EE325 HW2

Aditya Anand, Rishab Rao, Aakash Gore

September 7, 2022

1 Question 1 and 2

Q1

Recall the ‘capture-release-recapture’ problem: Catch m fish, mark them and release them back into the lake. Allow the fish to mix well and then you catch m fish. Of these p are those that were marked before. Assume that the actual fish population in the lakes is n and has not changed between the catches. Let Pm,p(n) be the probability of the event (for a fixed p recatches out of m) coming from n fish in the lake. Generate a plot for Pm,p(n) as a function of n for the following values of m and p : m = 100 and p = 10, 20, 50, 75. For each of these p, use the plots to estimate (educated guess) the actual value of n, i.e., what is the best guess for n if m = 100 and you catch p of the marked fish after mixing them up. Call these four estimates ˆn1, . . . , nˆ4. You define your notion of “best guess.” Do not search, THINK!

THE BEST GUESS FOR ’N’ ACCORDING TO US WILL BE THE ONE AT WHICH Pr(N) IS MAXIMUM.

Q2

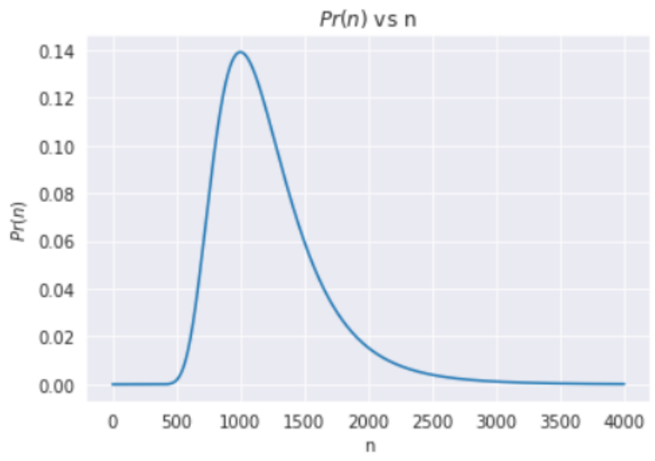
Let us now simulate this process as follows. Let n = 2000. Initialise an array of 2000 integers to 0. Select 100 random locations and mark them as 1. This corresponds to catching m = 100 random fish. Now randomly select 100 locations in the array to check for the marked fish and note the number of 1s in these 100. This corresponds to the p. Use the best guess algorithm from the previous part to determine ˆn, the estimate for the number of fish. Let e = ˆnin = ˆni1000 be error in the estimate. Repeat the experiment 500 times and collect 500 samples of the error and show a scatter plot of the errrors. Find the sample mean of the error and the sample variance of the error.

1.1 Python code

```
import matplotlib.pyplot as plt
import numpy as np
import operator as op                #for calculating nCr
```

