**Batch: A1 Roll No.: 16010121045**

**Experiment / Assignment / Tutorial No 5**

|  |
| --- |
| **Title: M/M/1 and M/G/1 Queuing Model Simulation** |

**Objective:** The objective of this lab experiment is to perform an analysis of the M/M/1 and M/G/1 queue model by considering different varying parameters and their impact on key performance metrics. The experiment includes the calculation of theoretical values, simulation of the queue, and statistical analysis of the results.

**Expected Outcome of Experiment:**

CO2: Analyse and apply general principles of event scheduling algorithm & various statistical methods on different applications.

**Books/ Journals/ Websites referred:**

1. “Discrete-Event System Simulation” by Jerry Banks, John S. Carson II, Barry L. Nelson, David M. Nicol.
2. SimPy Documentation: https://simpy.readthedocs.io/en/latest/
3. SciPy Documentation: https://docs.scipy.org/doc/scipy/

**Background:**

**M/M/1 Queue**:

* **Description**: An M/M/1 queue is a single-server queuing model where both the arrival and service times follow an **exponential distribution** (Markovian process), and there is one server.
* **Key Assumptions**:
	+ **M**: Memoryless (exponential inter-arrival times).
	+ **M**: Memoryless (exponential service times).
	+ **1**: One server.
* **Key Performance Metrics**:
	+ **Arrival rate (λ)**: Average rate at which customers arrive.
	+ **Service rate (μ)**: Average rate at which the server serves customers.
	+ **Utilization (ρ)**: ρ=λμ\rho = \frac{\lambda}{\mu}ρ=μλ​, fraction of time the server is busy.
	+ **Average number of customers in the system (L)**: L=λμ−λL = \frac{\lambda}{\mu - \lambda}L=μ−λλ​.
	+ **Average number of customers in the queue (Lq)**: Lq=λ2μ(μ−λ)Lq = \frac{\lambda^2}{\mu(\mu - \lambda)}Lq=μ(μ−λ)λ2​.
	+ **Average time in the system (W)**: W=1μ−λW = \frac{1}{\mu - \lambda}W=μ−λ1​.
	+ **Average time in the queue (Wq)**: Wq=λμ(μ−λ)Wq = \frac{\lambda}{\mu(\mu - \lambda)}Wq=μ(μ−λ)λ​.

**M/G/1 Queue**:

* **Description**: An M/G/1 queue is similar to M/M/1 but with **general distribution** for service times (G), allowing for arbitrary service time distributions, while arrivals remain exponential.
* **Key Assumptions**:
	+ **M**: Exponential inter-arrival times.
	+ **G**: General distribution for service times.
	+ **1**: One server.
* **Key Performance Metrics**:
	+ **Arrival rate (λ)** and **Utilization (ρ)**: Same as M/M/1.
	+ **Average number of customers in the system (L)**: L=ρ+λVar(S)2(1−ρ)L = \rho + \frac{\lambda \text{Var}(S)}{2(1 - \rho)}L=ρ+2(1−ρ)λVar(S)​, where Var(S)\text{Var}(S)Var(S) is the variance of service time.
	+ **Average time in the system (W)**: W=1μ+λVar(S)2(1−ρ)W = \frac{1}{\mu} + \frac{\lambda \text{Var}(S)}{2(1 - \rho)}W=μ1​+2(1−ρ)λVar(S)​.
	+ Other metrics are more complex due to the general service time distribution.

**Problem Statement 1:**

Perform analysis of the M/M/1 queue model by considering different levels of traffic intensity and their impact on key performance metrics.

Consider the following:

**Traffic Intensity (ρ) Levels:**

Low Traffic Intensity: ρ=0.5

Moderate Traffic Intensity: ρ=0.75

High Traffic Intensity: ρ=0.95

**Parameters:**

Arrival rate (𝜆): Adjusted based on the chosen 𝜌

Service rate (𝜇): Fixed at 1 customer per minute

Simulation time: 100,000 minutes

**Key Performance Metrics**

Average Waiting Time (Wq)

Average Number of Customers in the System (L)

Server Utilization (𝜌)

Queue Length Distribution

Waiting Time Distribution

**Problem Statement 2:**

* Simulate an M/G/1 queue to model peak-hour traffic at a toll booth, where service times vary throughout the day, and analyze the toll booth’s performance metrics.
* Vehicles arrive following a Poisson process with an average arrival rate.
* Service times vary according to a normal distribution with different means during peak and off-peak hours.
* Analyze the average waiting time, queue length, and server utilization during peak and off-peak hours.

