EE5150: Communication Networks
Programming Assignment Report

Ishaan Agarwal
EE20B046

May 15, 2023

1 Introduction

The aim of these problems is to simulate some fairly known concepts taught in the course and make observations in concurrence with theory taught in the lectures.

2 The Geo/Geo/1 Queue

We aim to simulate a discrete time Geo/Geo/1 queue i.e a discrete time queue with Bernoulli arrivals and services. We start by defining the functions for arrivals and services in the following way:

```python
def arrival(lamda):
    #This is a Bernoulli random variable with success
   probability lambda
    return np.random.choice([0,1], p=[1-lamda, lamda])

def service(mu):
    #This is a Bernoulli random variable with success
   probability mu
    return np.random.choice([0,1], p=[1-mu, mu])
```

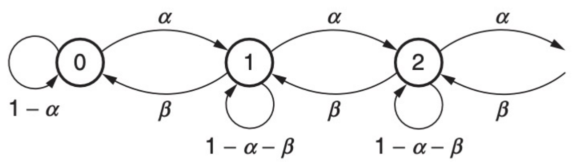


Figure 1: The Discrete time Geo/Geo/1 Queue

2.1 Simulating the Geo/Geo/1 Queue

We simulate the queue for $T = 10000$ time slots for $\lambda = 0.1, 0.2 \dots 1$ for a constant $\mu = 0.9$. This is done in the following code block:

```
mu = 0.9
T = 10000 #total time
#queue length = number of packets in the queue at any time
#sojourn time = time spent in queue + time spent in service
lamda_range = np.arange(0.1, 1.1, 0.1)
average_q = []
average_sojourn_time = []

for lamda in lamda_range:
    ql = 0 #queue length
    q = [] #queue length at each time instant just before the
    arrival
    q.append(0)
    arrival_times = []
    service_times = []
    sojourn_times = []
    t = 1

    while t < T:
        if arrival(lamda):
            ql += 1
            arrival_times.append(t)
        if service(mu):
            if(ql!=0):
                ql-=1
                service_times.append(t)
        q.append(ql)
        t += 1

    average_q.append(np.mean(q))
    for i in range(len(service_times)): #computing only for
    those who have been served
        sojourn_times.append(service_times[i] - arrival_times[i
    ])
    average_sojourn_time.append(np.mean(sojourn_times))
```

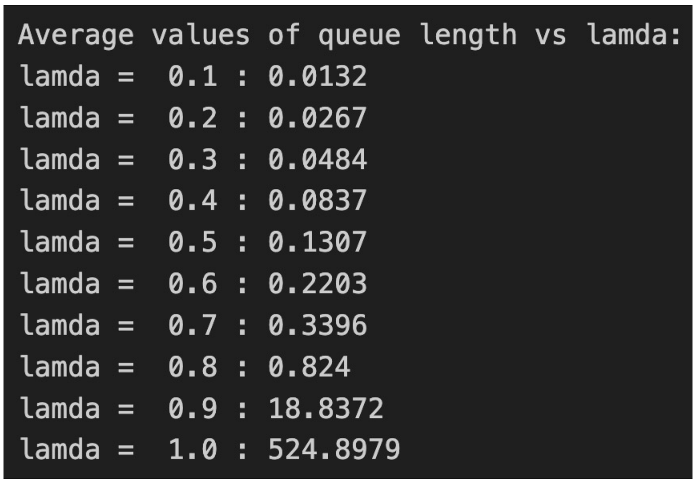
2.2 Average Queue Lengths

Now, we attempt to observe the average queue lengths versus the arrival rate λ. The corresponding code is:

```

print("Average values of queue length vs lamda:")
for i in range(1, len(average_q)+1):
    print("lamda = ", i/10, ":", average_q[i-1])

```

```python
plt.plot(lamda_range, average_q, 'ro-')
plt.xlabel('lamda')
plt.ylabel('average queue length')
plt.grid()
plt.show()
```

