

Math 350 - Homework 7 - Solutions

1. Write a program to generate the desired output for the model of Section 6.2. Use it to estimate the average time that a customer spends in the system and the average amount of overtime put in by the server, in the case where the arrival process is a Poisson process with rate 10, the service time density is

$$g(x) = 20e^{-40x}(40x)^2, \quad x > 0$$

and $T = 9$. First try 100 runs and then 1000.

First note that $g(x)$ is the density function of a gamma random variable with parameters $n = 3$ and $\lambda = 40$. Page 70 of the textbook (Example 5c) discusses how to simulate gamma random variables. The Matlab commands

$$Y = -(1/40) * \log(\text{prod}(\text{rand}(1, 3)));$$

can be used for this purpose. We will use it to generate a service time when needed in the program. (Recall that a gamma random variable with parameters n and λ is the sum of n i.i.d. exponential random variables with parameter λ .)

The following program implements the algorithm of Section 6.2 under the special assumptions about the distribution of arrival and service times of this exercise. I have added a variable that gives the server's idle times, so this program can be used in the second problem, too.

```
function [S, A, D, Overtime, Idle]=simple_queue(T)
%Input: T is the time of closing (from initial time 0).
%
%Output: S is the full sequence of events.
%       A is the sequence of arrival times.
%       D is the sequence of departure times.
%       Overtime is the times past T when last customer departs.
%       Idle is the amount of time the server remains idle.

%Time of first arrival, t_A.
t_A = -(1/10)*log(rand(1));
%Time of first departure, t_D.
t_D = 1/eps;

%Initialize variables:
t = 0; %Time of the most recent event.
N_A = 0; %Number of arrivals by time t.
N_D = 0; %Number of departures by time t.
state = 0; %Number of customers in the system at time t.
E = 0; %Event number.
```

```

%The variable Stop is either 0 or 1. It
%is initially 0, and is set to 1 when we want the program to stop running.
%A 'break' command could be used instead.
Stop = 0;
A     = zeros(1,500); %A(i) is the arrival time of customer i.
D     = zeros(1,500); %D(i) is the departure time of customer i.
S     = zeros(1,500); %S(j) is the state at each event time.
%We don't know the total number of customers beforehand. I've set aside A
%and D with length 500, but the length these vectors will only be fixed
%at the end of a run of the program.
Idle  = 0; %Total idle time of the server;

while Stop==0
    if t_A<t_D & t_A<T
        %Next event is an arrival:
        t_old = t;
        t     = t_A;
        N_A   = N_A + 1;
        %N_A serves as a number label for each arriving customer, like the
        %customer number you pick in a post office as you come in.
        state = state + 1;
        %Update the time of next arrival and departure:
        t_A   = t - (1/10)*log(rand(1));
        t_D   = t_D*(state>1) + (t - (1/40)*log(prod(rand(1,3))))*(state==1);
        %When state>1, t_D was already set; when state==1, we need to
        %generate the departure time of the new customer.
        %Collecting output data:
        A(N_A) = t;
        E     = E+1;
        S(E)  = state;
        Idle  = Idle + (state==1)*(t-t_old);
    elseif t_D<t_A & t_D<T
        %Next event is a departure:
        t     = t_D;
        state = state - 1;
        N_D   = N_D + 1;
        %Update t_D:
        Y     = -(1/40)*log(prod(rand(1,3)));
        t_D   = (state==0)*(1/eps) + (state>0)*(t+Y);
        %Collecting output data:
        D(N_D) = t; %Note that N_A and N_D are different as a function of
        %t, but the N_A and N_D associated to each customer are the same
        %since customers are served of a first come first serve basis.
        E     = E+1;
        S(E)  = state;
    elseif min(t_A,t_D)>T & state>0

