# Queueing theory assignment: Simulation in discrete time

EBB074A05

Nicky D. van Foreest

2022:01:19

## 1 Deterministic queues

Here we consider a simple queueing system in which all is deterministic.

### 1.1 one period, demand, service capacity, and queue

There is one server, jobs enter a queue in front of the server, and the server serves batches of customers, every hour say.

Python Code

```
L = 10
a = 5
d = 8
L = L + a -d
print(L)
```

Python Code

```
L = 3
a = 5
c = 7
d = min(c, L)
L += a -d
print(d, L)
```

### 1.2 two periods

Python Code

```
L = 3
a = 5
c = 7
d = min(c, L)
L += a - d

a = 6
d = min(c, L)
L += a - d
print(d, L)
```

**Ex 1.1.** Add a third period, and report your result in the assignment.

### 1.3 many periods, use vectors and for loops

Python Code

```
num = 5

a = 9*np.ones(num)
c = 10*np.ones(num)
L = np.zeros_like(a)

L[0] = 20
for i in range(1, num):
    d = min(c[i], L[i-1])
    L[i] = L[i-1] + a[i] - d

print(L)
```

**Ex 1.2.** Run the code for 10 periods and report your result.

**Ex 1.3.** What will be the queue after 100 or so periods?

## 2 Stochastic/random behavior

Real queueing systems show lots of stochasticity: in the number of jobs that arrive in a period, and the time it takes to serve these jobs. In this section we extend the previous models so that we can deal with randomness.

### 2.1 simulate demand

Python Code

```
import numpy as np

a = np.random.randint(5, 8, size=5)
print(a)
```

### 2.2 Set seed

Python Code

```
import numpy as np

np.random.seed(3)

a = np.random.randint(5, 8, size=5)
print(a)
```

**Ex 2.1.** Why do we set the seed?

### 2.3 Compute mean and std of simulated queue length for $\rho \approx 1$

We discuss the concept of load more formally in the course at a later point in time. Conceptually the load is the rate at which work arrives. For instance, if $\lambda = 5$ jobs arrive per hour, and each requires 20 minutes of work (on average), then we say that the load is $5 \times 20/60 = 5/3$. Since one server can do only

1 hour of work per hour, we need at least two servers to deal with this load. We define the utilization $\rho$ as the load divided by the number of servers present.

In discrete time, we define $\rho$ as the average number of jobs arriving per period divided by the average number of jobs we can serve per period. Slightly more formally, for discrete time,

$$\rho \approx \frac{\sum_{k=1}^{n} a_k}{\sum_{k=1}^{c_k}}. \tag{1}$$

And formally, we should take the limit $n \to \infty$ (but such limits are a bit hard to obtain in simulation).

Python Code

```
import numpy as np

np.random.seed(3)
num = 5000

a = np.random.randint(5, 10, size=num)
c = 7 * np.ones(num)
L = np.zeros_like(a)

L[0] = 20
for i in range(1, num):
    d = min(c[i], L[i-1])
    L[i] = L[i-1] + a[i] - d

print(a.mean(), c.mean(), L.mean(), L.std())
```

**Ex 2.2.** Read the numpy documentation on `numpy.random.randint` to explain the range of random numbers that are given by this function. In view of that, what should `a.mean()` approximately be? Is that larger, equal or smaller than `c`?

## 2.4 plot the queue length process

Glue the next code after the other code.

Python Code

```
import matplotlib.pyplot as plt

plt.clf()
plt.plot(L)
plt.savefig('queue-discrete_1.pdf')
```

**Ex 2.3.** Comment on the graph you get. Did you expect such large excursions of the queue lenght process?

## 2.5 A trap: integers versus floats

Suppose that we change the arrival rate a bit.

Python Code

```
num = 5000

np.random.seed(3)
a = np.random.randint(5, 9, size=num)
```

```
c = (5+8)/2 * np.ones(num)
L = np.zeros_like(a)

L[0] = 20
for i in range(1, num):
    d = min(c[i], L[i-1])
    L[i] = L[i-1] + a[i] - d


plt.clf()
plt.plot(L)
plt.savefig('queue-discrete-1-1.pdf')
```

Don't forget that the `numpy` library has to be imported (as we did before).

