Project 6 - Markov Chains

California State University Long Beach

EE 381

Probability and Statistics

Christopher Masferrer

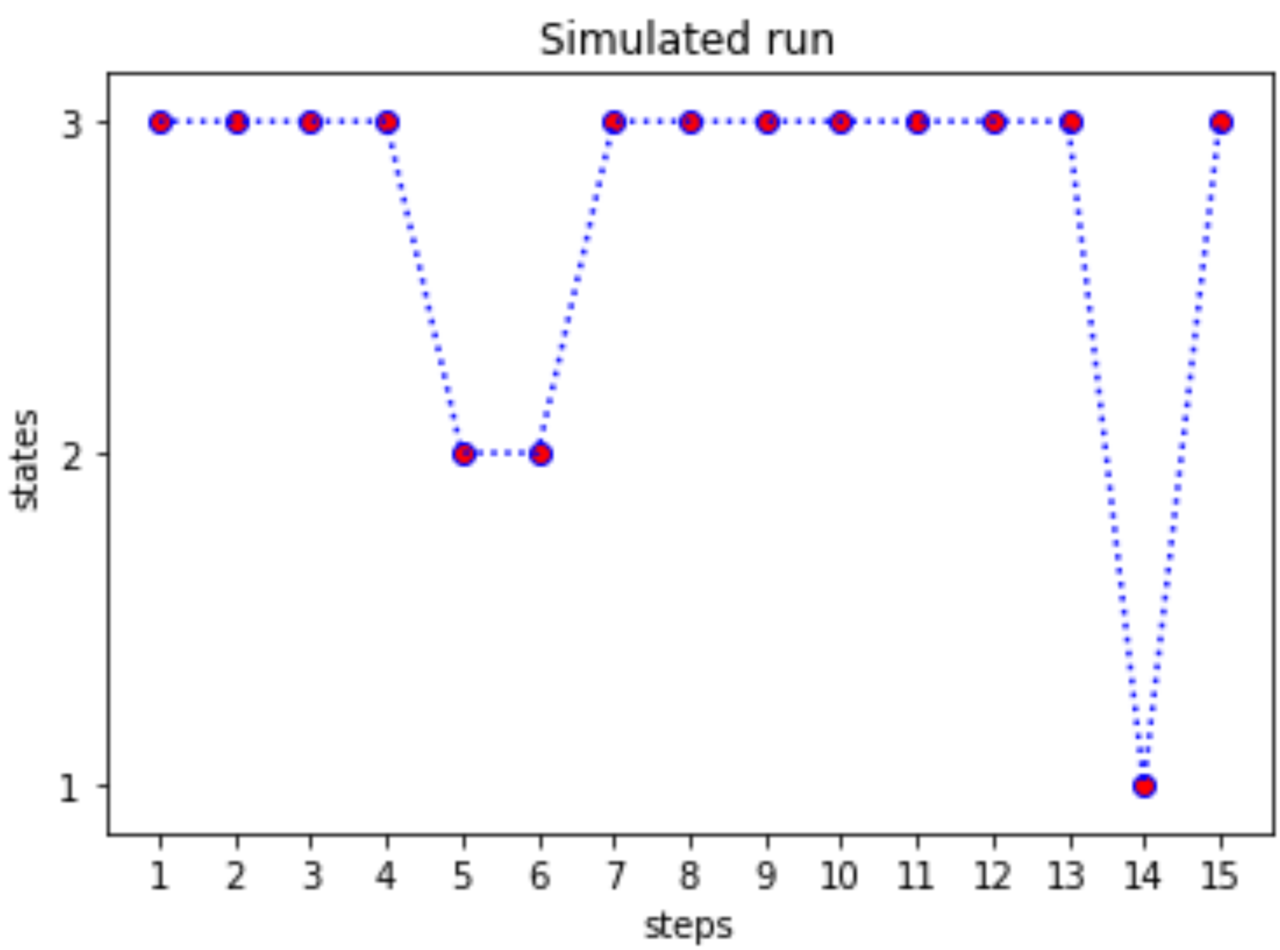
T/TH 5:30 - 6:20

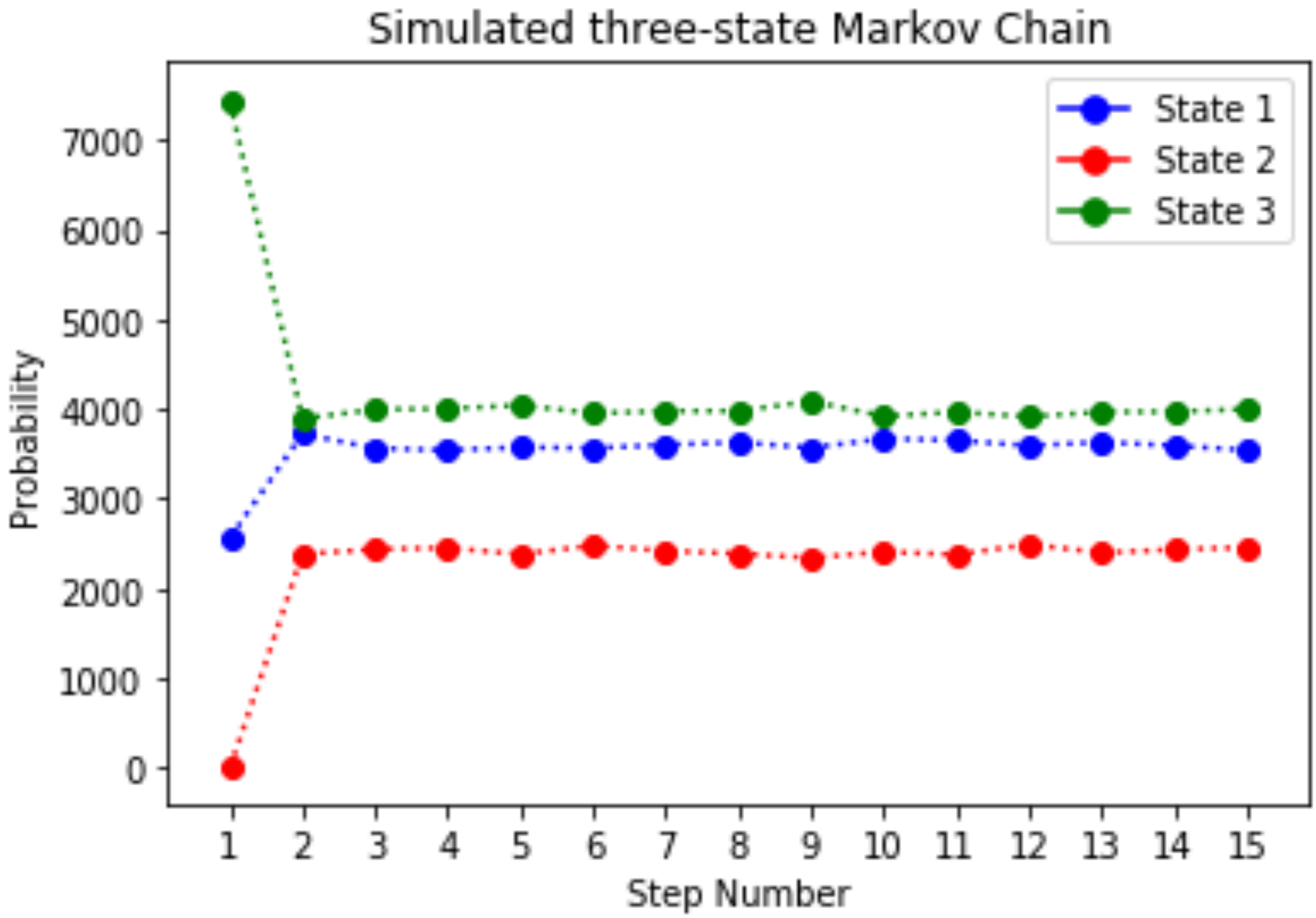
Experiment 1: A three-state Markov Chain

Intro: For this experiment I will be generating two charts. One will be a single simulation run of a three-state Markov Chain, the other will be the result of 10,000 runs. Both charts will be displayed in the report below.

