EE 381 Project 6

Markov Chains

by Alexandra McBride

December 9, 2019

California State University, Long Beach

Dr. Anastasios Chassiakos

Problem 1

Introduction

This problem involves a 3-state Markov Chain. The state $n-1$ determines the next step for state n. This Markov Chain has an initial state probability vector $u^{(0)}$ of size 3, where 0 is the initial step, and a state transition matrix P with 3 rows and 3 columns. For this problem, we will start by observing a single run of the chain up to 15 steps and see what state it is in at each step in the chain. We then want to find the probability of being in each state at a given step in the chain. We will get these probabilities in two ways: 1) We will simulate the chain 10,000 times, gather the number of times the chain was at each state at a given step, and divide each of them by 10,000 and create our plot from these numbers, and 2) We will calculate the probabilities of the chain being in a given state at a given step , the vector $u^{(n)}$, by multiplying the vector $u^{(n-1)}$ by P , which will give us the probabilities of each of the states being the state at step n, and create a second plot from them. We will then compare the results of the simulated plot with the calculated results plot to see how similar the results are.

Methodology

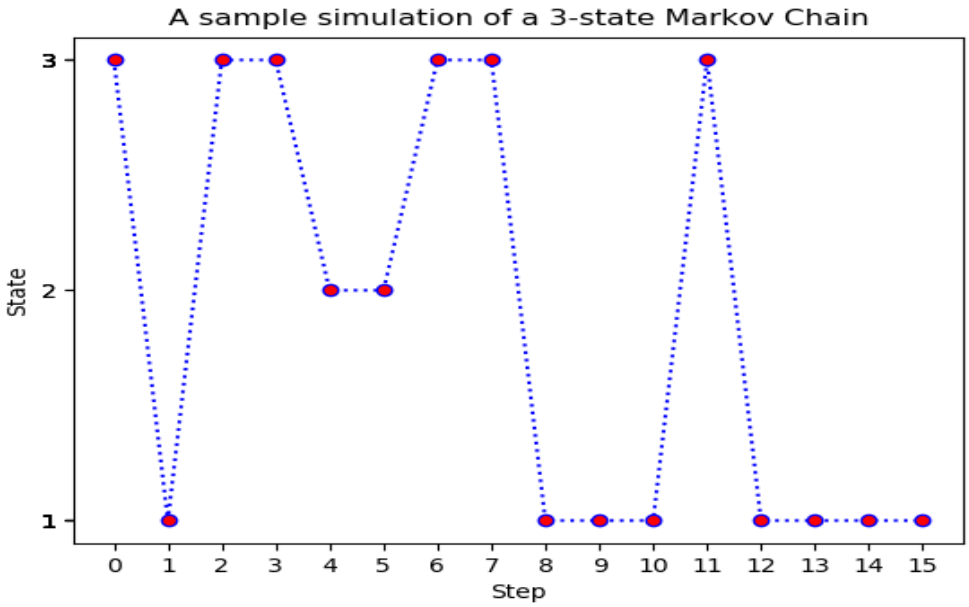
For this problem, I will be using Python in the PyCharm IDE. The tools used will include methods from the `numpy` and `mathplotlib.pyplot` libraries and a method called `nSidedDie`.

For the simulated plot, we will simulate the chain 10,000 times in order to find the probabilities. The program has 3 lists of size 16, which keeps count of how many times that step was in a given state. For the plot, I take each element in each of the lists, divide it by 10,000, then plot it on the graph.

