CS166 First Project

Waiting for the Bus

Code: https://colab.research.google.com/drive/1qTRntBj99fBeKkG4pUmbnW03OOeT7ExM?usp=sharing

Overview

In this project, I complete the implementation of a bus route simulation. I experiment with different strategies for setting departure times on the bus route and measure how well the different strategies work.

The purpose of this assignment is to go through the entire modeling, simulation, and analysis process. Throughout this course, this modelinng-simulation-analysis process will be repeated a few times. This is an opportunity to learn.

Feedback and Grading

The part of the work that I would like feedback on the most is the descriptive analysis of the comparison between expected analytical results and empirical results.

Basic Model (Required)

A circular bus route operates 24 hours per day. The buses on the route allow passengers to embark and disembark at 15 different stops before looping back to the start of the route. At each bus stop, passengers wait until the next bus arrives, get on the bus, and finally get off the bus at their chosen destination. The rate at which passengers join the queue at a bus stop is constant and does not depend on the time of day.

The time (in minutes) between consecutive passengers joining a queue at each bus stop is assumed to come from an exponential distribution with rate parameter λ = 1. Each passenger chooses uniformly at random a destination that is at most 7 stops away from where they start.

In queueing theory, we can use an M/G/1 queue as a model to a Bus System where arrivals are Markovian or in other words that come from an exponential distribution.

In this scenario the average arrival rate $\lambda = 1$ which means that the arrival distribution can be expressed in the following way.

```
average_arrival_rate = 1
arrival_distribution = sts.expon(scale=1/average_arrival_rate)
```

We can represent the 15 different stops in the bus route in a list.

```
bus_stops = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
```

The instructions mention that the passenger chooses uniformly at random a destination that is at most 7 stops away from where they start. An additional assumption is that their destination is at least 1 stop away from where they start because otherwise there is no reason for them to queue. For the sake of this implementation of the simulation, we can also assume that the passenger gets initially assigned to one of the stops uniformly at random. We represent that in the following way:

```
initial_bus_stop = random.randint(0,14)
destination_bus_stop = (initial_bus_stop + random.randint(1,7)) % 15
```

An example for a single passenger below:

```
import random
initial_bus_stop = random.randint(0,14)
destination_bus_stop = (initial_bus_stop + random.randint(1,7)) % 15
print("The initial bus stop is", initial_bus_stop, "and the destination bus stop is", destination_bus_stop)
```

```
The initial bus stop is 14 and the destination bus stop is 2
```

When a bus stops, the time (in minutes) it takes passengers to disembark is normally distributed with mean $\mu_1 = 0.03 \cdot n$ and standard deviation $\sigma_1 = 0.01 \cdot \sqrt{n}$ where n is the number of passengers who want to disembark.

** We know that the maximum capacity of each bus is 130 passengers, so the maximum number (given a full bus) of people that can disembark is also 130. There exists a possibility that no passenger disembarks which means that at a given stop if the bus has 130 passengers already and no passenger disembarks then no passenger in the queue embarks either. We can also assign the number of people in the bus (currently) and constrain the number of disembarking passengers by that number. We can assume that the number n of passengers who want to disembark is simulated uniformly at random.

```
import numpy as np
from math import sqrt

# The number of people in the bus in this case will have to be initialized manually
number_of_people_in_the_bus = 130
number_of_people_in_the_bus = 130

number_of_disembarking_passengers = random.randint(0, number_of_people_in_the_bus)
number_of_seats_available_before_disembarking = 130 - number_of_people_in_the_bus
number_of_seats_available_after_disembarking = number_of_disembarking_passengers + number_of_seats_available_before_disembarking
time_to_disembark = np.random.normal(loc=0.03 * number_of_disembarking_passengers, scale=0.01 * sqrt(number_of_disembarking_passengers))

print("There are", number_of_people_in_the_bus, "passengers, so there are", number_of_seats_available_before_disembarking, "seats available be
print("The time (in minutes) it takes", number_of_disembarking_passengers, "passengers to disembark is", time_to_disembark, "minutes.")
print("There are", number_of_seats_available_after_disembarking, "seats available after disembarking takes place.")
```

```
There are 130 passengers, so there are 0 seats available before disembarking.
The time (in minutes) it takes 8 passengers to disembark is 0.23989335123104355 minutes.
There are 8 seats available after disembarking takes place.
```

After making the calculation for a few of this scenarios, I feel like the mean and standard deviation for the normally distributed time to disembark might be unrealistic. However, I will keep it as it is for sake of the simulation.

For example, the time it takes 20 passengers to disembark is about 30 seconds which is not realistic.

