Computer Engineering 4DK4

Lab 1: Performance of Single Server Queueing Systems

Instructor: Prof. T. Todd

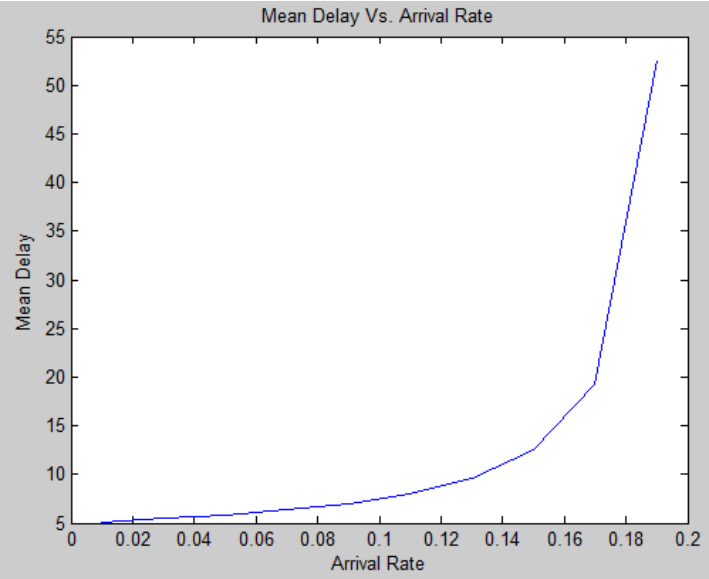
Moshiur Howlader - 1316948

Ankur Bargotra - 1306226

Experiment Results:

2) Below is the plot of Mean Delay Vs. Arrival Rate

Produced using seeds 6716759, 5259140, 3859164, and 1316948 (one of the student ID's).



Question 2 Data with random seed average values	
Arrival Rates	Mean Delay
0.01	5.1316
0.03	5.4413
0.05	5.8336
0.07	6.3465
0.09	7.0462
0.11	8.0571
0.13	9.6461
0.15	12.5071
0.17	19.1869
0.19	52.6519

When the arrival rate of the customers is lower, this implies that the queue (or the lineup) is being filled slower. Because there is more time for the queue to be cleared, the mean delay is lower at low values of arrival rates. Utilization factor is proportional to the mean delay. Higher/lower utilization percentage, longer/shorter mean delay. Also mean number of customer in the system has similar proportionality with mean delay. Higher mean delay leads to higher number of customers stuck in the system, waiting to be processed. This model definitely illustrates a single server queueing system which models customers coming in line one by one to have the server process or provide service before departure.

Equation (1):

$$0 < \lambda * X < 1$$

Alternatively, $0 < \lambda < 1/X$

Where λ is ARRIVAL_RATE and *X is SERVICE_TIME. Equation 1 needs to be satisfies for a stable system.*

As the arrival rate approaches the equality limit of 0.2 (axis asymptote), the mean delay increases to a certain large finite mean delay value depending on the number of customer to serve. For 50 million customers, the mean delay value for arrival rate of 0.199999 was 6336.096456. For 500

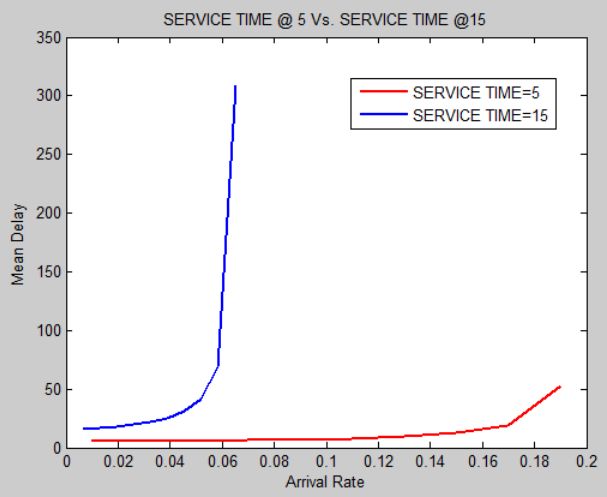
million customers, the mean delay value was 92246.103585. The mean delay curve shows the effect of increasing delays as arrival rates of customers increase.