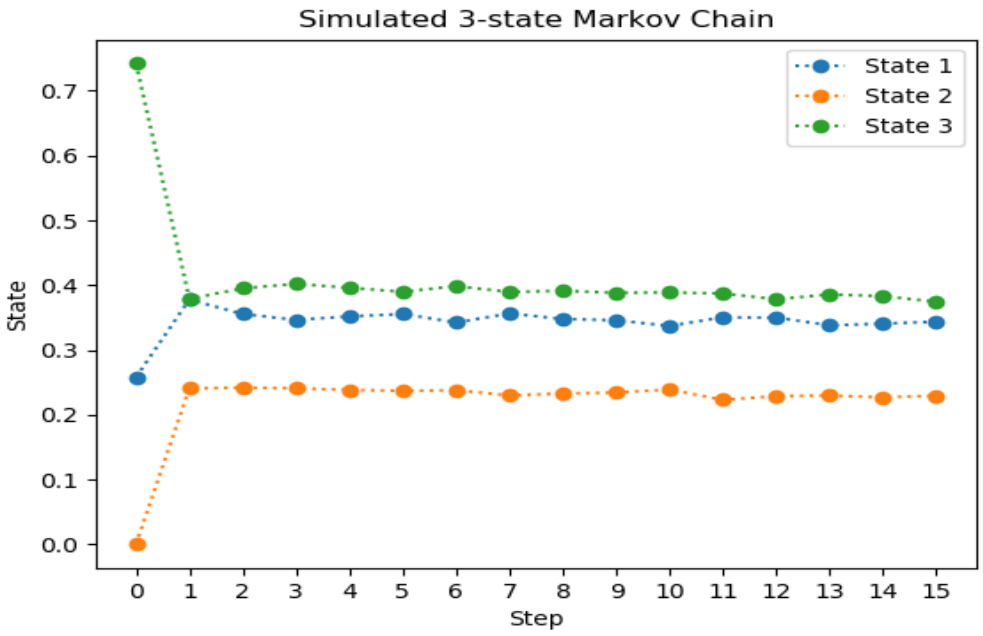
For the calculated plot, we will find the probabilities using the formula $u^{(n)} = u^{(n-1)}P$, where n=1,2,3…,15. For each step, we will plot each of the values inside the vector $u^{(n)}$ on the graph at step n. The plot will show starting from the initial step n = 0 up to n = 15.

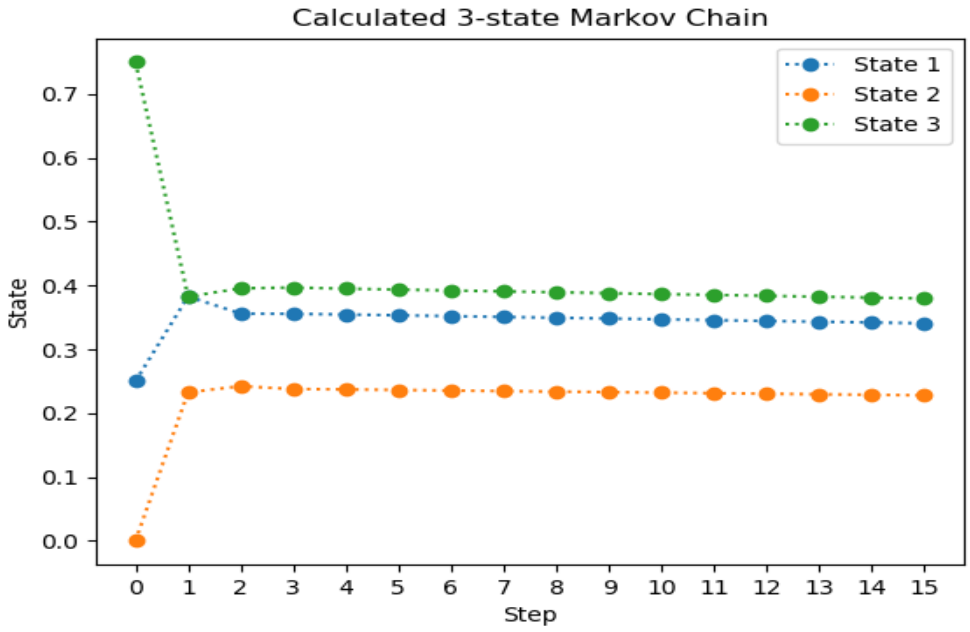
Results and Conclusion

Here is a plot of a sample simulation of the Markov Chain:



Here are the two plots of the combined results, which are 1) Simulated 3-state Markov Chain probabilities and 2) Calculated 3-state Markov Chain probabilities:





Comparing the results of the simulated probability graph to the calculated probability graph, the results are very similar, which confirms that the formula $u^{(n)} = u^{(n-1)}P$ does give us the correct values for these probabilities.

