Project 5

Qidi Han

March 11, 2020

Question 1

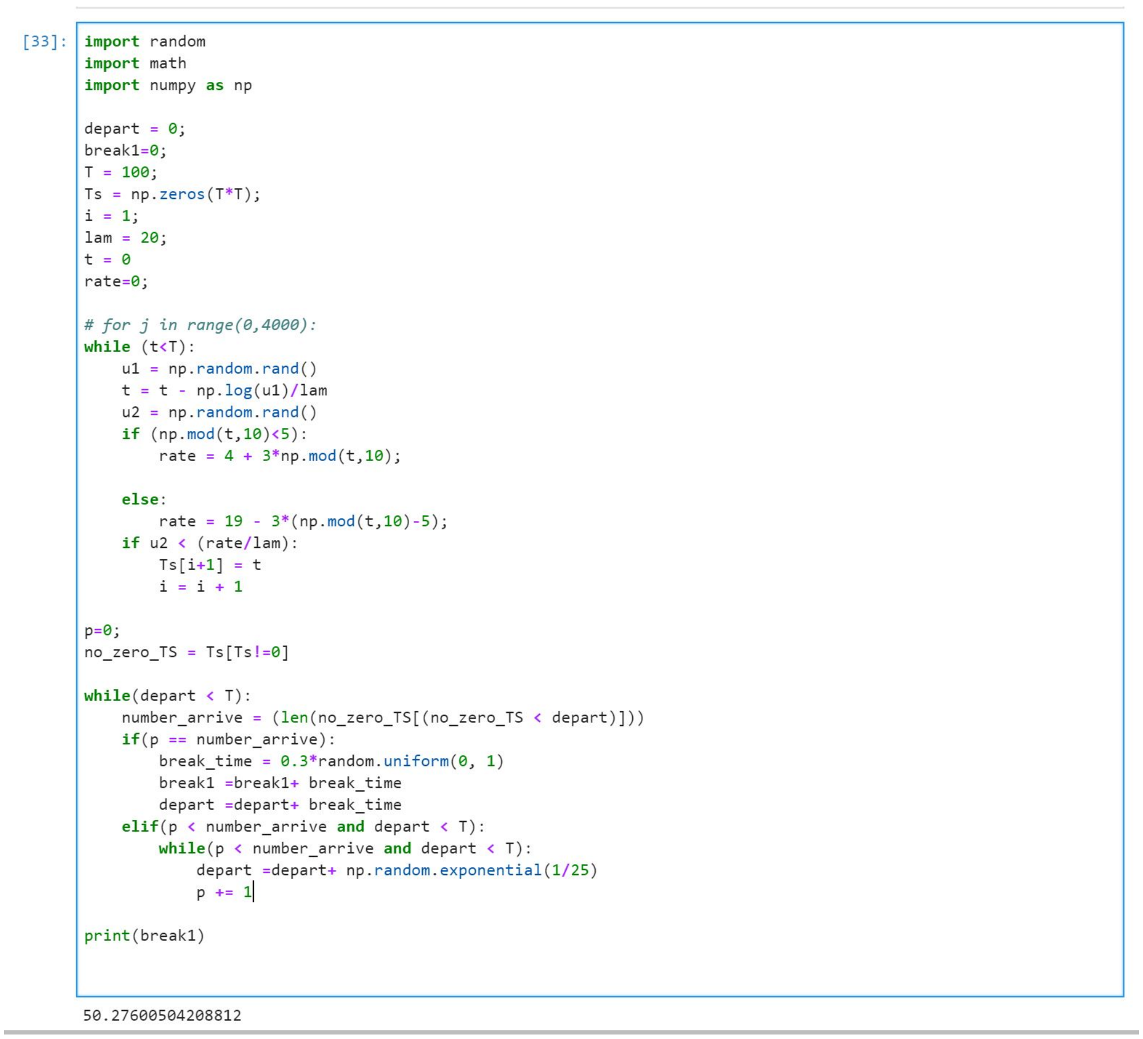
Suppose that jobs arrive at a single-server queue system according to a nonhomogeneous Poisson process. The arrival rate is initially 4 jobs per hour and increases steadily (linearly) until it hits 19 jobs per hour after 5 hours. The rate then decreases steadily until it returns to 4 jobs per hour after another 5 hours. The rate repeats indefinitely in this fashion: r(t+10)=r(t) Suppose that the service-time distribution is exponential with rate 25 jobs per hour. Suppose also that whenever the server completes a job and finds no jobs waiting it goes on break for a time that is uniformly distributed on (0, 0.3). The server goes on another break if upon returning from break there are still no jobs waiting. Estimate the expected amount of time that the server is on break in the first 100 hours of operation.

Theory

There are two parts of the this question, the first part is using the algorithm that proved in the class. there are 6 steps of this algorithm, I will show the algorithm in the equation part. There is a main idea of our question, the period between 4 and 19 jobs. When the time=0, it's at 4 jobs, when the time=5, it's at 19jobs. and when the time = 10, it's back to the 4 jobs. It's a periodic. From this periodic, we can calculate the line for both situation. The first situation is the line equal y=4+3x and the second line is y=19-3x. we will use those two equation to determine the time period for our question. The second part is calculate the break time during 100 hours. The service time use the exponential with rate 25 job per hour. We will set a server and waiting time, when the server completes a job, and finds no jobs waiting for it, and it will go break for a time.

