EE511

PROJECT 5

SHASHIKIRAN HEGDE

3103722542

**Q1)** A non-homogeneous Poisson process is similar to an ordinary Poisson process, except that the average rate of arrivals is allowed to vary with time. Many applications that generate random points in time are modeled more faithfully with such non-homogeneous processes. In our case, we have a Single Server Queuing system, where customers arrive according to a non-homogeneous Poisson process with intensity function lambda(t), t>=0. Server handles customer if free else joins queue. When server completes serving a customer, it begins handling longest waiting customer( First come first served) or remains free if queue is empty. The amount of time to service a customer follows a particular probability distribution. Fixed time T after which no new arrivals occur. But server continues to handle all customers in queue. The quantities of interest to us here is

* The average time customer spends in queue
* The average time past T that last customer departs.

N=100; %number of hours%

t=0;

x=0;

j=1;

lambda=20;

break\_time=0;

%Algorithm for non-homogeneous Poisson process where lambda\_t<lambda

while t<100

U1=rand();

t=t-(log(U1)/lambda);

if(mod(t,10) < 5)

lambda\_t=3\*t+4; %for the first 5 hours the arrival rate linearly increases

ratio=lambda\_t/lambda;

elseif(mod(t,10)>5)

lambda\_t=-3\*t-4; %for the next 5 hours the arrival rate linearly decreases

ratio=lambda\_t/lambda;

end

U2=rand();

if(U2<=ratio)

Ts(j)=t; %the time of the first arrival after some time%

j=j+1;

end

end

%Break time calculation

for i=1:length(Ts)-1

s=exprnd(25); %service time distribution with a rate of 25 jobs per hour

while((s + Ts(i)) < Ts(i+1)) %if there are no jobs waiting

idle=0.3\*rand(); %uniformly distributed break

break\_time=break\_time+idle; %total break time for 100 hours

s=s+idle;

end

end

disp(break\_time)

**CODE DESCRIPTION**

In the code we use an algorithm for non homogeneous Poisson process which is roughly

Suppose that λ(t) is the bounded intensity function (arrival function) for a non-homogenous Poisson process. To generate a sample Ts that is the time of the first arrival after time s:

Algorithm: Choose λ so that λ(t)<λ for all t. Given λ(t), t>0, and λ:

Let t=s.

Generate U1∼U[0,1].

Let t=t−1/λ⋅logU1.

Generate U2∼U[0,1].

If U2≤(λ(t)/λ) set Ts=t and stop.

Goto step 2.

For the calculation of lambda\_t there are 2 cases. In the first case it linearly increases from 4 to 19 hours for the first 5 hours, which is given by the expression 3\*t+4. In the second case it linearly decreases from 19 hours to 4 hours in the next 5 hours. It is given by the expression -3\*t-4.

The service‐time distribution is exponential with rate 25 jobs per hour. The function R = exprnd(MU) returns an array of random numbers chosen from the exponential distribution with mean parameter MU. Here MU=25. The waiting time is uniformly distributed on (0,0.3). The variable ‘idle’ in the program represents this. So for 100 hours we increment the break time if there are no jobs waiting in the queue. We get the answer to be 11.2251 which means the server is idle for that many hours in 100 hours.

**OUTPUT**

The break time is 11.2251 hours.

**Q2)** Consider an N × N input-queued switch - time is slotted, so that at most one packet can arrive (depart) per time slot - packets arrive at each input with probability p, independently across inputs/time - the destination of a packet is equally likely to be one of the outputs and independent across all packets. Head-of-line blocking occurs whenever traffic waiting to be transmitted prevents or blocks traffic destined elsewhere from being transmitted. Head-of-line blocking occurs most often when multiple high-speed data sources are sending to the same destination. A known phenomenon that comes with the input-buffered switch is the Head-Of-Line (HOL) blocking. What exactly happens is packets blocked at the head of the queue also block the packets behind them, even if some of these packets are destined for idle output ports. By using queuing analysis, HOL blocking is shown to reduce available throughput to 58% even under uniform traffic pattern. However, input-port buffering is the simplest to design as the internal speed of the buffer only operates at the same speed as the input/output links.