The obtained output is as follows:



```
Average values of queue length vs lamda:
lamda =  0.1 : 0.0132
lamda =  0.2 : 0.0267
lamda =  0.3 : 0.0484
lamda =  0.4 : 0.0837
lamda =  0.5 : 0.1307
lamda =  0.6 : 0.2203
lamda =  0.7 : 0.3396
lamda =  0.8 : 0.824
lamda =  0.9 : 18.8372
lamda =  1.0 : 524.8979
```

Figure 2: Output

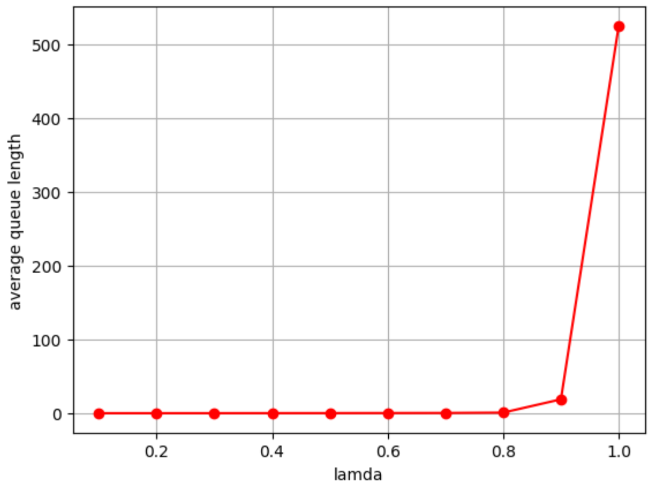


Figure 3: Average queue lengths vs λ

We see that the average values of queue lengths are very small for $\lambda < 0.9$, this is because in these cases the arrivals are slower than services and thus, the queue length does not blow up, whereas in the other two cases, $\lambda \geq \mu$, and thus the queue length is significantly high.

2.3 Simulations vs Theoretical

In the previous part, we have observed the simulated queue length vs λ. In this part, let us compare that with the theoretical average queue lengths, which is computed by taking the expectation under a stationary distribution for the same set of parameters.
We know that for a finite state Geo/Geo/1 queue, the stationary distribution is given by the following (for max n customers in the system)(since infinite is not realisable on a machine, we take very large n value): Local balance equations:

$$\pi(i)\lambda(1-\mu) = \pi(i+1)\mu(1-\lambda) \ \ \forall i = 0, 1, 2, \ldots, n-1$$

$$\implies \pi(i+1) = \pi(i)\rho \ \ \forall i = 0, 1, 2, ..., n-1$$

$$\implies Pi(i) = \frac{(1-p)\rho^i}{(1-p^{(n+1)}})\ \ \forall i = 0, 1, 2, ..., n$$

