Project #5 – Markov chains and discrete events

1. *Suppose that jobs arrive at a single‐server queue system per 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* λ(t+10) = λ(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.*

**Concept behind this problem:**

The question requires the knowledge of a ***Single-Server Queueing System*** and ***Discrete Event Simulation***. This is the simple model which assumes that the number of arrivals occurring within a given interval of time t, follows a Poisson distribution. with parameter (λ)t. It is characterized by three components:

1. **Arrival Process:** Upon arrival a customer either enters service if this server is free at that moment or else joins the waiting queue if the server is busy.
2. **Queue Discipline:** When the server completes serving a customer, it then either begins serving the customer that had been waiting the longest (the so-called “first come first served”-FIFO discipline) if there are any waiting customers, or, if there are no waiting customers, it remains free until the next customer’s arrival.
3. **Service Mechanism:** The amount of time it takes to service a customer is a random variable having probability distribution G. In addition, there is a fixed time after which no additional arrivals can enter the system, although the server completes servicing all those that are already in the system at time.

A **discrete-event simulation** (DES) models the operation of a system as a discrete sequence of events in time. Each event occurs at an instant in time and marks a change of state in the system. Between consecutive events, no change in the system is assumed to occur. Thus, the simulation can directly jump in time from one event to the next. It codifies the behavior of a complex system as an ordered sequence of well-defined events.

**Reference:**

**Chp-7-The-Discrete-Event-Simulation-Approach Simulation 5th Edition by Sheldon Ross**

**Code & Code Description:**

* Initialize variables and create an exponential function to calculate the theoretical value of the integral

%Initialize variables

mean=1/25;

index = 1;

arr\_cnt=0; %number of arrivals

dept\_cnt=0; %number of departures

cust\_cnt=0; %number of customers

curr\_time=0;

hrs\_of\_oper=100; %Total hours of operation

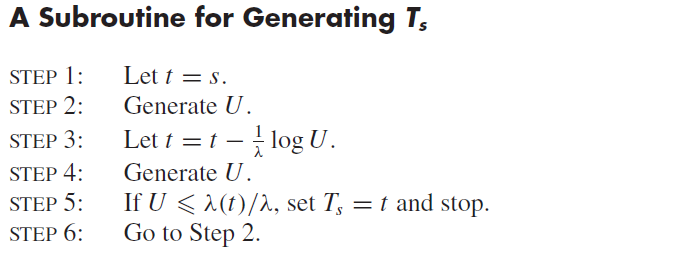
srvc\_time=0; %Service time of one job

break\_time=0; %Break time

total\_break\_time=0; %Total break time (calc for 100 hours)

srvc\_comp\_time=inf; %Service completion time

* The first step of creation of single server queue is to design an algorithm to get the arrival count. There are two ways of achieving this.
  1. Create a function which produces single arrival time and keep calling this function every time we want to get the current and the next arrival time
  2. Create a function which produces an array of arrival time(array\_of\_arr\_time), store it in a vector and use this vector the entire way. (Below code is for this implementation).



while(array\_of\_arr\_time(index) < hrs\_of\_oper)

lamda = 19;

curr\_time = array\_of\_arr\_time(index);

while(curr\_time<hrs\_of\_oper)

u\_rand1 = rand();

curr\_time = curr\_time - log(u\_rand1)/lamda;

u\_rand2 = rand();

mod\_time = mod(curr\_time,10);

if(mod\_time < 6)

comp\_time=mod(4+3\*mod\_time,10);

else

comp\_time = mod(19-3\*(mod\_time-5),10);

end

comp\_time = comp\_time/lamda;

if (u\_rand2 <= comp\_time)

array\_of\_arr\_time(index+1) = curr\_time;

array\_of\_arr\_time(index) = array\_of\_arr\_time(index+1);

index=index+1;

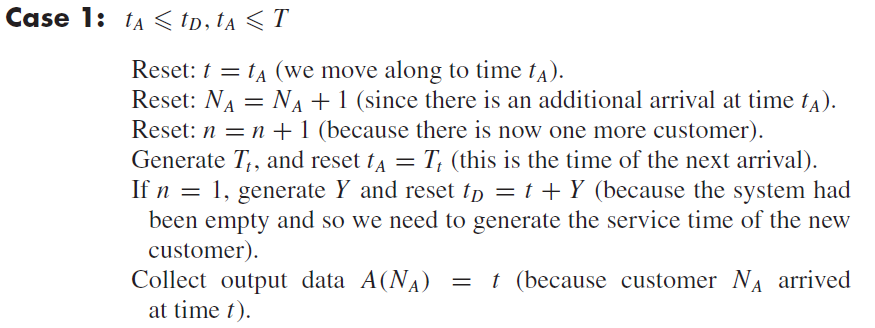
break;

end

end

end

* Once we have the arrival time stored in an array, we have four cases to consider.
  1. **Case-1:** service completion time of customer presently being served greater than arrival time.



breaktime = array\_of\_arr\_time(1);

arr\_time = breaktime;

while arr\_time <hrs\_of\_oper

arr\_time = array\_of\_arr\_time(index);

if((arr\_time <=srvc\_comp\_time) && (arr\_time <=hrs\_of\_oper))

curr\_time=arr\_time;

arr\_cnt = arr\_cnt + 1;

cust\_cnt = cust\_cnt +1;

if index == length(array\_of\_arr\_time)

arr\_time = array\_of\_arr\_time(index);

else

arr\_time = array\_of\_arr\_time(index+1);

end if(cust\_cnt == 1)

