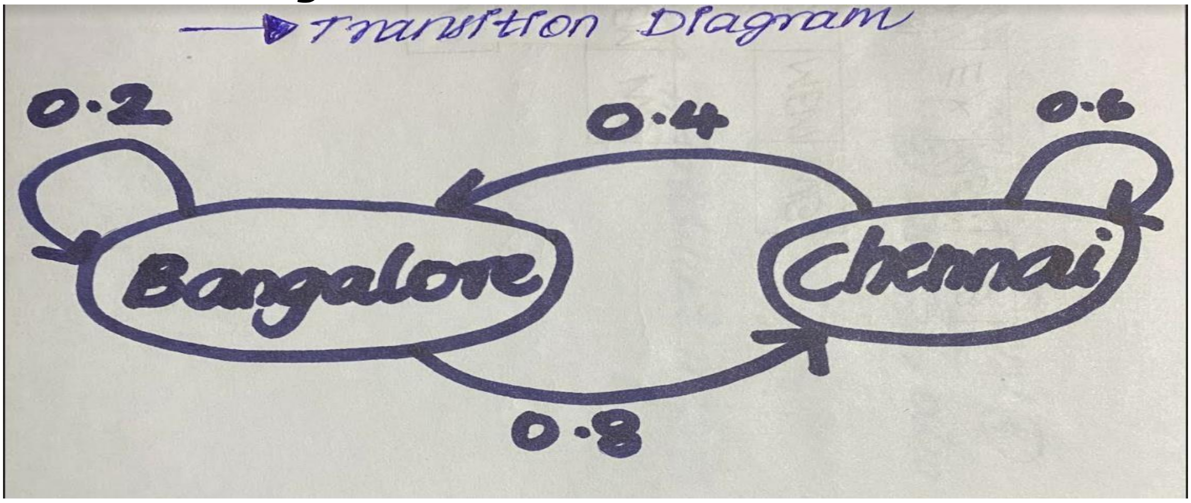
LINEAR ALGEBRA ASSIGNMENT

Unit 2

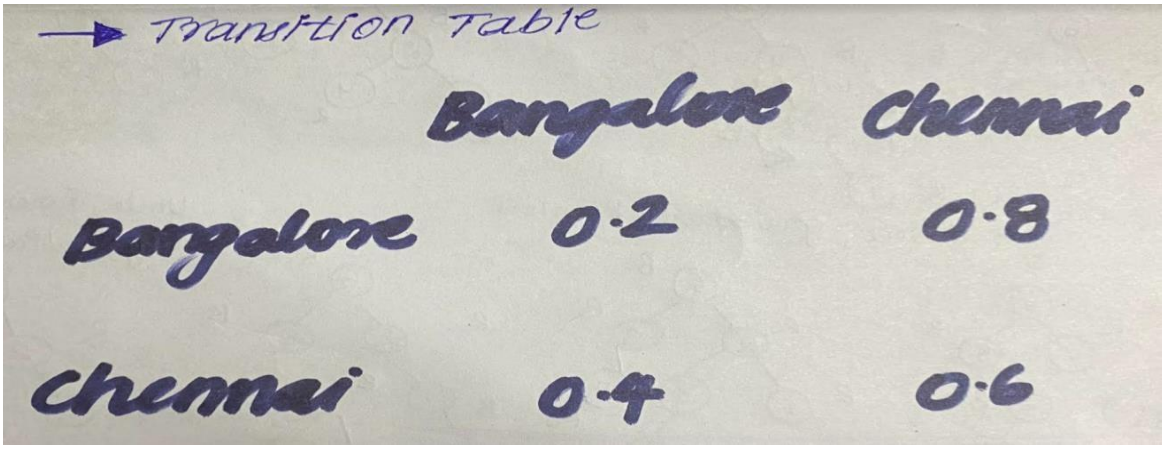
Implementation of Markov chains for:

