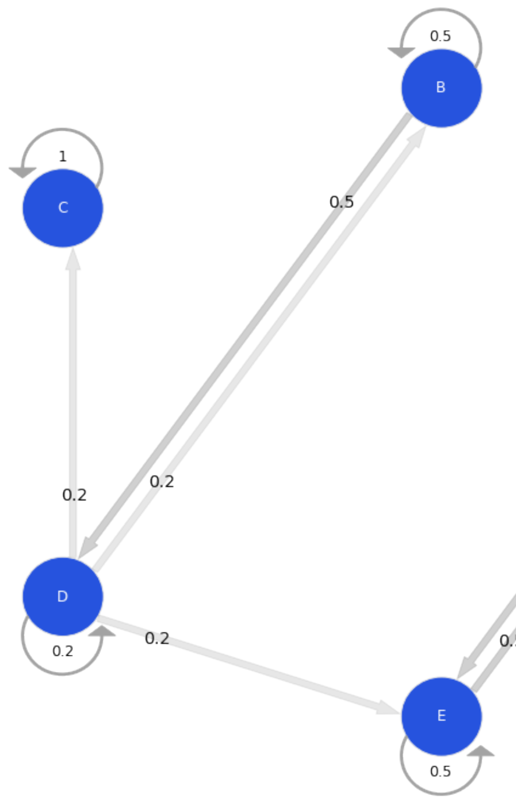


Figure 2

```python
def isreducible(P):
  numStates = P.shape[0]
  zeroTol = numStates*np.spacing(1)
  Z = P > zeroTol
  A1 = (np.eye(numStates)+Z)
  Q = np.linalg.matrix_power(A1,(numStates-1));
  F = np.column_stack([Q.flatten()])
  F1 = np.any(F < zeroTol)
  tf = np.full(1 ,F1)[0]
  return tf
```

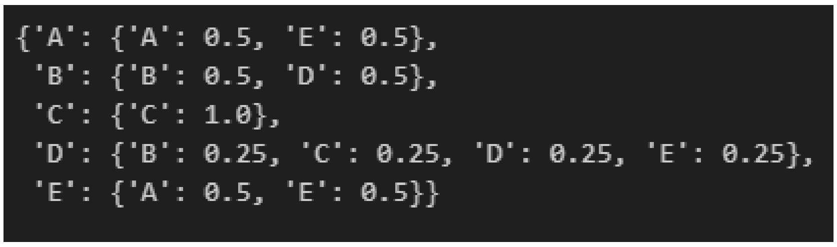
```python
print("It's reduciblele ---> " + str(isreducible(P)))
```

output : It's reduciblele — true

```python
def convert_to_graph(P, states:list):
  graph={}
  for i in range(len(states)):
    row={}
    for j in range(len(states)):
      if P[i][j] >0.0 :
        row[states[j]] = P[i][j]
    graph[states[i]] = row
  return graph
```

```
convert_to_graph(P.tolist(), states)
```

output:



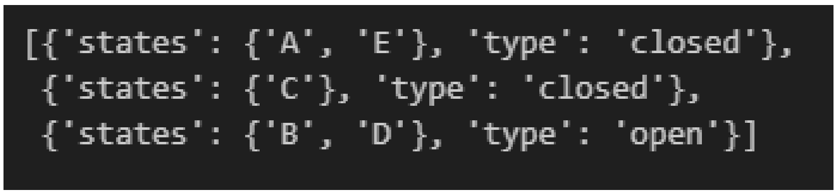
```
{'A': {'A': 0.5, 'E': 0.5},
 'B': {'B': 0.5, 'D': 0.5},
 'C': {'C': 1.0},
 'D': {'B': 0.25, 'C': 0.25, 'D': 0.25, 'E': 0.25},
 'E': {'A': 0.5, 'E': 0.5}}
```

Figure 3

```
def strongly_connected_components(graph):
    index, lowlinks = {}, {}
    stack, result = [], []
    index_counter = [0]

    def connect(node):
        index[node] = index_counter[0]
        lowlinks[node] = index_counter[0]
        index_counter[0] += 1
        stack.append(node)

        try:
            successors = graph[node]
        except KeyError:
            successors = []

        for succ in successors:
            if succ not in lowlinks:
                connect(succ)
                lowlinks[node] = min(lowlinks[node], lowlinks[
    succ])
            elif succ in stack:
                lowlinks[node] = min(lowlinks[node], index[succ])

        if lowlinks[node] == index[node]:
            connected_component = set()

            neighbours = set()
            while True:
                succ = stack.pop()
                connected_component.add(succ)

                for nb in graph[succ]:
                    neighbours.add(nb)
                if succ == node:
                    break

            cp_type = "closed" if neighbours <= connected_component else "open"

            result.append({
                "states": connected_component,
                "type": cp_type
            })

    for node in graph:
        if node not in lowlinks:
            connect(node)

    return result
```

```
strongly_connected_components(convert_to_graph(P.tolist(), states))
```

output:



```
[{'states': {'A', 'E'}, 'type': 'closed'},
 {'states': {'C'}, 'type': 'closed'},
 {'states': {'B', 'D'}, 'type': 'open'}]
```

Figure 4

Theoretical Question 7

To prove that in every finite Markov chain, there is at least one recurrent class, we will use a proof by contradiction.

Assume that in a finite Markov chain, there are no recurrent classes. This implies that all states belong to transient classes only.

In a transient class, there exists at least one state that can enter another class and never return. Let's start with an arbitrary state i in the Markov chain and denote the class to which it belongs as C_i. Since C_i is a transient class, there exists another class C_j such that the probability of transitioning from C_i to C_j is nonzero.

Let's consider the state j in class C_j and the probability of transitioning from i to j denoted as p_{ij}. Since the Markov chain is finite, there are only finitely many states. Therefore, we cannot have an infinite number of transient classes.

Starting at state i, if we follow the transition to state j, and from state j, follow transitions to other states, eventually we will either reach a recurrent class or revisit a previously visited state. However, since we assume that all classes are transient, we cannot reach a recurrent class. This means that the chain either revisits a previously visited state or enters an infinite loop.

If the chain enters an infinite loop, it will never leave that loop, contradicting the assumption that the Markov chain is finite. If the chain revisits a previously visited state, it forms a closed loop. However, as we assumed that all classes are transient and do not contain any recurrent states, there cannot be a closed loop, which again contradicts our assumption.

Theoretical Question 8

first of all f_{ii} is a function of n which is not mentioned in the question the probability the we come back to i after n transition is $1 - f_{ii}(n)$ and the probability that we don't come back to it (meaning we leave it for good and it is not accessible after that) is $f_{ii}(n)$ so we can model this with $geometric\,(1 - f_{ii}(n))$ as since it is the same as geometric distribution definition

Theoretical Question 9

Suppose that $A \leftrightarrow B.$. The result is trivial if $x = y$, so let's assume that $x \neq y$. Recall that there exist $j, k \in \mathbb{N}_+$ such that $P^j(x, y) > 0$ and $P^k(y, x) > 0$. But then $P^{j+k}(x, x) \geq P^j(x, y)P^k(y, x)$ and hence $d(x)|(j + k)$. Suppose now that n is a positive integer with $P^n(y, y) > 0$. Then $P^{j+k+n}(x, x) \neq P^j(x, y)P^n(y, y)P^k(y, x)$ and hence $d(x)|(j + k + n)$. It follows that $d(x)|n$. From the definition of period, $d(x)|d(y)$. Reversing the roles of x and y we also have $d(y)|d(x)$. Hence $d(x) = d(y)$

Theoretical Question 10

To go from one state to another state and then return to the initial state, we can say that we need to go back every way we go, so the number of steps will always remain a multiple of two.
So all n with positive $p_{ii}^{(n)}$ must be even.
now we need to prove that 2 is also in the set, since this is true for 2 either we go one step forward and then come back or we go one step back and then go forward the probability of that is $2p(1-p) > 0$ so we can say :
according to the definition of periodicity, since $d|2$ and $d \geq 2$ so it can be said that the periodicity will be equal to 2

Practical Question 2

sample for code sigma =50 and p=0.5

```
n = 1
p = 0.5
res = []
for i in range(1000):
  x = np.random.binomial(n, p, size=200)
  x[ x == 0] = -1
  res.append(sum(x))
x = plt.hist(res, bins=100,density=True)
E=plt.title("sigma=200 and p=0.5")
```

output:

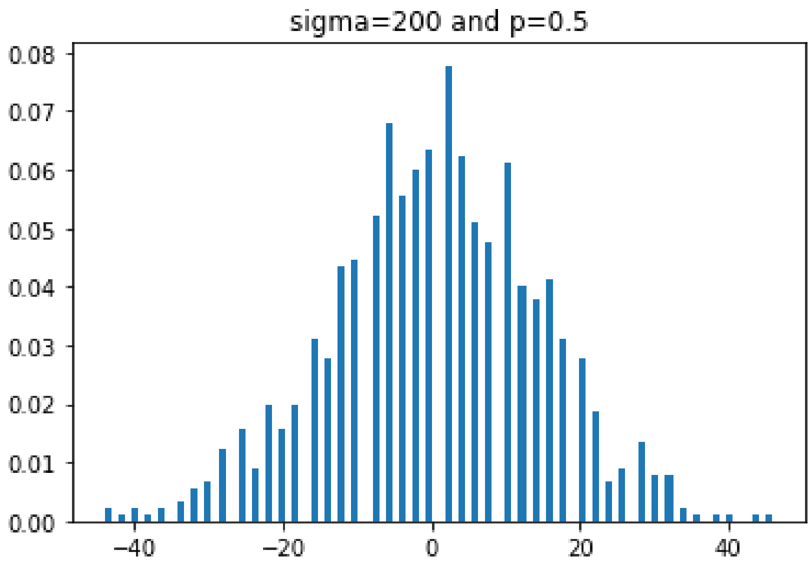


Figure 5

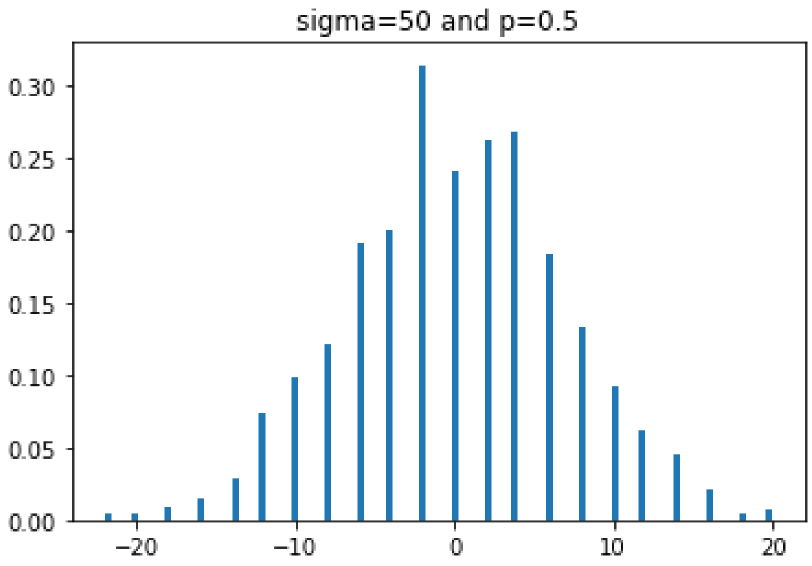


Figure 6

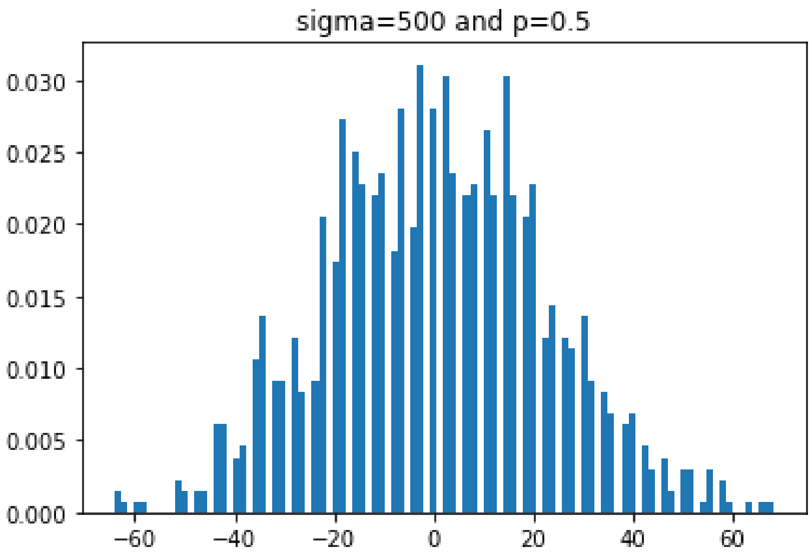


Figure 7

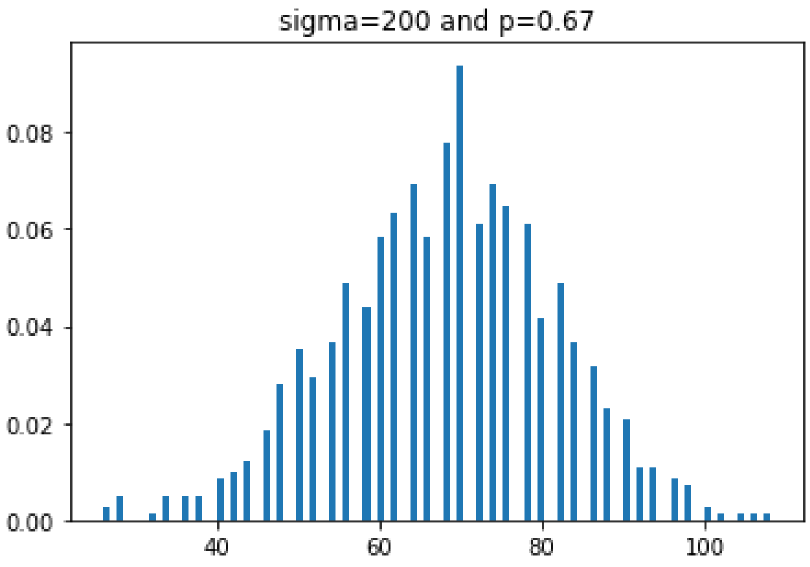


Figure 8

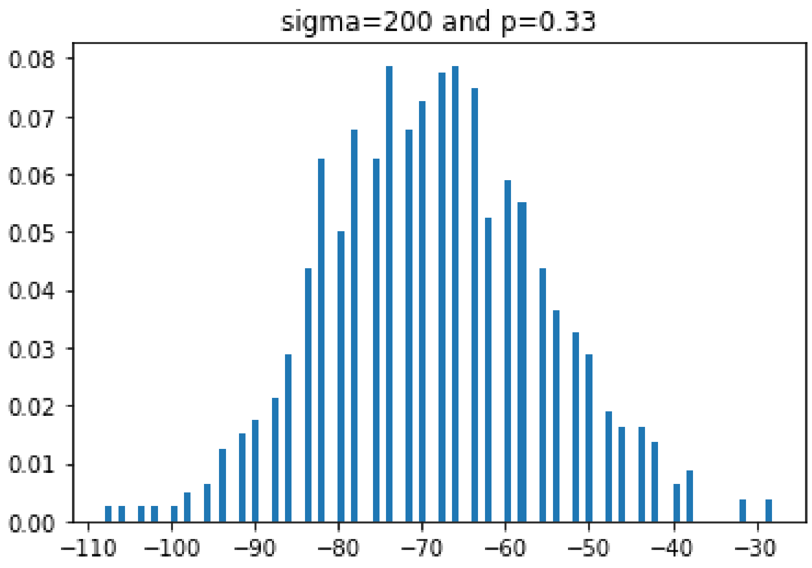


Figure 9

we started from 0 and we wanted to produce Bernoulli random variable to move forward with probability p if random variable was 1 and move backward with probability 1-p if random variable was 0 so we used $np.random.binomial(1, p)$ to produce Bernoulli random variable we changed random variable which equals 0 to -1 and calculate the sum of all produced random variables to measure resultant of movement which is 0 and it's OK because expected value of movement theoretically is 0(Ex = 1*p + (-1)*(1-p) if p=0.5 then Ex=0) so the final point is 0 we observed that sigma doesn't have significance effect on the final point but if we increase p the final point shift to right and if we decrease p the final point shift to left

3 stationary distribution

Theoretical Question 11

The stationary distribution of a Markov chain represents the long-term probabilities of being in each state as time approaches infinity. It is a probability distribution that remains unchanged over time, meaning that

$$\pi \mathbf{P} = \pi,$$

where π is the row vector representing the stationary distribution and $\mathbf{P}$ is the transition matrix.

On the other hand, the eigenvectors of the transfer matrix are the vectors that, when multiplied by the matrix, maintain only a scalar scaling effect. Mathematically, the equation defining an eigenvector $\mathbf{v}$ is:

$$\mathbf{P}\mathbf{v} = \lambda\mathbf{v},$$

where λ is the eigenvalue associated with the eigenvector $\mathbf{v}$.

The relationship between the stationary distribution and the eigenvectors lies in the fact that the stationary distribution vector is proportional to the eigenvector corresponding to the eigenvalue of 1. In other words, the eigenvector of 1 represents the stationary distribution, up to a scalar multiple.

This can be further formalized by noting that as the power of the transfer matrix approaches infinity

To prove the existence of the stationary distribution,

Before we start, Let us call a vector ν a stationary vector if νP=ν This is exactly like a stationary distribution, except without the normalisation condition that it has to sum to 1. Suppose that (X_n) is recurrent (either positive or null, for the moment). Our first task will be to find a stationary vector. Fix an initial state k, and let ν_i be the expected number of visits to i before we return back to k. That is,

$$\begin{aligned}\nu_i &= \mathbb{E}\left(\#\text{ visits to } i \text{ before returning to } k \mid X_0 = k\right)\\ &= \mathbb{E}\sum_{n=1}^{M_k} \mathbb{P}\left(X_n = i \mid X_0 = k\right)\\ &= \sum_{n=1}^{\infty} \mathbb{P}\left(X_n = i \text{ and } n \leq M_k \mid X_0 = k\right),\end{aligned}$$

where M_k is the return time, . Let us note for later use that, under this definition, $\nu_k = 1$, because the only visit to k is the return to k itself.

Since ν is counting the number of visits to different states in a certain (random) time, it seems plausible that $\boldsymbol{\nu}$ suitably normalised could be a stationary distribution, meaning that $\boldsymbol{\nu}$ itself could be a stationary vector. Let's check. We want to show that $\sum_i \nu_i p_{ij} = \nu_j$. Let's see what we have:

$$\begin{aligned}\sum_{i\in\mathcal{S}} \nu_i p_{ij} &= \sum_{i\in\mathcal{S}}\sum_{n=1}^{\infty} \mathbb{P}\left(X_n = i \text{ and } n \leq M_k \mid X_0 = k\right) p_{ij}\\ &= \sum_{n=1}^{\infty}\sum_{i\in\mathcal{S}} \mathbb{P}\left(X_n = i \text{ and } X_{n+1} = j \text{ and } n \leq M_k \mid X_0 = k\right)\\ &= \sum_{n=1}^{\infty} \mathbb{P}\left(X_{n+1} = j \text{ and } n \leq M_k \mid X_0 = k\right).\end{aligned}$$

(Exchanging the order of the sums is legitimate, because recurrence of the chain means that M_k is finite with probability 1.) We can now do a cheeky bit of monkeying around with the index n, by swapping out the visit to k at time M_k with the visit to k at time 0 . This means instead of counting the visits from 1 to M_k, we can count the visits from 0 to $M_k - 1$. Shuffling the index about, we get

$$\begin{aligned}\sum_{i\in\mathcal{S}} \nu_i p_{ij} &= \sum_{n=0}^{\infty} \mathbb{P}\left(X_{n+1} = j \text{ and } n \le M_k - 1 \mid X_0 = k\right) \\ &= \sum_{n+1=1}^{\infty} \mathbb{P}\left(X_{n+1} = j \text{ and } n+1 \le M_k \mid X_0 = k\right) \\ &= \sum_{n=1}^{\infty} \mathbb{P}\left(X_n = j \text{ and } n \le M_k \mid X_0 = k\right) \\ &= \nu_j.\end{aligned}$$

So ν is indeed a stationary vector. We now want normalise $\boldsymbol{\nu}$ into a stationary distribution by dividing through by $\sum_i \nu_i$. We can do this if $\sum_i \nu_i$ is finite. But $\sum_i \nu_i$ is the expected total number of visits to all states before return to k, which is precisely the expected return time μ_k. Now we use the assumption that (X_n) is positive recurrent. This means that μ_k is finite, so $\pi = (1/\mu_k)\,\nu$ is a stationary distribution.

Theoretical Question 12

Suppose the Markov chain is irreducible and positive recurrent, and suppose π is a stationary distribution. We want to show that $\pi_i = 1/\mu_i$ for all i. The only equation we have for μ_k is (12.2)

$$\mu_k = 1 + \sum_j p_{kj}\eta_{jk}$$

Since that involves the expected hitting times η_{ik}, let's write down the equation for them too: (12.3)

$$\eta_{ik} = 1 + \sum_j p_{ij}\eta_{jk} \quad \text{for all } i \ne k$$

In order to apply the fact that π is a stationary distribution, we'd like to get these into an equation with $\sum_i \pi_i p_{ij}$ in it. Here's a way we can do that: Take (12.3), multiply it by π_i and sum over all $i \ne k$, to get (12.4)

$$\sum_i \pi_i \eta_{ik} = \sum_{i\ne k} \pi_i + \sum_j \sum_{i\ne k} \pi_i p_{ij}\eta_{jk}$$

(The sum on the left can be over all i, since $\eta_{kk} = 0$.) Also, take (12.2) and multiply it by π_k to get (12.5)

$$\pi_k \mu_k = \pi_k + \sum_j \pi_k p_{kj} \eta_{jk}$$

Now add (12.4) and (12.5) together to get

$$\sum_i \pi_i \eta_{ik} + \pi_k \mu_k = \sum_i \pi_i + \sum_j \sum_i \pi_i p_{ij} \eta_{jk}$$

We can now use $\sum_i \pi_i p_{ij} = \pi_j$, along with $\sum_i \pi_i = 1$, to get

$$\sum_i \pi_i \eta_{ik} + \pi_k \mu_k = 1 + \sum_j \pi_j \eta_{jk}$$

But the first term on the left and the last term on the right are equal, and because the Markov chain is irreducible and positive recurrent, they are finite. Thus we're allowed to subtract them, and we get $\pi_k \mu_k = 1$, which is indeed $\pi_k = 1/\mu_k$. We can repeat the argument for every choice of k.

Now, let's consider the backward implication. If a Markov chain has a unique stationary distribution, it does not necessarily mean that it is a steady state Markov For example, consider a Markov chain with two states, where the transition probability from state 1 to state 2 is 1, and the transition probability from state 2 to state 1 is 0. This Markov chain has a unique stationary distribution of (1, 0), but state 2 is transient and not recurrent

Theoretical Question 13

To determine if the chain is irreducible, we need to check if all states communicate with each other.
we must prove for every $s_0 = i, s_n = j$ they are in relation
suppose $j > i$ and $s_k = k + i$ we want to go from s_k to s_{k+1} the probability that is $p > 0$ therefore we can conclude $\prod_{k=0}^{j-i-1} p_{s_{k+1}s_k} = p^{j-i} > 0$ so $i \to j$
we can make the same conclusion for $i \leftarrow j$ since $q^{j-i} > 0$
if $j < i$ the same argument is held
for state 0 and N we have a self loop so they have period of 1 and they are not periodical but for other state we at least have to go over two step to come back and we must comeback the whole way we go so it counts even it is like theoretical question 10 which we proved have the period of 2 so state 1 ,2 , ... , N-1 are periodical with preiod 2 based on 3.6 definition.

