EE511 ZHANG Fan USCID: 1417-68-5115

Q1:

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.

Analysis:

- Job arrival:
  - Nonhomogeneous Poisson process:
    - $\lambda=\begin{cases}3t+4, & 5i>t\geq 10(i-1)\\ 34-3t, & 10i\geq t\geq 5i,\quad i=1,2,3\ldots\end{cases}$
    - λ_{max}=19
  - Algorithm for Nonhomogeneous Poisson process
    - 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(in this case, we exclude $\mathrm{Ts}(1)$:
    1. Let $t=0$
    2. Generate $U_1\sim U[0,1]$
    3. Let $t=t-\frac{1}{\lambda}(logU_1)$ % generate random time interval for t
    4. Generate $U_2\sim U[0,1]$
    5. If $U_2\leq\frac{\lambda(t)}{\lambda}, set\ T_s=t\ and\ stop$ % ??
    6. Go to step2
  - T_s: the time from the first time the job arrivals(exclude $\mathrm{Ts}(1)$)
- Job service:
  - $s=\lambda e^{-\lambda t}, \lambda=1/25,\ 0\leq t<1$
  - We can substitute t with T_s
- Waiting time if no job
  - U~(0,0.3)
- Algorithm
  - Define $\mathrm{interval(i)}=\mathrm{Ts(i)}-\mathrm{Ts(i-1)}$, this is the time between two job arrivals.
  - Define remainder(i)=interval(i)-s(i)+remainder(i-1), this is the time when the server doesn't have any job
  - Define R=U~(0,0.3)
  - Remainder(i)=remainder(i-1)-U~R, sum=sum+R, until remainder(i)<0
  - i=i+1 and go back to the step two

Theoretically:

- The average λ for 100 hours is 11.5. That means. So Theoretically there will be 1150 jobs during 100 hours.
- For the service, the exponential distribution, the average serve-time is $\frac{1}{\lambda} = \frac{1}{25}$. So assume there are 1150 jobs, it means that it will spend $\frac{1150}{25} = 46$ hours to serve all of the jobs. Therefore, for the waiting time, it should be 100-46=54 hours.

Code:

```
clear
close all; clc;
% initial value of all variables
x = 0; % 50 products on hand    ???  x=0
y = 0;%number  ----job number
Ts = 0; %time interval
T = 100; % the total time we want to analyze
i = 1;
while(Ts(i) < T)dn

    % generate nonhomogeneous Poission event
    % lamda(t) = is the time !! for one job
    lambda = 11.5;
   t=0;
    while(t<T)
       u1 = rand ();
       t = Ts(i) - 1/lambda*log(u1);%why log?
       u2 = rand();
       if(mod(t,10)<5&&mod(t,10)>0)
           %pare=11111
           pare=(3*mod(t,10)+4)/lambda;
       else
           pare=(34-3*mod(t,10))/lambda;
       end
       if (u2 <= pare)% ok, draw a digram and you will understand
           Ts(i+1) = t;
           i = i + 1;
           break;
       end
    end
end
Ts(end)=100
interval=diff(Ts)
remainder=interval(1);
sum=0;

for i=2:length(Ts)-1
while(remainder>0)
    r=0.3*rand();
    remainder=remainder-r;
    sum=sum+r;
end
remainder=interval(i)-exprnd(1/25)+remainder
end
sum
```

Results: 24

```
sum =

   22.1097
```

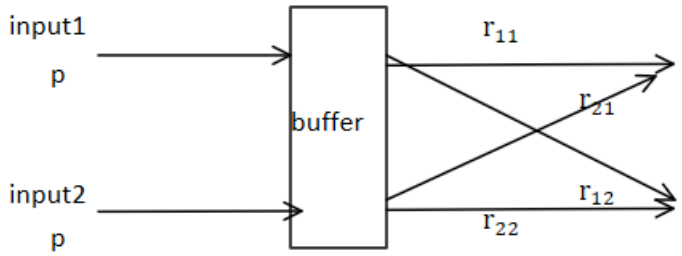
$\lambda_{max} = 19$

Comment:

- The simulation result is roughly equal to the theoretical value. So it can be considered as reasonable.

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 $r_2 = 0.25$.

Analysis:



The algorithm is as follow:

- let u1=rand() and u2=rand();
- when u1<p, buffer(2)=buffer(1)+1; when u2<p, buffer(2)=buffer(2)+1
- there are three condition:
  - input1->output1, input2->output2, this condition can transfer 2 packet at one time
  - both buffer(1) and buffer(2) don't have any packets, then the transfer 0 packet at one time
  - other conditions, including either buffer1 or buffer2 has a packet or collision occurs, then transfer 1 packet
- calculate the mean of the packets that is transferred each time slot