1. Population migration distribution between two Indian states.

Transition Diagram



Transition Table


Transition Table

	Bangalore	Chennai
Bangalore	0.2	0.8
Chennai	0.4	0.6

Code

Python code

```
#Population migration distribution between two Indian states.

#2 indian states - Bangalore ,Chennai
#Can migrate to any one of the two Indian states
#To pedict the migration distribution between 2 indian states
```

```
import numpy as np
import random as rm

#encoding indian_state to numbers
indian_state = {
    0:"Bangalore",
    1:"Chennai",
}
indian_state

#Transition Matrix
T= np.array([[0.2,0.8],[0.4,0.6]])

#Random Walk on Markov Chains

#when start_indian_state = 0  ,present location is Bangalore
#when start_indian_state = 1  ,present location is Chennai
for  start_indian_state in range(2):
    print("\nMigration Prediction of population staying
in",indian_state[start_indian_state],"for next 5 times")
    n=6                  #for next 5 predictions
    print(indian_state[start_indian_state],"--->",end="
")
    prev_indian_state = start_indian_state

    while n-1:                         #continue the loop for n-1 times, as we are
starting fron current  indian_state we are staying
        curr_indian_state =
np.random.choice([0,1],p=T[prev_indian_state])        #which indian_state
population might migrate next by folling the transition probability
(probabilty of going to indian_states bangalore or chennai from previous
indian_state)
        print(indian_state[curr_indian_state],"--->",end=" ")
        prev_indian_state=curr_indian_state
        n-=1
    print("stop")


#A stationary distribution of a Markov chain is a probability distribution
that remains unchanged in the Markov chain as time progresses.
steps = 10**5         #accuracy increses with number of steps,hence higher the
number higher the accuracy
start_indian_state=0
pi=np.array([0,0])
pi[start_indian_state] = 1
prev_indian_state = start_indian_state

i=0
while i<steps:
    curr_indian_state=np.random.choice([0,1],p=T[prev_indian_state])
    pi[curr_indian_state]+=1
    prev_indian_state=curr_indian_state
    i+=1
```

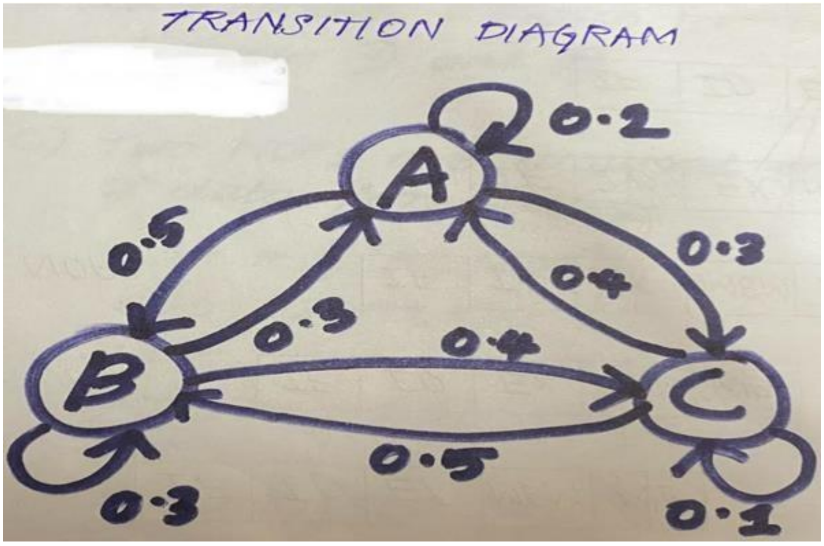
```
ans=pi/steps
print("\nOverall Probabity of population migrating to
Bangalore",ans[0],"\nOverall Probabity of population migrating to Chennai
=",ans[1])


                    **********      Thank you      **********
```

output

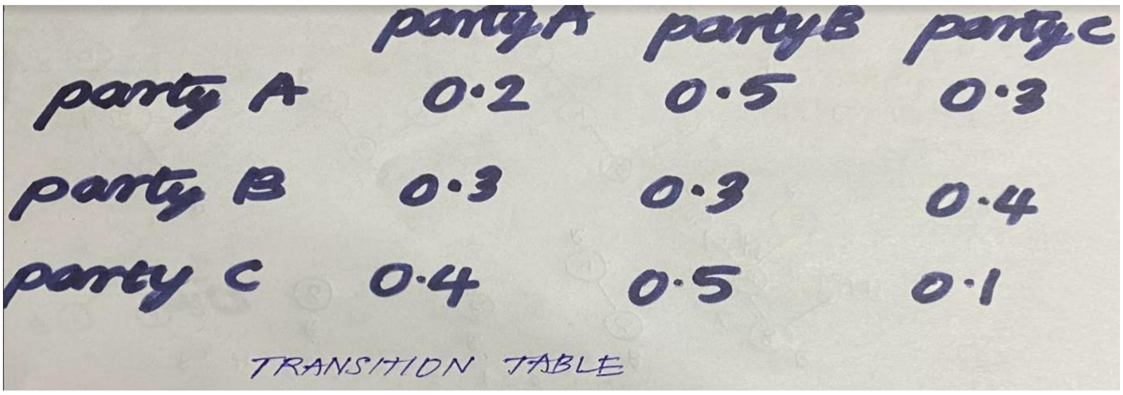
```
[Running] python -u "d:\PES\sem4\LA\marcov_2a.py"

Migration Prediction of population staying in Bangalore for next 5 times
Bangalore ---> Chennai ---> Bangalore ---> Bangalore ---> Chennai ---> Bangalore ---> stop

Migration Prediction of population staying in Chennai for next 5 times
Chennai ---> Bangalore ---> Chennai ---> Bangalore ---> Chennai ---> Chennai ---> stop

Overall Probabity of population migrating to Bangalore 0.33397
Overall Probabity of population migrating to Chennai = 0.66604

[Done] exited with code=0 in 1.706 seconds
```

