Assignment_3

September 27, 2023

1 Assignment 3

[11]:
```
import matplotlib.pyplot as plt
import math
import random
import numpy as np
import sys
```

1.1 Exercise 1

1.1.1 a)

This is Markov Matrix: $\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.9 & 0 & 0.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.8 & 0 & 0.2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.7 & 0 & 0.3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.6 & 0 & 0.4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.5 & 0 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.4 & 0 & 0.6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.3 & 0 & 0.7 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.2 & 0 & 0.8 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.1 & 0.9 \end{bmatrix}$

State vector: $\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \end{bmatrix}$,

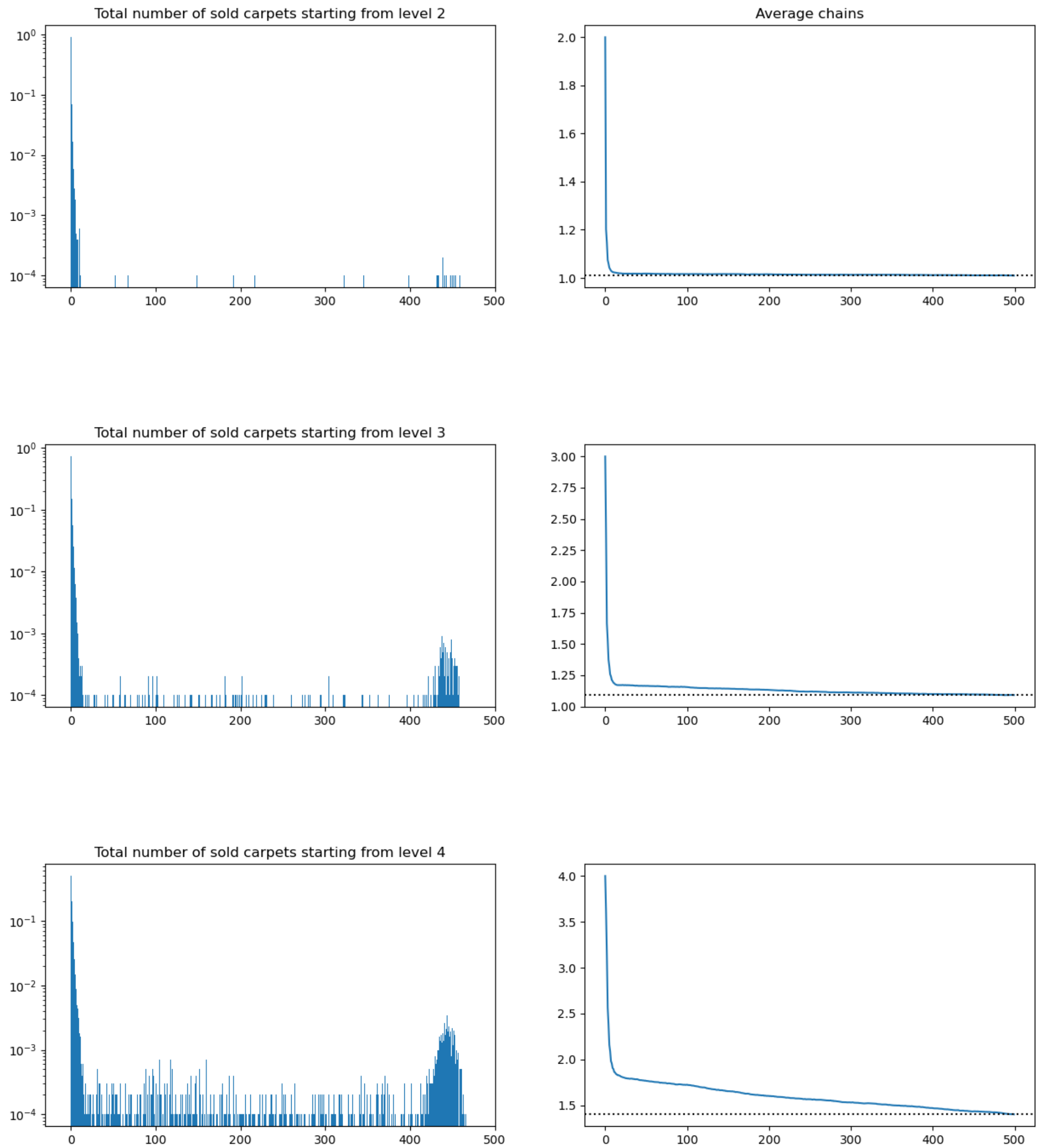
with sale probabilities $\begin{bmatrix} 0 & 0.1 & 0.2 & 0.3 & 0.4 & 0.5 & 0.6 & 0.7 & 0.8 & 0.9 \end{bmatrix}$

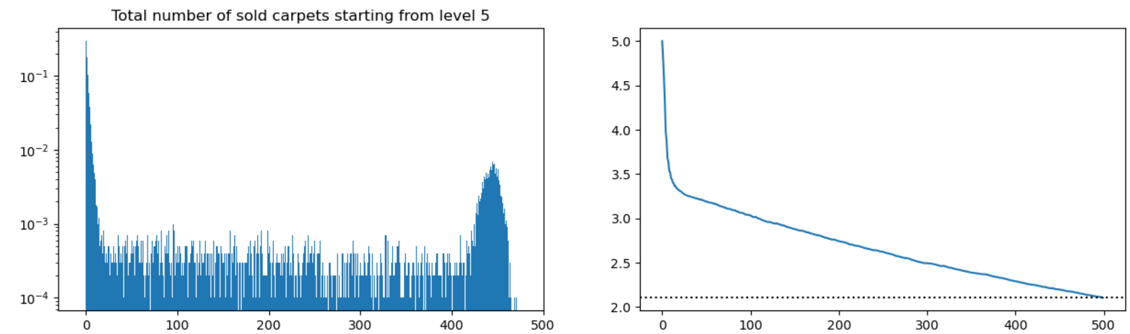
1.1.2 b and c)