Algorithm 1: explanation of algorithm

```
/* algorithm for                                                  */
1. let t=s 2. generate uniform distribution number u1
3. let t=t-1/lamda *log(u1)
4. generate uniform distribution number u2 5. if u<= lamda[t]/lamda,set Ts=t and
stp 6. else go step 1
```



```
[33]: import random
import math
import numpy as np

depart = 0;
break1=0;
T = 100;
Ts = np.zeros(T*T);
i = 1;
lam = 20;
t = 0
rate=0;

# for j in range(0,4000):
while (t<T):
    u1 = np.random.rand()
    t = t - np.log(u1)/lam
    u2 = np.random.rand()
    if (np.mod(t,10)<5):
        rate = 4 + 3*np.mod(t,10);

    else:
        rate = 19 - 3*(np.mod(t,10)-5);
    if u2 < (rate/lam):
        Ts[i+1] = t
        i = i + 1

p=0;
no_zero_TS = Ts[Ts!=0]

while(depart < T):
    number_arrive = (len(no_zero_TS[(no_zero_TS < depart)]))
    if(p == number_arrive):
        break_time = 0.3*random.uniform(0, 1)
        break1 =break1+ break_time
        depart =depart+ break_time
    elif(p < number_arrive and depart < T):
        while(p < number_arrive and depart < T):
            depart =depart+ np.random.exponential(1/25)
            p += 1

print(break1)

50.27600504208812
```

Figure 1: code and result of this question

Explanation

For the result of our question, the expected amount of time that the server is on break is around 45 to 50, This is not a very good time for first 100 hours of operation, because half of the time is in the break. Maybe the uniform distribution cause this issue, or maybe the the interval between each time arrive is too long, there are two interaction jobs during a long times.

Question 2

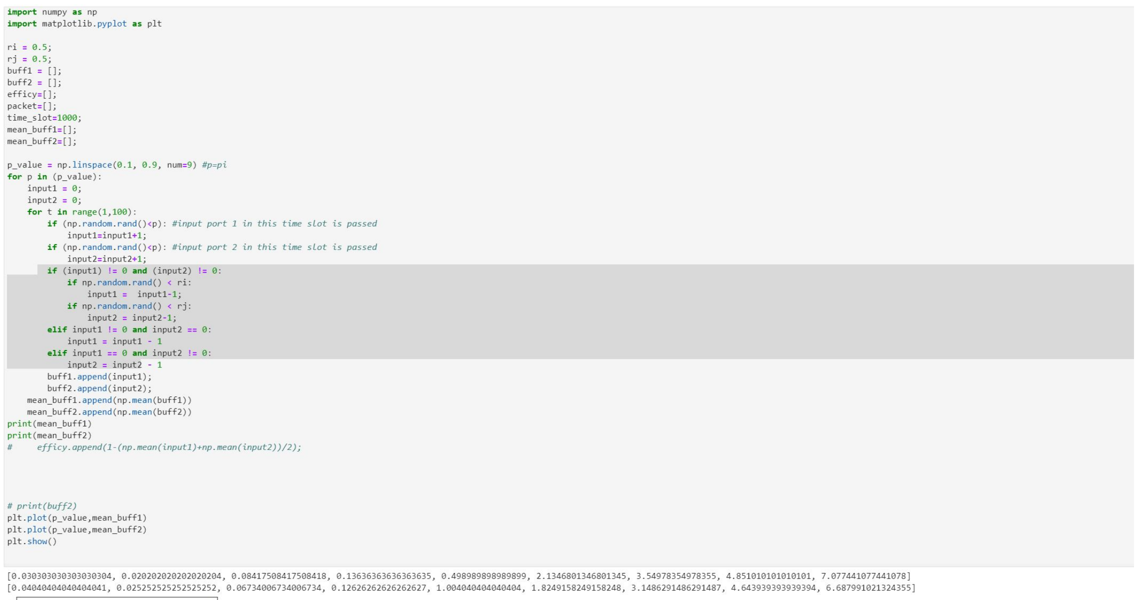
In class we considered the 2x2 HOL-blocking switch performance under the heavy-load assumption, i.e. there was always a packet at each input.A more realistic model includes modeling the buffer for each input. Assume the probability that a packet arrives at input port i in a time slot is pi=p (a constant) for each input. Assume also that the probability that a packet arriving at input i hould be switched to output j is rij Define the system state to be the number of packets in the buffer at each input in the middle of time slot k and the desired output port of the packet at the HOL of each input, i.e. decide the desired output when the packet moves to the HOL-slot. Assume that packets arrive to the input buffers at the end of a time slot. If there is a packet in the buffer at the beginning of a time slot the switch attempts. If the delivery attempt succeeds the switch removes the packet from the input queue at the end of the time slot. If the delivery attempt fails then the packet stays at the head of the line and the switch will attempt delivery in the next time slot.

Theory

For the 2*2 HOL-blocking switch performance problem, there are also two parts of this question. The first part is consider the probability that packet arrives at input port 1 and port 2. I will use 0.1 to 0.9 as my probability range for each packet arrives. Then I will use probability of switched to determine which packet in the buffer will be send first. In the first question, I will use the rij=0.5, the probability of each switch is 0.5. The second question, I will change the probability of each switch to 0.75 and 0.25. Each question, I will store the number of packet in the buffer 1 and buffer 2. Then I will calculate the mean of each buffer. For each different probability range for each packet arrives, I will plot a figure to show the final result. And I will plot the histogram for number of packets processed by the switch per time slot. There are only four case for the switch to process in this question. There are 0, 1 and 2. Finally, I will calculate the 95 confidence interval for both case.

Algorithm 2: Code and formula for empirical distribution.

```
/* There are only three case, the first case is when both buffer size is not 0, the
   second two case is when one buffer size is not 0                              */
if (input1) != 0 and (input2) != 0:
if np.random.rand() < ri:
input1 = input1-1;
if np.random.rand() < rj:
input2 = input2-1;
elif input1 != 0 and input2 == 0:
input1 = input1 - 1
elif input1 == 0 and input2 != 0:
input2 = input2 - 1
```



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

ri = 0.5;
rj = 0.5;
buff1 = [];
buff2 = [];
efficy=[];
packet=[];
time_slot=1000;
mean_buff1=[];
mean_buff2=[];

p_value = np.linspace(0.1, 0.9, num=9) #p=pi
for p in (p_value):
    input1 = 0;
    input2 = 0;
    for t in range(1,100):
        if (np.random.rand()<p): #input port 1 in this time slot is passed
            input1=input1+1;
        if (np.random.rand()<p): #input port 2 in this time slot is passed
            input2=input2+1;
        if (input1) != 0 and (input2) != 0:
            if np.random.rand() < ri:
                input1 =  input1-1;
            if np.random.rand() < rj:
                input2 = input2-1;
        elif input1 != 0 and input2 == 0:
            input1 = input1 - 1
        elif input1 == 0 and input2 != 0:
            input2 = input2 - 1
        buff1.append(input1);
        buff2.append(input2);
    mean_buff1.append(np.mean(buff1))
    mean_buff2.append(np.mean(buff2))
print(mean_buff1)
print(mean_buff2)
#     efficy.append(1-(np.mean(input1)+np.mean(input2))/2);




