Application of Continuous and Discrete Markov Chains in M(t)/M/S queueing

Harry Spearing

STORi MRes

Lancaster University Management School, Lancaster University
Department of Mathematics and Statistics.

March 23, 2018

Abstract

Continuous Time Markov Chains are applied to an M(t)/M/S queue using a transition rate matrix and the expected number in the queue $E(n(t))$ is tracked over time. This is tested on two different arrival rate functions and the experimental findings agree with the theoretical values. The upper 10 % tail of the distribution of the number in the queue varies in a similar way to the expected number $E(n(t))$ in the queue. For 3 servers, this peaks at 97 in our example. We conclude that the number of servers S, has more effect on $E(n(t))$ than on the upper 10 % tail of the distribution.
Discrete Time Markov Decision Chains are applied to and M(t)/M/1 queue and backwards recursion is used to find a closed form for the optimal policy.

1 Time-dependent behaviour of homogeneous Continuous Time Markov Chains.

The first section of this report will be based on Continuous Time Markov Chains (CTMC). Each Markov Chain has states $j = 0, 1..\infty$ and the rate at which it transitions from state i to j is given by q_{ij}. $\pi(t)_j$ is defined to be the probability of the system being in state j at a time t. Now consider a small interval of time h, over which $\pi(t)_j$ can be assumed to be constant. By considering all possible routes to state j at time $t+h$ conditional on the states at time t, $\pi(t+h)_j$ can be written as:

$$\pi(t+h)_j = \sum_{i \neq j} \pi(t)_i q_{ij} h + \pi(t)_j \left[1 - \sum_{k \neq j} q_{jk} h \right] \tag{1}$$

As $\lim_{h \to 0}$, equation (1) becomes a differential. By defining $q_{jj} = -\sum_{k \neq j} q_{jk}$ we can therefore write:

$$\frac{\mathrm{d}\pi(t)_j}{\mathrm{d}t} = \sum_i \pi(t)_i q_{ij} \tag{2}$$

To design the transition matrix for an M/M/1 queue, define the arrival rate λ, and the service rate μ. First, lets consider a transition probability matrix. An important assumption will be made, and that is that we choose our time intervals to be small enough such that only one person can join or leave the queue in one interval. This is a valid assumption as it will be ensured that the time intervals between calculations is small.
If we have 0 people in the queue, the probability of someone arriving in time interval t is simply λt. Since the only other option is the queue to remain as it is, the probability of no one arriving must be $1 - \lambda t$. Hence the first row of the transition probability matrix looks as follows:

$$\begin{pmatrix} 1-\lambda t & \lambda t & 0 & 0 & ... \\ . & & & & \\ . & & & & \end{pmatrix} \tag{3}$$

Then consider a system with 2 people in the queue. Now, the probability of someone being served in time interval t is just μt, and the probability of someone arriving is λt. This leaves the probability of no one arriving or leaving to be $1 - \lambda t - \mu t$. This follows for all queue lengths, $n(t) > 0$, thus the transition probability matrix becomes (4).

$$\begin{pmatrix} 1-\lambda t & \lambda t & 0 & 0 & 0 & ... \\ \mu t & 1-\lambda t-\mu t & \lambda t & 0 & 0 & ... \\ 0 & \mu t & 1-\lambda t-\mu t & \lambda t & 0 & ... \\ 0 & 0 & \mu t & 1-\lambda t-\mu t & \lambda t & ... \\ . & . & . & . & . & ... \end{pmatrix} \tag{4}$$

For CTMC we need a transition rate matrix, not a transition probability matrix. This is easy to derive from (5). By taking the derivative with respect to t, we

then have the transition rate matrix Q:

$$\begin{pmatrix} -\lambda & \lambda & 0 & 0 & 0 & ... \\ \mu & -\lambda-\mu & \lambda & 0 & 0 & ... \\ 0 & \mu & -\lambda-\mu & \lambda & 0 & ... \\ 0 & 0 & \mu & -\lambda-\mu & \lambda & ... \\ . & . & . & . & . & ... \end{pmatrix} \tag{5}$$