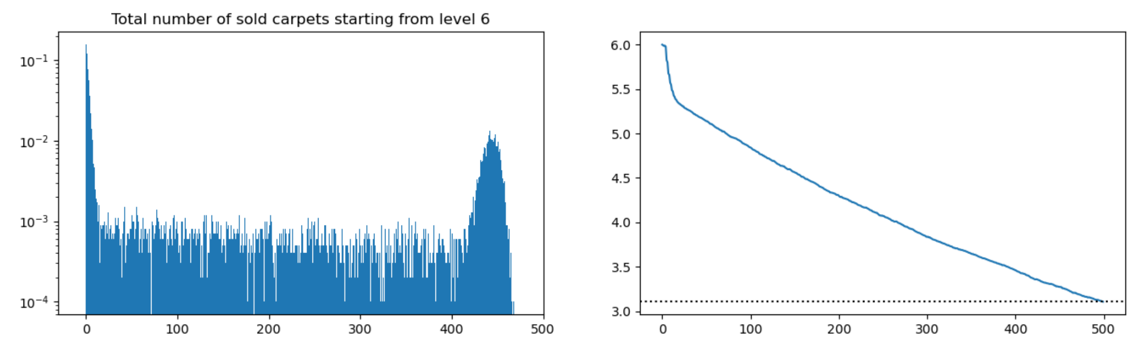
[1]:
```
import numpy as np
import matplotlib.pyplot as plt

state_vector = np.array([0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9])
states = np.array([2, 3, 4, 5, 6, 7, 8, 9, 10])
t = 500
N = 10000
tn = 5

m = np.empty(len(states))
```

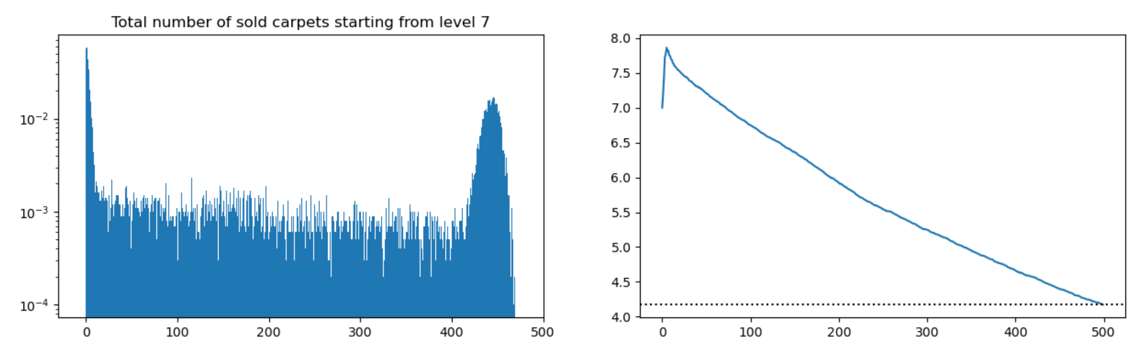
```python
avg_chain = np.empty((len(states), t))

for state in range(len(states)):
    sc = np.zeros(N)
    chain = np.empty((N, t))

    for i in range(N):
        s = states[state]
        for j in range(t):
            chain[i, j] = s
            if (np.random.random() < state_vector[s - 1]):
                sc[i] += 1
                if (s != 10):
                    s += 1
            else:
                if (s != 1):
                    s -= 1

    m[state] = np.floor(np.mean(sc))

    plt.figure(figsize=(15, 4))
    plt.subplot(121)
    h = plt.hist(sc, bins=int(np.max(sc) - np.min(sc)), density="True")[0]
    plt.yscale('log')
    plt.xlim(-30, 500)
    plt.title("Total number of sold carpets starting from level " + str(state + 2))

    plt.subplot(122)
    if (state == 0):
        plt.title("Average chains")

    for j in range(t):
        avg_chain[state, j] = np.mean(chain[:, j])

    plt.plot(range(t), avg_chain[state, :])
    plt.axhline(y=avg_chain[state, t - 1], linestyle=":", linewidth=1.5, color="BLACK")
    plt.show()

plt.figure(figsize=(15, 4))
plt.subplot(121)
plt.scatter(states, m)
plt.xlabel("Starting level")
plt.ylabel("Average number of carpets sold")
plt.subplot(122)
```

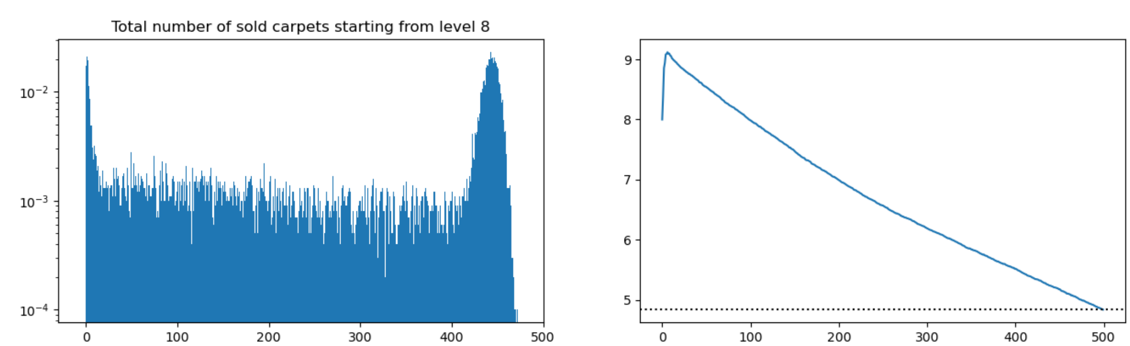
```python
avgg_chain = np.empty(t)
for j in range(t):
    avgg_chain[j] = np.mean(avg_chain[:, j])

plt.plot(range(t), avgg_chain)
plt.axhline(y=avgg_chain[t - 1], linestyle=":", linewidth=1.5, color="BLACK")
plt.show()
```

