Project 5

Wenjun Li

March 10, 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: $\lambda(t+10) = \lambda(t)$. 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.

Results and Analysis

According to the problem statement, we can write the rate of this nonhomogeneous Poisson process as $\lambda = [4, 7, 10, 13, 16, 19, 16, 13, 10, 7, 4, 7, ..., 7]$, i.e. [4, 7, 10, 13, 16, 19, 16, 13, 10, 7] will repeat ten times in the first 100 hours.

As the algorithm of generating nonhomogeneous Poisson process in the textbook shows, we need to generate a uniform random variable U and then set $X = \frac{-1}{\lambda} logU$. Then, we compute $t = t - \frac{-1}{\lambda} logU$ until $t > T$. After generating the arrival time sequence following above steps, we can use the break-time rule and sum up the total amount of break time. If the server completes a job and finds no job is waiting, then it will break for a time as $Uniform[0, 0.3]$. Then service-time of the server is a Exponential distribution with rate 25 jobs per hour.

In my simulation, I repeat the experiment for 100 times and compute the mean of the 100 samples of total break time as my final result. In each trial, I will generate a arrival time sequence at first. Then, I load this sequence and run it on the server. If a job is completed by the server, I will delete it and move forward the following jobs. If the server has no job waiting, then I generate a random number in $Uniform[0, 0.3]$. At the end of each trail, I sum up those break time and put them in a list ($break_time_hist$). The final result of the 100 simulations is shown below:

Expected Break Time = 48.99
Standard Deviation of Break Time = 1.910

Python Code

Listing 1: Code of Question 1

```
1 import numpy as np
2
3
```

```
N = 100                              # number of simulations
break_time_hist = np.zeros(N)        # record the total break time of each
    simulation
rate = [4, 7, 10, 13, 16, 19, 19, 16, 13, 10, 7]    # rate of the non-homogeneous
    poisson process


for trail in range(N):
    # generate arrival time sequence
    arrival_time = []

    for hour in range(100):
        time = hour
        while True:
            u = np.random.uniform()
            lamda = rate[hour % 10]
            time -= (1/lamda) * np.log(u)
            if time - hour > 1:
                break
            arrival_time.append(time)

    # serve the job and break...
    break_time = []
    time_point = 0

    while len(arrival_time):
        if arrival_time[0] > time_point:
            break_time.append(np.random.uniform(0, 0.3))
            time_point += break_time[-1]

        else:
            time_point += np.random.exponential(1/25)
            arrival_time.remove(arrival_time[0])

    break_time_hist[trail] = sum(break_time)


# print results
print(break_time_hist)
print('The expected break time is %.2f' % np.mean(break_time_hist))
print('The std is %.3f' % np.std(break_time_hist))
```

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 $p_i = p$ (a constant) for each input. Assume also that the probability that a packet arriving at input i should be switched to output j is r_{ij}. 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.

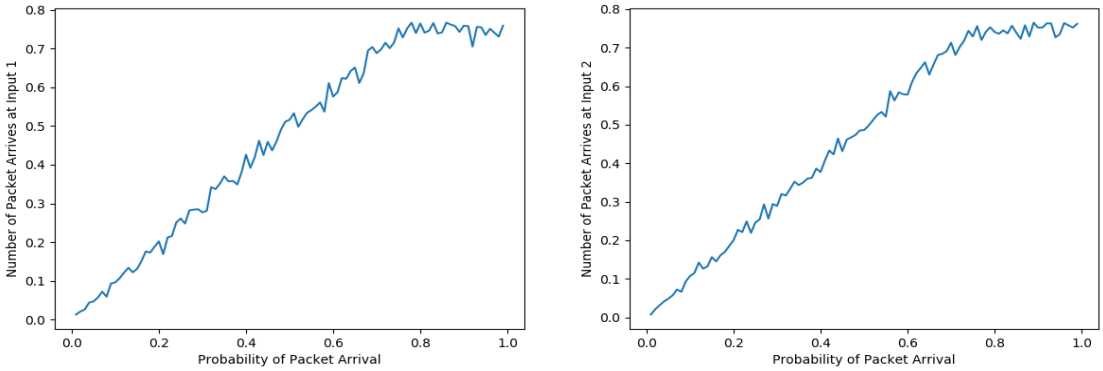
a. Assume $r_{ij} = 0.5$. Plot the distribution and compute the mean of the number of packets in the buffer at input 1 and input 2 as a function of the arrival probability p. Plot the distribution and compute the mean of the number of packets processed by the switch per time slot. Estimate a 95% confidence interval for the overall efficiency of the switch.

b. Repeat (a) assuming asymmetrically targeted packets: $r_1 = 0.75$ and $r_2 = 0.25$.