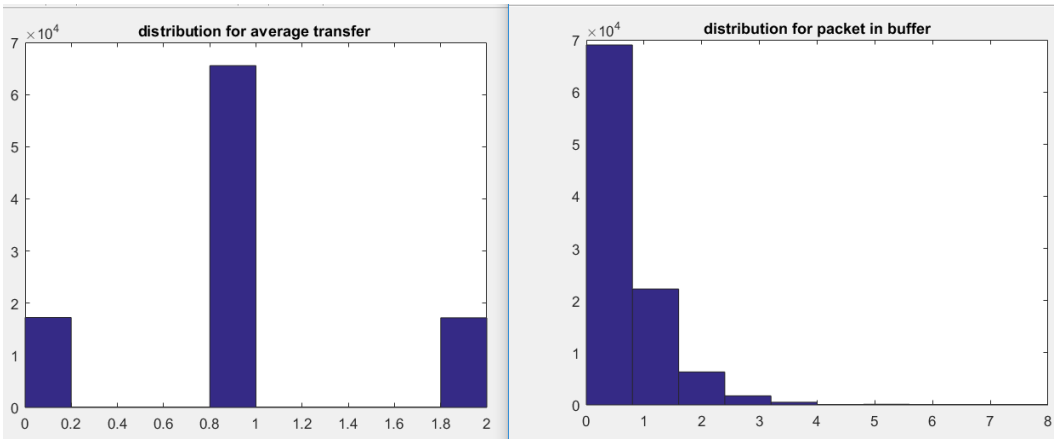
code for calculate the distribution and internal with a given p:

```
clc;clear
x=0;mean_buffer_temp=0;mean_buffer_temp2=0;
p=1
r=1/2
buffer=[0,0];%initial
```

```
out=[0,0];n=100000;sum_out=0;
    for i=1:n
    u1=rand();u2=rand();
        if(u1<p)
            buffer(1)=buffer(1)+1;
        end

        if(u2<p)
            buffer(2)=buffer(2)+1;
        end

    u3=rand();%the propability for input 1 to out 1, and input to out 2 is 1-u3
    u4=rand();%the probaility for input 2 to out 1, and input to out 2 is 1-u4
    if(buffer(1)~=0)
        if(u3<r)
            out(1)=out(1)+1;
        else
            out(2)=out(2)+1;
        end
        buffer(1)=buffer(1)-1;
    end

    if(buffer(2)~=0)
        if(u4<r)
            out(1)=out(1)+1;
        else
            out(2)=out(2)+1;
        end
        buffer(2)=buffer(2)-1;
    end

    %mean of per slot
        if(out(1)==1&&out(2)==1)%1+1
            sum_out=sum_out+2;
        elseif(out(1)==0&&out(2)==0)
            sum_out;
        else
            sum_out=sum_out+1;
        end

        %%collision!
        if(out(1)==2)%detect collision in out1
            u5=rand();%randomly choose one to send
            if(u5<1/2)
                buffer(1)=buffer(1)+1;
            else
                buffer(2)=buffer(2)+1;
            end
               out(1)=out(1)-1;
        end

        if(out(2)==2)%detect collision in out1
            u6=rand();%randomly choose one to send
            if(u6<1/2)
                buffer(1)=buffer(1)+1;
            else
                buffer(2)=buffer(2)+1;
            end
            out(2)=out(2)-1;
        end
    distribution_out(i)=sum(out);
    distribution_buffer(i)=sum(buffer);
    out=[0,0];
    mean_buffer_temp=mean_buffer_temp+buffer(1);
    end
    figure(1)
    hist(distribution_out);
    title('distribution for average transfer ')
    figure(2)
    hist(distribution_buffer);
    title('distribution for packet in buffer')
```

```
    p
    r
    z_afa = norminv([0.025 0.975],0,1);
    efficiency= distribution_out/2;
    MeanBuffer=mean(distribution_buffer)
    MeanProccesed=mean(efficiency)
    StdAll=std(efficiency);
BoundAll(1)=MeanProccesed-z_afa(2)*StdAll/sqrt(10000);
BoundAll(2)=MeanProccesed+z_afa(2)*StdAll/sqrt(10000);
BoundAll
```

Distribution for p=0.5, time slot number = 10000



The confidence for p=1, n=10000

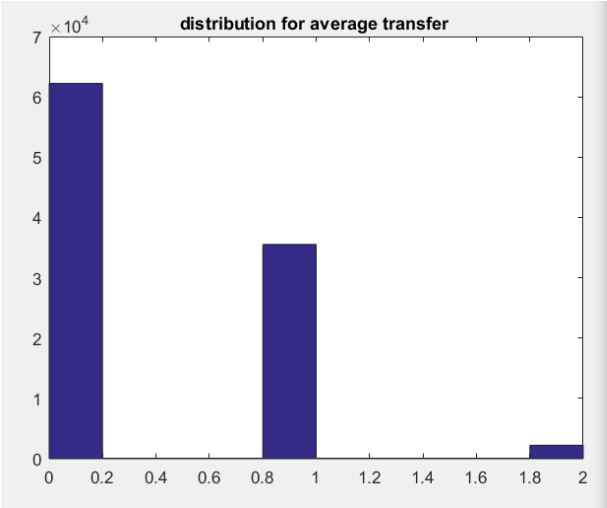
Confidence interval for p=0.2 is [0.4939,0.5054]

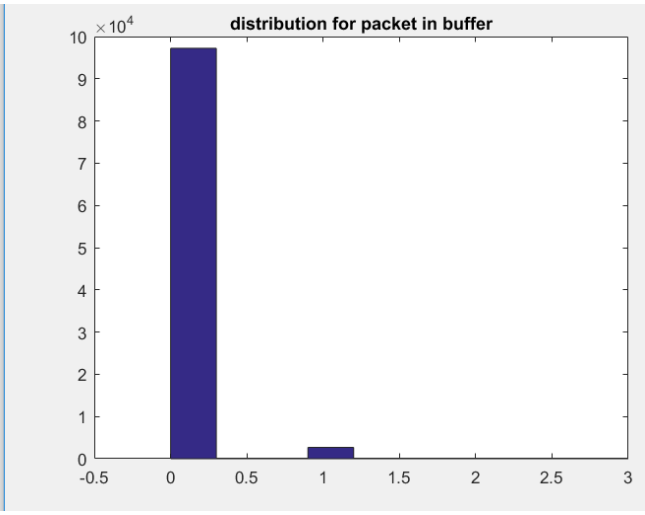
The mean of the number of the packet processed is 0.9992: and the effifiency is 0.4996

The mean of the packet in the buffer is 0.4309.

```
p =

    0.5000

r =

    0.5000

MeanBuffer =

    0.4309

MeanProccesed =

    0.4996

BoundAll =

    0.4939    0.5054
```

Distribution for p=0.2, time slot number =10000:





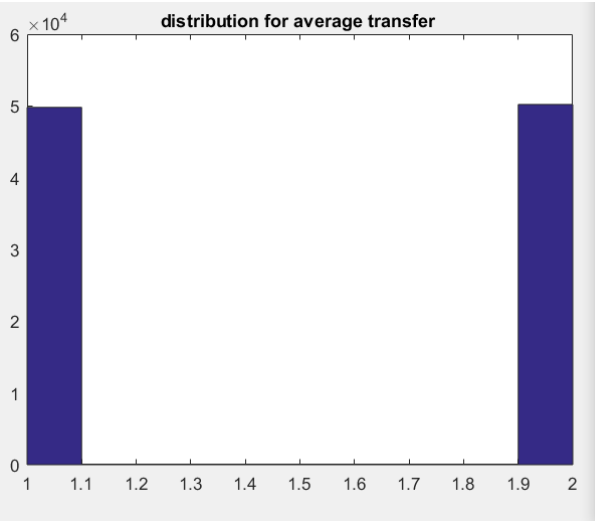
Confidence interval for p=0.2 is [0.1947,0.2052]

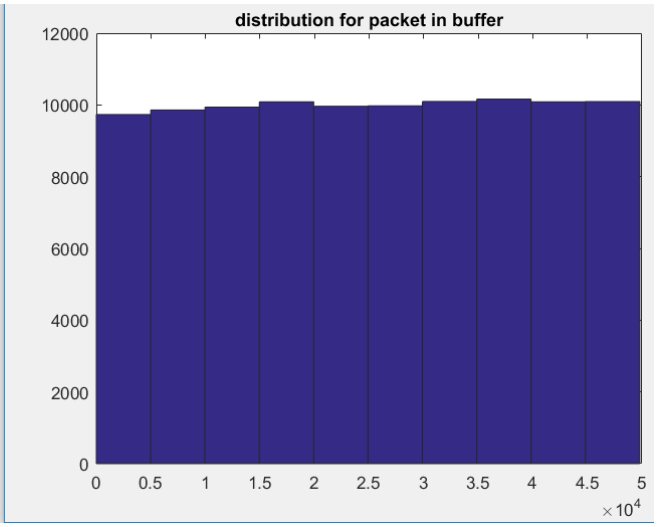
The mean of the number of the packet processed is 0.3998: and the efficiency is 0.1999

The mean of the packet in the buffer is 0.0288.

```
p =

    0.2000


r =

    0.5000


MeanBuffer =

    0.0288


MeanProccesed =

    0.1999


BoundAll =

    0.1947    0.2052
```

Distribution for p=1, time slot number =10000:






```
Confidence interval for p=0.2
```

Confidence interval for p=0.2 is [0.7461,0.7559]

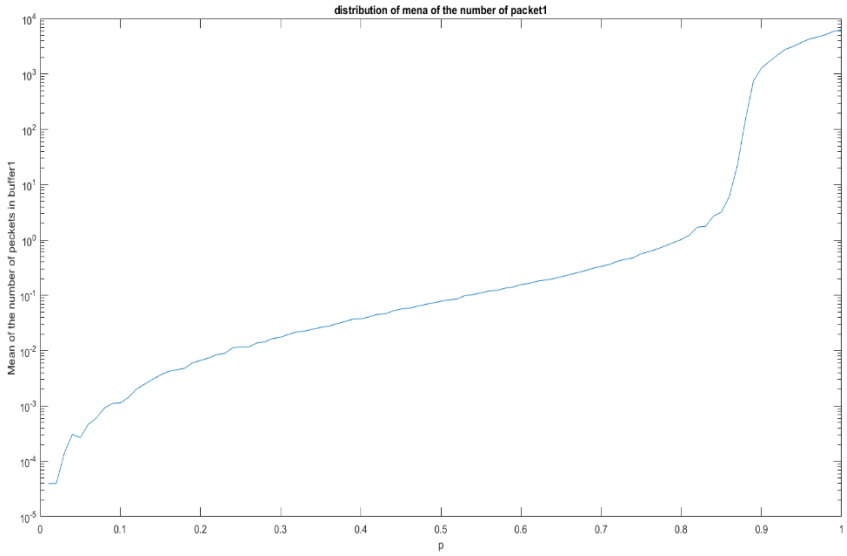
The mean of the number of the packet processed is 1.520: and the effieincy is 0.7510

The mean of the packet in the buffer is 2.5e+05, which is not convergent.

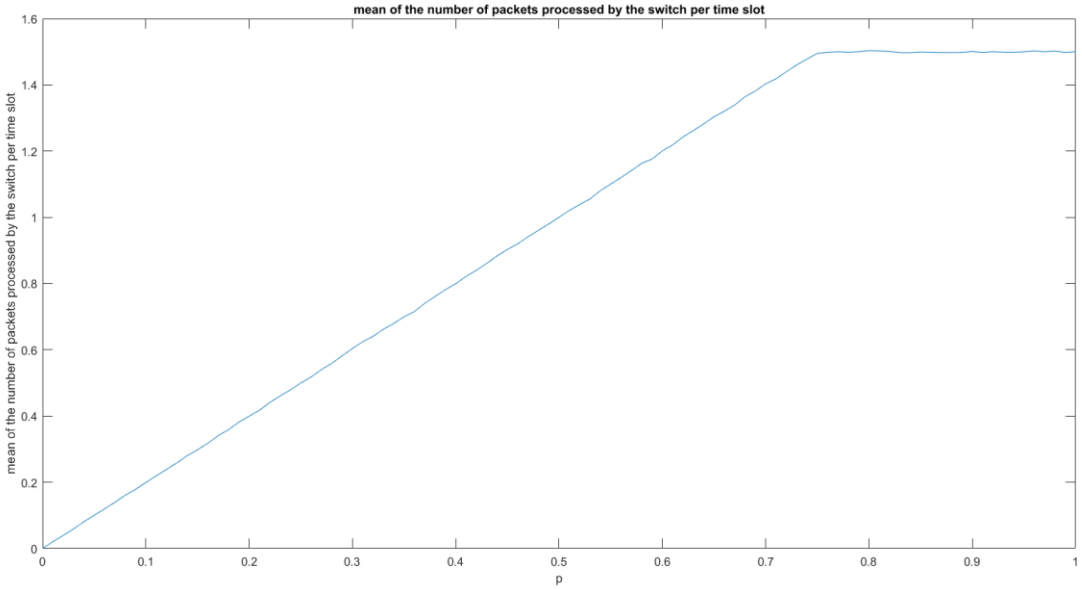
```
p =

     1


r =

    0.5000


MeanBuffer =

   2.5058e+04


MeanProccesed =

    0.7510


BoundAll =

    0.7461    0.7559
```

Then we simulate the mean of the number of packet in buffer 1 with p:

The mean of the number of packet in buffer2 with the different p:



The mean of the number of packet processed by the switch per time slot with the different p:



Comment:

In this case, time slot number t=10000 is selected.