```

```

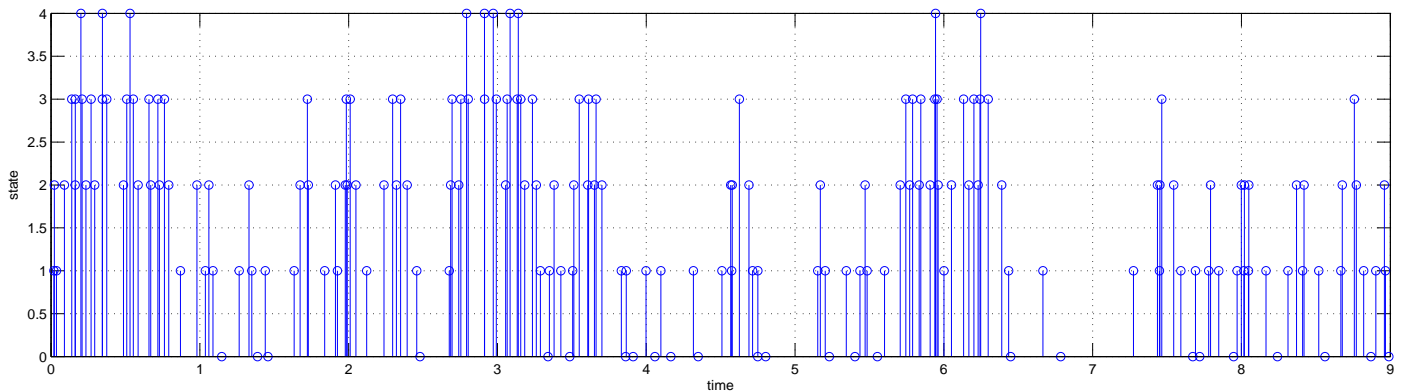
    %Closing time has passed but there are customers in the system.
    %The next events are necessarily departures.
    t      = t_D;
    state  = state - 1;
    N_D    = N_D + 1;
    Y      = -(1/40)*log(prod(rand(1,3)));
    t_D    = t + Y;
    D(N_D) = t;
    E      = E+1;
    S(E)   = state;
elseif min(t_A,t_D)>T & state==0
Stop = 1;
%Collecting data:
A      = A(1:N_A);
D      = D(1:N_D);
Overtime = max(0,t-T);%The time past T that last customer departs.
S      = S(1:N_A+N_D);
end
end
end

```

The following graph gives a good sense of what happens in one run of the program. I got it by first running the program once and then calling the command

```
stem(sort([A,D]),S,'o')
```

It shows the state of the system at each event time. Notice, in particular, that the queue never grew longer than 3 persons long (with one extra person being helped at the server.)



Notice that each entry of the vector $D - A$ gives the amount of time the respective customer spent in the system (from the time of arrival A to the time of departure D). The average time in the system is then $\text{sum}(D - A)/\text{length}(A)$. The next script gives the average time spent in the system, the average overtime of the server, and the average idle time (needed for the second problem).

```
Time_in_system = 0;
```

```

Over          = 0;
Idle_time     = 0;
m             = 1000;
for i=1:m
    [E,S,A,D,Overtime,Idle]=simple_queue(9);
    Idle_time     = Idle_time + Idle;
    Over          = Over + Overtime;
    Time_in_system = Time_in_system + sum(D-A)/length(A);
end
Time_in_system = Time_in_system/m
Over           = Over/m
Idle_time      = Idle_time/m

```

Conclusion. The above gave the following values (for 100 runs): mean overtime of server = 0.1436, and mean time customer spends in system = 0.2114. For $m = 1000$, I obtained the following (respective) values: 0.1511 and 0.2108. Although not asked, we also get the idle time of server: 2.3844.

Are these values reasonable? The serving time for each customer is on average $3/40$ (the mean of a gamma random variable with parameters 3 and 40). On average we expect approximately 90 customers to arrive into the system, as they arrive at the rate of 10 per unit time over the course of 9 time units. So the total serving time is approximately $(3/40) \times 90 = 6.75$. The program is giving approximately 2.38 time units of server idle time, so total time server spent in the system is $6.75 + 2.38 = 9.13$, suggesting an overtime of approximately 0.13. On the other hand, the simulated value for the overtime was approximately 0.15. It seems that the program is behaving well.

2. *Suppose in the model of Section 6.2 that we also wanted to obtain information about the amount of idle time a server would experience in a day. Explain how this could be accomplished.*

This has already been answered in problem 1.

3. *Suppose that jobs arrive at a single server queueing system according to a nonhomogeneous Poisson process, whose rate is initially 4 per hour, increases steadily until it hits 19 per hour after 5 hours, and then decreases steadily until it hits 4 per hour after an additional 5 hours. The rate then repeats indefinitely in this fashion—that is, $\lambda(t + 10) = \lambda(t)$. Suppose that the service distribution is exponential with rate 25 per hour. Suppose also that whenever the server completes a service and finds no jobs waiting he goes on break for a time that is uniformly distributed on $(0, 0.3)$. If upon returning from his break there are no jobs waiting, then he goes on another break. Use simulation to estimate the expected amount of time that the server is on break in the first 100 hours of operation. Do 500 simulation runs.*