Results and Analysis

Part (a)

In part (a), the arrival probability is symmertrical, *i.e.* $r_1 = r_2 = 0.5$. To simulate, I let the probability p start from 0 to 1 with a 0.01 interval, *i.e.* there are 100 intervals. In my experiment, I simulate N = 1000 times for each p and compute the average number of packets for input1 and input2. After I collect the mean of the number of packets in the buffer at input1 and input2 w.r.t probability p, I plot the figure as shown in Figure 1.



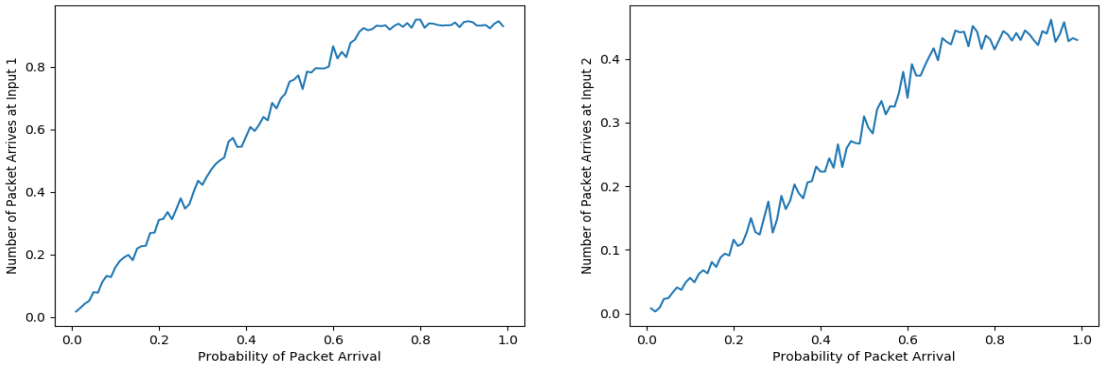
(a) (b)

Figure 1: Number of Packets in the Buffer at (a) input1 and (b) input2 w.r.t. Arrival Probability p when Symmetrical

As we can see, the number of packets in the buffer at input1 and input2 both increase as the probability p becoming larger. And, the number of packets at both buffers converge to 0.75 after $p > 0.75$. This is because the two input ports are the same, *i.e.* symmertrical.

Part (b)

In part (b), we only need to set $r_1 = 0.75$ and $r_2 = 0.25$. The simulation procedures are the same as in part (a). The results are shown in Figure 2.



(a) (b)

Figure 2: Number of Packets in the Buffer at (a) input1 and (b) input2 w.r.t. Arrival Probability p when Asymmetrical

In Figure 2., the convergence result of number of packets in the buffers is different from

Figure 1. The number of packets in input1 converges to 0.85 and the number of packets in input2 converges to 0.45, because the HOT input ports are asymmetrical.

Python Code

Listing 2: Code of Question 2

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


N = 1000
r_ij = 0.5
# r = [0.5, 0.5]            # part (a)
r = [0.75, 0.25]            # part (b)


output1_hist = []
output2_hist = []


for p in np.arange(0, 1, 0.01):
    buffer1 = 0; buffer2 = 0
    output1 = 0; output2 = 0

    for trail in range(N):
        if np.random.uniform() < r_ij:
            buffer1 += 1
        if np.random.uniform() < r_ij:
            buffer2 += 1

        # buffer a && buffer 2
        if buffer1 > 0 and buffer2 > 0:
            select1 = np.random.uniform()
            select2 = np.random.uniform()

            if select1 < r[0] and select2 < r[0]:
                output1 += 1
                if np.random.uniform() < r_ij:
                    buffer1 -= 1
                else:
                    buffer2 -= 1

            elif select1 >= r[0] and select2 >= r[0]:
                output2 += 1
```

```
                if np.random.uniform() < r_ij:
                    buffer1 -= 1
                else:
                    buffer2 -= 1

            else:
                output1 += 1; buffer1 -= 1
                output2 += 1; buffer1 -= 1