**Key Performance Metrics:**

Average Waiting Time (Wq)

Average Number of Customers in the System (L)

Server Utilization (𝜌)

Consider the following:

1. **Arrival Rate** (𝜆): Average 5 vehicles per minute.
2. **Service Time Distribution**:
* Peak Hours: Mean = 0.5 minutes, Standard Deviation = 0.1 minutes.
* Off-Peak Hours: Mean = 1.0 minutes, Standard Deviation = 0.2 minutes.
1. **Peak Hours Duration**: 7:00 AM - 9:00 AM.
2. **Off-Peak Hours Duration**: 9:00 AM - 5:00 PM.
3. **Simulation Time**: 10 hours (7:00 AM - 5:00 PM).

**Simulation:**

Implement the simulation using Python and the SimPy library to model the queue behavior. The key steps include defining the customer arrival process, handling the customer service process, and collecting statistics on waiting times and queue lengths.

**Implementation Steps with Screen shots:**

**M/M/1**

*import* simpy

*import* random

*import* numpy *as* np

*import* matplotlib.pyplot *as* plt

*# Simulation parameters*

SIM\_TIME = 100000 *# Total simulation time in minutes*

SERVICE\_RATE = 1 *# Service rate (μ) is fixed at 1 customer per minute*

*# Traffic intensity levels*

TRAFFIC\_INTENSITIES = {

 'Low': 0.5,

 'Moderate': 0.75,

 'High': 0.9

}

class MM1Queue:

 def \_\_init\_\_(*self*, *env*, *service\_rate*, *traffic\_intensity*):

 *self*.env = *env*

 *self*.server = simpy.Resource(*env*, *capacity*=1)

 *self*.service\_rate = *service\_rate*

 *self*.arrival\_rate = *traffic\_intensity* \* *service\_rate*

 *self*.wait\_times = []

 *self*.queue\_lengths = []

 *self*.customers\_in\_system = []

 *self*.server\_utilization\_time = 0

 *self*.customer\_count = 0

 def process\_customer(*self*, *customer\_id*):

 arrival\_time = *self*.env.now

 *with* *self*.server.request() *as* request:

 *yield* request

 wait\_time = *self*.env.now - arrival\_time

 *self*.wait\_times.append(wait\_time)

 *# Service the customer*

 service\_time = random.expovariate(*self*.service\_rate)

 *yield* *self*.env.timeout(service\_time)

 *self*.server\_utilization\_time += service\_time

 *self*.customers\_in\_system.append(*self*.server.count)

 *self*.customer\_count += 1

 def customer\_arrivals(*self*):

 *while* True:

 inter\_arrival\_time = random.expovariate(*self*.arrival\_rate)

 *yield* *self*.env.timeout(inter\_arrival\_time)

 *self*.env.process(*self*.process\_customer(*self*.customer\_count))

 *self*.queue\_lengths.append(len(*self*.server.queue))

*# Simulation function*

def run\_simulation(*traffic\_intensity\_label*, *traffic\_intensity*):

 env = simpy.Environment()

 mm1\_queue = MM1Queue(env, SERVICE\_RATE, *traffic\_intensity*)

 env.process(mm1\_queue.customer\_arrivals())

 env.run(*until*=SIM\_TIME)

 *# Performance metrics calculations*

 avg\_waiting\_time = np.mean(mm1\_queue.wait\_times)

 avg\_customers\_in\_system = np.mean(mm1\_queue.customers\_in\_system)

 avg\_queue\_length = np.mean(mm1\_queue.queue\_lengths)

 utilization = mm1\_queue.server\_utilization\_time / SIM\_TIME

 *return* avg\_waiting\_time, avg\_customers\_in\_system, utilization, mm1\_queue.wait\_times, mm1\_queue.queue\_lengths

