Programming Assignments

Harisankar K J
EE20B043

Programming Assignment-1

Simulate a discrete-time Geo/Geo/1 queue, i.e., a discrete time queue with Bernoulli arrivals and services. Play with it by changing different parameters. In particular, study the following scenarios.

a. Pick μ = 0.9. Simulate for λ = 0.1, 0.2, . . . 0.9, 1. By simulating the queue for T time slots (large T ≫ 1000), compute average queue length in each case, where Q(t) is the number of jobs in the system. Plot against λ.
b. You learned in class about the πP = π. Compare the above quantity with the expectation under the PMF π
c. Let Di be the sojourn time of the ith job/packet. (Sojourn time is the time from arrival till service completion.) Compute average sojourn time for large N for all λ in the above list. Plot against λ.
d. Plot the ratio of the two quantities computed in the previous parts against λ. (This relates to an interesting theorem in queuing theory called Little's theorem. Check that out in the textbook.

Solution:

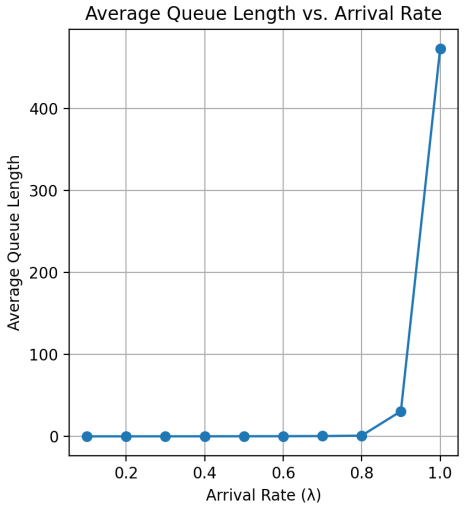
1. The code simulates a queueing system and analyzes various performance metrics, such as average queue length and average sojourn time, for different arrival rates. It uses the NumPy and Matplotlib libraries for numerical computations and data visualization

a.

```
def simulate_queue(lambda_val, mu_val, T):
    # initializing variables
    queue_length = np.zeros(T)
    queue = 0
    sojourn_times = []
    arrival_times = []
    service_times = []

    for t in range(T):
        # Arrival process
        if np.random.rand() < lambda_val:
            queue += 1
            arrival_times.append(t)

        # Service process
        if np.random.rand() < mu_val and queue > 0:
            queue -= 1
            service_times.append(t)

        queue_length[t] = queue
```

The simulate_queue function takes three parameters: **lambda_val** (arrival rate), **mu_val** (service rate=0.9), and **T** (simulation time). It initializes

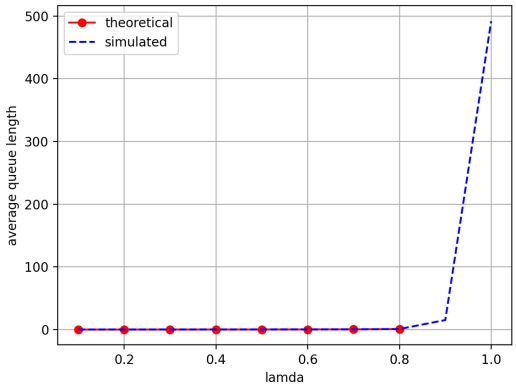
variables to keep track of the queue length, queue size, sojourn times of customers, arrival times, and service times.
The simulation runs for T time steps. At each time step, the arrival process is simulated by checking if a randomly generated number is less than the arrival rate (lambda_val). If it is, a customer arrives and the queue size is incremented, and the arrival time is recorded.
Next, the simulation of service time takes place and the service time and the arrival time is noted down.
The average queue length is noted down for each value of arrival rate varying from 0.1 to 1 and a fixed simulation time 10000 and is plotted.



against the arrival rate. The plot shows that as the arrival rate increases and passes the service rate, the average queue length increases drastically which may not lead to an efficient system. Thus an arrival rate slightly lesser or equal to the service rate is preferred for efficient functioning of the queuing system.

b. Here, we compare the expectation under a stationary distribution with the average queue length. We are considering a finite state geo/geo/1 system (for a maximum of n customers, n=1000 is taken).
We use the local balance equations P[i]*λ=P[i+1]*μ where ‘i’ varies from 0 to n-1, to find out the theoretical values of avg queue length. Where Pi(i) = (1-p)p^i/(1-p^(n+1)) and p=λ/μ. We find expectation under steady state distribution.
The values found are plotted against arrival rates.