N = 1000;

%p1 and p2 are the probabilites of packets arriving at port 1 and port 2

p1 = 0.7;

p2 = 0.7;

flag1 = 0;

flag2 = 0;

packetnum\_1 = 0;

packetnum\_2 = 0;

buff\_1 = 0;

buff\_2 = 0;

P1 = zeros(1,N);

P2 = zeros(1,N);

p\_switch1 = zeros(1,N);

p\_switch2 = zeros(1,N);

Num\_packets = zeros(1,N);

Buff1 = zeros(1,N);

Buff2 = zeros(1,N);

p1\_no = 0; p2\_no = 0; k = 1;

for i = 1:N

packet\_switched = 0; X = 1; Y = 1; nopacket\_flag = 0;

P1(i) = rand();

P2(i) = rand();

if (P1(i) < p1) %packet arrives at line 1

packetnum\_1 = packetnum\_1 + 1;

p\_switch1(i) = rand();

else

nopacket\_flag = 1;

end

if (P2(i) < p2) %packet arrives at line 2

packetnum\_2 = packetnum\_2 + 1;

p\_switch2(i) = rand();

else

nopacket\_flag = 1;

end

if (p\_switch1(i) < 0.75) %rij

p\_switch1(i) = 0;

else

p\_switch1(i) = 1;

end

if (p\_switch2(i) < 0.25)

p\_switch2(i) = 0;

else

p\_switch2(i) = 1;

end

if(i>1)

if(flag1 == 1) %if contention has occured

p\_switch2(i)=p\_switch2(i-1);

if ( buff\_2 == 0 ), flag1 = 0; end

end

if(flag2 == 1)

p\_switch1(i)=p\_switch1(i-1);

if ( buff\_1 == 0 ), flag2 = 0; end

end

end

if ((p\_switch1(i) == 1 && p\_switch2(i) == 0) || (p\_switch1(i) == 0 && ...

p\_switch2(i) == 1) || nopacket\_flag)%for states 01,10,no packet

packet\_switched = packet\_switched + ...

packetnum\_1 + packetnum\_2; X = 0; Y = 0;

else %for states 00 and 11

contention = rand(1);

if(contention < 0.5)

buff\_2 = buff\_2 + 1;

packet\_switched = packet\_switched + 1;

if (buff\_1 > 0), buff\_1 = buff\_1 - 1; end

flag1 = 1; X = 0;

else

buff\_1 = buff\_1 + 1;

packet\_switched = packet\_switched + 1;

if (buff\_2 > 0), buff\_2 = buff\_2 - 1; end

flag2 = 1; Y = 0;

end

end

Num\_packets(i) = packet\_switched;

if(packetnum\_1\*packetnum\_2 == 1)

Eff\_switch(k) = Num\_packets(i)/(packetnum\_1 + ...

packetnum\_2); %2 packets should be switched everytime

k = k + 1;

end

Buff1(i) = buff\_1; Buff2(i) = buff\_2;

p1\_no = p1\_no + packetnum\_1 - X;%packets at output 1

packetnum\_1 = 0;

p2\_no = p2\_no + packetnum\_2 - Y;%packets at output 2

packetnum\_2 = 0;

end

b = bootci(1000, @mean, Eff\_switch);

disp('Confidence interval for overall efficiency');

disp(b);

M\_switched = mean(Num\_packets);

M\_B1 = mean(Buff1); M\_B2 = mean(Buff2);

Total\_packets\_switched = p1\_no + p2\_no;

figure(1);

histogram(Buff1);

disp(M\_B1);

title('Distribution of number of packets in buffer at input line 1');

xlabel('Num of packets');

ylabel('Frequency');

figure(2);

histogram(Buff2);

disp(M\_B2);

title('Distribution of number of packets in buffer at input line 2');

xlabel('Num of packets');

ylabel('Frequency');

figure(3);

histogram(Num\_packets);

disp(M\_switched);

title('Distribution of number of packets switched in 1 timeslot');

xlabel('Number of packets');