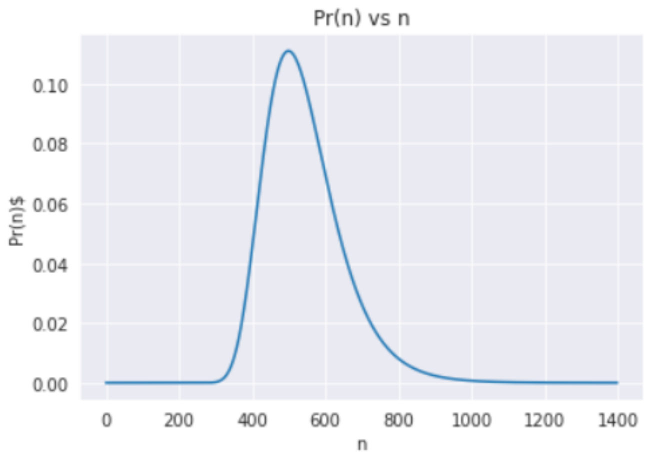
```
from functools import reduce
import math
import random

import seaborn as sns
sns.set_style("darkgrid")

def ncr(n, r):
    r = min(r, n-r)
    numer = reduce(op.mul, range(n, n-r, -1), 1)
    denom = reduce(op.mul, range(1, r+1), 1)
    return numer // denom


# making a function named Pr which will return (mCn*[(n-m)C(m-p)]/nCm) prob. of n
def Pr(m, p, n):
    if(m > n or p > m or n - 2*m + p < 0 or n < 0):  # Conditions due to nCr
        return 0
    else:
        return (ncr(m,p)*ncr(n-m,m-p)/ncr(n,m))

na = np.arange(0, 4000)
pa = np.array([Pr(100, 10, n) for n in na])

plt.xlabel('n')
plt.ylabel('Pr(n)')
plt.title('Pr(n) vs n')
plt.plot(na, pa)
plt.show()
A = max(pa)
N = np.argmax(pa)
print("Guess for N is:",N )
print("Probability of N is:", A)

nb = np.arange(0, 1400)
pb = np.array([Pr(100, 20, n) for n in nb])            #n>180 will have non zero prob

plt.xlabel('n')
plt.ylabel('Pr(n)')
plt.title('Pr(n) vs n')
plt.plot(nb, pb)
plt.show()
A = max(pb)
N = np.argmax(pb)
print("Guess for N is:",N )
print("Probability of N is:", A)
```

```
nc = np.arange(0, 300)
pc = np.array([Pr(100, 50, n) for n in nc])       #N>150 will have non zero prob

plt.xlabel('n')
plt.ylabel('P(n)')
plt.title('P(n) vs n')
plt.plot(nc, pc)
plt.show()
A = max(pc)
N = np.argmax(pc)
print("Guess for N is:",N )
print("Probability of N is:", A)

nd = np.arange(0, 200)
pd = np.array([Pr(100, 75, n) for n in nd])       #n>125 will have non zero prob

plt.xlabel('n')
plt.ylabel('P(n)')
plt.title('P(n) vs n')
plt.plot(nd, pd)
plt.show()
A = max(pd)
N = np.argmax(pd)
print("Guess for N is:",N )                                     #m^2/p
print("Probability of N is:", A)

s = [0]*2000        #m is fixed p is varying and n is also fixed now we
i = [0]*2000        #have to use previous algo. to find new n_hat and find error
j=0
while j < 2000:     #made an array to store index of s and then select
  i[j] = j          #100 random index out of 0-1999 and then make that s[i]=1
  j += 1

j=0
g=0
k=1000
x=np.arange(0,500,1)
storeE = [0]*500    # 500 p's will be generated for which we will get
                    # 500 n's WILL BE THERE SO 500 ERRORS WILL BE STORED HERE.
while g < 500:
  while j < 100:
    r = random.choice(i)
    s[r]=1
    j += 1
  j=0
  p=0


  while j < 100:       #selecting 100 out of 0-1999 indexes
                       #and checking how many are one
    r = random.choice(i)    #from the outputs we can see that out of
    if s[r]==1:             #100 most of the time less than 10 are 1.
      p=p+1
    j += 1

  ne = np.arange(0, 2000)
  pe = np.array([Pr(100, p, n) for n in ne])
  A = max(pe)
  N = np.argmax(pe)

  storeE[g] = N-k
  g +=1

plt.scatter(x,storeE)
plt.xlabel('No. of itterations')
plt.ylabel('Error')
plt.title('Scatter Plot of 500 errors')
print("Mean of errors:" ,np.mean(storeE))
print("Varience of errors:" ,np.var(storeE))
```



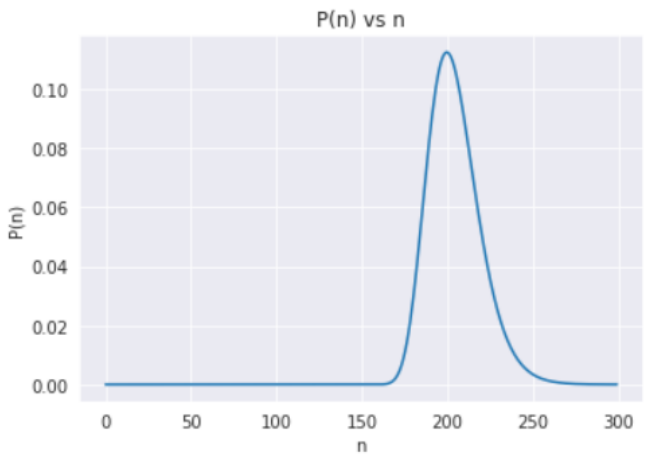
```
Guess for N is: 999
Probability of N is: 0.13898526767704464
```