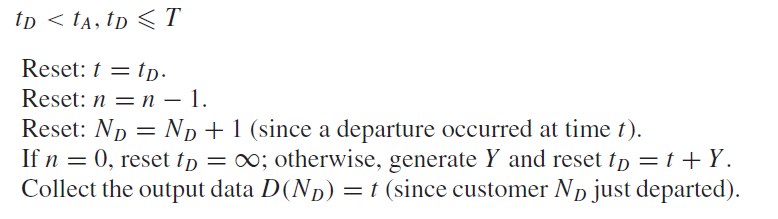
srvc\_comp\_time = breaktime+ exprnd(given\_mean);

end

A(arr\_cnt) = curr\_time;

End

* 1. **Case-2:** service completion time of customer presently being served lesser than arrival time



if((srvc\_comp\_time < arr\_time) && (srvc\_comp\_time <= hrs\_of\_oper))

curr\_time = srvc\_comp\_time;

cust\_cnt = cust\_cnt - 1;

dept\_cnt = dept\_cnt +1;

D(dept\_cnt) = curr\_time;

if(cust\_cnt == 0)

srvc\_comp\_time = inf;

breaktime = curr\_time;

while(breaktime <arr\_time)

breaktime = breaktime + 0.3\*rand();

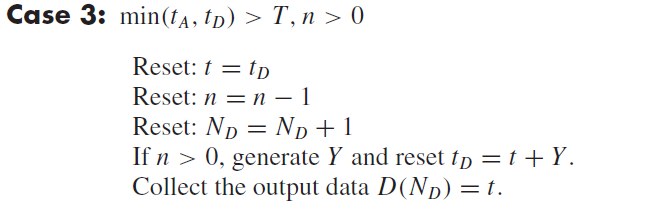
total\_break\_time = total\_break\_time+ breaktime;

end

else

srvc\_comp\_time = curr\_time+ exprnd(1/25); end end

* 1. **Case-3:** Minimum of Arrival and service completime time > Num of Operational hours & The number of customers in the system > 0



%CASE-3

if(min(arr\_time,srvc\_comp\_time) > hrs\_of\_oper && cust\_cnt >0)

curr\_time = srvc\_comp\_time;

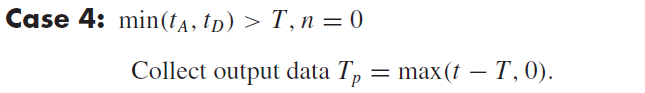
cust\_cnt = cust\_cnt - 1;

dept\_cnt = dept\_cnt +1;

srvc\_comp\_time = curr\_time+ exprnd(1/25);

D(dept\_cnt) = curr\_time; end

* 1. **Case-4:** Minimum of Arrival and service completime time > Num of Operational hours & The number of customers in the system == 0



%CASE-4

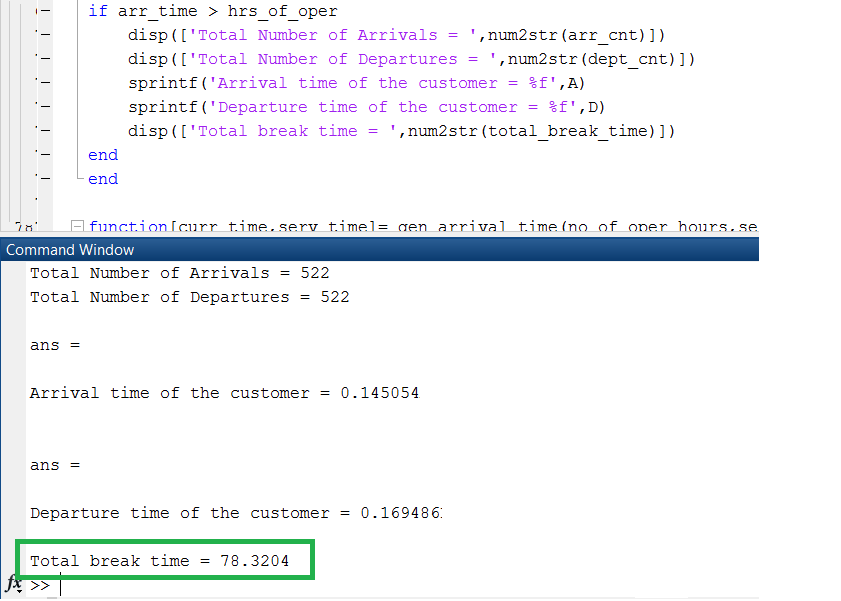
if(min(arr\_time,srvc\_comp\_time) > hrs\_of\_oper && cust\_cnt == 0)