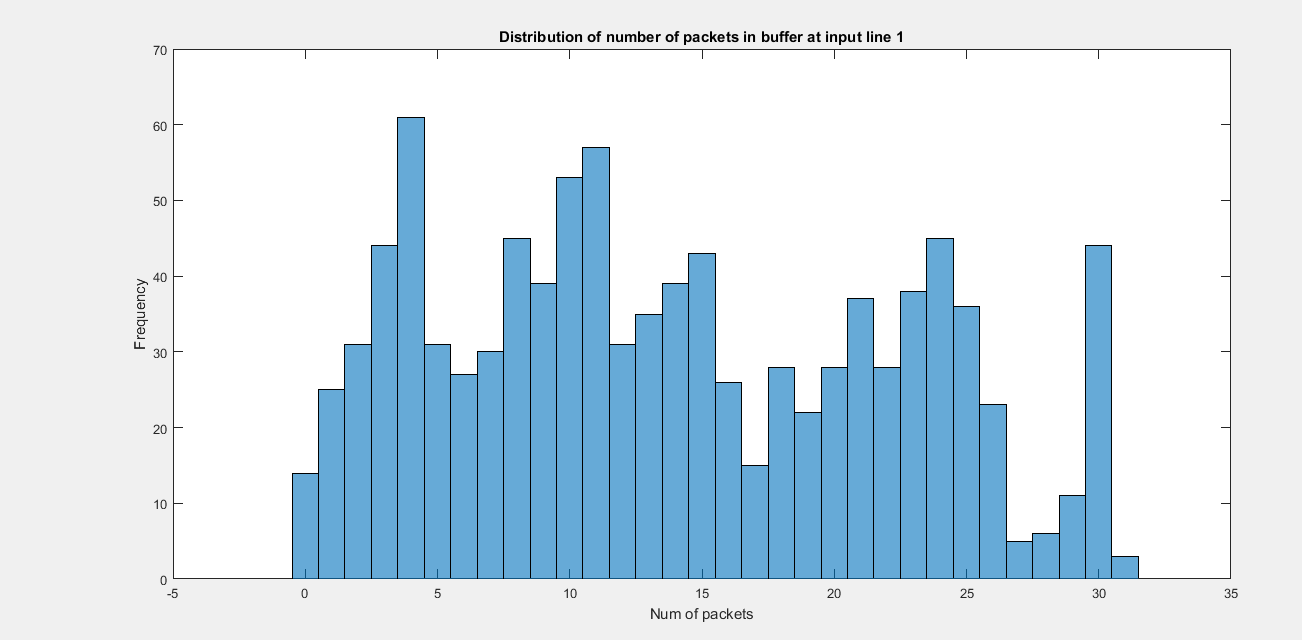
ylabel('Frequency');