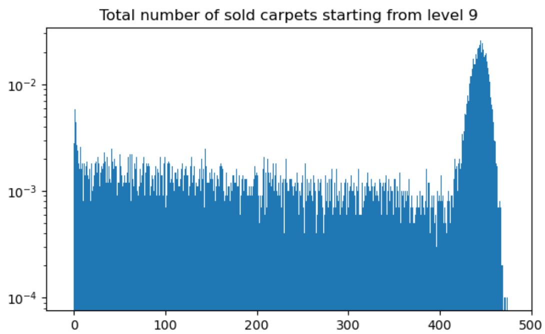


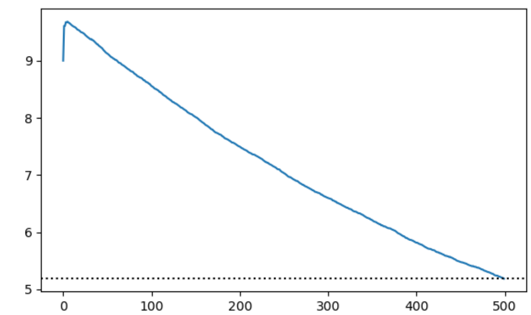

Total number of sold carpets starting from level 5

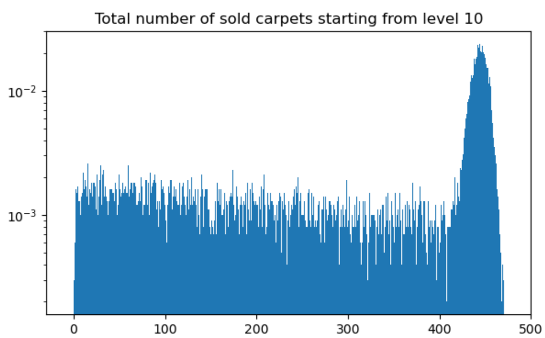

Total number of sold carpets starting from level 6

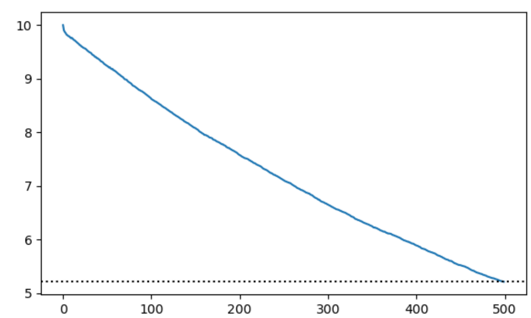

Total number of sold carpets starting from level 7