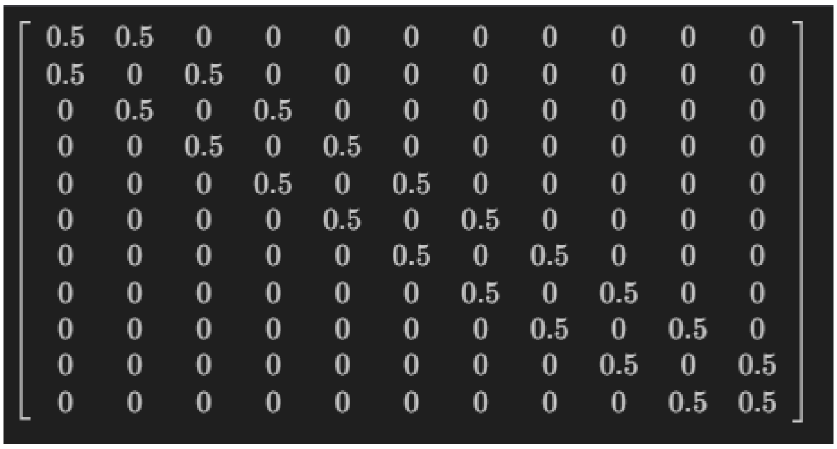
Practical Question 3

```python
n = 1
p = 0.5
sit = 0
N=10
for i in range(100000):
  x = np.random.binomial(n, p, size=1)
  if (sit==0 and x[0]==1):
    sit+=1
  if (0<sit<N and x[0]==1):
    sit+=1
  if (0<sit<N and x[0]==0):
    sit-=1
  if (sit==N and x[0]==0):
    sit-=1
print("Finally we are on location "+str(sit))
```

output: Finally we are on location 9
we started from 0 and we wanted to produce bernouli random variable to move forward with probability p if random variable was 1 and move backward with probability 1-p if random variable was 0 (except point 0 and N) so we used np.random.binomial(1,p) to produce bernouli random variable and sit is our situation variable

```python
q = 0.5
p=0.5
matrix = [
     [q, p, 0, 0, 0, 0, 0, 0, 0, 0, 0],
     [q, 0, p, 0, 0, 0, 0, 0, 0, 0, 0],
     [0, q, 0, p, 0, 0, 0, 0, 0, 0, 0],
     [0, 0, q, 0, p, 0, 0, 0, 0, 0, 0],
     [0, 0, 0, q, 0, p, 0, 0, 0, 0, 0],
     [0, 0, 0, 0, q, 0, p, 0, 0, 0, 0],
     [0, 0, 0, 0, 0, q, 0, p, 0, 0, 0],
     [0, 0, 0, 0, 0, 0, q, 0, p, 0, 0],
     [0, 0, 0, 0, 0, 0, 0, q, 0, p, 0],
     [0, 0, 0, 0, 0, 0, 0, 0, q, 0, p],
     [0, 0, 0, 0, 0, 0, 0, 0, 0, q, p],
     ]
mat_a = sp.Matrix(matrix)
a = mat_a.eigenvects()
mat_a
```

output:

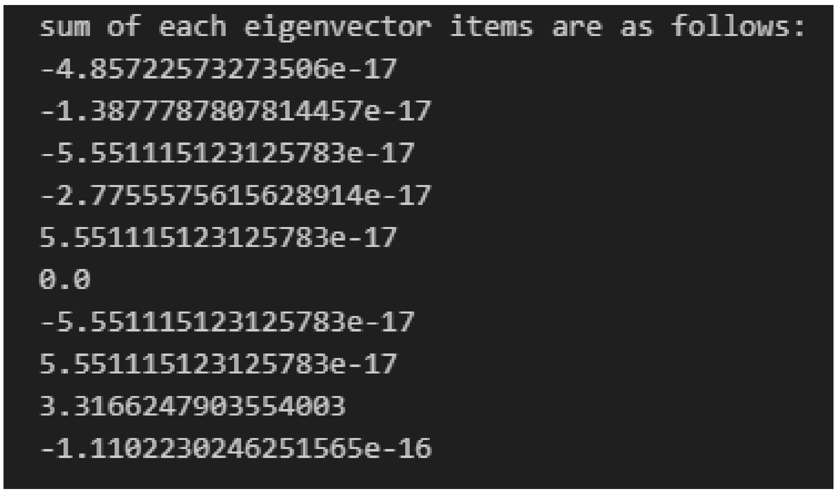


```
⎡ 0.5  0.5   0    0    0    0    0    0    0    0    0  ⎤
  0.5   0   0.5   0    0    0    0    0    0    0    0
   0   0.5   0   0.5   0    0    0    0    0    0    0
   0    0   0.5   0   0.5   0    0    0    0    0    0
   0    0    0   0.5   0   0.5   0    0    0    0    0
   0    0    0    0   0.5   0   0.5   0    0    0    0
   0    0    0    0    0   0.5   0   0.5   0    0    0
   0    0    0    0    0    0   0.5   0   0.5   0    0
   0    0    0    0    0    0    0   0.5   0   0.5   0
   0    0    0    0    0    0    0    0   0.5   0   0.5
⎣  0    0    0    0    0    0    0    0    0   0.5  0.5 ⎦
```

Figure 10

```python
print("sum of each eigenvector items are as follows:")
for i in range(10):
  ss = np.array(a[i][2]).astype(np.float64)
  print(ss.sum())
```

output:



```
sum of each eigenvector items are as follows:
-4.85722573273506e-17
-1.3877787807814457e-17
-5.551115123125783e-17
-2.7755575615628914e-17
5.551115123125783e-17
0.0
-5.551115123125783e-17
5.551115123125783e-17
3.3166247903554003
-1.1102230246251565e-16
```

Figure 11

In this section for calculationg steady state theorically first we formed transfer matrix and found eigenvalue and eigenvector of this matrix to see wether none of its eigenvector has sum of items equal to 1 or not and we saw they hadn't so we found that this problem has no steady state

Theoretical Question 14

we can calculate the number of doctors at the beginning of the next week by subtracting the number of doctors that got sick from the number of doctors at the beginning of the current week. This calculation depends only on the current week, and not on any previous weeks. so X(n) is a Markov Chain.
the states are the number of doctors available at the beginning of each week. Since each doctor is independent of the other doctor and they get sick with probability p, we can use a binomial distribution to calculate the probability of Y(n) doctors getting sick in any given week.

$$P(Y(n) = y|X(n) = x) = \binom{x}{y} p^y (1-p)^{x-y}$$

The transition probabilities are calculated as follows:

$Pij = P(Xn + 1 = j|Xn = i)$

where i and j represent the number of doctors available at the beginning of each week.

Since we know that if a doctor gets sick, he will be replaced by another doctor and this process takes a week, we can calculate the transition probabilities as follows:

$$P_{ij} = P(X_{n+1} = j|X(n) = i)$$

Since the number of doctors at the beginning of the next week is equal to the number of doctors at the beginning of the current week minus the number of doctors that get sick, we have:

$$P_{ij} = P(Y(n) = i - j|X(n) = i) = \binom{i}{i-j} p^{i-j} (1-p)^j$$

for i = 0, 1, ..., N and j = 0, 1, ..., i.

To find the stationary distribution, we need to solve the following system of linear equations:

$$\pi P = \pi$$

with the constraint that the elements of π sum to 1:

$$\sum_{i=0}^{N} \pi_i = 1$$

The solution is:

$$\pi_i = \binom{N}{i} p^i (1-p)^{N-i}$$

where N is the total number of doctors and i is the number of doctors at the beginning of the week. This distribution is a binomial distribution with parameters N and p.

Once we have found the stationary distribution π, we can calculate the expected value of the number of doctors in the stationary state as follows:

$$E[X] = \sum_{i=0}^{N} i \cdot \pi_i$$

Substituting the expression for π_i, we get:

$$E(X) = \sum_{i=0}^{N} i \binom{N}{i} p^i (1-p)^{N-i}$$

we could do all the calculation but since we that this is binomial distribution we can say that

$$E(X) = Np$$

4 hit time

Practical Question 4

```
start = np.random.randint(1,5)
out= []
lens= []
for i in range(1000000):
  k = start
  res= [k]
  while True:
    a = np.random.randint(k+1,6)
    res.append(a)
    k= a
    if a==5:
      break
  out.append(res)
  lens.append(len(res)-1)
```

output: Mean of length we will travel to reach location 5 is 1.500026 First we chose our starting point for this we used *np.random.randint* because starting point has discrete uniform distribution
Then we went from starting point to next points until we reached point 5 and each step we needed to choice next point uniformly so we used *np.random.randint* we did this 10000000 times and saved our move path in array res to calculate the distribution of possible movements then we used
function *len*() to calculate movement length and its distribution we observed that the mean of length we will travel to reach location 5 ranges from 1.4 to 1.8 which its mean is 1.6 and that is so closed to what we calculated theoretically

Theoretical Question 15

For each i in A it's obvious that we are already in a state of A so the probability is 1
For $\forall i \notin A$ after this state we may go to one of state$j \in X$ so the probability is the sum of each next state
And for $\forall j \in X$, we go to state j (from now state i) with probability P_{ij} and then the probability to get a state in A from state j is h_j^A which also is a variable of equation system , so we can rewrite the answer as the equation of below

$$\begin{cases} h_i^A = 1 & \forall i \in A \\ h_i^A = \sum_{j \in X} p_{ij} h_j^A & \forall i \notin A \end{cases} \tag{1}$$

To show that h_i^A is the minimal solution, suppose $x = (x_i : i$ non-negative solution, i.e.

$$x_i = \begin{cases} 1 & i \in A \\ \sum_{j \in S} p_{i,j} x_j A & i \notin A \end{cases},$$

If $i \in A$, we have $h_i^A = x_i = 1$. Otherwise, we can write

$$\begin{aligned} x_i &= \sum_j p_{i,j} x_j \\ &= \sum_{j \in A} p_{i,j} x_j + \sum_{j \notin A} p_{i,j} x_j \\ &= \sum_{j \in A} p_{i,j} + \sum_{j \notin A} p_{i,j} x_j \\ &\geq \sum_{j \in A} p_{i,j} \\ &= \mathbb{P}_i \left(H^A = 1 \right). \end{aligned}$$

11 2 Classification of chains and statdB Markov Chains (Theorems wit) By iterating this process, we can write

$$\begin{aligned}
x_i &= \sum_{j\in A} p_{i,j} + \sum_{j\notin A} p_{i,j}\left(\sum_{k} p_{i,k}x_k\right)\\
&= \sum_{j\in A} p_{i,j} + \sum_{j\notin A} p_{i,j}\left(\sum_{k\in A} p_{i,k}x_k + \sum_{k\notin A} p_{i,k}x_k\right)\\
&\geq \mathbb{P}_i\left(H^A = 1\right) + \sum_{j\in A, k\in A} p_{i,j}p_{j,k}\\
&= \mathbb{P}_i\left(H^A = 1\right) + \mathbb{P}_i\left(H^A = 2\right)\\
&= \mathbb{P}_i\left(H^A \leq 2\right).
\end{aligned}$$

By induction, we obtain

$$x_i \geq \mathbb{P}_i\left(H^A \leq n\right)$$

for all n. Taking the limit as $n \to \infty$, we get

$$x_i \geq \mathbb{P}_i\left(H^A \leq \infty\right) = h_i^A.$$

So h_i^A is minimal.

Theoretical Question 16

$$\begin{cases} k_i^A = 0 & \text{for } i \in A\\ k_i^A = 1 + \sum_{j\notin A} p_{ij}k_j^A & \text{for } i \notin A \end{cases}$$

First of all for $\forall i \in A$ we know that inf of needed step to get to a state in A is zero $\left(H^A \mid X_0 = i = 0\right)$ so basically k is also zero. Now for $\forall i \notin A$ assume that we are in state i we sum over next state possibilities , for $\forall j \in X$ with probability p_{ij} we go to state j and the $H^A \mid X_1 = j$ is plus one and we know that E is a linear operation so we have : we also know that $\sum_{j\in X} p_{ij} = 1$ because it is all possibilities over state i also E(constant) = constant

$$k_i^A = \mathbb{E}\left[H^A \mid X_0 = i\right] = \sum_{j\in X} \mathbb{E}\left[p_{ij}\left(H^A + 1\right) \mid X_0 = j\right]$$

$$k_i^A = \sum_{j\in X} \mathbb{E}\left[p_{ij}\right] + \sum_{j\in X} p_{ij}\cdot \mathbb{E}\left[H^A \mid X_0 = j\right] \Rightarrow k_i^A = 1 + \sum_{j\in X} p_{ij}\cdot k_j^A$$

which is what we expected

Now let $(y_i : i \in S)$ be a non-negative solution. We show that $y_i \geq k_i^A$. If $i \in A$, we get $y_i = k_i^A = 0$. Otherwise, suppose $i \notin A$. Then we have

$$\begin{aligned}
y_i &= 1 + \sum_j p_{i,j} y_j \\
&= 1 + \sum_{j \in A} p_{i,j} y_j + \sum_{j \notin A} p_{i,j} y_j \\
&= 1 + \sum_{j \notin A} p_{i,j} y_j \\
&= 1 + \sum_{j \notin A} p_{i,j} \left(1 + \sum_{k \notin A} p_{j,k} y_k \right) \\
&\geq 1 + \sum_{j \notin A} p_{i,j} \\
&= \mathbb{P}_i \left(H^A \geq 1 \right) + \mathbb{P}_i \left(H^A \geq 2 \right).
\end{aligned}$$

By induction, we know that

$$y_i \geq \mathbb{P}_i \left(H^A \geq 1 \right) + \cdots + \mathbb{P}_i \left(H^A \geq n \right)$$

for all n. Let $n \to \infty$. Then we get

$$y_i \geq \sum_{m \geq 1} \mathbb{P}_i \left(H^A \geq m \right) = \sum_{m \geq 1} m \mathbb{P}_i \left(H^A = m \right) = k_i^A.$$