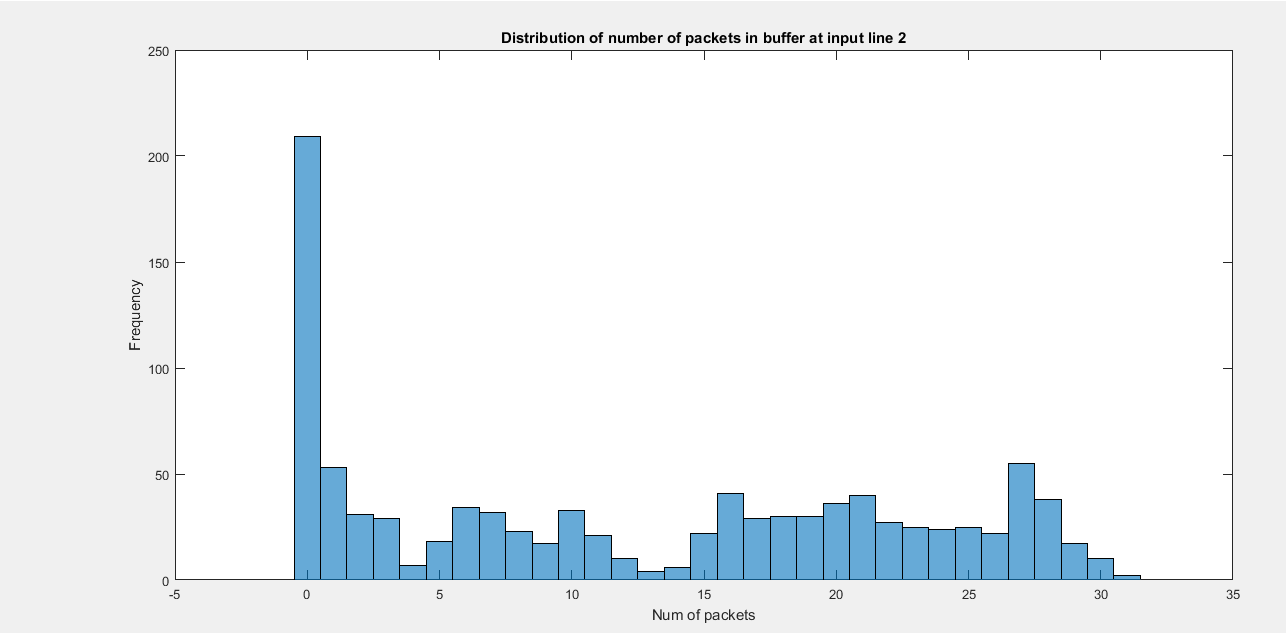
**CODE DESCRIPTION**

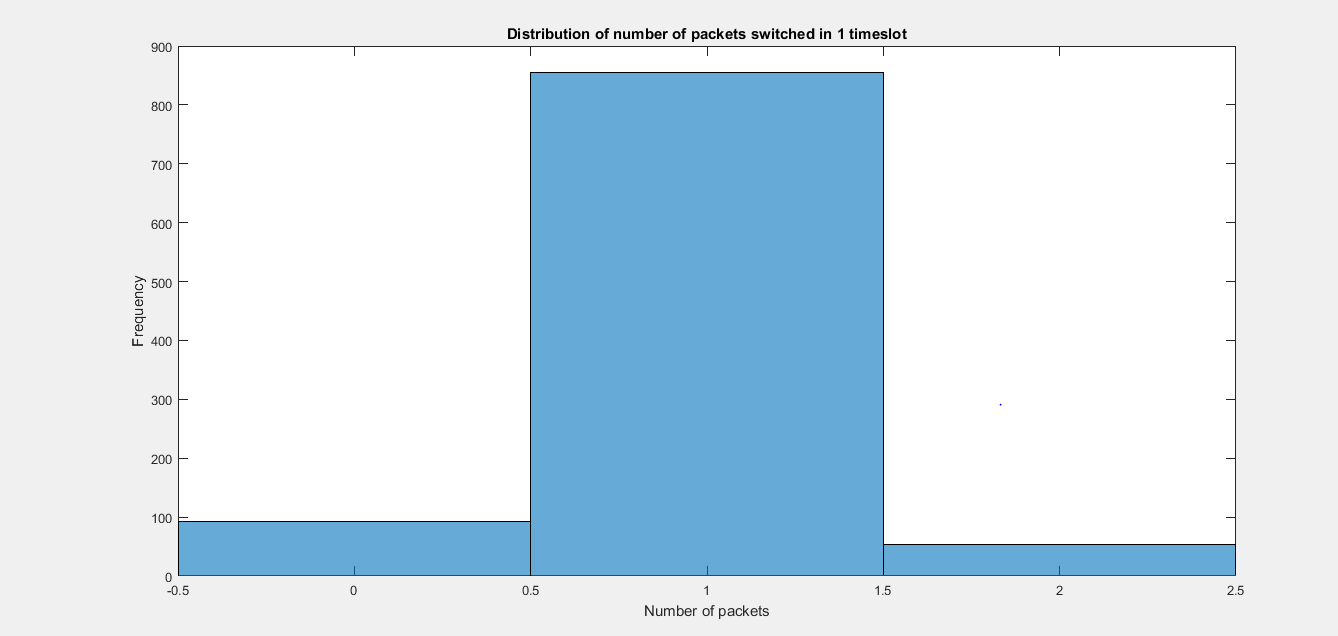
We assign 2 probabilities of packet reaching port1 and port2(0.7) in our case. We generate a random number and check if the value of that is less than 0.7. If it is less, then the packet has arrived at that port. ‘p\_switch’ is a variable where if the variable is less than 0.5 in the first case, it wont be switched. In the assymetric case if the variable is less than 0.75 at port1 and 0.25 at port2 it wont be switched. There are 4 states possible, (00),(01),(10),(11). For the states 11 and 00 contention (collision) occurs, only one packet can be sent, if a packet has arrived at the other port it must be buffered. The buffered packet will be sent in the next time slot. For states 01 and 10 packets at both ports can be sent. For the contention case, we are holding 2 flag variables. If flag1=1 then we change the state of port2 to the previous state since packet was in buffer. We reset the buffer and flag for port2. If flag2=1 then we change the state of port1 to the previous state since packet was in buffer. We reset the buffer and flag for port1. We calculate the efficiency of the switch by the total number of packets sent by the total number of packets received in each timeslot. The program asks us to calculate the 95% confidence interval which can be done using the bootci command.