Appendix

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

#State Transition Matrix  P=[1/3  1/3  1/3;  1/3   1/6   1/2 ;  2/5    1/5    2/5]
p1= [0.33, 0.33, 0.33]
p2= [0.33, 0.17, 0.5]
p3= [0.4, 0.2, 0.4]

#Initial probability distribution vector:  [1/4    0    3/4]
V = [0.25,0.0, 0.75]

#Plotting a run of the chain

N=10000; n=15
states=np.linspace(0,15,16)
run=np.zeros((n+1,1))
initial=nsd.rollDie(V)
run[0]=initial
print(initial)
next=0
```

```
#Plotting a run of the chain
for j in range(1,n+1):
    if initial == 1:
        next = nsd.rollDie(p1)
    elif initial== 2:
        next = nsd.rollDie(p2)
    elif initial== 3:
        next = nsd.rollDie(p3)
    run[j]=next
plt.plot(states,run,'b',linestyle='dotted', marker='o', markerfacecolor='r')
plt.xticks(states)
plt.yticks(run)
plt.title("A sample simulation of a 3-state Markov Chain")
plt.xlabel("Step")
plt.ylabel("State")
plt.savefig("Simulation.png")
plt.close('all')

#Plotting based off 10,000 runs
state_1_probs= np.zeros((n+1,1))
state_2_probs= np.zeros((n+1,1))
state_3_probs= np.zeros((n+1,1))

for i in range(N):
    state= nsd.rollDie(V)
    if state== 1: state_1_probs[0]+=1
    elif state == 2: state_2_probs[0]+=1
    elif state == 3: state_3_probs[0]+=1
    for j in range(1,n+1):
        if state ==1:
            state=nsd.rollDie(p1)
        elif state == 2:
            state = nsd.rollDie(p2)
        elif state == 3:
            state = nsd.rollDie(p3)
        if state == 1:
            state_1_probs[j] += 1
        elif state == 2:
            state_2_probs[j] += 1
        elif state == 3:
            state_3_probs[j] += 1


plt.plot(states,(state_1_probs/N),'C0',linestyle='dotted', marker='o', markerfacecolor='C0',
label= 'State 1')
plt.plot(states,(state_2_probs/N),'C1',linestyle='dotted', marker='o', markerfacecolor='C1',
```

```
label= 'State 2')
plt.plot(states,(state_3_probs/N),'C2',linestyle='dotted', marker='o', markerfacecolor='C2',
label= 'State 3')
plt.legend(loc = "upper right")
plt.xticks(states)
plt.title("Simulated 3-state Markov Chain")
plt.xlabel("Step")
plt.ylabel("State")
plt.savefig("SimulatedResult.png")
plt.close("all")

#Plotting based off calculated probabilities
state_1_probs= np.zeros((n+1,1))
state_2_probs= np.zeros((n+1,1))
state_3_probs= np.zeros((n+1,1))
st=np.array([p1,p2,p3])
w=V
state_1_probs[0]=w[0]
state_2_probs[0]=w[1]
state_3_probs[0]=w[2]

for i in range(1,n+1):
    w=np.dot(w,st)
    print(w)
    state_1_probs[i] = w[0]
    state_2_probs[i] = w[1]
    state_3_probs[i] = w[2]