Theoretical Question 17

$$h^{\{4\}} = \{h_1^A, h_2^A, h_3^A, h_4^A\}$$
$$A = \{4\}$$

Here A has only one member 4 , so we need to calculate the probability of going to 4 from each state:
based on question 15 we can see : state 4 : $h_4^A = 1$
line 3 for state 3 $p_{32} = p_{34} = 0.5$ state 3 : $h_3^A = 0.5.h_2^A + 0.5$
line 2 for state 2 $p_{21} = p_{23} = 0.5$ state 2 : $h_2^A = 0.5.h_1^A + 0.5h_3^A$
line 1 for state 1 $p_{11} = 1$ no equation , so if we are in state 1 it will never change to state4 and for good it remain state 1 so h is zero.
solving equations:
$h_2^A = 0.5.h_1^A + 0.5h_3^A = 0.5h_3^A \Rightarrow h_3^A = 0.5.(0.5h_3^A) + 0.5 \Rightarrow$

$$h_3^A = \frac{2}{3}, h_2^A = \frac{1}{3}$$

$$h^{\{4\}} = \{0, \frac{1}{3}, \frac{2}{3}, 1\}$$

Now next question:

$$A = \{1, 4\}$$

$$k^{\{1,4\}} = \{k_1^A, k_2^A, k_3^A, k_4^A\}$$

based on question 16 we can see :
state 4 : $k_4^A = 0$
state 1 : $k_1^A = 0$
line 3 for state 3 $p_{32} = p_{34} = 0.5$ state 3 : $k_3^A = 1 + 0.5.k_2^A$
line 2 for state 2 $p_{21} = p_{23} = 0.5$ state 2 : $k_2^A = 1 + 0.5k_3^A$
solving equations: $k_2^A = 1 + 0.5(1 + 0.5.k_2^A) = 1.5 + 0.25k_2^A \Rightarrow k_2^A = \frac{1.5}{0.75} = 2 = k_3^A$

$$k^{\{1,4\}} = \{0, 2, 2, 0\}$$

Theoretical Question 18

We have explained in question 16 , hitting time for a certain state if it means to have the set A with only one member called state a , we have $k_a^A = 0$ and replace it in other equations and we can solve them based on its transition matrix, the equation can rewritten as a linear equations matrix so we can solve it use those algorithms (like Cramer's rule or Gaussian elimination) to solve it.
Now as we know k_i is expected value of H while X_0 is i, so for each state we have the expected value of H
Suppose we are at state b and $A = \{a\}$ so a is what we want to go, we want distribution of H, H is 1 by probability of p_{ba}, H is n by probability of getting from state b to a in n steps which is p_{ba}^n . By having the transition matrix we can easily find H distribution.

To calculate the hitting time between any two states suppose 1 as state A, 2 as B and 3 as C :
1. $A = \{1\}$

$$\begin{cases} k_1^A = 0 \\ k_2^A = 1 + p_{22}k_2^A + p_{23}k_3^A = 1 + 0.7k_2^A + 0.2k_3^A \\ k_3^A = 1 + 0.4k_2^A + 0.5k_3^A \end{cases}$$

$$k_2^A = 10 \quad , k_3^A = 10$$

2. $A = \{2\}$

$$\begin{cases} k_2^A = 0 \\ k_1^A = 1 + p_{11}k_1^A + p_{13}k_3^A = 1 + 0.8k_1^A + 0.1k_3^A \\ k_3^A = 1 + 0.1k_1^A + 0.5k_3^A \end{cases}$$

$$k_1^A = \frac{20}{3} \quad , k_3^A = \frac{10}{3}$$

3. $A = \{3\}$

$$\begin{cases} k_3^A = 0 \\ k_2^A = 1 + 0.7k_2^A + 0.1k_1^A \\ k_1^A = 1 + 0.1k_2^A + 0.8k_1^A \end{cases}$$

$$k_1^A = 8 \quad , k_2^A = 6$$

stationary distribution: the stationary distribution for Markov Chain with transition matrix P is: $\pi = \pi P$ where the sum of π equals to 1.

so we have:

$$\begin{pmatrix} \pi_1 & \pi_2 & \pi_3 \end{pmatrix} = \begin{pmatrix} \pi_1 & \pi_2 & \pi_3 \end{pmatrix} \begin{pmatrix} 0.8 & 0.1 & 0.1 \\ 0.1 & 0.7 & 0.2 \\ 0.1 & 0.4 & 0.5 \end{pmatrix}$$

considering that $\pi_1 + \pi_2 + \pi_3 = 1$ we have

$$\pi = \begin{pmatrix} \frac{1}{3} & \frac{5}{21} & \frac{3}{7} \end{pmatrix}$$

we know that $k_i = \frac{1}{\pi_i}$:

$$k = \begin{pmatrix} 3 & \frac{21}{5} & \frac{7}{3} \end{pmatrix}$$

for $A = \{1\}$ we know $k^A = (0, 10, 10)$ so we have $(0, \frac{50}{21}, \frac{30}{7})$
the probability of H being zero is $\frac{1}{3}$
being 1 is $\frac{5}{21}p_{21} + \frac{3}{7}p_{31}$ ...
being n is $\frac{5}{21}p_{21}^n + \frac{3}{7}p_{31}^n$
for $A = \{2\}$ we know $k^A = (\frac{20}{3}, 0, \frac{10}{3})$ so we have $(\frac{20}{6}, 0, \frac{30}{21})$
the probability of H being zero is $\frac{5}{21}$
being 1 is $\frac{1}{3}p_{12} + \frac{3}{7}p_{32}$ ...
being n is $\frac{1}{3}p_{12}^n + \frac{3}{7}p_{32}^n$
for $A = \{3\}$ we know $k^A = (8, 6, 0)$ so we have $(\frac{8}{3}, \frac{30}{21}, 0)$

the probability of H being zero is $\frac{3}{7}$
being 1 is $\frac{1}{3}p_{13} + \frac{5}{21}p_{23}$ ...
being n is $\frac{1}{3}p_{13}^n + \frac{5}{21}p_{23}^n$

5 Game theory

Theoretical Question 19

Prisoner's Dilemma

this is called the prisoners paradox and it is a famous problem in game theory The prisoner's dilemma is a paradox in decision analysis in which two individuals acting in their own self-interests do not produce the optimal outcome. Today, the prisoner's dilemma is a paradigmatic example of how strategic thinking between individuals can lead to sub optimal outcomes for both players.
here we define the game :

- Pat an Mat, have been arrested for burning electrical department and are being interrogated in separate rooms.
- Police have no other witnesses, and can only prove the case against them if they can convince at least one of the the criminals to betray their accomplice and testify to the crime.
- Each the criminal is faced with the choice to cooperate with their accomplice and remain silent or to defect from the gang and testify for the prosecution.
- If they both co-operate and remain silent, then the authorities will only be able to convict them on a lesser charge resulting in one year in jail for each (1 year for Pat + 1 year for Mat = 2 years total jail time).
- If one testifies and the other does not, then the one who testifies will go free and the other will get 20 years (0 years for the one who defects + 20 for the one convicted = 20 years total).
- However, if both testify against the other, each will get five years in jail for being partly responsible for the robbery (5 years for Pat + 5 years for Mat = 10 years total jail time).

Pat\Mat	Mat stays silent (cooperate)	Mat testifies (deflect)
Pat stays silent (Cooperate)	(1,1)	(20,0)
Pat testifies (deflect)	(0,20)	(5,5)

It is assumed that both prisoners understand the nature of the game, have no loyalty to each other, and will have no opportunity for retribution or reward outside of the game. Regardless of what the other decides, each prisoner gets a higher reward by betraying the other (”defecting”). The reasoning involves analyzing both players’ best responses: Mat will either cooperate or defect. If Mat cooperates, Pat should defect, because going free is better than serving 1 year. If Mat defects, Pat should also defect, because serving 5 years is better than serving 20. So, either way, Pat should defect since defecting is Pat’s best response regardless of Mat’s strategy. Parallel reasoning will show that Mat should defect.

Defection always results in a better payoff than cooperation, so it is a strictly dominant strategy for both Pat and Mat. Mutual defection is the only strong Nash equilibrium in the game (i.e., the only outcome from which each player could only do worse by unilaterally changing strategy). The dilemma is that mutual cooperation yields a better outcome than mutual defection but is not the rational outcome because the choice to cooperate, from a self-interested perspective, is irrational. Thus, Prisoner’s dilemma is a game where the Nash equilibrium is not Pareto efficient.

Escape from the Prisoner’s Dilemma

Over time, people have worked out a variety of solutions to prisoner’s dilemmas in order to overcome individual incentives in favor of the common good.

First, in the real world, most economic and other human interactions are repeated more than once. A true prisoner’s dilemma is typically played only once or else it is classified as an iterated prisoner’s dilemma. In an iterated prisoner’s dilemma, the players can choose strategies that reward cooperation or punish defection over time. By repeatedly interacting with the same individuals we can even deliberately move from a one-time prisoner’s dilemma to a repeated prisoner’s dilemma.

Second, people have developed formal institutional strategies to alter the incentives that individual decision-makers face. Collective action to enforce cooperative behavior through reputation, rules, laws, democratic or other collective decision-making procedures, and explicit social punishment for defections transforms many prisoner's dilemmas toward the more collectively beneficial cooperative outcomes.

Last, some people and groups of people have developed psychological and behavioral biases over time such as higher trust in one another, long-term future orientation in repeated interactions, and inclinations toward positive reciprocity of cooperative behavior or negative reciprocity of defecting behaviors. These tendencies may evolve through a kind of natural selection within a society over time or group selection across different competing societies. In effect, they lead groups of individuals to "irrationally" choose outcomes that are actually the most beneficial to all of them together.

Put together, these three factors (the repeated prisoner's dilemmas, formal institutions that break down prisoner's dilemmas, and behavioral biases that undermine "rational" individual choice in prisoner's dilemmas) help resolve the many prisoner's dilemmas we would all otherwise face.

Theoretical Question 20

.1

here we don't have a pure strategy Nash equilibrium because if each player knows what the other one plays they would definitely change their strategy instead we have mix strategy Nash equilibrium suppose the shooter goes to right with probability p (so it goes to left with probably 1-p) and the Goalkeeper goes to right with probability q (so it goes to left with probably 1-q)
here a mixed mix strategy Nash equilibrium happens when changes of p or q while knowing the other players strategy doesn't result in better utility
the game is defined as below since A is a matrix of probability that the shooter scores(shooter's utility) therefor $1 - A$ is the probability that the keeper scores(keeper's utility)

		q	1-q
	shooter\goal keeper	goes right	goes left
p	goes right	(0.8 , 0.2)	(0.15 , 0.85)
1-p	goes left	(0.2 , 0.8)	(0.95 , 0.05)

$$0.2p + 0.8(1-p) = (0.85p + 0.05(1-p)) \Rightarrow p = \frac{15}{28}$$

$$0.8q + 0.15(1-q) = 0.2q + 0.95(1-q) \Rightarrow q = \frac{4}{7}$$

So the Nash Equilibrium is

$$(\frac{15}{28},\frac{13}{28}),(\frac{4}{7},\frac{3}{7})$$

.2

There's no difference for goal keeper if the shooter plays middle so q is the same.

$$0.8q + 0.15(1-q) = 0.2q + 0.95(1-q) \Rightarrow q = \frac{4}{7}$$

And for shooter we set r the probability of middle strategy.

$$U = 0.8p + 0.05 + 0.45r = -0.6p + 0.8 - 0.3r \Rightarrow r = 1 - 1.4p \Rightarrow p < 1/1.4$$

$$U = 0.8p + 0.45(1-1.4p) + 0.05 = 0.5 + 0.17p$$

Therefor maximizing utility we maximize p and it's the same as before.

