APPM 4560 Lab Three
Simulating A Single Server Queue

Zoe Farmer

December 9, 2016

The goal of this lab is to understand Markovian Queues.

Let $X = (X_t)_{t\geq 0}$ denote the number of customers in an M/M/1-queue with arrival rate $\lambda > 0$ and service rate $\mu > 0$. In addition, let $T > 0$ be a fixed time.

1. What condition is necessary and sufficient for the queue to have a stationary distribution? What is it?

   In order for the queue to have a stationary distribution, λ must be less than μ. To put this in simpler terms, the rate of incoming events must be lower than the rate of outgoing, or else the system will infinitely increase. We can define a new parameter $\rho = \lambda/\mu$. This value must be less than 1 in order for there to be a stationary distribution. Following the example in the book on page 160, we can find the stationary distribution to be the following.

   $$\pi(n) = \left(1 - \frac{\lambda}{\mu}\right)\left(\frac{\lambda}{\mu}\right)^n \qquad \text{for } n \geq 0$$

2. When X is stationary what's the distribution of X_T?

   If we assume that X is stationary, this means that $\rho < 1$, which means that $\lambda < \mu$. If this is true, then the system will "fall" to zero, and the system after duration T will be geometrically distributed with parameter $1 - \rho = 1 - \lambda/\mu$.

3. When X is stationary, what fraction of the time is the server busy?

   Using our fancy new parameter again, we note that the fraction of the time that the server is busy is simply the fraction of the rates, or ρ.

4. Write pseudo-code that simulates the queue. Input should be λ, μ, and the number n of people that arrive at time 0. The output must be a list of points in the form (τ, y) where $0 \leq \tau \leq T$ and $y = (+1)$ if at time τ there was an arrival, however $y = (-1)$ if at time τ there was a departure.

We can use the fact about the Poisson Point Process that the spacing of events is exponentially distributed.

1	We are given an arrival rate λ, a departure rate μ, an initial number of people n, and a duration T.
2	Define $c = 0$.
3	Define an empty array $\mathcal{E}$.
4	Simulate an exponentially distributed random variable with rate λ and call it a.
5	Simulate an exponentially distributed random variable with rate μ and call it d.
6	If $a < d$, go to 7.
7	Set $c = c + a$, add $(c, +1)$ to $\mathcal{E}$, and set $n = n + 1$. Go to 10.
8	Else if $a > d$, if $n \neq 0$ then go to 9, else go to 7.
9	Set $c = c + d$, add $(c, -1)$ to $\mathcal{E}$, and set $n = n - 1$. Go to 10.
10	If $c < T$, go to 4, otherwise stop and remove the last element from $\mathcal{E}$.

Table 1: Single Server Queue Simulation

5. Based on the pseudo-code, how can you determine X_T?

   Given the initial number of people in the Queue we can simply add each -1 or $+1$ to that number resulting in our final amount at the end.

6. Based on the pseudo-code how can you determine the fraction of time the server was busy between time 0 and T?

   If we assume that a non-busy server is one with zero "jobs" in the queue, then we can look at the duration of time where there were zero jobs and compare that to our total time T.

7. Based on the pseudo-code how can you determine the inter-departure times from the queue (if any) between time 0 and T?

   Any time we have a -1 as a result of the algorithm we can assume that it was a departure (by definition). Therefore these times are just the times that -1 values appear.

   Implement your code. Let $(\lambda, \mu, T) = (1, 2, 50)$ and n according to the stationary distribution.

   We can find our stationary distribution now that we have actual values.

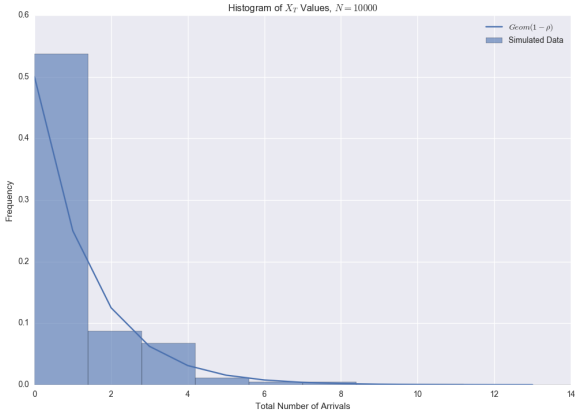
n	$\pi(n)$
0	0.5
1	0.25
2	0.125
3	0.0625
4	0.03125
5	0.015625
6	0.0078125
7	0.00390625
8	0.001953125
9	0.0009765625

Table 2: Stationary Distribution, $\pi(n)$

This is just a Geometric Distribution (starting at 0). See Appendix A for code.

8. Verify your answer in part (2) via simulations.

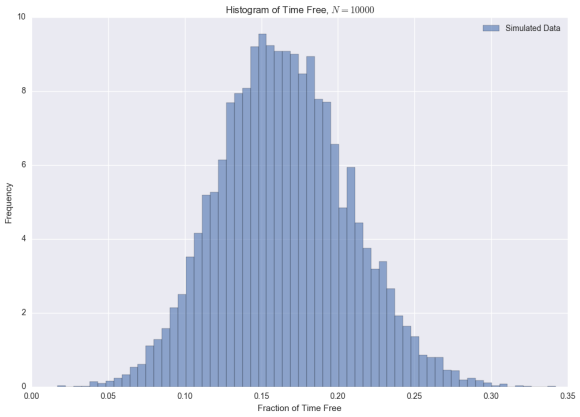
   We can plot the histogram.