The time it takes passengers in the queue to get on the bus is also normally distributed with mean $\mu_2 = 0.05 \cdot n$ and standard deviation $\sigma_2 = 0.01 \cdot \sqrt{n}$ where n can be one of two options: it is either the number of seats available in the bus (when there are more people in the queue vs seats available in the bus) or it is the number of people in the queu (when there are enough seats available in the bus). Aditionally, no passengers can embark while other passengers are still disembarking which means that the total time to "embark" for the people in the queue would be the time it takes for the passengers to disembark plus the time it takes them to embark.

```
# The number of people in the queue in this case will have to be initialized manually
number_of_people_in_the_queue = 35

if number_of_seats_available_after_disembarking < number_of_people_in_the_queue:
    number_of_embarking_passengers = number_of_seats_available_after_disembarking
else:
    number_of_embarking_passengers = number_of_people_in_the_queue

time_to_embark = np.random.normal(loc=0.05 * number_of_embarking_passengers, scale=0.01 * sqrt(number_of_embarking_passengers))
total_time_to_embark = time_to_embark + time_to_disembark

print("The time it takes for", number_of_embarking_passengers, "passengers to embark is", time_to_embark, "minutes.")
print("However, passengers in the queue have to wait", time_to_disembark, "minutes before they can embark.")
print("So, in total, the time it takes for", number_of_embarking_passengers, "passengers to embark is", total_time_to_embark, "minutes.")
```

```
The time it takes for 8 passengers to embark is 0.366506192504418 minutes.
```

```
However, passengers in the queue have to wait 0.23989335123104355 minutes before they can embark.
So, in total, the time it takes for 8 passengers to embark is 0.6063995437354616 minutes.
```

Similarly to disembarking time, after making the calculation for a few of this scenarios, the mean and standard deviation for the normally distributed time to embark is a bit unrealistic. However, I will keep it as it is for sake of the simulation.

For example, the time it takes 8 passengers to embark is about 20 seconds (without considering the time they have to wait for passengers to disembark).

The maximum capacity of each bus is 130 passengers. If the bus is full, no more passengers can embark and they have to wait for the next bus to arrive.

The travel time of the bus between consecutive bus stops follows a normal distribution with parameters $\mu_3 = 2$ minutes and $\sigma_3 = 0.5$ minutes.

```python
travel_time_between_bus_stops = np.random.normal(loc=2, scale=0.5)
print("The travel time between bus stops is", travel_time_between_bus_stops)
```

```
The travel time between bus stops is 2.357526791869281
```

So, the average service rate would include the travel time, the time for disembarking, and the time of embarking.

```python
average_service_time = travel_time_between_bus_stops + total_time_to_embark
print("The average service time (amount of time taken for passenger in the next bus stop to embark) is", average_service_time)
```

```
The average service time (amount of time taken for passenger in the next bus stop to embark) is 2.9639263356047425
```

Considering the basic model above, we can calculate the average passenger waiting time for a bus system followin M/G/1 queues propertie. Waiting time here is the duration between a passenger joining a queue at a bus stop and when the passenger gets on the bus.

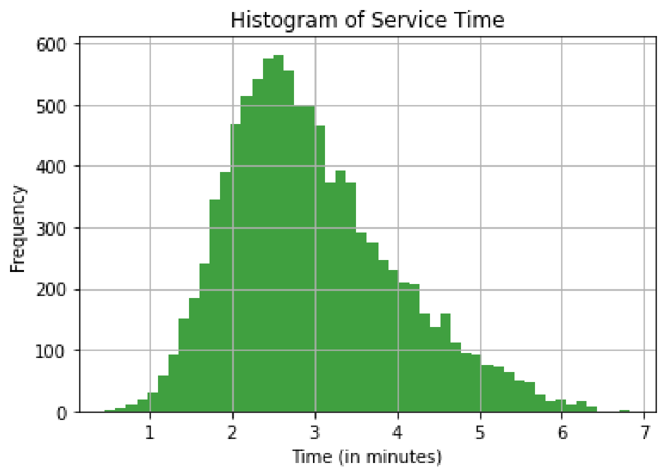
The average service time $\tau = \frac{1}{\lambda} = 1$

```python
import statistics
mu = statistics.mean(service_time_list)
```

```python
sigma_stdev = statistics.stdev(service_time_list)
```

```
# the histogram of the data
n, bins, patches = plt.hist(service_time_list, 50, density=False, facecolor='g', alpha=0.75)

plt.xlabel('Time (in minutes)')
plt.ylabel('Frequency')
plt.title('Histogram of Service Time')
plt.grid(True)
plt.show()
```



```
# the histogram of the data
n, bins, patches = plt.hist(total_time_spent, 50, density=False, facecolor='orange', alpha=0.75)

plt.xlabel('Time (in minutes)')
plt.ylabel('Frequency')
plt.title('Histogram of Total Time Spent in System')
plt.grid(True)
plt.show()
```