This will work for an M/M/1 queue, but we would like to generalise this to an M(t)/M/S queue. The overall service rate will be proportional to the number of active servers, s. For $S < j$ all servers are active so μ becomes $S\mu$, otherwise μ becomes $j\mu$. The arrival rate is now a function of time, therefore the transition rate matrix for a M(t)/M/S queue is described by (6)

$$\begin{pmatrix} -\lambda(t) & \lambda(t) & 0 & 0 & 0 & 0 & 0 & ... \\ \mu & -\lambda(t)-\mu & \lambda(t) & 0 & 0 & 0 & 0 & ... \\ 0 & 2\mu & -\lambda(t)-2\mu & \lambda(t) & 0 & 0 & 0 & ... \\ 0 & 0 & 3\mu & -\lambda(t)-3\mu & \lambda(t) & 0 & 0 & ... \\ .. & .. & .. & .. & .. & .. & .. & ... \\ 0 & 0 & 0 & S\mu & -\lambda(t)-S\mu & \lambda(t) & 0 & ... \\ .. & .. & .. & .. & .. & .. & .. & ... \\ 0 & 0 & 0 & 0 & 0 & 0 & S\mu & -S\mu \end{pmatrix} \tag{6}$$

One extra complication arises due infinite nature of the queue since it is not possible for Matlab to store a matrix of infinite length. The number of states j has to be artificially cut short to a length, n, which changes the very last row of the the transition rate matrix as seen in (6). This is cut to a length such that $\pi(t)_n$ is negligible. The Matlab code for this can be found in appendix A

1.1 Testing the Code

To evaluate equation (1), the code in appendix B was written. To use this code, there are 6 variables that must be passed to this function. Firstly, n represents the maximum number in the queue. There are many choices available for this, but the key is to select a number such that the probability of the queue reaching this length is negligible. *mu* represents μ, the service rate, *lambda* the arrival rate, and S the number of servers. h is the time-step. This should be chosen such that $\pi_j(t)$ remains approximately constant over this period. Finally, T is the total length we wish to run the simulation.

Firstly, the case of a constant λ was analysed. To test the code in this case, the limiting behaviour of $\pi(t)_j$ as $\lim_{t\to\infty}$ was calculated.

$$\lim_{t\to\infty} \pi(t)_0 = \pi_0 = \left[\sum_{i=0}^{S-1} \frac{\left(\frac{\lambda}{\mu}\right)}{i!} + \frac{\left(\frac{\lambda}{\mu}\right)^S}{S!} \frac{S\mu}{S\mu-\lambda} \right]^{-1} \tag{7}$$

$$\lim_{t\to\infty} \pi(t)_j = \pi_j = \begin{cases} \frac{\left(\frac{\lambda}{\mu}\right)^j}{j!}\pi_0, & \text{if } j <= S \\ \frac{\left(\frac{\lambda}{\mu}\right)^j}{j!j^{n-s}}\pi_0, & \text{otherwise} \end{cases} \tag{8}$$

For a constant arrival rate $\lambda = 1$, service rate $\mu = 2.5$ and with $S = 3$, $\pi_0 = 0.6701$, $\pi_1 = 0.2680$, $\pi_2 = 0.0536$ which agrees with equation (7).

Then the code was tested on a time dependent $\lambda(t)$. To test something with known limiting behaviour of $E(n(t))$, $\lambda(t) = 10e^{-}t$ was chosen. Here we would expect $E(n(t))$ to increase initially since $\lambda(0) > S\mu$, and then as t increases $E(n(t))$ should fall away to 0 as the arrival rate falls exponentially. We can see this does happen in figure 1.