- (a) **Preliminaries.** We first observe that the function $\lambda(t)$ can be written in the following form:

$$\lambda(t) = 19 - 3|\text{mod}(t, 10) - 5|$$

where $\text{mod}(t, 10)$ is the remainder of division by 10. (Notice that this is a periodic function of period 10.) For convenience, let us define a function file for it:

```

function l=lambda(t)
l=19-3*abs(mod(t,10)-5);

```

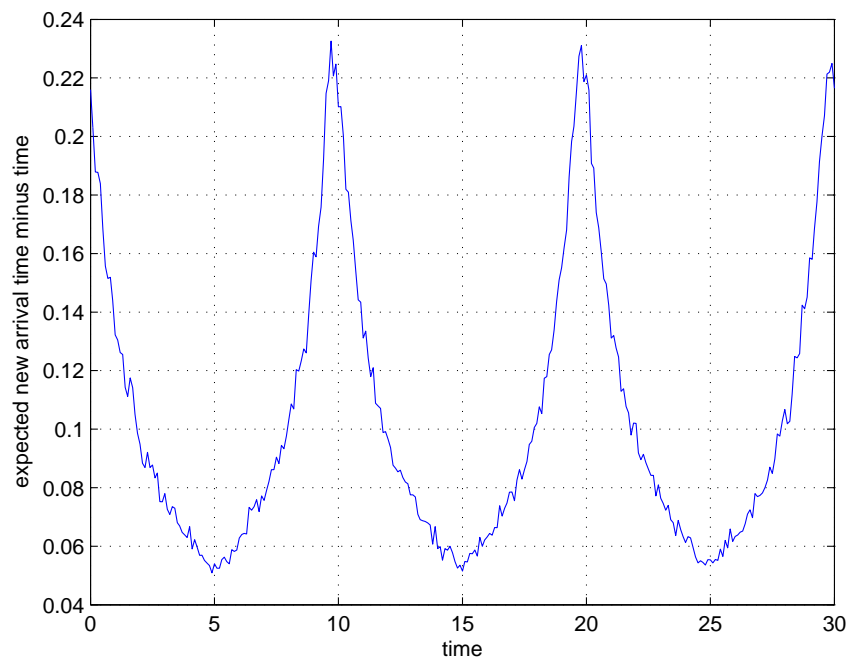
The following function script generates the next arrival time:

```

function T_next=new_arrival_time(t)
%Input: time t of last arrival
%Output: time T_next of next arrival after t
%
%It is assumed that arrival times follow a Poisson
%process with the intensity function defined in problem 3 of HW 7.
U_2=2;
while U_2 > lambda(t)/19
    U_1 = rand(1);
    U_2 = rand(1);
    t = t - (1/19)*log(U_1);
end
T_next=t;

```

To get a sense of what this function is doing, we simulate the data for the graph of the difference of the mean next arrival time minus the time, i.e., the expected waiting time to next arrival.



The above plot was done using the following script:

```

t=0:0.1:30;
V=zeros(1,301);
for i=1:301
    R=zeros(1,1000);
    for j=1:1000
        R(j)=new_arrival_time(t(i));
    end
    V(i)=sum(R)/1000;
end

```

```

end
plot(t,V-t)

```

- (b) **Description of the algorithm.** Before showing the program, I give an overview of the algorithm I intend to use. A possible choice for the set of **states** is $\{0, 1, 2, \dots\} \times \{0, 1\}$, so that a state is a pair (n, i) where n represents the number of jobs (either waiting in queue or under service) and i indicates whether the server is at service ($i = 1$) or on a break ($i = 0$). Notice that it makes sense to regard $(0, 1)$ and $(0, 0)$ as representing the same state since as soon as the server finishes a job and the waiting queue is empty, he immediately takes a break. Similarly, upon coming back from a break and noticing that the queue is still empty, he immediately returns to the break. So there is not time separation between $(0, 0)$ and $(0, 1)$. I indicate this identification of the two states by writing $(0, 0) \equiv (0, 1)$. If the server returns from a break and after finding no job to work on immediately leaves again on a break, I will count each break segment of mean 0.3 as a separate leave. It would be also O.K. to combine contiguous breaks as a single leave.

State transitions occur when:

- i. A new job arrives, whether the server is present or not: $(n, i) \rightarrow (n + 1, i)$;
- ii. A job is completed and other jobs are waiting: $(n, 1) \rightarrow (n - 1, 1)$, where $n \geq 2$;
- iii. A job is completed and no other job is waiting: $(1, 1) \rightarrow (0, 1) \equiv (0, 0)$;
- iv. The server returns from a break and finds waiting jobs: $(n, 0) \rightarrow (n, 1)$, where $n \geq 1$;
- v. The server returns from a break and finds no job waiting: $(0, 0) \rightarrow (0, 1) \equiv (0, 0)$.