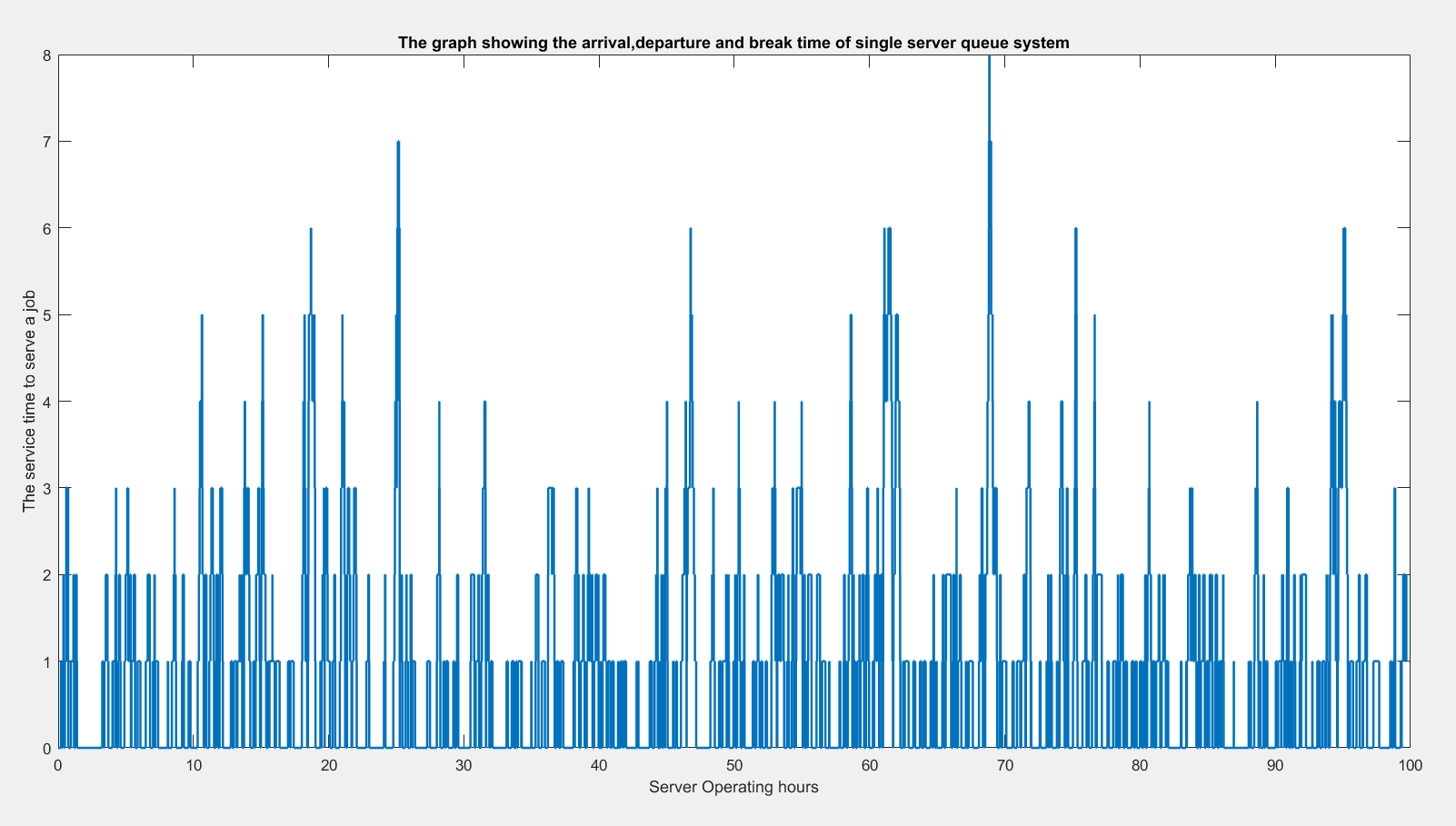
Tp = max(curr\_time-hrs\_of\_oper, 0);

End end

**Results and Theoretical Values:**

We observe that the total expected amount of time that the server is on break in the first 100 hours of operation is close to 78 hours.

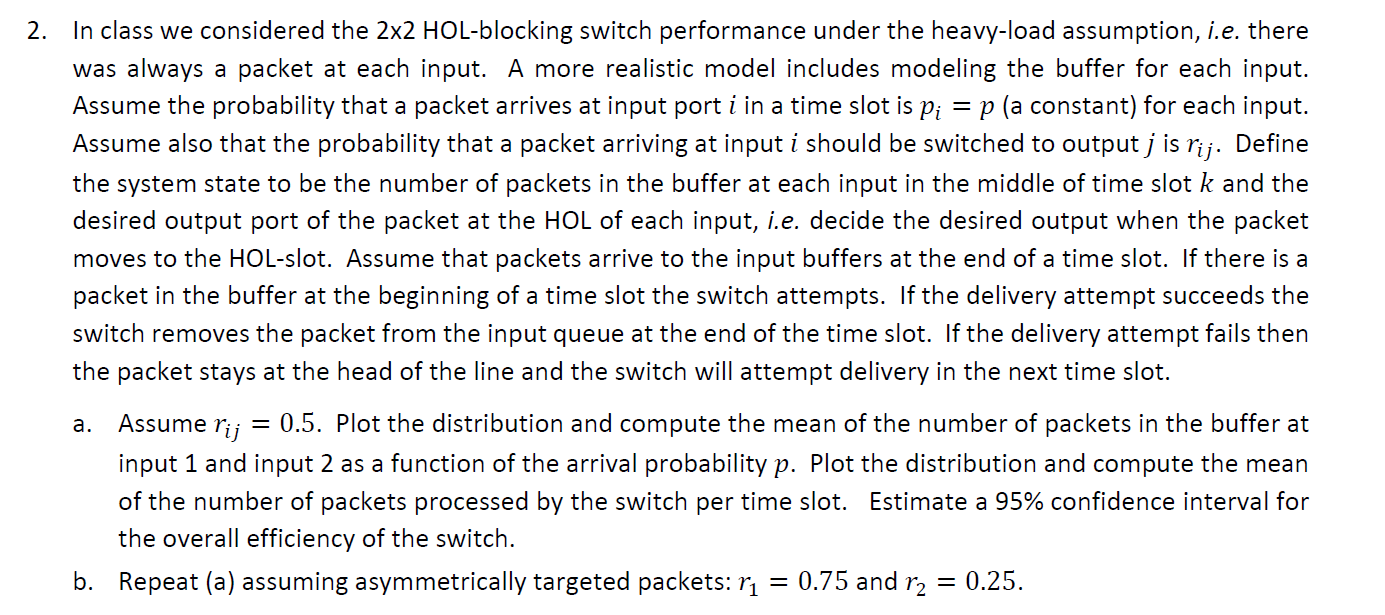
****



**Observations:**

* The expected amount of time that the server is on break in the first 100 hours of operation is observed to be in the range of 75-80 hours.
* The expected amount of time that the server is on break in the first 100 hours of operation is observed to be close to the intuitive value.
* Since in the question the service rate is greater than the maximum arrival rate, the single queue server is on break for majority of the time.
* Since the arrival rate follows the non-homogeneous Poisson distribution with maximum arrival rate is 19 and service distribution follows the exponential service rate with 25 jobs per hour, the Service rate is much faster than the arrival rate and the queue will not be as significant. Had it been uniform, intuitively, the total number of job arrived should be 1190 while in simulation it is around 535.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

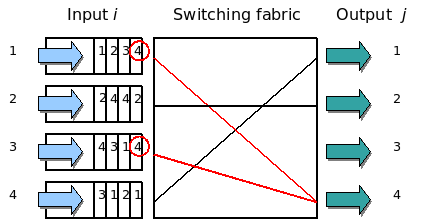


**Concept behind this problem:**

The problem uses the concept of **HOL switch** with **Markov Chains**.