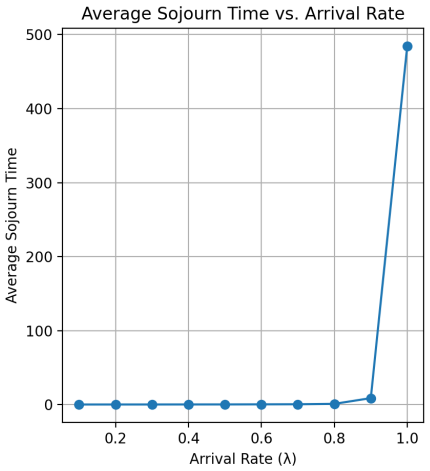
```
n = 1000
theoretical_average_q = []

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

# plotting the values
plt.plot(lambda_values, theoretical_average_q, 'ro-', label='theoretical')
plt.plot(lambda_values, average_lengths, 'b--', label='simulated')
plt.xlabel('lamda')
plt.ylabel('average queue length')
plt.legend()
plt.grid()
plt.show()
```



The theoretical value of the average length that we got from the expectation stays consistent with the simulated values.

c. Now we look at the calculation of average sojourn time.The service process here is simulated by checking if a randomly generated number is less than the service rate (mu_val) and if the queue is not empty (queue > 0). If both conditions are met, a customer is served, the queue size is decremented, and the service time is recorded.
Here the arrival time is subtracted from the service time to get the sojourn time for each arrival time and is averaged to get the average sojourn time. The graph is plotted.

```python
lambda_values = np.arange(0.1, 1.1, 0.1)  # ranges of values of arrival time
mu = 0.9
T = 10000

# Lists to store results
average_lengths = []
average_sojourn_times = []

# Simulate the queue for different arrival rates
for lambda_val in lambda_values:
    average_length, average_sojourn_time = simulate_queue(lambda_val, mu, T)
    average_lengths.append(average_length)
    average_sojourn_times.append(average_sojourn_time)
```




```python
# Plot average queue length against arrival rate
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.plot(lambda_values, average_lengths, marker='o')
plt.xlabel('Arrival Rate (λ)')
plt.ylabel('Average Queue Length')
plt.title('Average Queue Length vs. Arrival Rate')
plt.grid(True)

# Plot average sojourn time against arrival rate
plt.subplot(1, 2, 2)
plt.plot(lambda_values, average_sojourn_times, marker='o')
plt.xlabel('Arrival Rate (λ)')
plt.ylabel('Average Sojourn Time')
plt.title('Average Sojourn Time vs. Arrival Rate')
plt.grid(True)
```

Here also we notice that as the arrival rate is increased beyond the service rate, the waiting time also increases drastically.

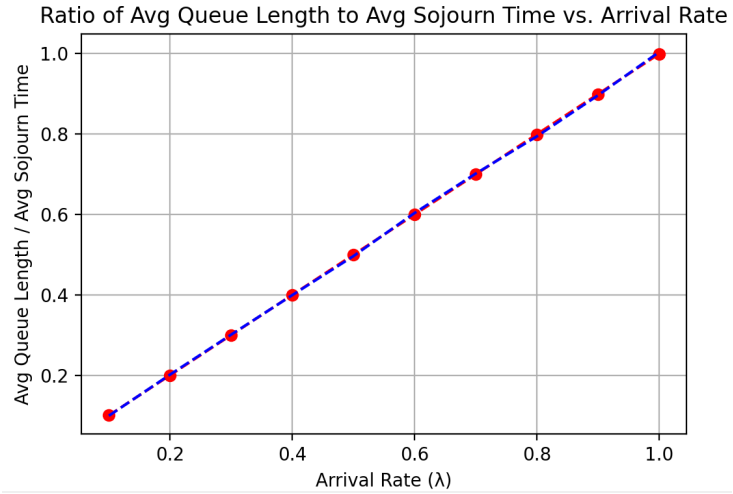
d. Now we find the simulated ratio of average queue length and average sojourn time for each value of arrival rate for a fixed simulation time and service rate(mu=0.9). This ratio lets us know how efficiently the system operates.

```python
# Calculate the ratio of average queue length to average sojourn time
ratio_lengths_sojourn_times = np.divide(average_lengths, average_sojourn_times)

# Plot the ratio against arrival rate simulated and theoretical compared
plt.figure(figsize=(6, 4))
plt.xlabel('Arrival Rate (λ)')
plt.ylabel('Avg Queue Length / Avg Sojourn Time')
plt.title('Ratio of Avg Queue Length to Avg Sojourn Time vs. Arrival Rate')
plt.grid(True)
```

We also find out the theoretical ratio of average queue length and average sojourn and plot it against arrival rate. This shows us an example of the Little's Theorem which states that the average queue length = average arrival rate * average sojourn time. In order for the system to be stable the arrival rate must be less than the service rate.

```
80  # Perform linear regression to fit the y=mx+c
81  m, c = np.polyfit(lambda_values, ratio_lengths_sojourn_times, deg=1)
82
83  # Generate the fitted line using the obtained coeffratio_lengths_sojourn_timesicients
84  fitted_line = m * lambda_values + c
85
86  # Plot the original data and the fitted line
87  plt.plot(lambda_values, fitted_line, 'ro--', label="Theoretical")
88  plt.plot(lambda_values, ratio_lengths_sojourn_times, 'b--', label='simulated')
89  plt.show()
```

Here we get the theoretical values by fitting the least squares line (y=mx+c) to the data.



Now we notice that the theoretical values(in red) match very closely with the simulated values(in blue). This validates Little's Theorem.