

Queueing Theory: Simulation in discrete time

EBB074A05

Nicky D. van Foreest

2020:11:15

1 General info

Here is the code that goes with these youtube movies:

- <https://youtu.be/DfYxayoQmjYc>
- <https://youtu.be/D8BIAoBICnw>
- https://youtu.be/_BoagRyH5c0

1.1 Running python code in the browser

Here are two options to run python code in the browser so that you don't have to install python locally (i.e., on your computer):

- diggy
- colab

I prefer to work within an editor, and run the code in a terminal. This works much faster and more conveniently, but requires a bit (but not much) of intellectual effort and investment in how to configure things on your computer. If you're interested in the power tools I use, check out my tools page.

If you copy paste the code below into your own python environment or in the browser (on the above sites), the code should run more or less as is.

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 L
```

7

```

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

```

5 6

2.3 simulate many periods, make vectors

```

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

```

20 19 18 17 16

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

```

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 L.mean(), L.std()

```

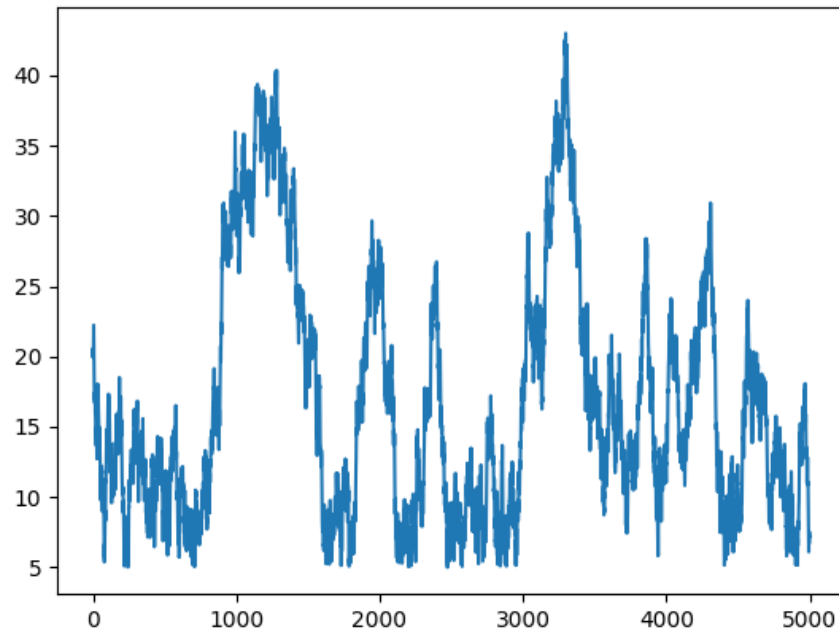
16.78550665013682 8.695855000533511

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 $\rho = 1/2$

```

1 num = 5_000
2
3 np.random.seed(3)
4 a = np.random.uniform(5, 8, size=num)
5 c = (5+8) * 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 L.mean(), L.std()

```

6.5051538887388 0.890452952271703

2.9 show the drift when $\rho > 1$

```

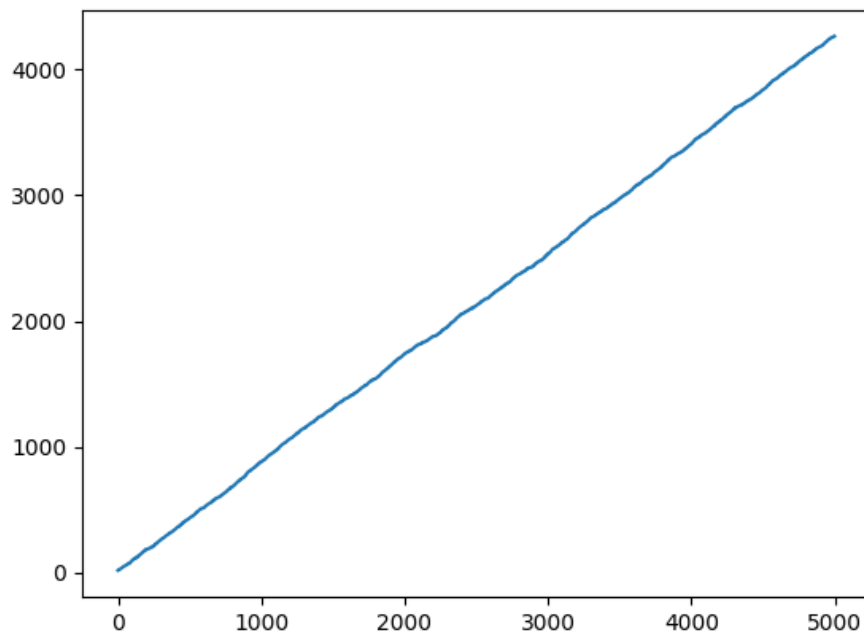
1 num = 5_000
2
3 np.random.seed(3)
4 a = np.random.uniform(5, 8, size=num)
5 c = (5+8)/2.3 * np.ones(num)
6 L = np.zeros_like(a) # queue length at the end of a period
7

```