*# Arrays to store metrics for plotting*

traffic\_intensity\_vals = []

waiting\_times = []

customers\_in\_system\_vals = []

utilizations = []

queue\_lengths = []

wait\_time\_distributions = []

queue\_length\_distributions = []

*# Run simulations for different traffic intensities and store metrics*

*for* label, intensity *in* TRAFFIC\_INTENSITIES.items():

 avg\_waiting\_time, avg\_customers\_in\_system, utilization, wait\_times, queue\_lengths\_data = run\_simulation(label, intensity)

 traffic\_intensity\_vals.append(intensity)

 waiting\_times.append(avg\_waiting\_time)

 customers\_in\_system\_vals.append(avg\_customers\_in\_system)

 utilizations.append(utilization)

 wait\_time\_distributions.append(wait\_times)

 queue\_length\_distributions.append(queue\_lengths\_data)

*# Create subplots for all the performance metrics*

fig, axs = plt.subplots(3, 2, *figsize*=(12, 12)) *# Create a 3x2 grid of subplots*

*# Plot Average Waiting Time (Wq) vs Traffic Intensity (ρ)*

axs[0, 0].plot(traffic\_intensity\_vals, waiting\_times, *marker*='o', *label*="Avg Waiting Time (Wq)")

axs[0, 0].set\_xlabel('Traffic Intensity (ρ)')

axs[0, 0].set\_ylabel('Avg Waiting Time (minutes)')

axs[0, 0].set\_title('Avg Waiting Time (Wq) vs Traffic Intensity (ρ)')

*# Plot Average Number of Customers in System (L) vs Traffic Intensity (ρ)*

axs[0, 1].plot(traffic\_intensity\_vals, customers\_in\_system\_vals, *marker*='o', *label*="Avg Customers in System (L)")

axs[0, 1].set\_xlabel('Traffic Intensity (ρ)')

axs[0, 1].set\_ylabel('Avg Customers in System')

axs[0, 1].set\_title('Avg Customers in System (L) vs Traffic Intensity (ρ)')

*# Plot Server Utilization (ρ) vs Traffic Intensity (ρ)*

axs[1, 0].plot(traffic\_intensity\_vals, utilizations, *marker*='o', *label*="Server Utilization")

axs[1, 0].set\_xlabel('Traffic Intensity (ρ)')

axs[1, 0].set\_ylabel('Server Utilization')

axs[1, 0].set\_title('Server Utilization vs Traffic Intensity (ρ)')

*# Plot Queue Length Distribution for different traffic intensities*

*for* i, label *in* enumerate(TRAFFIC\_INTENSITIES.keys()):

 axs[1, 1].hist(queue\_length\_distributions[i], *bins*=20, *alpha*=0.7, *label*=f"Queue Length ({label})")

axs[1, 1].set\_xlabel('Queue Length')

axs[1, 1].set\_ylabel('Frequency')

axs[1, 1].set\_title('Queue Length Distribution')

axs[1, 1].legend()

*# Plot Waiting Time Distribution for different traffic intensities*

*for* i, label *in* enumerate(TRAFFIC\_INTENSITIES.keys()):

 axs[2, 0].hist(wait\_time\_distributions[i], *bins*=20, *alpha*=0.7, *label*=f"Waiting Time ({label})")

axs[2, 0].set\_xlabel('Waiting Time (minutes)')

axs[2, 0].set\_ylabel('Frequency')

axs[2, 0].set\_title('Waiting Time Distribution')

axs[2, 0].legend()

*# Remove the last empty plot (bottom right)*

fig.delaxes(axs[2, 1])

*# Adjust layout for better readability*

plt.tight\_layout()

*# Show the plot*

plt.show()

****

**M/G/1**

*import* simpy

*import* random

*import* numpy *as* np

*# Simulation parameters*

SIM\_TIME = 100000 *# Total simulation time in minutes*

PEAK\_HOURS = [50000, 60000] *# Define peak hour time range (minutes)*

ARRIVAL\_RATE\_PEAK = 0.8 *# Average arrival rate during peak hours (vehicles per minute)*

ARRIVAL\_RATE\_OFF\_PEAK = 0.3 *# Average arrival rate during off-peak hours (vehicles per minute)*

