EE511 – Project #7 Muthulakshmi Chandrasekaran

Simulation of CTMC 4486180242

1.1 Objective of the Project

The objective of this project is to simulate a Continuous Time Markov Chain. This is simulated by bank operations. The tellers in a bank serving the customers is simulated. Three scenarios are simulated. The performance measures of the system are found for each case.

1.2 Problem Statement

The teller operations in a bank are simulated. The customer arrival Rates are Poisson with parameter λ. The inter – arrival times are exponentially distributed. The customers can have both simple and complex transactions to do – the service times of each customer is different. 75% of the customers have a simple transaction to do. The service time of such customers is 2 minutes, that follows Erlang – 2 distribution. The rest 25% customers have complex transactions to do. The service time in that case is 6 minutes, that is an Erlang – 5 distribution.

The Erlang random variable is chosen according to the formula $S_n = \frac{-\ln(\prod_{i=1}^{n} U_i)}{\lambda}$ where $U_i \sim Uniform(0,1)$. There are three servers – or three tellers, and the length of the queue is infinite i.e. it can accommodate any number of customers.

Scenario 1 employs only a single queue. Whenever a teller becomes free, the next waiting customer at the queue is served.

Scenario 2 employs three separate queues for three tellers. When a customer arrives, he joins the shortest queue. A customer cannot switch between queues.

Scenario 3 has two separate queues – one for simple transactions and one for complex transactions. The customer with the corresponding service time enters the corresponding queue. One teller can handle only simple transactions, while the other two tellers can handle both simple and complex transactions. When an experienced teller becomes free, he chooses a customer from any of the non-empty queues. When the queue is empty, the teller becomes idle. When a customer arrives, and finds one or more tellers idle, the customer chooses the teller, based on the type of transaction he has to do.

These scenarios are simulated. Parameters of queue length distribution, waiting time of the customers, the distribution of queue length, mean and variance of the distribution etc. are computed.

1.3 Mathematical Basis

1.3.1 Continuous Time Markov Chains [Reference 1]

In a continuous time, Markov chain, the state transitions may occur at any time, and the time between transitions is exponentially distributed. Since the exponential distribution is memoryless, the future outcome of the process depends only on the present state and does not depend on when the last transition occurred or what any of the previous states were.

We denote the state of the system at time t as $X(t)$. The state probability at time t is the probability that the system is in state j at time t, and is denoted as:

$$p_j(t) = Prob\{X(t) = j\}$$

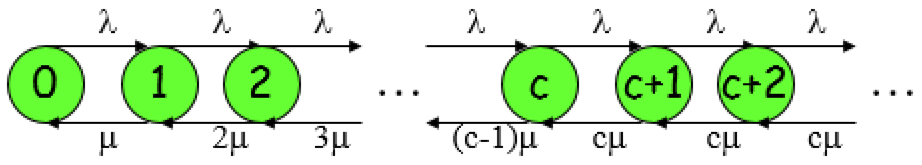
The steady-state or limiting probability of being in state j is given by

$$p_j = \lim_{n\to\infty} p_j(t)$$

For a continuous time, Markov chain, the *intensity matrix* or *rate matrix* Q can be defined. The elements q_{ij} of Q indicate the rate of transitions from state i to state j for $i \neq j$. In other words, the time to make a transition to state j given that the process is in state i is exponentially distributed with rate parameter q_{ij} .

1.3.2 Observations from the CTMC

A M/M/c queue can be modeled as [Reference2]



Some parameters that can be computed using the CTMC are

i. $Average\ Waiting\ Time\ = \frac{\sum Waiting\ time\ in\ the\ queue}{Total\ Number\ of\ Customers}$

ii. $Probability\ that\ a\ customer\ has\ to\ wait\ = \frac{Number\ of\ customers\ who\ wait}{Total\ Number\ of\ Customers}$

iii. $Proportion\ of\ Server\ Idle\ Times\ = \frac{\sum Idle\ time\ of\ Server}{Simulation\ Run\ Time}$

iv. $Average\ Service\ Time\ = \frac{\sum Service\ Time}{Number\ of\ Customers}$

1.4 Matlab Simulation

1.4.1 Description of the Codes

Part A – Single Queue Scenario

The total number of customers in the system are fixed. The arrival rates are Poisson, the interarrival rates are exponential. The customer transactions are assigned as 75% simple and 25% complex. The service times are chosen based on the random decision of customer transactions. The arrival rates are stored in the array arr_time, and the service times are stored in $serv\_time$. Whenever a teller becomes free, the next customer on the queue is served.

Part B – Three queues Scenario

This is like the Scenario 1, where the arrival processes are Poisson. Hence the interarrivals would be Exponential. The tellers can handle any type of transactions. There are three queues as well. The arriving customer chooses the shortest queue to enter in. Hence there are three separate queues to enter in, and the teller services each customer for a service time mentioned in $serv\_time$.

Part C – Different Tellers handling different transactions

In the third scenario, there are two queues – one for simple transactions and one for complex transactions. One teller can handle simple transactions only, the remaining tellers can handle any type of transactions. Here the teller is idle only if there are no customers in the queue. The arriving customers choose the teller if the tellers are idle. The tellers can handle their respective transactions only.

1.4.2 Observations

In all the scenarios, the input arrival rates are Poisson. Hence the input parameter λ is given to the function and the results are simulated.

Figure 1 shows the arrival times, which are Poisson distributed.

