Homework #6

Jonathan Beaubien
Math 381 - Discrete Modeling
University of Washington

February 26th, 2021

Queue Modeling

In this write-up I will simulate a queuing system containing three queues and three servers. My main goal of this paper is to analyze the effects of 'jockeying'.
Jockeying (in a queuing context) is the process of an individual changing which queue they are in aiming to join a shorter queue and receive service earlier.
For simplification, the only time a person will switch queues will be if they are in the longest queue and they can join a queue where there is at least one less person in front of them once they switch.
This is equivalent to when the longest queue is at least two people longer than the shortest queue. The three queues may have differing departure rates while the arrival rate is fixed. Another feature is that new arrivals will join the shortest queue when they arrive.
Because we will use Poisson arrival and Poisson departure, we can use Kendall's notation to say we are analyzing a M/M/3 queuing system.

Simulation

To simulate the queue, we need to first define some some variables.
Suppose that 5 people arrive at this set of queues per hour. This will be modeled by a Poisson process where λ is our arrival rate per hour.
Also suppose that each queue has a different serving rate represented by μ_i where i is the queue number for $1 \leq i \leq 3$ (because there are 3 queues).

$$\lambda = 5, \mu_1 = 2, \mu_2 = 3, \mu_3 = 5$$

To model these queues we must break time into small intervals. Specifically, we need these intervals to be small enough to ignore the probability that more than one person enters the queue during that interval.
For this simulation, we will use $m = 1000$ intervals for each hour of simulation.

$$1000\frac{\text{iterations}}{\text{hour}} = 3.6\frac{\text{sec}}{\text{iterations}}$$

The probability of a person arriving to the set of queues is a Poisson process and is therefore:

$$e^{-\frac{5}{1000}}\frac{0.005}{1!} \approx 0.004975$$

This is close enough to $\frac{\lambda}{m} = 0.005$ so I will use this for modeling.
The probability of more than one person arriving is so small that we can ignore it:

$$1 - (e^{-\frac{5}{1000}} \frac{0.005^0}{0!} + e^{-\frac{5}{1000}} \frac{0.005^1}{1!}) \approx 0.0000124$$

For the same reason, we can approximate each queue's departure rate with

$$\frac{\mu_i}{m}$$

Running the Simulation

The first run of the simulation turned jockeying off in order to isolate its effects.
These graphs include 4 runs of the queue for a continuous 1000 hours. The line tracks the cumulative average up to that point. Therefore, the most accurate estimates of the true average length of each queue will get more accurate and converge the more hours we run the simulation. The y-axis is average length (persons) and the x-axis is hours of simulation.