We can conclude that, the mean of the number of packets processed by the switch per time slot increase linearly with the p, and at last it is a steady value 1.5 when p>0.88.

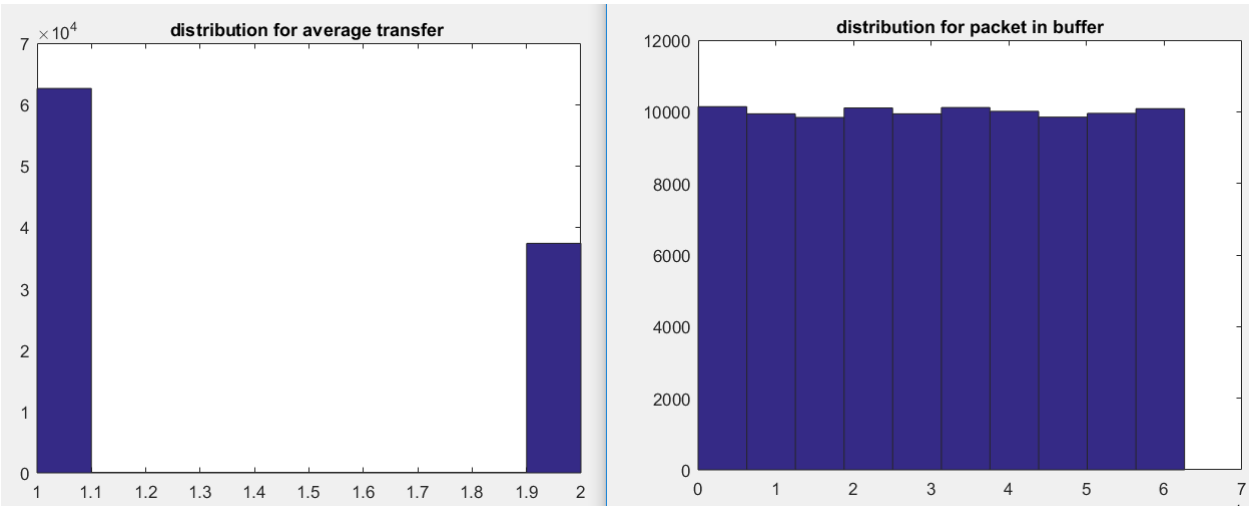
And the mean packets in the buffer will increase with the time slot when p is large enough; and while it is zero when p is small. The bound is also around 0.88 .

Q2(b). analysis:

The algorithm in question 2 is the same is that in question a. Revise the parameter to let $r_1 = 0.75$ and $r_2 = 0.25$, we obtain the following results:

Mean of the packet number in buffer with the 0<P<1 and the mean of the packet number transferred with the 0<p<1.

Distribution for p=1, n=1000 and r1 =0.25, r2=0.75



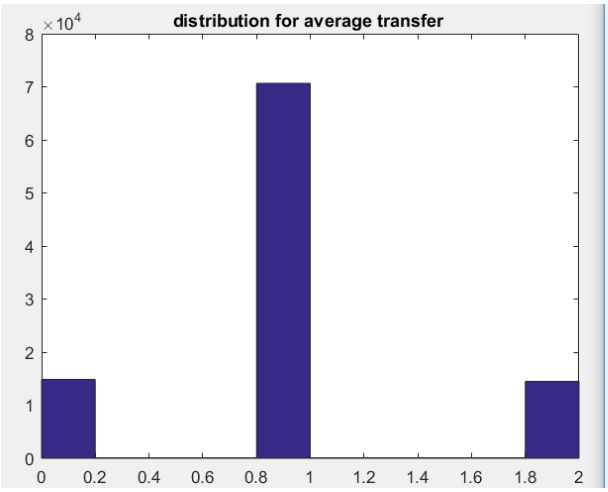
The confidence interval for p=1 is [0.6815,0.6910]

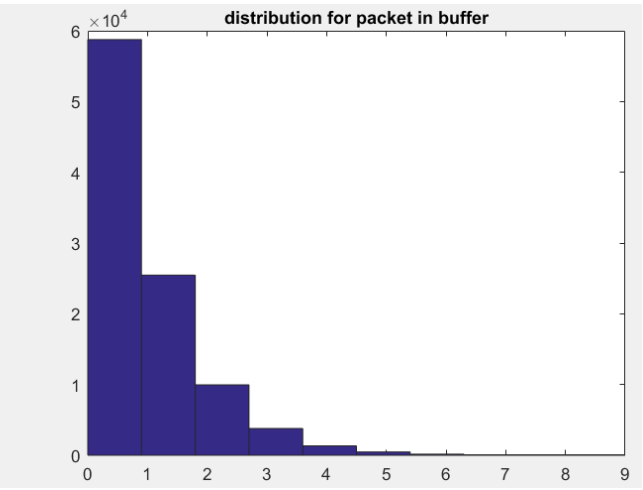
The mean of the packet number in the buffer is 3.1347e+04, which is not convergence

And the mean of the packet processed is 0.6863*2=1.3626, and the effifiency is 0.6863

```
p =

     1


r =

    0.2500


MeanBuffer =

   3.1347e+04


MeanProccesed =

    0.6863


BoundAll =

    0.6815    0.6910
```