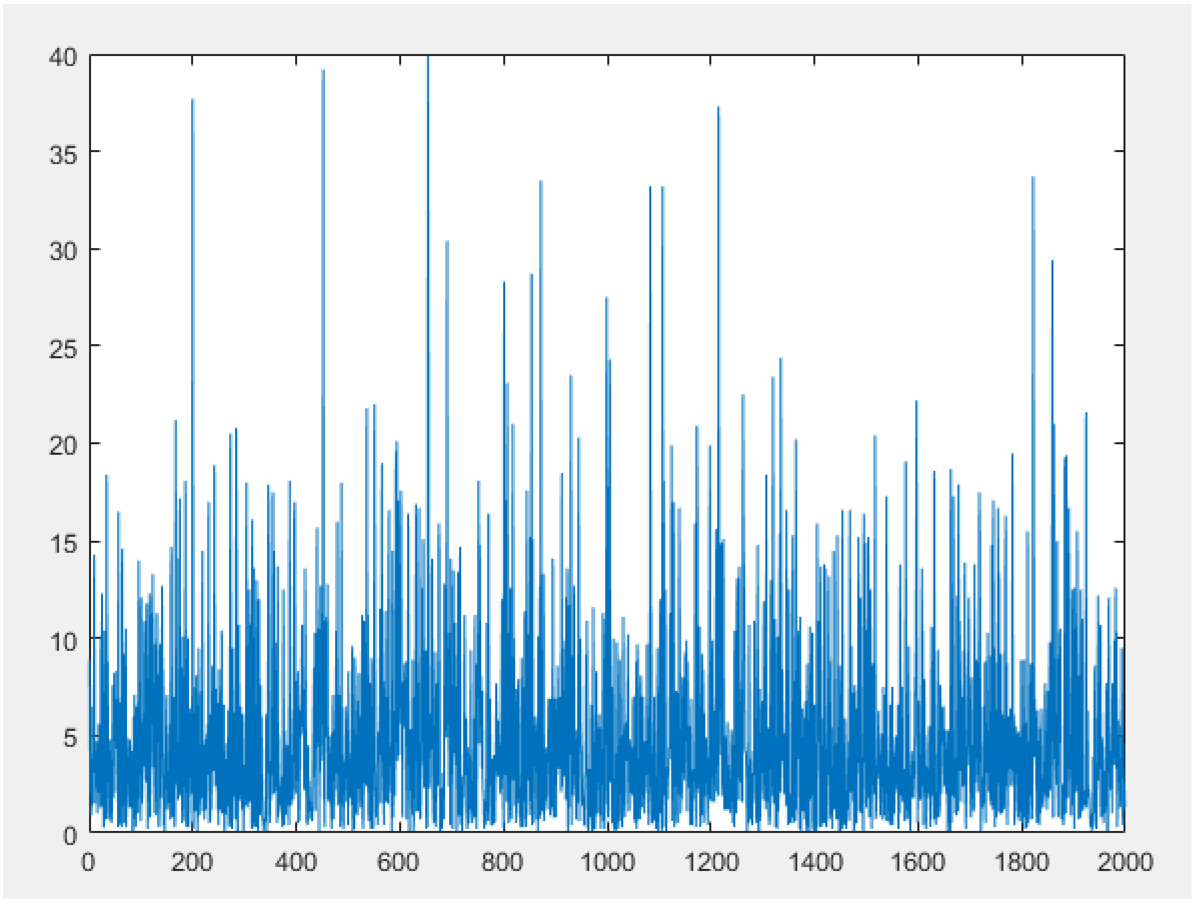


Figure 1 Arrival Rates are Poisson Distributed

Figure 2 shows the graph of the interarrival times for the arrival rate =0.2. Hence the interarrival times are exponentially distributed, as observed in Figure 2.

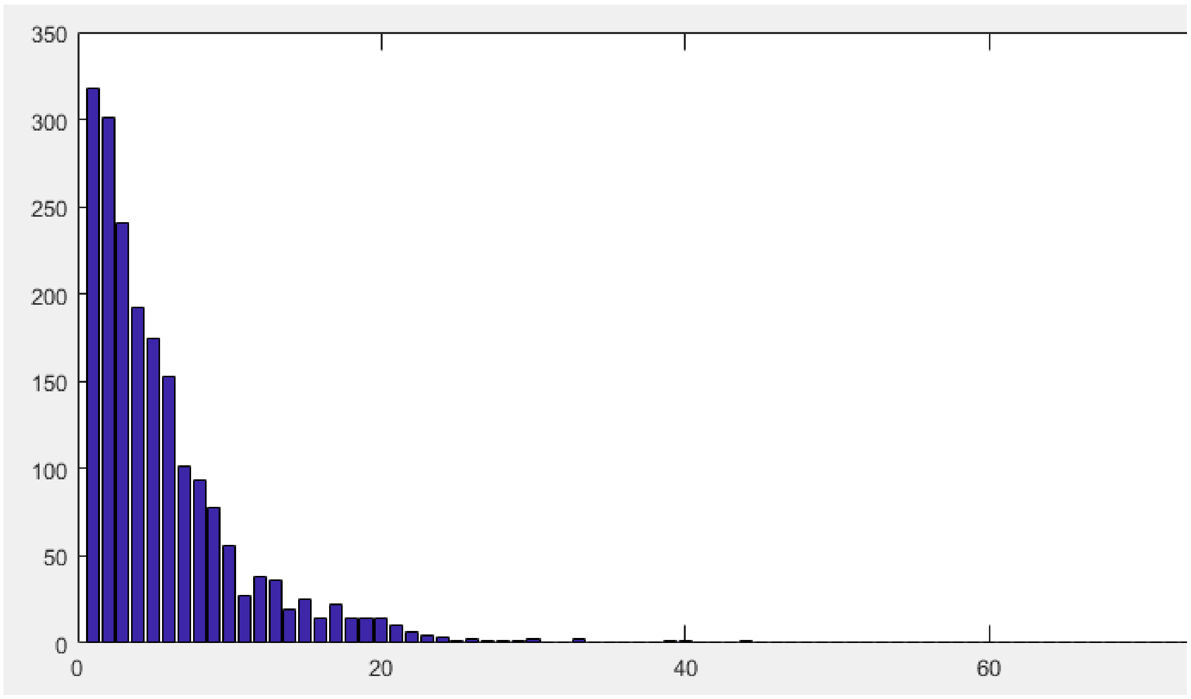
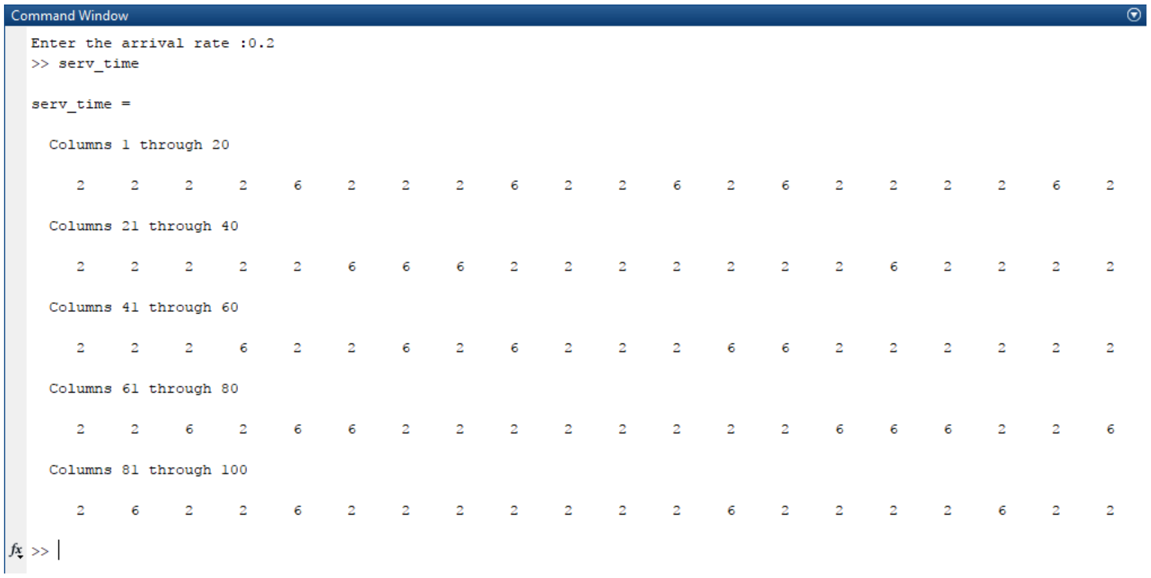


