First Project

Minerva University

CS166: Modeling and Analyses of Complex Systems

Professor Ribeiro

January 2023

Table of Contents

Word count: 2868

Introduction and Model Description

Queueing theory is a mathematical model of waiting lines and queues that, if applied correctly, can help optimize waiting time, queue length, and other variables.

The queueing theory can be applied to describe a system that possesses the following characteristics:

- Customer arrival process with a constant average arrival rate.
- Customer serving process.
- Certain number of servers.
- 1 limitless queue.
- Arrival events are independent of each other: the time a customer decides to join the queue does not depend on other customers.

In this project, we are applying queueing theory to describe the grocery store system. Specifically, we are going to apply c M/G/1 models, where c is the number of queues (or servers since we have 1 server per queue), M denotes exponentially distributed arrival rate, G - normally distributed service rate, 1 - 1 server per queue. The assumptions of this type of model according to the queueing theory are as follows:

1. The waiting times between arrivals (or arrival rate) are exponentially distributed (M in the model name abbreviation).
2. Service time follows a general distribution: we would need to know the mean and variance of this distribution (G in the model name abbreviation).

3. There is exactly 1 server (1 in the model name abbreviation).
4. There is no limit on the queue length.

Now that we have discussed the implications of the queueing theory and M/G/1 model specifically, we can analyze how well this model fits our scenario. Our system describes the arrival of customers to the supermarket who join the queue independently and at a constant rate. In our case, the arrival rate follows an exponential distribution with a constant rate parameter $\lambda = 1$ customer per minute. Simulating the arrival rate with an exponential distribution is an essential characteristic of this model due to the distribution's memoryless property. This property means that the past has no impact on the future, meaning no matter how long a customer has been already waiting, the time until the next customer joins the queue is $\frac{1}{\lambda}$. Our system has a serving process with 1 server per queue where the service time follows the normal distribution with a mean parameter $\tau_1 = 3$ minutes and a standard deviation parameter $\sigma_1 = 1$ minute. In addition to the above characteristics, our scenario describes the extended service of customers after they have been served. The extended service time is also normally distributed with mean parameter $\tau_2 = 5$ minutes and standard deviation parameter $\sigma_3 = 2$ minutes. The queues in our system do not have a maximum length.

Additional assumptions of our system are as follows:

- The store is open from 9 am till 8 pm.
- Customers join the shortest queue.

- The servers take the same time to treat every customer (the service distribution and its parameters always stay the same).
- The servers take no breaks.
- There are no ‘rush hours’ when people start joining the queue more frequently (the arrival distribution and its parameters always stay the same).
- The extended service takes the same time for all customers (follows the same distribution with the same parameters that do not change).

We can further characterize the model of our scenario in terms of the variables (something we can measure or estimate), updating rules (the rules of the simulation that guide the queuing process, arrival, service, and other aspects and could be updated), and inputs and outputs:

- Variables:
    - The time when a certain customer joins the queue.
    - The time between certain customers joining the queue (interarrival time).
    - The time it takes to serve a customer.
    - The time it takes to provide extended service to a customer
    - Average queue length (includes the number of customers waiting in the queue only).
    - Average waiting time (the total amount of time the customer spends in the queue and does not include the time spent being served).
    - Average response time (how long it takes a customer to get all the way through the system, from the time of arrival until the departure time after having been served, not including extended service).

  - Average number of customers in the system (includes the number of customers waiting in the queue plus the number of customers currently being served).
- Updating rules:
  - Exponential distribution with $\lambda = 1$ for arrival rate.
  - Normal distribution with mean $\tau_1 = 3$ minutes and SD $\sigma_1 = 1$ minute for service time.
  - Normal distribution with mean $\tau_1 = 5$ minutes and SD $\sigma_1 = 2$ minute for extended service time.
  - The probability that a customer would need extended service $p = 0.05$.

Overall, the model takes the updating rules (the described distributions above for the defined variables), the number of queues (and servers), and the time when the store closes as inputs and produces the values for average queue length, average waiting time, average response time, and average number of customers in the system as outputs.

We can see that our system satisfies most of the queueing theory and M/G/1 model assumptions, with an exception of the number of queues, and hence, the process of choosing which queue to join. In our scenario, we have multiple queues while queuing theory allows only 1 queue, and the customers join the shortest queue rather than join the queues randomly, which is not even specified in the queueing theory model assumptions since there is supposed to be only 1 queue.

Although this assumption does not hold in our case, the rest of the assumptions are satisfied, which means we can use queuing theory to model the system and apply the theoretical tools associated with it to characterize the system accurately enough and produce reasonable approximations of the system's key features, such as average waiting time, average queue length, and average response time.

Theoretical Analyses

As discussed above, for our scenario we are implementing c M/G/1 models. In our scenario, we have multiple queues with 1 server each, hence multiple servers, where customers join the shortest queue, while the model assumes there is only 1 queue with 1 server. In addition, we introduce extended service in our system, which would affect the response time and the total number of customers in the system. Since these assumptions are violated, mathematical formulas associated with this M/G/1 model will not yield the exact correct results for average waiting time, average customer response time, average queue length, and an average number of customers in the system but rather produce good enough approximate estimates for these variables. Although the assumption of 1 server is violated, we are going to apply M/G/1 model for each queue separately since each queue has 1 server.

Since we are applying M/G/1 model to each of the c queues instead of the whole system, we need to adjust the model's parameters. Since we are assuming that each queue gets the same number of people per unit of time, we set the number of people joining one particular queue per unit of time (minute) to be $\frac{\lambda}{c} = \frac{1}{c}$ by dividing the original rate equally between all queues.

With that in mind, we can compute the aforementioned variables to characterize the system using M/G/1 model's mathematic formulas:

- Average waiting time:

  $\frac{\rho\tau_1}{2(1-p)}(1+\frac{\sigma_1^2}{\tau_1^2}) = \frac{\frac{1}{c}\cdot 3\cdot 3}{2(1-\frac{1}{c}\cdot 3)}(1+\frac{1}{9}) = \frac{9}{2c-6}\cdot\frac{10}{9} = \frac{5}{c-3}$ *minutes*, where τ_1 is average service time (mean), σ^2 is the variance of the service time, and ρ is utilization, defined as $\lambda\tau_1$.