3) A slight increase of ARRIVAL_RATE above 1/SERVICE_TIME (0.21) lead to a great increase in mean delay, even more than ARRIVAL_RATE approaching 1/SERVICE_TIME. The stability of Eqn 1 is broken. For example, ARRIVAL_RATE=0.21, and customer of 10000 led to a mean delay of 1935.698071. An increase of the customer count of 1000000 or 100000000 led to a mean delay of 124399.408576 or 12491303.893257 respectively. It seems the run length has an approximately linear proportionality (increase by factor of 10 translates to increase by a factor of 10).

The single server queuing system after a certain rate of arrival rate and customer size, simply does not have the capability or the SERVICE_TIME speed to process such enormous size of customer and arrival speed. Because ARRIVAL_RATE > 1/SERVICE_TIME, the queue is growing faster than the speed SERVICE_TIME can reduce that queue size. Because the product of ARRIVAL_RATE and SERVICE_TIME is essentially the rate of grow factor of the queue size, it is critical that this product is less than 1 for the system to be stable.

4) SERVICE_TIME is set to 15 for this portion of the experiment.

Produced using seeds 6716759, 5259140, 3859164, and 1316948 (one of the student ID's).



Question 4 Data with random seed average values	
Arrival Rates	Mean Delay
0.0065	15.8105
0.013	16.8175
0.0195	18.1017
0.026	19.7966
0.0325	22.1369
0.039	25.5783
0.0455	31.1339
0.052	41.6192
0.0585	68.8186
0.065	308.8297

Based on the results of SERVICE_TIME for 5 and 15, we see that both curves follow a similar pattern of a curve reaching its asymptote of the 1/SERVICE_TIME factor. Only difference is small increase in ARRIVAL_RATE for the SERVICE_TIME=15 significantly increases Mean Delay, whereas for SERVICE_TIME=5, similar ARRIVAL_RATEs only increase by a fraction of the SERVICE_TIME=15 case. Most likely cause of this is the fact the higher value of SERVICE_TIME results in a delay of clearing the queue, which compounds the buildup of the queue when arrival rates increase by even the slightest amount. This is seen in SERVICE_TIME=15 case.

5) Using a Matlab script, the analytic computation of the dMD1 equation were computed:

Q2 SERVICE_TIME=5	Q2 SERVICE_TIME=15
5.131578947	15.81024931
5.441176471	16.81677019
5.833333333	18.10070671
6.346153846	19.79508197
7.045454545	22.13414634
8.055555556	25.57228916
9.642857143	31.12204724
12.5	41.59090909
19.16666667	68.7244898
52.5	307.5

a) Analytical Results

Q2 Experimental Mean Delay	Q4 Experimental Mean Delay
5.1316	15.8105
5.4413	16.8175
5.8336	18.1017
6.3465	19.7966
7.0462	22.1369
8.0571	25.5783
9.6461	31.1339
12.5071	41.6192
19.1869	68.8186
52.6519	308.8297

b) Experimental Results

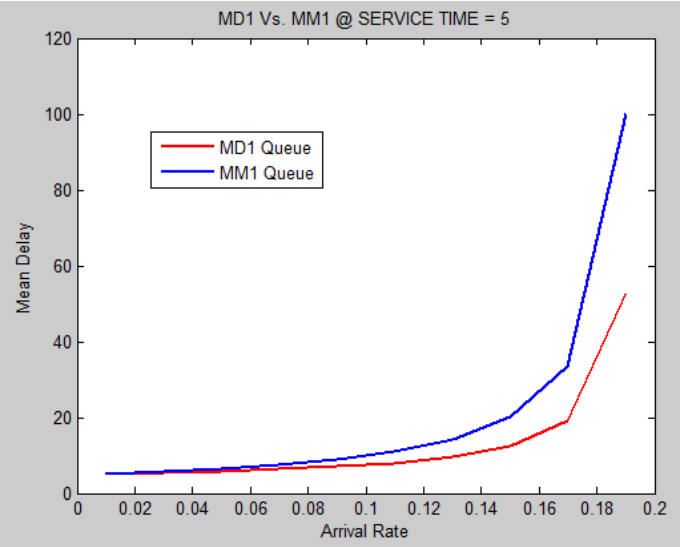
The experiment/simulation results are very close to the analytic results. For example, comparing dMD1 equation to the experimental value for ARRIVAL_RATE of 0.15 and SERVICE_TIME=5:

δ = Abs[(12.5071-12.5)/12.5] = 5.6800e-04 or 0.0568% error. Very negligible.

6) This is for an M/M/1 Single Server Queue. Note in the portion of the C code wherever "SERVICE_TIME" appeared to service a fixed service time for the arriving customers, those portions of code (3 instances) were changed to exponential_generator((double)SERVICE_TIME). This change served to provide exponentially distributed service time instead of fixed times.

Part a) for replica of question 2 procedure:

SERVICE_TIME = 5



	SERVICE_TIME = 5	
ARRIVAL_RATE	MD1	MM1
0.01	5.1316	5.2626
0.03	5.4413	5.8818
0.05	5.8336	6.6663
0.07	6.3465	7.6919
0.09	7.0462	9.0918
0.11	8.0571	11.11
0.13	9.6461	14.2834
0.15	12.5071	20.0037
0.17	19.1869	33.3574
0.19	52.6519	100.1538

Based on the results, MD1 (with fixed time SERVICE_TIME) has greater performance in terms of the less mean delay with respect to arrival rate compared to MM1 (with exponential service time). From the lecture, the bus example of fixed time vs exponential is simulated in this example. It is better for service to be fixed, then to have exponentially distributed service times.

Part b) for question 5 procedure

Question 2 and 4 required the simulation of MD1 queue data for SERVICE_TIME = 5 and 15 respectively. Then the verification of those MD1 queue results analytically. The newly simulated values for MM1 data, for SERVICE_TIME = 5 and 15, will be used to verify analytically whether these simulated numbers for MM1 data is correct. First the simplified delay equation for M/M/1 queue is:

$$\frac{\bar{X}}{(1-\rho)}$$

Using a Matlab script, the analytic computation of the dMM1 equation were computed:

MM1 Queue Analytical Results	
SERVICE_TIME=5	SERVICE_TIME=15
5.263157895	16.62049861
5.882352941	18.63354037
6.666666667	21.20141343
7.692307692	24.59016393
9.090909091	29.26829268
11.11111111	36.14457831
14.28571429	47.24409449
20	68.18181818
33.33333333	122.4489796
100	600

a) Analytical Result of MM1 Queue

MM1 Queue Experimental Result	
SERVICE_TIME=5	SERVICE_TIME=15
5.2626	16.6191
5.8818	18.6307
6.6663	21.1981
7.6919	24.5894
9.0918	29.2609
11.11	36.1464
14.2834	47.259
20.0037	68.2021
33.3574	122.4508
100.1538	600.3592

b) Simulation Result of MM1 Queue

Based on the theoretical results from Matlab, the results are very close to the experimental results from NetBeans C compiler.

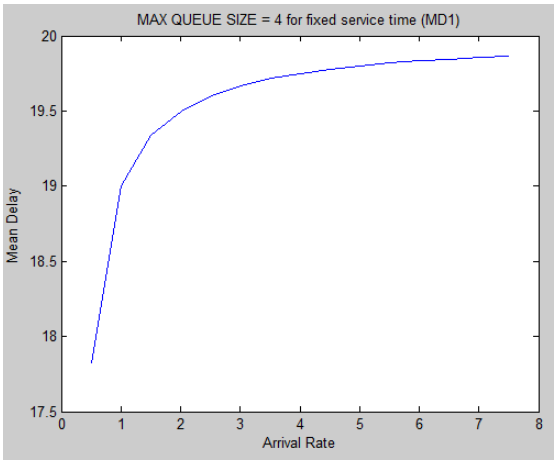
7. The system for this question was done with fixed service time (MD1), and with Random Seed of 1316948.