Figure 2 Interarrival Times – Exponentially distributed

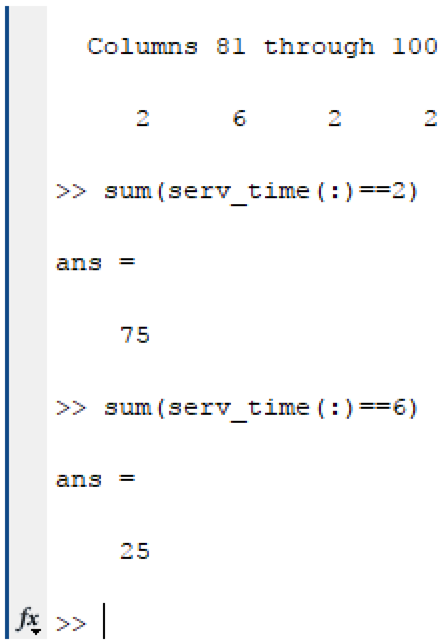
The transactions are either simple or complex. 75% of the customers have simple transactions to do and 25% of the customers have complex transactions to do. This is modeled using the concept of random numbers. An array of the total number of customers is generated. Random numbers are assigned to each customer. The first 75% values of the total number of customers is assigned a simple transaction, and the rest are assigned a complex transaction. Figure 3 shows a screenshot of the type of transaction assigned to each customer.



```
Enter the arrival rate :0.2
>> serv_time

serv_time =

  Columns 1 through 20

     2     2     2     2     6     2     2     2     6     2     2     6     2     6     2     2     2     2     6     2

  Columns 21 through 40

     2     2     2     2     2     6     6     6     2     2     2     2     2     2     2     6     2     2     2     2

  Columns 41 through 60

     2     2     2     6     2     2     6     2     6     2     2     2     6     6     2     2     2     2     2     2

  Columns 61 through 80

     2     2     6     2     6     6     2     2     2     2     2     2     2     2     6     6     6     2     2     6

  Columns 81 through 100

     2     6     2     2     6     2     2     2     2     2     2     2     6     2     2     2     2     6     2     2

fx >> |
```

Figure 3 Type of transactions

Figure 4 shows the percentage of simple and complex transactions for 100 customers.



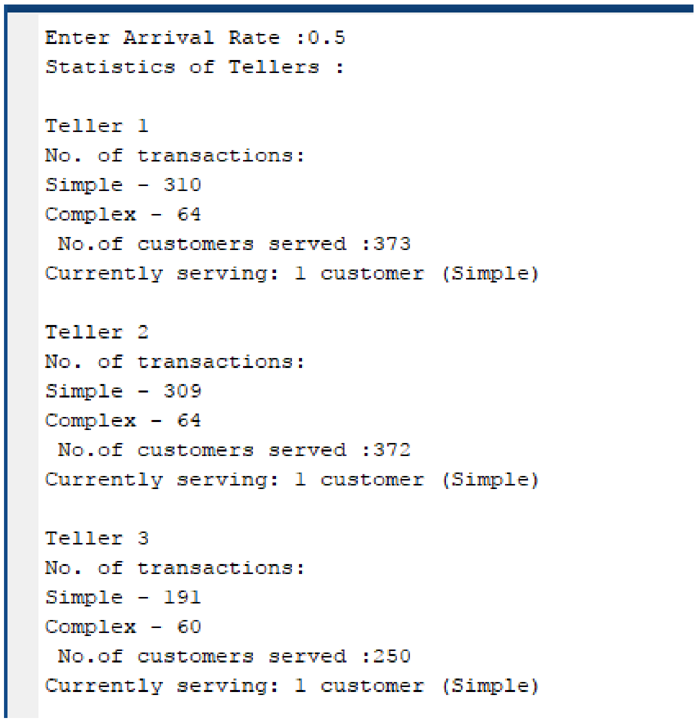
```
  Columns 81 through 100

     2     6     2     2

>> sum(serv_time(:)==2)

ans =

    75

>> sum(serv_time(:)==6)

ans =

    25

>> 
```

Figure 4 Percentage of simple and complex transactions for 100 customers

The number of servers is three, and the storage space is infinite. For stability, the arrival rate is determined to be lesser than the service rate.

Based on these the three scenarios are simulated.

