Queueing Theory: Assignment 1, Simulation in discrete time
EBB074A05

Nicky D. van Foreest

2020:11:15

1 General info

This file contains the code and the results that go with these youtube movies:

- `https://youtu.be/DfYxayoQmjYc`

- `https://youtu.be/D8BIAoBICnw`

- `https://youtu.be/_BoagRyH5c0`

There are a number of exercises you have to address in your report. Keep your answers short. You don't have to win the Nobel prize on literature.

2 Simulation in Discrete time

2.1 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.

```
1  L = 10
2  a = 5
3  d = 8
4  L = L + a -d
5  print(L)
```

```
7
```

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

```
3 5
```

2.2 two periods

```
1   L = 3
2   a = 5
3   c = 7
4   d = min(c, L)
5   L += a - d
6
7   a = 6
8   d = min(c, L)
9   L += a - d
10  print(d, L)
```

```
5 6
```

Ex 2.1. Add a third period, and report your result.

2.3 simulate many periods, make vectors

```
1   import numpy as np
2
3   a = np.random.uniform(5, 8, size=5)
4   print(a)
```

```
[7.66616096 5.46469981 5.28343426 7.84771106 5.80769232]
```

2.4 Set seed

```
1   import numpy as np
2
3   np.random.seed(3)
4
5   a = np.random.uniform(5, 8, size=5)
6   print(a)
```

```
[6.65239371 7.12444347 5.87271422 6.53248282 7.67884086]
```

2.5 update with a for loop

```
1   num = 5
2
3   #a = np.random.uniform(5, 8, size=num)
4   #c = np.random.uniform(5, 8, size=num)
5   a = 9*np.ones(num)
6   c = 10*np.ones(num)
7   L = np.zeros_like(a)
8
9   L[0] = 20
10  for i in range(1, num):
11      d = min(c[i], L[i-1])
```

```
12        L[i] = L[i-1] + a[i] - d
13
14  print(L)
```

```
[20. 19. 18. 17. 16.]
```

Ex 2.2. Run the code for 10 periods and report your result.

2.6   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 ρ as the load divided by the number of servers present.

In discrete time, we define ρ 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).

```
1   num = 5_000
2
3   np.random.seed(3)
4   a = np.random.uniform(5, 8, size=num)
5   c = (5+8)/1.99 * np.ones(num)
6   L = np.zeros_like(a)   # queue length at the end of a period
7
8   L[0] = 20
9   for i in range(1, num):
10      d = min(c[i], L[i-1])
11      L[i] = L[i-1] + a[i] - d
12
13  print(L.mean(), L.std())
```
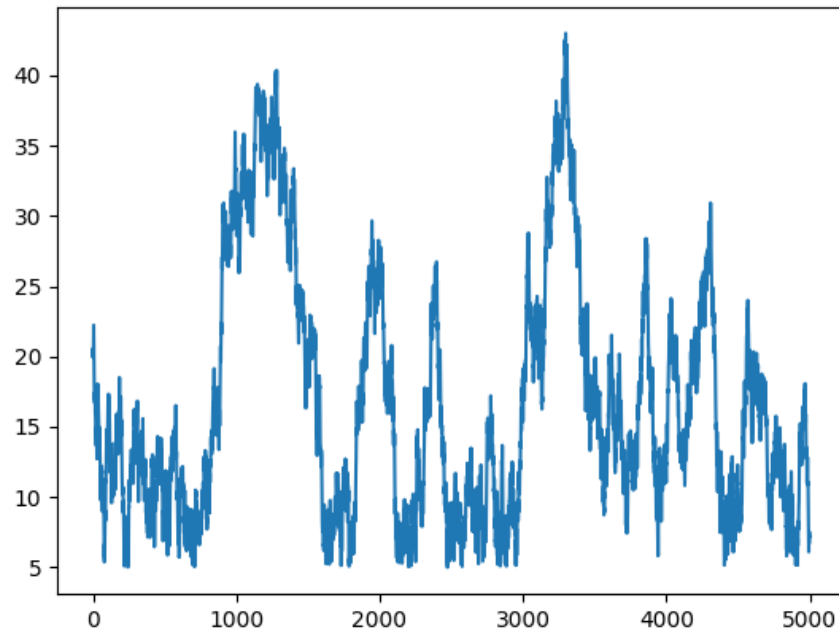
```
16.78550665013682 8.695855000533511
```

2.7   plot the queue length process

```
1   import matplotlib.pyplot as plt
2
3   plt.clf()
4   plt.plot(L)
5   plt.savefig('queue-discrete_1.png')
6   'queue-discrete_1.png'
```

2.8 Compute mean and std of simulated queue length for $\rho = 1/2$

```
num = 5_000

np.random.seed(3)
a = np.random.uniform(5, 8, size=num)
c = (5+8) * np.ones(num)
L = np.zeros_like(a)   # queue length at the end of a period

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.mean(), L.std())
```

6.5051538887388 0.890452952271703

Ex 2.3. Change the code such that the arrivals that occur in period i can also be served in period i. Explain how this works, make a graph and compare your results with the results of the simulation we do here (i.e. arrivals cannot be served in the periods in which they arrive).