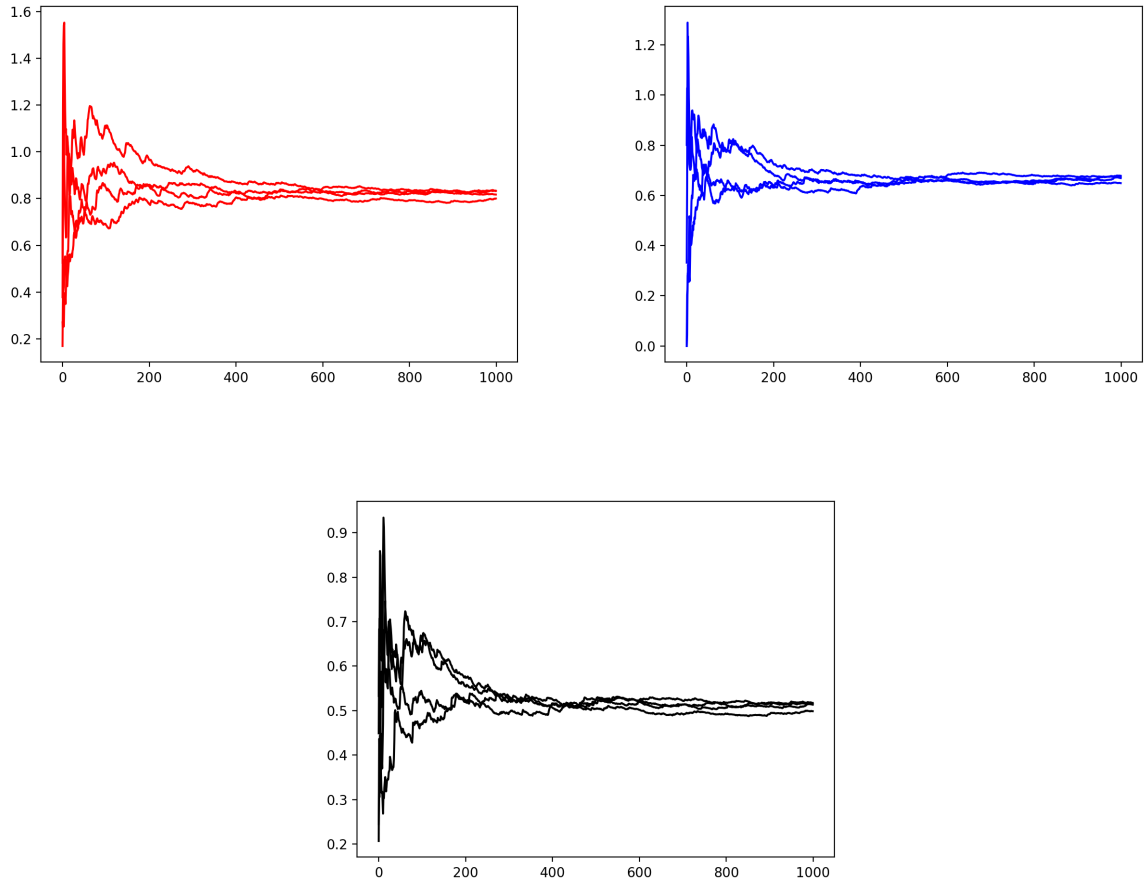


Figure 1: [Jockeying Disabled] Queues 1 (red), 2 (blue), and 3 (black) Average Lengths

As we can see, all three queues converge to a certain length thanks to the Law of Large Numbers.

Theorem 1. The law of large numbers establishes that the average of the results obtained from a large number of trials should be close to the expected value and will tend to become closer to the expected value as more trials are performed.

I decided to run 1000 trials of 100 hours of simulation and averaged each queue's length to showcase the Central Limit Theorem.

Theorem 2. The central limit theorem establishes that, in many situations, when independent random variables are added, their properly normalized sum tends toward a normal distribution even if the original variables themselves are not normally distributed.

The x-axis is the average queue length for a 100 hour simulation while the y-axis is the frequency that it occurred (there is a total of 1000 trials). I used 30 bins for each histogram.

