Queueing Theory: 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.

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

```
7
```

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

```
3 5
```

2.2 two periods

```
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)
```

```
5 6
```

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

2.3 simulate many periods, make vectors

```
import numpy as np

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

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

2.4 Set seed

```
import numpy as np

np.random.seed(3)

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

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

2.5 update with a for loop

```
num = 5

#a = np.random.uniform(5, 8, size=num)
#c = np.random.uniform(5, 8, size=num)
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)
```

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

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

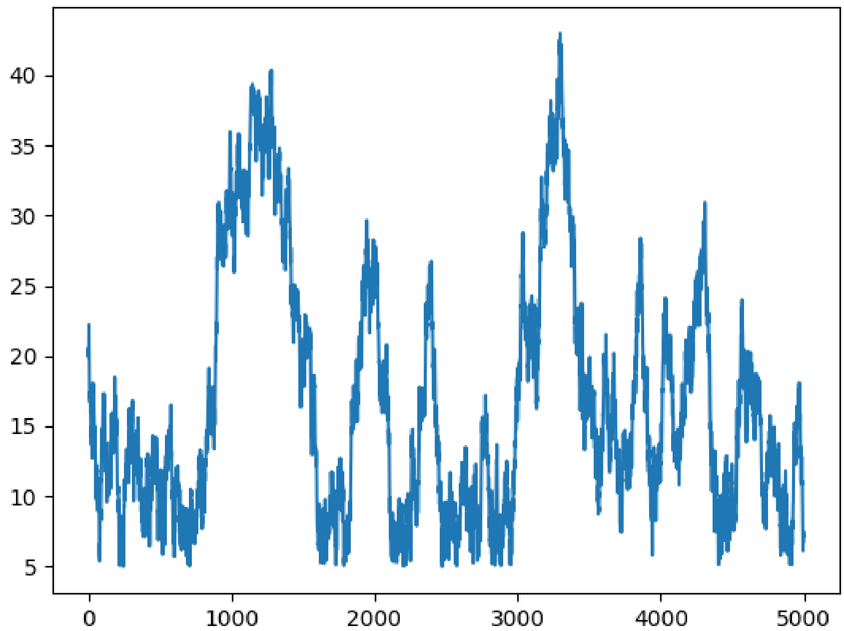
2.6 Compute mean and std of simulated queue length for $\rho \approx 1$

```
num = 5_000

np.random.seed(3)
a = np.random.uniform(5, 8, size=num)
c = (5+8)/1.99 * 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())
```

```
16.78550665013682 8.695855000533511
```

2.7 plot the queue length process

```
import matplotlib.pyplot as plt

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

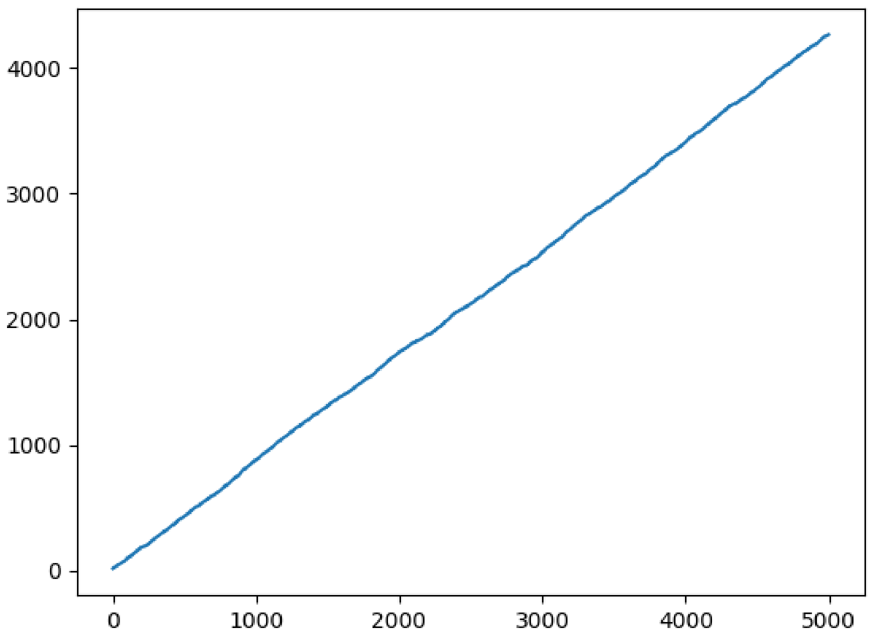
```
6.5051538887388 0.890452952271703
```

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)
```