plt.plot(states,state_1_probs,'C0',linestyle='dotted', marker='o', markerfacecolor='C0', label=
'State 1')
plt.plot(states,state_2_probs,'C1',linestyle='dotted', marker='o', markerfacecolor='C1', label=
'State 2')
plt.plot(states,state_3_probs,'C2',linestyle='dotted', marker='o', markerfacecolor='C2', label=
'State 3')
plt.legend(loc = "upper right")
plt.xticks(states)
plt.title("Calculated 3-state Markov Chain")
plt.xlabel("Step")
plt.ylabel("State")
plt.savefig("CalculatedResult.png")
plt.close("all")
```

Problem 2

Introduction

This problem looks at Google's PageRank algorithm, which is an algorithm that uses Markov Chains in order to rank webpages for searching purposes. As in Problem 1, we start initial state probability vector $u^{(0)}$ of size 5 for pages A-E and a state transition matrix P with 5 rows and 5 columns. We then want to find the probability of being in each state at a given step in the chain, so we will again use the formula $u^{(n)} = u^{(n-1)}P$ to calculate the probabilities. We will then plot these probabilities on a graph. We will also change the initial vector $u^{(0)}$ and create a second plot in order to view how the probabilities have changed. We will observe the result of both graphs and compare them.

Methodology

For this problem, I will be using Python in the PyCharm IDE. The tools used will include methods from the `numpy` and `mathplotlib.pyplot` libraries and a method called `nSidedDie`.

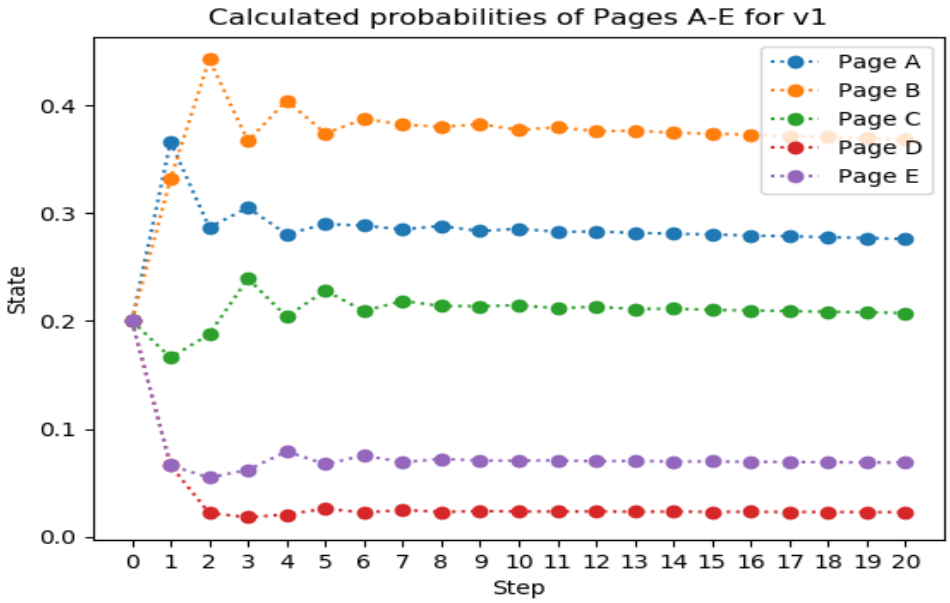
For calculating the probabilities, we will find the probabilities using the formula $u^{(n)} = u^{(n-1)}P$, where n=1,2,3…,15. For each step, we will plot each of the values inside the vector $u^{(n)}$ on the graph at step n. The plot will show starting from the initial step n = 0 up to n = 20.

Results & Conclusion

Here is the matrix *P* that was constructed:

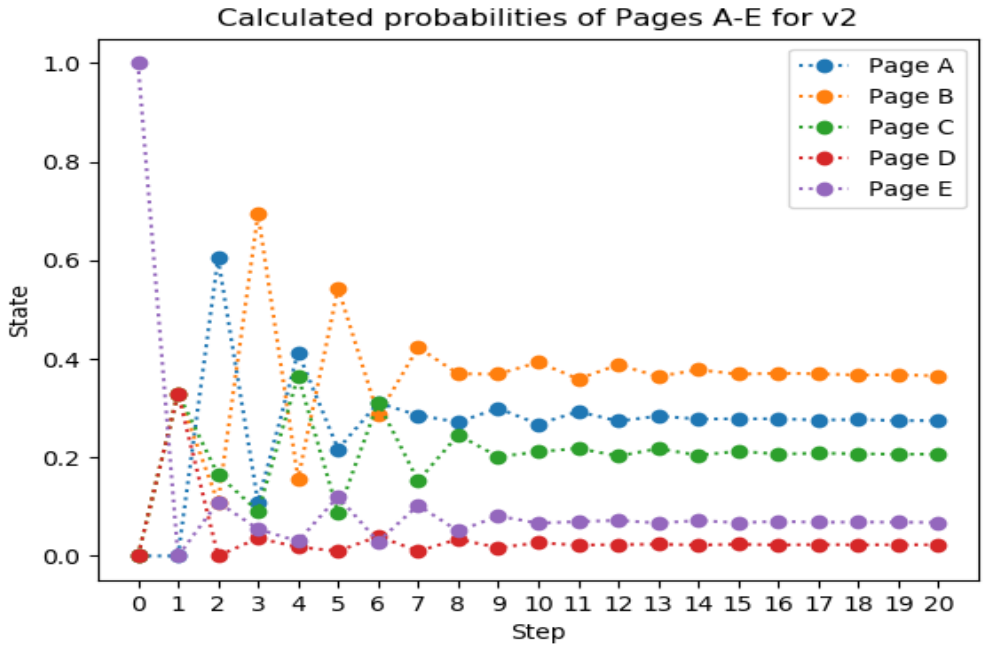
$$\begin{bmatrix} 0.0 , 1.0, 0.0, 0.0 , 0.0 \\ 0.5 \ ,0.0, 0.5, 0.0, 0.0 \\ 0.33 , 0.33, 0.0, 0.0,0.33 \\ 1.0 \ , 0.0, \ \ 0.0, 0.0, 0.0 \\ 0.0 , \ 0.33, 0.33, 0.33, 0.0 \end{bmatrix}$$

Here is the plot of the calculated probabilities for initial vector v_1 followed by their rankings:



Initial Probability Vector v_1		
Rank	Page	Probability
1	B	0.36827399
2	A	0.27603665
3	C	0.20731014
4	E	0.06865078
5	D	0.02270821

Here is the plot of the calculated probabilities for initial vector v_2 followed by their rankings:



Initial Probability Vector v_2		
Rank	Page	Probability
1	B	0.36506877
2	A	0.2746869
3	C	0.20650271
4	E	0.06805584
5	D	0.02257496

From the results, we see that even as the initial probability vector changes from v_1 to v_2, the results for the rankings still remains the same, which shows that the probability vector reached a fixed set of values over many steps.

Appendix