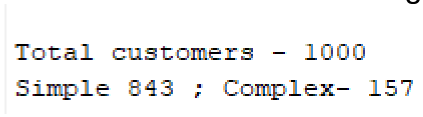
In the first scenario, there is only one queue. Customers join the queue according to the corresponding arrival times. Whenever a teller becomes free, the customer in the queue is serviced next. This is a M/M/3 queue, where there is a single queue serviced by three servers. The teller statistics are observed as in Figure 5.



```
Enter Arrival Rate :0.5
Statistics of Tellers :

Teller 1
No. of transactions:
Simple - 310
Complex - 64
 No.of customers served :373
Currently serving: 1 customer (Simple)

Teller 2
No. of transactions:
Simple - 309
Complex - 64
 No.of customers served :372
Currently serving: 1 customer (Simple)

Teller 3
No. of transactions:
Simple - 191
Complex - 60
 No.of customers served :250
Currently serving: 1 customer (Simple)
```

Figure 5 Servers Statistics for Scenario 1

The total number of customers serviced is observed as in Figure 6.



```
Total customers - 1000
Simple 843 ; Complex- 157
```

Figure 6 Serviced customer statistics for Scenario 1

The average waiting time of the customers is observed as a graph as in Figure 7.

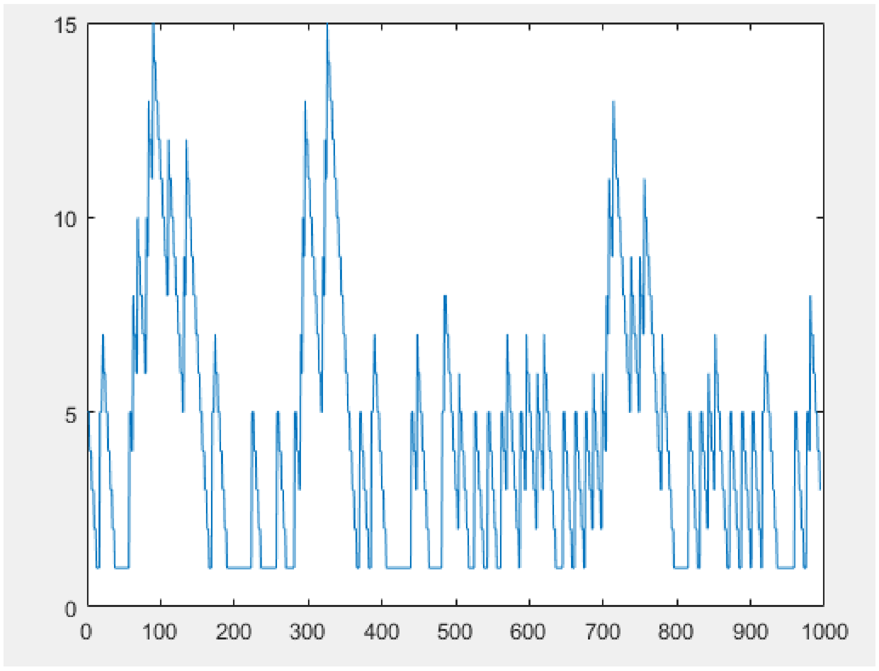
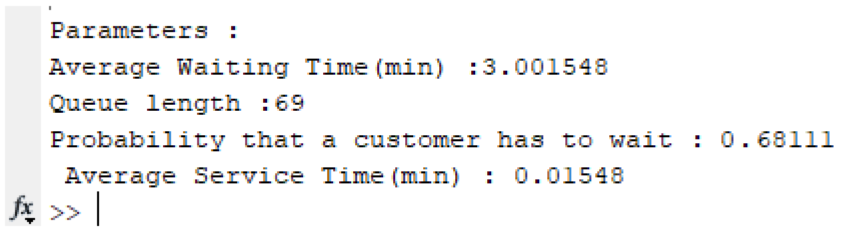


Figure 7 Average waiting time of the customers

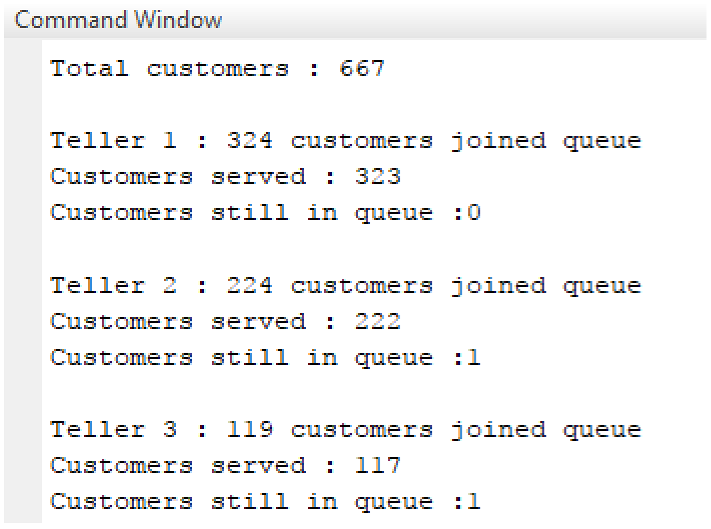
The other parameters that were observed is shown in Figure 8.



```
Parameters :
Average Waiting Time(min) :3.001548
Queue length :69
Probability that a customer has to wait : 0.68111
 Average Service Time(min) : 0.01548
>>
```

Figure 8 Parameters observed

In the second scenario, there is one separate queue for each teller. Customers arrive according to the respective arrival times, which are Poisson distributed. Whenever the teller becomes free, the first customer in the corresponding queue is serviced next. This can be considered as three separate M/M/1 queues. The only constraint is on the inclusion of the arriving customer on the respective queue. The teller and customer statistics are observed as in Figure 9.



```
Command Window
Total customers : 667

Teller 1 : 324 customers joined queue
Customers served : 323
Customers still in queue :0

Teller 2 : 224 customers joined queue
Customers served : 222
Customers still in queue :1

Teller 3 : 119 customers joined queue
Customers served : 117
Customers still in queue :1
```

Figure 9 Server and Customer Statistics for Scenario 2

The average waiting time of the customers is observed as a graph as in Figure 10.

