НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
Факультет прикладної математики
Кафедра прикладної математики

Лабораторна робота №4

«Моделювання ланцюгів Маркова та розрахунок їх характеристик»
з кредитного модуля «Випадкові процеси»

Варіант 11

Виконав:
студент групи КМ-93
Костенко Олександр Андрійович
Викладач:
Пашко Анатолій Олексійович

Київ — 2021

Зміст

Завдання 1 3

Завдання 2 9

Завдання 1

Задано поглинаючи1 ланцюг Маркова.

11	0,18	0,23	0,09	0,12	0,12	0,29	0,22	0,15	0,01	0,0	0,3	0,1	0,6

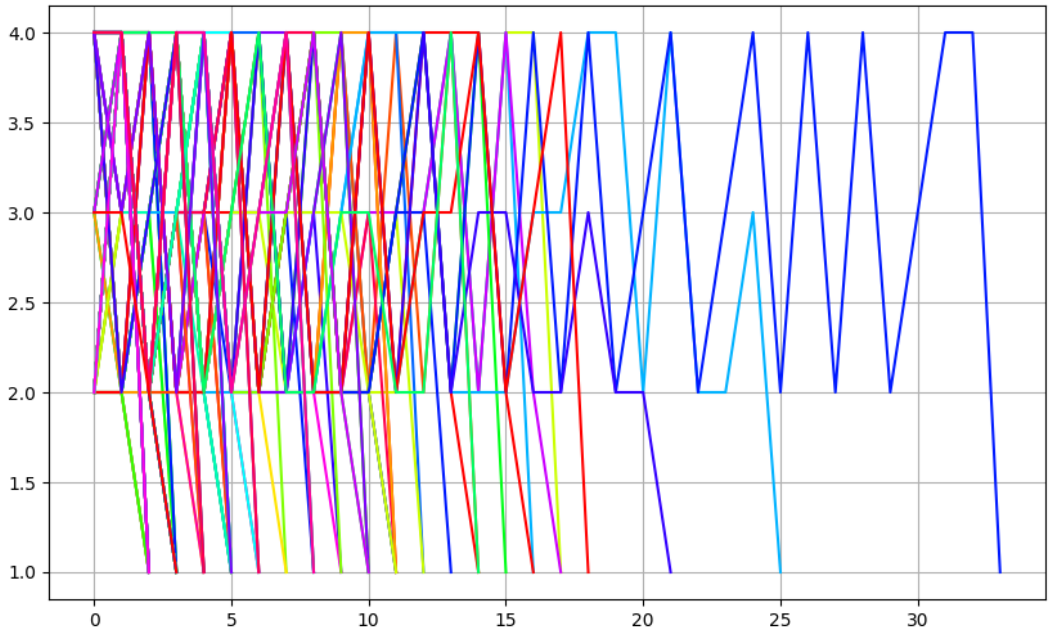
Змоделювати ланцюг Маркова з поглинанням для заданих перехідних і початкових ймовірностей (довжина реалізації – до поглинання). Кількість реалізацій – більше 100.

За отриманими реалізаціями знайти оцінки матриці переходів.

За отриманою матрицею переходів знайти:

- фундаментальну матрицю;
- середню кількість кроків, яку ланцюг знаходиться в стані j коли процес почався зі стану n;
- середній час поглинання;
- ймовірності поглинання.

Порівняти отримані результати з результатами лабораторної роботи 3.



1.1 Оцінки матриці переходів

```
[[1.         0.         0.         0.        ]
 [0.14393939 0.18939394 0.29924242 0.36742424]
 [0.09375    0.0859375  0.234375   0.5859375 ]
 [0.17421603 0.58188153 0.02439024 0.2195122 ]]
```

1.2 Фундаментальна матриця

```
Фундаментальна матриця:
[[3.132 1.302 2.452]
 [2.191 2.249 2.72 ]
 [2.403 1.041 3.194]]
```

1.3 При n=1 j=3, в даному випадку середня кількість кроків =0

1.4 Середній час поглинання

```
[[8.13888889]
 [8.66666667]
 [7.34615385]]
```

1.5 Ймовірність поглинання

```
Ймовірність поглинання:
[[1.00005838]
 [1.00008256]
 [0.99992611]]
```

Результати четвертої та третьої лабораторної роботи відрізняються але незначно.