SERVICE\_MEAN\_PEAK = 0.7 *# Average service time during peak hours (minutes)*

SERVICE\_STD\_PEAK = 0.2 *# Standard deviation for service time during peak hours*

SERVICE\_MEAN\_OFF\_PEAK = 0.5 *# Average service time during off-peak hours (minutes)*

SERVICE\_STD\_OFF\_PEAK = 0.1 *# Standard deviation for service time during off-peak hours*

class MG1Queue:

 def \_\_init\_\_(*self*, *env*):

 *self*.env = *env*

 *self*.server = simpy.Resource(*env*, *capacity*=1)

 *self*.wait\_times = []

 *self*.queue\_lengths = []

 *self*.server\_utilization\_time = 0

 *self*.customer\_count = 0

 def process\_vehicle(*self*, *arrival\_time*, *service\_time*):

 *with* *self*.server.request() *as* request:

 *yield* request

 *# Calculate waiting time in the queue*

 wait\_time = *self*.env.now - *arrival\_time*

 *self*.wait\_times.append(wait\_time)

 *# Service the vehicle (Normal distribution for service times)*

 *yield* *self*.env.timeout(*service\_time*)

 *self*.server\_utilization\_time += *service\_time*

 *self*.customer\_count += 1

 *self*.queue\_lengths.append(len(*self*.server.queue))

 def vehicle\_arrivals(*self*):

 *while* True:

 current\_time = *self*.env.now

 *if* PEAK\_HOURS[0] <= current\_time <= PEAK\_HOURS[1]:

 inter\_arrival\_time = random.expovariate(ARRIVAL\_RATE\_PEAK)

 service\_time = max(0, random.gauss(SERVICE\_MEAN\_PEAK, SERVICE\_STD\_PEAK))

 *else*:

 inter\_arrival\_time = random.expovariate(ARRIVAL\_RATE\_OFF\_PEAK)

 service\_time = max(0, random.gauss(SERVICE\_MEAN\_OFF\_PEAK, SERVICE\_STD\_OFF\_PEAK))

 *yield* *self*.env.timeout(inter\_arrival\_time)

 *self*.env.process(*self*.process\_vehicle(*self*.env.now, service\_time))

*# Simulation function*

def run\_simulation():

 env = simpy.Environment()

 mg1\_queue = MG1Queue(env)

 env.process(mg1\_queue.vehicle\_arrivals())

 env.run(*until*=SIM\_TIME)

 *# Performance metrics calculations*

 avg\_waiting\_time = np.mean(mg1\_queue.wait\_times)

 avg\_queue\_length = np.mean(mg1\_queue.queue\_lengths)

 utilization = mg1\_queue.server\_utilization\_time / SIM\_TIME

 print(f"Results for Peak and Off-Peak Hours:")

 print(f"Average Waiting Time (Wq): {avg\_waiting\_time:.2f} minutes")

 print(f"Average Queue Length: {avg\_queue\_length:.2f} vehicles")

 print(f"Server Utilization (ρ): {utilization:.2f}")

*# Run the simulation*

run\_simulation()

****

**Conclusion:**

In this experiment, we performed an in-depth analysis of the **M/M/1** and **M/G/1** queue models by varying key parameters such as arrival rate (λ), service rate (μ), and service time distribution. The theoretical values calculated for metrics like utilization, average queue length, and average time in the system were compared with the results obtained through simulation.

**Post lab Questions:**

**Network of Queues (M/M/1 and M/M/3)**

Simulate a network of interconnected queues (M/M/1 and M/M/3) where customers pass through multiple service stations with different service rates.

**Consider the following Scenario**:

* Customers first arrive at a check-in counter (M/M/1).
* After check-in, customers move to a service desk (M/M/1).
* After the service desk, customers may visit one of several specialized service counters (M/M/3).
* Each queue has different arrival and service rates.
* Analyze the overall system performance, including average waiting time, queue length, and server utilization at each station.