**OUTPUT**

For the first case r=0.5







Confidence interval for overall efficiency

0.6800

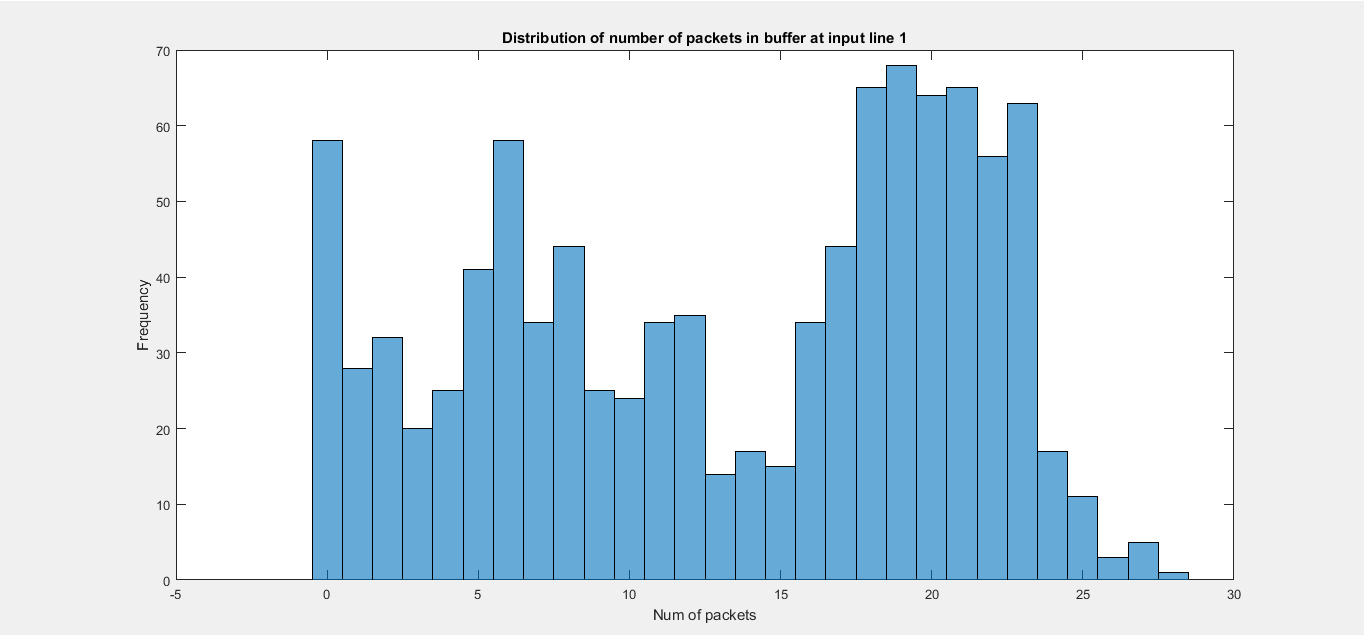
0.7070