Q1 Plot when p=10



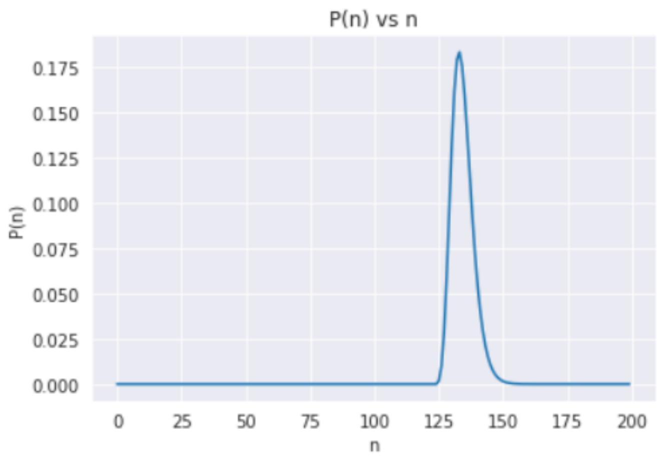
```
Guess for N is: 499
Probability of N is: 0.11099673235151158
```

Q1 Plot when p=20



```
Guess for N is: 199
Probability of N is: 0.11241557570404212
```

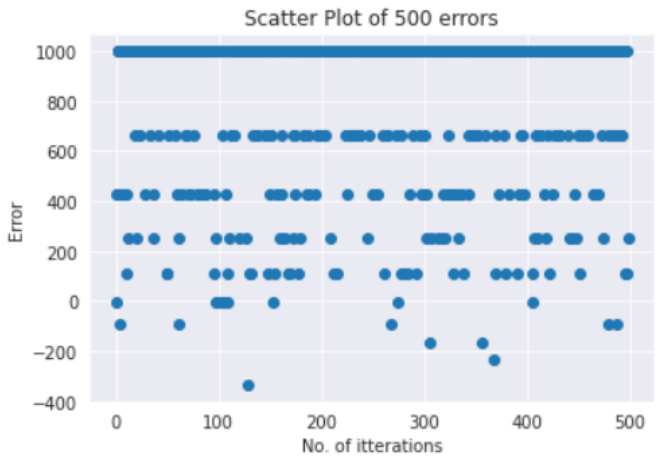
Q1 Plot when p=50



```
Guess for N is: 133
Probability of N is: 0.18346999713931525
```

Q1 Plot when p=75

```
Mean: 767.298
Varience: 110107.021196
```



Q2 Error Scatter plot

2 Question 3 and 4

Q3

Consider the following discrete time system. Packets arrive randomly at at router to be transmitted on a link. The time between successive packets are independent geometric random variables with parameter . Packet transmission times are also random and have a geometric distribution with parameter µ. Only one packet can be transmitted at any time and packets that arrive during the transmission of a previously arrived packet wait in buffer memory. There is infinte memory and can accommodate any number of packets. This can be simulated as follows. At the beginning of each second, if there is a packet in the buffer, it leaves with probability µ. And a new packet is added to the queue with probability . Simulate this queue for 1,000,000 time steps. For n = 0, . . . 50, plot p(n), the fraction of time that there are are n packets in the queue. Also find the time average of the number of packets in the memory. Use = 0.3 and µ = 0.4.

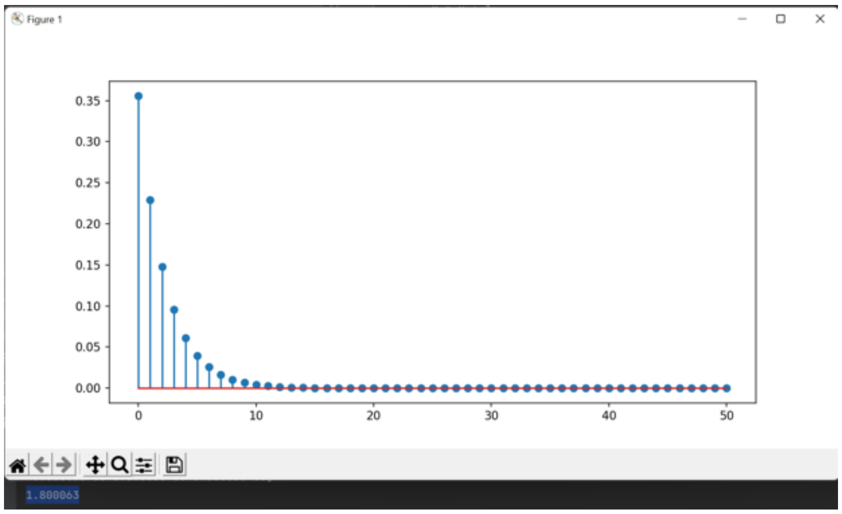
AVG. CAME OUT TO BE - 1.800063

Q4

Now extend the program from the previous part to simulate 10,000 queues in simultaneously parallel. When you stop the simulation after 100,000 time steps you have 10,000 values for the number of packets in the system. Use this data to plot p(n) the fraction of queues that have n packets in the system and calculate the sample average from this 10,000 samples.