```
#Google PageRank Problem
import numpy as np
import matplotlib.pyplot as plt

n=20 #number of steps
steps=np.linspace(0,n,n+1)
#State Transition Matrix
P=np.array([[0.0,1.0,0.0,0.0,0.0],
        [0.5, 0.0, 0.5, 0.0, 0.0],
        [0.33, 0.33, 0.0, 0.0, 0.33],
        [1.0, 0.0, 0.0, 0.0, 0.0],
        [0.0, 0.33, 0.33, 0.33, 0.0]])
```

```
#Initial Probability vector v1

v1=[0.2,0.2,0.2,0.2,0.2]

A_probs=np.zeros((n+1,1))
B_probs=np.zeros((n+1,1))
C_probs=np.zeros((n+1,1))
D_probs=np.zeros((n+1,1))
E_probs=np.zeros((n+1,1))
A_probs[0]=v1[0]
B_probs[0]=v1[1]
C_probs[0]=v1[2]
D_probs[0]=v1[3]
E_probs[0]=v1[4]

for i in range(1,n+1):
    v1 = np.dot(v1, P)
    print(v1)
    A_probs[i]=v1[0]
    B_probs[i]=v1[1]
    C_probs[i]=v1[2]
    D_probs[i]=v1[3]
    E_probs[i]=v1[4]


plt.plot(steps,A_probs,'C0',linestyle='dotted', marker='o', markerfacecolor='C0', label= 'Page A')
plt.plot(steps,B_probs,'C1',linestyle='dotted', marker='o', markerfacecolor='C1', label= 'Page B')
plt.plot(steps,C_probs,'C2',linestyle='dotted', marker='o', markerfacecolor='C2', label= 'Page C')
plt.plot(steps,D_probs,'C3',linestyle='dotted', marker='o', markerfacecolor='C3', label= 'Page D')
plt.plot(steps,E_probs,'C4',linestyle='dotted', marker='o', markerfacecolor='C4', label= 'Page E')
plt.legend(loc = "upper right")
plt.xticks(steps)
plt.title("Calculated probabilities of Pages A-E for v1")
plt.xlabel("Step")
plt.ylabel("State")
plt.savefig("PageRankResult_v1.png")
plt.close("all")