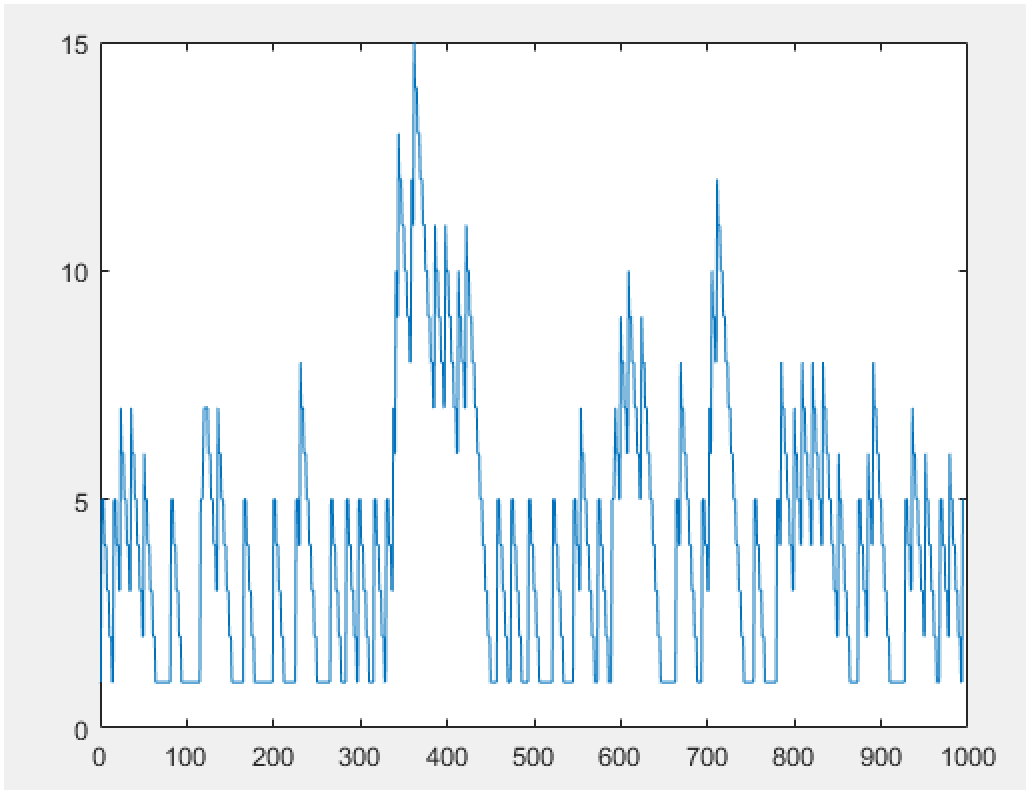
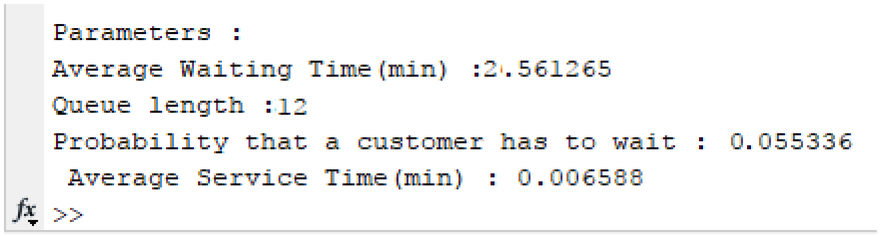


Figure 10 Average waiting time of the customers

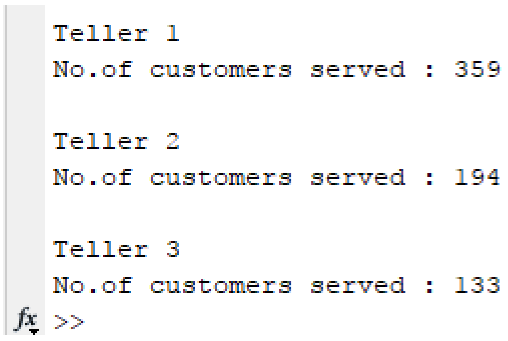
The other parameters that were observed is shown in Figure 11.



```
Parameters :
Average Waiting Time(min) :2.561265
Queue length :12
Probability that a customer has to wait :  0.055336
 Average Service Time(min) : 0.006588
>>
```

Figure 11 Parameters observed

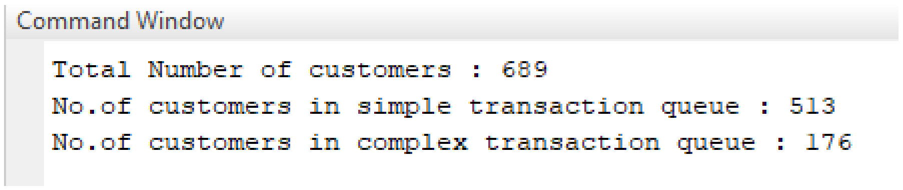
In the third scenario, there are two queues – one for simple transactions and one for complex transactions. One teller can handle simple transactions only, the remaining tellers can handle any type of transactions. Here the teller is idle only if there are no customers in the queue. The arriving customers choose the teller if the tellers are idle. The tellers can handle their respective transactions only. The teller statistics for this scenario is shown in Figure 12.



```
Teller 1
No.of customers served : 359

Teller 2
No.of customers served : 194

Teller 3
No.of customers served : 133
>>
```

Figure 12 Servers Statistics for Scenario 3

The total number of customers serviced is observed as in Figure 13.



```
Total Number of customers : 689
No.of customers in simple transaction queue : 513
No.of customers in complex transaction queue : 176
```

Figure 13 Serviced customer statistics for Scenario 3

The average waiting time of the customers is observed as a graph as in Figure 14.