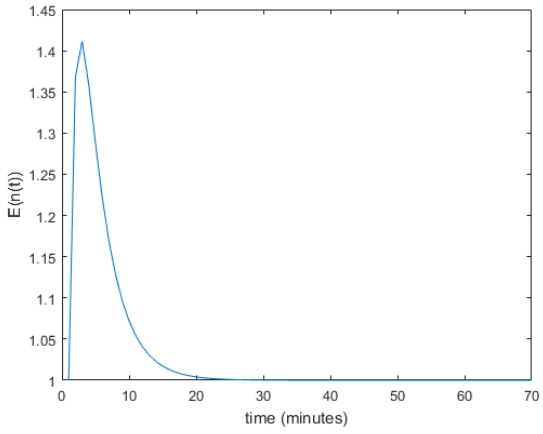

Figure 1: How the expected number in the queue varies with an exponentially decreasing arrival rate ($\lambda(t) = 10e^{-}t$)

1.2 Expected Queue Length with an Oscillating Arrival Rate

Now, the arrival rate varies sinusoidally with time. Specifically $\lambda(t) = \lambda(1 + \gamma sin(\frac{2\pi t}{720}))$. This resulted in a oscillating expected queue length. Figure 2 shows this oscillating queue length. It is clear that a larger number of servers decreases the expected queue length, but there are more subtle observations to be made. We can see a lag in the peaks of $E(n(t))$. This is due to a larger number of servers being able to get the queue under control more quickly. Also, in the case $S = 3$, there are huge spikes in the expected queue length. This is because

$$max(\lambda(t)) = 1.5 > S\mu = 1.2, (S = 3)$$

thus the queue length tends towards infinity, before regaining control when the arrival rate drops below the service rate again. The code for these calculations is shown in appendix C.

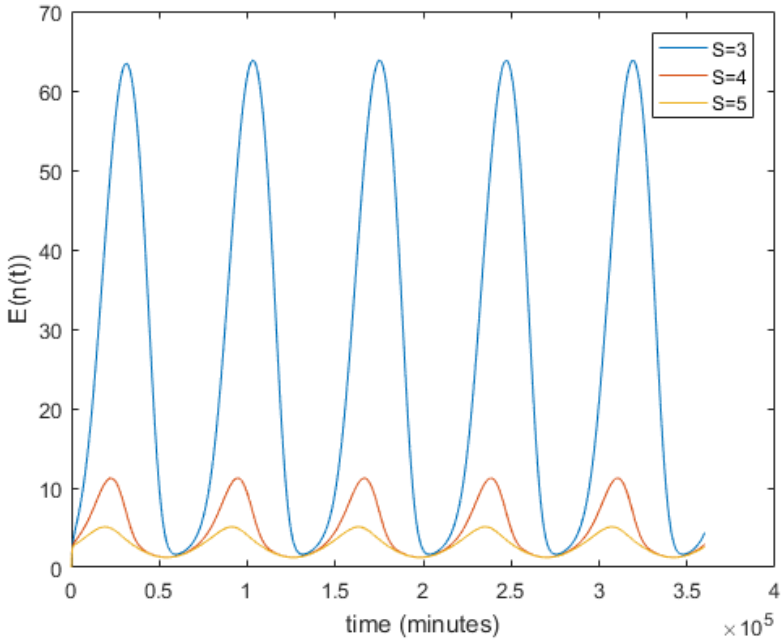

Figure 2: How the expected number in the queue varies over time with an oscillating arrival rate.)

1.3 Behaviour of the right-hand 10% tail of the distribution of $n(t)$.

The behaviour of the right-hand tail (RHS) behaves similarly to the expected number in the queue. In figure 3 we see a pattern very similar to figure 2 which is to be expected, however it is worth noting that here the number of servers has slightly less of an effect. For example, when we decrease the number of servers from 4 to 3, max(RHS) increases by a factor of 3.9 (97/25), But when we make this comparison with $\max(E(n(t)))$, it increases by a factor of 5.8. Similar conclusions can be drawn when making these comparisons between $S = 3$ and $S = 5$, and $S = 4$ and $S = 5$.

