

Queueing Theory: Queues in continuous time

EBB074A05

Nicky D. van Foreest

2020:12:17

1 General info

This file contains the code and the results that go with this youtube movie: <https://youtu.be/bCU3oP6r-00>.

1.1 Set theme and font size

Set the theme and font size so that it is easier to read on youtube

```
(load-theme 'material-light t)
(set-face-attribute 'default nil :height 200)
```

By the way, this is emacs lisp; you cannot run it in python.

1.2 Load standard modules

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from matplotlib import style
4
5 style.use('ggplot')
6
7 np.random.seed(3)
```

2 Computing waiting times

2.1 Interarrival times

```
1 labda = 3
2 X = np.random.exponential(scale=labda, size=10)
3 X
```

2.40084716 3.69452354 1.03129621 2.14512092 6.70329244 6.79855956 0.40260163 0.69671515 0.1585

2.2 Arrival times

```
1 A = X.cumsum()
2 A
```

2.40084716 6.0953707 7.12666691 9.27178782 15.97508026 22.77363983 23.17624146 23.8729566 24.0

This is wrong. Why?
This is better:

```
1 A = np.zeros(len(X) + 1)
2 A[1:] = X.cumsum()
3 A
```

0 2.40084716 6.0953707 7.12666691 9.27178782 15.97508026 22.77363983 23.17624146 23.8729566

2.3 Service times

```
1 mu = 1.2 * labda
2 S = np.random.exponential(scale=mu,size=len(A))
3 S
```

0.10919375 2.19721993 3.77056631 1.17505899 4.06007571 3.21733717 0.08738687 2.94616653 1.0803

Note, $S[0]$ remains unused; it corresponds to job 0, but there is no job 0.

2.4 Departure times

```
1 D = np.zeros_like(A)
2
3 for k in range(1, len(A)):
4     D[k] = max(D[k-1], A[k]) + S[k]
5
6 D
```

0 4.59806709 9.86593702 11.040996 15.10107171 19.19241743 22.86102669 26.12240799 27.20275141

2.5 Waiting times

```
1 W = D - A
2 W
```

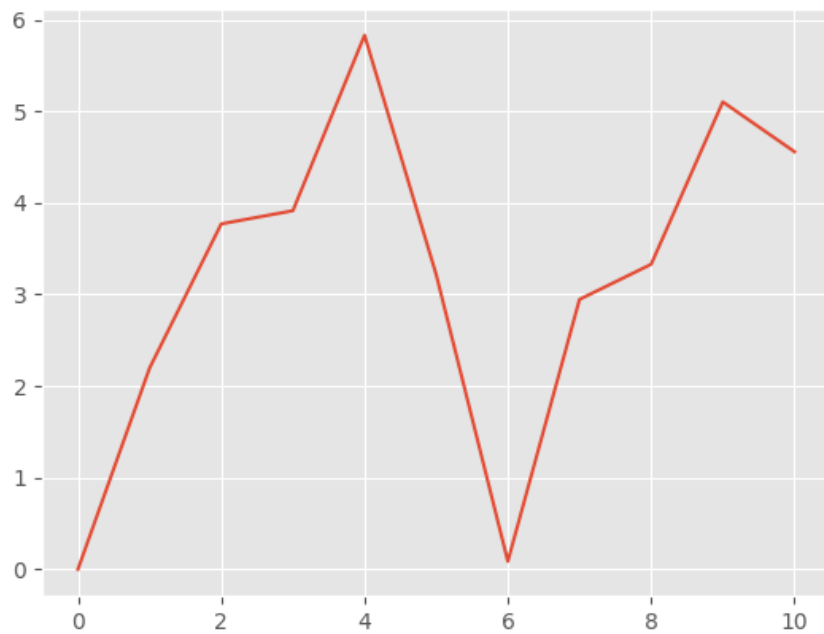
0 2.19721993 3.77056631 3.9143291 5.82928389 3.21733717 0.08738687 2.94616653 3.32979481 5.1

2.6 KPIs and plot