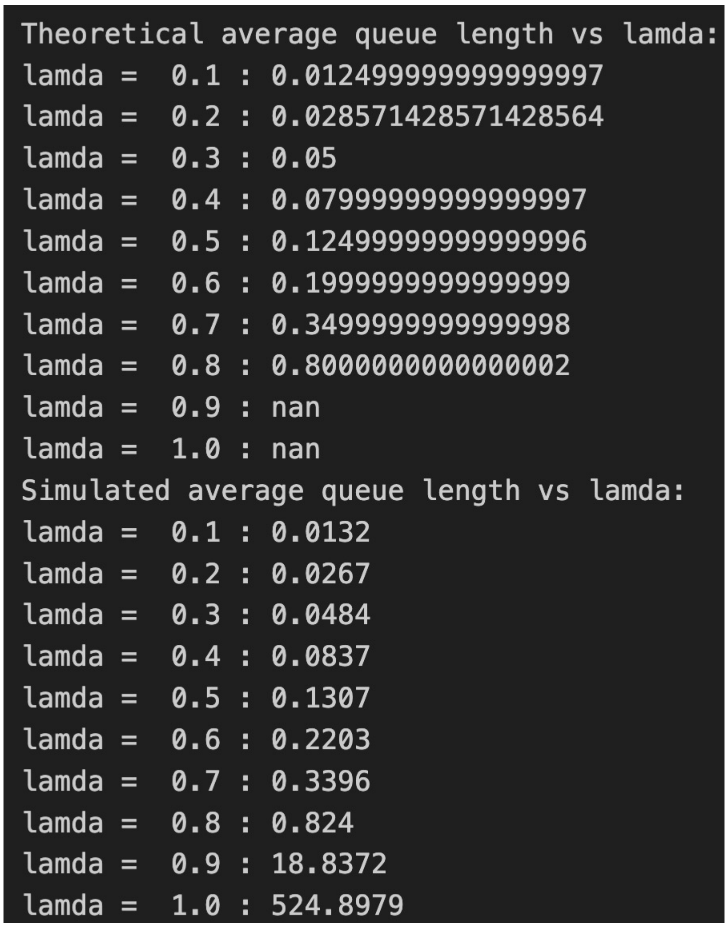
$$\rho = \frac{\lambda(1-\mu)}{\mu(1-\lambda)}$$

The codeblock for the following part is:

```python

#part (b):
n = 1000 #max queue length
mu = 0.9

theoretical_average_q = []

for lamda in lamda_range:
    p = lamda*(1-mu)/(mu*(1-lamda))
    Pi = np.zeros(n+1)
    #set Pi[0] = (1-p)(1-p**(n+1))
    Pi[0] = (1-p)/(1-p**(n+1))
    for i in range(1, n+1):
        Pi[i] = p**i*Pi[0]
    #Finding the expectation under the steady state
   distribution
    E = 0
    for i in range(n+1):
        E += i*Pi[i]
    theoretical_average_q.append(E)

print("Theoretical average queue length vs lamda:")
for i in range(1, len(theoretical_average_q)+1):
    print("lamda = ", i/10, ":", theoretical_average_q[i-1])

print("Simulated average queue length vs lamda:")
for i in range(1, len(average_q)+1):
    print("lamda = ", i/10, ":", average_q[i-1])

```

```python
#plot theoretical average and simulated average on the same
    graph
plt.plot(lamda_range, theoretical_average_q, 'ro-', label='
    theoretical')
plt.plot(lamda_range, average_q, 'b--', label='simulated')
plt.xlabel('lamda')
plt.ylabel('average queue length')
plt.legend()
plt.grid()
plt.show()
```

The obtained output is:



```
Theoretical average queue length vs lamda:
lamda =  0.1 : 0.012499999999999997
lamda =  0.2 : 0.028571428571428564
lamda =  0.3 : 0.05
lamda =  0.4 : 0.07999999999999997
lamda =  0.5 : 0.12499999999999996
lamda =  0.6 : 0.1999999999999999
lamda =  0.7 : 0.3499999999999998
lamda =  0.8 : 0.8000000000000002
lamda =  0.9 : nan
lamda =  1.0 : nan
Simulated average queue length vs lamda:
lamda =  0.1 : 0.0132
lamda =  0.2 : 0.0267
lamda =  0.3 : 0.0484
lamda =  0.4 : 0.0837
lamda =  0.5 : 0.1307
lamda =  0.6 : 0.2203
lamda =  0.7 : 0.3396
lamda =  0.8 : 0.824
lamda =  0.9 : 18.8372
lamda =  1.0 : 524.8979
```