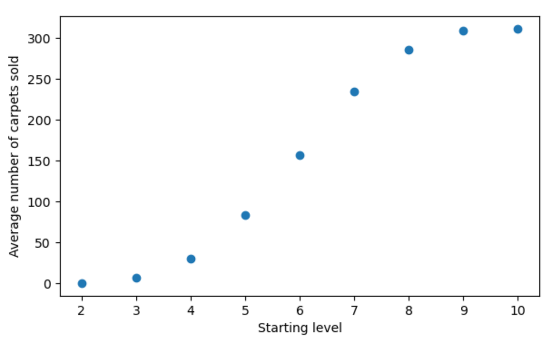

Total number of sold carpets starting from level 8

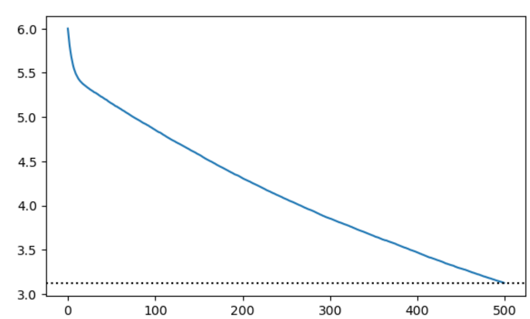








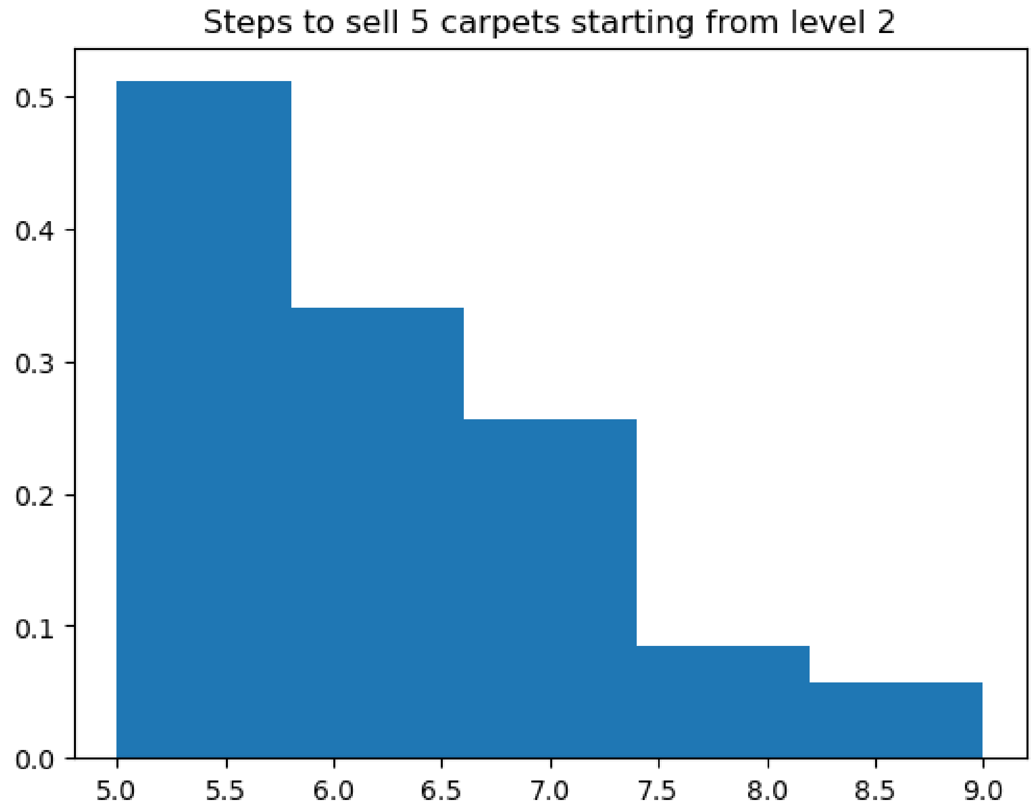


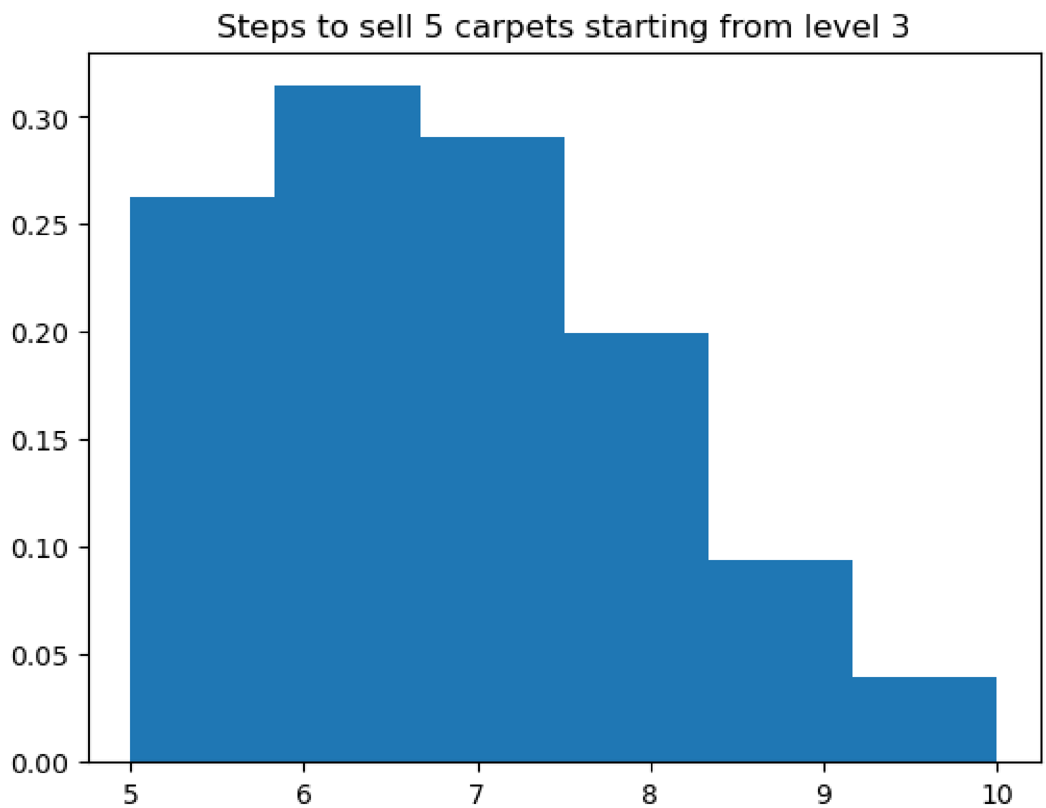


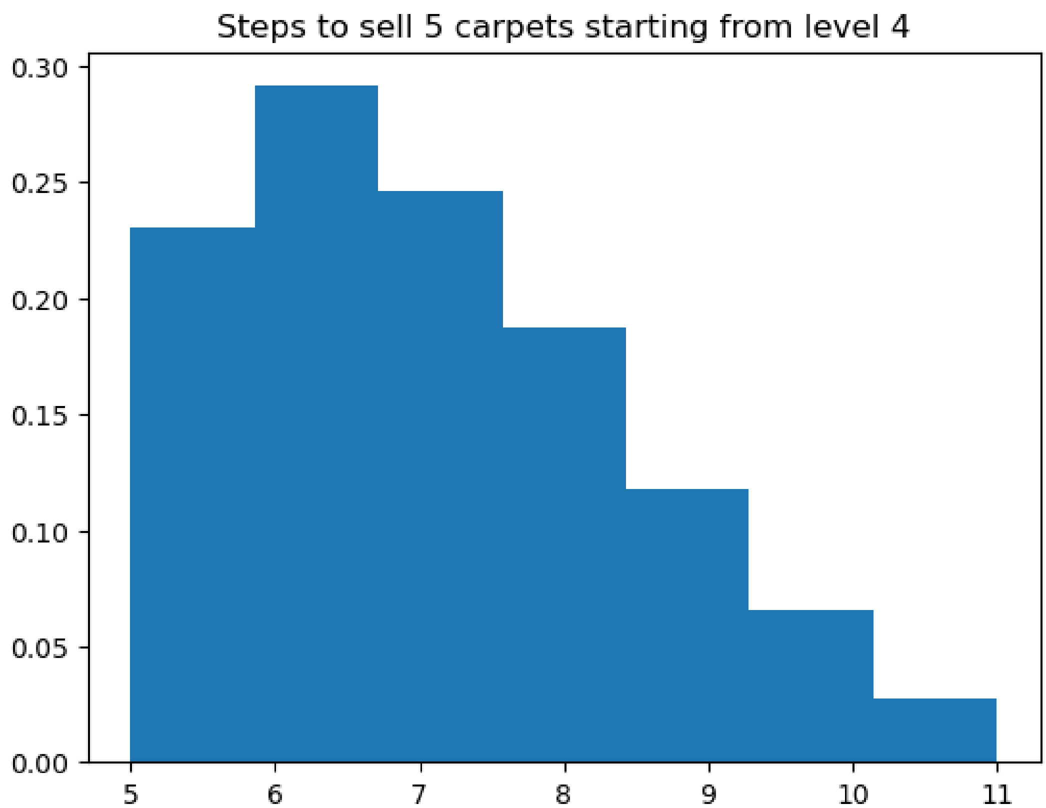
1.1.3 d, e, and f)