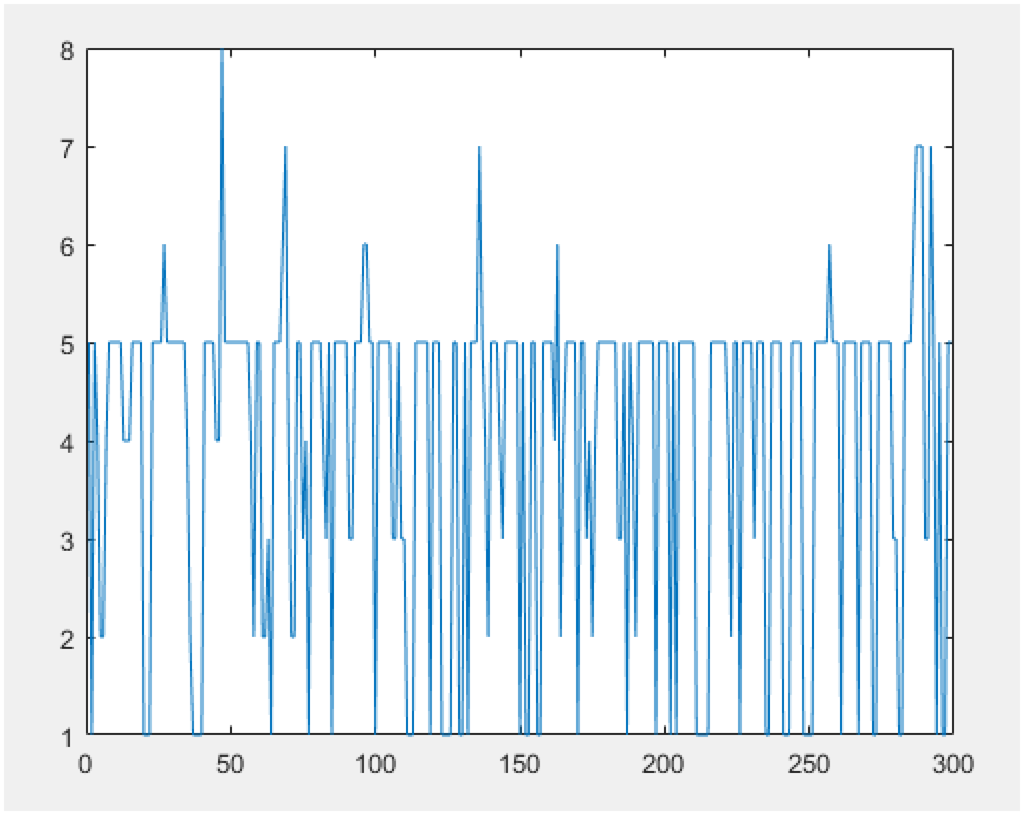
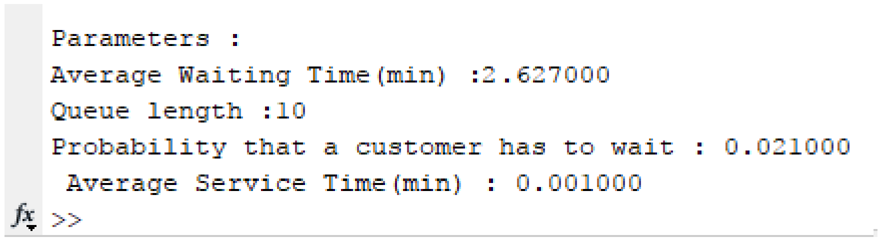


Figure 14 Average waiting time of the customers

The other parameters that were observed is shown in Figure 15.



```
Parameters :
Average Waiting Time(min) :2.627000
Queue length :10
Probability that a customer has to wait : 0.021000
 Average Service Time(min) : 0.001000
>>
```

Figure 15 Parameters observed

1.4.3 Results

The three scenarios are simulated and the corresponding graphs are obtained. The following observations are made.

i. The arrival rates are Poisson in all the scenarios. This means the interarrival rates are exponentially distributed, making it a M/M/c queueing system.
ii. The Poisson arrival rates are chosen to be lesser than the Erlang service rates. This is to make the system stable. If the arrival rate was greater than the service rate, then the system would go into instability.
iii. The first scenario is an example of M/M/3 queueing system. The second could be considered as three separate M/M/1 queues with a single arrival rate. The third could be a combination of both, with server specific queues. These scenarios are validated with a few performance measures.
iv. Comparing the graph of waiting time for the three scenarios, the average waiting time of the first scenario is more fluctuating, and the average is also high. In the second scenario, the average is still higher, but with lesser fluctuations. In the third scenario, the average waiting time is reduced.
v. The customer statistics as well as the teller statistics are observed for each scenario. It is found that the servers are most efficient in the third scenario.
vi. The other parameters are computed. The average waiting time is the least for the third scenario. This means the services are efficiently done.
vii. Another crucial factor that determines the efficiency of the system is the Queue Length. The queue length for the first case is maximum, since there is only one queue to accommodate the incoming customers. The presence of three queues in the second scenario is also

efficient. In the third scenario, there are two separate queues for simple and complex transactions. Here the service time of the complex transaction queue will be more. There was no significant difference observed for queue lengths in second and third scenarios.

viii. Probability that a customer must wait before being serviced is also found. As expected, the waiting time is more in the first scenario, and hence the probability of a customer waiting in the queue is also higher.

ix. The average service time is also computed. It is found to be the least for the scenario with relatively fast-moving queues i.e. Scenario 3.

Thus, out of all the three simulated scenarios, the scenario 3 is found to give better results compared to the scenario 1 and scenario 2 in terms of performance metrics.

1.5 Reference

1. www.columbia.edu/~ks20/stochastic-I/stochastic-I-CTMC.pdf

2. https://www.utdallas.edu/~jjue/cs6352/markov/node5.html

3. Ross – Simulation – Edition 5

4. Matlab Tutorials

1.6 Matlab Code

Poisson Arrivals – Poisson Process

```
function [arr,count1]=poisson_arrv(lambda)
la=lambda; % No.of arrivals per second
T=1*10000; % simulation time tot_time
delta=0.1; % simulation step size in second
N=T/delta; % number of simulation steps

event=zeros(N,1); % array recording at each step if a "packet" arrived.
 % initialize it to zeros
R=rand(size(event)); % generate a random array (with elements in [0,1]) of the same
size as "event"
event(R<la*delta)=1; % set each element of event to 1 with probability lambda*delta
inds=find(event==1); % getting indices of arrivial
int_times=diff(inds)*delta; % interarrival times in seconds
arr=int_times;
edges=0:1:100; % define histogram bin
count1=histc(int_times,edges);
```

```
end
```