So the Nash Equilibrium is

$$(\frac{15}{28},\frac{13}{28}),(\frac{4}{7},\frac{3}{7})$$

.3

the game is defined as :

		q	1-q
	shooter\goal keeper	goes left	goes right
p	goes left	$(x\ ,\ 2)$	(0 , 0)
1-p	goes right	(0 , 0)	(2 , 2)

so if $(p, 1-p)$ and $(q, 1-q)$ are Nash equilibrium the equations below must be hold

$$2p = 2(1-p) \Rightarrow p = \frac{1}{2}$$

$$xq = 2(1-q) \Rightarrow q = \frac{2}{2+x}$$

so by increasing x q decreases

Theoretical Question 21

.1

since this is a zero sum game the game matrix is defined as below for a one by one game

A\B	rock	paper	Scissors
rock	(0,0)	(-1,1)	(1,-1)
paper	(1,-1)	(0,0)	(-1,1)
Scissors	(-1,1)	(1,-1)	(0,0)

- if both player play the same thing no one will will (tie) each player utility is equal to zero
- if one player plays rock depending on the other player action Scissors or Paper he/she will get 1 or -1 utility
- if one player plays Paper depending on the other player action rock or Scissors he/she will get 1 or -1 utility
- if one player Scissors depending on the other player action paper or rock he/she will get 1 or -1 utility
- and since it's a zero sum game the second player's utility will always be -utility of the first player

.2

we know that in general form

$$u_i(\sigma_1, \sigma_2) = \sum_{r \in S_1, c \in S_2} \sigma_1(r)\sigma_2(c)u_i(r, c) \tag{2}$$

this is kind of like finding the Expected value since σ_1 and σ_2 are in probability form
so in the question scenario where Farhad plays rock and Parsa player paper we will have $U_1 = 1$ and $U_2 = -1$ this could have easily be calculated without the formula 2 by just looking at the table since its just a pure strategy i

.3

A natural way of representing a two player normal form game is using a bi-matrix. If we assume that $N = 2$ and $S_1 = \{r_i \mid 1 \leq i \leq m\}$ and

$S_2 = \{c_j \mid 1 \leq j \leq n\}$ then a bi-matrix representation of the game considered is shown.

$$\begin{pmatrix} (u_1(r_1,c_1),u_2(r_1,c_1)) & (u_1(r_1,c_2),u_2(r_1,c_2)) & \ldots & (u_1(r_1,c_n),u_2(r_1,c_n)) \\ (u_1(r_2,c_1),u_2(r_2,c_1)) & (u_1(r_2,c_2),u_2(r_2,c_2)) & \cdots & (u_1(r_2,c_n),u_2(r_2,c_n)) \\ \vdots & \ldots & \ldots & \vdots \\ (u_1(r_m,c_1),u_2(r_m,c_1)) & (u_1(r_m,c_2),u_2(r_m,c_2)) & \cdots & (u_1(r_m,c_n),u_2(r_m,c_n)) \end{pmatrix}$$

In an N player normal form game a mixed strategy for player i denoted by $\sigma_i \in [0,1]_R^{|S_i|}$ is a probability distribution over the pure strategies of player i. So:

$$\sum_{j=1}^{|S_i|} (\sigma_i)_j = 1$$

For a given player i we denote the set of mixed strategies as ΔS_i.
here we can use eq: 2 or we could use matrix form of it

$$U_i(\sigma_1,\sigma_2) = \sigma_1 U_i \sigma_2^T$$

.4

It is immediately obvious that this game has no Nash equilibrium in pure strategies: The player who loses or ties can always switch to another strategy and win This game is symmetric, and we shall look for symmetric mixed strategy equilibria first. Let p, q, and $1–p–q$ be the probability that a player chooses R, P, and S respectively. We first argue that we must look only at completely mixed strategies (that is, mixed strategies that put a positive probability on every available pure strategy). Suppose not, so p1 = 0 in some (possibly asymmetric) MSNE. If player 1 never chooses R, then playing P is strictly dominated by S for player 2, so she will play either R or S. However, if player 2 never chooses P, then S is strictly dominated by R for player 1, so player 1 will choose either R or P in equilibrium. However, since player 1 never chooses R, it follows that he must choose P with probability 1. But in this case player 2's optimal strategy will be to play S, to which either R or S are better choices than P. Therefore, p1 = 0 cannot occur in equilibrium. Similar arguments establish that in any equilibrium, any strategy must be completely mixed. We now look for a symmetric equilibrium. Player 1's payoff from R is $p(0) + q(-1) + (1–p-q)(1) = 1-p-2q$. His payoff from P is $2p+q-1$. His payoff from S is $q-p$. In an MSNE, the payoffs from all threepure strategies must be the same, so:

$$1-p-2q = 2p+q-1 = q-p$$

Solving these equalities yields $p = q = 1/3$.

Whenever player 2 plays the three pure strategies with equal probability, player 1 is indifferent between his pure strategies, and hence can play any mixture. In particular, he can play the same mixture as player 2, which would leave player 2 indifferent among his pure strategies. This verifies the first condition in Proposition 1. Because these strategies are completely mixed, we are done. Each player's strategy in the symmetric Nash equilibrium is (1/3, 1/3, 1/3). That is, each player chooses among his three actions with equal probabilities. Is this the only MSNE? We already know that any mixed strategy profile must consist only of completely mixed strategies in equilibrium. Arguing in a way similar to that for the pure strategies, we can show that there can be no equilibrium in which players put different weights on their pure strategies. You should check for MSNE in all combinations. That is, you should check whether there are equilibria, in which one player chooses a pure strategy and the other mixes; equilibria, in which both mix; and equilibria in which neither mixes. Note that the mixtures need not be over the entire strategy spaces, which means you should check every possible subset. Thus, in a 2×2 two-player game, each player has three possible choices: two in pure strategies and one that mixes between them. This yields 9 total combinations to check. Similarly, in a 3 x 3 two-player game, each player has 7 choices: three pure strategies, one completely mixed, and three partially mixed. This means that we must examine 49 combinations! (You can see how this can quickly get out of hand.) Note that in this case, you must check both conditions of Proposition 1.

We have established that Rock Paper Scissors does not have a dominant strategy for either of the players. How do you use that information to incur that there is no Nash equilibrium? Quite simple! If Player 2's strategy is Rock, Player 1 should choose Paper, but if Player 1 chooses Paper, it is profitable for Player 2 to deviate and choose Scissors instead. When player 2 chooses Scissors, Player 1 would want to deviate and choose Rock, and so forth. Thus, we can see that there is no Nash Equilibrium for this game owing to the cyclical manner of the game.

Theoretical Question 22

first we find the best response

$$\frac{\partial U_1(y_1,y_2,y_3)}{\partial y_1} = 1 + y_2 - 2y_1 = 0 \Rightarrow y_1 = BR_1(y_2,y_3) = \frac{1+y_2}{2}$$
$$\frac{\partial U_2(y_1,y_2,y_3)}{\partial y_2} = 1 + y_1 - 2y_2 = 0 \Rightarrow y_2 = BR_2(y_1,y_3) = \frac{1+y_2}{2}$$
$$\frac{\partial U_3(y_1,y_2,y_3)}{\partial y_3} = 10 - y_1 - y_2 - 2y_3 = 0 \Rightarrow y_3 = BR_3(y_1,y_2) = \frac{10-y_1-y_2}{2}$$

the points (y_1^*, y_2^*, y_3^*) are Nash equilibrium if they are the answer to the below equations

$$y_1^* = \frac{1+y_2^*}{2}$$
$$y_2^* = \frac{1+y_1^*}{2}$$
$$y_3^* = \frac{10-y_1^*-y_2^*}{2}$$

Bob and Alice utility function is symmetric so : $y_1^* = y^* = y_2^*$ and therefore we will have

$$y^* = \frac{1+y^*}{2}$$
$$y_3^* = \frac{10-2y^*}{2}$$

so

$$y_1^* = 1\ , \quad y_2^* = 1, \quad y_3^* = 4$$

6 Nash Py

finding Nash Equilibrium

here we use nashpy library there are actually many methods and algorithms computing Nash Equilibrium here we expalain three method that are implemented in nashpy library and then for the game Rock Paper Scissors we calculate Nash Equilibrium using each method separately
firstly we import two library needed

```python
import nashpy as nash
import numpy as np
```

.1 Support enumeration

The support enumeration algorithm implemented in Nashpy is based on the one described in [Nisan2007].(Nisan, Noam, et al., eds. Algorithmic game theory. Vol. 1. Cambridge: Cambridge University Press, 2007.)
The algorithm is as follows:
For a degenerate 2 player game $(A, B) \in R^{m \times n2}$ the following algorithm returns all nash equilibria:
1. For all $1 \leq k_1 \leq m$ and $1 \leq k_2 \leq n$;
2. For all pairs of support (I, J) with $|I| = k_1$ and $|J| = k_2$.
3. Solve the following equations (this ensures we have best responses):

$$\sum_{i \in I} \sigma_{ri} B_{ij} = v \text{ for all } j \in J$$
$$\sum_{j \in J} A_{ij} \sigma_{cj} = u \text{ for all } i \in I$$

4. Solve - $\sum_{i=1}^{m} \sigma_{ri} = 1$ and $\sigma_{ri} \geq 0$ for all i - $\sum_{j=1}^{n} \sigma_{ci} = 1$ and $\sigma_{cj} \geq 0$ for all j
5. Check the best response condition. Repeat steps 3,4 and 5 for all potential support pairs.
Discussion
1. Step 1 is a complete enumeration of all possible strategies that the equilibria could be.
2. Step 2 can be modified to only consider degenerate games ensuring that only supports of equal size
are considered $|I| = |J|$. This is described further in Degenerate games.
3. Step 3 are the linear equations that are to be solved, for a given pair of supports these ensure that neither player has an incentive to move to another strategy on that support.
4. Step 4 is to ensure we have mixed strategies.
5. Step 5 is a final check that there is no better utility outside of the supports.
In Nashpy this is all implemented algebraically using Numpy to solve the linear equations.
python code:

```python
A = np.array([[0, -1, 1], [1, 0, -1], [-1, 1, 0]])
#since our game is symtric:
rps = nash.Game(A)
print(list(rps.support_enumeration()))
```

the output: [(array([0.33333333, 0.33333333, 0.33333333]), array([0.33333333, 0.33333333, 0.33333333]))]
which is the same what we calculated before for mixed strategy

.2 vertex enumeration

The vertex enumeration algorithm implemented in Nashpy is based on the one described in [Nisan2007].

The algorithm is as follows:
For a nondegenerate 2 player game $(A, B) \in \mathbb{R}^{m\times n^2}$ the following algorithm returns all nash equilibria:

1. Obtain the best response Polytopes P and Q
2. For all pairs of vertices of P and Q
3. Check if the pair is fully labeled and return the normalised probability vectors.

Repeat steps 2 and 3 for all pairs of vertices.
discussion:
Before creating the best response Polytope we need to consider the best response Polyhedron. For the row player, this corresponds to the set of all the mixed strategies available to the row player as well as an upper bound on the utilities to the column player. Analogously for the column player:

$$\bar{P} = \left\{(x, v) \in \mathbb{R}^m \times \mathbb{R} \mid x \geq 0, 1x = 1, B^T x \leq 1v\right\}$$
$$\bar{Q} = \{(y, u) \in \mathbb{R}^n \times \mathbb{R} \mid y \geq 0, 1y = 1, Ay \leq 1u\}$$

Note that in both definitions above we have a total of $m + n$ inequalities in the constraints.
For P, the first m of those constraints correspond to the elements of $\boldsymbol{x}$ being greater or equal to 0 . For a given x, if $x_i = 0$, we say that x has label :math i. This corresponds to strategy i not being in the support of $\boldsymbol{x}$.

For the last n of these inequalities, when they are equalities they correspond to whether or not strategy $1 \leq j \leq n$ of the other player is a best response to x. Similarly, if strategy j is a best response to x then we say that x has label $m + j$.

This all holds analogously for the column player. If the labels of a pair of elements of $\bar{P}$ and $\bar{Q}$ give the full set of integers from 1 to $m + n$ then they represent strategies that are best responses to each other. Since, this would imply that either a pure stragey is not played or it is a best response to the other players strategy.

The difficulty with using the best response Polyhedron is that the upper bound on the utilities of both players (u, v) is not known. Importantly, we do not need to know it. Thus, we assume that in both cases: $u = v = 1$ (this simply corresponds to a scaling of our strategy vectors). This allows us to define the best response Polytopes:

$$P = \left\{(x, v) \in \mathbb{R}^m \times \mathbb{R} \mid x \geq 0, B^T x \leq 1\right\}$$
$$Q = \{(y, u) \in \mathbb{R}^n \times \mathbb{R} \mid y \geq 0, Ay \leq 1\}$$

Step 2: The vertices of these polytopes are the points that will have labels (they are the points that lie at the intersection of the underlying halfspaces [Ziegler2012]).