        # buffer 1
        elif buffer1 > 0:
            buffer1 -= 1
            if np.random.uniform() < r[0]:
                output1 += 1
            else:
                output2 += 1

        # buffer 2
        elif buffer2 > 0:
            buffer2 -= 1
            if np.random.uniform() < r[0]:
                output1 += 1
            else:
                output2 += 1

    output1_hist.append(output1 / N)
    output2_hist.append(output2 / N)

# plot
plt.figure(1)
x = np.arange(0, 1, 0.01)
plt.plot(x, output1_hist)
plt.xlabel('Probability of Packet Arrival')
plt.ylabel('Number of Packet Arrives at Input 1')
plt.show()

plt.figure(2)
plt.plot(x, output2_hist)
plt.xlabel('Probability of Packet Arrival')
plt.ylabel('Number of Packet Arrives at Input 2')
plt.show()
```

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 remains constant between generations

b. no selective difference between alleles

c. non-overlapping generations.

Consider a gene with 2 alleles (A1 and A2) in a population with N diploid individuals. The population contains $2N$ copies of the gene since each diploid individual has 2 copies of the gene. Let the state vector $x(t)$ represent the allele distribution at time t. Then at time t:

The Wright-Fisher model produces successive generations with a 2-step process. The model first creates N pairs of parents selected randomly and with replacement from the population. Then each pair produces a single offspring with its genotype inherited by selecting one gene from each parent. All parents die after mating. The allele distribution x(t) is a Markov chain that advances by random sampling with replacement from the pool of parent genes. The density of alleles evolves according to a binomial probability density with

$$P[x(t+1) = j | x(t) = i = Bin(j, 2N, \frac{i}{2N})]$$

Thus the Markov chain transition matrix has elements

$$P_{i,j} = \binom{2N}{j} \left(\frac{i}{2N}\right)^j \left(1 - \frac{i}{2N}\right)^{2N-j}$$