# print(buff2)
plt.plot(p_value,mean_buff1)
plt.plot(p_value,mean_buff2)
plt.show()
```

```
[0.030303030303030304, 0.020202020202020204, 0.08417508417508418, 0.13636363636363635, 0.498989898989899, 2.1346801346801345, 3.54978354978355, 4.851010101010101, 7.077441077441078]
[0.04040404040404041, 0.025252525252525252, 0.06734006734006734, 0.12626262626262627, 1.004040404040404, 1.8249158249158248, 3.1486291486291487, 4.643939393939394, 6.687991021324355]
```

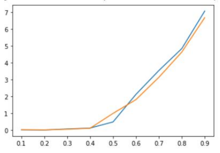
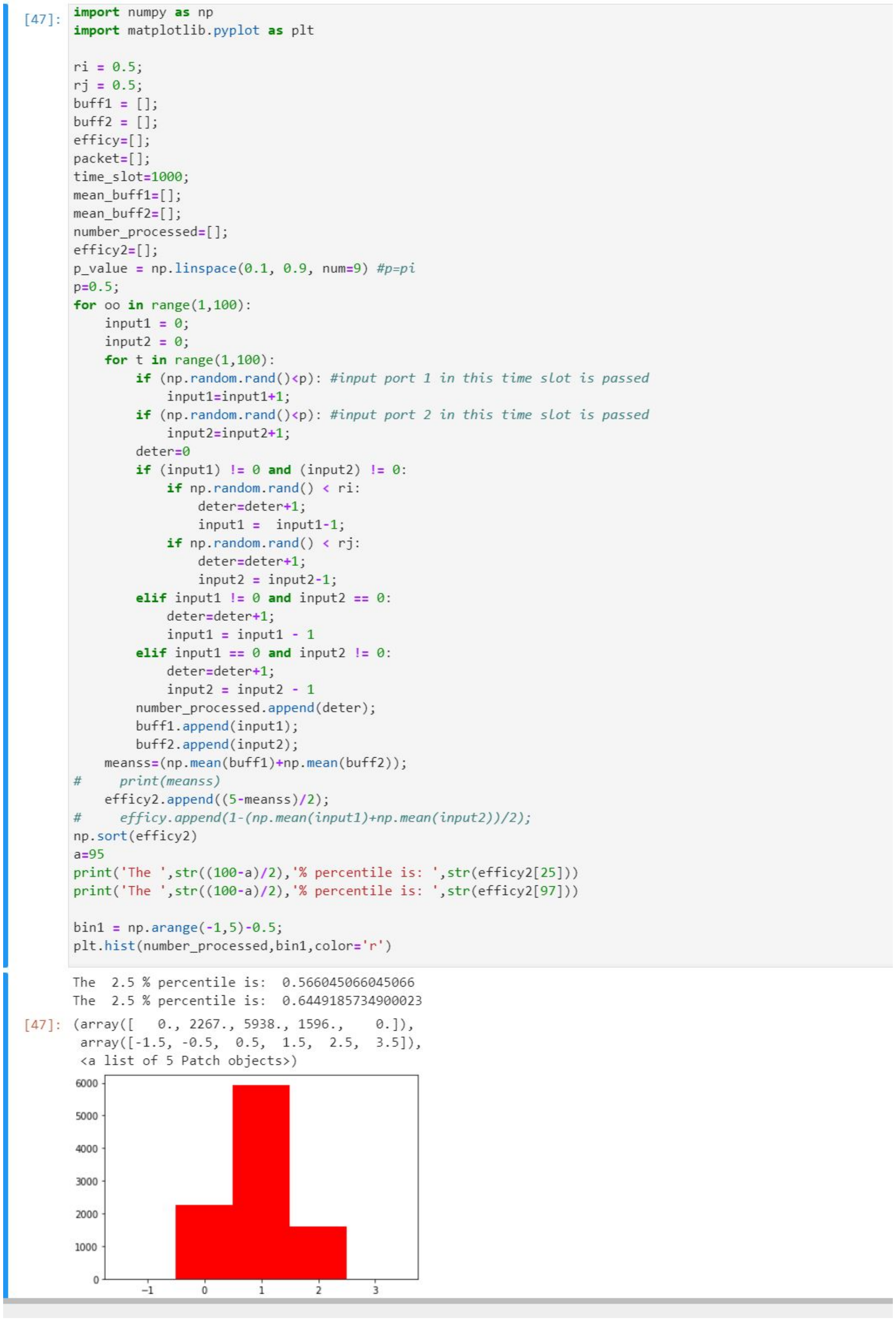


Figure 2: the code and result for the distribution and compute the mean of the number of packets in the buffer at input 1 and input 2 when r1=r2=0.5



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

      ri = 0.5;
      rj = 0.5;
      buff1 = [];
      buff2 = [];
      efficy=[];
      packet=[];
      time_slot=1000;
      mean_buff1=[];
      mean_buff2=[];
      number_processed=[];
      efficy2=[];
      p_value = np.linspace(0.1, 0.9, num=9) #p=pi
      p=0.5;
      for oo in range(1,100):
          input1 = 0;
          input2 = 0;
          for t in range(1,100):
              if (np.random.rand()<p): #input port 1 in this time slot is passed
                  input1=input1+1;
              if (np.random.rand()<p): #input port 2 in this time slot is passed
                  input2=input2+1;
              deter=0
              if (input1) != 0 and (input2) != 0:
                  if np.random.rand() < ri:
                      deter=deter+1;
                      input1 =  input1-1;
                  if np.random.rand() < rj:
                      deter=deter+1;
                      input2 = input2-1;
              elif input1 != 0 and input2 == 0:
                  deter=deter+1;
                  input1 = input1 - 1
              elif input1 == 0 and input2 != 0:
                  deter=deter+1;
                  input2 = input2 - 1
              number_processed.append(deter);
              buff1.append(input1);
              buff2.append(input2);
          meanss=(np.mean(buff1)+np.mean(buff2));
      #     print(meanss)
          efficy2.append((5-meanss)/2);
      #     efficy.append(1-(np.mean(input1)+np.mean(input2))/2);
      np.sort(efficy2)
      a=95
      print('The ',str((100-a)/2),'% percentile is: ',str(efficy2[25]))
      print('The ',str((100-a)/2),'% percentile is: ',str(efficy2[97]))

      bin1 = np.arange(-1,5)-0.5;
      plt.hist(number_processed,bin1,color='r')
```

```
      The  2.5 % percentile is:  0.560045066045066
      The  2.5 % percentile is:  0.6449185734900023
[47]: (array([   0., 2267., 5938., 1596.,    0.]),
       array([-1.5, -0.5,  0.5,  1.5,  2.5,  3.5]),
       <a list of 5 Patch objects>)
```