A HOL switch may be composed of buffered input ports, a switch fabric and buffered output ports. If [first-in first-out](https://en.wikipedia.org/wiki/FIFO_(computing_and_electronics)) (FIFO) input buffers are used, only the oldest packet is available for forwarding. More recent arrivals cannot be forwarded if the oldest packet cannot be forwarded because its destination output is busy. The output may be busy if: 🡪 There is output [contention](https://en.wikipedia.org/wiki/Resource_contention) (see diagram) (or) 🡪 Most commonly when the output buffer is full - [congestion](https://en.wikipedia.org/wiki/Network_congestion) (for example the combined rate of multiple inputs exceeds the output rate)

Head-of-line blocking (HOL blocking) in computer networking is a performance-limiting phenomenon that occurs when a line of packets is held up by the first packet. Examples include input buffered network switches, out-of-order delivery and multiple requests in HTTP pipelining. Without HOL blocking, the new arrivals could potentially be forwarded around the stuck packet to their respective destinations. The phenomenon can have severe performance-degrading effects in input-buffered systems.



**Effect on switch performance:** This phenomenon limits the throughput of switches. For FIFO input buffers, a simple model of fixed-sized cells to uniformly distributed destinations, causes the throughput to be limited to 58.6% of the total as the number of links becomes large. HOL can significantly increase packet reordering

A **Markov chain** is a type of Markov process that has either discrete state space or discrete index set (often representing time), but the precise definition of a Markov chain varies. For example, it is common to define a Markov chain as a Markov process in either discrete or continuous time with a countable state space (thus regardless of the nature of time), but it is also common to define a Markov chain as having discrete time in either countable or continuous state space (thus regardless of the state space). It consists of a finite number of states and some known probabilities Pxy(Also called transition probabilities), where Pxy is the probability of moving from state y to state x. We have a set of states, S = {s1, s2,...,sr}. The process starts in one of these states and moves successively from one state to another. Each move is called a step. The process can either remain in the state it is(Pxx or Pyy) in or change its state(Pxy or Pyx). Also we usually specify a particular state as the starting state.

**Code and Description:**

* My implementation uses user to enter Ri and Rj so that the code is generic to handle Ri=Rj=0.5 case as well as Ri=0.75 and Rj=0.25.
* Then we define arrival rate of each of the packet at the buffer (100 samples) and choose a number from the above array in each iteration for packets arriving at buffer-1 and those of packets arriving at buffer-2.
* We then define the Probability of switching the arrived packet to Output port 1 or port 2 by choosing another rand number.
* Now Generate a random number and compare with the arrival probability. If greater, then it means that the packet has arrived. Otherwise the packet in the buffer (if there is any) will be processed.
* To move a packet at input onto output port, we choose the number and move it to output 1 or output 2 based if that number is less or greater than the packet arrival probability at each time slot.

Ri=input ('Enter the Ri');

Rj=input ('Enter the Rj');

no\_of\_time\_slots=100;

P\_arr=[1:100];

P\_arr = P\_arr./100;

ip1\_buf=0;

ip2\_buf=0;

op1\_buf=0;

op2\_buf=0;

cnt1=0;

cnt2=0;

pre\_st\_p1=0;

pre\_st\_p2=0;

for iter1=1:1:no\_of\_time\_slots

pckts\_proc=0;

if(rand()>P\_arr(iter1))% Packets arriving at Input buffer 1

ip1\_buf=ip1\_buf+1;

end

if(rand()>P\_arr(iter1))% Packets arriving at Input buffer 2

ip2\_buf=ip2\_buf+1;

end

swit\_prob=rand();

r\_number=rand();

* Now we have four possibilities of getting which packet can arrive. (0 0), (0 1), (1 0) and (1 1).
* After we process the numbers in the first-time slot, we must take the previous state information also into consideration and calculate the case for the next time slot.
* Let us say for example if we get (1 1) in the first attempt. In that case, we can only process one of them to output -2 and keep the other one in buffer. The next set can again have four cases which follows the Markov chain.

if(ip1\_buf>0 && ip2\_buf>0)

if(swit\_prob>Ri && r\_number<=Rj)% (1,0)

pckts\_proc=pckts\_proc+1;

if(pre\_st\_p1==0 && cnt1~=0)%(0,0)

cnt1=cnt1+1;

op1\_buf=op1\_buf+1;

if(rand()<=Ri)

ip1\_buf=ip1\_buf-1;

pre\_st\_p2=0;

else

ip2\_buf=ip2\_buf-1;

pre\_st\_p1=0;

end

elseif(pre\_st\_p2==1 && cnt2~=0)% (1,1)

cnt2=cnt2+1;

op2\_buf=op2\_buf+1;

if(rand()<=Ri)

ip1\_buf=ip1\_buf-1;

pre\_st\_p2=1;

else

ip2\_buf=ip2\_buf-1;

pre\_st\_p1=1;

end

else %(1,0)Both the packets will be delivered.