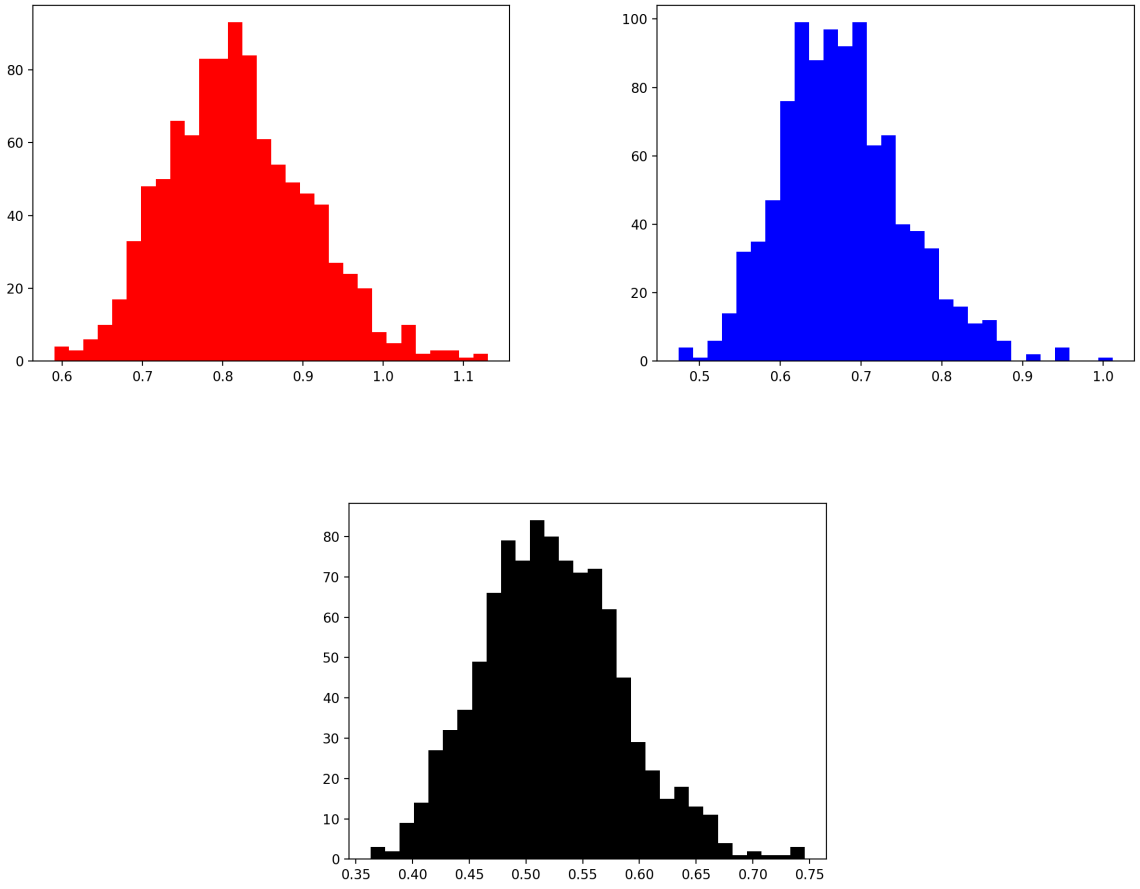


Figure 2: [Jockeying Disabled] Queues 1 (red), 2 (blue), and 3 (black) Length Histogram

At this point, I could do a calculation of the mean and standard deviation to find a confidence interval for each queue's average length but I decided to take another approach. This is mainly because the histogram has a bigger standard deviation than I expected.
To return to the first data set (4 trials of 1000 hours), I will propose a different way of calculating a confidence interval for each queue.
Using the central limit theorem again, we can assume that these 4 trials are normally distributed and can calculate the probability that ALL of the trial averages (the average at time = 1000 hours) are on one side of the true average. This is calculated as so,

$$\frac{2}{2^4} = \frac{1}{2^3} = 0.125$$

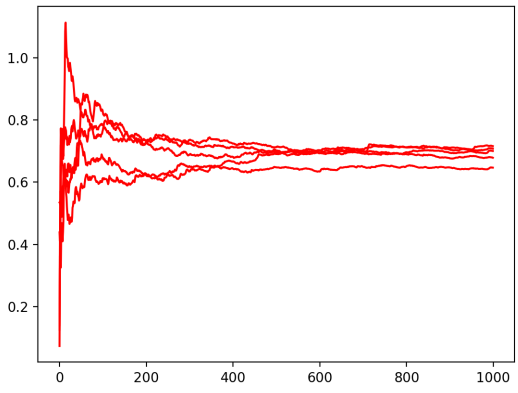
Therefore, we can obtain an 87.5% confidence interval that the true average is between the maximum trial average and the minimum trial average. These results are summarized in this table. I have rounded the average queue lengths to the nearest 5 decimal places.