Figure 4: Output

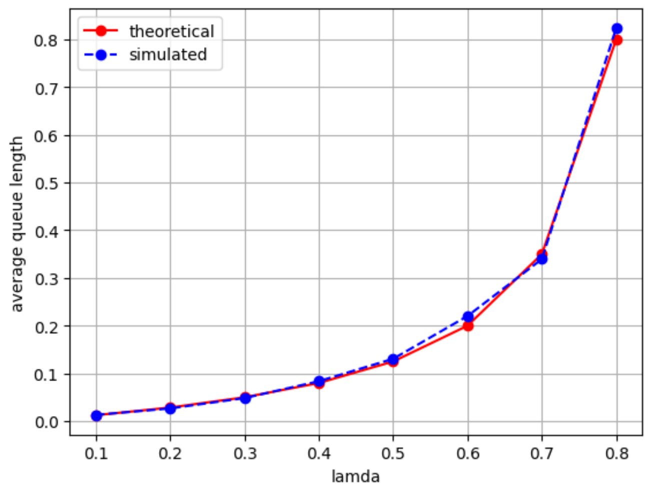


Figure 5: Theoretical average queue lengths and Simulated Average Queue lengths vs λ

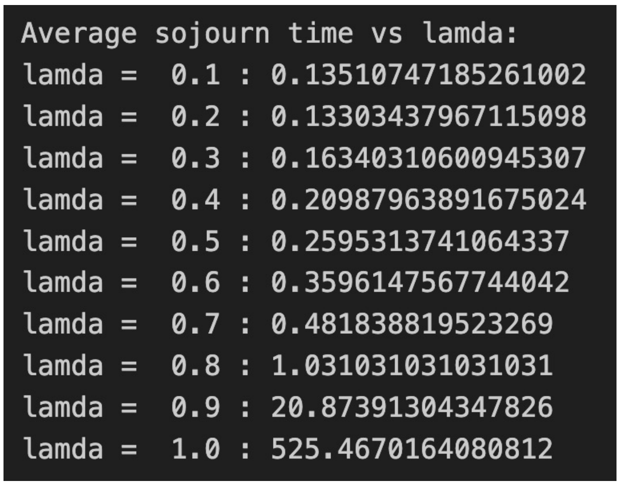
Note that the theoretical averages are not defined for $\lambda \geq 0.9$ because the sum of the geometric series only converges if $\lambda < \mu$, which is also our necessary condition for positive recurrence of the discrete time Markov chain corresponding to this process.

2.4 Sojourn times

Sojourn time is defined as the time from arrival till service completion of a packet, we want to observe the average sojourn times for each λ. This was done in the code block provided above by subtracting the arrival time from the service time for each packet and then taking the average.

```
print("Average sojourn time vs lamda:")
for i in range(1, len(average_sojourn_time)+1):
    print("lamda = ", i/10, ":", average_sojourn_time[i-1])
plt.plot(lamda_range, average_sojourn_time, 'ro-')
plt.xlabel('lamda')
plt.ylabel('average sojourn time')
plt.grid()
plt.show()
```

The obtained output is as follows:



```
Average sojourn time vs lamda:
lamda =  0.1 : 0.13510747185261002
lamda =  0.2 : 0.13303437967115098
lamda =  0.3 : 0.16340310600945307
lamda =  0.4 : 0.20987963891675024
lamda =  0.5 : 0.2595313741064337
lamda =  0.6 : 0.3596147567744042
lamda =  0.7 : 0.481838819523269
lamda =  0.8 : 1.031031031031031
lamda =  0.9 : 20.87391304347826
lamda =  1.0 : 525.4670164080812
```