Methodology: I created a main function that generates both charts. The results of the first run are recorded separately from the combined runs of 10,000 simulations. The n-sided dice function from the first lab is used to generate the result of a single roll.

Results:




Simulated three-state Markov Chain
State 1
State 2
State 3
Probability
Step Number
0
1000
2000
3000
4000
5000
6000
7000
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Code:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Nov 22 12:41:40 2019

@author: christophermasferrer
"""

#Christopher Masferrer
#EE 381
#Lab 6
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.lines as mlines


def threeState():
    n = 15
    N = 10000
    firstRun = np.zeros(n)
    state1 = np.zeros(n)
    state2 = np.zeros(n)
    state3 = np.zeros(n)
    for i in range (0, N):
        index = 0
        roll = 0
        for j in range (0, n):
            if index == 0:
                roll = nSidedDie([1/4, 0, 3/4])
                index = roll
                firstRun[j] = index
            if index == 1:
                roll = nSidedDie([1/3, 1/3, 1/3])
                index = roll
                state1[j] += 1
                firstRun[j] = index
            elif index == 2:
                roll = nSidedDie([1/3, 1/6, 1/2])
                index = roll
                state2[j] += 1
                firstRun[j] = index
            elif index == 3:
                roll = nSidedDie([2/5, 1/5, 2/5])
                index = roll
                state3[j] += 1
```

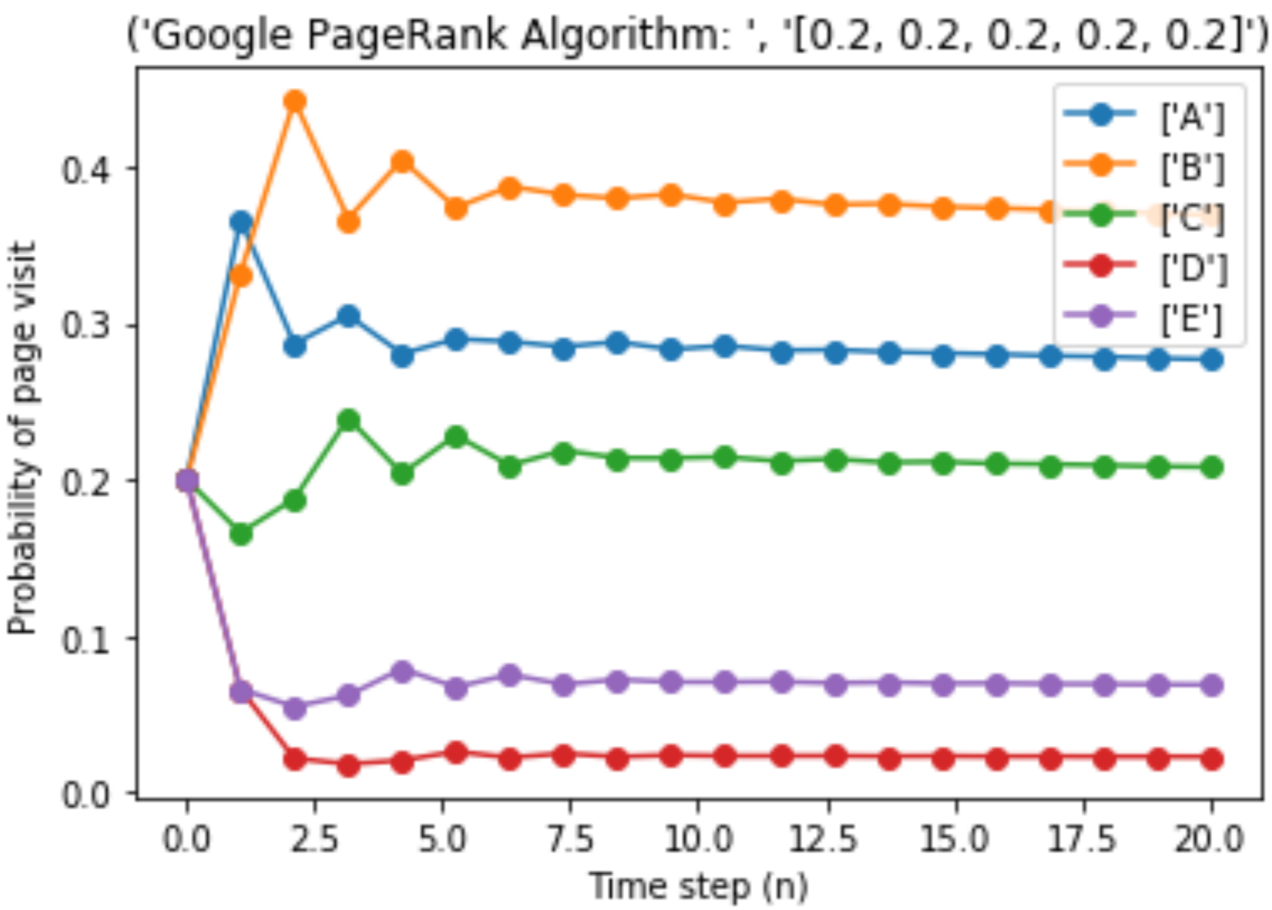
```
                firstRun[j] = index