```
1 W.mean(), W.std()
```

3.177509575645523 1.7638308028443408

```
1 plt.clf()
2 plt.plot(W)
3 plt.savefig("wait.png")
4 "wait.png"
```



2.7 Is the queue stable?

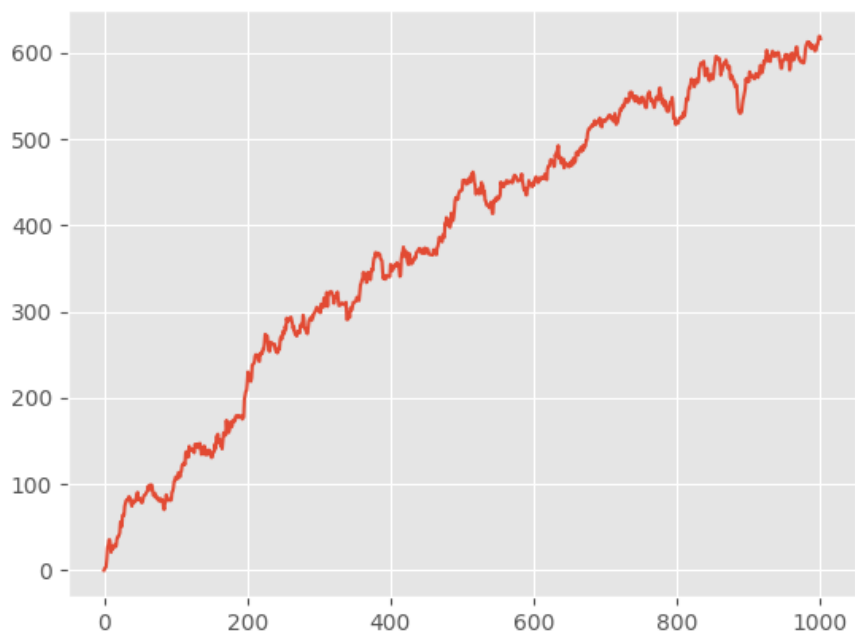
Let's do a longer simulation

```
1 num = 1000
2 labda = 3
3 X = np.random.exponential(scale=labda, size=num)
4 A = np.zeros(len(X) + 1)
5 A[1:] = X.cumsum()
6 mu = 1.2 * labda
7 S = np.random.exponential(scale=mu, size=len(A))
8 D = np.zeros_like(A)
9
10 for k in range(1, len(A)):
11     D[k] = max(D[k-1], A[k]) + S[k]
12
```

```

13 W = D - A
14
15 plt.clf()
16 plt.plot(W)
17 plt.savefig("wait2.png")
18 "wait2.png"

```

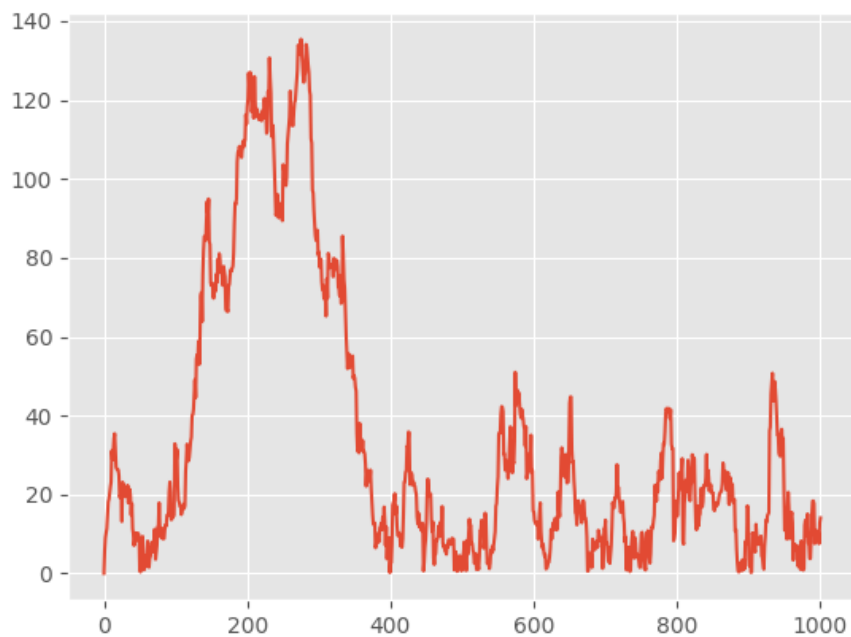


It's increasing. Let's try for another mu.

```

1 num = 1000
2 labda = 3
3 X = np.random.exponential(scale=labda, size=num)
4 A = np.zeros(len(X) + 1)
5 A[1:] = X.cumsum()
6 mu = 0.9 * labda # changed 1.2 to 0.9
7 S = np.random.exponential(scale=mu, size=len(A))
8 D = np.zeros_like(A)
9
10 for k in range(1, len(A)):
11     D[k] = max(D[k-1], A[k]) + S[k]
12
13 W = D - A
14
15 plt.clf()
16 plt.plot(W)
17 plt.savefig("wait3.png")
18 "wait3.png"

```



2.8 Queue length

We have the waiting times, but not the number of jobs in queue. How to compute the number of jobs in queue? We walk backwards!