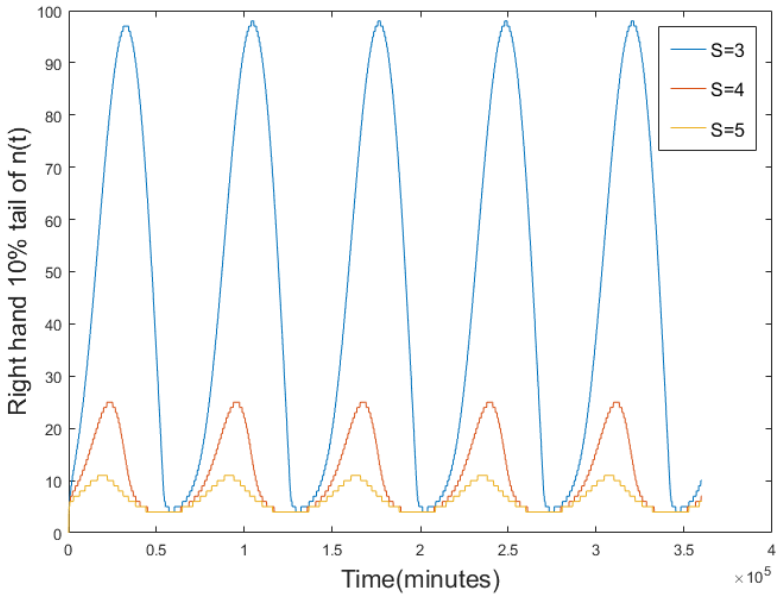

Figure 3: Behaviour of the right-hand 10% tail of the distribution of $n(t)$

2 Discrete Time Markov Decision Chains

A Discrete time Markov chain was used to find the optimal policy of a queuing system. We will define the 'policy' to be the length of queue at which the door will be shut. The optimal policy therefore will be the length of queue at which the door will be shut which maximises system reward. The optimal policy will be time dependent.

2.1 The System

Customers arrived into the system at a rate $\lambda(t)$. The arrivals are assumed to enter the system at the end of the period in which they arrive. The service is made by 1 server and service times have a geometric distribution with service completion probability μ . That is, the mean service time is $\frac{1}{\mu}$. Service completions also occur at the end of the period of service. The arrival and service processes are independent and mutually exclusive so that $\lambda(t) + \mu < 1$ for all time t.

If there are L or less customers in the queue, there are sufficient chairs to sit on. Every customer over this threshold number has to stand, and the system is therefore charged a rate c per period. The system earns d for every completed service. We can choose to open the waiting room, in which case customers flow through the system, or we can close the waiting room door, in which case no more customers arrive.

2.2 Formulation

We can begin to see how this problem can be formulated. There is a trade off. If we choose to leave the door closed, then no more customers will arrive and we avoid penalties for customers standing, however we run the risk that the queue becomes depleted meaning we cannot earn anything. Equally, if the door is left open all the time, the queue could become so large that we are actually paying out more money than we earn from each service.

The reward vector (9) contains the reward from being in each state. State 1 being 0 people in the system, and state 3 for example means there are 2 people in the system, so 1 person is being served and one person is queueing. The coding for this vector can be found in appendix F. The reward vector is the same for both open and closed because of assumptions presented in section 2.1.

$$r = \{0, \mu d, \mu d, ..., \mu d - c, \mu d - 2c, ..., \mu d - (n - (L + 1))c, ...\} \tag{9}$$

Since this is discrete time, a transition probability matrix is used, with $t = 1$. There must be two separate matrices, since the transition probabilities depend on the action taken i.e whether the door is open or not. transition matrix (10) represents the probabilities of moving between states when the waiting room door is open, and (11) represents this when closed. The coding for these matrices can be found in appendix D.

$$\begin{pmatrix} 1-\lambda(t) & \lambda(t) & 0 & 0 & 0 & ... & 0 & 0 \\ \mu & 1-\lambda(t)-\mu & \lambda(t) & 0 & 0 & ... & 0 & 0 \\ 0 & \mu & 1-\lambda(t)-\mu & \lambda(t) & 0 & ... & 0 & 0 \\ .. & .. & .. & .. & .. & .. & .. & ... \\ 0 & 0 & 0 & 0 & 0 & 0 & \mu & 1-\mu \end{pmatrix} \tag{10}$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & ... & 0 & 0 \\ \mu & 1-\mu & 0 & 0 & 0 & ... & 0 & 0 \\ 0 & \mu & 1-\mu & 0 & 0 & ... & 0 & 0 \\ .. & .. & .. & .. & .. & .. & .. & ... \\ 0 & 0 & 0 & 0 & 0 & 0 & \mu & 1-\mu \end{pmatrix} \tag{11}$$

There is no λ term in the matrix (11) since the door is closed and no one can arrive.