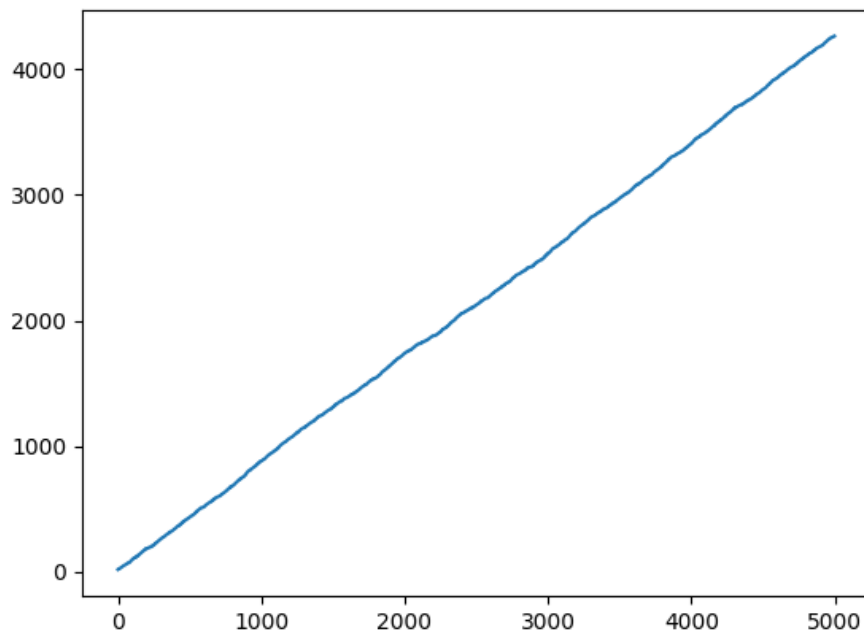## 2.9 show the drift when $\rho > 1$

```
num = 5_000

np.random.seed(3)
```

4

```
4   a = np.random.uniform(5, 8, size=num)
5   c = (5+8)/2.3 * np.ones(num)
6   L = np.zeros_like(a)   # queue length at the end of a period
7
8   L[0] = 20
9   for i in range(1, num):
10      d = min(c[i], L[i-1])
11      L[i] = L[i-1] + a[i] - d
12
13
14  plt.clf()
15  plt.plot(L)
16  plt.savefig('queue-discrete_2.png')
17  'queue-discrete_2.png'
```


2.10 Start with a large queue, take $\rho < 1$, show the drift
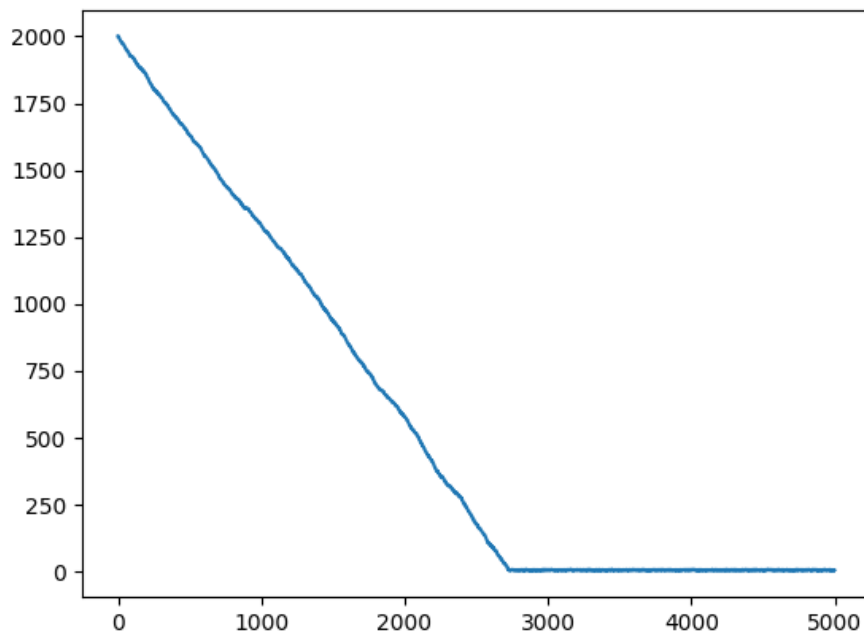
```
1   num = 5_000
2
3   np.random.seed(3)
4   a = np.random.uniform(5, 8, size=num)
5   c = (5+8)/1.8 * np.ones(num)
6   L = np.zeros_like(a)   # queue length at the end of a period
7
8   L[0] = 2_000
9   for i in range(1, num):
10      d = min(c[i], L[i-1])
11      L[i] = L[i-1] + a[i] - d
```

```
12
13
14   plt.clf()
15   plt.plot(L)
16   plt.savefig('queue-discrete_3.png')
17   'queue-discrete_3.png'
```


Things to memorize:

- if the capacity is equal or less than the arrival rate, the queue lenght will explode.