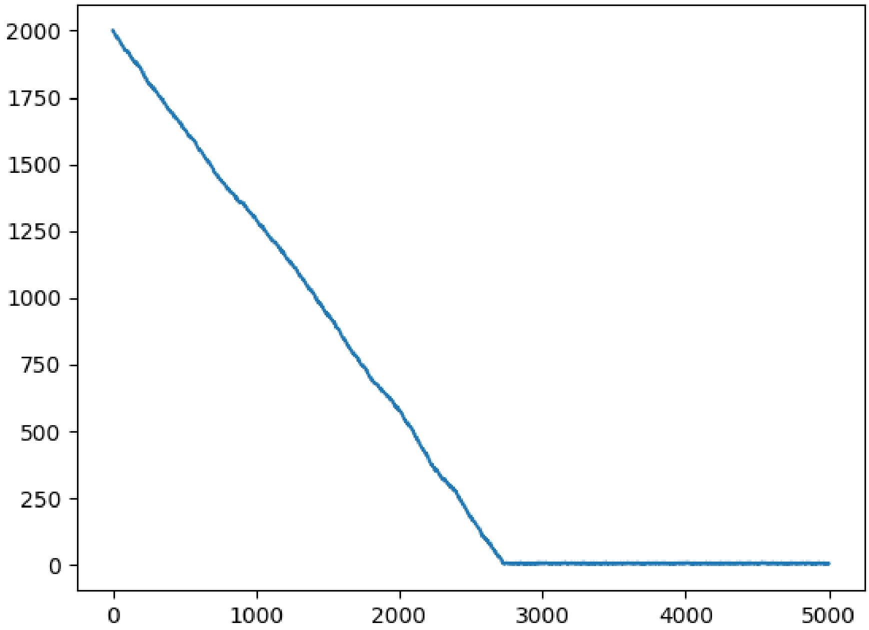
```
a = np.random.uniform(5, 8, size=num)
c = (5+8)/2.3 * np.ones(num)
L = np.zeros_like(a)  # queue length at the end of a period

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


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



2.10 Start with a large queue, take ρ< 1, show the drift

```
num = 5_000

np.random.seed(3)
a = np.random.uniform(5, 8, size=num)
c = (5+8)/1.8 * np.ones(num)
L = np.zeros_like(a)  # queue length at the end of a period

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

2.11 Queues with blocking.

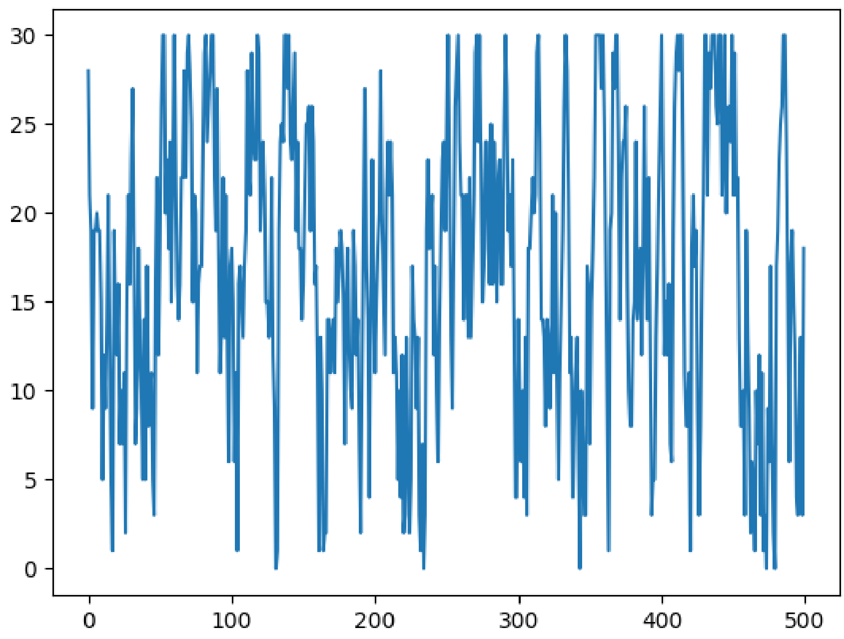
We have a queue subject to blocking. When the queue exceeds K, say, then whatever of the batch of items coming in exceeds K is rejected. This is the so-called complete reject rule. Two more assumptions: service occurs before arrival, and jobs arriving in a period cannot be served.

```
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
```

```
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:

```
for i in range(1, num):
    d = min(c[i], L[i-1])
```

```
    loss[i] = max(L[i-1] + a[i] - K, 0) # service at end of period
    L[i] = L[i-1] + a[i] - d - loss[i]

lost_fraction = sum(loss)/sum(a)
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 $\pi_{K-2} \approx 0$. Explain also that $\pi(K-1) \approx 1/(1+\rho)$. Use simulation to check your estimates.