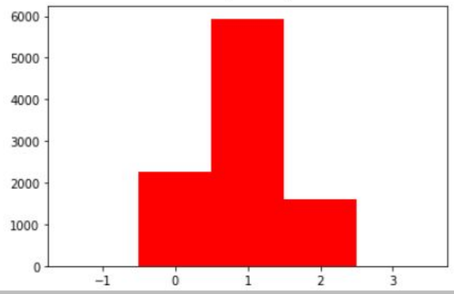
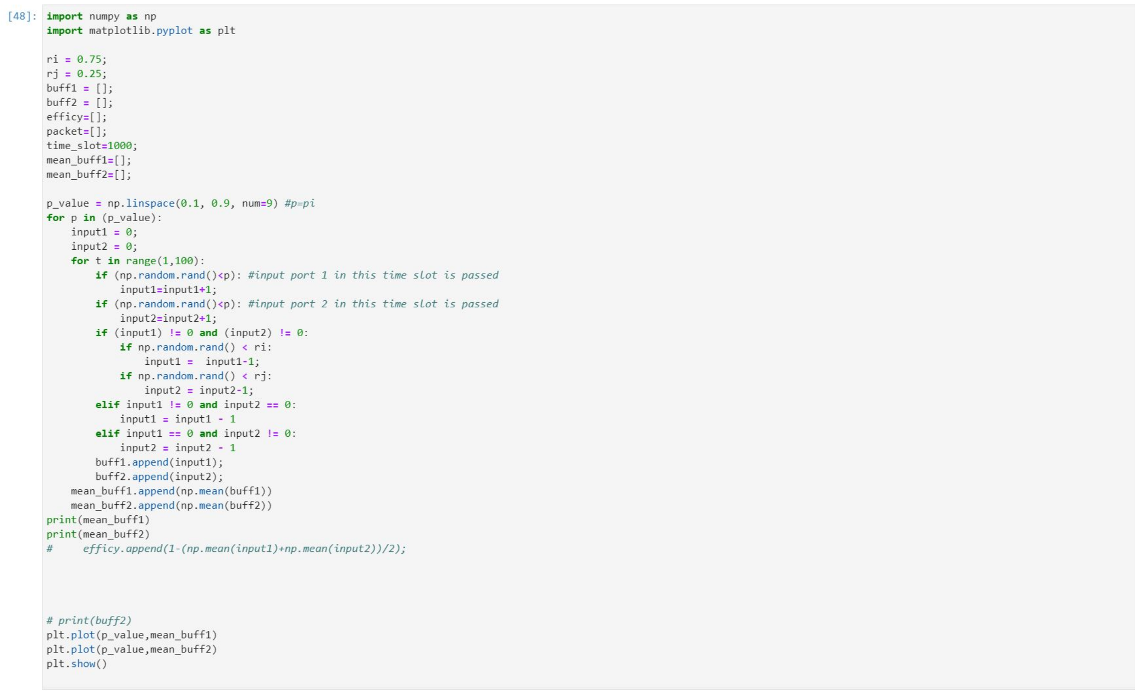


Figure 3: the code and result for the distribution and compute the mean of the number of packets processed by the switch per time slot when r1=r2=0.5



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

      ri = 0.75;
      rj = 0.25;
      buff1 = [];
      buff2 = [];
      efficy=[];
      packet=[];
      time_slot=1000;
      mean_buff1=[];
      mean_buff2=[];

      p_value = np.linspace(0.1, 0.9, num=9) #p=pi
      for p in (p_value):
          input1 = 0;
          input2 = 0;
          for t in range(1,100):
              if (np.random.rand()<p): #input port 1 in this time slot is passed
                  input1=input1+1;
              if (np.random.rand()<p): #input port 2 in this time slot is passed
                  input2=input2+1;
              if (input1) != 0 and (input2) != 0:
                  if np.random.rand() < ri:
                      input1 =  input1-1;
                  if np.random.rand() < rj:
                      input2 = input2-1;
              elif input1 != 0 and input2 == 0:
                  input1 = input1 - 1
              elif input1 == 0 and input2 != 0:
                  input2 = input2 - 1
              buff1.append(input1);
              buff2.append(input2);
          mean_buff1.append(np.mean(buff1))
          mean_buff2.append(np.mean(buff2))
      print(mean_buff1)
      print(mean_buff2)
      #     efficy.append(1-(np.mean(input1)+np.mean(input2))/2);