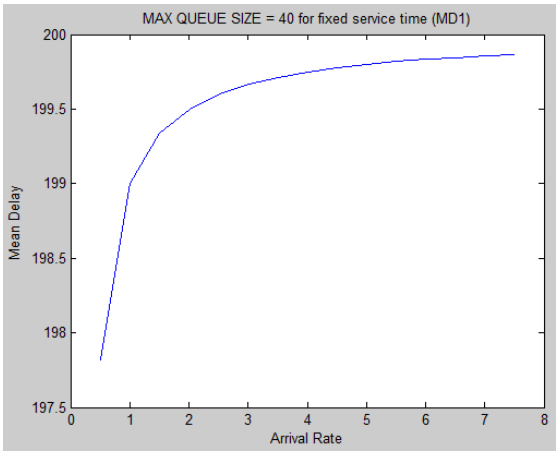
When SERVICE_TIME = 5, we adjusted the MAX_QUEUE_SIZE between 4, 40 and 400:

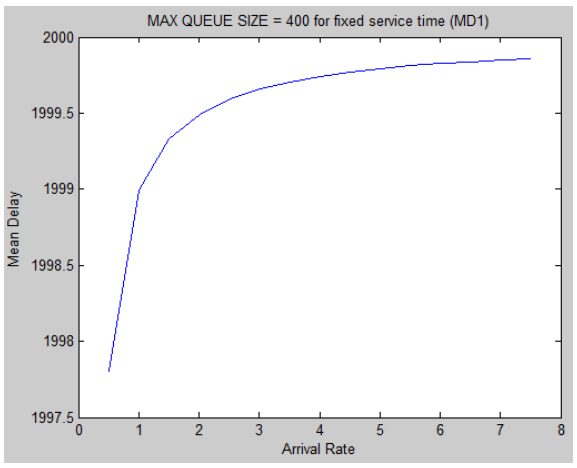
Arrival_Rate was varied between 0 to 8. For each case the asymptote reached around MAX_QUEUE_SIZE*5 (seconds). Note mean delay was calculated similarly to previous questions: mean_delay = integral_of_n/total_served.

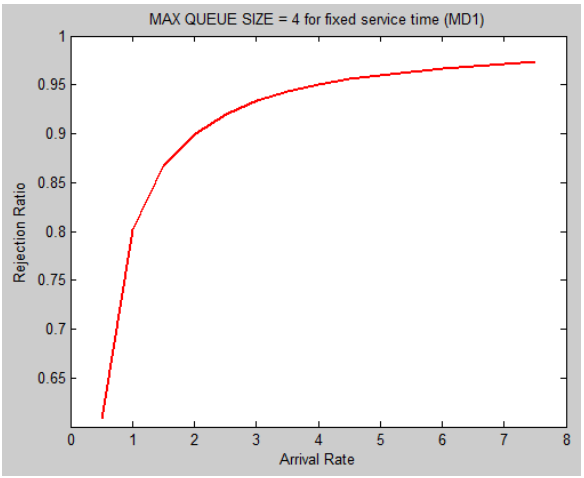
Results for Q7:

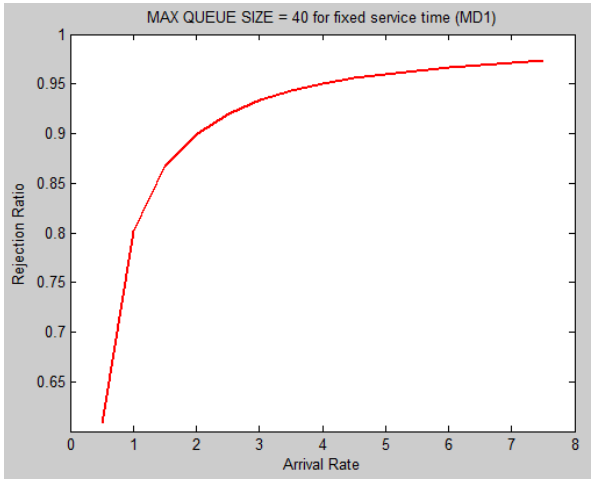
Note, look at end of appendix for list of data used to plot the following graphs!:

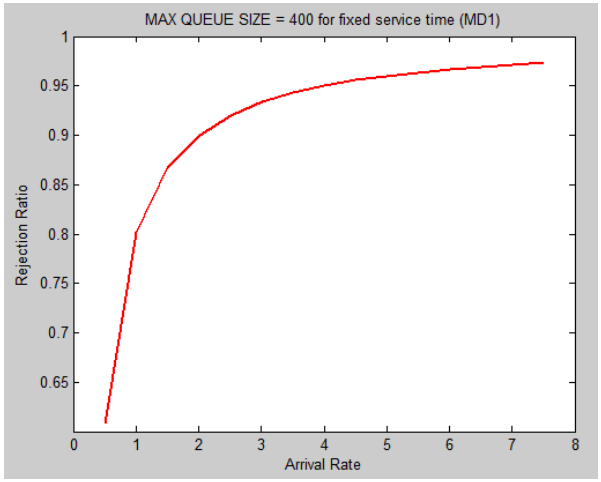












Q7 contd) Based on the results from simulation, the imposed queue size MAX_QUEUE_SIZE of the single server queue system reaches an asymptote at a certain arrival rate. After that arrival rate (in our case approximately 7.5), the mean delay no longer increases. Unlike the case from before with no queue size limit (infinite queue size), the mean delay stagnates or ceases to increase in value after a certain arrival rate is achieved. This is reasonable as limiting queue size limits how fast large number of customer can be served. As we see in the graph, the speed or size of mean delay approaches a limit after a certain arrival rate is achieved, regardless how fast arrival rate is.

Regarding other system parameters, mean number in system is constant at around the MAX_QUEUE_SIZE ± fraction of a percent. As the mean delay asymptote is approached, rejection ratio increases to nearly 100%. As the mean delay asymptote is approached, (for queue size 5) the number of customer service decreased by 97% (customer served ration = 1331510/19601402) as we went from 0 arrival rate to 7.5 arrival rate. From arrival rate 0 to 3.5, decrease of 66% (customer served ratio =1331510/3983922). The customer served decreases exponentially. Other parameters were relatively constant regardless of arrival rate.

Appendix:

Below was the code used for Q1 to Q4. Few parameter changes were made for the for-loop and the counter variables at the beginning few codes to output required results. The seeds used were 6716759 5259140 1316948 3859164.

```
/*
 * Simulation Parameters
 */

//6716759 5259140 1316948 3859164
#define RANDOM_SEED 1316948
#define NUMBER_TO_SERVE 50e6

#define SERVICE_TIME 15 //or 5
#define BLIP_RATE 10000000

int main()
{
  int numIterate;
  float ARRIVAL_RATE = 0;
  for(numIterate=0; numIterate<10; numIterate++){
  printf("RANDOM_SEED = %d\n", RANDOM_SEED);
    ARRIVAL_RATE+=0.0065; //change this value according to the intervals
required
  printf("ARRIVAL_RATE = %f\n", ARRIVAL_RATE);
  double clock = 0; /* Clock keeps track of simulation time. */
  /* System state variables. */
  int number_in_system = 0;
  double next_arrival_time = 0;
  double next_departure_time = 0;
  /* Data collection variables. */
  long int total_served = 0;
  long int total_arrived = 0;
  double total_busy_time = 0;
  double integral_of_n = 0;
  double last_event_time = 0;
  random_generator_initialize(RANDOM_SEED);
  while (total_served < NUMBER_TO_SERVE) {
   if(number_in_system == 0 || next_arrival_time < next_departure_time) {
     clock = next_arrival_time;
     next_arrival_time = clock + exponential_generator((double)
1/ARRIVAL_RATE);
     integral_of_n += number_in_system * (clock - last_event_time);
     last_event_time = clock;
     number_in_system++;
     total_arrived++;
     if(number_in_system == 1) next_departure_time = clock + SERVICE_TIME;
   } else {
     clock = next_departure_time;
     integral_of_n += number_in_system * (clock - last_event_time);
     last_event_time = clock;
     number_in_system--;
```

```
      total_served++;
      total_busy_time += SERVICE_TIME;
      if(number_in_system > 0) next_departure_time = clock + SERVICE_TIME;
      if (total_served == 50000000)
          printf("Customers served = %ld (Total arrived = %ld)\r",
           total_served, total_arrived);
    }
  }
  /* Output final results. */
  printf("\nUtilization = %f\n", total_busy_time/clock);
  printf("Fraction served = %f\n", (double) total_served/total_arrived);
  printf("Mean number in system = %f\n", integral_of_n/clock);
  printf("Mean delay = %f\n\n", integral_of_n/total_served);
}
  return 0;
}
```