**Ex 2.4.** Compare the definition of `a` and `c` to what we had earlier. What is `a.mean()` now approximately? Observe that the mean of `c` is now around 6.5.

Here is the figure.

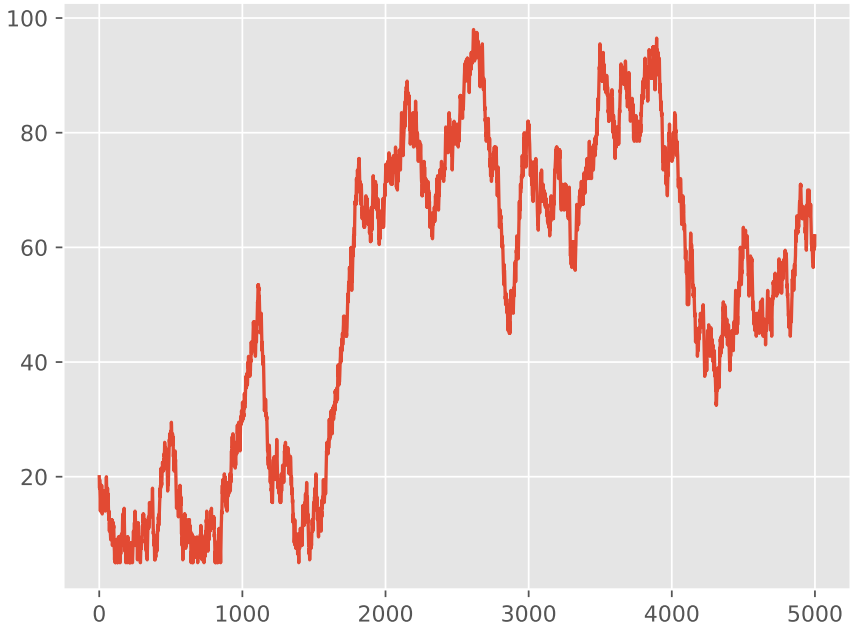
You should see that is is very different from the one before. To explain why this is so, this somehwat cryptic question will be helpful, hopefully.

**Ex 2.5.** What is $9-6.5$? What is `int(9-6.5)`, that is, run the code in python, and type in the answer in your answer sheet. What is the difference between $9-6.5$ and `int(9-6.5)`? Explain that since `L` stores *integers* we actually use a service capacity of 7, *not* of 6.5.

**Ex 2.6.** The code below repairs this. Explain which line is different of the previous code. How did that change repair the problem? Now explain the title of this section. (BTW, I made this type of error many, many times. The reason to include these exercises is to make you aware of the problem, so that you can spot it when you make the same error.)

Python Code

```
num = 5000

np.random.seed(3)
a = np.random.randint(5, 9, size=num)
c = (5+8)/2 * np.ones(num)
L = np.zeros_like(a, dtype=float)

L[0] = 20
for i in range(1, num):
    d = min(c[i], L[i-1])
    L[i] = L[i-1] + a[i] - d


plt.clf()
plt.plot(L)
plt.savefig('queue-discrete-1-2.pdf')
```




## 2.6 Serve arrivals in the same period as they arrive

**Ex 2.7.** Change the code such that the arrivals that occur in period $i$ can also be served in period $i$. Include your code in your assignment. Then make a graph (include that too of course) and compare your results with the results of the simulation we do here (recall, in the simulations up to now, arrivals cannot be served in the periods in which they arrive).