#Initial Probability vector v1
print()
```

```
v2=[0.0,0.0,0.0,0.0,1.0]

A_probs=np.zeros((n+1,1))
B_probs=np.zeros((n+1,1))
C_probs=np.zeros((n+1,1))
D_probs=np.zeros((n+1,1))
E_probs=np.zeros((n+1,1))
A_probs[0]=v2[0]
B_probs[0]=v2[1]
C_probs[0]=v2[2]
D_probs[0]=v2[3]
E_probs[0]=v2[4]

for i in range(1,n+1):
    v2 = np.dot(v2, P)
    print(v2)
    A_probs[i]=v2[0]
    B_probs[i]=v2[1]
    C_probs[i]=v2[2]
    D_probs[i]=v2[3]
    E_probs[i]=v2[4]

plt.plot(steps,A_probs,'C0',linestyle='dotted', marker='o', markerfacecolor='C0', label= 'Page A')
plt.plot(steps,B_probs,'C1',linestyle='dotted', marker='o', markerfacecolor='C1', label= 'Page B')
plt.plot(steps,C_probs,'C2',linestyle='dotted', marker='o', markerfacecolor='C2', label= 'Page C')
plt.plot(steps,D_probs,'C3',linestyle='dotted', marker='o', markerfacecolor='C3', label= 'Page D')
plt.plot(steps,E_probs,'C4',linestyle='dotted', marker='o', markerfacecolor='C4', label= 'Page E')
plt.legend(loc = "upper right")
plt.xticks(steps)
plt.title("Calculated probabilities of Pages A-E for v2")
plt.xlabel("Step")
plt.ylabel("State")
plt.savefig("PageRankResult_v2.png")
plt.close("all")
```

Problem 3

Introduction

For this problem, we are looking at absorbing Markov Chains, where certain states are called absorbing state where once reached, the chain will always stay in that state in the following steps. In this problem, we want to the Drunkard's Walk scenario, where state 1-3 are transient states, while 0 and 4 are absorbing states. So, we plot two simulations, one where the chain is absorbed by state 0, and one where the chain is absorbed by state 4. We will then analyze the plots and discuss the results.

Methodology