- Average response time (how long it takes for a customer to leave the system from the moment they join the queue, not including extended service):

  $\tau_1 + \textit{average waiting time} = 3 + \frac{5}{c-3} = \frac{3c-4}{c-3}$ *minutes*

- Average queue length: $\lambda \cdot \textit{average waiting time} = \frac{1}{c}\cdot\frac{5}{c-3} = \frac{5}{c^2-3c}$ *customers*

- Average number of customers in the system:

  $\lambda \cdot \textit{average response time} * \textit{number of queues} = \frac{1}{c}\cdot\frac{3c-4}{c-3}\cdot c = \frac{3c-4}{c-3}$ *customers*

Note that since the definitions of the formulas apply per queue, we need to multiply the average number of customers in the system by the number of queues c. Also, the formulas do not account for 5% of people getting extended service and the time it takes to get it, meaning the real values (corresponding to our scenario) of average response time and an average number of customers in the system would be slightly bigger.

The results of this theoretical analysis inform an interesting regime in the parameter space. The formulas above hold under certain conditions. The utility value From the average waiting time estimate, we can tell that the number of queues in our system should be at least 3 to satisfy the conditions of our model. If the number of queues is 3, the denominator of the average waiting time is 0, leading to invalid results. If the number of queues is less than 3, then the average waiting time is negative, which is impossible.

Now we can adjust the formulas to produce estimates for the extended service scenario. We can start by imagining that after customers have been served, they join 1 queue according to an exponential distribution with some parameter λ where the manager is 1 server and provides service to customers according to a normal distribution with given parameters. These characteristics allow us to apply M/G/1 model to model the extended service.

Before using the formulas, we need to decide what the value of λ could be. Since we are given that $\lambda = 1$ customer per unit of time (minute) joins the queue, and the probability of each customer being in need of extended service is 0.05, then the rate of joining the extended service queue would be $\lambda_2 = \lambda * p = 1 * 0.05 = 0.05$ customers per minute. Since the arrival rate parameter has changed, we also need to find the new utilization parameter: $\rho_2 = \lambda_2\tau_2$.

Now we can compute the variables to characterize the extended service system using M/G/1 model's mathematic formulas:

- Average waiting time in the extended service queue::

  $\frac{\rho\tau_2}{2(1-\rho)}(1 + \frac{\sigma_2^2}{\tau_1^2}) = \frac{0.05\cdot5\cdot5}{2(1-0.05\cdot5)}(1 + \frac{4}{25}) = \frac{1.25}{1.5} \cdot 1.16 = 0.97\, minutes$, where τ_2

is average extended service time (mean), σ_2^2 is the variance of the extended service time, and ρ_2 is utilization, defined as $\lambda_2\tau_2$.

- Average response time (how long it takes for a customer to leave the system from the moment they join the queue, not including extended service):

  $\tau_2 + average\ waiting\ time\ =\ 5\ +\ 0.97\ =\ 5.97\ minutes$

- Average queue length:

  $\lambda\ \cdot\ average\ waiting\ time\ =\ 0.05\ \cdot\ 0.97\ =\ 0.0483\ customers$

- Average number of customers in the system:

  $\lambda\ \cdot\ average\ response\ time\ =\ 0.05\ \cdot\ 5.97\ =\ 0.2985\ customers$

Python Implementation

To implement the Grocery Store system described above, we utilized Event and Schedule python classes and adjusted the Queue and GroceryStore python classes from the class. The adjusted class methods and functions are as follows:

class Queue:

In the __*init*__ method, we added additional attributes such as queue_type to mark if the queue is a regular queue or a manager one. Three counter variables and three empty dictionaries were added to record the number of people in different statuses (either “in queue” or “being served”) and the customer arrival, departure, and response times.

When a new customer joins the cashier queue, the *add_customer* method adds the customer to the shortest queue in the store at the given time. The *start_serving_customer* method samples from the service distribution (manager service distribution for manager queue) when scheduling the event with *finish_serving_customer* method. As the service distribution is Gaussian, we account for the possibility of sample values less than or equal to 0 by resampling if it occurs (see Appendix A).

class GroceryStore:
To this class, we added additional attributes such as *manager_service_distribution*, *num_of_queues*, and *run_until* to account for the different number of queues in the store and slightly probable extended services. *Run_until* tells the number of minutes after the store opens, i.e., 9 AM in our case.

The function *run_simulation* runs a simulation of the Grocery Store system with given arguments. It initiates a *plot_data* dictionary for keeping the recorded data for measurements such as queue lengths, customer waiting and response times, and the number of people in the system for each simulation. This dictionary also stores those measurements for the manager queue, which were used for comparison with theoretical estimates (see Appendix B).

The function *test_simulation* runs a sample simulation of the system, and we used this function to make sure that the store served the right number of customers in a day and that the queues were empty when the simulation ended (see Appendix C). This function also plots the average queue length over time. From our test cases with 1 and 3 queues, we concluded that the simulation runs without any error as the number of customers served and the average queue length plots were close to our intuition. For the 1-server queue system, we expected an increasing trend in the average queue length until the closing time, which is around 660 mins, because the average customer arrival time (~1 min) was significantly less than the average service time (~5 min).

For our theoretical analyses, we defined three additional functions, including *confidence_interval*, *calculate_statistics*, and *run_experiment*. The function *run_experiment* takes a list of a different number of queues, i.e., 1 to 10, runs the simulation with differing numbers of queues, calculates the statistics used in the theoretical analysis, and plots the data and confidence intervals for the mean for each measurement.

Empirical Analysis

We will first, compare the theoretical and empirical results to examine how the difference in assumption might have brought numerical differences. We will compare the below metrics to see what assumptions may have affected them.

- Average waiting time: $\frac{5}{c-3}$.
- Average response time: $\frac{3c-4}{c-3}$
- Average queue length: $\frac{5}{c^2-3c}$
- Average number of customers in the system: $\frac{3c-4}{c-3}$

# of cashiers	Empirical results	Theoretical results
1	686	$\frac{1}{2}$
2	168	-2
3	14.8	$\frac{5}{0}$ ← undetermined
4	2.07	8
5	1.74	$\frac{11}{2} = 5.5$
6	1.65	$\frac{14}{3} = 4.67$
7	1.66	$\frac{15}{4} = 3.75$
8	1.63	4
9	1.65	$\frac{23}{6} = 3.83$
10	1.64	$\frac{26}{7} = 3.71$

Table 1: Empirical results v.s. Theoretical results for average waiting time.

We can ignore until c = 3, as it produces either a negative number or an undetermined value. Upon that, the value difference between empirical analysis and theoretical analysis decreases.