Question 5 was done via Matlab:

```
%compute Analytic Result Q5 dMD1
%use question 2 results first, SERVICE_TIME=5
%use question 4 results second, SERVICE_TIME=15
% dMD1
arrival_rate_5 =
[0.0100,0.03000,0.0500000,0.0700000,0.09000,0.11000,0.130000,0.15000,0.17000,
0.19000];
mean_delay_5 =
[5.13160,5.44130,5.83360,6.34650,7.04620,8.05710,9.64610,12.50710,19.18690,52
.65190];
arrival_rate_15 =
[0.0065000,0.0130000,0.01950000,0.02600000,0.03250000,0.039000000,0.04550000,
0.052000,0.05850000,0.0650000];
mean_delay_15 =
[15.810500,16.81750,18.10170,19.79660,22.13690,25.57830,31.133900,41.61920,68
.818600,308.829700];

dMD1_results_5 = [0 0 0 0 0 0 0 0 0 0];
dMD1_results_15 = [0 0 0 0 0 0 0 0 0 0];

for i =1:10
X = 5;  %SERVICE_TIME
lambda = arrival_rate_5(i);
rho = lambda*X;


dMD1 = X*(2-rho)/(2*(1-rho));
dMD1_results_5(i) = dMD1;

end

for i =1:10
X = 15;  %SERVICE_TIME
lambda = arrival_rate_15(i);
```

```
rho = lambda*X;


dMD1 = X*(2-rho)/(2*(1-rho));
dMD1_results_15(i) = dMD1;

end

%check in matlab workspace for
%results of dMD1_results_5 and dMD1_results_15
```

First half of Q6 was done in a similar manner to Q1 to 4 code, second half of Q6 code as below:

```
%compute Analytic Result Q6 dMM1
%Similar to Q5 but using exponential(SERVICE_TIME) instead of servicing fixed
%service time
%use question 2 SERVICE_TIME=5 first
%use question 4 SERVICE_TIME=15 second
% dMM1
arrival_rate_5 =
[0.0100,0.03000,0.0500000,0.0700000,0.09000,0.11000,0.130000,0.15000,0.17000,
0.19000];
mean_delay_5 =
[5.2626,5.8818,6.6663,7.6919,9.0918,11.1100,14.2834,20.0037,33.3574,100.1538]
;
arrival_rate_15 =
[0.00650,0.0130,0.01950,0.02600,0.03250,0.039000,0.04550,0.0520,0.058500,0.06
500];
mean_delay_15 =
[16.6191,18.6307,21.1981,24.5894,29.2609,36.1464,47.2590,68.2021,122.4508,600
.3592];

dMM1_results_5 = [0 0 0 0 0 0 0 0 0 0];
dMM1_results_15 = [0 0 0 0 0 0 0 0 0 0];

for i =1:10
X = 5;  %SERVICE_TIME
lambda = arrival_rate_5(i);
rho = lambda*X;


dMM1 = X/(1-rho);
dMM1_results_5(i) = dMM1;

end

for i =1:10
X = 15;  %SERVICE_TIME
lambda = arrival_rate_15(i);
rho = lambda*X;


dMM1 = X/(1-rho);
```

```
dMM1_results_15(i) = dMM1;

end

%check in matlab workspace for
%results of dMM1_results_5 and dMM1_results_15
```