   This looks exactly as we imagined it would.

9. Verify your answer in part (3).

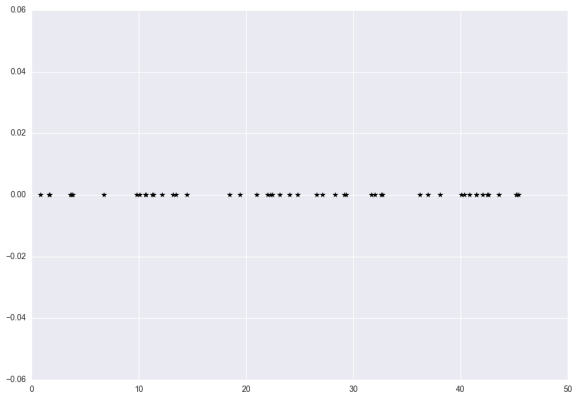
   We can create another histogram.



I was completely wrong about the distribution here, I had imagined it would be simply a ratio of our λ to our μ, but these results are completely different. Looking up the results online, it turns out that this is a modified Bessel function of the first kind.[1]

10. According to Theorem 4.10, if X is in equilibrium then the output process of the queue is a homogeneous Poisson (point) process with intensity λ. Verify by simulating the inter-departure times from the queue.

    We can draw a similar graph as in the last lab to show the Poisson Point Process.



    This looks exactly like a Poisson Process!

[1] `https://en.wikipedia.org/wiki/M/M/1_queue#Busy_period_of_server`

A Code

```
#!/usr/bin/env python3.5

import sys
import matplotlib.pyplot as plt
import numpy as np
import scipy.stats as sc_st
from tqdm import tqdm
import seaborn
import concurrent.futures


def main():
    l = 1
    m = 2
    T = 50

    print('---')
    print('n & pi(n)\\')
    rho = l / m
    for i in range(10):
        print('{} & {}\\'.format(i, (1 - rho) * (rho**i)))
    print('---')

    size = 10000
    initial = sc_st.geom(1 - rho, loc=-1)
    nvals = initial.rvs(size=size)

    events, history = MM1_queue(l, m, nvals[0], T)
    filtered = [t for t, s, in events if s < 0]
    plt.figure(figsize=(12, 8))
    plt.plot(filtered, np.zeros(len(filtered)), 'k*')
    plt.savefig('poissonprocess.png')

    points = np.zeros(size)
    time_free = np.zeros(size)
    for i in tqdm(range(size)):
        events, history = MM1_queue(l, m, nvals[i], T)
        points[i] = history[-1]
        time_free[i] = frac_busy(events, history, T)

    hist, edges = np.histogram(points, bins=10, density=True)
    width = edges[1] - edges[0]
    rv = initial
    x = np.arange(0, edges[-1])
    plt.figure(figsize=(12, 8))
    plt.bar(edges[:-1], hist, width,
```

```
            label='Simulated Data',
            alpha=0.6)
    plt.plot(x, rv.pmf(x), label=r'$Geom(1 - \rho)$')
    plt.legend(loc=0)
    plt.xlabel('Total Number of Arrivals')
    plt.ylabel('Frequency')
    plt.title(r'Histogram of $X_T$ Values, $N={}$'.format(len(points)))
    plt.savefig('xt_dist.png')

    hist, edges = np.histogram(time_free, bins='auto', density=True)
    width = edges[1] - edges[0]
    plt.figure(figsize=(12, 8))
    plt.bar(edges[:-1], hist, width,
            label='Simulated Data',
            alpha=0.6)
    plt.legend(loc=0)
    plt.xlabel('Fraction of Time Free')
    plt.ylabel('Frequency')
    plt.title(r'Histogram of Time Free, $N={}$'.format(len(time_free)))
    plt.savefig('time_free.png')



def frac_busy(events, history, T):
    free = False
    time_free = 0
    for i, z in enumerate(zip(events, history)):
        event, status = z
        if status == 0:
            if free:
                time_free += event[0] - events[i - 1][0]
            free = True
        else:
            if free:
                time_free += event[0] - events[i - 1][0]
            free = False
    return time_free / T


def MM1_queue(arrival_rate, service_rate, initial_num, time_length):
    """
    arrival_rate => \lambda
    service_rate => \mu
    initial_num  => n
    time_length  => T

    https://en.wikipedia.org/wiki/M/M/1_queue
    """
```

```
    arrival = sc_st.expon(scale=1 / arrival_rate)
    service = sc_st.expon(scale=1 / service_rate)

    ctime = 0

    events = []
    history = [initial_num]
    while ctime < time_length:
        new_arrival = arrival.rvs()

        # Only let new services if there's something in the queue
        if history[-1] == 0:
            new_service = 1000
        else:
            new_service = service.rvs()

        if new_arrival < new_service:
            ctime += new_arrival
            events.append((ctime, 1))
            history.append(history[-1] + 1)
        else:
            ctime += new_service
            events.append((ctime, -1))
            history.append(history[-1] - 1)

    return events[:-1], history[:-1]



if __name__ == '__main__':
    sys.exit(main())
```