      # print(buff2)
      plt.plot(p_value,mean_buff1)
      plt.plot(p_value,mean_buff2)
      plt.show()
```

```
[0.0, 0.010101010101010102, 0.02356902356902357, 0.05303030303030303, 0.19797979797979798, 0.335016835016835, 0.3997113997113997, 0.9318181818181818, 1.7115600448933783]
[0.0, 0.015151515151515152, 0.05387205387205387, 0.1893939393939394, 2.111111111111111, 3.6094276094276094, 4.865800865800866, 7.973484848484849, 10.241301907968575]
```

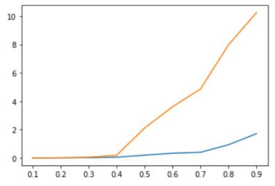
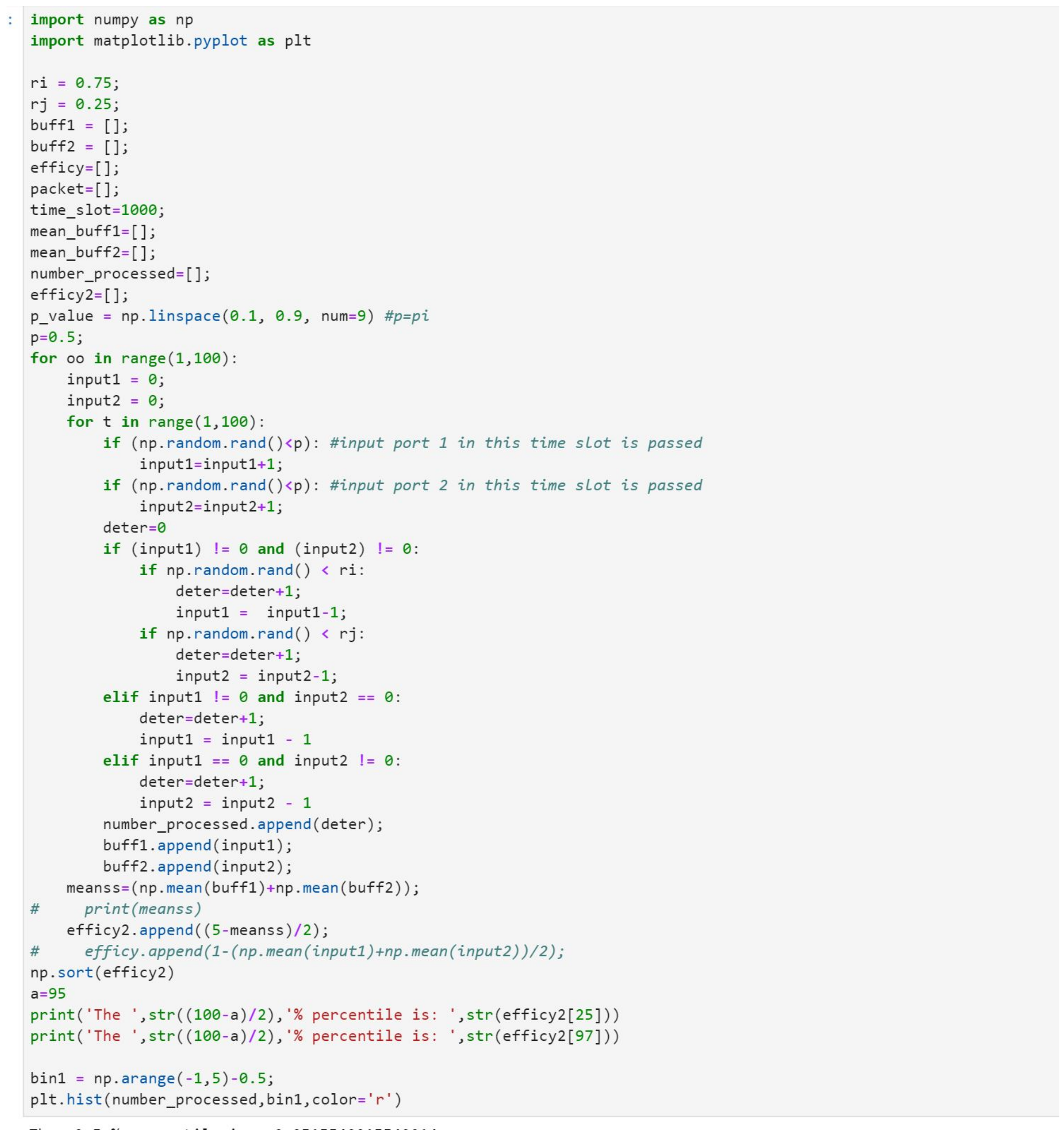


Figure 4: the code and result for the distribution and compute the mean of the number of packets in the buffer at input 1 and input 2 when r1=0.75 and r2=0.25



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

ri = 0.75;
rj = 0.25;
buff1 = [];
buff2 = [];
efficy=[];
packet=[];
time_slot=1000;
mean_buff1=[];
mean_buff2=[];
number_processed=[];
efficy2=[];
p_value = np.linspace(0.1, 0.9, num=9) #p=pi
p=0.5;
for oo in range(1,100):
    input1 = 0;
    input2 = 0;
    for t in range(1,100):
        if (np.random.rand()<p): #input port 1 in this time slot is passed
            input1=input1+1;
        if (np.random.rand()<p): #input port 2 in this time slot is passed
            input2=input2+1;
        deter=0
        if (input1) != 0 and (input2) != 0:
            if np.random.rand() < ri:
                deter=deter+1;
                input1 =  input1-1;
            if np.random.rand() < rj:
                deter=deter+1;
                input2 = input2-1;
        elif input1 != 0 and input2 == 0:
            deter=deter+1;
            input1 = input1 - 1
        elif input1 == 0 and input2 != 0:
            deter=deter+1;
            input2 = input2 - 1
        number_processed.append(deter);
        buff1.append(input1);
        buff2.append(input2);
    meanss=(np.mean(buff1)+np.mean(buff2));
#     print(meanss)
    efficy2.append((5-meanss)/2);
#     efficy.append(1-(np.mean(input1)+np.mean(input2))/2);
np.sort(efficy2)
a=95
print('The ',str((100-a)/2),'% percentile is: ',str(efficy2[25]))
print('The ',str((100-a)/2),'% percentile is: ',str(efficy2[97]))

bin1 = np.arange(-1,5)-0.5;
plt.hist(number_processed,bin1,color='r')
```

Figure 5: the code for the distribution and compute the mean of the number of packets processed by the switch per time slot when r1=0.75 and r2=0.25

```
The  2.5 % percentile is:  0.6958041958041958
The  2.5 % percentile is:  0.7170171098742528
(array([   0., 1640., 7167.,  994.,    0.]),
 array([-1.5, -0.5,  0.5,  1.5,  2.5,  3.5]),
 <a list of 5 Patch objects>)
```