Код програми:

```
import numpy as np
from matplotlib import pyplot as plt
from colorsys import hls_to_rgb

p0=[0.0,0.3,0.1,0.6]
p=np.array([1,0,0,0,0.18,0.23,0.5,0.09,0.12,0.12,0.29,0.47,0.22,0.62,0.15
,0.01]).reshape((4,4))
state=[]
time=[]
currentRow=p[0]

state=[]
time=[]
currentRow=p[0]


for i in range(100):
    state.append([])

def first(state,i):
    x=np.random.uniform(0.0,1.0)
    if x<p0[1]:
        k=2
        currentRow=p[1]
    elif x>=p0[1] and x<p0[2]+p0[1]:
        k=3
```

```
        currentRow=p[2]
    else:
        k=4
        currentRow=p[3]

    state[i].append(k)
    result=[]
    result.append(k)
    result.append(currentRow)
    return result

for i in range(100):
    k=first(state,i)[0]
    currentRow=first(state,i)[1]
    while k!=1:
        x=np.random.uniform(0.0,1.0)
        if x<currentRow[0]:
            k=1
            state[i].append(k)
            break
        elif x>=currentRow[0] and x<currentRow[1]+currentRow[0]:
            k=2
            currentRow=p[1]
        elif x>=currentRow[1] and x< currentRow[2]+currentRow[1]:
            k=3
            currentRow=p[2]
        else:
            k=4
            currentRow=p[3]
        state[i].append(k)



hue=0
fig=plt.subplots(figsize=(10,6))
plt.grid()

for i in range(len(state)):
    time=[]
    for j in range(len(state[i])):
        time.append(j)
    hue+=0.05
    color=hls_to_rgb(hue, 0.5, 1)
    plt.plot(time, state[i], color=color)
```

```
plt.show()


state_sum=0
for i in range(100):
    state_sum+=len(state[i])
print(state_sum)


matrix=[1,0,0,0]
count=[]
moves=[]

for i in range(3):
    moves.append(0)
    count.append([])
    for j in range(4):
        count[i].append(0)
for k in range(2,5):
    for i in range(100):
        for j in range(len(state[i])):
            if state[i][j]==k:
                if state[i][j+1]==1:
                    count[k-2][0]+=1
                elif state[i][j+1]==2:
                    count[k-2][1]+=1
                elif state[i][j+1]==3:
                    count[k-2][2]+=1
                else:
                    count[k-2][3]+=1
                moves[k-2]+=1

for i in range(3):
    for j in range(4):
        matrix.append(count[i][j]/moves[i])

matrix=np.array(matrix).reshape((4,4))
print(count)
print(matrix)

I=[]
O=[]
R=[]
Q=[]
I=matrix[0][0]
O=matrix[0][1:4]
```

```
for i in range(1,4):
    R.append(matrix[i][0])
    Q.append(matrix[i][1])
    Q.append(matrix[i][2])
    Q.append(matrix[i][3])
R=np.array(R).reshape((3,1))
Q=np.array(Q).reshape((3,3))
E=np.array([1,0,0,0,1,0,0,0,1]).reshape((3,3))
IminusQ=np.subtract(E,Q)
print(IminusQ)
fundmatrix=np.linalg.matrix_power(IminusQ,-1)
fundmatrix=np.matrix.round(fundmatrix, 3)
print("Фундаментальна матриця:")
print(fundmatrix)

av_time=state_sum/100
print(av_time)


sum1=0
sum2=0
sum3=0
c1=0
c2=0
c3=0

for i in range(100):
    if state[i][0]==2:
        sum1+=len(state[i])
        c1+=1
    elif state[i][0]==3:
        sum2+=len(state[i])
        c2+=1
    else:
        sum3+=len(state[i])
        c3+=1

av_time_list=[]
av_time_list.append(sum1/c1)
av_time_list.append(sum2/c2)
av_time_list.append(sum3/c3)

av_time_list=np.array(av_time_list).reshape(3,1)
print(av_time_list)

n=int(input("Введіть:"))
```

```
j=int(input("Введіть:"))

steps_j=0
steps_c=0
for i in range(100):
    if state[i][0]==n:
        steps_j+=state[i].count(j)
        steps_c+=1

if steps_c!=0:
    av_steps_j=steps_j/steps_c
else:
    av_steps_j=0

print(av_steps_j)

B=np.matmul(fundmatrix,R)
print("Ймовірність поглинання:")
print(B)
```

Завдання 2

Задано регулярний ланцюга Маркова.