## 2.7 Drift when $\rho > 1$

Python Code

```
num = 5_000

np.random.seed(3)
a = np.random.randint(5, 9, size=num)
c = (5+8)/2.3 * np.ones(num)
L = np.zeros_like(a, dtype=float)

L[0] = 20
for i in range(1, num):
    d = min(c[i], L[i-1])
    L[i] = L[i-1] + a[i] - d


plt.clf()
plt.plot(L)
plt.savefig('queue-discrete_2.pdf')
```

**Ex 2.8.** What is `c.mean()-a.mean()`? Explain the slope of the line (fitted through the points.)

## 2.8 Drift when $\rho < 1$, start with large queue.

Python Code

```
num = 5_000

np.random.seed(3)
a = np.random.randint(5, 9, size=num)
c = (5+8)/1.8 * np.ones(num)
L = np.zeros_like(a, dtype=float)

L[0] = 2_000
for i in range(1, num):
    d = min(c[i], L[i-1])
    L[i] = L[i-1] + a[i] - d


plt.clf()
plt.plot(L)
plt.savefig('queue-discrete_3.pdf')
```

**Ex 2.9.** Explain the slope of the lines (if you would fit that through the points.)

## 2.9 Hitting time

**Ex 2.10.** When $\rho < 1$ and $L_0$ is some large number, such as here, why is the normal distribution reasonable to model the time $\tau$ until the queue hits zero for the first time?

**Ex 2.11.** The code below uses simulation to estimate the expected time $\mathsf{E}[\tau]$ and variance $\mathsf{V}[\tau]$. Explain how it works; don't forget to address the following points:

1. Why don't I need to use =d = min(L, c)=?
2. Explain that running up to time 5000 is just a guess.

Python Code

```
import numpy as np

num = 5_000


def hitting_time(seed):
    np.random.seed(seed)
    a = np.random.randint(5, 9, size=num)
    c = (5 + 8) / 1.8 * np.ones(num)

    L = 2_000
    for i in range(1, num):
        L += +a[i] - c[i]
        if L <= 0:
            return i


```

```
N = 10
tau = np.zeros(N)
for n in range(N):
    tau[n] = hitting_time(n)

print(tau.mean(), tau.std())
```

**Ex 2.12.** Run the above simulation with some seeds to your liking. What are the numbers you get? Use properties of uniform random variables to explain the values you get.

### 2.10 Things to memorize

1. If the capacity is equal or less than the arrival rate, the queue length will explode.
2. If the capacity is larger than the arrival rate, the queue length will stay around 0 (between quotes).
3. If we start with a huge queue, but the service capacity is larger than the arrival rate, then the queue will drain rather fast, in fact, about linear.

## 3 Queues with blocking

Consider a queue that is subject to blocking: this means that when the queue exceeds $K$, say, then the excess jobs are rejected.

### 3.1 A simple rejection rule

Python Code

```
num = 500

np.random.seed(3)
a = np.random.randint(0, 20, size=num)
c = 10*np.ones(num)
L = np.zeros_like(a)
loss = np.zeros_like(a)

K = 30 # max people in queue, otherwise they leave

L[0] = 28
for i in range(1, num):
    d = min(c[i], L[i-1])
    loss[i] = max(L[i-1] + a[i] - d - K, 0)  # this
    L[i] = L[i-1] + a[i] - d - loss[i] # this 2

lost_fraction = sum(loss)/sum(a)
print(lost_fraction)
```

**Ex 3.1.** Explain how the line marked as `this` works, in other words, how does that line implement a queue with loss? In line `this 2` we subtract `loss[i]`; why?

**Ex 3.2.** Why is `dtype=float` not necessary in the definition of `L`?

Here is the graph of the (simulated) queue length process.

Python Code

```
plt.clf()
plt.plot(L)
plt.savefig('queue-discrete-loss.pdf')
```

**Ex 3.3.** Does the blocking rule work as it should?

## 3.2 Rejection at the start of the period

If we would assume that rejection occurs as the start of the period, the code has to be as follows:

Python Code

```
for i in range(1, num):
    d = min(c[i], L[i-1])
    loss[i] = max(L[i-1] + a[i] - K, 0)
    L[i] = L[i-1] + a[i] - d - loss[i]

lost_fraction = sum(loss)/sum(a)
print(lost_fraction)
```