2. Vote changing pattern of Three Political parties from one election to the next.

Transition Diagram



Transition Table



	party A	party B	party C
party A	0·2	0·5	0·3
party B	0·3	0·3	0·4
party C	0·4	0·5	0·1

TRANSITION TABLE

Code	Python code

Python code

```
#Vote changing pattern of Three Political parties from one election to  the
 next.

#3 political parties - partyA , partyB , partyC
#Any one of the political parties can win the upcomming elections.
#Predicting which will be the next rulling party with each initial state being
PartyA ,PartyB and PartyC
import numpy as np
import random as rm

#encoding state to numbers
state = {
    0:"partyA",
    1:"partyB",
    2:"partyC"
}
state

#Transition Matrix
T= np.array([[0.2,0.5,0.3],[0.3,0.3,0.4],[0.4,0.5,0.1]])

#Random  Walk  on  Markov Chains

#when start_state = 0 ,current ruling party is partyA
#when start_state = 1 ,current ruling party is partyB
#when start_state = 2 ,current ruling party is partyC
for start_state in range(3):
    print("\nPrediction of rulling parties for next 5 elections when",
state[start_state], "is the current rulling party:")
    n=6             #for next 5 predictions
    print(state[start_state],"--->",end=" ")
    prev_state = start_state       #bcoz we have just visited start state

    while n-1:                         #continue the loop for n-1 times, as the
first Election is already over
        curr_state = np.random.choice([0,1,2],p=T[prev_state])       #which
state it  is going next by folling the transition probability (probabilty of
going to states 1,2,3 from previous  state)

        print(state[curr_state],"--->",end=" ")
        prev_state=curr_state
        n-=1
    print("stop")

#A stationary distribution of a Markov chain is a probability distribution
that remains unchanged in the Markov chain as time progresses.
steps = 10**5         #accuracy increses with number of steps,hence higher the
number higher the accuracy
```

```python
start_state=0
pi=np.array([0,0,0])
pi[start_state] = 1
prev_state =  start_state

i=0
while i<steps:
    curr_state=np.random.choice([0,1,2],p=T[prev_state])
    pi[curr_state]+=1
    prev_state=curr_state
    i+=1

ans=pi/steps
print("\nOverall Probabity of partyA being the rulling party
=",ans[0],"\nOverall Probabity of partyB being the rulling party
=",ans[1],"\nOverall Probabity of partyC being the rulling party =",ans[2])


#                  *******     Thank you     ********
```

output

```
[Running] python -u "d:\PES\sem4\LA\markov_2b.py"

Prediction of rulling parties for next 5 elections when partyA is the current rulling party:
partyA ---> partyB ---> partyC ---> partyA ---> partyB ---> partyA ---> stop

Prediction of rulling parties for next 5 elections when partyB is the current rulling party:
partyB ---> partyC ---> partyB ---> partyB ---> partyA ---> partyB ---> stop

Prediction of rulling parties for next 5 elections when partyC is the current rulling party:
partyC ---> partyC ---> partyB ---> partyA ---> partyA ---> partyC ---> stop

Overall Probabity of partyA being the rulling party = 0.29928
Overall Probabity of partyB being the rulling party = 0.41462
Overall Probabity of partyC being the rulling party = 0.28611

[Done] exited with code=0 in 1.6 seconds
```