Distribution for p=0.5, n=10000 and r1 =0.25, r2=0.75





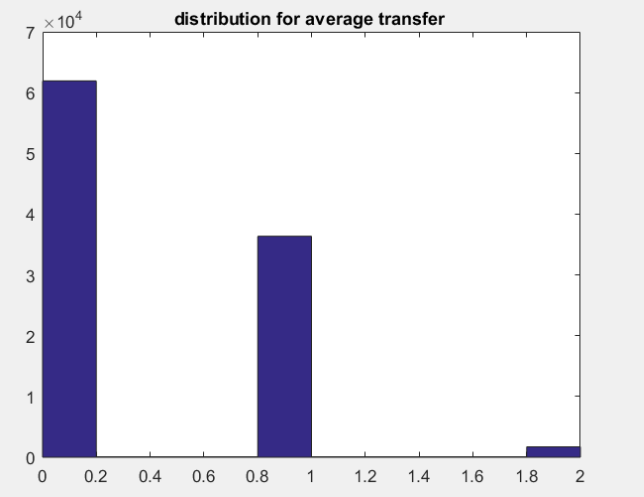
Mean of the transferred(processed) packet number=0.9976, and the efficiency is 0.4988

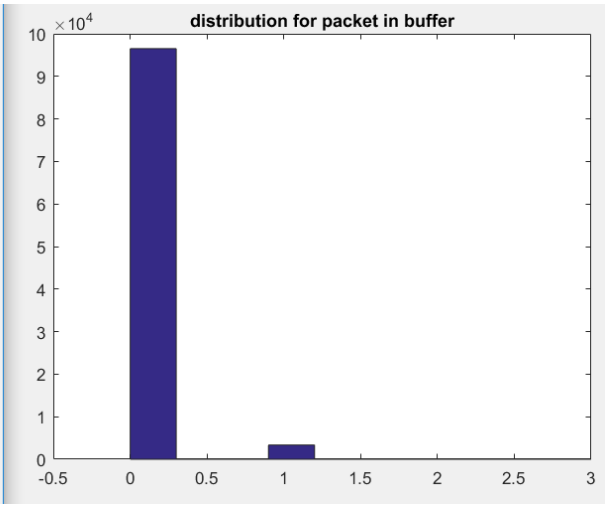
The confidence interval for p=0.5 is [0.4935,0.5041]

Mean of the packet number in the buffer is : 0.6546

```
r =

    0.2500


p =

    0.5000


r =

    0.2500


MeanBuffer =

    0.6546


MeanProccesed =

    0.4988


BoundAll =

    0.4935    0.5041
```

Distribution for p=0.2, n=10000 and r1 =0.25, r2=0.75



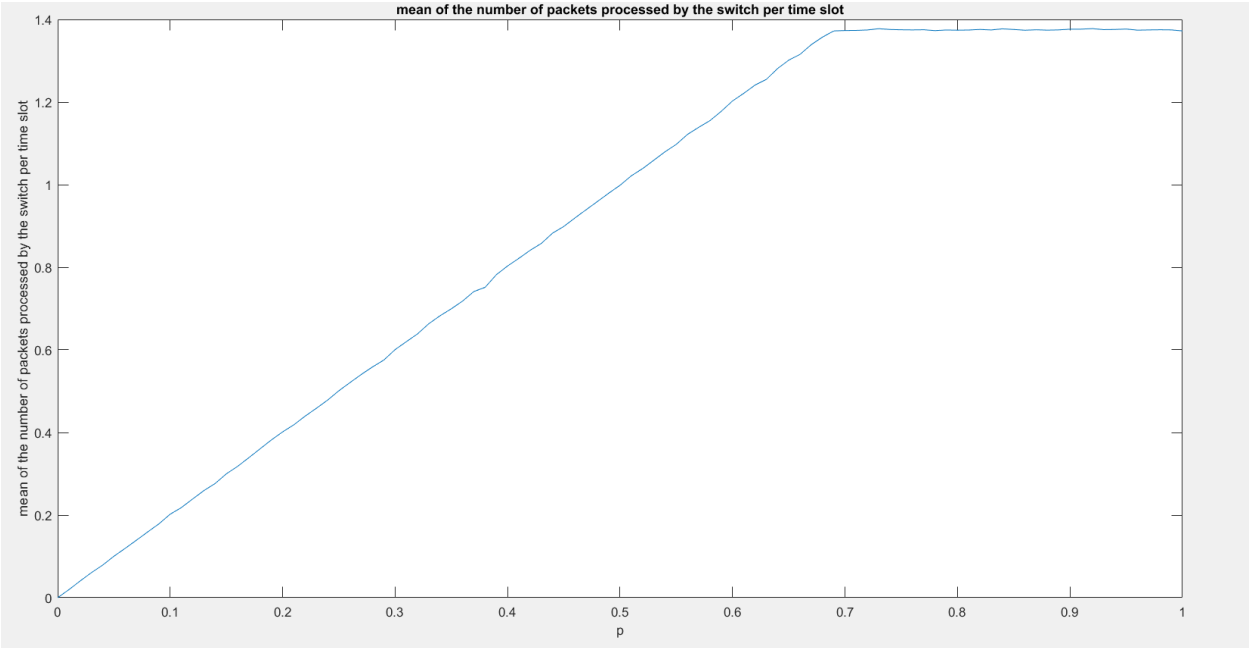



```
p =

    0.2000


r =

    0.2500


MeanBuffer =

    0.0368


MeanProccesed =

    0.1997


BoundAll =

    0.1946    0.2049
```

The mean of the packet in the buffer is 0.0368

Mean of the proccesed packet number=0.3994, and the efficiency is 0.1994

The condifence interval for p=0.2 is [0.1946,0.2049]



Comment:

The different from the question 1 is that steady mean of transferred packet changes to 1.4 while the original number is 1.5.

Compare the question **a** and **b**, we can conclude that the more symmetrical it is to transfer the packet, the higher efficiency it will be. Since the more symmetrical, the lower probability that the collision will occurs, which can results in higher frequency to transfer 2 packets without collision.

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.