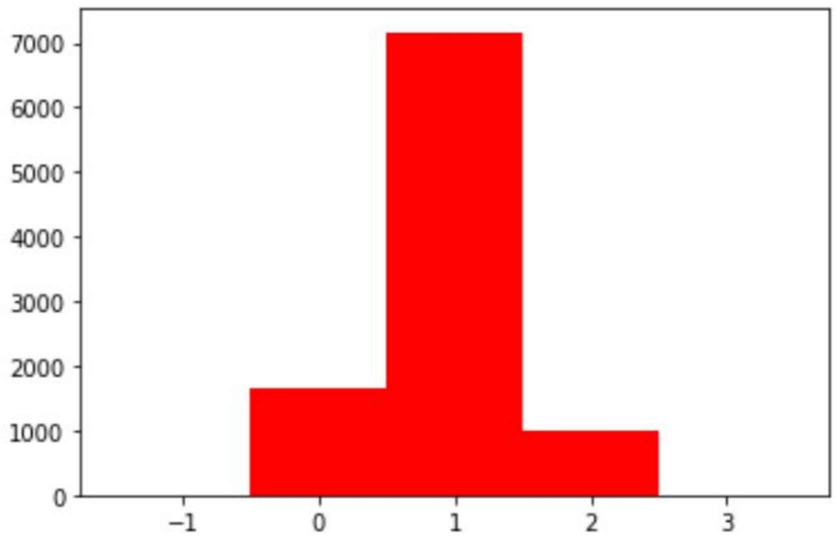


Figure 6: the result for the distribution and compute the mean of the number of packets processed by the switch per time slot when r1=0.75 and r2=0.25

Explanation

For the first case, when the r1=r2=0.5, the figure 2 shows two similar lines. and for the case when r1=0.75 and r2=0.25, the line for r2 is higher than r1, it's is correct, because the packet in the buffer 1 will have high chance to be delivered. And the for processed packets. I ran the 10000 times, the figure shows a reasonable graph, the case of 1 should be the most, because the probability of case of 1 is 1/2. The 95 confidence interval of r1=0.75 and r2=0.25 is 0.695 and 0.717. The efficiency is also reasonable in this case. For the r1=r2=0.5 case, the confidence interval is 0.566045066045066 and 0.6449185734900023

Question 3

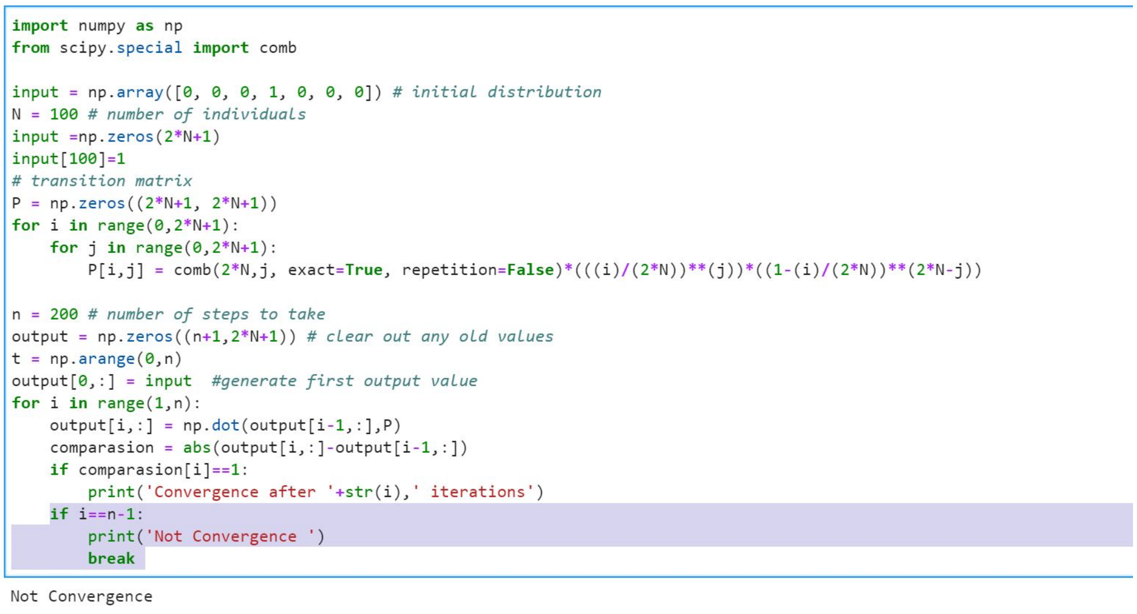
The Wright-Fisher model uses a Markov chain to simulate stochastic genotypic drift during successive generations. The Wright-Fisher model applies to populations under the following assumptions: a. the population size N remains constant between generations b. no selective difference between alleles c. non-overlapping generations. ....

Theory

In this question, I will use Wright Fisher model to uses a Markov chain to simulate stochastic genotypic drift during successive generation. For our example, we have 100 individuals have A1 and A2 genotype. There are 100 A1 and 100 A2 in our data point, the dot product have 0 to 200 times. Therefore, we have 201 data point with our initial distribution matrix. There are two steps of our question, the first step is the transition matrix, and the second step is checking when the convergence will be happened if the matrix use the dot product with preview matrix multiple with our transition matrix. Finally, I will set user input to change our probability of allele distributions. In the figure part, I will show a example of this transition process. And I will show a figure that from article shows the number of generation and frequency of allele

Algorithm 3: .