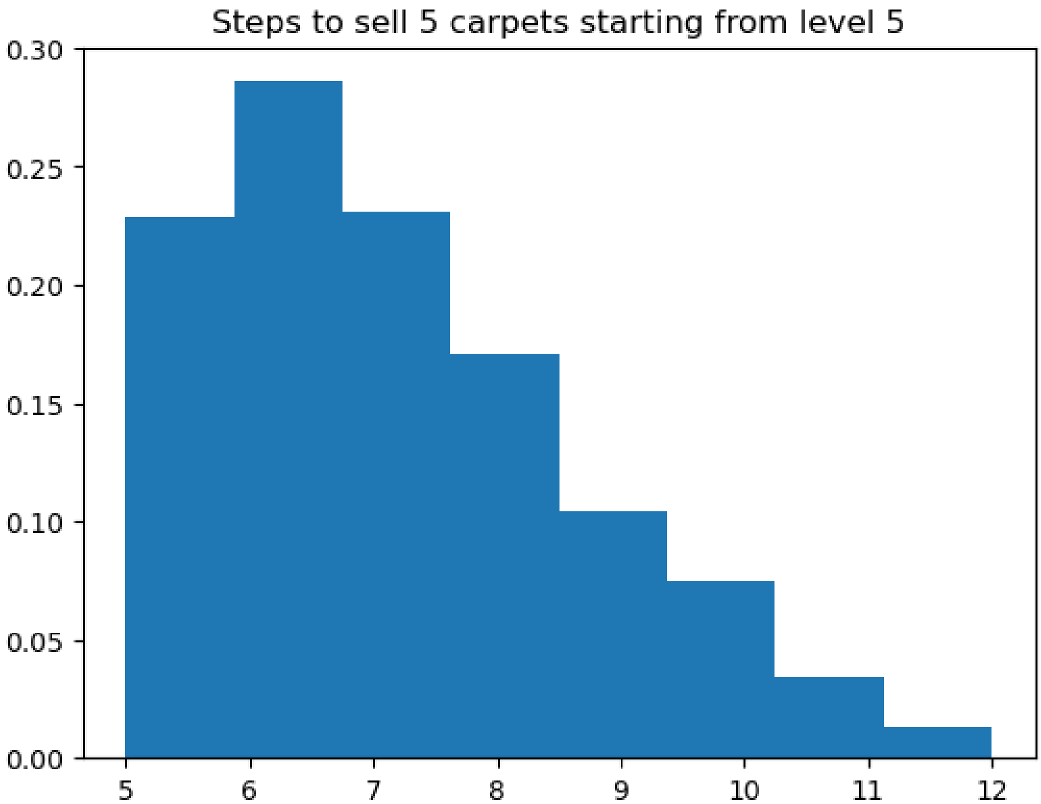
[6]:
```
m = np.empty(len(states))
p = np.empty(len(states))

for state in range(len(states)):
    sc = np.zeros(N)
```

```
    sn = np.zeros(N)

    for i in range(N):
        s = states[state]

        for j in range(t):
            if np.random.random() < state_vector[s - 1]:
                sc[i] += 1

                if sc[i] == tn:
                    sn[i] = j + 1
                    break

                if s != 10:
                    s += 1
            else:
                if s != 1:
                    s -= 1

    m[state] = np.mean(sn[np.where(sn != 0)])
    p[state] = (1 - len(np.where(sn != 0)[0]) / len(sn)) * 100

    h = plt.hist(sn[np.where(sn != 0)], bins=1 + int(np.max(sn) - np.min(sn[np.
↪where(sn != 0)])), density=True)[0]
    plt.title("Steps to sell " + str(tn) + " carpets starting from level " +␣
↪str(state + 2))
    plt.show()

plt.figure(figsize=(16, 4))
plt.subplot(121)
plt.scatter(states, m)
plt.xlabel("Starting level")
plt.ylabel("Average # steps")
plt.subplot(122)
plt.scatter(states, p)
plt.xlabel("Starting level")
plt.ylabel("Probability")
plt.show()
```