Scenario 1 – Single Queue

```
%Scenario 1 - Only one queue
clc;clear all;close all;
%q is the variable that defines the number of queues
q=1;
%N tells the number of servers or tellers
N=3;
%No. of customers is stored in cust_count
cust_count=0;
%Two types of transactions simple and complex
simple=0;
complex=0;
%Define the properties of the queue
queue=0;
sq=zeros(1,3);
cq=zeros(1,3);
%Service times is stored in the array serv_time
serv_time=zeros(1,3);
%The number of served_cust customers is in the array served_cust_cust
served_cust=zeros(1,3);
%The total simulation time(secs)
T=1000*60;

%Arrival Rates are Poisson
%lambda for poisson
%1person every 90 seconds
l=input('Enter Arrival Rate :');
%Service time for simple customers - Erlang2
lambda1=1;n1=2;

%Service time for complex transactions -Erlang5
lambda2=0.81;n2=5;

%simulation step size in seconds
delta=60;
%simulation time
S=T/delta;
Sn=zeros(1,1000);
%complex transaction
c=zeros(1,2000);
nextcust=1;
%3 tellers
teller=zeros(1,3);
%simple or complex process (s_or_c=0 for simple and s_or_c=1 for complex)
s_or_c=zeros(1,3);
%waiting time
wait=[];
start=[];
e=[];

for sim=1:S
    %random value between 0 and 1
    R=rand(1);
    if R<=(l*delta)
        %total customers
        cust_count=cust_count+1;
        %single queue
        queue=queue+1;
        start(cust_count)=sim;
        %Check if the transaction is complex or simple
        if R<=0.167
            complex=complex+1;
            c(cust_count)=1;
        else
            simple=simple+1;
        end
    end
    update=zeros(1,3);
    %checking if any teller is free
    for t=1:3
        if teller(t)==0 & queue>=1 & serv_time(t)==0
            teller(t)=1;
            queue=queue-1;
            %if complex transaction(6 minutes)
            if c(nextcust)==1
                serv_time_sc(t)=1;
                cq(t)=cq(t)+1;
                for N=1:1000
                    %Generate Sn by inversion method from Uniform RV
                    x=rand(1,n2);
                    Sn(N)= -log(prod(x))./lambda2;
                end
            end
            %if simple transaction(2 minutes)
            if c(nextcust)==0
                serv_time_sc(t)=0;
                sq(t)=sq(t)+1;
                for N=1:1000
                    %Generate Sn by inversion method from Uniform RV
                    x=rand(1,n1);
                    Sn(N)= -log(prod(x))./lambda1;
                end
            end
        serv_time(t)=round(sum(Sn)/1000);
    end
    %Decreement the service times
    if serv_time(t)>0
    serv_time(t)=serv_time(t)-1;
    %served_cust customers
    if serv_time(t)==0
        e(nextcust)=sim;
        served_cust(t)=served_cust(t)+1;
        nextcust=nextcust+1;
        teller(t)=0;
    end
    end
    end

end

last=length(e);
for i=1:last
```

```
    wait(i)=e(i)-start(i);
end
queue_length=cust_count+randi([1 50]);

%Computing average service time
avg_serv_time = sum(serv_time)/cust_count;
%results
plot(wait);
disp('Statistics of Tellers :');
for r=1:3
fprintf("\nTeller %d\n",r);
fprintf("No. of transactions:\nSimple - %d\nComplex - %d\n No.of customers served
:%d\n",sq(r),cq(r),served_cust(r));
if teller(r)==1
    if s_or_c(r)==1
        fprintf("Currently serving: %d customer (Complex)\n",teller(r));
    else
        fprintf("Currently serving: %d customer (Simple)\n",teller(r));
    end
end
end
total_serv=served_cust(1)+served_cust(2)+served_cust(2);
fprintf(" \nTotal customers - %d \nSimple %d ; Complex-
%d\n",cust_count,simple,complex);

fprintf("\nParameters :");
fprintf("\nAverage Waiting Time(min) :%f",sum(wait)/cust_count);
fprintf("\nQueue length :%d",queue_length);
fprintf("\nProbability that a customer has to wait : %f",queue_length/cust_count);
fprintf("\n Average Service Time(min) : %f\n",avg_serv_time);
```

Scenario 2 – Three queues

```
%Scenario 2 - Three queues
clc;clear all;close all;
%q is the variable that defines the number of queues
q=3;
%N tells the number of servers or tellers
N=3;
%No. of customers is stored in cust_count
cust_count=0;
%Two types of transactions simple and complex
simple=0;
complex=0;
%Define the properties of the queue
queue=0;
sq=zeros(1,3);
cq=zeros(1,3);
%Service times is stored in the array serv_time
serv_time=zeros(1,3);
%The number of served_cust customers is in the array served_cust_cust
served_cust=zeros(1,3);
%The total simulation time(secs)
T=1000*60;

%Arrival Rates are Poisson
%lambda for poisson
%1person every 90 seconds
l=input('Enter Arrival Rate :');
```

```
%Service time for simple customers - Erlang2
lambda1=1;n1=2;

%Service time for complex transactions -Erlang5
lambda2=0.81;n2=5;
%simulation step size in seconds
delta=60;
S=T/delta;
Sn=zeros(1,1000);
%cpx transaction
c=zeros(1,1000);
next_customer=1;
%3 servers
server=zeros(1,3);
%waiting time
start=[];
e=[];
wait=[];
people=1;
simple=0;cpx=0;
for sim=1:S
    %random value between 0 and 1
    R=rand(1);
    if R<=(l*delta)
        %total customers
        cust_count=cust_count+1;
        %adding newly arrived customers to shortest queue
        [cust_numb,position]=min(queue);
        queue(position)=queue(position)+people;
        actual_q(position)=actual_q(position)+people;
        start(cust_count)=sim;
        %cpx transaction
        if R<=0.167
            c(cust_count)=1;
            cpx=cpx+people;
        else
            simple=simple+people;
        end
    end
    %checking if any server is free
    update=zeros(1,3);
    for t=1:3
    if server(t)==0 & queue(t)>=1 & serv_time(t)==0
    server(t)=1;
    queue(t)=queue(t)-1;
    %if cpx transaction(6 minutes)
    if c(next_customer)==1
    for N=1:1000
    x=rand(1,n2);
    Sn(N)= -log(prod(x))./lambda2;
    end
    end
    %if simple transaction(2 minutes)
    if c(next_customer)==0
    for N=1:1000
    x=rand(1,n1);
    Sn(N)= -log(prod(x))./lambda1;
    end
    end
    serv_time =round(sum(Sn)/1000);
    end
    %serving time
    if serv_time(t)>0
    serv_time(t)=serv_time(t)-1;
    %served customers
    if serv_time(t)==0
        e(next_customer)=sim;
        cust_serv(t)=cust_serv(t)+1;
   %     next_customer=next_customer+1;
    %    server(t)=0;
    end
    end
    end
end

plot(wait)
fprintf("Total customers : %d\n",cust_count);
for r=1:3
fprintf("\nserver %d : ",r);
fprintf("%d customers joined queue\n",actual_q(r));
fprintf("Customers served : %d \n",cust_serv(r));
fprintf("Customers still in queue :%d\n",queue(r));
end

fprintf("\nParameters :");
fprintf("\nAverage Waiting Time(min) :%f",sum(wait)/cust_count);
fprintf("\nQueue length :%d",queue_length);
fprintf("\nProbability that a customer has to wait : %f",queue_length/cust_count);
fprintf("\n Average Service Time(min) : %f\n",avg_serv_time);
```