```

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'

```



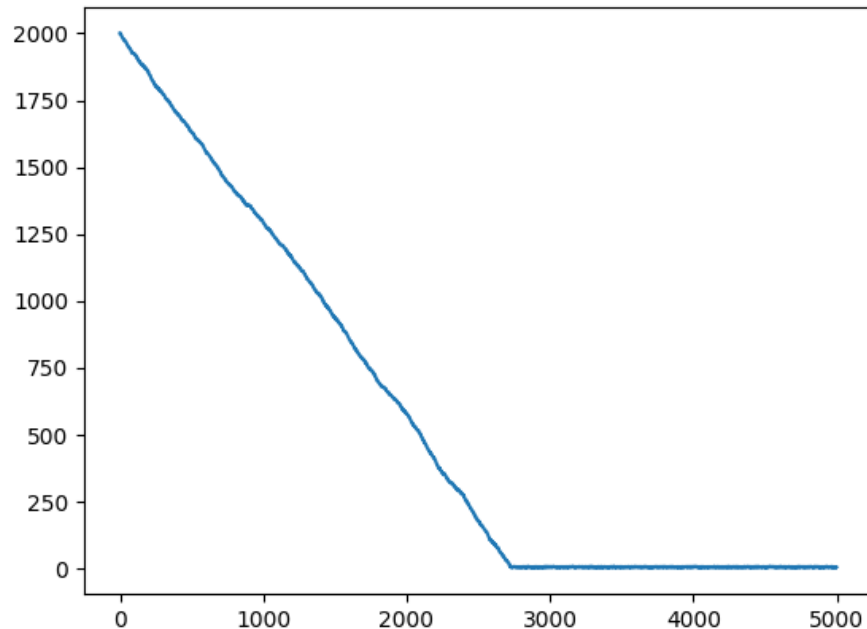
2.10 Start with a large queue, take $\rho < 1$, show the drift

```

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 length 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.

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.

```
1 num = 500
2
3 np.random.seed(3)
4 a = np.random.randint(0, 20, size=num)
5 c = 10*np.ones(num)
6 L = np.zeros_like(a) # queue length at the end of a period
7 loss = np.zeros_like(a) # queue length at the end of a period
8
9 K = 30 # max people in queue, otherwise they leave
10
```

```

11 L[0] = 28
12 for i in range(1, num):
13     d = min(c[i], L[i-1])
14     loss[i] = max(L[i-1] + a[i] - d - K, 0) # service before arrivals.
15     L[i] = L[i-1] + a[i] - d - loss[i]
16
17 lost_fraction = sum(loss)/sum(a)
18 lost_fraction

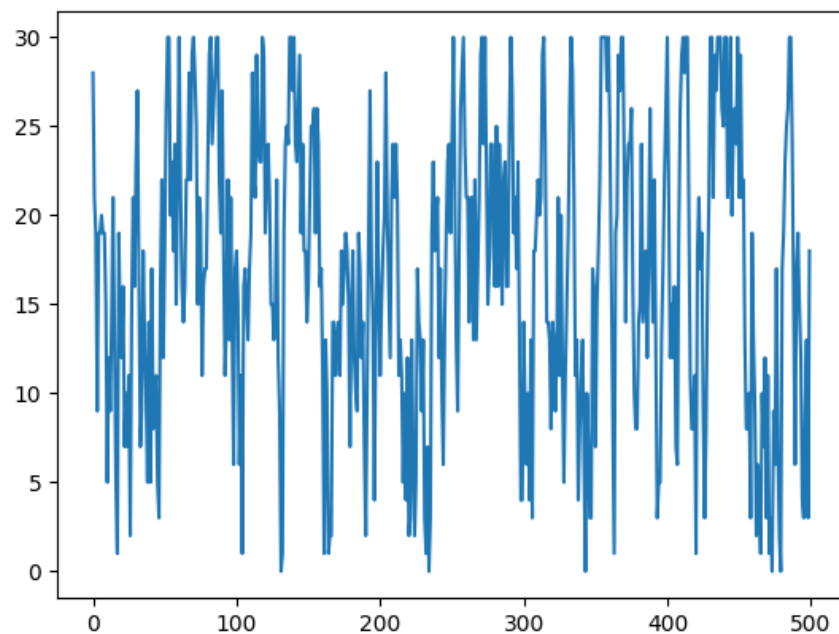
```

0.026064291920069503

```

1 plt.clf()
2 plt.plot(L)
3 plt.savefig('queue-discrete_loss.png')
4 '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 lost_fraction

```