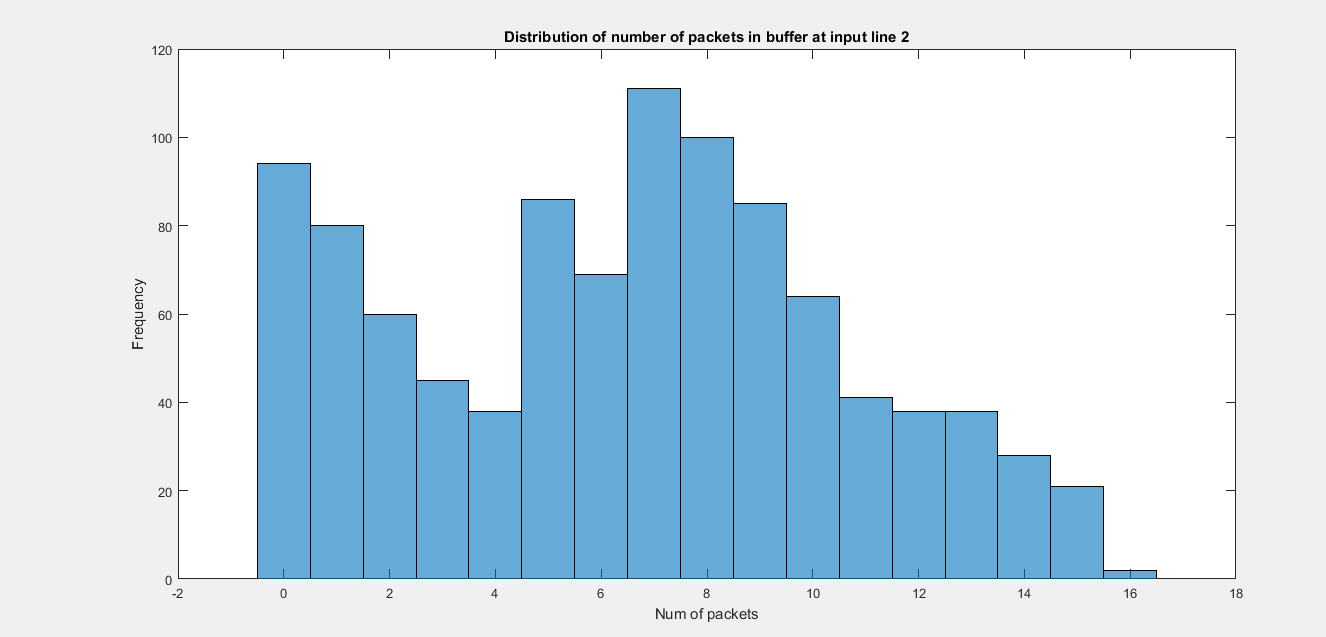
Mean number of packets at port 1= 13.9090

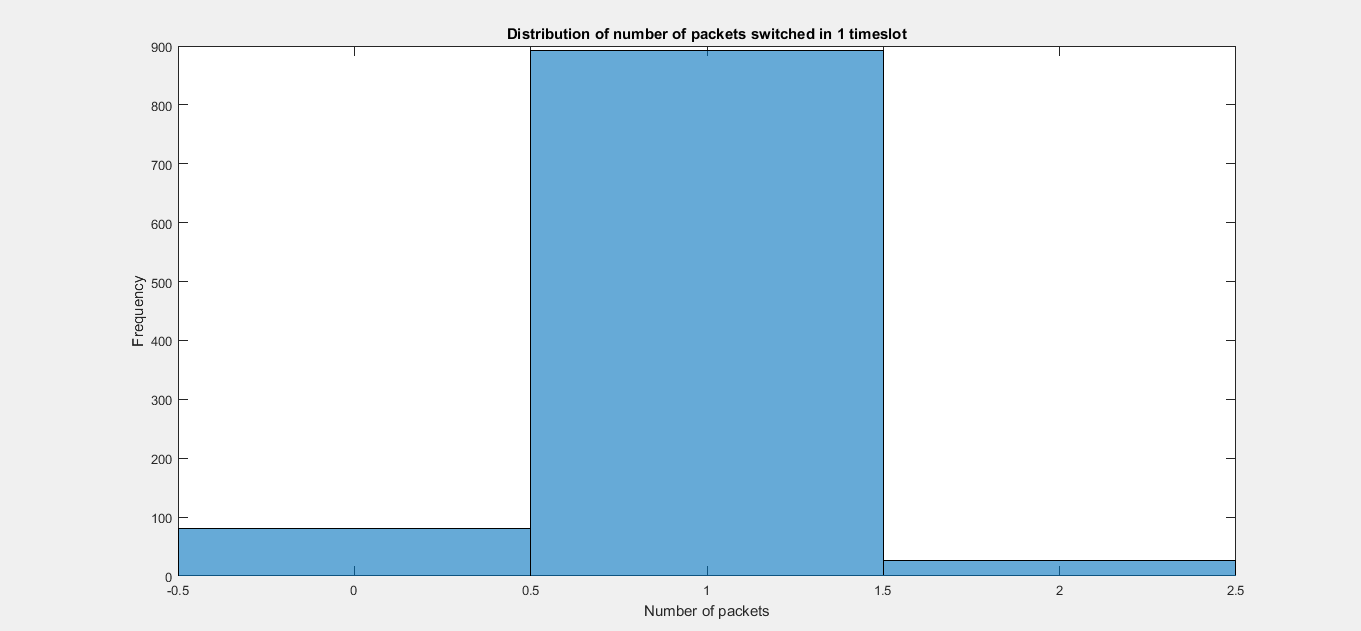
Mean number of packets at port 2= 12.3970