Figure 6: Output

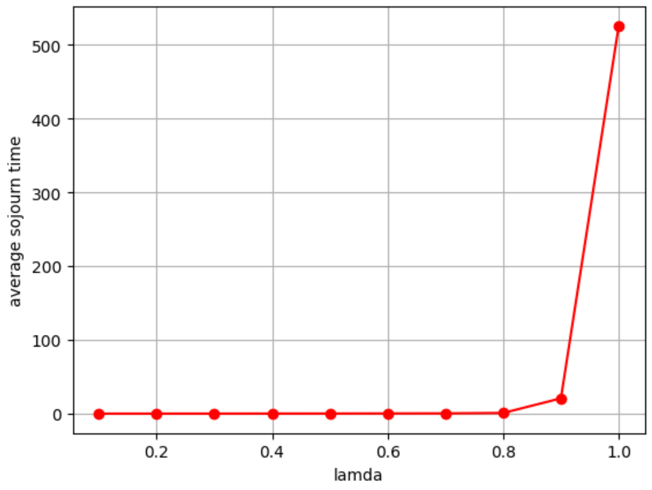


Figure 7: Average sojourn times vs λ

2.5 Little's Law

Little's Law states that the average queue length is equal to the product of the mean arrival rate and the mean sojourn times. Let us verify this law from our simulations.

```
#Part c: Little's Theorem
#plotting the ration of average queue length and average
    sojourn time vs lamda
ratio = np.divide(average_q, average_sojourn_time)
print("Ratio of average queue length and average sojourn time
    vs lamda:")
```

```python
for i in range(1, len(ratio)+1):
    print("lamda = ", i/10, ":", ratio[i-1])
plt.plot(lamda_range, ratio, 'ro--')
plt.xlabel('lamda')
plt.ylabel('ratio of average queue length and average sojourn
    time')
plt.grid()
plt.show()
```

Now, to verify that this ratio is indeed equal to the mean arrival rate, let us plot this ratio against λ, and then fit a least squares line to this using `np.linalg.lstsq` function.

```python
#Let us verify this by fitting a line to the plot
#We want to fit a least squares line to the data
#Fit a line y = mx + c to the data using numpy.linalg.lstsq
m, c = np.linalg.lstsq(np.vstack([lamda_range, np.ones(len(
    lamda_range))]).T, ratio, rcond = None)[0]
print("m = ", m, "c = ", c)
plt.plot(lamda_range, ratio, 'ro-', label='Original data',
    markersize=10)
plt.plot(lamda_range, m*lamda_range + c, 'b--', label='Fitted
    line')
plt.legend()
plt.grid()
plt.show()
#calculating mean squared error in percentage
mse = np.mean(np.square(np.subtract(ratio, m*lamda_range + c)))
print("Mean squared error in percentage = ", mse*100)
```

The graph obtained is as follows, with a slope of $m = 1.00394$ and an intercept of $c = -0.00067$, with a mean squared error of 0.001% which shows that the obtained line is very close to the line $y = x$.

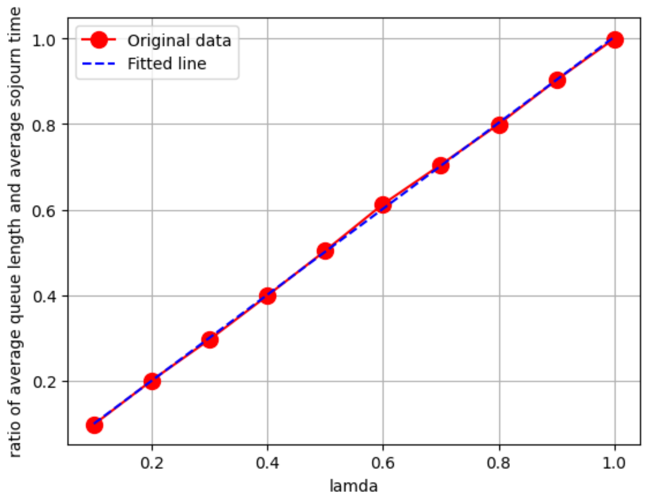


Figure 8: Little's Law

3 Conclusion

We thus have simulated the discrete time Geo/Geo/1 queue and have verified the following results.

- Average queue lengths increase with arrival rate.
- Average sojourn time increase with arrival rate.
- The simulated average queue length was found to be very close to the theoretical average queue length.
- Little's law was simulated and verified.

The code can be found here: Code