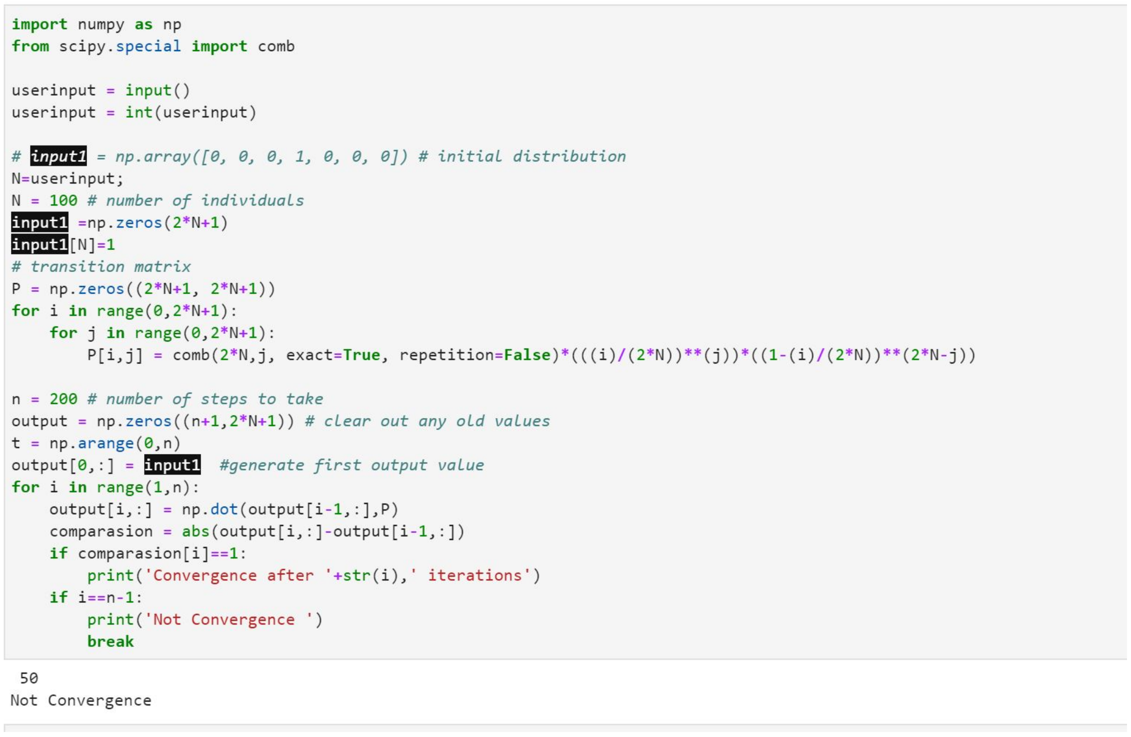
```
/* Calculate the transition matrix */
1 P = np.zeros((2*N+1, 2*N+1))  for i in range(0,2*N+1):
2 for j in range(0,2*N+1):
3 P[i,j] = comb(2*N,j, exact=True,
  repetition=False)*(((i)/(2*N))**(j))*((1-(i)/(2*N))**(2*N-j))
/* Convergence of number iterations */
4 output[0,:] = input generate first output value
5 for i in range(1,n):
6 output[i,:] = np.dot(output[i-1,:],P)
7 if np.allclose(output[i,:],output[i-1,:]):
8 print('Convergence after '+str(i),' iterations')
9 break
```



```
import numpy as np
from scipy.special import comb

input = np.array([0, 0, 0, 1, 0, 0, 0]) # initial distribution
N = 100 # number of individuals
input =np.zeros(2*N+1)
input[100]=1
# transition matrix
P = np.zeros((2*N+1, 2*N+1))
for i in range(0,2*N+1):
    for j in range(0,2*N+1):
        P[i,j] = comb(2*N,j, exact=True, repetition=False)*(((i)/(2*N))**(j))*((1-(i)/(2*N))**(2*N-j))

n = 200 # number of steps to take
output = np.zeros((n+1,2*N+1)) # clear out any old values
t = np.arange(0,n)
output[0,:] = input  #generate first output value
for i in range(1,n):
    output[i,:] = np.dot(output[i-1,:],P)
    comparasion = abs(output[i,:]-output[i-1,:])
    if comparasion[i]==1:
        print('Convergence after '+str(i),' iterations')
    if i==n-1:
        print('Not Convergence ')
        break
```

```
Not Convergence
```

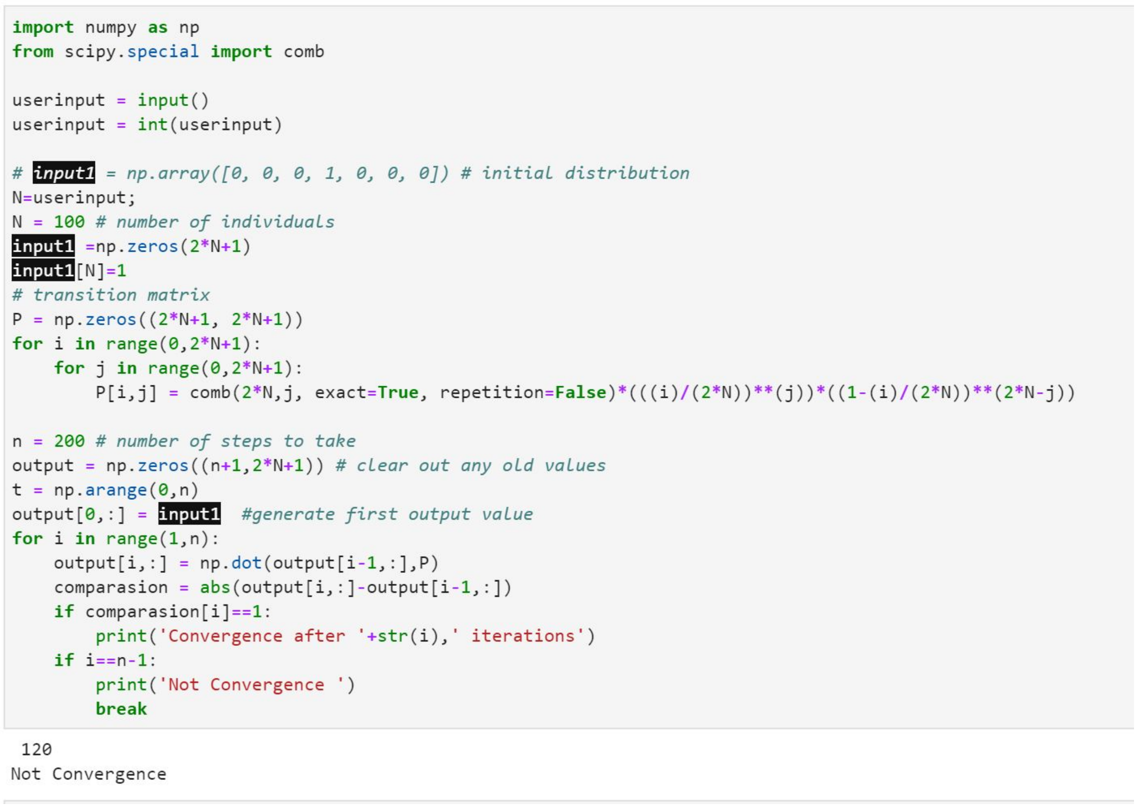
Figure 7: When 100 individuals have (A1 A2) genotype



```
import numpy as np
from scipy.special import comb

userinput = input()
userinput = int(userinput)

# input1 = np.array([0, 0, 0, 1, 0, 0, 0]) # initial distribution
N=userinput;
N = 100 # number of individuals
input1 =np.zeros(2*N+1)
input1[N]=1
# transition matrix
P = np.zeros((2*N+1, 2*N+1))
for i in range(0,2*N+1):
    for j in range(0,2*N+1):
        P[i,j] = comb(2*N,j, exact=True, repetition=False)*(((i)/(2*N))**(j))*((1-(i)/(2*N))**(2*N-j))

n = 200 # number of steps to take
output = np.zeros((n+1,2*N+1)) # clear out any old values
t = np.arange(0,n)
output[0,:] = input1  #generate first output value
for i in range(1,n):
    output[i,:] = np.dot(output[i-1,:],P)
    comparasion = abs(output[i,:]-output[i-1,:])
    if comparasion[i]==1:
        print('Convergence after '+str(i),' iterations')
    if i==n-1:
        print('Not Convergence ')
        break
```

```
 50
Not Convergence
```