```

1 num = 10
2 X = np.random.exponential(scale=labda, size=num)
3 A = np.zeros(len(X) + 1)
4 A[1:] = X.cumsum()
5 mu = 0.9 * labda # changed 1.2 to 0.9
6 S = np.random.exponential(scale=mu, size=len(A))
7 D = np.zeros_like(A)
8
9 for k in range(1, len(A)):
10     D[k] = max(D[k-1], A[k]) + S[k]
11
12 L = np.zeros_like(A)
13 for k in range(1, len(A)):
14     l = k - 1
15     while D[l] > A[k]:
16         l -= 1
17     L[k] = k - l - 1
18
19 L

```

A longer run.

```

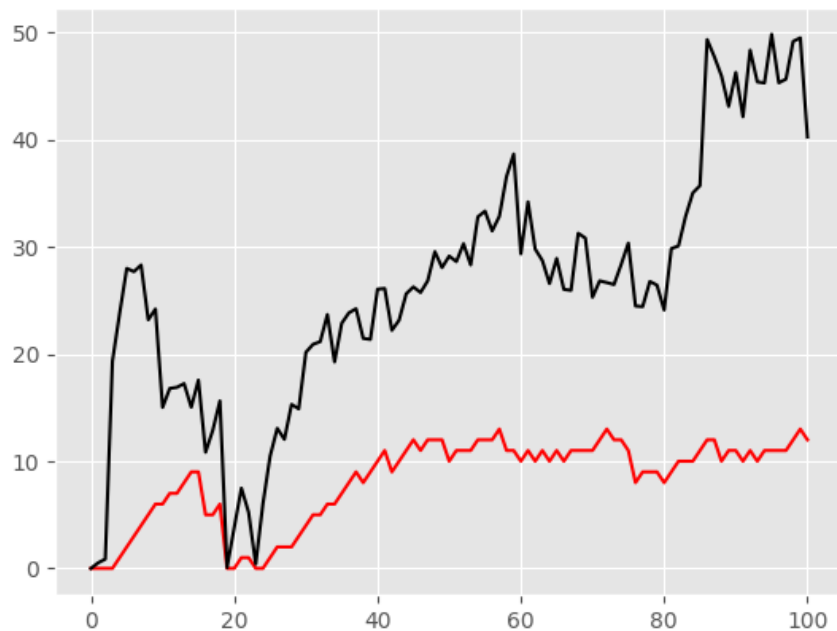
1 num = 100

```

```

2  X = np.random.exponential(scale=labda, size=num)
3  A = np.zeros(len(X) + 1)
4  A[1:] = X.cumsum()
5  mu = 0.9 * labda # changed 1.2 to 0.9
6  S = np.random.exponential(scale=mu,size=len(A))
7  D = np.zeros_like(A)
8
9  for k in range(1, len(A)):
10     D[k] = max(D[k-1], A[k]) + S[k]
11
12  W = D - A
13
14  L = np.zeros_like(A)
15  for k in range(1, len(A)):
16     l = k
17     while D[l] > A[k]:
18         l -= 1
19     L[k] = k - l - 1
20
21  plt.clf()
22  plt.plot(L, color='red')
23  plt.plot(W, color='black')
24  plt.savefig("wait4.png")
25  "wait4.png"

```



3 Multiserver queue in continuous time

```
1 m = 3
2 N = 10
3
4 one = np.ones(m, dtype=int) # vector with ones
5
6 X = np.ones(N + 1, dtype=int)
7 S = 5 * np.ones(N, dtype=int)
8 w = np.zeros(m, dtype=int)
9 W = J = A = D = 0 # helper variables.
10
11 for k in range(1, N):
12     A += X[k] # arrival time
13     s = w.argmin() # server chosen
14     W = w[s] # waiting time
15     J = W + S[k] # sojourn time
16     D = A + J # departure time
17     w[s] += S[k]
18     w = np.maximum(0, w - X[k + 1] * one)
19
20 W
```

6 7 8

4 Interesting challenges

- Estimate the distribution of the server's busy and idle times in the single server case
- Design tests to check that the code to compute L is correct, or else, to find and repair the bugs.
- Can the code for the multi server be changed such that the individual servers can have different speeds? I think that the vector one has to be changed such that the entries correspond to the speeds of the servers, but is that so? And if so, does the algorithm allow for this change?

5 Restore my emacs settings

```
(load-theme 'material t)
(set-face-attribute 'default nil :height 100)
```