        if i == 0:
            plt.yticks([1, 2, 3])
            plt.scatter(np.arange(len(firstRun)), firstRun, color='r', edgecolors='b')
            plt.xticks(np.arange(len(firstRun)), np.arange(1, len(firstRun) + 1))
            plt.plot(firstRun, 'b:')
            plt.ylabel('states')
            plt.xlabel('steps')
            plt.title('Simulated run')
            plt.show()

    plt.scatter(np.arange(len(state1)), state1, color='b', edgecolors='b')
    plt.scatter(np.arange(len(state2)), state2, color='r', edgecolors='r')
    plt.scatter(np.arange(len(state3)), state3, color='g', edgecolors='g')
    plt.xticks(np.arange(len(state1)), np.arange(1, len(state1) + 1))
    plt.plot(state1, 'b:')
    plt.plot(state2, 'r:')
    plt.plot(state3, 'g:')
    plt.ylabel('Probability')
    plt.xlabel('Step Number')
    plt.title('Simulated three-state Markov Chain')
    l1 = mlines.Line2D([], [], color='blue', marker='.', label='State 1', markersize=15)
    l2 = mlines.Line2D([], [], color='red', marker='.', label='State 2', markersize=15)
    l3 = mlines.Line2D([], [], color='green', marker='.', label='State 3', markersize=15)
    plt.legend(handles=[l1, l2, l3])
    plt.show()

def nSidedDie(p):
    n = np.size(p)

    cs = np.cumsum(p)
    cp = np.append(0,cs)

    r = np.random.rand()
    for k in range (0, n):
        if r > cp[k] and r <= cp[k + 1]:
            d = k+1
    return d