ip1\_buf=ip1\_buf-1;

ip2\_buf=ip2\_buf-1;

op1\_buf=op1\_buf+1;

op2\_buf=op2\_buf+1;

pckts\_proc=pckts\_proc+1;

end

end

if(swit\_prob<=Ri && r\_number>Rj)% (0,1)

pckts\_proc=pckts\_proc+1;

if(pre\_st\_p2==0 && cnt1~=0)%(0,0)

cnt1=cnt1+1;

op1\_buf=op1\_buf+1;

if(rand()<=Ri)

ip1\_buf=ip1\_buf-1;

pre\_st\_p2=0;

else

ip2\_buf=ip2\_buf-1;

pre\_st\_p1=0;

end

elseif(pre\_st\_p1==1 && cnt2~=0)% (1,1)

cnt2=cnt2+1;

op2\_buf=op2\_buf+1;

if(rand()<=Ri)

ip1\_buf=ip1\_buf-1;

pre\_st\_p2=1;

else

ip2\_buf=ip2\_buf-1;

pre\_st\_p1=1;

end

else %(0,1)Both the packets will be delivered.

ip1\_buf=ip1\_buf-1;

ip2\_buf=ip2\_buf-1;

op1\_buf=op1\_buf+1;

op2\_buf=op2\_buf+1;

pckts\_proc=pckts\_proc+1;

end

end

if(swit\_prob<=Ri && r\_number<=Rj)% (0,0)

pckts\_proc=pckts\_proc+1;

if((pre\_st\_p1==0 && cnt1~=0) || (pre\_st\_p2==1 && cnt2~=0))

ip1\_buf=ip1\_buf-1;

op1\_buf=op1\_buf+1;

ip2\_buf=ip2\_buf-1;

op2\_buf=op2\_buf+1;

pckts\_proc=pckts\_proc+1;

else %(0,0)Only one of thr packets will be delivered

cnt1=cnt1+1;

op1\_buf=op1\_buf+1;

if(rand<=Ri)

ip1\_buf=ip1\_buf-1;

pre\_st\_p2=0;

else

ip2\_buf=ip2\_buf-1;

pre\_st\_p1=0;

end

end

end

if(swit\_prob>Ri && r\_number>Rj)% (1,1)

pckts\_proc=pckts\_proc+1;

if(pre\_st\_p2==0 && cnt1~=0)||(pre\_st\_p1==0 && cnt2~=0)

ip1\_buf=ip1\_buf-1;

op1\_buf=op1\_buf+1;

ip2\_buf=ip2\_buf-1;

op2\_buf=op2\_buf+1;

pckts\_proc=pckts\_proc+1;

else %(1,1)Only one of thr packets will be delivered

cnt2=cnt2+1;

op1\_buf=op1\_buf+1;

if(rand<=Ri)

ip1\_buf=ip1\_buf-1;

pre\_st\_p2=1;

else

ip2\_buf=ip2\_buf-1;

pre\_st\_p1=1;

* Store each calculated length of input buffer-1 and buffer-2 in a vector.
* Calculate the throughput, efficiency and packets which are processed by switch for every time slot.
* We then calculate the mean of the number of packets in the buffer at input 1 and input2 as well as the mean of the number of packets processed by the switch per time slot.
* We then must estimate the bootstrap confidence interval for the overall efficiency with 95% confidence interval.

Total\_Buffer\_Size(:,iter1)=ip2\_buf+ip1\_buf;

buf\_size1(:,iter1)=ip1\_buf;

buf\_size2(:,iter1)=ip2\_buf;

%if(Packets\_Processed~=0)

Throughput(:,iter1)=(pckts\_proc);

Efficiency(:,iter1)=(Throughput(:,iter1)/2)\*100;

%end

Packets\_Processed\_Switch(:,iter1)=pckts\_proc;end

Mean\_Packets\_Processed\_Switch=mean(Packets\_Processed\_Switch);

Mean\_ThroughPut=mean(Throughput);

Mean\_Efficiency=mean(Efficiency);

disp(['The mean Efficiency = ',num2str(mean(Efficiency))]);

disp(['The mean Throughput = ',num2str(mean(Throughput))]);

disp(['The mean of the number of packets in the buffer at input 1 = ',num2str(mean(buf\_size1))]);

disp(['The mean of the number of packets in the buffer at input 2 = ',num2str(mean(buf\_size1))]);

disp(['The Mean of the total buffer size = ', num2str(mean(Total\_Buffer\_Size))]);

disp(['The Mean of the packets processed by SWITCH in each slot = ', num2str(mean(Mean\_Packets\_Processed\_Switch))]);

BOOT=bootci(no\_of\_time\_slots,@mean,Efficiency);

disp('The bootstrap confidence interval of efficiency are');

disp(BOOT);

plot(P\_arr,buf\_size1,'r',P\_arr,buf\_size2,'b--o');

title('The distribution for Prob of arrival vs the Input1-Input- 2 Buffers');

xlabel('Probability of arrival');

ylabel('The Buffer at input-1 and input-2');

legend('Input Buffer1','Input Buffer2');