2.1 Python code Q3

```
import matplotlib.pyplot as plt;
import numpy as np;
plt.rcParams['figure.figsize']= [10,5]

arr = np.zeros(51)


x = 0  # number of packets in queue

for t in range(0, 1000000):
    rec = np.random.random()
    if rec <= 0.3:
        x = x + 1
    tran = np.random.random()
    if tran <= 0.4 and x > 0:
        x = x - 1
    #if x <= 50:
    arr[x] = arr[x] + 1

s=0
print(arr)
for i in range(0,51):
    s = s + i*arr[i]
print(s/1000000)

n= np.arange(0,51)
plt.stem(n, arr/1000000)
        # k=0;



#plt.stem(n,k)
plt.show()
```

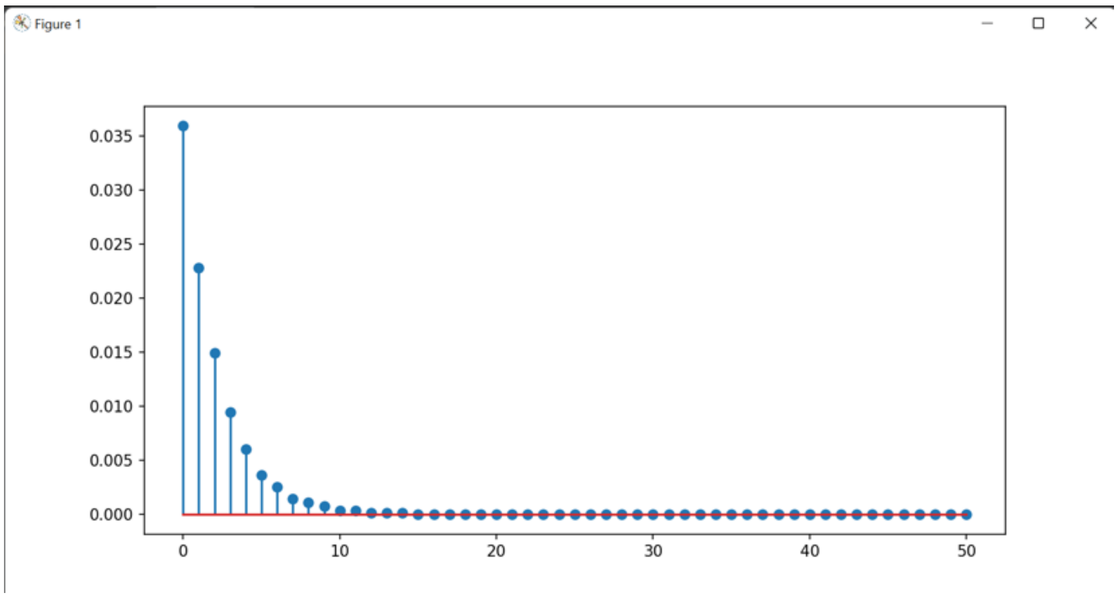


```
1.800063
```

2.2 Python code Q4

```
    import matplotlib.pyplot as plt;
    import numpy as np;
    plt.rcParams['figure.figsize']= [10,5]

arr = np.zeros(51)


x = 0  # number of packets in queue

for i in range(0, 10000):
    for t in range(0, 100000):
        rec = np.random.random()
        if rec <= 0.3:
            x = x + 1
        tran = np.random.random()
        if tran <= 0.4 and x > 0:
            x = x - 1
        # if x <= 50:
        # arr[x] = arr[x] + 1
    arr[x] = arr[x] + 1
    x = 0



#print(arr)
```

```
n= np.arange(0,51)
plt.stem(n, arr/100000)
        # k=0;



#plt.stem(n,k)
plt.show()
```



Q4 Plot