2.3 Solution

To find the optimal policy, backwards recursion was used:

- Step 0: $V_x(0) = 0$ and $V_x(1) = r_x$ for all x.
- Step 1: $1 \leq t \leq T - 1$

$$V_x(t + 1) = r_x + \sum p_{x,y} V_y(t) \text{ for all } x.$$

It follows, that at each time-step t, we should choose the action that maximises the reward.

2.4 Testing the Solution

The arrival rate was a function of time:

$$\lambda(t) = \lambda(1 + \gamma sin(\frac{2\pi t}{720}))$$

The code was then tested for $\lambda = 0.4$, $\gamma = 0.75$,mean service time of 2.4 periods, $L = 3$, $c = 1$, $d = 20$. The Matlab code for this can be found in appendix E.

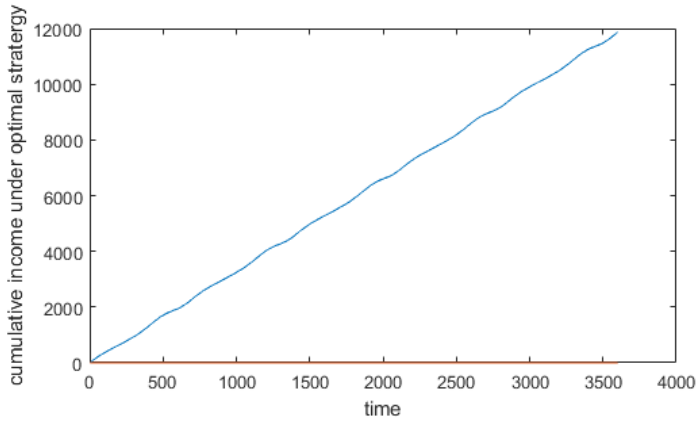

Figure 4: The cumulative income under the optimal policy

Figure 2.4 shows the cumulative income under the optimal policy. The oscillatory nature is due to the varying arrival rate. It is important to always make

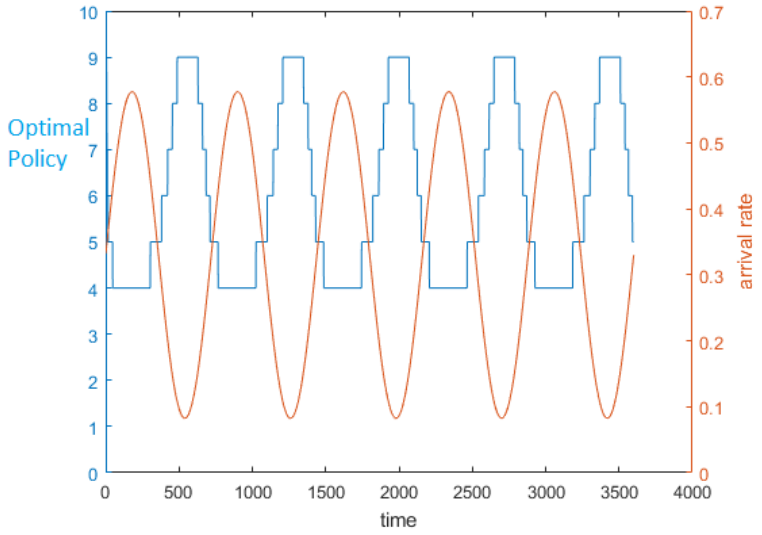

Figure 5: As the arrival rate increases, the optimal policy decreases as there is no need to hold as many people in reserve.

sure there is 1 person in the queue, so that to system can earn money. To ensure

that the probability of a queue length reaching 0 is small, the optimal policy holds more people 'in reserve' when the arrival rate is lower. It is interesting to note that the optimal policy never dips below 4. This means the system is always losing some amount of money due to people having to stand, but this is worthwhile so that the probability of having no one to serve is small.

The optimal policy changes throughout the day as the arrival rate oscillates, but there are 5 periods coinciding with 5 days, meaning that the optimal policy on any given day is always the same. The main piece of coding for this section can be seen in appendix G.

