|  |
| --- |
| ECE 358 S20 |
| M/M/1 and M/M/1/K Queue Simulation |
| Lab 1 |

|  |
| --- |
| Arjun Bawa, 20711916  Mayuka Bulathsinghala, 20608116  6-1-2020 |

Contents

[Question 1 2](#_Toc42015354)

[Question 2 2](#_Toc42015355)

[Packet Generation 2](#_Toc42015356)

[expn\_random function 2](#_Toc42015357)

[question1 function 2](#_Toc42015358)

[gen\_ functions 2](#_Toc42015359)

[M/M/1 2](#_Toc42015360)

[simulateMM1 function 2](#_Toc42015361)

[question3 and question4 functions 2](#_Toc42015362)

[Question 3 2](#_Toc42015363)

[Question 4 2](#_Toc42015364)

[Question 5 2](#_Toc42015365)

[M/M/1/K 2](#_Toc42015366)

[simulateMM1K function 2](#_Toc42015367)

[Event class 2](#_Toc42015368)

[question 6 function 2](#_Toc42015369)

[Question 6 2](#_Toc42015370)

# Question 1

Our code generated the following experimental results for the mean and variance of 1000 exponential random variables

|  |  |
| --- | --- |
| Mean | Variance |
| 0.0137490037457974 | 0.000184033045328294 |

For an exponential random variable, the mean is . For , this is .

The percentage error between this and the experimental value is

For an exponential random variable, the variance is . For , this is .

The percentage error between this and the experimental value is

The percent errors for these two values is small enough that our exponential random variable generator code won’t negatively impact the remainder of the experiment.

# Question 2

Talk here about the structure of the implementation (helper functions, simulation function, output function, etc). Talk about the variables used, maybe show how the SIM\_TIME and other constants are defined and why they’re made global.

## Packet Generation

### expn\_random function



Figure 1 Helper function that generates an exponential random number

### question1 function



Figure 2 Function to encapsulate generation of random variables, computing statistics on them and saving the results

### gen\_ functions



Figure 3 Helper function to generate arrival events based on rate of arrival and simulation time. Stores events as list of dictionaries



Figure 4 Helper function to generate observer events based on rate of arrival and simulation time. Stores events as list of dictionaries



Figure 5 Function to generate departure events based on arrival events. Works in similar fashion to gen\_arrival and gen\_observer



Figure 6 Helper function to generate random service times for packets



Figure 7 Function to aggregate the generation of arrival, departure and observer events. Based on whether its used for the M/M/1 or M/M/1/K simulation, it will either include or exclude the generation of departure events

Explain a bit on how this approach helps to maintain code reusability and testability. Talk about certain design decisions like why dict and how the key=lambda function ties into that

## M/M/1

### simulateMM1 function



Figure 8 The simulation of the M/M/1 queue. Returns the computed statistics from the simulation

Talk about each variable declared in this function and how its used. What is happening in each iteration of the for loop? How is arrival rate, P\_idle and E[N] calculated? What exactly is the function returning?

### question3 and question4 functions



Figure 9 Function to encapsulate running the M/M/1 simulator with the range of queue utilization/traffic intensity values

Explain how the results[-1][t] part works (its due to how the data is returned by the simulation function)



Figure 10 Similar to the question3 function, this function runs the M/M/1 simulation with just one queue utilization value, 1.2

# Question 3

Figure 11 Graph showing the trend of average number of packets in queue (E[N]) with variation in traffic intensity/queue utilization. Shows results from simulation time T = 1000 (blue) and T = 2000 (red)

Figure 12 Graph showing the trend of probability of an idle server with variation in traffic intensity/queue utilization. Shows results from simulation time T = 1000 (blue) and T = 2000 (red)

# Question 4

# Question 5

## M/M/1/K

### simulateMM1K function

# Simulate M/M/1/K

*def* simulateMM1K(*q\_util*, *K*):

    pkt\_type\_count = {

        'arrival':0, # N\_a

        'departure':0, # N\_d

        'observation':0 # N\_o

    }

    idle\_count = 0

    q\_len\_observed\_over\_time = []

    current\_queue\_length = 0

    pkts\_lost\_count = 0

    prev\_d\_time = 0

    arrival\_rate = q\_util\*TRANS\_RATE/AVG\_PKT\_LEN

    event\_list = gen\_events(arrival\_rate, K)

    # converts events stores as dictionaries to Event

    #   objects for use with heapq

    event\_list = [Event(e['time'],e['type']) for e in event\_list]

    # an initial heapifying of the event list,

    #   maintaining the heap invariant: event time

    heapq.heapify(event\_list)

    while len(event\_list) > 0:

        pkt = heapq.heappop(event\_list)

        if pkt.type=='arrival':

            serv\_time = gen\_service\_time()

            if current\_queue\_length < K:

                d\_time = 0

                if current\_queue\_length > 0:

                    d\_time = prev\_d\_time + serv\_time

                else:

                    d\_time = pkt.time + serv\_time

                prev\_d\_time = d\_time

                heapq.heappush(event\_list,Event(d\_time,'departure'))

                pkt\_type\_count[pkt.type]+=1

                current\_queue\_length+=1

            else:

                pkts\_lost\_count+=1

        elif pkt.type=='departure':

            current\_queue\_length-=1

            pkt\_type\_count[pkt.type]+=1

        else:

            pkt\_type\_count[pkt.type]+=1

            # an observer event. observe q\_len and save that info

            q\_len\_observed\_over\_time.append(current\_queue\_length)

            # if q empty right now, its idle

            if current\_queue\_length==0:

                idle\_count+=1

    P\_idle = idle\_count/pkt\_type\_count['observation']

    TIME\_AVG\_PKTS\_IN\_Q

= sum(q\_len\_observed\_over\_time)/len(q\_len\_observed\_over\_time)

    # P\_loss := ratio of packets lost to total packets attempting to arrive

    P\_loss = pkts\_lost\_count/(pkt\_type\_count['arrival']+pkts\_lost\_count)

    return {TITLES\_K[0]:q\_util,

            TITLES\_K[1]:K,

            TITLES\_K[2]:pkt\_type\_count['arrival'],

            TITLES\_K[3]:pkt\_type\_count['departure'],

            TITLES\_K[4]:pkt\_type\_count['observation'],

            TITLES\_K[5]:P\_idle,

            TITLES\_K[6]:TIME\_AVG\_PKTS\_IN\_Q,

            TITLES\_K[7]:P\_loss}

Talk about why events are made into Event objects (and subsequently why heapq was used). Say what the heapify is for (its an extra measure, not necessary but only takes logn time once, so it doesn’t influence the total execution time much). Explain each step in the loop iteration. Explain how P\_loss computed and why.

### Event class



Figure 13 Wrapper class for representing an event. Used in the M/M/1/K simulator.

### question 6 function



Figure 14 Function to encapsulate running the M/M/1/K simulator with 3 different buffer sizes for each queue utilization/traffic intensity value

# Question 6

Figure 15 Graph showing the trend of average number of packets in queue (E[N]) with variation in traffic intensity/queue utilization. Shows results with buffer size K=10, 25, 50.

Figure 16 Graph showing the trend of probability of packet loss with variation in traffic intensity/queue utilization. Shows results with buffer size K=10, 25, 50.