Event times can be one of the following:

- i. t_A = time of a new job arrival;
- ii. t_D = time that a job is completed;
- iii. t_B = time when the server returns from a break.

Counter variables I will keep track of are:

- i. N_A = job arrival number (or the number of arrivals by time t);
- ii. N_D = job completion number (or the number of jobs completed by time t);
- iii. N_B = break number (or the number of server breaks by time t).

Output variables we may like to collect are:

- i. $A(i)$ = arrival time of job i ;
- ii. $D(i)$ = completion time of job i ;
- iii. $B_0(j)$ = time server leaves for his j th break;
- iv. $B_1(j)$ = time server returns from his j th break.

General organization of the program. We set a “while loop” that keeps running until the minimum of the t_A, t_D, t_B becomes greater than T . At each run of the loop these times are updated by simulating the non-homogeneous Poisson time for t_A , the exponential random variable with parameter 25 for t_D and a uniformly distributed time between 0 and 0.3 for t_B . The updating of the variables depends on three 3 cases:

- i. t_A is the least among t_A, t_B , and t_D (next event is a new job arrival);
- ii. t_D is the least among t_A, t_B , and t_D (next event is a job completion);
- iii. t_B is the least among t_A, t_B , and t_D (next event is that the server returns from a break);

- (c) **The program.** The following function file obtains a single run of the process. (A period of $T = 100$ units of time.)

```

function [A, D, B_0, B_1, E, S, J, K]=simple_queue2(T)
%Input: T is the stopping time when jobs and arrivals are interrupted.
%
%Output: A is the sequence of arrival times.
%        D is the sequence of completion times.
%        B_0 is the sequence of times server leaves for a break.
%        B_1 is the sequence of times server returns from a break.

%Time of new job arrival, t_A.
t_A = new_arrival_time(0);
%Time of server return from break:
t_B = 0.3*rand(1);
%Time of job completion, t_D.
t_D = 1/eps;

%Initialize variables:
t = 0; %Time of the most recent event.
N_A = 0; %Number of arrivals by time t.
N_B = 1; %Number of breaks by time t. I assume server begins on a break.
N_D = 0; %Number of job completions by time t.
A = zeros(1,5000); %A(i) is the arrival time of job i.
D = zeros(1,5000); %D(i) is the completion time of job i.
B_0 = zeros(1,5000); %B_0(j) is the starting time of the j-th break.
B_1 = zeros(1,5000); %B_1(j) is the ending time of the j-th break.
E = zeros(1,5000);
S = zeros(1,5000);
J = zeros(1,5000);
K = 1; %K counts the number of events by time t.
E(K) = 0; %E is the time of the k-th event.
S(K) = 0; %S is the number of jobs at k-th event time.
J(K) = 0; %J is the state of the server at k-th event time.
B_0(N_B) = 0;

while min(t_A, min(t_B, t_D))<=T
    K = K+1;
    if t_A<t_D & t_A<t_B
        %Next event is an arrival:
        t = t_A;
        N_A = N_A + 1;
        A(N_A) = t;
        E(K) = t;
        S(K) = S(K-1)+1;
        J(K) = J(K-1);
        %Update the time of next arrival and departure:

```

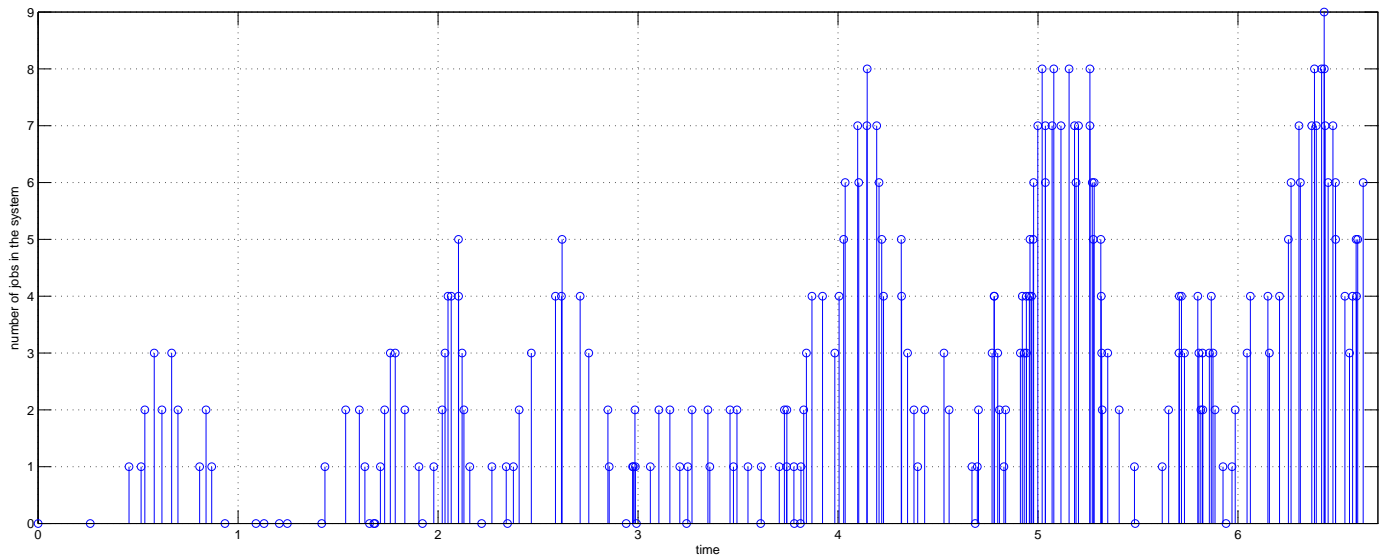
```

t_A    = new_arrival_time(t);
t_D    = (t_D*(S(K)>1) + ...
          (t-(1/25)*log(rand(1)))*(S(K)==1))*(J(K)==1)+(1/eps)*(J(K)==0);
elseif t_D<t_A & t_D<t_B
    %Next event is a job completion:
    t      = t_D;
    N_D    = N_D + 1;
    E(K)   = t;
    S(K)   = S(K-1)-1;
    J(K)   = (S(K)>0);%If S(K)==0 server goes on a break.
    %Update t_D and t_B:
    t_D    = (S(K)==0)*(1/eps) + (S(K)>0)*(t-(1/25)*log(rand(1)));
    t_B    = (J(K)==0)*(t+0.3*rand(1)) + (1/eps)*(J(K)==1);
    %Collecting output data:
    D(N_D) = t;
    if S(K)==0
        N_B      = N_B+1;
        B_0(N_B) = t;
    end
elseif t_B<t_A & t_B<t_D
    t      = t_B;
    B_1(N_B) = t;
    E(K)   = t;
    S(K)   = S(K-1);
    J(K)   = (S(K)>0);%If S(K)==0 server returns to break.
    if S(K)==0
        N_B      = N_B+1;
        B_0(N_B) = t;
    end
    %Update t_D and t_B:
    t_D    = (S(K)==0)*(1/eps) + (S(K)>0)*(t-(1/25)*log(rand(1)));
    t_B    = (S(K)==0)*(t+0.3*rand(1)) + (1/eps)*(J(K)==1);
end
end
A = A(1:N_A);
D = D(1:N_D);
B_0 = B_0(1:N_B);
if J(K)==0
    B_1(N_B)=T;
end
B_1 = B_1(1:N_B);
E=E(1:K);
S=S(1:K);
J=J(1:K);

```

The following graph gives the number of jobs in the system as a function of the times of the first 300 state changes. (When the server arrives from a break, the number of jobs at that time does not change.) The graph was obtained as follows:

```
[A, D, B_0, B_1, E, S, J, K]=simple_queue2(100);
stem(E(1:200),S(1:200))
```



We wish now to obtain the mean time when the server is on a break in the first 100 hours. The following script does it:

```
m=500;
BT=0;
for i=1:m
    [A, D, B_0, B_1, E, S, J, K]=simple_queue2(100);
    BT=BT+sum(B_1-B_0);
end
MBT=BT/m

MBT =
```

53.9864

Conclusion: The amount of time the server is on a break during those 100 time units is shown by simulation to be on average approximately 53.98. (More than half the time.)

Is this value reasonable? Jobs arrive at a mean rate of $(19 + 4)/2 = 11.5$, so over the period of 100 time units we expect to see approximately 1150 jobs. It takes about $1/25 = 0.04$ time units for each job to be completed, so the server will be busy during $1150 \times 0.04 = 46$ time units. This means that the time spent on break is approximately $100 - 46 = 54$. This is in very close agreement with the simulation.