threeState()
```

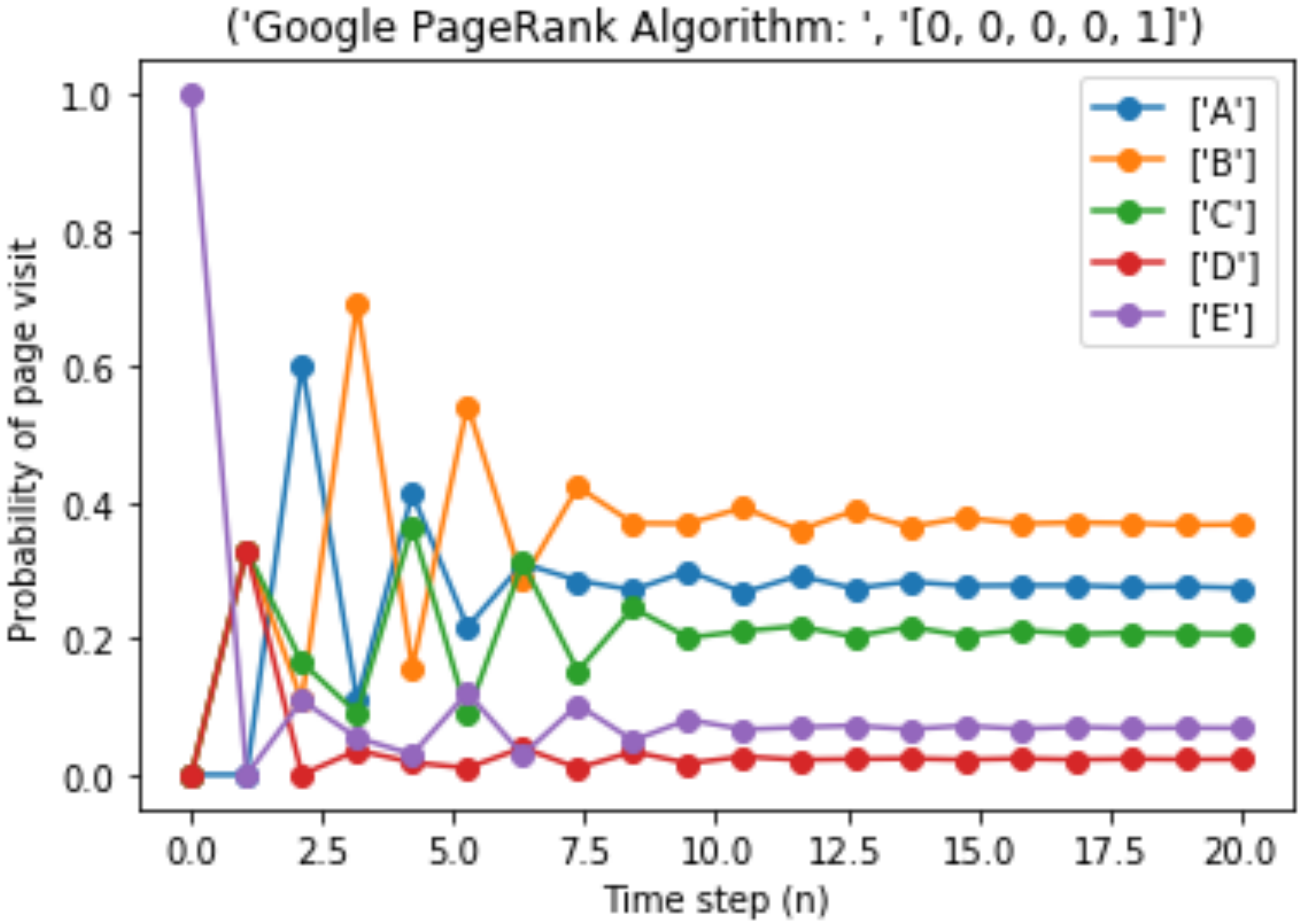
Experiment 2: Google PageRank Algorithm

Intro: For this experiment, I will be calculating the probabilities of a page displaying after n time steps have passed. The experiment will be conducted twice, once for vector v1 and once for vector v2, and the results will be displayed on two charts

Methodology: I created both v1 and v2 as well as the state transition matrix. A loop generates the percentages and is then used for the charts.

Results:




('Google PageRank Algorithm: ', '[0, 0, 0, 0, 1]')
['A']
['B']
['C']
['D']
['E']
1.0
0.8
0.6
0.4
0.2
0.0
Probability of page visit
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Time step (n)

Code:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sat Nov 30 18:01:43 2019

@author: christophermasferrer
"""

#Christopher Masferrer
#EE 381
#Lab 6

import numpy as np
import matplotlib.pyplot as plt

v1 = [.2, .2, .2, .2, .2]
v2 = [0, 0, 0, 0, 1]
transMatrix = np.matrix([[0, 1, 0, 0, 0], [0.5, 0, 0.5, 0, 0], [0.33, 0.33, 0, 0, 0.33], [1, 0, 0, 0, 0],
[0, 0.33, 0.33, 0.33, 0]])

def pageRank(v):
    n = 20
    result = np.zeros((n, 5))
    initial = v
    result[0, :] = initial
    for i in range(0, n-1):
        result[i + 1, :] = np.matmul(result[i, :], transMatrix)

    nv = np.linspace(0, n, num=20)
    plt.figure()
    plt.plot(nv, result, marker='o', markersize=6)
    plt.title(('Google PageRank Algorithm: ', np.str(v)))
    plt.xlabel('Time step (n)')
    plt.ylabel('Probability of page visit')
    plt.legend((['A'], ['B'], ['C'], ['D'], ['E']))
    plt.show()

pageRank(v1)
pageRank(v2)
```