Comparing theoretical results with empirical results

# of cashiers	Empirical results	Theoretical results
1	657	$\frac{1}{2}$
2	170	-2
3	16.5	$\frac{5}{0}$ ← undetermined
4	5.07	9
5	4.66	$\frac{11}{2} = 5.5$
6	4.60	$\frac{14}{3} = 4.67$
7	4.67	$\frac{17}{4} = 4.25$
8	4.59	4
9	4.68	$\frac{23}{6} = 3.83$
10	4.63	$\frac{26}{7} = 3.71$

Table 2: Empirical results v.s. Theoretical results for average response time

The first three values are, again, out of the regime of the theoretical analysis, so we can ignore them. From c = 4, it slowly converges to the empirical results but accelerates the decrease faster than the empirical results. This may be due to the extra manager time taken in the empirical simulation whereas the theoretical results do not take this into account.

# of cashiers	Empirical results	Theoretical results
1	221	$-\frac{5}{2}$
2	56.6	$-\frac{5}{2}$
3	5.34	$\frac{5}{0} \rightarrow$ undefined
4	0.680	$\frac{5}{4} = 1.25$
5	0.430	$\frac{1}{2} = 0.5$
6	0.350	$\frac{5}{18} = 0.278$
7	0.300	$\frac{5}{28} = 0.179$
8	0.260	$\frac{1}{8} = 0.125$

9	0.230	$\frac{5}{54} = 0.0926$
10	0.210	$\frac{1}{14} = 0.0714$

Table 3: Empirical results v.s. Theoretical results for average queue length.

Even though the values are different, average queue length, both empirically and theoretically, decreases with a similar pattern. 3 is the regime where, below the point, waiting time enters are negative. Hence, we can ignore the results of # of cashiers < 3. We still see much variance between the empirical results and the theoretical results. This could be due to the difference in assumptions we have between theoretical and empirical analysis. In the empirical analysis, customers always entered the shortest queue, which reflects the reality of grocery store queues. In the theoretical analysis, however, we simply divide the arrival rate by 3 for each of the M/G/1 queues. Hence, we see larger values for theoretical results than the empirical result.

# of cashiers	Empirical results	Theoretical results
1	230	$\frac{1}{2}$
2	111	-2
3	17.7	$\frac{5}{0}$
4	5.78	8

5	5.54	$\frac{11}{2} = 5.5$
6	5.37	$\frac{14}{3} = 4.67$
7	5.44	$\frac{17}{4} = 3.75$
8	5.59	4
9	5.39	$\frac{23}{3} = 3.83$
10	5.43	$\frac{26}{7} = 3.71$

Table 4: Empirical results v.s. theoretical results for an average number of customers in the system

Similarly, the number of customers in the system has a similar trend of average decrease.

Optimizing for the number of cashiers

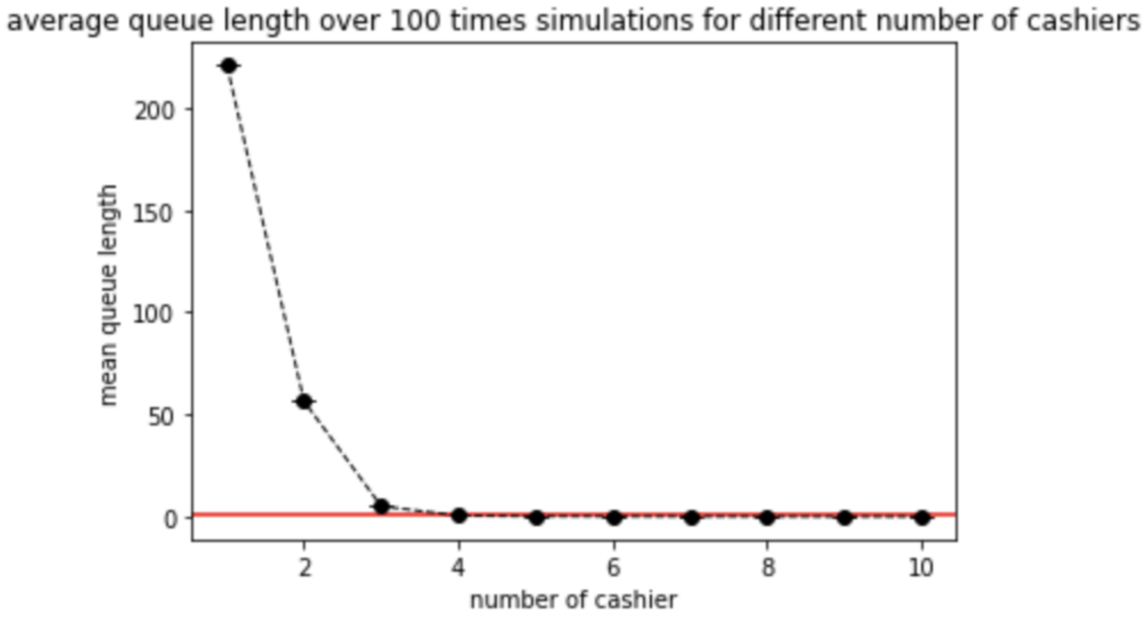


Figure 1: Average queue length for different numbers of cashiers

We see a sharp decline in mean queue length when we increase the number of cashiers from 1 to 2, and the rest are asymptotic around queue length = 0. This means that an increase in the number of cashiers does not have a constant effect on the mean queue length, but rather behaves in an exponential manner. We could zoom in further to see the asymptotic behavior closer below.