**Results:**

>> ee511\_p5\_q2 **For Part-a**

Enter the Ri 0.5

Enter the Rj 0.5

**The mean Efficiency = 71%**

The mean Throughput = 1.5

The mean of the number of packets in the buffer at input 1 = 0.30

The mean of the number of packets in the buffer at input 2 = 2.0

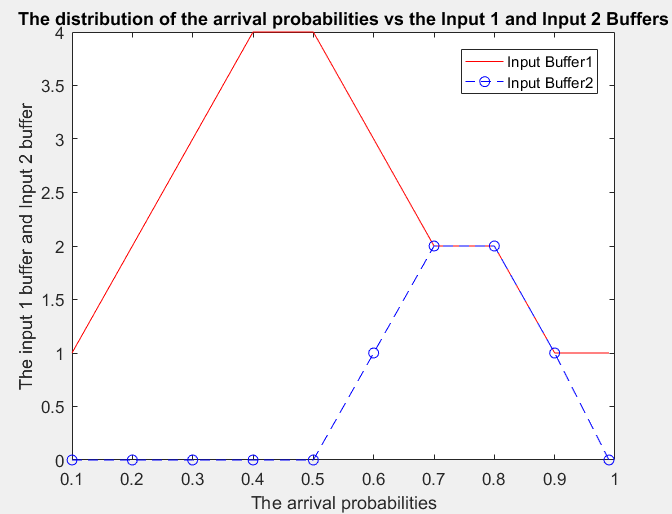
The Mean of the total buffer size = 2.3

The Mean of the packets processed by SWITCH in each slot = 0.8

The bootstrap confidence interval of efficiency are

57.0000

80.1901



>> ee511\_p5\_q2 **For Part-b**

Enter the Ri 0.75

Enter the Rj 0.25

**The mean Efficiency = 69.99%**

The mean Throughput = 1.877

The mean of the number of packets in the buffer at input 1 = 1.20

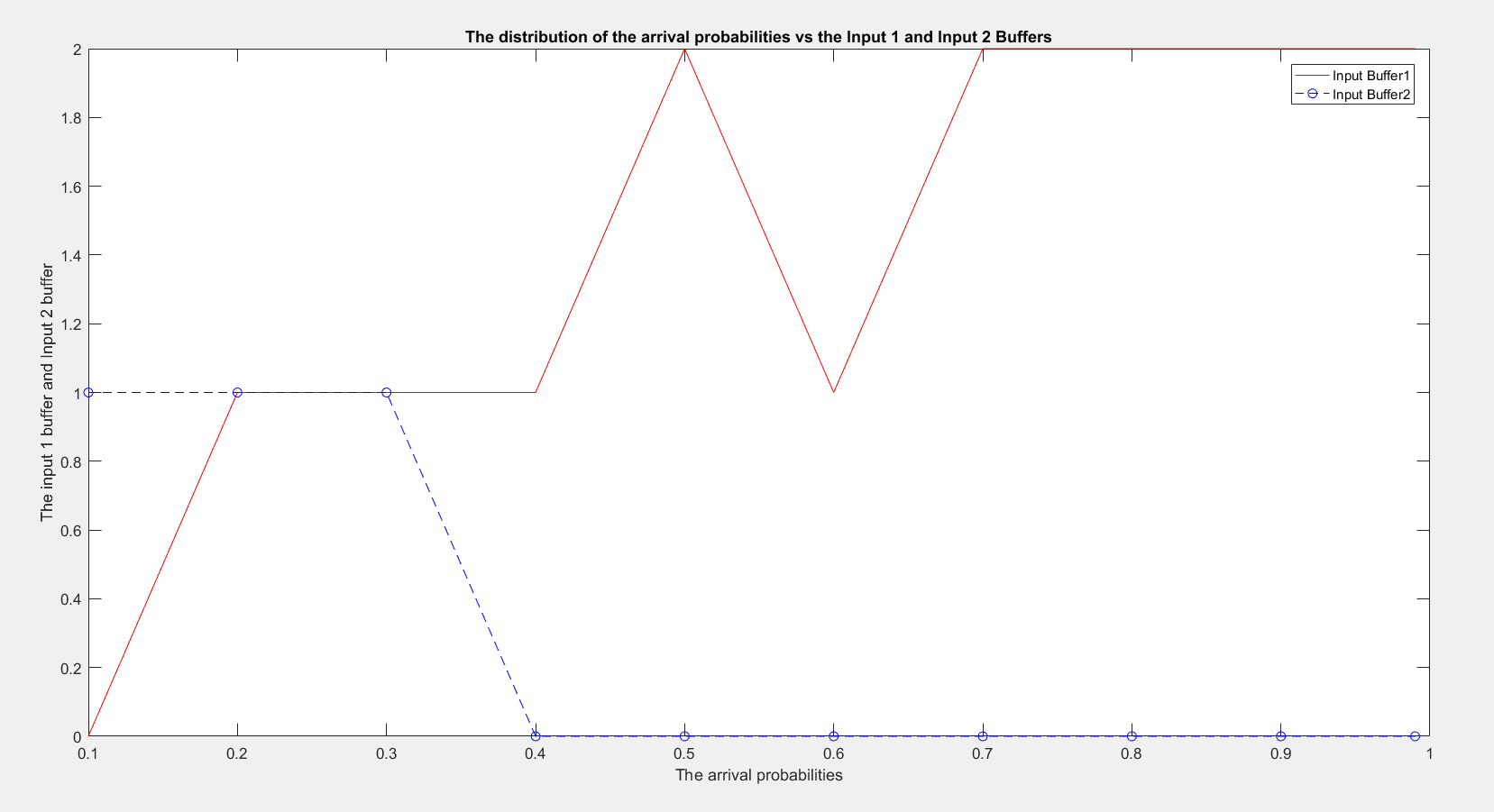
The mean of the number of packets in the buffer at input 2 = 0.10

The Mean of the total buffer size = 1.3

The Mean of the packets processed by SWITCH in each slot = 1.35

The bootstrap confidence interval of efficiency are

54.2300 73.4101



**Observations:**

* Ideally speaking, the input buffer sizes will increase with the increase in the arrival probabilities. However, the question mentions that the probability that a packet arriving at input I to be switched to j is denoted by Rij which determines the packet to be sent to output ports. So, if both the incoming packets are destined to be output at different ports then the more packets are processed while if they are destined to same port then less packets processed.
* The incoming packets destined to same or different outputs depends on the transition matrix using the Markov Chain.
* In case of Rij=0.5, the distribution of packets are symmetric .