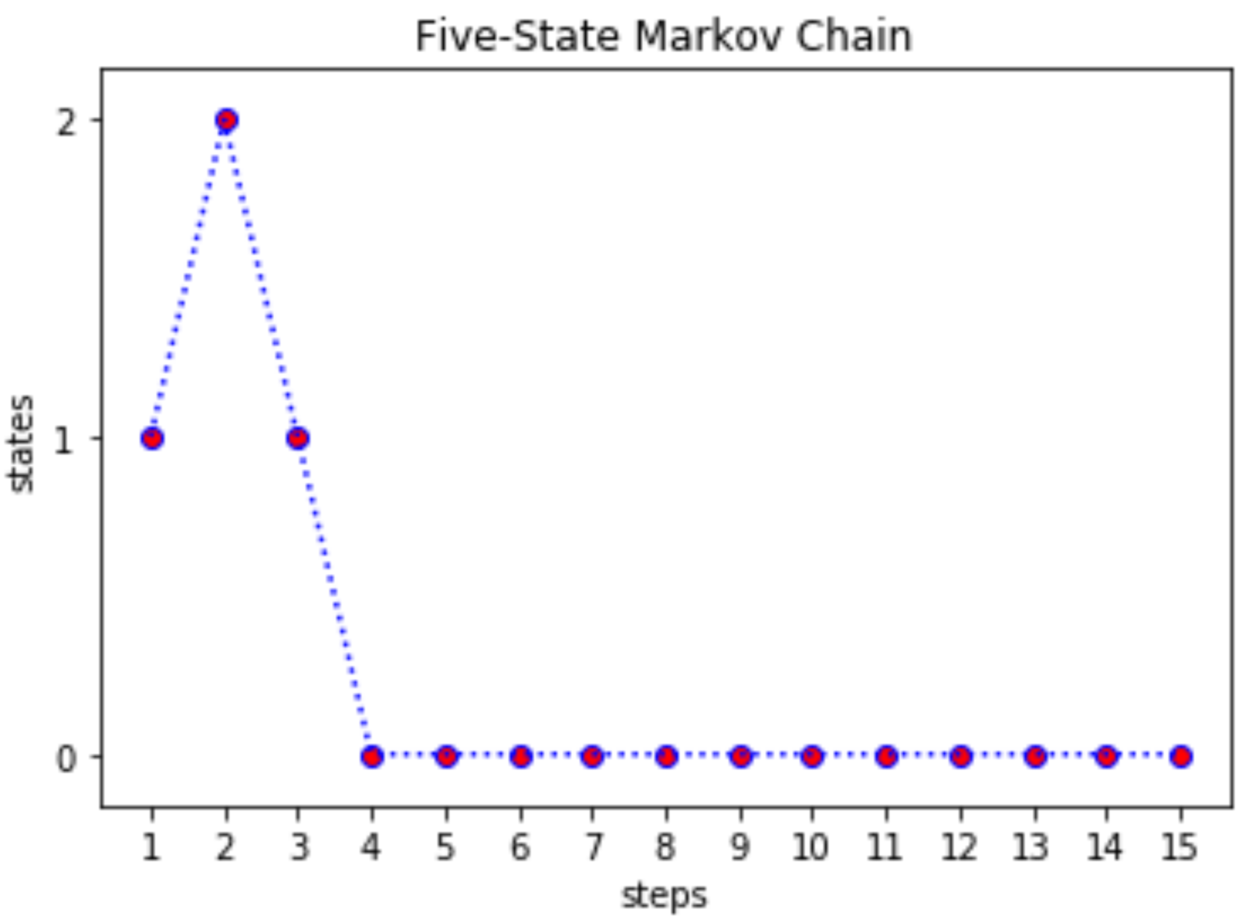
Experiment 3: Five-state absorbing Markov chain

Intro: For this experiment, I will be simulating a five-state Markov chain. The end result of the experiment will either end in state 0 or state 4 and their respective graphs will be printed and displayed.