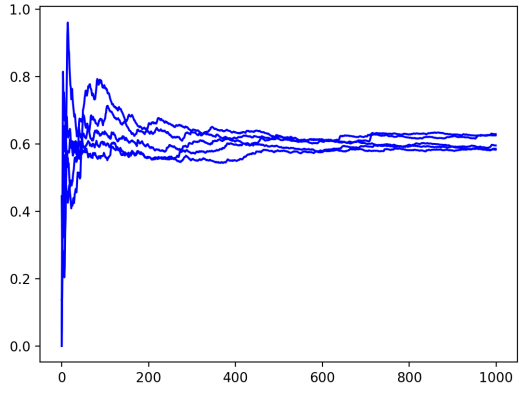
Trial	Queue 1	Queue 2	Queue 3
1	0.83338	0.66886	0.51437
2	0.81717	0.67809	0.51348
3	0.83246	0.67545	0.51726
4	0.80000	0.64886	0.49873
Conf. Interval	[0.80000, 0.83338]	[0.64886, 0.67809]	[0.49873, 0.51726]

Table 1: [Jockeying Disabled] Trial Confidence Intervals (85%)

These will prove useful for comparing to the simulation with jockeying enabled. Here is the same analysis but with jockeying enabled.

Here are the 4 trials of 1000 hour simulations where the y-axis is average queue length and the x-axis is the hour of simulation.





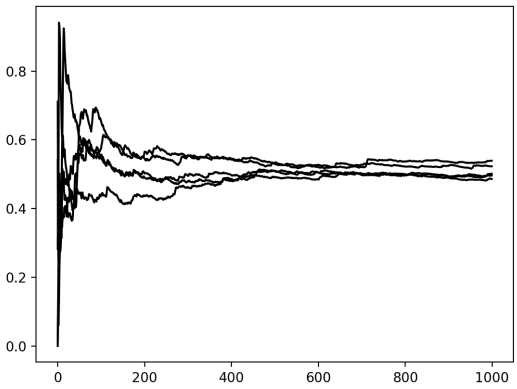


Figure 3: [Jockeying Enabled] Queues 1 (red), 2 (blue), and 3 (black) Average Lengths

And the histograms of 1000 trials of 100 hour simulations:

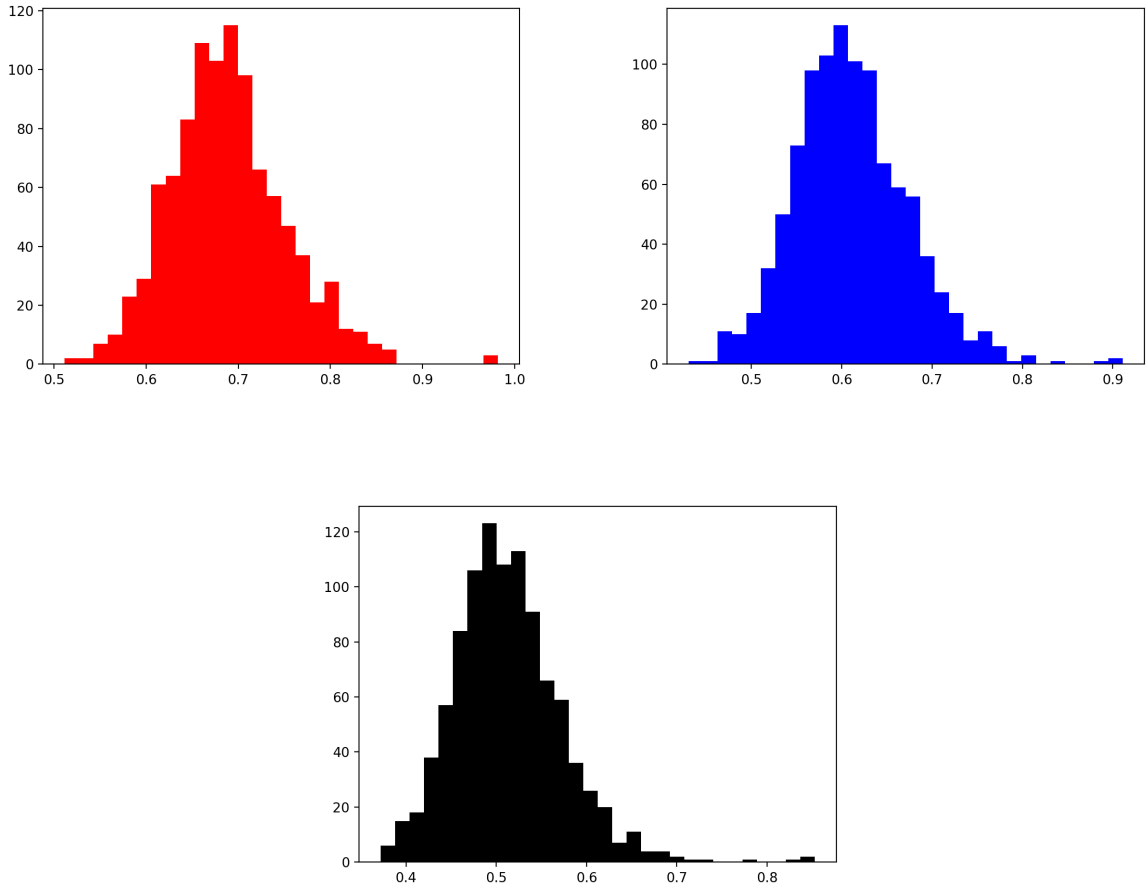


Figure 4: [Jockeying Enabled] Queues 1 (red), 2 (blue), and 3 (black) Length Histogram

And the confidence intervals for the 4 trials of 1000 hour simulations:

Trial	Queue 1	Queue 2	Queue 3
1	0.71586	0.62560	0.52313
2	0.70798	0.62923	0.53924
3	0.67859	0.58555	0.49639
4	0.70030	0.59551	0.50148
Conf. Interval	[0.67859, 0.71586]	[0.58555, 0.62923]	[0.49639, 0.53924]

Table 2: [Jockeying Enabled] Trial Confidence Intervals (85%)

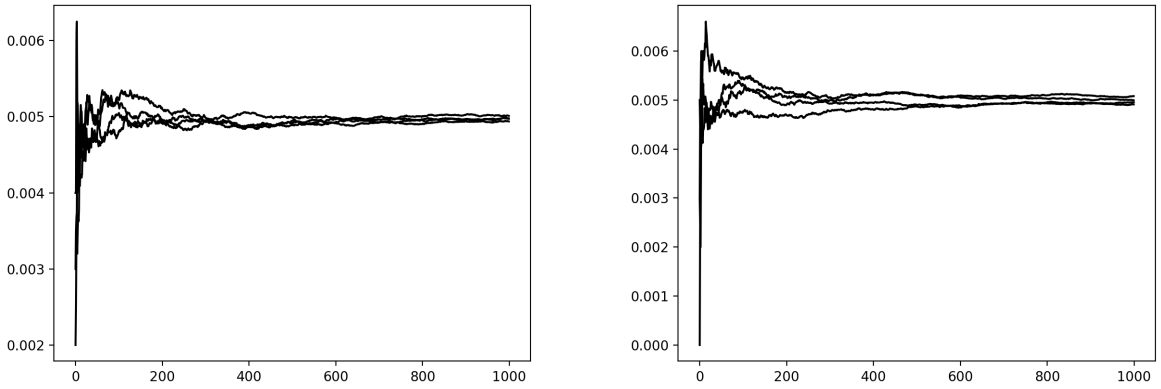
Now that we have our confidence intervals, let us look at their comparison in this table,

Jockeying	Queue 1	Queue 2	Queue 3
Disabled	[0.80000, 0.83338]	[0.64886, 0.67809]	[0.49873, 0.51726]
Enabled	[0.67859, 0.71586]	[0.58555, 0.62923]	[0.49639, 0.53924]

Table 3: Comparing Confidence Intervals

As we can see, with the exception of queue 3, the queues have shorter lengths on average when jockeying is enabled. This is most pronounced in queue 1 because it has the lowest departure rate (μ_1). Individuals who arrive in the slowest queue are more likely to switch queues to the fast ones.