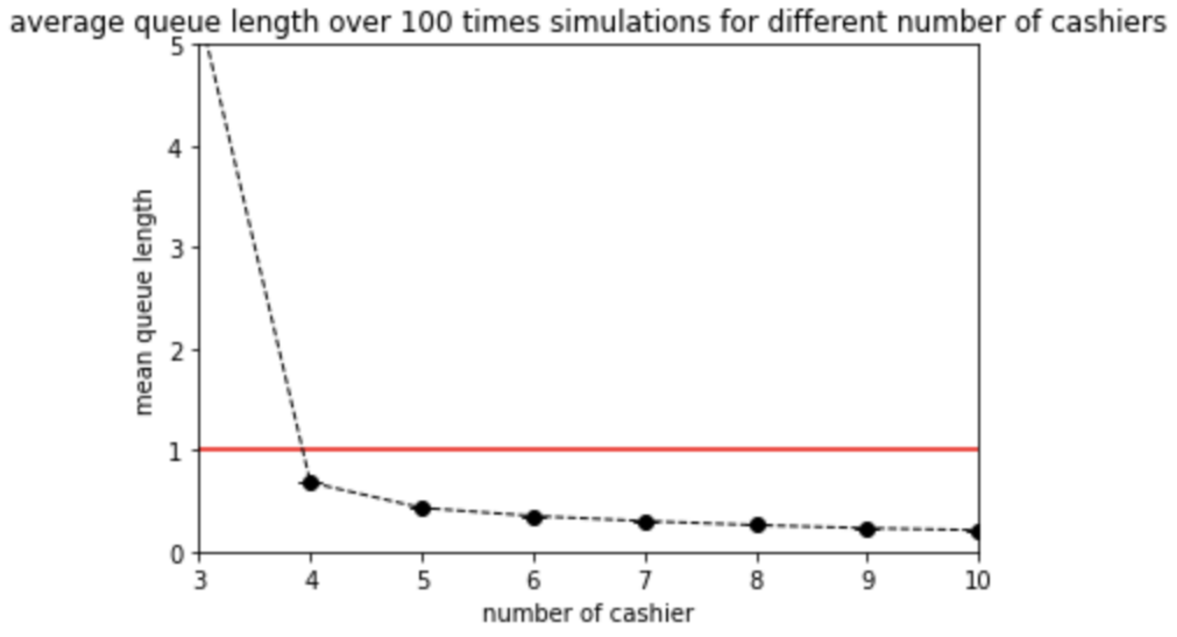


Figure 2: Average queue length for a different number of cashiers (zoomed in)

We see that the mean queue length is already below 1 when we have 4 cashiers, and it declines slowly and converges to 0 as we increase the available cashiers.

To optimize the profit of the grocery store, we can plot the mean queue length against the marginal cost of hiring an additional cashier. The minimum wage in San Francisco is 16.99 USD per hour. The store works from 9am to 8pm, costing $16.99\times11 = 186.89$USD per additional cashier.

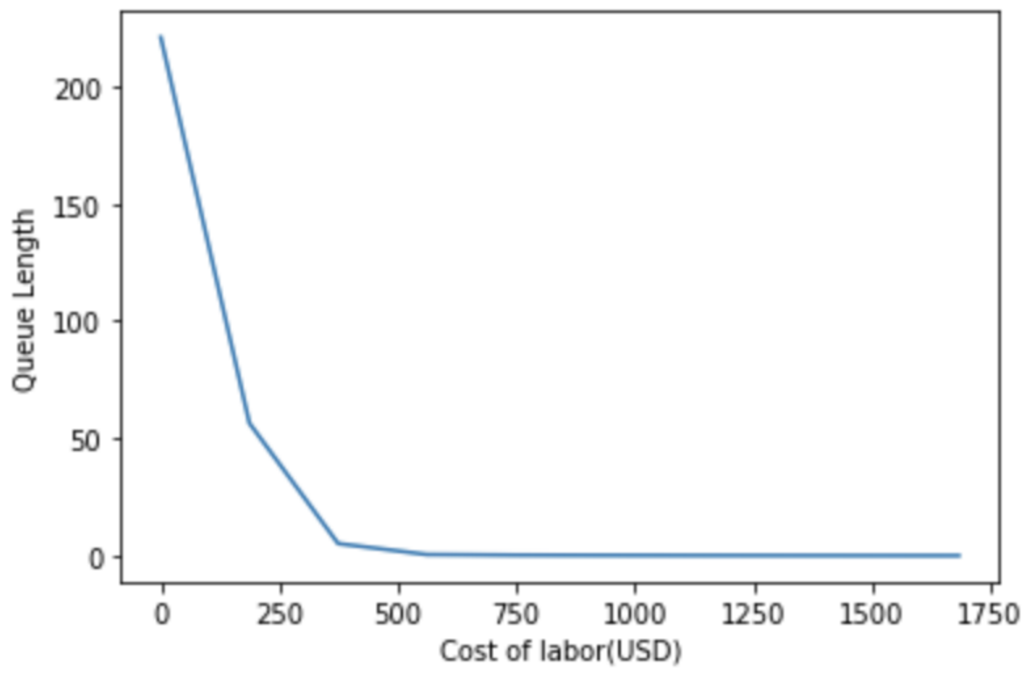


Figure 3: Cost for increasing the cashier and decrease in queue length

As grocery store owners, we would want the most cost-efficient way to reduce the queue lengths. We can take the delta of queue length per increase in labor to observe this.

# of cashier	Delta (queue length change per one additional cashier)	Cost per 1 queue length decrease(USD)
Calculation	Queue_length(t1) - queue_length(t), where t is the number of cashiers	labor cost per day(186.89USD)/delta
1	-	-

2	-164	1.14
3	-51.26	3.65
4	-4.66	40.1
5	-0.25	747.56
6	-0.12	1557
7	-0.05	3738
8	-0.04	4672
9	-0.03	6230
10	-0.02	9345

Table 5: cost per one unit decrease in queue length

Increasing the number of cashiers from 1 to 2 results in the decrease of queue length by 164, and the cost per one customer decrease in the queue is 1.14USD, which is reasonable. From when the number of cashiers = 4, however, it seems to be less cost-efficient. If there is a significant increase in revenue by decreasing the queue length by 4.66 people, it would be worthwhile but it does not sound reasonable to pay 40.1USD to decrease one customer from the queue. From this table, we can advise having either 2 or 3 cashiers, which would be optimal from a cost-minimizing point of view.

Looking at the other metrics also supports this idea. Zooming in, we see a critical decrease in average waiting time, response time, max queue length, and total people in the system when we increase the cashier up until c = 4, and the change stabilizes into asymptotic behavior afterward. (see appendix B)

LO/ HC Applications

#***Modeling*** - We clearly described our scenario in terms of assumptions and relevant characteristics, described the model of our choice, and justified why the model we chose is a reasonable representation of our scenario. Specifically, we described the model's assumptions and compared them to the assumptions of our scenario, commenting on how our scenario violates certain assumptions and how it affects the model being a better or worse approximation. We also characterized the model in terms of its variables, updating rules, inputs, and outputs. Lastly, we made clear connections between the modeling theory and the particulars of the scenario by constantly comparing and juxtaposing their features and assumptions.