Mean number of packets switched in 1 timeslot= 0.9610

For the second case where r1=0.75 and r2=0.25







Confidence interval for overall efficiency

0.5180

0.5369

Mean number of packets at port 1= 13.3030

Mean number of packets at port 2= 6.4530

Mean number of packets switched in 1 timeslot= 0.9470

**CONCLUSION**

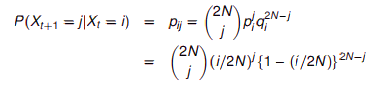
The confidence interval in the symmetric case is more than that in the assymetric case. We can say that efficiency of the switch is more in Symmetric case.

**Q3)** The Wright Fisher model is explained as follows. Assume a simple haploid model; it consists of a population of 2N genes (or alternatively – N diploid organisms) of random reproduction, with each haploid possessing either allele A1 or allele A2. Initially, we may disregard mutation as well as selective forces. At each time-step, a gene (allele) reproduces some number of offspring (which are the exact copies of itself) and dies immediately after that; thus has a life-span of only one generation. The process, modeled thus, describes how the genes get transmitted from one generation to the next

Process of birth and death in the population remains hidden. The only observable is the frequency of alleles changing from generation to generation. The allele frequency of the next generation is governed only by a genetic drift. The Genetic drift is defined as a force that reduces heterozygosity by the random loss of alleles. We should just focus on the frequency of allele A1 in the population of 2N haploids. Think of this process as changing from one generation to the next in terms of a Markov Chain, where the state X of the chain corresponds to the number of haploids (genes) of type A1.

In any generation X takes one of the values 0, 1, . . . , 2N, which constitutes a state space. Denote the value taken by X in generation t as Xt . The model assumes that genes for the generation t + 1 are derived by sampling with replacement from the genes of generation t. Thus, the make-up of the next generation is determined by 2N independent Bernoulli trials so that Xt is a binomial random variable. Let the initial generation consist of i genes of type A1 and 2N − i genes of type A2. Then we define a probability of success (resulting in allele A1) pi and a probability of failure qi (resulting in allele A2) for each Bernoulli trial as pi = i/2N, qi = 1 – i/2N. The process generates a Markov Chain {Xn}, where Xn is the number of A1 genes in the nth generation, among a constant population size of 2N individuals. Basically, Xt+1 is a binomial random variable with index 2N and parameter (probability of success) Xt/2N.

the transition probabilities from Xt = i to Xt+1 = j for this Markov Chain are computed according to the binomial distribution as