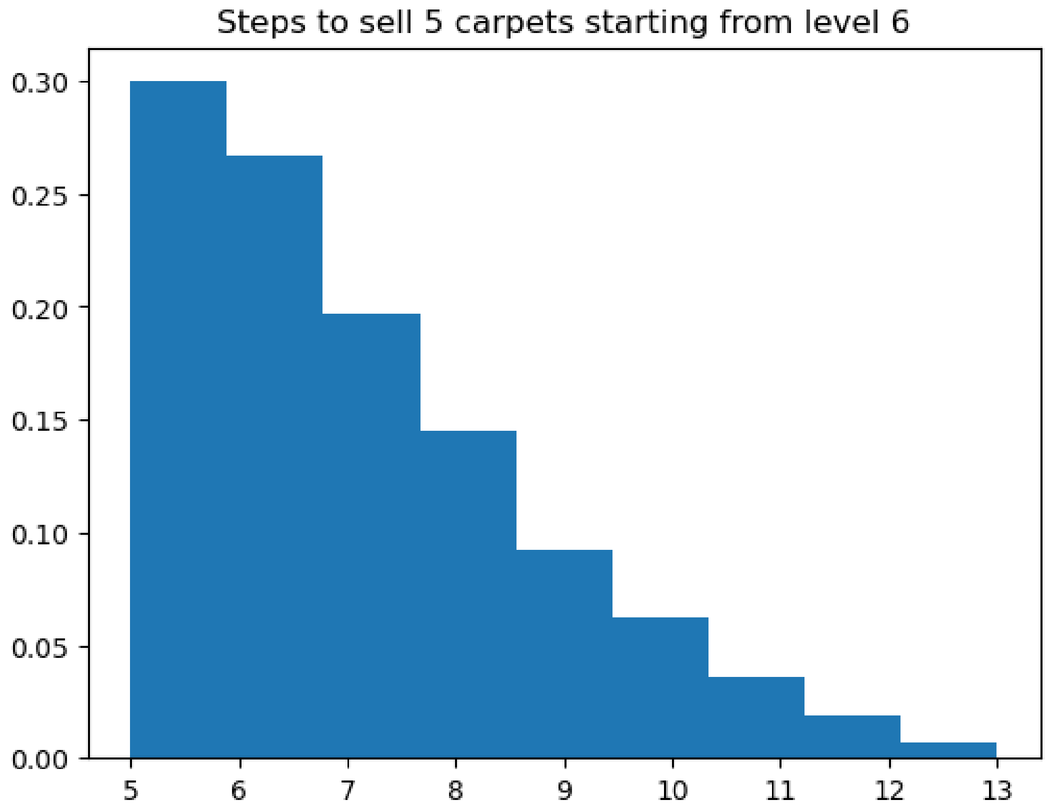

Steps to sell 5 carpets starting from level 2
0.5
0.4
0.3
0.2
0.1
0.0
5.0
5.5
6.0
6.5
7.0
7.5
8.0
8.5
9.0

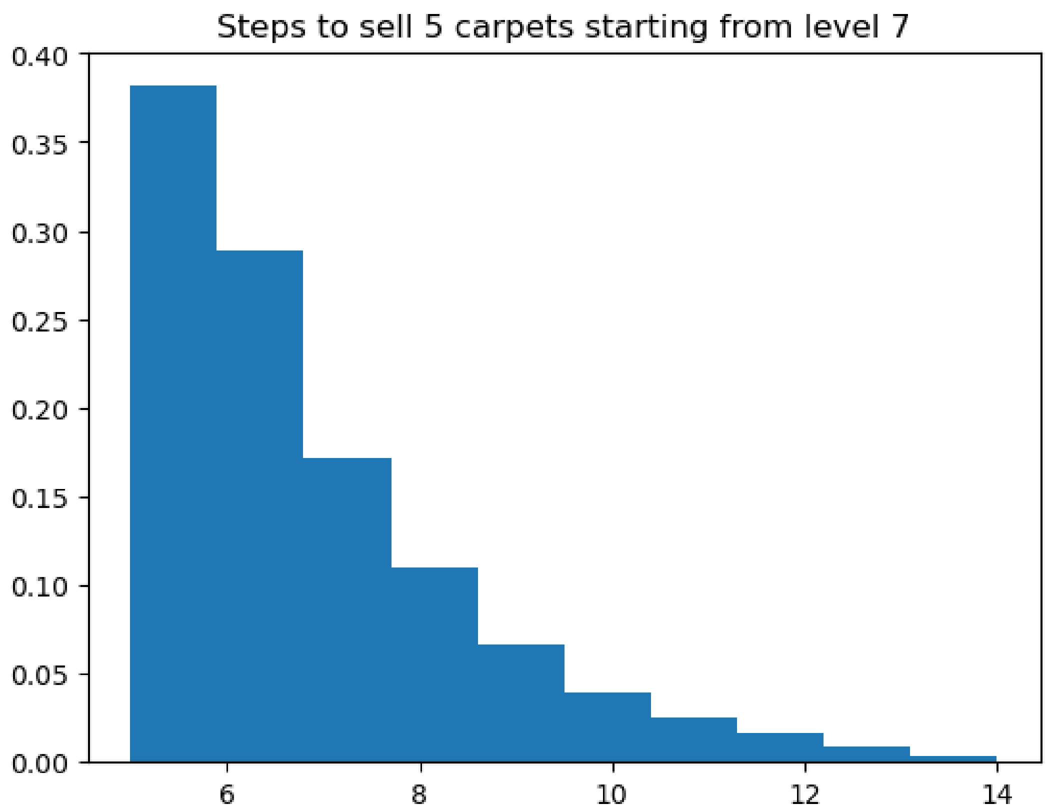

Steps to sell 5 carpets starting from level 3
0.30
0.25
0.20
0.15
0.10
0.05
0.00
5
6
7
8
9
10