Ans)

*import* simpy

*import* random

*import* numpy *as* np

*# Simulation parameters*

SIM\_TIME = 50000 *# Total simulation time in minutes*

*# Arrival and service rates for each station*

ARRIVAL\_RATE\_CHECKIN = 1/2 *# Customers arrive every 2 minutes on average*

SERVICE\_RATE\_CHECKIN = 1/3 *# Check-in takes on average 3 minutes (M/M/1)*

SERVICE\_RATE\_SERVICE\_DESK = 1/4 *# Service desk takes on average 4 minutes (M/M/1)*

SERVICE\_RATE\_SPECIALIZED = 1/5 *# Specialized service takes on average 5 minutes (M/M/3)*

SPECIALIZED\_SERVERS = 3 *# M/M/3 configuration*

class QueueSystem:

 def \_\_init\_\_(*self*, *env*, *servers*, *service\_rate*, *label*):

 *self*.env = *env*

 *self*.server = simpy.Resource(*env*, *capacity*=*servers*)

 *self*.service\_rate = *service\_rate*

 *self*.label = *label*

 *self*.wait\_times = []

 *self*.queue\_lengths = []

 *self*.server\_utilization\_time = 0

 *self*.customer\_count = 0

 def process\_customer(*self*, *arrival\_time*):

 *with* *self*.server.request() *as* request:

 *yield* request

 wait\_time = *self*.env.now - *arrival\_time*

 *self*.wait\_times.append(wait\_time)

 *# Service the customer*

 service\_time = random.expovariate(*self*.service\_rate)

 *yield* *self*.env.timeout(service\_time)

 *self*.server\_utilization\_time += service\_time

 *self*.customer\_count += 1

 *self*.queue\_lengths.append(len(*self*.server.queue))

*# Simulate the check-in counter*

class CheckInCounter(QueueSystem):

 def \_\_init\_\_(*self*, *env*):

 super().\_\_init\_\_(*env*, 1, SERVICE\_RATE\_CHECKIN, "Check-in Counter")

*# Simulate the service desk*

class ServiceDesk(QueueSystem):

 def \_\_init\_\_(*self*, *env*):

 super().\_\_init\_\_(*env*, 1, SERVICE\_RATE\_SERVICE\_DESK, "Service Desk")

*# Simulate the specialized service counters (M/M/3)*

class SpecializedServiceCounter(QueueSystem):

 def \_\_init\_\_(*self*, *env*):

 super().\_\_init\_\_(*env*, SPECIALIZED\_SERVERS, SERVICE\_RATE\_SPECIALIZED, "Specialized Counter")

*# Simulation function*

def run\_simulation():

 env = simpy.Environment()

 check\_in = CheckInCounter(env)

 service\_desk = ServiceDesk(env)

 specialized\_counter = SpecializedServiceCounter(env)

 def customer\_arrival(*env*):

 *while* True:

 *# Customer arrives at the check-in counter*

 *yield* *env*.timeout(random.expovariate(ARRIVAL\_RATE\_CHECKIN))

 *env*.process(check\_in.process\_customer(*env*.now))

 *# After check-in, customer goes to service desk*

 *yield* *env*.process(service\_desk.process\_customer(*env*.now))

 *# After the service desk, customer goes to one of the specialized service counters*

 *yield* *env*.process(specialized\_counter.process\_customer(*env*.now))

 *# Start the customer arrival process*

 env.process(customer\_arrival(env))

 env.run(*until*=SIM\_TIME)

 *# Performance metrics calculations*

 def print\_metrics(*queue\_system*):

 avg\_waiting\_time = np.mean(*queue\_system*.wait\_times)

 avg\_queue\_length = np.mean(*queue\_system*.queue\_lengths)

 utilization = *queue\_system*.server\_utilization\_time / (SIM\_TIME \* *queue\_system*.server.capacity)

 print(f"Results for {*queue\_system*.label}:")

 print(f"Average Waiting Time (Wq): {avg\_waiting\_time:.2f} minutes")

 print(f"Average Queue Length: {avg\_queue\_length:.2f} customers")

 print(f"Server Utilization (ρ): {utilization:.2f}\n")

 *# Print metrics for each station*

 print\_metrics(check\_in)

 print\_metrics(service\_desk)

 print\_metrics(specialized\_counter)

*# Run the simulation*

run\_simulation()