#***TheoreticalAnalysis*** - We correctly identified the specific type of model we are using, which helped us properly apply relevant formulas to our analyses to characterize the variables of interest. We performed appropriate mathematical analysis to find the values for average queue length, waiting time, and other variables. We interpreted the results in the context of our model and scenario, commenting on the validity and accuracy of the results given that some of the model assumptions are violated, and we are simplifying the scenario (manager extended service is separated). Lastly, we analyzed the theoretical results, which helped us discover interesting parameter regimes, such as the number of servers should be more than 3.

#***PythonImplementation*** - We implemented the simulation of the model in Python by using Python classes and methods appropriately. Additionally, we used lists and dictionaries effectively to store the simulation results and data. We defined a specific function that runs a sample simulation of the model. Then, we ran two simple test simulations of the model: one and three servers. The test results matched our intuition, and the function showed that the simulation ran until no more customers were left in the queue. Also, the test_simulation function outputs a plot showing the average number of customers in a queue over time to appropriately demonstrate the state of the system over time.

#***EmpiricalAnalysis*** - Based on our implementation of the grocery store system, we experimented with different numbers of cashiers in the system. We demonstrated our empirical results using appropriate statistical measures and types of plots. Then, we used those to draw a conclusion on how many cashiers we should employ in the grocery store. We defined where the parameter regime lies and did a comparative analysis with the Theoretical results. Lastly, we incorporated the cost-benefit analysis on employing the different numbers of cashiers and made a clear argument on why we should employ 3 cashiers.

#***CodeReadability*** - The code is easily readable and documented appropriately using docstrings and in-line comments. Also, we optimized the code implementation by using simple yet appropriate Python built-in functions, modules, and data structures. We used object-oriented programming appropriately to implement the model functionalities and keep the data in an organized and efficient manner. Lastly, we utilized descriptive variable names throughout the code so that the reader could easily identify what each variable represents.

#***Professionalism*** - We completed all the questions prompted in the assignment instructions through the application of the knowledge we acquired in class. Aside from the well-structured report and code notebook, we presented our theoretical and empirical analyses effectively through the use of mathematical equations and well-produced figures that are easily understood. All the tables and figures are labeled and captioned well.

#***variables*** - When implementing the system, we defined the number of queues as an independent variable and measured the mean waiting and response times and average queue lengths over time as they were dependent on the number of queues. Even if some parts of the system, such as the arrival and service times, were random variables sampled from probability distributions, we defined that the average waiting times and queue lengths were significantly influenced by the independent variables - time and the number of cashiers. While we cannot change or keep time constant, we changed the number of queues, a numerical independent variable, to see the impact it had on the system. Then, we used the outcomes in our empirical analysis.

#***confidenceintervals*** - Through grocery store model simulations, what we want to achieve is to identify population parameter values such as the average customer waiting time and the average queue length of the population. To do so, we used the Central Limit Theorem, which states that the mean of samples follows a normal distribution and can be used to estimate the mean of the population distribution. Thus, with modeling, we assume that we are sampling from the population. With the standard error and mean of the samples, we computed the 95% confidence interval for the population means. Our confidence interval outcomes came out quite narrow because of a large sample size. The larger the sample size, the more confident we become about the population means.

Appendix A

```
import numpy as np
class Queue:
   '''
   Store the properties of a single queue. Each queue has a service time
distribution and
   a queue type (either regular or manager).
   '''
   def __init__(self, service_distribution, queue_type = "regular", manager_queue =
None):
       self.service_distribution = service_distribution
       self.queue_type = queue_type
       if self.queue_type == "regular":
           # If queue type is regular, manager queue is defined
           self.manager_queue = manager_queue
       # We start with an empty queue, the server not busy, and the total 0 people
joined
       self.people_in_queue = 0
       self.people_being_served = 0
       self.total_people_joined = 0

       # Record arrival times
       self.arrival_times = []
       # Record queue departure times
       self.departure_times = []
       # Record service finish times
       self.finish_times = []

   def add_customer(self, schedule):
       '''
       It adds a new customer to the queue.
       '''
       # Add the customer to the queue
       self.people_in_queue += 1
       # Add the customer to the total people joined
       self.total_people_joined += 1
       # Add customer arrival time to queue arrival times list
       self.arrival_times.append(schedule.now)
       if self.people_being_served < 1: # If cashier is not busy
           # This customer can be served immediately
           schedule.add_event_after(0, self.start_serving_customer)


 def start_serving_customer(self, schedule):
       '''
```

```
        This method starts serving a customer and schedules the end of service.
        '''
        # Move the customer from the queue to a server
        self.people_in_queue -= 1
        # Mark the cashier busy
        self.people_being_served += 1
        # Add queue departure time to the queue departure times
        self.departure_times.append(schedule.now)
        # Schedule when the server will be done with the customer
        service_time = self.service_distribution.rvs()
        while service_time < 0: # Resample if the service time sample is less than 0
            service_time = self.service_distribution.rvs()
        schedule.add_event_after(
            service_time,
            self.finish_serving_customer)


    def finish_serving_customer(self, schedule):
        '''
        This finishes the service and starts serving the next customer in the queue.
        If the queue is regular, see if it is in the 5% that joins the manager
queue.
        '''
        # Remove the customer from the server
        self.people_being_served -= 1
        self.finish_times.append(schedule.now)
        if self.people_in_queue > 0:
            # There are more people in the queue so serve the next customer
            schedule.add_event_after(0, self.start_serving_customer)
        # Add customer to the manager queue in the rare case of 5% probability
        if self.queue_type == "regular" and np.random.rand() < 0.05:
            schedule.add_event_after(0, self.manager_queue.add_customer)

class GroceryStore:
    '''
    Implement the grocery store system with one or more queues. It stores the
attributes of
    of the grocery store, including number of queues, arrival distribution, cashier
and
    manager service distribution.
    '''
    def __init__(self, arrival_distribution, service_distribution,
manager_service_distribution, num_of_queues, run_until):
        self.arrival_distribution = arrival_distribution
        self.num_of_queues = num_of_queues
        self.run_until = run_until
        self.manager_queue = Queue(manager_service_distribution, queue_type =
"manager")
        self.queue_list = []
        for i in range(num_of_queues):
            self.queue_list.append(Queue(service_distribution,
manager_queue=self.manager_queue))

    def add_customer(self, schedule):
        # Define the shortest queue
        queue = self.queue_list[0]
        for i in range(self.num_of_queues):
            if self.queue_list[i].people_in_queue < queue.people_in_queue:
                queue = self.queue_list[i]
        # Add the customer to the shortest queue
        queue.add_customer(schedule)
        # Schedule when to add another customer
        if schedule.now < self.run_until:
            # Schedule new customer arrival until the closing time
            schedule.add_event_after(
                self.arrival_distribution.rvs(),
                self.add_customer)

    def run(self, schedule):
        # Schedule when the first customer arrives
        schedule.add_event_after(
            self.arrival_distribution.rvs(),
            self.add_customer)
```