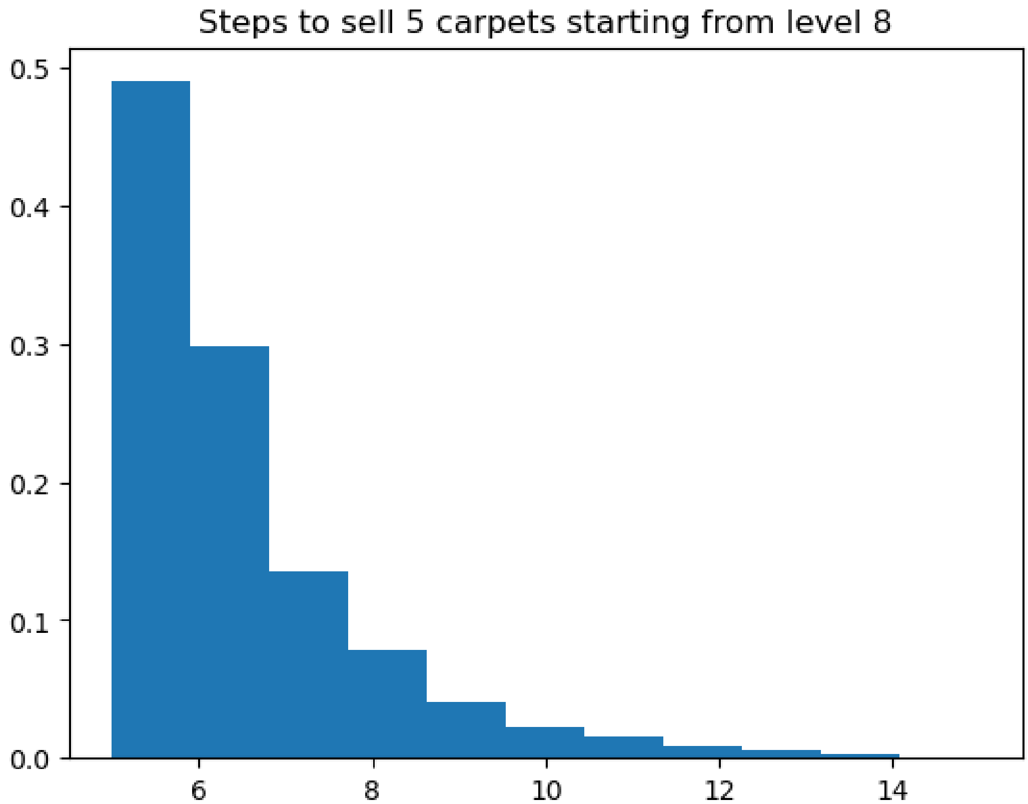

Steps to sell 5 carpets starting from level 4
0.30
0.25
0.20
0.15
0.10
0.05
0.00
5
6
7
8
9
10
11