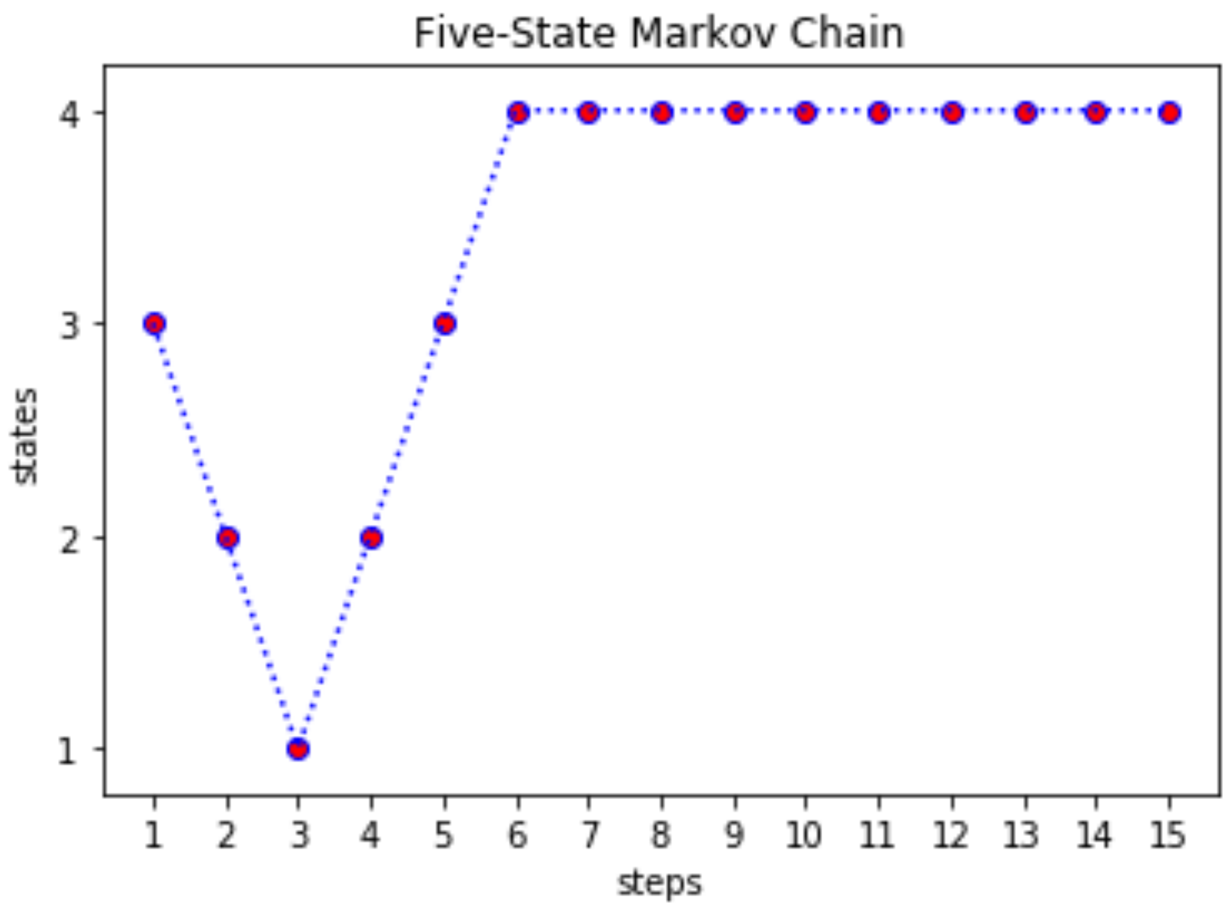
Methodology: Most of the code is similar to the three state Markov chain in experiment 1. I used the n-sided dice function to generate a single value that would be either 1, 2, or 3. A loop is created to determine which array will be used to generate the next value. The result is then printed in a chart