***Theo\_efficiency in this case is = ((Rij \* (1packets/second) + Rij\*(2 packets/second))/(2 Packets /second))\*100*** = 75%.

* In case where Rij = 0.75 or 0.25, the distribution of packets is asymmetric. Given R1=0.75 and R2=0.25.

***Theo\_efficiency in this case is = ((R1 \* (1packets/second) + R2\*(2 packets/second))/(2 Packets /second))\*100*** = 62.5%.

* Thus, the efficiency is more in case of symmetric distribution of the packets.
* Also, observed that the theoretical values and the calculated values are almost similar.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

3) *The simulation of Wright Fisher model using Markov Chain for stochastic genotypic drift during successive generations. Consider a population of 􀜰 􀵌 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?*

**Reference:**

**“Noise can speed convergence in Markov chains” Brandon Franzke and Bart Kosko**

<https://mhi.usc.edu/files/2012/01/franzke_paper.pdf>

**Concept behind this problem:**

The problem uses the concept of ***Markov Chain*** *to simulate stochastic genotypic drift during successive generations proposed by* ***Wright Fisher****.*

In the Wright-Fisher model, we have N diploid individuals, that is, everyone has two copies of each allele. Generations are non-overlapping. At each generation, each allele inherits its genetic material from a uniformly chosen allele from the previous generation, independently from all other alleles. The assumptions of this model are as follows:

* Non-overlapping generations
* Constant population size: N haploid adults
* Autosomal locus segregating alleles A and a
* No selection or mutation
* Each individual in generation t + 1 chooses its parent uniformly at random and with

replacement from the N adults alive in generation t.

The **Wright-Fisher model** can be simulated in several ways. The most straightforward (but not

the most efficient) approach is to keep track of individual genotypes in each generation and to

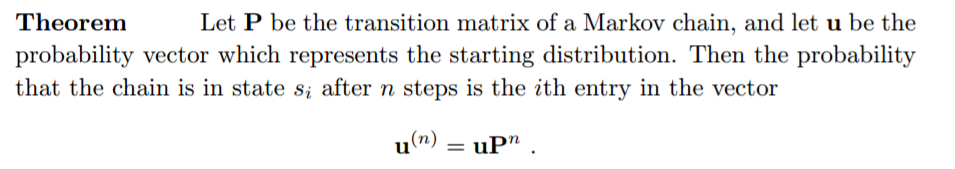
randomly sample the parent of each individual from the previous generation. Alternatively, if

we are only interested in the dynamics of the allele frequencies, then it suffices to just record

these and to generate a binomially-distributed random variable Xt+1 ∼ Binomial(2N, pt) and

set pt+1 = Xt+1/2N.

A **Markov chain** is a type of Markov process that has either discrete state space or discrete index set (often representing time), but the precise definition of a Markov chain varies. For example, it is common to define a Markov chain as a Markov process in either discrete or continuous time with a countable state space (thus regardless of the nature of time), but it is also common to define a Markov chain as having discrete time in either countable or continuous state space (thus regardless of the state space).

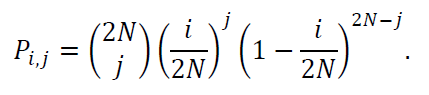


**Code & Code Description:**

* Variables are initialized.
* The number of individuals =100 and the number of generations = 5000
* A for loop is run for 7 iterations where each loop contains the below A1 A2 distribution.
  1. Loop 1-5 have 200 A1 alleles and 1 A2 alleles with the value 1 places in
     + Center of 2N+1 distribution
     + Edges of the 2N+1 distribution
     + In between 0 to 2N+1/2 and 2N+1/2 to 2N+1
  2. Loop 6-7 has an entirely different composition of A1 and A2 i.e. it has 50 A1 and 151 A2 or 100 A1 and 101 A2.

**Note**: Care was taken to make sure the summation of a row =1 in initial distribution. Hence the values are 0.02 or 0.01(1/50,1/100).

* We then create Transition Matrix P by running two for loops from 1 to 2N+1 following the distribution given in the question



* Once we have the Transition Matrix, we essentially perform the below : Output(Output{[Output[i]\*P]}P)P and so on until we reach a steady state using the function ismembertol
* What essentially steady state means is that after what generation(Iterations), the Prob[A1,A1] and Prob[A2,A2] in an individual stabilizes and what is the distribution of those probabilities.

warning off;

clear all; close all; clc;

no\_of\_ind = 100; % Number of individuals

num\_gen=5000; % Number of Generations

output=zeros(num\_gen+1,2\*no\_of\_ind+1); % clear out any old values

t=0:num\_gen; % time indices

for iter=1:7

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

%0's represent A1 and 1's represent A2 Alleles

if iter == 1 %1 is in the middle(101 position) of 201 numbers

position = 101;

elseif iter == 2 %1 is at the beginning of 201 numbers

position = 1;

elseif iter == 3 %1 is at the end of 201 numbers

position = 201;

elseif iter == 4 %1 is in the 155th position of 201 numbers

position = 155;

elseif iter == 5

position = 55;

elseif iter == 6

position = 50;

input([1:position],1) = 0.02;

else

position = 100;

input([1:position],1) = 0.01;

end

if iter<6

input(position,1) = 1;

disp(['The steady state for the initial distribution with 200 A1 and 1 A2 where 1 is placed in the ',num2str(position),' position is'])

else

disp(['The steady state for the initial distribution with ',num2str(position),' A2 and the remaining A1 is '])

end

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

%Creation of transition matrix using the formula specified in the question

trans\_matrx=zeros(2\*no\_of\_ind+1,2\*no\_of\_ind+1);

for iter\_i = 1:2\*no\_of\_ind+1

for iter\_j = 1:2\*no\_of\_ind+1

trans\_matrx(iter\_i,iter\_j) = nchoosek(2\*no\_of\_ind,iter\_j-1)\*((iter\_i-1)/(2\*no\_of\_ind))^(iter\_j-1)\*(1-(iter\_i-1)/(2\*no\_of\_ind))^(2\*no\_of\_ind-iter\_j+1); end end

for iter\_steady\_st=1:num\_gen,

output(iter\_steady\_st+1,:) = output(iter\_steady\_st,:)\*trans\_matrx;