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:

$$x_0(t) = P\,[0 \text{ copies A1}, 2N \text{ copies A2}]$$
$$x_1(t) = P\,[1 \text{ copies A1}, 2N - 1 \text{ copies A2}]$$

$$x_2(t) = P\,[2 \text{ copies A1}, 2N-2 \text{ copies A2}]$$
$$\ldots$$
$$x_{2N}(t) = P\,[2N \text{ copies A1}, 0 \text{ copies A2}]$$

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

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

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?

Algorithm:

When N=100, 201 condition is considered as the statement shows: $x_{1(t)}, x_{2(t)}$…., select the initial state like [0…0,1], [0..0,1,1]…[1,0,0…0],[0,1,0,.0]. All of them are 201 dimensions.

Create the transition matrix as the statement of question shows.

Each time it changes, which can be denoted by the Output(i+1)=output(i), output(1)=input;

Code: the code is revised from the one that professor shows in the piazza

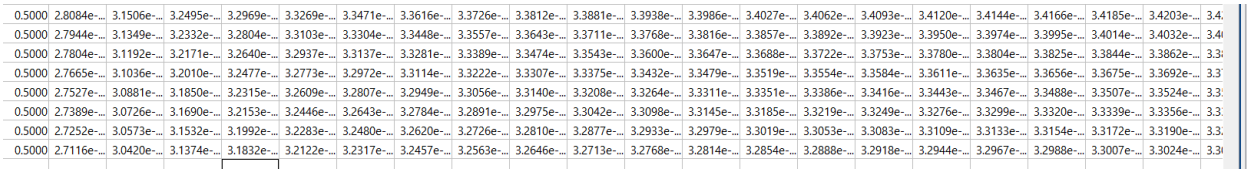
```
% Program to simulate a Markov chain
% The program assumes that the states are labeled 1, 2, ...
% Below is a sample, which you can change them according to the project
 clear all; close all; clc;
input=eye(201)
 N = 100;          % number of individuals
% transition matrix
 P=zeros(2*N+1,2*N+1);
 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
 for state=1:201
n=2000;           % 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(state,:); % generate first output value
i = 0;
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 = ismembertol(output(i+1,:),output(i,:));
  if all(LIT == 1)
      break;
  end
        steady(state,:)=output(end,:);
end
 end
 plot(steady(:,1)')
 hold on
 plot(steady(:,end)')
 legend('(200A1,0A2)','(200A2,0A1)')
 xlabel('initial state where the "1" locates');
 title('propability for state 1 (200A1, 0A2)] and (0A1, 200A2)');
ylabel('propability');
```

results and comment:

The steady result depends on the initial input:$[k_1, k_2, \ldots k_{200}, k_{201}]$

All of the initial condition will result in a steady condition: [200A1, 0A2] or [0A1, 200A2]. E.g. The following result is when the copies of A1 are the same as the copies of A2, that is [100 A1, 100A2], which is state 101($k_1 \sim k_{100} = 0, k_{101} = 1, k_{102} \sim k_{201} = 0$). The steady result is as follows:

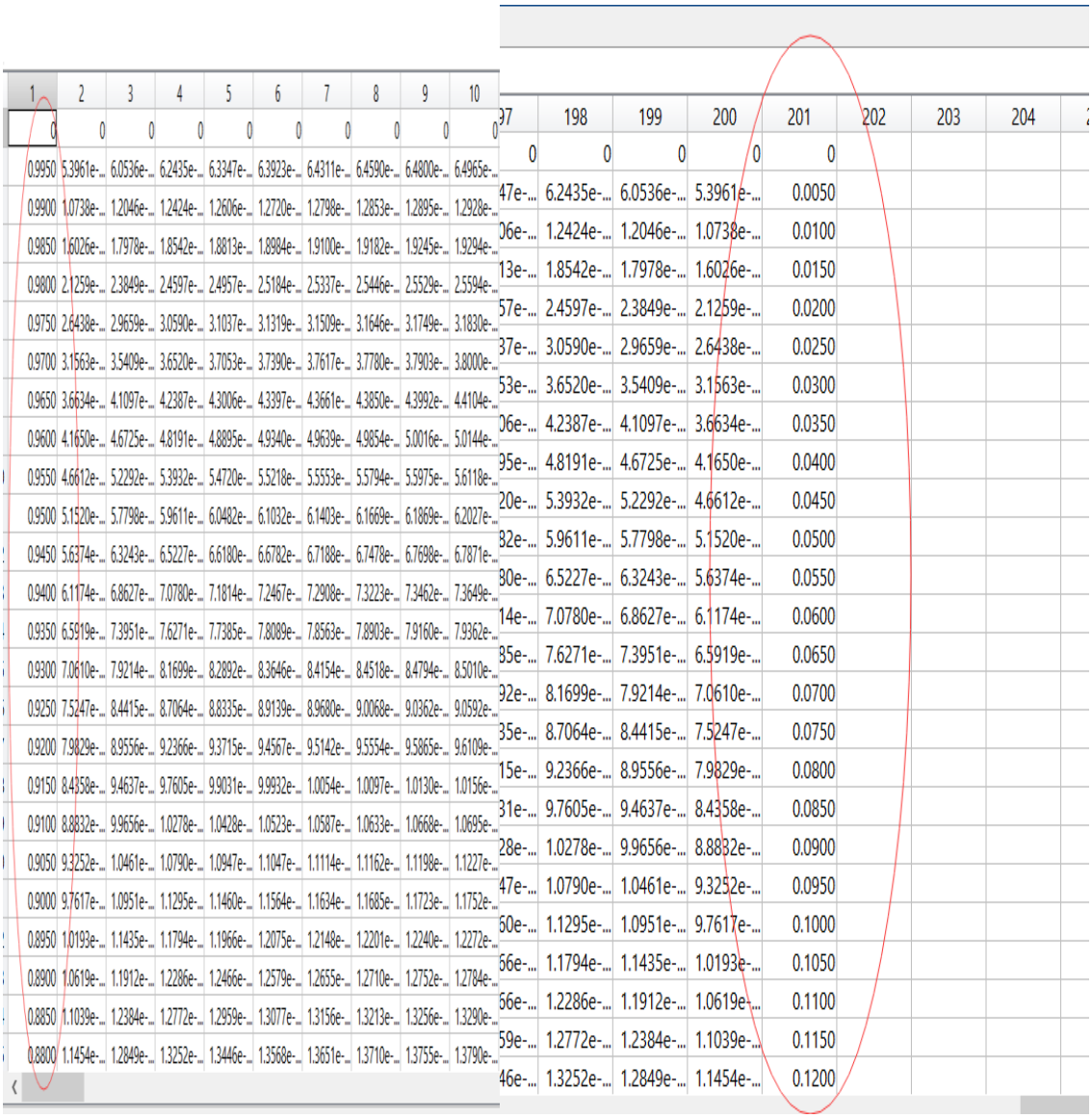


initial input = [00000..1]->steady result [00000,…1];

initial input = [1,0…..0]->steady result [1,0000…00];

initial input = [0,1,0,00000]->steady result [0.9950,000000,0.005]

initial input=[0000…0,1,0]->steady result [0005,000000,0.9950]