I decided to graph the total departure rate for the entire system in order to see if jockeying makes a difference.



The graph on the left is jockeying disabled while the right is jockeying enabled.
The x axis is hours of simulation while the y-axis is (accidentally) the departure rate for the combined queue system per iteration (oops, should be per hour) but multiplying by $m = 1000$ gives us an average of 5. There is no noticeable difference in the efficiency of the queue. However, in hindsight, this is definitely due to the small value of λ that is limiting the departure rate to 5 people per hour.

Further Inquiry

I believe there are some interesting further questions that could be investigated. The limiting factor was the arrival rate as the queues easily handled the total amount of people that arrived. Increasing the arrival rate λ and modifying the jockeying rules (e.g. only some people change queues or there must be a greater difference in queue length than 2) may prove interesting experiments.

Code

This code created a simulation of h hours, with m intervals, l arrival rate, u_i departure rates for each queue, and a boolean: true if you want to enable jockeying and false otherwise.

```
import math
import random
import matplotlib.pyplot as plt
import numpy as np
import json

def createQueueSimulationList(hours, m, l, u, jockeying):
  iteration_u = (u[0] / m, u[1] / m, u[2] / m)
  queue = [0, 0, 0]
  queue_history = []
  queue_productivity = []
  for iteration in range(0, hours * m):
    if random.random() < l / m:
      if queue[0] == queue[1] == queue[2]:
        queue[random.randint(0, 2)] += 1
      elif queue[0] == queue[1] and queue[2] > queue[0]:
        queue[random.randint(0, 1)] += 1
      elif queue[1] == queue[2] and queue[0] > queue[1]:
        queue[random.randint(1, 2)] += 1
      elif queue[0] == queue[2] and queue[1] > queue[0]:
        queue[0 if random.randint(0, 1) else 2] += 1
      else:
        queue[min(range(len(queue)), key=queue.__getitem__)] += 1

      # add another person to the shortest queue

    if jockeying:
      min_index = min(range(len(queue)), key=queue.__getitem__)
      max_index = max(range(len(queue)), key=queue.__getitem__)
      min_length = queue[min_index]
      max_length = queue[max_index]
      if max_length - min_length >= 2:
        options = [0, 1, 2]
        options.remove(min_index)
        options.remove(max_index)
        if queue[options[0]] == max_length:
```

```
            queue[min_index] += 1
            queue[options[0] if random.randint(0, 1) else max_index]
                -= 1
          elif queue[options[0]] == min_length:
            queue[options[0] if random.randint(0, 1) else min_index]
                += 1
            queue[max_index] -= 1
          else:
            queue[min_index] += 1
            queue[max_index] -= 1


      productivity = [0, 0, 0]
      for i in range(3):
        if random.random() < iteration_u[i] and queue[i] >= 1:
          productivity[i] += 1
          queue[i] -= 1

      # print(queue)
      queue_productivity.append(list(productivity))
      queue_history.append(list(queue))
    return (queue_history, queue_productivity)
```

While that simulates the queue, we need some functions to take averages. I used these two function to do cumulative averages for each hour as well as a cumulative average for an entire simulation without bothering with calculating any 'sub-averages'

```
def averageQueueLengths(queueHist, hours, intervals):
  hourly_averages = []
  for h in range(hours):
    hourly_average = [0, 0, 0]
    for i in range(0, (h + 1) * intervals):
      for j in range(3):
        hourly_average[j] += queueHist[i][j] / ((h+1) * intervals)
    hourly_averages.append(list(hourly_average))
  return hourly_averages

def averageOfRun(queueHist, hours, intervals):
  run_average = [0, 0, 0]
  for i in range(0, hours * intervals):
      for j in range(3):
        run_average[j] += queueHist[i][j] / (hours * intervals)
  return run_average
```

I have not included any of the visualization code but I have use matplotlib, numpy, and json (for storing simulations). As you might guess, the visualization code is as messy as my simulation code so you probably don't want to see it.