A matrix for which all the column vectors are probability vectors is called transition or stochastic matrix. A Markov chain is a process that consists of a finite number of states and some known probabilities pij, where pij is the probability of moving from state j to state i. Of particular interest is a probability vector **p** such that **Ap**=**p**, that is, an eigenvector of **A** associated to the eigenvalue 1. Such vector is called a steady state vector. The **Perron–Frobenius theorem**, asserts that a [real square matrix](https://en.wikipedia.org/wiki/Real_square_matrix) with positive entries has a unique largest real [eigenvalue](https://en.wikipedia.org/wiki/Eigenvalue) and that the corresponding [eigenvector](https://en.wikipedia.org/wiki/Eigenvector) can be chosen to have strictly positive components, and also asserts a similar statement for certain classes of nonnegative matrices. A Markov chain is ergodic if there is a number N such that any state can be reached from any other state in at most N steps (in other words, the number of steps taken are bounded by a finite positive integer N). In case of a fully connected transition matrix, where all transitions have a non-zero probability, this condition is fulfilled with N=1. A Markov chain with more than one state and just one out-going transition per state is either not irreducible or not aperiodic, hence cannot be ergodic.

clc;

clear all;

close all;

user=input('Enter the location of A1 allele');%Accepting user input

input=zeros(1,201); % initial distribution

input(user)=1;%Passing user input

N = 100; % number of individuals

% transition matrix

P=zeros(2\*N+1,2\*N+1);

for i = 1:2\*N+1

for j = 1:2\*N+1

P(i,j) = nchoosek(2\*N,j-1)\*((i-1)/(2\*N))^(j-1)\*(1-(i-1)/(2\*N))^(2\*N-j+1);

end

end

n=1000; % number of time steps to take

output=zeros(n+1,2\*N+1); % clear out any old values

t=0:n; % time indices

output(1,:)=input; % generate first output value

for i=1:n,

output(i+1,:) = output(i,:)\*P;

%a tolerance check to automatically stop the simulation when the density is close to its steady-state

LIT =(output(i+1,:)- output(i,:)); %calculate the threshold

if all(LIT == 1)

break;

end

end

plot(t,output);

xlabel('Time axis');

ylabel('Probability of a particular state')

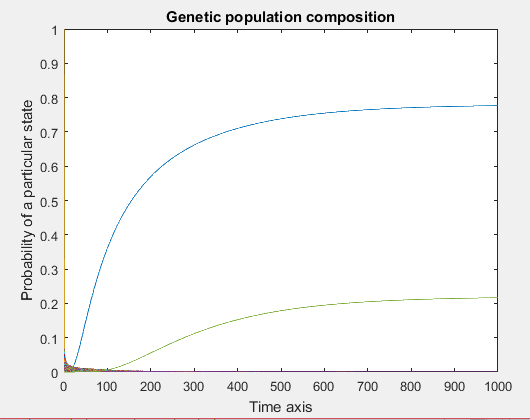
title('Genetic population composition')

**CODE DESCRIPTION**

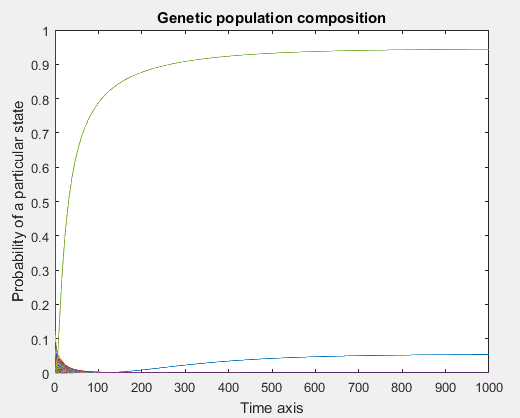
The code starts off with prompting the user to enter the value of the location of the A1 Allele in the gene. Initialize an array of zeros as in the code and the change the value of the location entered by the user to 1. Consider N to be 100. We then initialize a transition or stochastic matrix of 201x201 to zeros. We then populate each element of the matrix using the formula described above. Nchoosek function gives us nCr value. We choose n=1000 as the time indice. We then calculate the output as in the code below by multiplying with the transition matrix a number of times. The output is plotted and the results can be seen to be in a steady state.

**OUTPUT**

For user input=35



For user input=180



**CONCLUSION**

As can be seen from the graphs above with different allele initial distribution, the steady state reaches more quickly in the first case when compared to the second graph. Hence, this defies the Perron-Frobenius theorem and the Markov chain ergodic theorem as can be clearly seen from above.