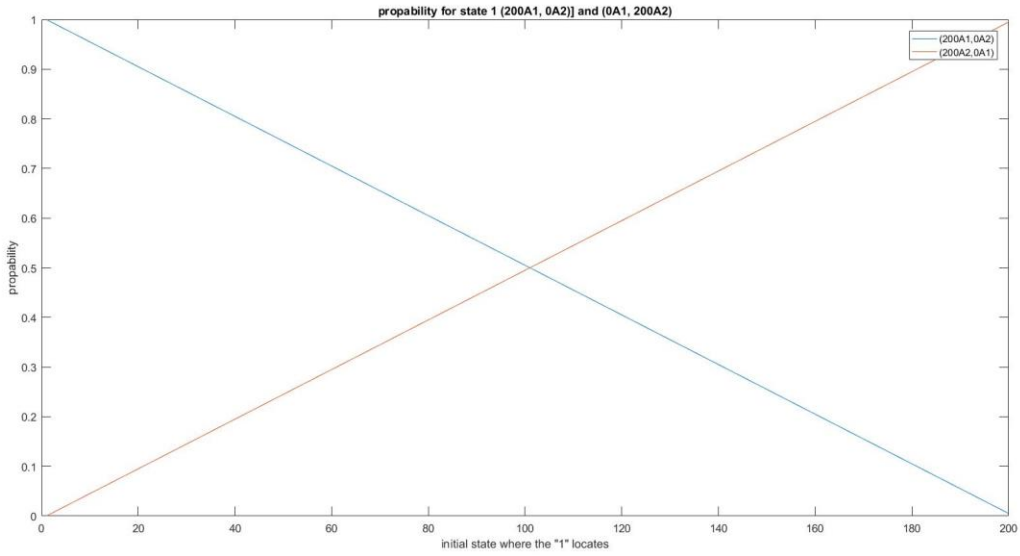


1	2	3	4	5	6	7	8	9	10	197	198	199	200	201	202	203	204
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
0.9950	5.3961e-...	6.0536e-...	6.2435e-...	6.3347e-...	6.3923e-...	6.4311e-...	6.4590e-...	6.4800e-...	6.4965e-...	47e-...	6.2435e-...	6.0536e-...	5.3961e-...	0.0050			
0.9900	1.0738e-...	1.2046e-...	1.2424e-...	1.2606e-...	1.2720e-...	1.2798e-...	1.2853e-...	1.2895e-...	1.2928e-...	06e-...	1.2424e-...	1.2046e-...	1.0738e-...	0.0100			
0.9850	1.6026e-...	1.7978e-...	1.8542e-...	1.8813e-...	1.8984e-...	1.9100e-...	1.9182e-...	1.9245e-...	1.9294e-...	13e-...	1.8542e-...	1.7978e-...	1.6026e-...	0.0150			
0.9800	2.1259e-...	2.3849e-...	2.4597e-...	2.4957e-...	2.5184e-...	2.5337e-...	2.5446e-...	2.5529e-...	2.5594e-...	57e-...	2.4597e-...	2.3849e-...	2.1259e-...	0.0200			
0.9750	2.6438e-...	2.9659e-...	3.0590e-...	3.1037e-...	3.1319e-...	3.1509e-...	3.1646e-...	3.1749e-...	3.1830e-...	37e-...	3.0590e-...	2.9659e-...	2.6438e-...	0.0250			
0.9700	3.1563e-...	3.5409e-...	3.6520e-...	3.7053e-...	3.7390e-...	3.7617e-...	3.7780e-...	3.7903e-...	3.8000e-...	53e-...	3.6520e-...	3.5409e-...	3.1563e-...	0.0300			
0.9650	3.6634e-...	4.1097e-...	4.2387e-...	4.3006e-...	4.3397e-...	4.3661e-...	4.3850e-...	4.3992e-...	4.4104e-...	06e-...	4.2387e-...	4.1097e-...	3.6634e-...	0.0350			
0.9600	4.1650e-...	4.6725e-...	4.8191e-...	4.8895e-...	4.9340e-...	4.9639e-...	4.9854e-...	5.0016e-...	5.0144e-...	95e-...	4.8191e-...	4.6725e-...	4.1650e-...	0.0400			
0.9550	4.6612e-...	5.2292e-...	5.3932e-...	5.4720e-...	5.5218e-...	5.5553e-...	5.5794e-...	5.5975e-...	5.6118e-...	20e-...	5.3932e-...	5.2292e-...	4.6612e-...	0.0450			
0.9500	5.1520e-...	5.7798e-...	5.9611e-...	6.0482e-...	6.1032e-...	6.1403e-...	6.1669e-...	6.1869e-...	6.2027e-...	82e-...	5.9611e-...	5.7798e-...	5.1520e-...	0.0500			
0.9450	5.6374e-...	6.3243e-...	6.5227e-...	6.6180e-...	6.6782e-...	6.7188e-...	6.7478e-...	6.7698e-...	6.7871e-...	80e-...	6.5227e-...	6.3243e-...	5.6374e-...	0.0550			
0.9400	6.1174e-...	6.8627e-...	7.0780e-...	7.1814e-...	7.2467e-...	7.2908e-...	7.3223e-...	7.3462e-...	7.3649e-...	14e-...	7.0780e-...	6.8627e-...	6.1174e-...	0.0600			
0.9350	6.5919e-...	7.3951e-...	7.6271e-...	7.7385e-...	7.8089e-...	7.8563e-...	7.8903e-...	7.9160e-...	7.9362e-...	85e-...	7.6271e-...	7.3951e-...	6.5919e-...	0.0650			
0.9300	7.0610e-...	7.9214e-...	8.1699e-...	8.2892e-...	8.3646e-...	8.4154e-...	8.4518e-...	8.4794e-...	8.5010e-...	92e-...	8.1699e-...	7.9214e-...	7.0610e-...	0.0700			
0.9250	7.5247e-...	8.4415e-...	8.7064e-...	8.8335e-...	8.9139e-...	8.9680e-...	9.0068e-...	9.0362e-...	9.0592e-...	35e-...	8.7064e-...	8.4415e-...	7.5247e-...	0.0750			
0.9200	7.9829e-...	8.9556e-...	9.2366e-...	9.3715e-...	9.4567e-...	9.5142e-...	9.5554e-...	9.5865e-...	9.6109e-...	15e-...	9.2366e-...	8.9556e-...	7.9829e-...	0.0800			
0.9150	8.4358e-...	9.4637e-...	9.7605e-...	9.9031e-...	9.9932e-...	1.0054e-...	1.0097e-...	1.0130e-...	1.0156e-...	31e-...	9.7605e-...	9.4637e-...	8.4358e-...	0.0850			
0.9100	8.8832e-...	9.9656e-...	1.0278e-...	1.0428e-...	1.0523e-...	1.0587e-...	1.0633e-...	1.0668e-...	1.0695e-...	28e-...	1.0278e-...	9.9656e-...	8.8832e-...	0.0900			
0.9050	9.3252e-...	1.0461e-...	1.0790e-...	1.0947e-...	1.1047e-...	1.1114e-...	1.1162e-...	1.1198e-...	1.1227e-...	47e-...	1.0790e-...	1.0461e-...	9.3252e-...	0.0950			
0.9000	9.7617e-...	1.0951e-...	1.1295e-...	1.1460e-...	1.1564e-...	1.1634e-...	1.1685e-...	1.1723e-...	1.1752e-...	60e-...	1.1295e-...	1.0951e-...	9.7617e-...	0.1000			
0.8950	1.0193e-...	1.1435e-...	1.1794e-...	1.1966e-...	1.2075e-...	1.2148e-...	1.2201e-...	1.2240e-...	1.2272e-...	66e-...	1.1794e-...	1.1435e-...	1.0193e-...	0.1050			
0.8900	1.0619e-...	1.1912e-...	1.2286e-...	1.2466e-...	1.2579e-...	1.2655e-...	1.2710e-...	1.2752e-...	1.2784e-...	56e-...	1.2286e-...	1.1912e-...	1.0619e-...	0.1100			
0.8850	1.1039e-...	1.2384e-...	1.2772e-...	1.2959e-...	1.3077e-...	1.3156e-...	1.3213e-...	1.3256e-...	1.3290e-...	59e-...	1.2772e-...	1.2384e-...	1.1039e-...	0.1150			
0.8800	1.1454e-...	1.2849e-...	1.3252e-...	1.3446e-...	1.3568e-...	1.3651e-...	1.3710e-...	1.3755e-...	1.3790e-...	46e-...	1.3252e-...	1.2849e-...	1.1454e-...	0.1200			

……..

And we can find all of the middle values are zero. So we only consider the sides value that is the sate 1=[200A1, 0A2] and state 201 = [0A1, 200A2]. And a figure of the first coloumn and the last coloumn of steady

result is obtained as the following shows:



And then find the steady distribution. We can find that there are only two kinds of distribution [200A1, 0A2] or [0 A1 or 200A2] when it is reach to steady distribution. And the probability of [200A1, 0A2] and [0A1, 200A2] when it is steady are as the above figure shows.

We can find the probability of steady distribution [200A1, 0 A2] and [0A1, 200A2] changes linearly with different initial distribution.

Perron-Frobenius theorem: Let P be a regular statistic matrix, suppose P is irreducible and aperiodic . Then P has a unique positive eigenoector π with $\pi_i > 0$ for $1 \leq i \leq N$

$$\pi P = \pi$$

With the MATLAB command eig(P'), we can obtain a series of eignoetors(as the following figure shows). And each eignoetor has a unique convergent result. So it DOESN'T defy the Perro-Frobenius theory.

201x1 complex dou

	1	2
1	1	
2	1	
3	0.9950	
4	0.9851	
5	0.9703	
6	0.9509	
7	0.9271	
8	0.8993	
9	0.8678	
10	0.8331	
11	0.7956	
12	0.7558	
13	0.7143	
14	0.6714	
15	0.6278	
16	0.5838	
17	0.5400	
18	0.4968	
19	0.4546	
20	0.4137	
21	0.3744	
22	0.3369	

Markov chain ergodic theorem: for every non-negative vector x $\in \mathbf{R^N}$, then if P satisfies the requirement for Peron-Fronisu

$$\lim_{n\to\infty} XP^n = \pi$$

Actually, this experiment does also doesn't defy the Markov chain ergodic theorem. Because a discrete-time Markov chain is a sequence of random variables X1, X2, X3, ... with the Markov property, namely that the probability of moving to the next state depends only on the present state and not on the previous states

P(Xn = xn|Xn-1 = xn-1, Xn-2 = xn-2,..., X0=x0) = P(Xn = xn|Xn-1 = xn-1).