Question 7:

The major change for question 7 code was the logic to handle the case when queue was full and to reject the customer. An integral or sum of rejected n was accumulated for later meaningful calculations.

```
 /* Test if the next event is a customer arrival or departure. */
if(number_in_system == 0 || next_arrival_time < next_departure_time) {

  /*
   * A new arrival is occurring.
   */
 if(number_in_system == MAX_QUEUE_SIZE){ //full
 //reject arrival, will not get serviced

 clock = next_arrival_time;
 next_arrival_time = clock + exponential_generator((double) 1/ARRIVAL_RATE);
 //service the service time

 //update integral of number in system for this clock cycle
  integral_of_n += number_in_system * (clock - last_event_time);
  rejected_integral_of_n+= number_rejected * (clock - last_event_time);
  last_event_time = clock;

  //do not increment number_in_system
 number_rejected++;
 total_arrived++;

  }else{ //able to service arrival, service the arrival below
```

Note mean delay is calculated by integral_of_n/total_served:

```
/* Output final results. */
printf("Utilization = %f\n", total_busy_time/clock);
printf("Fraction served = %f\n", (double) total_served/total_arrived);
printf("Rejection Ratio = %f\n", (double) number_rejected/total_arrived);
printf("Mean number in system = %f\n", integral_of_n/clock);
printf("Mean delay = %f\n", (integral_of_n)/total_served);  //need
printf("Total served is: %d\n", total_served);
printf("Sum of total_served+number_rejected is %d\n", number_rejected);
printf("Total number that arrived = %d\n", total_arrived);
printf("integral_of_n = %f\n", integral_of_n);
printf("rejected_integral_of_n = %f\n\n", rejected_integral_of_n);
```

Data used to complete Question 7:

```
%Queue_size = 4, 40, 400
mean_delay_4 = [17.826515, 19.003150, 19.337471, 19.502382, 19.601942,
19.667937, 19.714915, 19.750441, 19.778412, 19.800396, 19.818600, 19.833662,
19.846294, 19.857352, 19.866749];
 mean_delay_40
=[197.818753,199.003153,199.337390,199.502311,199.601865,199.667891,199.71488
4,199.750412,199.778393,199.800364,199.818608,199.833649,199.846315,199.85738
4,199.866748];

mean_delay_400 = [1997.805985,
1998.993349,1999.328430,1999.493863,1999.593578,1999.659566,1999.706485,1999.
742437,1999.770501,1999.792797,1999.811178,1999.825943,1999.839082,1999.84916
6,1999.858648];

rejection_ratio_4 =
[0.607972,0.801987,0.867554,0.900501,0.920321,0.933557,0.943022,0.950126,0.95
5656,0.960081,0.963704,0.966723,0.969279,0.971470,0.973370];
rejection_ratio_40 =
[0.607856,0.801986,0.867554,0.900500,0.920321,0.933556,0.943021,0.950126,0.95
5655,0.960080,0.963703,0.966723,0.969278,0.971470,0.973369];
rejection_ratio_400 =
[0.607849,0.801979,0.867546,0.900493,0.920314,0.933549,0.943014,0.950118,0.95
5648,0.960073,0.963696,0.966715,0.969271,0.971462,0.973362];


arrival_rate =
[0.510000,1.010000,1.510000,2.010000,2.510000,3.010000,3.510000,4.010000,4.51
0000,5.010000,5.510000,6.010000,6.510000,7.010000,7.510000];
```