11	0,40	0,33	0,13	0,08	0,27	0,32	0,04	0,42	0,54

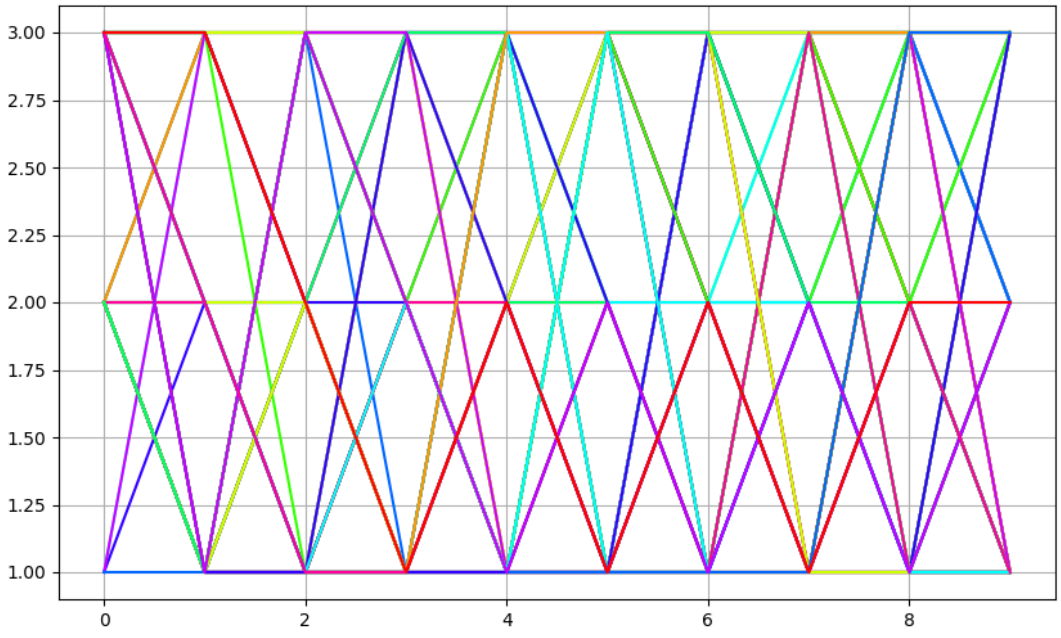
Змоделювати регулярний ланцюг Маркова для заданих перехідних і початкових ймовірностей (довжина реалізації – не менше 10). Кількість реалізацій – більше 100.

За отриманими реалізаціями знайти оцінки матриці переходів.

За знайденою матрицею переходів знайти:

- фінальні ймовірності;
- фундаментальну матрицю;
- середній час перебування в заданому стані за $n = 4$ кроків;
- середній час виходу ланцюга в заданий стан (в стан j, коли процес почався зі стану n);
- середній час виходу ланцюга в заданий стан в стаціонарному режимі (коли початковий стан не заданий).

Порівняти отримані результати з результатами лабораторної роботи 3.



1.1 Оцінки матриці переходів

```
[[0.373297   0.30790191 0.31880109]
 [0.80827068 0.11278195 0.07894737]
 [0.22846442 0.37827715 0.39325843]]
```

1.2 Маьриця фінальних ймовірностей

```
Матриця фінальних ймовірностей:
[[0.452753  0.2737383 0.2735087]
 [0.452753  0.2737383 0.2735087]
 [0.452753  0.2737383 0.2735087]]
```

1.3 Фундаментальна матриця

```
Фундаментальна матриця:
[[ 0.96687348  0.04602486 -0.01289834]
 [ 0.30897874  0.86839328 -0.17737202]
 [-0.14115976  0.11527694  1.02588282]]
```

1.4 середній час перебування в заданому стані за 4 кроки

```
[[1.47]
 [1.37]
 [1.16]]
```

1.5 При n=1 j=3 середній час виходу ланцюга в заданий стан = 2.33333333333335

Результати четвертої та третьої лабораторної роботи відрізняються але незначно.