Appendix B

```
def run_simulation(arrival_distribution, service_distribution,
manager_service_distribution, run_until, num_of_queues):
   '''
   This function runs a simulation of the system with given inputs.
   Input:
       - arrival_distribution: customer arrival distribution (exponential)
       - service_distribution: cashier service time distribution (Gaussian)
       - manager_service_distribution: manager service distribution (Gaussian)
       - run_until: the time duration to run the simulation in minutes
       - num_of_queues: the number of cashiers in the grocery store
   Output:
       - grocery_store: the grocery store object of GroceryStore class
       - plot_data: dictionary containing waiting times, response times, people in
queue for all queues
```

```
        and total people in system
    '''
    plot_data = {"Waiting times": [], "Response times": [], "People in queue": [], 
"Total people in system": [],
        "Waiting times manager queue": [], "Response times manager queue": [], 
"People in manager queue": [], "Total people in extended system": []}
    schedule = Schedule()
    grocery_store = GroceryStore(arrival_distribution=arrival_distribution, 
service_distribution=service_distribution, 
manager_service_distribution=manager_service_distribution, 
num_of_queues=num_of_queues, run_until=run_until)
    grocery_store.run(schedule)
    while len(schedule.priority_queue) > 0:
        # Run the next event into the scheduler until the priority queue is empty
        schedule.run_next_event()
        # Put together total people in system
        total_customers = 0
        for i in range(num_of_queues):
            total_customers += grocery_store.queue_list[i].people_in_queue
            total_customers += grocery_store.queue_list[i].people_being_served
        plot_data["Total people in system"].append((total_customers, schedule.now))
        # Put together queue length measures for all queues
        for i in range(num_of_queues):
            plot_data["People in 
queue"].append((grocery_store.queue_list[i].people_in_queue, schedule.now))
        # Put together total people in extended service system
        plot_data["Total people in extended 
system"].append((grocery_store.manager_queue.people_in_queue + 
grocery_store.manager_queue.people_being_served, schedule.now))
        # Put together queue length measure for manager queue
        plot_data["People in manager 
queue"].append((grocery_store.manager_queue.people_in_queue, schedule.now))
    # Record waiting times
    for i in range(num_of_queues):
        plot_data["Waiting times"] += 
(np.array(grocery_store.queue_list[i].departure_times) - 
np.array(grocery_store.queue_list[i].arrival_times)).tolist()
    # Record response times
    for i in range(num_of_queues):
        plot_data["Response times"] += 
(np.array(grocery_store.queue_list[i].finish_times) - 
np.array(grocery_store.queue_list[i].arrival_times)).tolist()
    # Record manager queue waiting times
    plot_data["Waiting times manager queue"] += 
(np.array(grocery_store.manager_queue.departure_times) - 
np.array(grocery_store.manager_queue.arrival_times)).tolist()
    # Record manager queue response times
```

```
   plot_data["Response times manager queue"] +=
(np.array(grocery_store.manager_queue.finish_times) -
np.array(grocery_store.manager_queue.arrival_times)).tolist()
   return grocery_store, plot_data
```

Appendix C

```
# Run a short test with an M/G/1 * c queue
import scipy.stats as sts
import matplotlib.pyplot as plt
from tqdm import tqdm

def test_simulation(num_of_queues):
   '''
   This function runs a test simulation for the given number of queues to see if
the model
   simulation works properly.
   '''
   arrival_distribution = sts.expon(scale=1/1)
   service_distribution = sts.norm(loc=3, scale=1)
   manager_service_distribution = sts.norm(loc=5, scale=2)
   np.random.seed(123)
   # mins since 9 AM (until 8pm)
   run_until = 660

   # Run simulation of the grocery store with given number of queues
   grocery_store, plot_data =
run_simulation(arrival_distribution=arrival_distribution,
service_distribution=service_distribution,
       manager_service_distribution=manager_service_distribution,
run_until=run_until, num_of_queues=num_of_queues)

   # Total number of people that joined each queue
   for i in range(len(grocery_store.queue_list)):
       print(f'The total number of people that have joined queue No. {i+1}:
{grocery_store.queue_list[i].total_people_joined}')
   print(f'The total number of people that have joined manager queue:
{grocery_store.manager_queue.total_people_joined}')

   data = plot_data["People in queue"]
   y = list(zip(*data))[0]
   x = list(zip(*data))[1]
   plt.title(f"M/G/1 * c test for c={num_of_queues}")
   plt.xlabel("Time [mins since 9 AM]")
   plt.ylabel("Average queue length")
```

```
    plt.plot(x, y)
    total_num_customers = 0
    for i in range(len(grocery_store.queue_list)):
        total_num_customers += grocery_store.queue_list[i].total_people_joined
    print(f"Store finished serving {total_num_customers} customers {x[-1]//60} hours
{round(x[-1]%60,2)} mins after 9AM")
```