Results:

State 0 Result:



State 4 Result:



Code:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Dec  4 21:49:22 2019

@author: christophermasferrer
"""

#Christopher Masferrer
#EE 381
#Lab 6
import numpy as np
import matplotlib.pyplot as plt

def fiveState():
    n = 15
    singleRun = np.zeros(n)
    roll = nSidedDie([0, 1/3, 1/3, 1/3, 0])
    index = roll - 1
    for i in range (0, n):
        if index == 0:
            singleRun[i] = index
            roll = nSidedDie([1, 0, 0, 0, 0])
            index = roll - 1
        elif index == 1:
            singleRun[i] = index
            roll = nSidedDie([2/3, 0, 1/3, 0, 0])
            index = roll - 1
        elif index == 2:
            singleRun[i] = index
            roll = nSidedDie([0, 3/5, 0, 2/5, 0])
            index = roll - 1
        elif index == 3:
            singleRun[i] = index
            roll = nSidedDie([0, 0, 3/10, 0, 7/10])
            index = roll - 1
        elif index == 4:
            singleRun[i] = index
            roll = nSidedDie([0, 0, 0, 0, 1])
            index = roll - 1

    plt.yticks([0, 1, 2, 3, 4])
    plt.scatter(np.arange(len(singleRun)), singleRun, color='r', edgecolors='b')
    plt.xticks(np.arange(len(singleRun)), np.arange(1, len(singleRun) + 1))
```

```
    plt.plot(singleRun, 'b:')
    plt.ylabel('states')
    plt.xlabel('steps')
    plt.title('Five-State Markov Chain')
    plt.show()

def nSidedDie(p):
    n = np.size(p)

    cs = np.cumsum(p)
    cp = np.append(0,cs)

    r = np.random.rand()
    for k in range (0, n):
        if r > cp[k] and r <= cp[k + 1]:
            d = k+1
    return d

fiveState()
```

Experiment 4: Absorption using the simulated chain

Intro: For this experiment, I will be performing a variation of the previous experiment. The experiment will be run 10,000 times and the results of the number of successes for state 0 and state 4 will be printed

Methodology: Using similar code to the previous experiment, I removed the single run container and replaced it with accumulators of the amount of state 0 successes and state 4 successes. I kept the loop the same, just repeated 10,000 times, and recording whether it ended with state 0 or state 4. The results are as follows.

Results:

Absorption Probabilities (via Simulations)			
b_{20}	0.5895	b_{24}	0.4102

Code:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Dec  5 09:36:16 2019

@author: christophermasferrer
"""

#Christopher Masferrer
#EE 381
#Lab 6
import numpy as np

def absorption():
    n = 15
    N = 10000
    state0 = 0
    state4 = 0
    for i in range(0, N):
        roll = nSidedDie([0, 0, 1, 0, 0])
        index = roll - 1
        for j in range (0, n):
            if index == 0:
                roll = nSidedDie([1, 0, 0, 0, 0])
                index = roll - 1
            elif index == 1:
                roll = nSidedDie([2/3, 0, 1/3, 0, 0])
                index = roll - 1
            elif index == 2:
                roll = nSidedDie([0, 3/5, 0, 2/5, 0])
                index = roll - 1
            elif index == 3:
                roll = nSidedDie([0, 0, 3/10, 0, 7/10])
                index = roll - 1
            elif index == 4:
                roll = nSidedDie([0, 0, 0, 0, 1])
                index = roll - 1

        if index == 0:
            state0 += 1
        else:
            state4 += 1

    print("State 0 Successes: ", np.str(state0))
```

```
    print("State 4 Successes: ", np.str(state4))

def nSidedDie(p):
    n = np.size(p)

    cs = np.cumsum(p)
    cp = np.append(0,cs)

    r = np.random.rand()
    for k in range (0, n):
        if r > cp[k] and r <= cp[k + 1]:
            d = k+1
    return d

absorption()
```