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

steady\_value = ismembertol(output(iter\_steady\_st+1,:),output(iter\_steady\_st,:));

if all(steady\_value == 1)

st\_st = iter\_steady\_st;

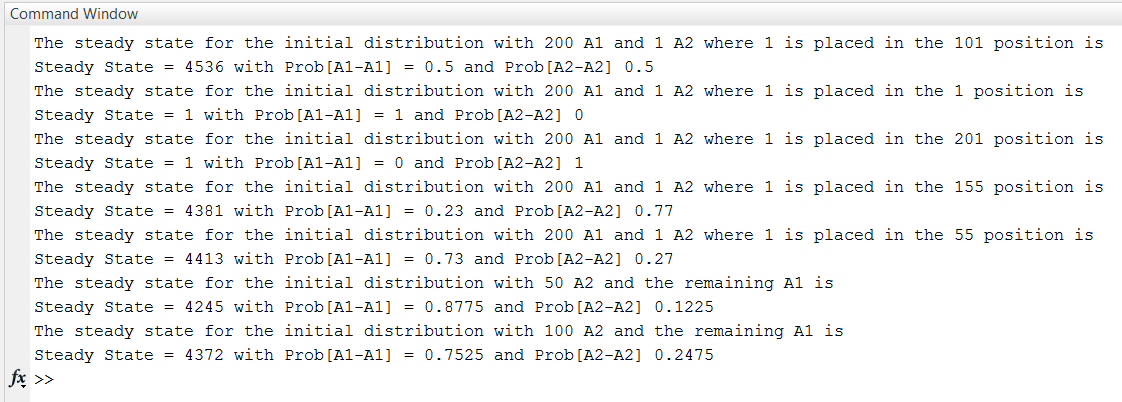
break; end end

disp(['Steady State = ',num2str(iter\_steady\_st),' with Prob[A1-A1] = ',num2str(output(st\_st,1)),' and Prob[A2-A2] ',num2str(output(st\_st,201))])

end

**Results:**

* With the Initial state of 200 A2 and 1 A1 alleles (1 in the middle of the distribution), we observe that the steady state reaches at 4536th generation.
* The above simulation shows that P[steady state = (A1,A1)] = 0.5 = P[steady state = (A2,A2)].
* If we run the above in a loop of 100 times, we observe that the value is constant with steady state ~4536.
* If we run with different composition, we find the below behavior
  + Placing 1 in the middle has the highest number of generation run(4536) to stabilize or reach the steady state of P[A1-A1]=P[A2-A2]=0.5.
  + Placing 1 in the first position of the 201 alleles as initial distribution is resulting in attainment of steady state of P[A1-A1]= 1 & P[A2-A2]=0.
  + Placing 1 in the last position of the 201 alleles as initial distribution is resulting in attainment of steady state of P[A1-A1]= 0 & P[A2-A2]=1.
  + Placing 1 in between 155th position is resulting in the number of generation runs to reach steady state of P[A1-A1]= 0.23 P[A2-A2]=0.77 as 4381
  + Placing 1 in between 55th position is resulting in the number of generation runs to reach steady state of P[A1-A1]= 0.77 P[A2-A2]=0.23 as 4413
* If at all we use different A1 and A2 composition as initial state, we observe that the steady state is reached almost same as that of previous one. (additional exercise)
  + When we take 151 A1 and 50 A2 composition i.e. X150 (t) as initial state for observing the behavior, we see that the steady state is reached in 4244 generations run of Markov chain process with P[A1-A1] = 0.8775 and p[A2-A2] = 0.1225.
  + Similarly, when we take 101 A1 and 100 A2, we see steady state is reached in 4371 generations run of Markov chain process with P[A1-A1] = 0.7525 and p[A2-A2] = 0.2475.



**Observations:**

* In case N is large, say N ≥ 10000, even this approach may be too slow for many purposes. In this case, we can replace the binomial distribution by a pair of approximations based on the central limit theorem and the law of rare events to compute the n choose k values.
* The simulation tells us that after few generations, there will be generations whose individuals will have either A1-A1 alleles or A2-A2 alleles based on the probability distribution above.
* If we are taking only one A2, then Placing 1 matters a lot in the distribution where placing it at the beginning will lead to generations (after n iterations) where there might not be any A2-A2 composition at all. The steady state for this example is homozygous (A2,A2) because future generations can no longer inherit the extinct A1 gene. Similarly, if we place the 1 at the end, then A1-A1 probability after few generations might be lesser.
* **Perron–Frobenius theorem:** Asserts that a real square matrix with positive entries has a unique largest real eigenvalue and that the corresponding eigenvector can be chosen to have strictly positive components, and asserts a similar statement for certain classes of nonnegative matrices.
  + The Perron frobenius theorem mentions that the eigenvector can take positive values while if we compute eigenvector (using eig in matlab), we see value ‘0’ as well.
* **Markov Chain Ergodic Theorem:** A Markov chain is called an ergodic chain if it is possible to go from every state to every state (not necessarily in one move) i.e. 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). 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.
  + In our case, providing different initial states does not lead us to the same steady state i.e. if we pick an initial state, it might not have traversed(ergodic) all the possible states, choosing another state in this simulation does not lead us to travelling all the possible states.

Hence this scenario seems to defy the assertions of the Perron‐Frobenius theorem and the

Markov chain ergodic theorem