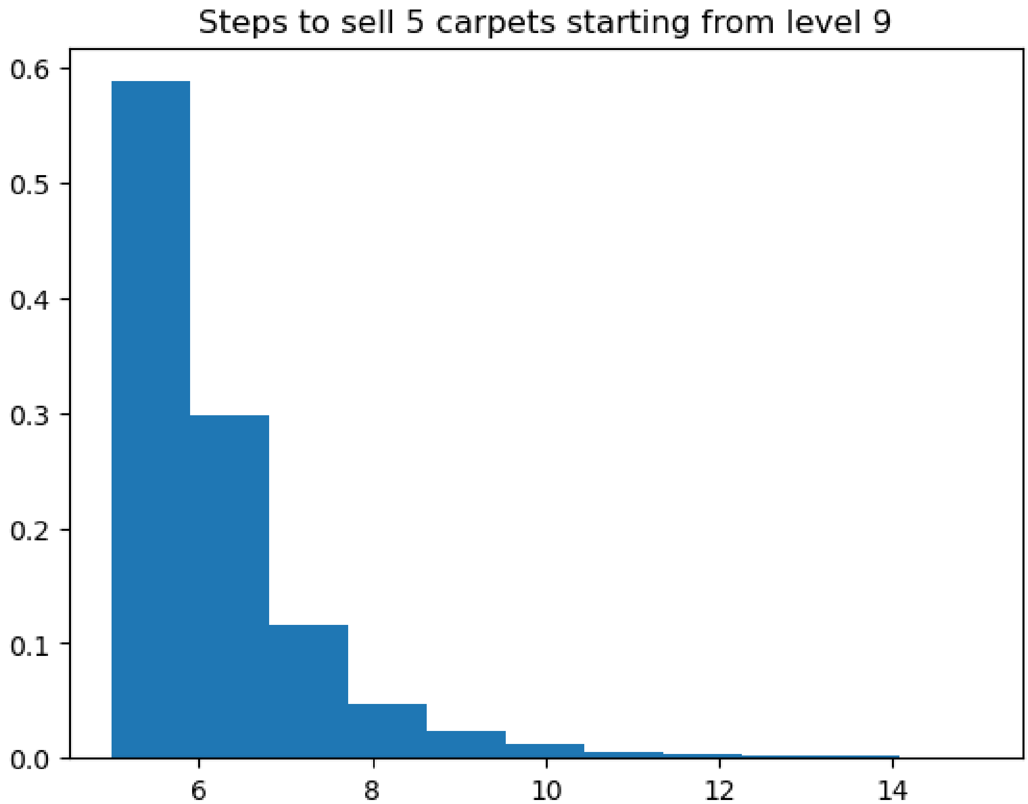


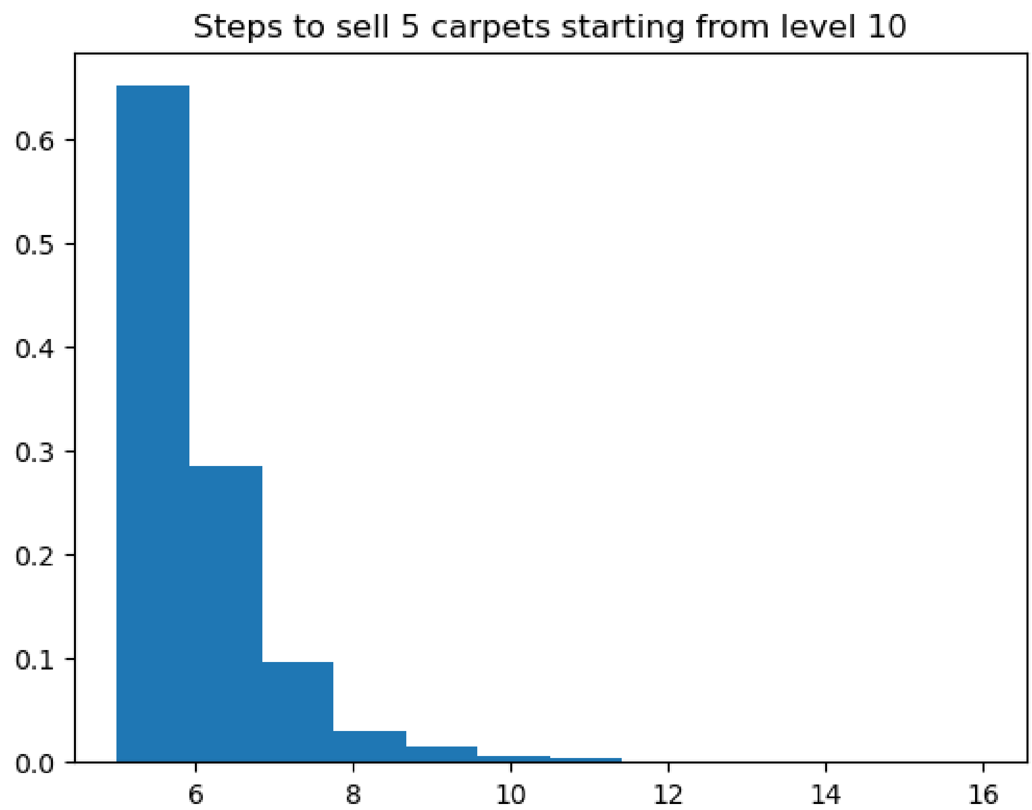
Steps to sell 5 carpets starting from level 5
0.30
0.25
0.20
0.15
0.10
0.05
0.00
5
6
7
8
9
10
11
12

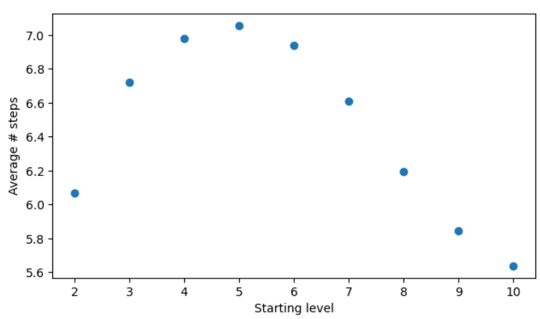

Steps to sell 5 carpets starting from level 6
0.30
0.25
0.20
0.15
0.10
0.05
0.00
5
6
7
8
9
10
11
12
13

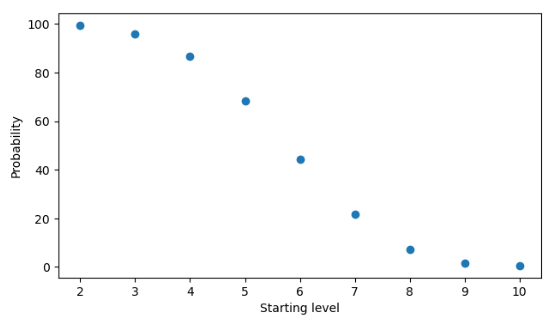

Steps to sell 5 carpets starting from level 7
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
6
8
10
12
14


Steps to sell 5 carpets starting from level 8
0.5
0.4
0.3
0.2
0.1
0.0
6
8
10
12
14


Steps to sell 5 carpets starting from level 9
0.6
0.5
0.4
0.3
0.2
0.1
0.0
6
8
10
12
14







1.1.4 g)

When the initial excitement levels are sufficiently high, the probability mass functions (PMFs) tend to exhibit a clear and distinct mathematical pattern known as a Geometric Distribution. This distribution closely resembles a discrete exponential distribution. This observation arises because Markov chains possess a property known as the Markov Property, denoted as $P(S_{t+1}|S_t) = P(S_{t+1}|S_0, \ldots, S_t)$, which essentially characterizes the system as "memoryless." This characteristic is analogous to the memoryless property of the exponential distribution, a well-known mathematical concept.

1.2 Exercise 2

1.2.1 a)

Simulate the phenomenon by sampling the waiting times between splits:

[12]:
```
ts_1 = []
N0 = 10000
l = 0.2
N = 100

def inv_cdf(p, l):
    return -math.log(1 - p) / l

for n in range(N):
    k = 0
    t = 0
    while k < N0 / 2:
        r = np.random.random()
        nt = inv_cdf(r, (N0 - k) * l)
        t += nt
        k += 1
    ts_1.append(t)
```

Simulate in discrete time steps:

[13]:
```
N0 = 10000
ts_2 = []
l = 0.2
sequences = 1000
dt = 0.01

def poisson_sample(l):
    k = 0
    p = 1
    L = np.exp(-l)
    while p > L:
        k += 1
        p *= np.random.random()
    return k-1

for seq in range(sequences):
    k = 0
    step = 0
    while k < 0.5*N0:
        t = dt*step
        sample = poisson_sample((N0-k)*l*dt)
        k += sample
        step += 1
```

```
    ts_2.append(step * dt)
```

1.2.2 b)

[14]:
```
mean = sum(ts_1)/len(ts_1)
var = sum([(t-mean)**2 for t in ts_1])/len(ts_1)
print("Time (mean, variance): ", (mean, var))
```

```
Time (mean, variance):  (3.4658571603686936, 0.002686840270915175)
```

[15]:
```
mean = sum(ts_2)/len(ts_2)
var = sum([(t-mean)**2 for t in ts_2])/len(ts_2)
print("Time (mean, variance): ", (mean, var))
```

```
Time (mean, variance):  (3.462589999999998, 0.0024433919000000116)
```