<u>Scenario 3</u>

```
%Scenario 3 - Two queues
clc;clear all;close all;
%q is the variable that defines the number of queues
q=2;
%N tells the number of servers or tellers
N=3;
%No. of customers is stored in cust_count
cust_count=0;
%Two types of transactions simple and complex
simple=0;
complex=0;
%Define the properties of the queue
queue=0;
sq=zeros(1,3);
cq=zeros(1,3);
%Service times is stored in the array serv_time
serv_time=zeros(1,3);
%The number of served_cust customers is in the array served_cust_cust
served_cust=zeros(1,3);
%The total simulation time(secs)
T=1000*60;

%Arrival Rates are Poisson
```

```
%lambda for poisson
l=input('Enter Arrival Rate :');
%Service time for simple customers - Erlang2
lambda1=1;n1=2;

%Service time for complex transactions -Erlang5
lambda2=0.81;n2=5;
%simulation time in seconds
T=1000*60;
%simulation step size in seconds
delta=60;
%simulation time
S=T/delta;
Sn=zeros(1,1000);
%complex transaction
c=zeros(1,1000);
nextcust=1;
%3 servers
server=zeros(1,3);
%waiting time
start=[];
e=[];
wait=[];
for sim=1:S
    %random value between 0 and 1
    R=rand(1);
    if R<=(l*delta)
           cust_count=cust_count+1;
            start(cust_count)=sim;
        %complex transaction queue
        if R<=0.167
            queue(2)=queue(2)+people;
            act_queue(2)=act_queue(2)+people;
            c(cust_count)=1;
        else
            %simple transaction queue
            queue(1)=queue(1)+people;
            act_queue(1)=act_queue(1)+people;
        end
    end
    %check if any server is free
    update=zeros(1,3);
    if queue(1)>0 & server(1)==0 & m(1)==0
        server(1)=1;
        queue(1)=queue(1)-1;
        for N=1:1000
            %generate Erlang RV
            x=rand(1,n1);
            Sn(N)= -log(prod(x))./lambda1;
        end
        m(1)=round(sum(Sn)/1000);
    end
    %serving time for simple transactions
    if m(1)>0
        m(1)=m(1)-1;
        if m(1)==0
            served_cust(1)=served_cust(1)+1;
            server(1)=0;
            e(nextcust)=sim;
            nextcust=nextcust+1;
        end
    end
    %checking if other servers are free
    for t=2:3
            %both queues non empty
            if queue(1)>0 & queue(2)>0 & server(t)==0 & m(t)==0
                server(t)=1;
                %choose to serve customer from either queues
                choose=randi(2);
                %simple transaction(2 minutes)
                if choose==1
                    queue(1)=queue(1)-1;
                    for N=1:1000
                        x=rand(1,n1);
                        Sn(N)= -log(prod(x))./lambda1;
                    end
                end
                %complex transaction(6 minutes)
                if choose==2
                    queue(2)=queue(2)-1;
                    for N=1:1000
                        x=rand(1,n2);
                        Sn(N)= -log(prod(x))./lambda2;
                    end
                end
                m(t)=round(sum(Sn)/1000);
            end
            %if simple transaction queue non empty
            if queue(1)>0 & queue(2)==0 & server(t)==0 & m(t)==0
                server(t)=1;
                queue(1)=queue(1)-1;
                for N=1:1000
                    x=rand(1,n1);
                    Sn(N)= -log(prod(x))./lambda1;
                end
                m(t)=round(sum(Sn)/1000);
            end
            %if complex transaction queue non empty
            if queue(2)>0 & queue(1)==0 & server(t)==0 & m(t)==0
                server(t)=1;
                queue(2)=queue(2)-1;
                for N=1:1000
                    x=rand(1,n2);
                    Sn(N)= -log(prod(x))./lambda2;
                end
                m(t)=round(sum(Sn)/1000);
            end
            %serving time
            if m(t)>0
                m(t)=m(t)-1;
                %number of served customers
                if m(t)==0
                    e(nextcust)=sim;
                    served_cust(t)=served_cust(t)+1;
                    server(t)=0;
                    nextcust=nextcust+1;
                end
            end
            end
        end

        fprintf("Total Number of customers : %d\n",cust_count);
        fprintf("No.of customers in simple transaction queue : %d\n",act_queue(1));
        fprintf("No.of customers in complex transaction queue : %d\n",act_queue(2));
        for r=1:3
        fprintf("\nserver %d\n",r);
        fprintf("No.of customers served_cust : %d \n",served_cust(r));
        end

        plot(wait)
        fprintf("\nParameters :");
        fprintf("\nAverage Waiting Time(min) :%f",sum(wait)/cust_count);
        fprintf("\nQueue length :%d",queue_length);
        fprintf("\nProbability that a customer has to wait : %f",queue_length/cust_count);
        fprintf("\n Average Service Time(min) : %f\n",avg_serv_time);
```