To find these vertices, nashpy uses scipy which has a handy class for creating Polytopes using the inequality definitions and being able to return the vertices. Here is the wrapper written in nashpy that is used by the vertex enumeration algorithm to give the vertices and corresponding labels:

To find these vertices, nashpy uses scipy which has a handy class for creating Polytopes using the inequality definitions and being able to return the vertices. Here is the wrapper written in nashpy that is used by the vertex enumeration algorithm to give the vertices and corresponding labels:

```
>>> import nashpy as nash
>>> import numpy as np
>>> A = np.array([[3, 1], [1, 3]])
>>> halfspaces = nash.polytope.build_halfspaces(A)
>>> vertices = nash.polytope.non_trivial_vertices(halfspaces)
>>> for vertex in vertices:
...     print(vertex)
(array([0.333..., 0...]), {0, 3})
(array([0..., 0.333...]), {1, 2})
(array([0.25, 0.25]), {0, 1})
```

Step 3, we iterate over all pairs of the vertices of both polytopes and pick out the ones that are fully labeled. Because of the scaling that took place to create the Polytope from the Polyhedron, we will need to return a normalisation of both vertices.

python code:

```
A = np.array([[0, -1, 1], [1, 0, -1], [-1, 1, 0]])
#since our game is symtric:
rps = nash.Game(A)
print(list(rps.vertex_enumeration()))
```

the output: [(array([0.33333333, 0.33333333, 0.33333333]), array([0.33333333, 0.33333333, 0.33333333]))]
which is the same what we calculated before for mixed strategy

.3 The Lemke Howson Algorithm

The Lemke Howson algorithm implemented in Nashpy is based on the one described in [Nisan2007] originally introduced in [Lemke1964].

The algorithm is as follows:
For a nondegenerate 2 player game $(A, B) \in \mathbb{R}^{m \times n^2}$ the following algorithm returns **a single** Nash equilibria:

1. Obtain the best response Polytopes P and Q
2. Choose a starting label to drop, this will correspond to a vertex of P or Q
3. in that polytope, remove the label from the corresponding vertex and move to the vertex that shared that label. A new label will be picked up and duplicated in the other polytope.
4. in the other polytope drop the duplicate label and move to the vertex that shared that label.

Repeat steps 3 and 4 until there are no duplicate labels.

```python
A = np.array([[0, -1, 1], [1, 0, -1], [-1, 1, 0]])
#since our game is symtric:
rps = nash.Game(A)
print(list(rps.lemke_howson(initial_dropped_label=0)))
```

the output : [array([0.33333333, 0.33333333, 0.33333333]), array([0.33333333, 0.33333333, 0.33333333])]
which is the same what we calculated before for mixed strategy
The *initial_dropped_label* is an integer between 0 and $sum(A.shape) - 1$

.4 Comparison:

Support Enumeration

The `game.support_enumeration()` method provided by Nashpy is used to enumerate all the Nash equilibria in a game. It exhaustively searches through all possible combinations of strategies to find Nash equilibria. This method guarantees finding all pure strategy Nash equilibria and potentially mixed strategy Nash equilibria if they exist. However, it can be computationally expensive for large games due to the exponential number of strategy profiles to be evaluated.

Vertex Enumeration

The `game.vertex_enumeration()` method provided by Nashpy is used to enumerate all pure strategy Nash equilibria in a game. It specifically looks for pure strategy Nash equilibria by traversing through the vertices of the strategy polytope. Vertex enumeration provides a finite and non-exhaustive enumeration of pure strategy Nash equilibria. However, for games with multiple Nash equilibria, it may not find all of them. Vertex enumeration is generally faster than support enumeration and is well-suited for games with a small number of strategies.

Lemke-Howson Algorithm

The Lemke-Howson algorithm is an iterative algorithm used to solve the linear complementarity problem (LCP) associated with a game. It efficiently finds and traverses through the vertices of the best response polytopes to find Nash equilibria. The algorithm is particularly effective for two-player games and has a polynomial-time complexity for certain classes of games. However, it may struggle with games that have degenerate or complex polytopes. The Lemke-Howson algorithm guarantees finding at least one Nash equilibrium, if it exists, but it won't necessarily find all Nash equilibria.

Summary

- **Completeness:** Support enumeration guarantees finding all Nash equilibria (both pure and mixed). Vertex enumeration finds all pure strategy Nash equilibria but not necessarily mixed strategy equilibria. The Lemke-Howson algorithm finds at least one Nash equilibrium if it exists but won't necessarily find all Nash equilibria.

- **Computational Complexity:** Support enumeration can be computationally expensive, especially for large games. Vertex enumeration is generally faster since it focuses on pure strategies. The complexity of the Lemke-Howson algorithm can vary depending on the properties of the game, but it is efficient for certain classes of games.

- **Applicability:** Support enumeration and vertex enumeration are useful for exploring the complete equilibria space, while the Lemke-Howson algorithm is well-suited for finding specific equilibria or analyzing particular game properties.

In summary, support enumeration and vertex enumeration provide exhaustive or partial enumeration of Nash equilibria, respectively, but can be computationally demanding. The Lemke-Howson algorithm provides an efficient method to find specific equilibria. The choice of method depends on the specific requirements of the game and the desired level of completeness.

Matching Pennies Game

The Matching Pennies game is a two-player zero-sum game in game theory. In this game, each player has a penny and simultaneously chooses between two possible actions: ”Heads” (H) or ”Tails” (T). The players are trying to predict and match each other’s choices.

.1 Payoff Matrix

The payoff matrix for the Matching Pennies game is as follows:

	H	T
H	(1, -1)	(-1, 1)
T	(-1, 1)	(1, -1)

In the payoff matrix, the first value in each pair represents the payoff to the row player, while the second value represents the payoff to the column player.

.2 Game Strategies

Each player has two available strategies: choosing ”Heads” or ”Tails”. The players can either adopt a pure strategy by consistently choosing a specific action or use mixed strategies by randomizing their choices according to a probability distribution.

.3 Nash Equilibrium

In the Matching Pennies game, there is no pure strategy Nash equilibrium since no single strategy is optimal for both players. However, a mixed strategy Nash equilibrium can be reached. If both players choose their actions randomly with an equal probability of 0.5 for each action, the game is in equilibrium as no player can unilaterally improve their expected payoff by changing their strategy.

.4 Interpretation

The Matching Pennies game can represent situations of competition or conflict where the players want to outsmart or deceive each other. It is often used to illustrate concepts such as mixed strategies, correlated equilibria, and the idea that rational players may choose actions randomly to maximize their expected payoff.

python code

```python
import nashpy.repeated_games
A = np.array([[1, -1], [-1, 1]])
matching_pennies = nash.Game(A)
repeated = nashpy.repeated_games.obtain_repeated_game(game=
    matching_pennies, repetitions=2)
print(repeated)
repeated.payoff_matrices[0].shape
```

output:
Zero sum game with payoff matrices:

Row player: [[2. 2. 2. ... 0. -2. -2.] [2. 2. 2. ... 0. -2. -2.] [2. 2. 2. ... 0. -2. -2.] ... [0. 0. 0. ... 2. 0. 2.] [-2. -2. -2. ... 0. 2. 0.] [-2. -2. -2. ... 2. 0. 2.]]

Column player: [[-2. -2. -2. ... 0. 2. 2.] [-2. -2. -2. ... 0. 2. 2.] [-2. -2. -2. ... 0. 2. 2.] ... [0. 0. 0. ... -2. 0. -2.] [2. 2. 2. ... 0. -2. 0.] [2. 2. 2. ... -2. 0. -2.]]
(32,32)

Fictitious Play Algorithm

The Fictitious Play algorithm is an iterative method used to approximate Nash equilibria in repeated games. It is a best response dynamics algorithm in which each player assumes that the opponents' strategies are fixed and plays to maximize their expected payoff against these strategies.

.1 Algorithm Steps

The Fictitious Play algorithm proceeds as follows:

For a game $(A, B) \in \mathbb{R}^{m\times n}$ define $\kappa_t^i : S^{-1} \to \mathbb{N}$ to be a function that in a given time interval t for a player i maps a strategy s from the opponent's strategy space S^{-1} to a number of total times the opponent has played s. Thus:

$$\kappa_t^i\left(s^{-i}\right) = \kappa_{t-1}\left(s^{-i}\right) + \begin{cases} 1, & \text{if } s_{t-1}^{-i} = s^{-i} \\ 0, & \text{otherwise} \end{cases}$$

In practice:

$$\kappa_t^1 \in \mathbb{Z}^n \quad \kappa_t^2 \in \mathbb{Z}^m$$

At stage t, each player assumes their opponent is playing a mixed strategy based on κ_{t-1} :

$$\frac{\kappa_{t-1}}{\sum \kappa_{t-1}}$$

They calculate the expected value of each strategy, which is equivalent to:

$$s_t^1 \in \text{argmax}_{s \in S_1} A\kappa_{t-1}^2 \quad s_t^2 \in \text{argmax}_{s \in S_2} B^T \kappa_{t-1}^1$$

In the case of multiple best responses, a random choice is made.

The Fictitious Play algorithm iteratively refines the players' strategies by repeatedly calculating best responses and updating strategies based on observed opponent behavior. The learning rule used for strategy updates can vary, and different rules may lead to different convergence properties.

.2 Convergence

The convergence of the Fictitious Play algorithm depends on the game structure and the learning rule employed. In some cases, the algorithm converges to a Nash equilibrium, while in others, it may converge to a correlated equilibrium or an approximate equilibrium.

The algorithm's convergence guarantees are generally less robust in games with complex strategies or incomplete information. However, Fictitious Play has been shown to converge under certain conditions, such as in zero-sum games and certain types of symmetric games.

.3 Applications

The Fictitious Play algorithm has various applications in game theory, including repeated games, game learning, and evolutionary game theory. It provides a simple and intuitive framework to explore the dynamics of strategic interactions and approximate equilibrium solutions.

python Code

```
A = np.array([[0, 1, 0], [0, 0, 1], [1, 0, 0]])
B = np.array([[0, 0, 1], [1, 0, 0], [0, 1, 0]])
game = nash.Game(A, B)
iterations = 10000
np.random.seed(0)
play_counts = tuple(game.fictitious_play(iterations=iterations))
play_counts[-1]
```

output:

```
[array([5464., 1436., 3100.]), array([2111., 4550., 3339.])]
```

completed code:

```python
import matplotlib.pyplot as plt
plt.figure()
probabilities = [row_play_counts / np.sum(row_play_counts) for
    row_play_counts, col_play_counts in play_counts]
for number, strategy in enumerate(zip(*probabilities)):
plt.plot(strategy, label=f"$s_{number}$")
plt.xlabel("Iteration")
plt.ylabel("Probability")
plt.title("Actions taken by row player")
plt.legend()
```

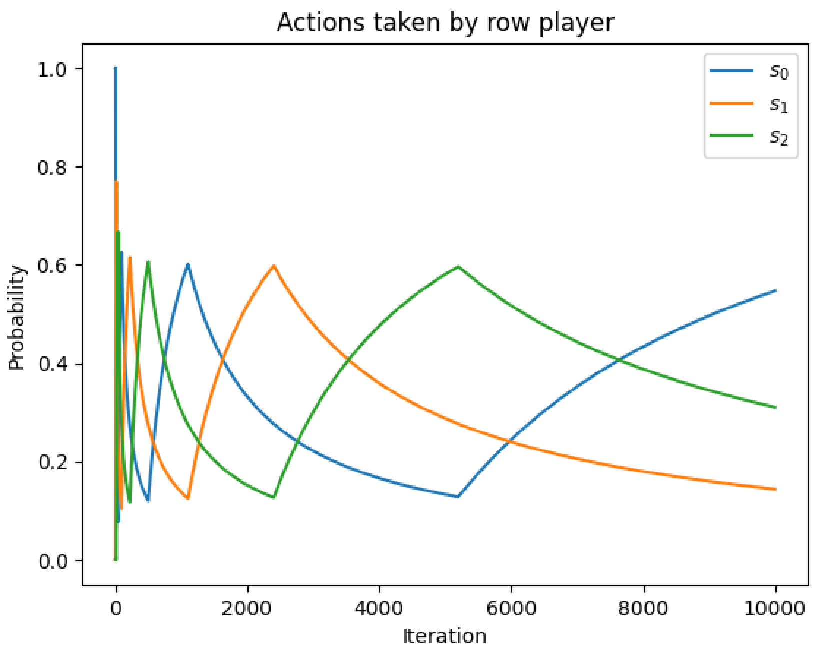


Figure 12. output not converging

another example

```python
A = np.array([[1 / 2, 1, 0], [0, 1 / 2, 1], [1, 0, 1 / 2]])
B = np.array([[1 / 2, 0, 1], [1, 1 / 2, 0], [0, 1, 1 / 2]])
game = nash.Game(A, B)
np.random.seed(0)
play_counts = tuple(game.fictitious_play(iterations=iterations))
play_counts[-1]
```

output=[array([3290., 3320., 3390.]), array([3356., 3361., 3283.])]

```python
import matplotlib.pyplot as plt
plt.figure()
probabilities = [row_play_counts / np.sum(row_play_counts) for
    row_play_counts, col_play_counts in play_counts]
for number, strategy in enumerate(zip(*probabilities)):
    plt.plot(strategy, label=f"$s_{number}$")
plt.xlabel("Iteration")
plt.ylabel("Probability")
plt.title("Actions taken by row player")
plt.legend()
```