Код програми:

```
import numpy as np
from matplotlib import pyplot as plt
from colorsys import hls_to_rgb


p=np.array([0.4,0.33,0.27,0.79,0.13,0.08,0.27,0.41,0.32]).reshape((3,3))
p0=[0.04,0.42,0.54]


state=[]
fig, ax=plt.subplots(figsize=(10,6))
hue=0
luminosity= 0.5
saturation=1
time=[]
for i in range(10):
    time.append(i)


for i in range(100):
    hue+=1/10
    color=hls_to_rgb(hue, luminosity, saturation)
    state.append([])
    currentRow=p0
    for j in range(10):
```

```
        x=np.random.uniform(0.0,1.0)
        if x>=0 and x<currentRow[0]:
            k=1
            currentRow=p[0]
        elif x>=currentRow[0] and x<(currentRow[1]+currentRow[0]):
            k=2
            currentRow=p[1]
        else:
            k=3
            currentRow=p[2]
        state[i].append(k)
    ax.plot(list(time),list(state[i]), color=color, label=str(i+1)+"-ий
ряд")

plt.grid()
plt.show()


step_sum=100*10


matrix=[]
count=[]
moves=[]

for i in range(3):
    moves.append(0)
    count.append([])
    for j in range(3):
        count[i].append(0)
for k in range(1,4):
    for i in range(100):
        for j in range(9):
            if state[i][j]==k:
                if state[i][j+1]==1:
                    count[k-1][0]+=1
                elif state[i][j+1]==2:
                    count[k-1][1]+=1
                elif state[i][j+1]==3:
                    count[k-1][2]+=1
                else:
                    count[k-1][3]+=1
                moves[k-1]+=1

for i in range(3):
    for j in range(3):
```

```
        matrix.append(count[i][j]/moves[i])

matrix=np.array(matrix).reshape((3,3))

print(matrix)


#Матриця фінальних імовірностей
wcurrent=np.around(np.matmul(p0, np.linalg.matrix_power(matrix,1)),7)

for i in range(2,15):

        wnext=np.around(np.matmul(p0, np.linalg.matrix_power(matrix,i)),
7)
        isequal=(wcurrent==wnext).all()
        if isequal==True:
            wline=wnext
            k=i-1
            break

        wcurrent=wnext

W=[]
for i in range(3):
    W.append(wline)

W=np.array(W).reshape((3,3))
print("Матриця фінальних ймовірностей:")
print(W)


#Фундаментальна матриця
I=np.array([1,0,0,0,1,0,0,0,1]).reshape((3,3))
Z=np.linalg.matrix_power(np.subtract(I,np.subtract(p,W)),-1)
print("Фундаментальна матриця:")
print(Z)

#Середній час перебування в заданому стані за n=4 кроків
state_n4=[]
for i in range(3):
    state_n4.append(0)

for i in range(100):
    for j in range(4):
        if state[i][j]==1:
            state_n4[0]+=1
```

```
        elif state[i][j]==2:
            state_n4[1]+=1
        else:
            state_n4[2]+=1

for i in range(3):
    state_n4[i]/=100

state_n4=np.array(state_n4).reshape(3,1)
print(state_n4)

#Середній час виходу ланцюга в заданий стан (в стан j , коли
процеспочався зі стану n)
n=int(input("Введіть:"))
j=int(input("Введіть:"))
jsum=0
jcount=0

for i in range(100):
    if state[i][0]==n:
        if j in state[i]:
            jsum+=state[i].index(j)
            jcount+=1
        else:
            pass

print(jsum/jcount)


#Середній час виходу ланцюга в заданий стан (коли початковий стан не
заданий)

jsum=0
jcount=0

for i in range(100):
    if j in state[i]:
        jsum+=state[i].index(j)
        jcount+=1
    else:
        pass

print(jsum/jcount)
```