3 Further Work

Throughout this project the arrival rate had been deterministic in time, it follows a simple oscillating path. But what if the arrival rate has a stochastic element? For example adding some normal noise.

$$\lambda(t) = \lambda(1 + \gamma sin(\frac{2\pi t}{720}) + N(0, \sigma))$$

I expect that this extension will result in a optimal policy corresponding to a longer queue length, and the added length will be an increasing function of σ. Of course, this added noise is symmetric, and so at first glance we may not expect this result, however it is optimal for the system to pay a bit extra on average (because more people are standing) to ensure that the queue never becomes depleted.
When we add some noise to the arrival rate the uncertainty added will increase the probability that the queue becomes depleted, and therefore the system should hold more in 'reserve'.
This is an obvious yet challenging extension to the problem, however it may be more applicable to real world problems especially when trying to optimise queues in real time.

Appendices

A Transition Rate Matrix

```
function [Q] = Matrix(n,mu,lambda,S)
 Q=zeros(n); %n is the maximum length of the system
 Q(1,1:2)=[-lambda,lambda];
 Q(n,n-1:n)=[S*mu,-S*mu];
  for i = 2:n-1
      Q(i,i+1)=lambda;
      if(i<=S)
      Q(i,i-1) = (i-1)*mu;
      else
       Q(i,i-1) = S*mu;
      end
      Q(i,i)=-Q(i,i+1)-Q(i,i-1);
  end
end
```

B Evaluation of equation (1)

```
function [pi] = Task1A_a(n,mu,lambda,S,h,T)

pi = zeros(n); %n is the maximum length of the queue
pi(1,1) = 1; %start pi(0) with one person in the queue
for t=1:T/h %t is the number of time steps in time T

    pi(:,t+1) = pi(:,t) + (Matrix(n,mu,lambda,S)*(pi(:,t)
        ))*h;
    pi(:,t+1) = pi(:,t+1)/sum(pi(:,t+1)); %normalising pi
        vector
end
```

C Calculation of Expected Queue Length

```
function [states,top10index] = Task1A(n,mu,lambda,S,h,T)
tic;
%states = zeros(T/h,n);
%states(1,1) = 1;

statesa = zeros(T/h,n,3); %n is the maximum length of the
    queue
statesa(1,1,:) = 1; %start pi(0) with one person in the
   queue
expected = zeros(1, T/h,3);
top10 = zeros(1, T/h,3);
top10index = zeros(3,T/h); %A measure of the right hand
   said 10% tail of the distribution
a = 0:1:(n-1);

for t=1:T/h %t is the number of time steps in time T
    gamma = 0.5;
    lambdat = lambda*(1 + gamma*sin(2*pi*t*h/720));
    %lambdat = lambda*exp(-t*h); %Test function

    statesa(t+1,:,1) = statesa(t,:,1) + ((statesa(t,:,1))
       *Matrix(n,mu,lambdat,S))*h;
    statesa(t+1,:,2) = statesa(t,:,2) + ((statesa(t,:,2))
       *Matrix(n,mu,lambdat,S+1))*h;
    statesa(t+1,:,3) = statesa(t,:,3) + ((statesa(t,:,3))
       *Matrix(n,mu,lambdat,S+2))*h;
    %states(t+1,:) = states(t,:) + ((states(t,:))*Matrix(
       n,mu,lambda,S))*h;
    %test function

    expected(1,t,1) = statesa(t,:,1)*a';
    expected(1,t,2) = statesa(t,:,2)*a';
    expected(1,t,3) = statesa(t,:,3)*a';

    top10(1,t,1) = statesa(t,1,1);
    top10(1,t,2) = statesa(t,1,2);
    top10(1,t,3) = statesa(t,1,3);

    for i = 2:n
        if  top10(1,t,1) <= 0.9;
            top10(1,t,1) = statesa(t,i,1) + top10(1,t,1);
            top10index(1,t) = i;
        end
    end
```

```
38
39      for i = 2:n
40          if  top10(1,t,2) <= 0.9;
41              top10(1,t,2) = statesa(t,i,2) + top10(1,t,2);
42              top10index(2,t) = i;
43          end
44
45      end
46
47      for i = 2:n
48          if  top10(1,t,3) <= 0.9;
49              top10(1,t,3) = statesa(t,i,3) + top10(1,t,3);
50              top10index(3,t) = i;
51          end
52      end
53
54  end
55  x = 1:t;
56  y1 = expected(1,:,1);
57  y2 = expected(1,:,2);
58  y3 = expected(1,:,3);
59
60  %plot(x,y1)
61  plot(x, y1,x, y2,x, y3)
62  xlabel('time (minutes)')
63  ylabel('E(n(t))')
64  legend('S=3','S=4','S=5')
65
66  y4 = top10index(1,:);
67  y5 = top10index(2,:);
68  y6 = top10index(3,:);
69  plot(x, y4,x, y5,x, y6)
70  toc;
71  states = statesa(1:100:T,:,:);
72  %states = states(1:1/h:end,:);
```