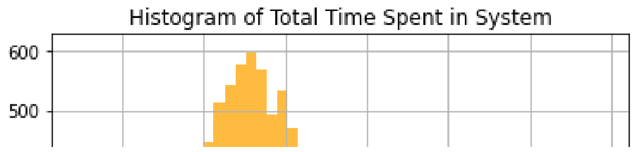


Figure. The average waiting time in queueing theory means that in this scenario the passenger waits for the bus to arrive and for passengers to disembark. The time spent embarking is the time being served. This represents the total time spent in the system which also includes serving time.

```python
# Run a short test with an M/M/1 queue
import scipy.stats as sts
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm

average_arrival_rate = 1
arrival_distribution = sts.expon(scale=1/average_arrival_rate)

distribution_time_to_disembark = sts.norm(loc=0.03 * number_of_disembarking_passengers, scale=0.01 * sqrt(number_of_disembarking_passengers))
distribution_time_to_embark = sts.norm(loc=0.05 * number_of_embarking_passengers, scale=0.01 * sqrt(number_of_embarking_passengers))
distribution_travel_time_between_bus_stops = sts.norm(loc=2, scale=0.5)

tau = 1/mu
sigma = sigma_stdev
service_distribution = sts.norm(loc=tau, scale=sigma)

# service_distribution = sts.expon(scale=1/1)
bus_system = run_simulation(arrival_distribution, service_distribution, run_until=100)
print(f'There is/are {bus_system.queue.people_in_queue} people in the queue')
```

```
There is/are 1 people in the queue
```

At equilibrium, the values such as the waiting time average to total amount a customer spends in the queue, so that would actually only include the bus travel time (passenger waiting for the bus) and the time it takes for current passengers to disembark.

The waiting time for M/G/1 queues follows

$$\frac{\rho \cdot \tau}{2 \cdot (1-\rho)} \cdot \left(1 + \frac{\sigma^2}{\tau^2}\right)$$

where tau and sigma squared are the mean and variance of the servie time and not the rate distribution. From above we got that tau is 1 / mu

```python
tau_equilibrium = 1 / mu
variance_equilibrium = sigma_stdev ** 2
rho_equilibrium = tau_equilibrium # this gets multiplied by lambda which is 1

average_waiting_time = (rho_equilibrium * tau_equilibrium) / (2 * (1 - rho_equilibrium)) * (1 + (variance_equilibrium/tau_equilibrium**2))
print("The average waiting time is", average_waiting_time, "minutes")
```

```
The average waiting time is 0.8568096372768091 minutes
```

```python
# Run the M/G/1 queue experiment

tau = 1/mu
sigma = sigma_stdev
service_distribution = sts.norm(loc=tau, scale=sigma)

np.random.seed(123)
mg1_mean, mg1_std_err = run_experiment(
    arrival_rate_list, service_distribution, run_until, num_trials)
print('\nM/G/1 experiment complete')
```

```
100%|██████████| 6/6 [00:20<00:00,  3.40s/it]
M/G/1 experiment complete
```

```python
# Plot the M/G/1 results

def theoretical_mg1(rho):
    return rho**2 / 2 / (1-rho) * (1 + sigma**2 / tau**2)

rho = arrival_rate_list * tau
make_error_plot('M/G/1', rho, mg1_mean, mg1_std_err, theoretical_mg1)
```

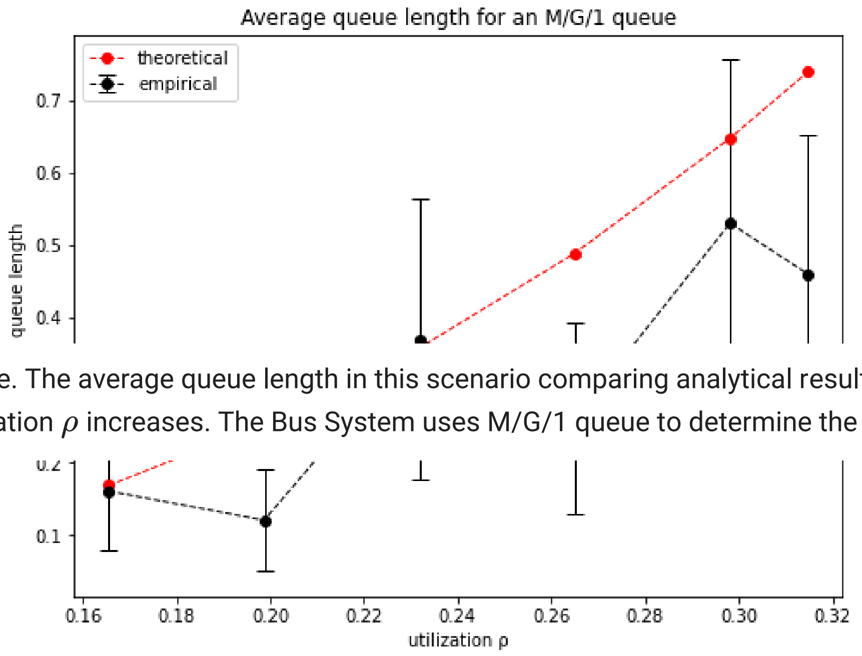


Figure. The average queue length in this scenario comparing analytical results and empirical results. Reveals the average queue lenght as the utilization ρ increases. The Bus System uses M/G/1 queue to determine the properties of the different components of the simulation.