Appendix D

```
def confidence_interval(data, confidence=0.95):
    a = 1.0 * np.array(data)
    n = len(a)
    m, se = np.mean(a), sts.sem(a)
    h = se * sts.t.ppf((1 + confidence) / 2., n-1)
    return [m-h, m+h]


def calculate_statistics(data):
    #mean
    p_tail = 0.025
    m = np.mean(data)
    n = len(data)
    df = n-1
    m_approx = round(m, 2)
    #starndard error
    sd = np.std(data)
    se = sd/(n-1) #Bessel's correction
    se_approx = round(se, 2)
    confidence_int = confidence_interval(data)
    return[m_approx, se_approx, confidence_int]

def run_experiment(num_queues_list, measurement, num_trials=10, run_until=660):
    '''
    Run an experiment with different number of queues for an M/G/1 * c model.
    Input:
        - num_queues_list: [list] a list that contains different number of queues to
run the experiment with.
        - measurement: [str] a measurement variable
        - num_trials: [int] the number of trials to run a simulation for different
number of queues
        - run_until: [int] the time duration to run the simulation in minutes
    Output:
        - results_mean: the mean value of the given measurement
        - results_std_err: the standard error of the mean for each measurement
    '''
```

```
   # We record only the mean and standard error of the mean for each
experiment/measurement
   results_mean = []
   results_std_err = []

   arrival_distribution = sts.expon(scale=1/1)
   service_distribution = sts.norm(loc=3, scale=1)
   manager_service_distribution = sts.norm(loc=5, scale=2)

   for num in tqdm(num_queues_list):
       results = []
       for trial in range(num_trials):
           grocery_store, plot_data =
run_simulation(arrival_distribution=arrival_distribution,
service_distribution=service_distribution,
               manager_service_distribution=manager_service_distribution,
run_until=run_until, num_of_queues=num)
           if measurement in ["Total people in system", "People in queue", "Total
people in extended system", "People in manager queue"]:
               data = plot_data[measurement]
               results += list(zip(*data))[0]
           elif measurement == "Max queue length":
               data = plot_data["People in queue"]
               results += [max(list(zip(*data))[0])]
           else:
               results += plot_data[measurement]
       #print(results)
       stats = calculate_statistics(results)
       m = stats[0]
       se = stats[1]
       confidence_mean = stats[2] # for the mean
       quantiles = np.quantile(results, [0.025, 0.975]) # 95% HDI for the samples
       results_mean.append(m)
       results_std_err.append(se)

       if (not "manager" in measurement) and (not "extended" in measurement):
       # Make a histogram
           plt.figure()
           plt.hist(results, bins = 14, edgecolor='white')
           plt.axvline(x = m, color = 'red', label = 'mean')
           plt.axvline(x = confidence_mean[0], color = 'orange', label = '95% CI
for the mean')
           plt.axvline(x = confidence_mean[1], color = 'orange')
           plt.axvline(x = quantiles[0], color = 'black', label = '95% HDI for
samples')
           plt.axvline(x = quantiles[1], color = 'black')
           plt.xlabel(measurement)
           plt.ylabel('Frequency')
           plt.title(f'{measurement} over {num_trials} trial of simulation with
{num} cashiers')
           plt.legend()
           plt.show()
           print(stats)

   # Convert lists to arrays so we can easily add, subtract, and multiply them
   results_mean = np.array(results_mean)
   results_std_err = np.array(results_std_err)

   return results_mean, results_std_err
```