Consider a population of $N = 100$ diploid heterozygous individuals, i.e. all 100 individuals have (A1,A2) genotype. Simulate the population's genetic drift using a Markov chain simulation. Repeat the experiment 100 times. Comment on the steady-state population's genetic composition. Repeat the process above using different initial allele distributions. Comment on the steady-state population's genetic composition.

How does the composition of the initial population affect the steady-state outcome? Why does this scenario seem to defy the assertions of the Perron-Frobenius theorem and the Markov chain ergodic theorem?

Results and Analysis

Part (a)

This is a discrete-time Markov chain problem. Markov chain has the property that the probability of moving to the next state depends only on the present state and not on the previous

states, *i.e.* P(Xn = xn|Xn-1 = xn-1, Xn-2 = xn-2,..., X0=x0) = P(Xn = xn|Xn-1 = xn-1).

If all individuals are diploid heterozygous at the beginning, then all $P[icopies A1, 2N - icopies A2] = 0$ except for $P[N copies A1, N copies A2] = 1$ because each individual has one A1 and one A2. So, our input is a (1, 201) zero array with the 100th value = 1. Then, we generate the Markov transition matrix as $P_{i,j} = \binom{2N}{j} \left(\frac{i}{2N}\right)^j \left(1 - \frac{i}{2N}\right)^{2N-j}$. After n = 100 steps, we obtain the final alleles distribution.

In this question, I repeat the Markov process for n = 50, 70, 100 and 500. The alleles distribution results are shown in Figure 3. As we can conclude, as the number of generation increases, the distribution of alleles is more uneven, i.e. there will be less and less heterozygous individuals, but more and more homozygous. In Figure 3., when n=50, there are still some heterozygous; when n=500, almost all the individuals are homozygous. This phenomenon can be concluded from the alleles distribution.

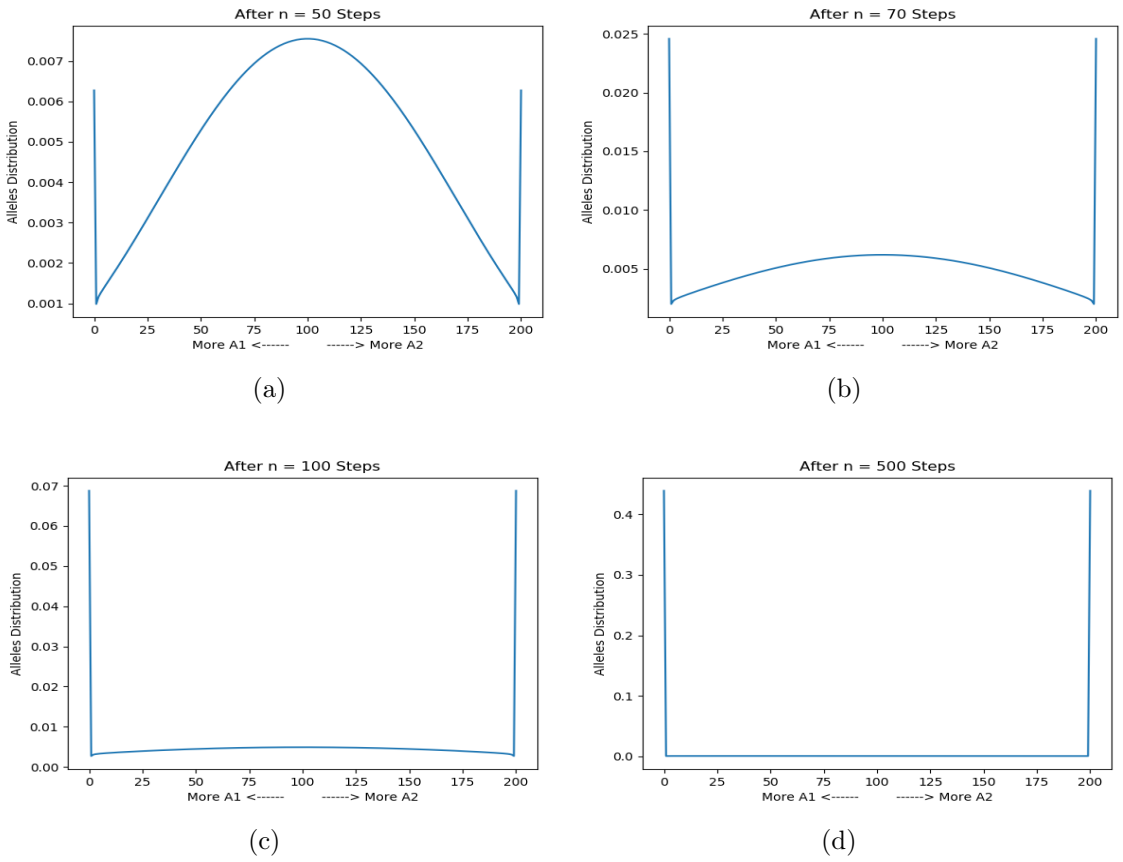


Figure 3: Alleles Distribution with (a)50, (b)70, (c)100 and (d)500 Generations

Part (b)

After doing above simulations, I repeat the process with two different initial allele distributions. The two initial alleles distributions are as below:

a. P[40 copies A1, 160 copies A2] = P[80 copies A1, 120 copies A2] = P[120 copies A1, 80 copies A2] = P[160 copies A1, 40 copies A2] = 0.25

b. P[50 copies A1, 150 copies A2] = 1

The simulation results are displayed in Figure 4.

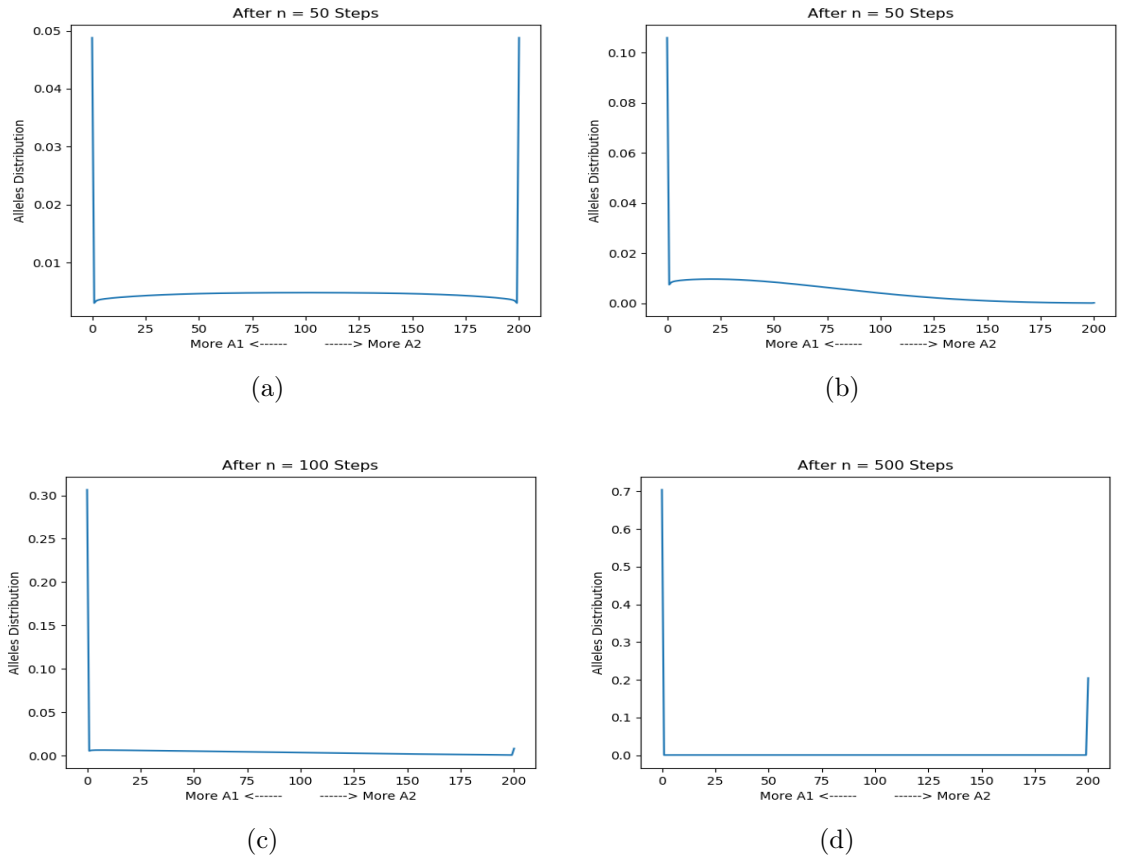


Figure 4: Alleles Distribution with (a) and (b) Initial Alleles Distribution

From the above results, we can conclude that

1. whatever the initial alleles distributions are, they all converge to the same result as in part (a), i.e. half homozygous of A1 and half homozygous of A2.

2. more even the initial alleles distributions is, faster it converges. Specifically, the even initial distribution (a) almost converges after only 50 steps, while (b) needs more than 500 steps.

This scenario actually does not defy the assertions of the Perron-Frobenious theorem and the Markov chain ergodic theorem, because we ignore many natural factors in our simulation.

Python Code

Listing 3: Code of Question 3

```
import numpy as np
from scipy.special import comb
import matplotlib.pyplot as plt


N = 100         # number of individuals
n = 500         # number of steps to take


# define initial input

# all of the N individuals are diploid heterozygous, i.e. P(100—A1, 100—A2) = 1
input = np.zeros((1, 2*N))
input = np.insert(input, 100, 1)

# input = np.append(input, 0.5)     # change the initial allele distributions


# M.C. 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)


# output
output = np.zeros((n+1, 2*N+1))
output[0, :] = input            # generate first output value by copy input value
    to 1st row
t = np.arange(0, n)

for i in range(1, n):
    output[i, :] = np.dot(output[i—1, :], P)
```

```
    if np.allclose(output[i, :], output[i-1, :]):
        print('Convergence after ' + str(i), 'iterations')
        print(output[i, :])
        break


plt.figure(1)
x = np.arange(0, 2*N+1, 1)
plt.plot(x, output[n-1, :])
plt.title('After n = {} Steps'.format(n))
plt.ylabel('Alleles Distribution')
plt.xlabel('More A1 <-------          ------> More A2')
plt.savefig('Q3_n_{}'.format(n))
plt.show()
```