Figure 8: The user input



```python
import numpy as np
from scipy.special import comb

userinput = input()
userinput = int(userinput)

# input1 = np.array([0, 0, 0, 1, 0, 0, 0]) # initial distribution
N=userinput;
N = 100 # number of individuals
input1 =np.zeros(2*N+1)
input1[N]=1
# transition matrix
P = np.zeros((2*N+1, 2*N+1))
for i in range(0,2*N+1):
    for j in range(0,2*N+1):
        P[i,j] = comb(2*N,j, exact=True, repetition=False)*(((i)/(2*N))**(j))*((1-(i)/(2*N))**(2*N-j))

n = 200 # number of steps to take
output = np.zeros((n+1,2*N+1)) # clear out any old values
t = np.arange(0,n)
output[0,:] = input1  #generate first output value
for i in range(1,n):
    output[i,:] = np.dot(output[i-1,:],P)
    comparasion = abs(output[i,:]-output[i-1,:])
    if comparasion[i]==1:
        print('Convergence after '+str(i),' iterations')
    if i==n-1:
        print('Not Convergence ')
        break
```

```
 120
Not Convergence
```

Figure 9: The second user input

$$\begin{pmatrix} 1 & 0.75 & 0.5 & 0.25 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0.25 & 0.5 & 0.75 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0.5 \\ 0 \\ 0 \\ 0 \\ 0.5 \end{pmatrix}.$$

Figure 10: Example of transition matrix from the article

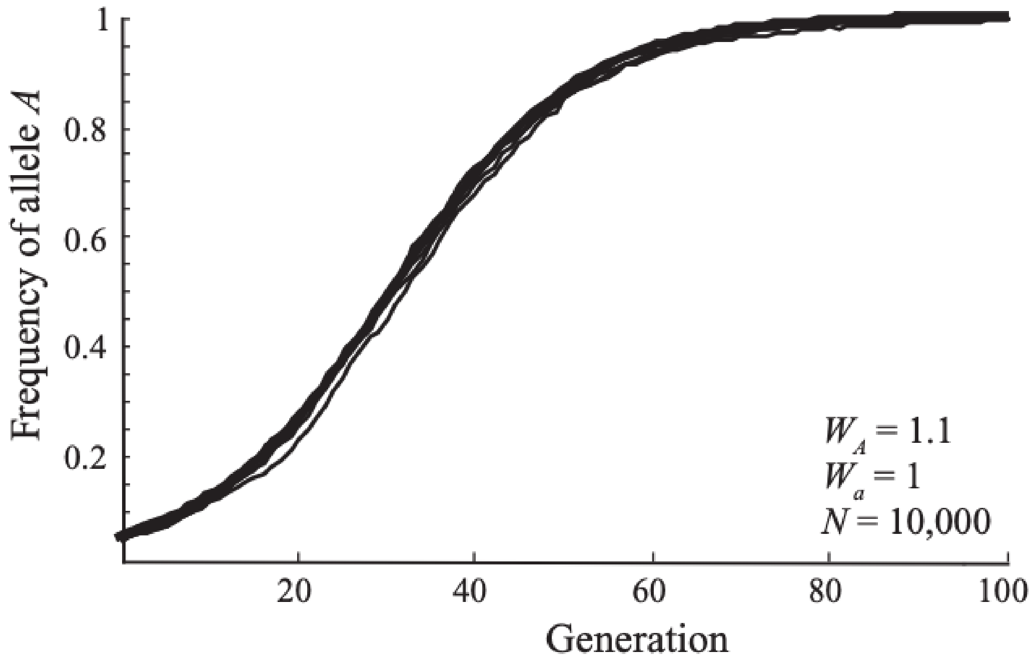


Figure 11: The graph for when the number generation increase, it will converge

Explanation

In the first case, the the convergence is not happened for 200 times, but the number is close to convergence. Since I will say the convergence is at 200 iterations. When the user input 50. 120, all the matrices are not converge. Maybe the number is too small, When I increase the number, the value is close to 0.9, it means that it will converge when the number is large.

Reference

http://assets.press.princeton.edu/chapters/s13$_8$458.*pdf*