Appendix E

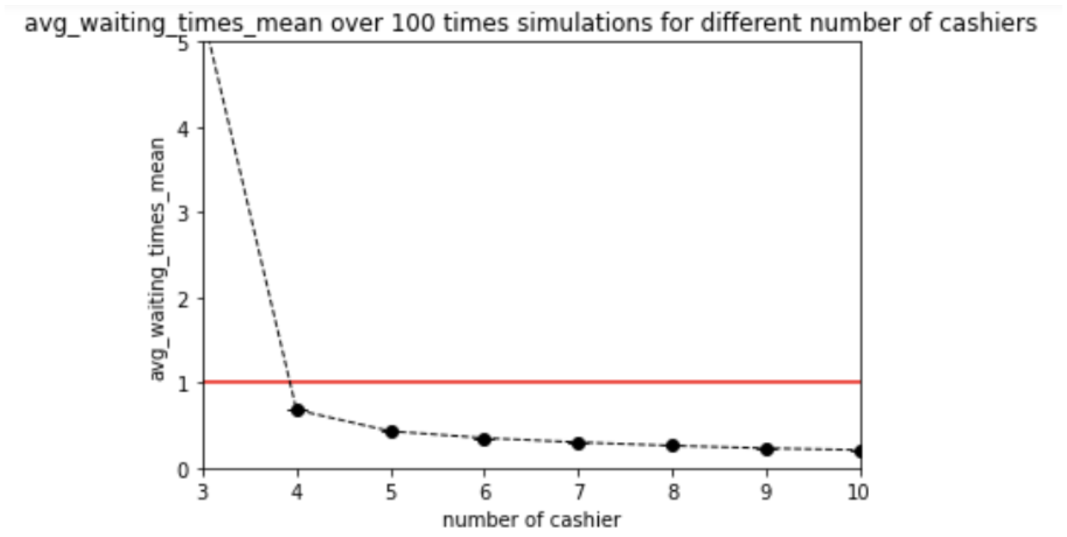


Figure E1: Average waiting time for different number of cashier

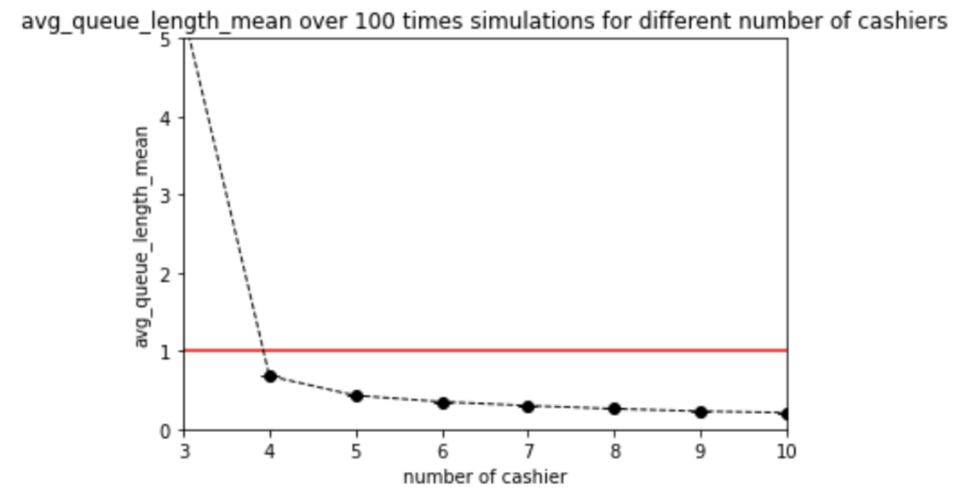


Figure E2: Average waiting time for different number of cashier

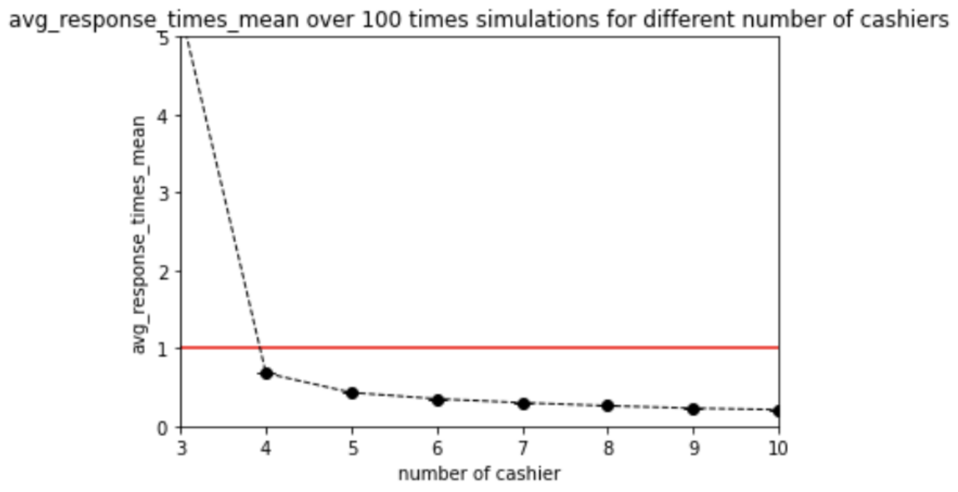


Figure E3: Average response time for different numbers of cashiers

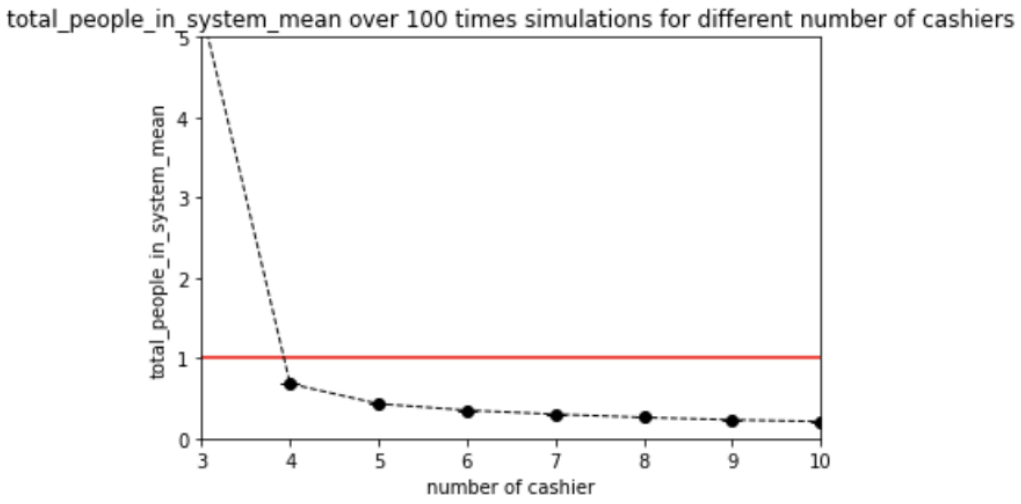
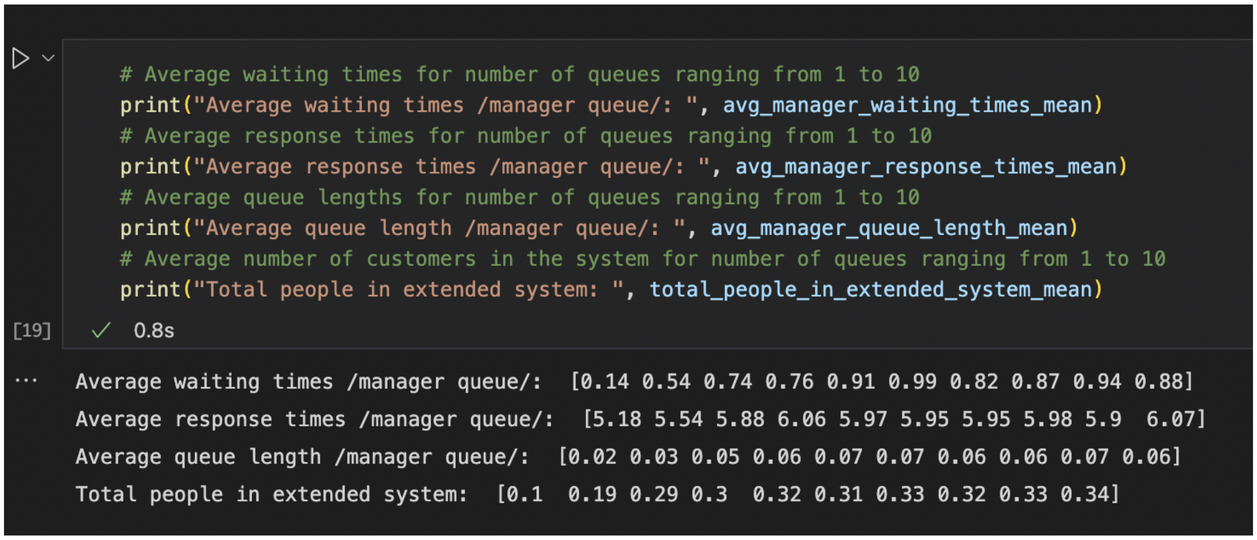


Figure E4: total people in the system for different numbers of cashier

Appendix F



```
# Average waiting times for number of queues ranging from 1 to 10
print("Average waiting times /manager queue/: ", avg_manager_waiting_times_mean)
# Average response times for number of queues ranging from 1 to 10
print("Average response times /manager queue/: ", avg_manager_response_times_mean)
# Average queue lengths for number of queues ranging from 1 to 10
print("Average queue length /manager queue/: ", avg_manager_queue_length_mean)
# Average number of customers in the system for number of queues ranging from 1 to 10
print("Total people in extended system: ", total_people_in_extended_system_mean)
```

[19] ✓ 0.8s

```
Average waiting times /manager queue/:  [0.14 0.54 0.74 0.76 0.91 0.99 0.82 0.87 0.94 0.88]
Average response times /manager queue/:  [5.18 5.54 5.88 6.06 5.97 5.95 5.95 5.98 5.9  6.07]
Average queue length /manager queue/:  [0.02 0.03 0.05 0.06 0.07 0.07 0.06 0.06 0.07 0.06]
Total people in extended system:  [0.1  0.19 0.29 0.3  0.32 0.31 0.33 0.32 0.33 0.34]
```

The statistics for the extended service /manager/ queue system based on the empirical results