For this problem, I will be using Python in the PyCharm IDE. The tools used will include methods from the `numpy` and `mathplotlib.pyplot` libraries and a method called `nSidedDie`.

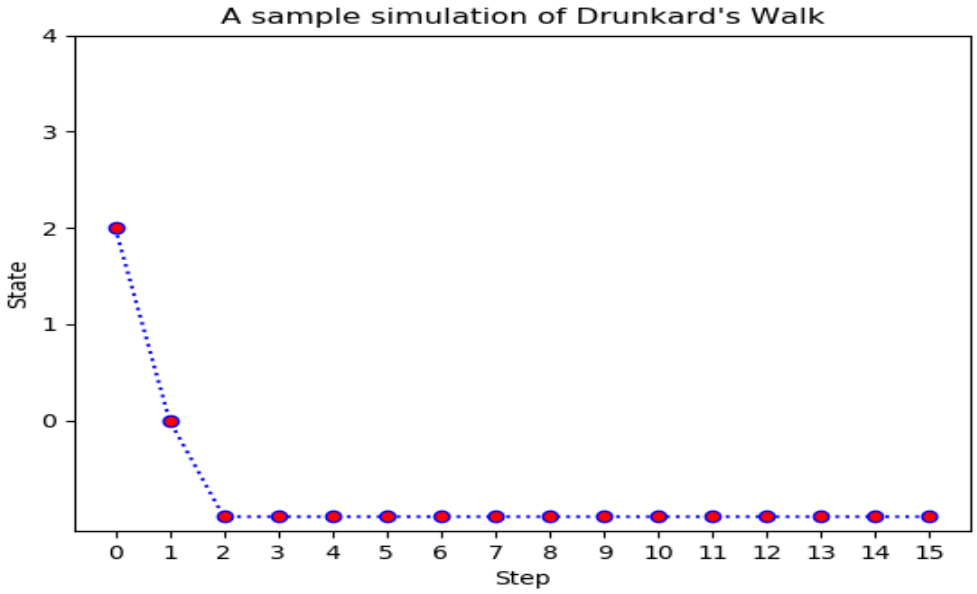
For the simulated plots, we will have our initial probability vector $u^{(0)}$ and state transition matrix P. We will generate a result using the nsidedDie function by passing in $u^{(0)}$, and for the following states, we will pass in the row in P which corresponds to the state. (i.e state 0 uses $P[0]$). We then save the state at each step in a results array and plot them.

Results & Conclusion

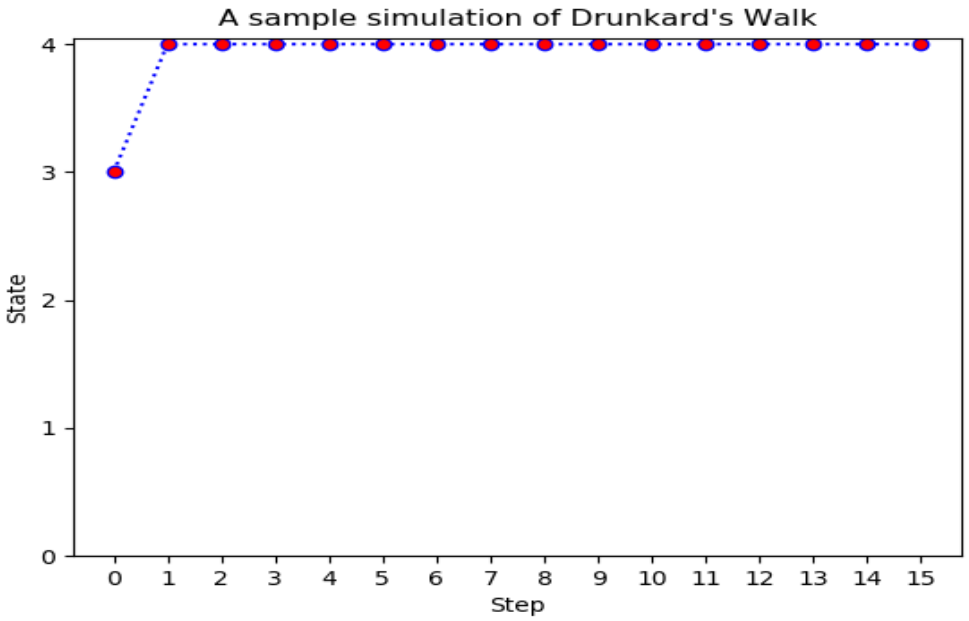
Here is the state transition matrix:

[1.0,0.0,0.0,0.0,0.0]
[0.67, 0.0, 0.33, 0.0, 0.0]
[0.0, 0.6, 0.0, 0.4, 0.0]
[0.0, 0.0, 0.3, 0.0, 0.7]
[0.0, 0.0, 0.0, 0.0, 1.0]

Here is a simulation of the Drunkard's Walk where the chain is absorbed at state 0:



Here is a simulation of the Drunkard's Walk where the chain is absorbed at state 4:



From these two results, we see the same pattern, where once state 0 or state 4 is reached, it remains in that state until the last step. We can now visually see the behavior of an absorbing chain.

Appendix

```
#Absorbing States          Transition Probabilities:   a=2/3;  b=3/5;  c=3/10;
import numpy as np
import matplotlib.pyplot as plt
import nSidedDie as nsd

n=15
P=np.array([[1.0,0.0,0.0,0.0,0.0],
            [0.67, 0.0, 0.33, 0.0, 0.0],
            [0.0, 0.6, 0.0, 0.4, 0.0],
            [0.0, 0.0, 0.3, 0.0, 0.7],
            [0.0, 0.0, 0.0, 0.0, 1.0]])

w=[0.0,0.33,0.33,0.33,0.0]

steps=np.linspace(0,n,n+1)
states=np.linspace(0,4,5)
run=np.zeros((n+1,1))
initial=nsd.rollDie(w)
run[0]=initial
print(initial)
next=initial
print(next)