- If the capacity is larger than the arrival rate, the queue length will stay around 0 (between quotes).

- 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.

Ex 2.4. When $\rho < 1$ and L_0 is some large number, such as here. Why is the normal distribution reasonable to model the time τ until the queue hits zero for the first time? Use simulation to estimate the expected time $\mathsf{E}[\tau]$ and variance $\mathsf{V}[\tau]$. (You can also try to use EVE's law here, if you're fanatic; but if you're not, just skip the analytic part.)

Repeat the above simulation a number of times with different seeds (why?) to estimate $\mathsf{E}[\tau]$ and $\mathsf{V}[\tau]$.

Spoiler alert, Hint: When $L_0 \gg 1$, then per period a_k jobs arrive and c_k leave. For ease, write $h_k = c_k - a_k$. Then the time τ to hit 0, i.e., the queue is empty, is such that $\sum_{k=1}^{\tau} h_k \geq L_0$. But then, $\mathsf{E}[\tau]\,\mathsf{E}[h_k] = L_0$. Moreover, consider any sum of random variables, then with $\mu = \mathsf{E}[X]$, σ the std of X, and N the normal distribution, and by the central limit law,

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

2.11 Queues with blocking.

We have a queue subject to blocking. When the queue exceeds K, say, then whatever of the batch of items coming in exceeds K is rejected. This is the so-called complete reject rule. Two more assumptions: service occurs before arrival, and jobs arriving in a period cannot be served.

```python
num = 500

np.random.seed(3)
a = np.random.randint(0, 20, size=num)
c = 10*np.ones(num)
L = np.zeros_like(a)    # queue length at the end of a period
loss = np.zeros_like(a)    # queue length at the end of a period

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) # service before arrivals.
    L[i] = L[i-1] + a[i] - d - loss[i]

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

```
0.026064291920069503
```

```python
plt.clf()
plt.plot(L)
plt.savefig('queue-discrete_loss.png')
'queue-discrete_loss.png'
```
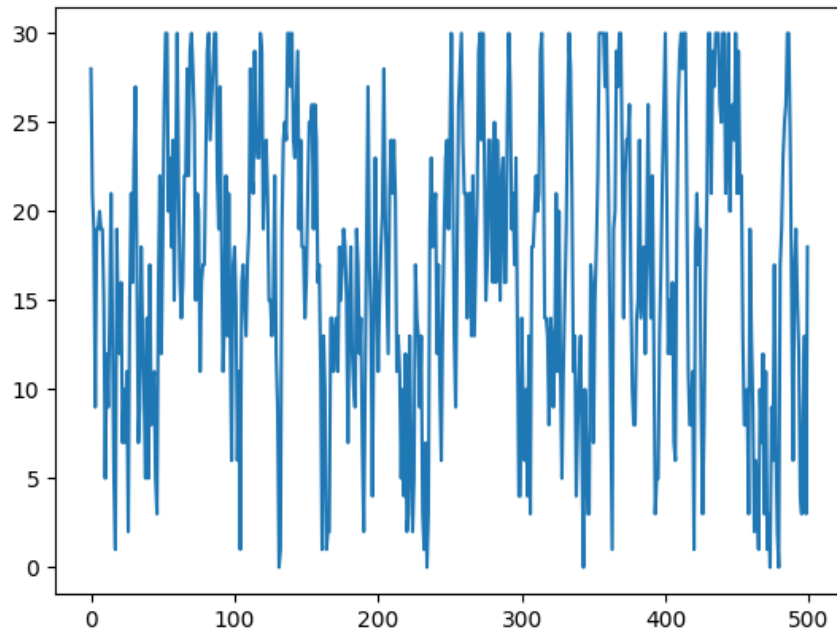
If we would assume that departures occur at the end of a period (hence, after the arrivals), then the code has to be as follows:

```
1  for i in range(1, num):
2      d = min(c[i], L[i-1])
3      loss[i] = max(L[i-1] + a[i] - K, 0) # service at end of period
4      L[i] = L[i-1] + a[i] - d - loss[i]
5
6  lost_fraction = sum(loss)/sum(a)
7  print(lost_fraction)
```

Ex 2.5. When $\rho = 1.51$, simulate a number of times to estimate the distribution $\pi_i = \mathsf{P}\,[L = i]$.

Ex 2.6. When $\rho = 10$, use a simple argument to show that $\pi_{K-2} \approx 0$.