D Transition Probaility Matrix

```
function [P] = Discrete(n,mu,lambda,S,t)
 P=zeros(n,n,2); %n is the maximum length of the system

 P(1,1:2,1)=[1-lambdat(lambda,t),lambdat(lambda,t)];%
     open
 P(n,n-1:n,1)=[S*mu,1-S*mu];% open

 P(1,1:2,2)=[1,0];% closed
 P(n,n-1:n,2)=[S*mu,1-S*mu];% closed


  for i = 2:n-1
      P(i,i+1,1)=lambdat(lambda,t); % open
      if(i<=S)
      P(i,i-1,1) = (i-1)*mu;% open
      P(i,i-1,2) = (i-1)*mu;% closed
      else
       P(i,i-1,1) = S*mu;% open
       P(i,i-1,2) = S*mu;% closed
      end
      P(i,i,1)=1-P(i,i+1,1)-P(i,i-1,1);% open
      P(i,i,2)=1-P(i,i+1,2)-P(i,i-1,2);% closed

  end
end
```

E Arrival Rate

```
function [lam] = lambdat(lambda,t)

gamma = 0.75;
lam = lambda*(1 + gamma*sin(2*pi*t/720));

end
```

F Reward Vector

```
function [rewardVec] = rewardVectors(d,c,mu,n,L)

rewardVec =  zeros(2,n);

    for j=2:(L+1)
        rewardVec(1,j) = mu*d; %open
        rewardVec(1,j) = mu*d; %closed
    endd

    for j=(L+2):n
        rewardVec(1,j) = mu*d-c*(j-(L+1)); %open
        rewardVec(1,j) = mu*d-c*(j-(L+1)); %closed
    end

end
```

G Optimal Policy

```
function [cumMoney, Threshold, Policy, VV, Money] = Task1B(n,
    mu, lambda, S, T, L, c, d)

t=1;
rwd1 = rewardVectors(d,c,lambda,mu,n,L,t);
V = zeros(T,n); %outputs reward for each policy
V(1,:) = rwd1(1,:); %step 0

R_tx = zeros(T,n);
cumMoney = zeros(T);
Money = zeros(T);
Threshold = zeros(T,1);
Policy = zeros(T,1);

for t=2:T
    for x=1:n
        rwd = rewardVectors(d,c,lambda,mu,n,L,t);
        Mtrx = Discrete(n,mu,lambda,S,t);
        temp = [rwd(1,x) + sum(Mtrx(x,:,1)*transpose(V(t
            -1,:))) , rwd(2,x) + sum(Mtrx(x,:,2)*transpose
            (V(t-1,:)))];


        if temp(1)> temp(2)
            V(t,x) = temp(1);
            R_tx(t,x) = 1; %1 if open
            Policy(t) = 1;
        else
            V(t,x) = temp(2);
            R_tx(t,x) = 0; %0 if closed
            Policy(t) = 0;
        end
    end

    Threshold(t) = sum(R_tx(t,:)); %number of people in
        queue
    cumMoney(t) = V(t,Threshold(t)); %cumultive income
    Money(t) = cumMoney(t)-cumMoney(t-1); %money made
        each timestep


end
VV = cumMoney(1:100:T);
end
```