#Plotting a run of the chain
for j in range(1,n+1):
    if  next== 0:
        next= nsd.rollDie(P[0])
    elif next == 1:
        next = nsd.rollDie(P[1])
    elif next== 2:
        next = nsd.rollDie(P[2])
    elif next== 3:
        next = nsd.rollDie(P[3])
    elif next == 4:
         next = nsd.rollDie(P[4])
    print(next)
    run[j]=next
plt.plot(steps,run,'b',linestyle='dotted', marker='o', markerfacecolor='r')
plt.xticks(steps)
plt.yticks(states)
plt.title("A sample simulation of Drunkard's Walk")
plt.xlabel("Step")
plt.ylabel("State")
plt.savefig("Simulation_P3.png")
plt.close('all')
```

Problem 4

Introduction

For this problem, we are looking again at absorbing Markov Chains and the Drunkard's Walk scenarion. However in this problem, we want to get the probability b_{20} that the chain will be absorbed by state 0 and b_{24} that the chain will be absorbed by state 4. For these, the state will always begin in state 2. In this problem, we want to find these probabilities after simulating the absorbing chain 10,000 times.

Methodology

For this problem, I will be using Python in the PyCharm IDE. The tools used will include methods from the `numpy` and `mathplotlib.pyplot` libraries and a method called `nSidedDie`.

For the simulated plot, we will simulate the chain 10,000 times in order to find the probabilities b_{20} and b_{24}. For each run of the chain, we will go up to 15 steps, then check if the chain ends up at either 0 or 4, then add to counters made for both these states to see how many times out of 10,00 runs that the state ends in that state. We will then divide the result by 10,000 to get the probabilities.

Results & Conclusion

Here is the state transition matrix:

[1.0,0.0,0.0,0.0,0.0]
[0.67, 0.0, 0.33, 0.0, 0.0]
[0.0, 0.6, 0.0, 0.4, 0.0]
[0.0, 0.0, 0.3, 0.0, 0.7]
[0.0, 0.0, 0.0, 0.0, 1.0]

Absorption Probabilities (via simulation)			
b_{20}	0.5882	b_{24}	0.4116

We see that the probability of the chain being absorbed at state 0 is higher than the probability of the chain being absorbed at state 4, which is because the if the state starts at 1 or 2, it is more likely to move to state 0, whereas at state 3, it is more like to move to state 4.

Appendix

```
#Problem 4
N=10000
v1=[0.0,0.0,1.0,0.0,0.0]

state_0_ends=0
state_4_ends=0


for i in range(N):
    state= nsd.rollDie(v1)
    for j in range(1,n):
        if state == 0:
            state=nsd.rollDie(P[0])
        elif state == 1:
            state = nsd.rollDie(P[1])
        elif state== 2:
            state= nsd.rollDie(P[2])
        elif state == 3:
            state = nsd.rollDie(P[3])
        elif state == 4:
            state = nsd.rollDie(P[4])
    if state == 0:
      state_0_ends += 1
    elif state == 4:
      state_4_ends += 1

print(state_0_ends/N)
print(state_4_ends/N)
```

`nSidedDie` program used in the problems

```
import numpy as np
def rollDie(p):
    #Getting n (number of sides)
    n=len(p)

    #Setting up the cumulative sum
    cumulProbs = np.cumsum(p)  # array of sums
    cumulProbs0 = np.append(0, cumulProbs)

    #random number between 0 and 1
    randNum = np.random.rand()
    for j in range(0, len(p)):
        # checks for the range that the dieRoll is in and get the probability for that range
        if randNum > cumulProbs0[j] and randNum <= cumulProbs0[j + 1]:
            side = j+1
            return side
    return 0
```