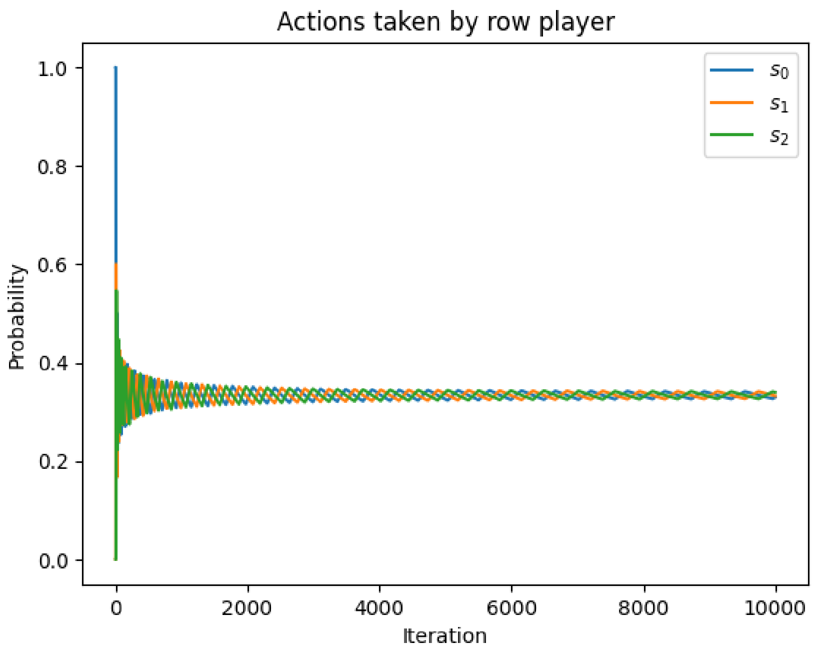


Figure 13. output converging

Replicator Dynamics Algorithm

The Replicator Dynamics algorithm is a widely used method in game theory and evolutionary biology to study the evolution of populations playing strategic games. It models the dynamics of strategy proportions within a population over time.

.1 Algorithm Steps

The Replicator Dynamics algorithm proceeds as follows:

1. **Initialization:** Begin with an initial population distribution where individuals are assigned random strategy proportions.

2. **Payoff Calculation:** Calculate the payoff for each strategy in the current population by playing the strategic game using the current strategy proportions.

3. **Average Payoff Calculation:** Calculate the average payoff of the population by taking the weighted average of the payoffs, with the weights being the strategy proportions.

4. **Update Strategy Proportions:** For each strategy, update the proportion in the population according to the difference between the strategy's payoff and the average population payoff. Higher-payoff strategies will increase in proportion, while lower-payoff strategies will decrease.

5. **Normalization:** Normalize the updated strategy proportions so that they sum up to 1, ensuring that the proportions represent a valid distribution.

6. **Iteration:** Repeat steps 2-5 until convergence is reached or for a predetermined number of iterations.

The Replicator Dynamics algorithm captures the idea that strategies with higher payoffs have a higher chance of being adopted and replicated in subsequent generations. By updating the strategy proportions based on the difference between individual payoffs and the average population payoff, the algorithm simulates an evolutionary process that can lead to the emergence of stable strategy distributions.

.2 Convergence

The convergence properties of the Replicator Dynamics algorithm depend on the specific game being studied, the initial conditions, and the dynamics of strategy updates. In some cases, the algorithm converges to a stable state corresponding to a Nash equilibrium of the game. In other cases, cyclic behaviors or stable mixed strategies may emerge.

The algorithm's convergence behavior provides insights into the long-term behavior and evolutionary stability of different strategies within a population. By studying the dynamics of strategic interactions and the evolution of behaviors in games, researchers gain a deeper understanding of the dynamics of complex systems.

python code

```
import nashpy as nash
import numpy as np
A = np.array([[3, 2], [4, 2]])
B = np.array([[1, 3], [2, 4]])
game = nash.Game(A,B)
plt.figure()
probabilities = [row_play_counts / np.sum(row_play_counts) for
    row_play_counts, col_play_counts in play_counts]
for number, strategy in enumerate(zip(*probabilities)):
    plt.plot(strategy, label=f"$s_{number}$")
plt.xlabel("Iteration")
plt.ylabel("Probability")
plt.title("Actions taken by row player")
plt.legend()
```

answer:

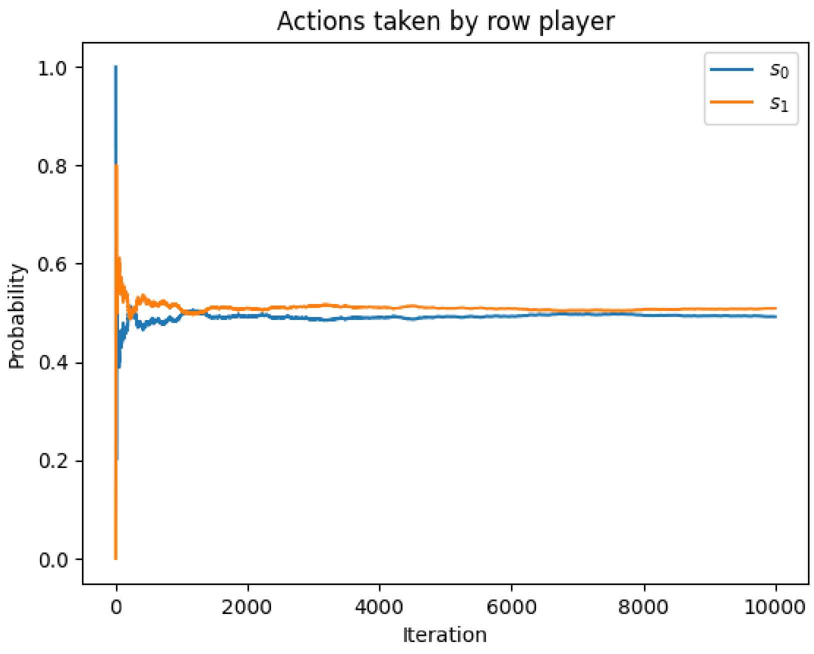


Figure 14. output converging

```
game.replicator_dynamics()
```

output:
array([[0.5 , 0.5], [0.49875032, 0.50124968], [0.49750377, 0.50249623], ..., [0.10199196, 0.89800804], [0.10189853, 0.89810147], [0.10180527, 0.89819473]])
It is also possible to pass a y0 variable in order to assign a starting strategy. Otherwise the probability is divided equally amongst all possible actions. Passing a timepoints variable gives the algorithm a sequence of timepoints over which to calculate the strategies:

```
y0 = np.array([0.9, 0.1])
timepoints = np.linspace(0, 10, 1000)
game.replicator_dynamics(y0=y0, timepoints=timepoints)
```

output:
array([[0.9 , 0.1], [0.89918663, 0.10081337], [0.89836814, 0.10163186], ..., [0.14109126, 0.85890874], [0.1409203 , 0.8590797], [0.14074972, 0.85925028]])

7 Data analysis

Practical Question 5 , 6

library needed for this part:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy
import scipy.stats as stats
```

```
df = pd.read_csv("Data.csv", header= None)
plt.figure(figsize = (20, 20))
beta0 = []
beta1 = []
nrvm = []
nrvs = []
for i in range(6):
    print("company number "+str(i+1)+" Data")
    x=np.arange(1 , 31 , 1)
    y= np.array(df.iloc[:,i])
    expected = np.array([x, y])
    coef=np.polyfit(x,y,1)
    ye=np.poly1d(coef)
    coef = np.array(coef)
    beta0.append(coef[1])
    beta1.append(coef[0])

    plt.subplot(6,3 ,i*3+1)
    plt.plot(x,y,".r",x,ye(x),"-y")
    plt.title("company number " + str(i+1))
    plt.savefig("5_"+str(i+1) + ".png")
    print("company number "+ str(i+1)+" beta0 = "+str(round(beta0[i],2)))
    print("company number "+ str(i+1)+" beta1 = "+str(round(beta1[i],2)))
    rv=[]
    for j in range(len(y)):
        rv.append(ye(x[j])-y[j])
#     plt.show()
    plt.subplot(6,3 ,i*3+2)

    plt.hist(rv,bins=20)
    plt.title("histogram of difference yhat and y")
    plt.savefig("6_histogram of difference yhat and y number "+str(i+1) + ".png")
#     plt.show()
    nrv=np.array(rv)
    nrvm.append(nrv.mean())
    nrvs.append(nrv.std())

    print("company number "+ str(i+1)+" sigma = "+str(round(nrvs[i],2)))
    normaldist=np.random.normal(nrvm[i],nrvs[i],len(nrv))

    plt.subplot(6,3 ,i*3+3)
    plt.hist(normaldist,bins=20)
    plt.title("histogram of normal")
    if i==5:
        plt.savefig("6_histogram of normal number "+str(i+1) + ".png")
#     plt.show()
    print("p-value= " +str(round((stats.ttest_ind(normaldist ,nrv).pvalue),4)))
    if(stats.ttest_ind(normaldist , nrv).pvalue < 0.05) :
      print("Event H1 :>,it isn't like normal distribution!")
    else :
      print ("Event H0 :>,it's like normal distribution!")
    print("--------------------------------------------------")
    plt.tight_layout()
```

output:

```
company number 1 Data
company number 1 beta0 = 2.92
company number 1 beta1 = 0.1
company number 1 sigma = 0.09
p-value= 0.523
Event H0 :,Yes ,it's like normal distribution!
————————————————————————
company number 2 Data
company number 2 beta0 = 2.53
company number 2 beta1 = 0.32
   company number 2 sigma = 0.32
   p-value= 0.792
Event H0 :,Yes ,it's like normal distribution!
————————————————————————
company number 3 Data
company number 3 beta0 = 2.56
company number 3 beta1 = 0.51
company number 3 sigma = 0.55
p-value= 0.703
Event H0 :,Yes ,it's like normal distribution!
————————————————————————
company number 4 Data
company number 4 beta0 = 2.06
company number 4 beta1 = 0.72
company number 4 sigma = 0.82
p-value= 0.2448
Event H0 :,Yes ,it's like normal distribution!
————————————————————————
company number 5 Data
company number 5 beta0 = 1.91
company number 5 beta1 = 0.94
company number 5 sigma = 0.81
p-value= 0.3607 Event H0 :,Yes ,it's like normal distribution!
————————————————————————
company number 6 Data
company number 6 beta0 = 2.29
company number 6 beta1 = 1.13
```

company number 6 sigma = 1.03
p-value= 0.5009
Event H0 :,Yes ,it's like normal distribution!

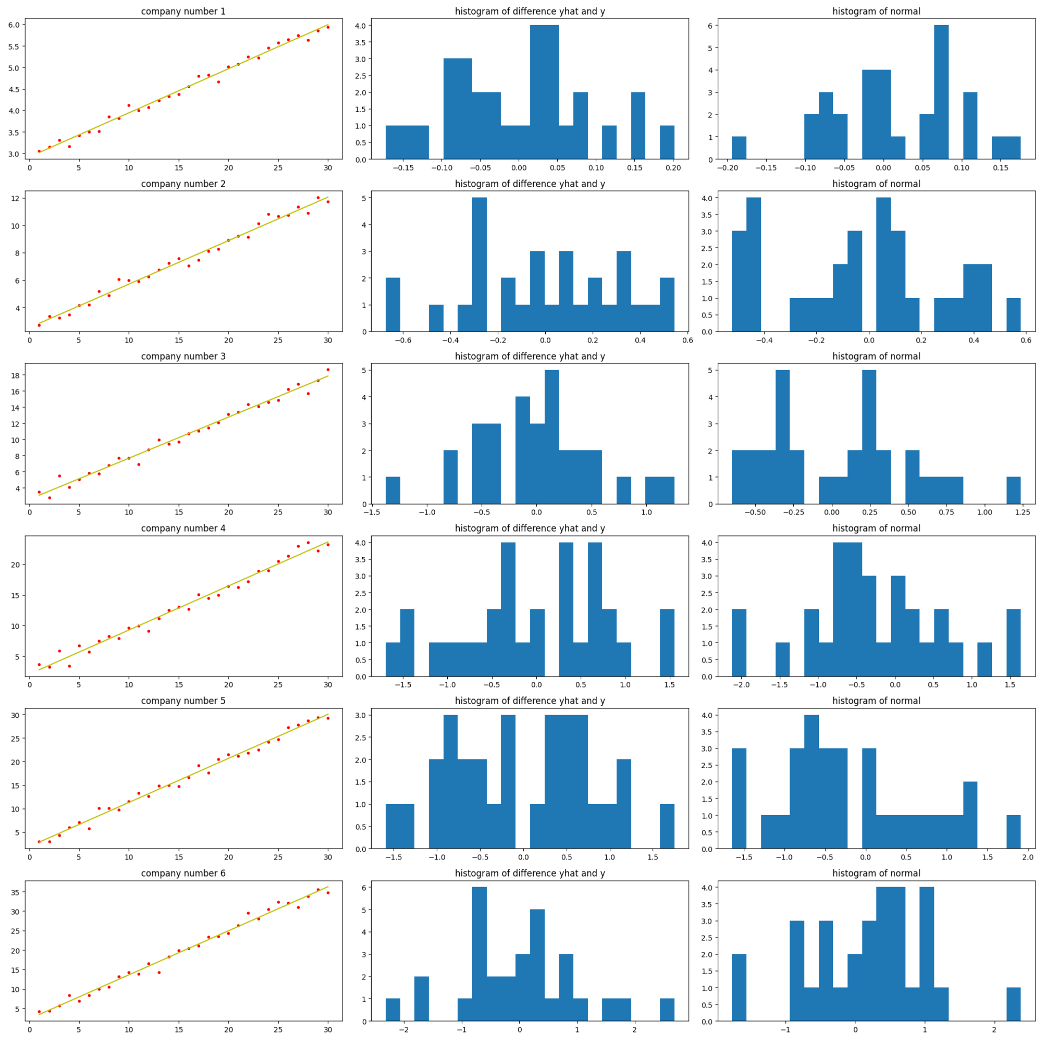


Figure 15. output plot