**Ex 3.4.** Explain the line in the code that changed.

**Ex 3.5.** Explain that this rule has the same effect as assuming that departures occur after the rejection.

## 3.3 Estimating rejection probabilities

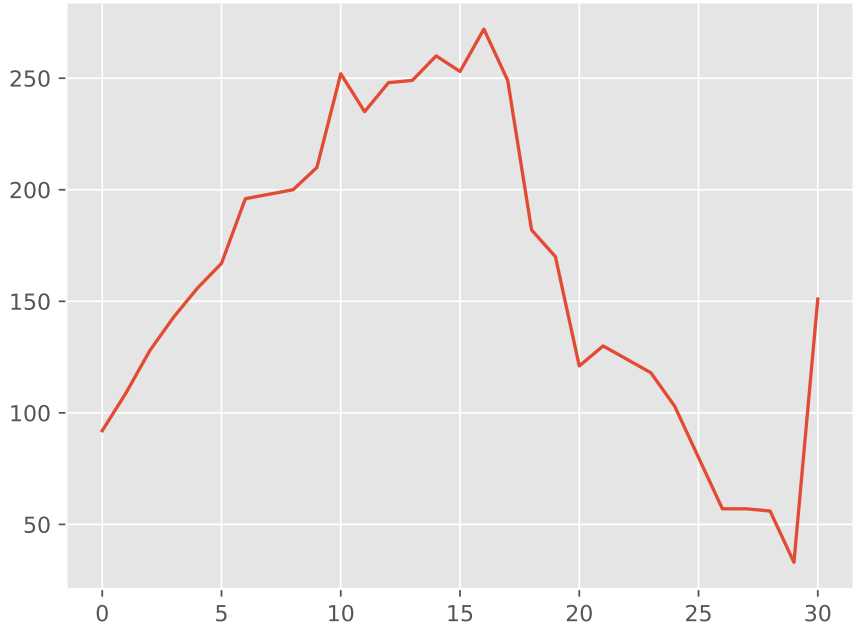
With the code below we can estimate the distribution $p_i = \mathsf{P}[L = i]$.

Python Code

```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(3)

num = 5000

np.random.seed(3)
a = np.random.randint(0, 18, size=num)
c = 10 * np.ones(num)
L = np.zeros_like(a)

K = 30
p = np.zeros(K + 1)

L[0] = 28
for i in range(1, num):
    d = min(c[i], L[i - 1])
    L[i] = min(L[i - 1] + a[i] - d, K)
    p[L[i]] += 1

plt.clf()
plt.plot(p)
plt.savefig('queue-discrete-loss-p.pdf')
```

**Ex 3.6.** Why should p have a length of K+1? Then explain how the code estimates $p_i$.

Here is the graph of p.




**Ex 3.7.** Note that p is not normalized (i.e., sums up to 1). The following code repairs that. Explain how it works.

Python Code

```
p /= p.sum()
```

### 3.4 Rejection probabilities under high loads

**Ex 3.8.** Change the parameters such $\rho = 1.51$. Then make again a plot of $p$. (Just copy my code, but change the parameters or the distribution of a, or c.)

**Ex 3.9.** Now change the parameters such $\rho \approx 10$, use a simple argument to show that $\pi_{K-2} \approx 0$. Check this intuintuion by doing a simulation with the appropriate parameters. Include your code and the graph of $p$.

## 4 Hints

**h.2.10.** When $L_0 \gg 1$, then $a_k$ jobs arrive in period $k$ and $c_k$ jobs leave. For ease, write $h_k = c_k - a_k$. Then the time $\tau$ to hit 0, i.e., the time until the queue is empty, is the smallest $\tau$ such that $\sum_{k=1}^{\tau} h_k \geq L_0$. But then, by Wald's theorem: $\mathsf{E}[\tau]\,\mathsf{E}[h_k] = L_0$. What is $\mathsf{E}[\tau]$?

Moreover, consider any sum of random variables, then with $\mu = \mathsf{E}[X]$, $\sigma$ the std of $X$, and $N$ the normal distribution, and by the central limit theorem,

$$\frac{1}{n}\sum_{i=1}^{n} X_i \sim N(\mu, \frac{\sigma^2}{n}) \implies \sum_{i=1}^{n} X_i \sim N(n\mu, n\sigma^2). \tag{2}$$