```
rv1=np.random.normal(0,2,100)
rv2=np.random.normal(0,2,100)
stats.ttest_ind(rv1, rv2)
```

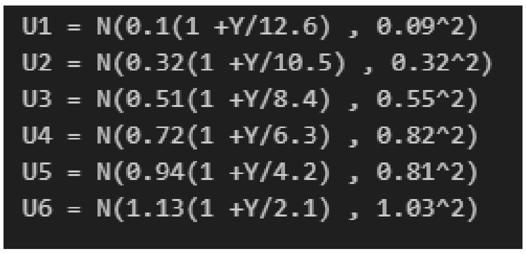
output:
$Ttest\_indResult(statistic = -1.0301649226966483, pvalue = 0.3041893323391019)$

Theoretical Question 23

here we use the data that found in the previous section and i have written a code to print the desired distribution

```
for i in range(6):
    mi = beta1[i]
    s = nrvs[i]
    Hi = 2.1*(7-(i+1))
    print("U"+ str(i+1)+" = N("+str(round(mi,2))+"(1 +Y/"+str(
  round(Hi,2))+") , "+str(round(s,2))+"^2)")
```

output:



```
U1 = N(0.1(1 +Y/12.6) , 0.09^2)
U2 = N(0.32(1 +Y/10.5) , 0.32^2)
U3 = N(0.51(1 +Y/8.4) , 0.55^2)
U4 = N(0.72(1 +Y/6.3) , 0.82^2)
U5 = N(0.94(1 +Y/4.2) , 0.81^2)
U6 = N(1.13(1 +Y/2.1) , 1.03^2)
```

Figure 16. Ui distributions

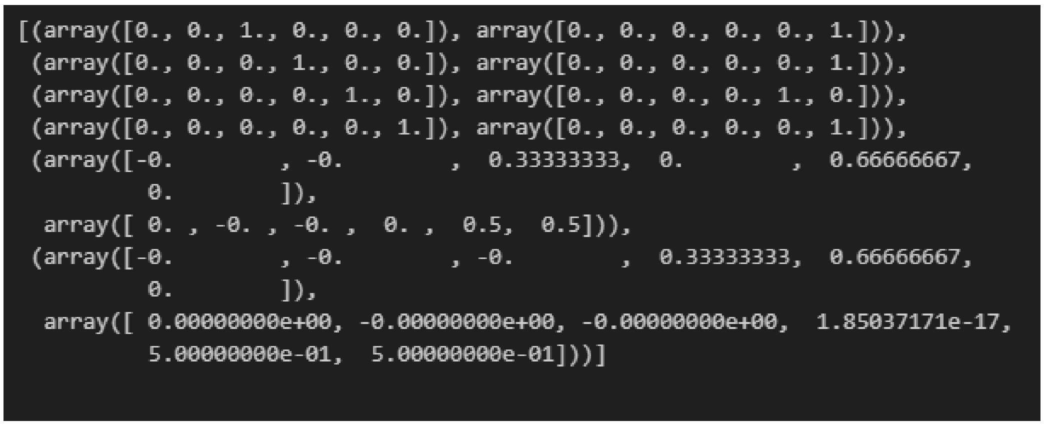
Theoretical Question 24

I used python code to calculate the game matrix
for that first we check whether the two bank are investing in the same company or not because if they are ,Y in that case is the summation of the total investment other wise each bank gets its own investment profit independently

```
Utility1 = np.zeros((6,6))
Utility2 = np.zeros((6,6))
for i in range(1,7):
    for j in range(1,7):
        ai = 7 - i
        bj = np.sqrt(49-j*j)
        Hi = 2.1*(7-(i))
        mi = beta1[i-1]
        Hj = 2.1*(7-(j))
        mj = beta1[j-1]
        if i==j:
            Yi = ai+bj
            Yj = Yi
        else:
            Yi = ai
            Yj = bj
        E1 = ai*mi*(1+Yi/Hi)
        E2 = bj*mj*(1+Yj/Hj)
        Utility1[i-1,j-1] = round(E1,3)
        Utility2[i-1,j-1] = round(E2,3)
print('compony 1 utility'+'\n')
print(Utility1)
print("compony 2 utility"+'\n')
print(Utility2)
print("compony 1 rounded"+"\n")
print(Utility1.astype(int))
print("compony 2 rounded")
print(Utility2.astype(int))


out:

compony 1 utility
[[1.245 0.907 0.907 0.907 0.907 0.907]
 [2.34  3.352 2.34  2.34  2.34  2.34 ]
 [3.009 3.009 4.543 3.009 3.009 3.009]
 [3.186 3.186 3.186 5.154 3.186 3.186]
 [2.768 2.768 2.768 2.768 4.955 2.768]
 [1.671 1.671 1.671 1.671 1.671 3.614]]

compony 2 utility
[[ 1.438  3.485  5.649  7.9    9.95  11.087]
 [ 1.1    4.497  5.649  7.9    9.95  11.087]
 [ 1.1    3.485  7.184  7.9    9.95  11.087]
 [ 1.1    3.485  5.649  9.868  9.95  11.087]
 [ 1.1    3.485  5.649  7.9   12.137 11.087]
 [ 1.1    3.485  5.649  7.9    9.95  13.031]]

compony 1 rounded
[[1 0 0 0 0 0]
 [2 3 2 2 2 2]
 [3 3 4 3 3 3]
 [3 3 3 5 3 3]
 [2 2 2 2 4 2]
 [1 1 1 1 1 3]]

compony 2 rounded
[[ 1  3  5  7  9 11]
 [ 1  4  5  7  9 11]
 [ 1  3  7  7  9 11]
 [ 1  3  5  9  9 11]
 [ 1  3  5  7 12 11]
 [ 1  3  5  7  9 13]]
```

Practical Question 7

```
rps = nash.Game(Utility1 , Utility2)
eqs = rps.support_enumeration()
list(eqs)
```



```
[(array([0., 0., 1., 0., 0., 0.]), array([0., 0., 0., 0., 0., 1.])),
 (array([0., 0., 0., 1., 0., 0.]), array([0., 0., 0., 0., 0., 1.])),
 (array([0., 0., 0., 0., 1., 0.]), array([0., 0., 0., 0., 1., 0.])),
 (array([0., 0., 0., 0., 0., 1.]), array([0., 0., 0., 0., 0., 1.])),
 (array([-0.        , -0.        ,  0.33333333,  0.        ,  0.66666667,
          0.        ]),
  array([ 0. , -0. , -0. ,  0. ,  0.5,  0.5])),
 (array([-0.        , -0.        , -0.        ,  0.33333333,  0.66666667,
          0.        ]),
  array([ 0.00000000e+00, -0.00000000e+00, -0.00000000e+00,  1.85037171e-17,
          5.00000000e-01,  5.00000000e-01]))]
```

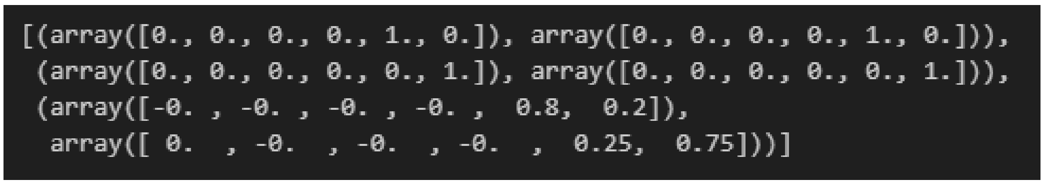
Figure 17. nash equilibrium

Practical Question 8

here the only difference from the previous part is that the utility is random $E1 = ai * np.random.normal(mi * (1 + Yi/Hi), nrvs[i-1])$ we do every step we have done before

```
Utility1 = np.zeros((6,6))
Utility2 = np.zeros((6,6))
for i in range(1,7):
    for j in range(1,7):
        ai = 7 - i
        bj = np.sqrt(49-j*j)
        Hi = 2.1*(7-(i))
        mi = beta1[i-1]
        Hj = 2.1*(7-(j))
        mj = beta1[j-1]
        if i==j:
            Yi = ai+bj
            Yj = Yi
        else:
            Yi = ai
            Yj = bj
        np.random.seed(400101034)
        E1 = ai*np.random.normal(mi*(1+Yi/Hi),nrvs[i-1])
        E2 = bj*np.random.normal(mj*(1+Yj/Hj),nrvs[j-1])
        Utility1[i-1,j-1] = round(E1,3)
        Utility2[i-1,j-1] = round(E2,3)

print('compony 1 utility'+'\n')
print(Utility1)
print("compony 2 utility"+'\n')
print(Utility2)
print("compony 1 rounded"+"\n")
print(Utility1.astype(int))
print("compony 2 rounded")
print(Utility2.astype(int))

compony 1 utility
[[0.843 0.505 0.505 0.505 0.505 0.505]
 [1.134 2.147 1.134 1.134 1.134 1.134]
 [1.349 1.349 2.883 1.349 1.349 1.349]
 [1.335 1.335 1.335 3.303 1.335 1.335]
 [1.551 1.551 1.551 1.551 3.738 1.551]
 [0.895 0.895 0.895 0.895 0.895 2.839]]

compony 2 utility
[[ 1.268  2.893  4.689  6.604  8.859 10.065]
 [ 0.93   3.906  4.689  6.604  8.859 10.065]
 [ 0.93   2.893  6.224  6.604  8.859 10.065]
 [ 0.93   2.893  4.689  8.572  8.859 10.065]
 [ 0.93   2.893  4.689  6.604 11.046 10.065]
 [ 0.93   2.893  4.689  6.604  8.859 12.008]]

compony 1 rounded
[[0 0 0 0 0 0]
 [1 2 1 1 1 1]
 [1 1 2 1 1 1]
 [1 1 1 3 1 1]
 [1 1 1 1 3 1]
 [0 0 0 0 0 2]]

compony 2 rounded
[[ 1  2  4  6  8 10]
 [ 0  3  4  6  8 10]
 [ 0  2  6  6  8 10]
 [ 0  2  4  8  8 10]
 [ 0  2  4  6 11 10]
 [ 0  2  4  6  8 12]]
```

```python
rps = nash.Game(Utility1.astype(int),Utility2.astype(int))
eqs = rps.support_enumeration()
list(eqs)
```



```
[(array([0., 0., 0., 0., 1., 0.]), array([0., 0., 0., 0., 1., 0.])),
 (array([0., 0., 0., 0., 0., 1.]), array([0., 0., 0., 0., 0., 1.])),
 (array([-0. , -0. , -0. , -0. ,  0.8,  0.2]),
  array([ 0.  , -0.  , -0.  , -0.  ,  0.25,  0.75]))]
```

Figure 18. nash equilibrium