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 TODO Set theme and font size

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

```
(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

```python
import numpy as np
import matplotlib.pylab as plt
from matplotlib import style

style.use('ggplot')

np.random.seed(3)
```

2 Computing waiting times

2.1 Interarrival times

```python
labda = 3
X = np.random.exponential(scale=labda, size=10)
print(X)
```

```
[2.40084716 3.69452354 1.03129621 2.14512092 6.70329244 6.79855956
 0.40260163 0.69671515 0.15851674 1.74379707]
```

2.2 Arrival times

```python
A = X.cumsum()
print(A)
```

```
[ 2.40084716  6.0953707   7.12666691  9.27178782 15.97508026 22.77363983
 23.17624146 23.8729566  24.03147334 25.77527041]
```

This is wrong. Why?

This is better:

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

```
[ 0.          2.40084716  6.0953707   7.12666691  9.27178782 15.97508026
 22.77363983 23.17624146 23.8729566  24.03147334 25.77527041]
```

2.3 Service times

```python
mu = 1.2 * labda
S = np.random.exponential(scale=mu,size=len(A))
print(S)
```

```
[0.10919375 2.19721993 3.77056631 1.17505899 4.06007571 3.21733717
 0.08738687 2.94616653 1.08034342 1.93073916 1.20028334]
```

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

2.4 Departure times

```python
D = np.zeros_like(A)

for k in range(1, len(A)):
    D[k] = max(D[k-1], A[k]) + S[k]

print(D)
```

```
[ 0.          4.59806709  9.86593702 11.040996   15.10107171 19.19241743
 22.86102669 26.12240799 27.20275141 29.13349057 30.33377391]
```

2.5 Waiting times

```python
W = D - A
print(W)
```

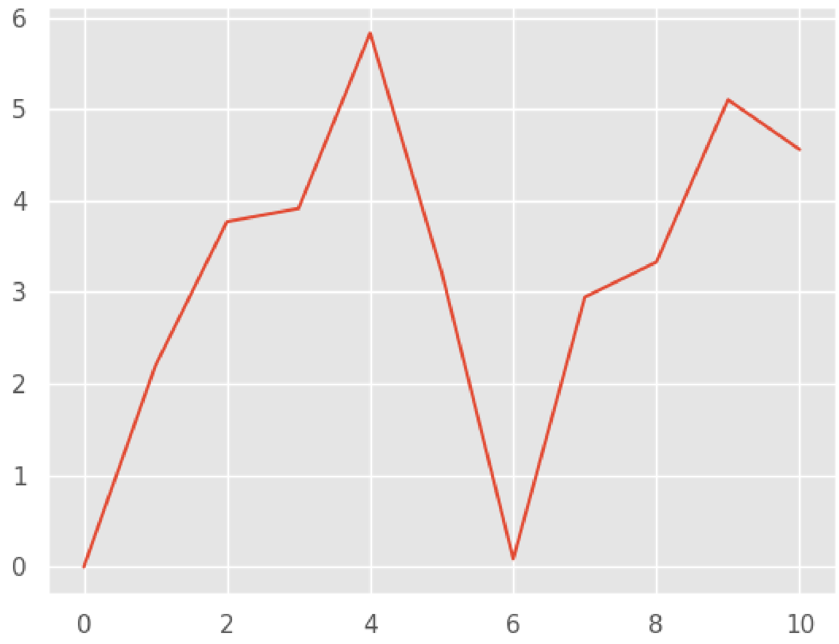
```
[0.         2.19721993 3.77056631 3.9143291  5.82928389 3.21733717
 0.08738687 2.94616653 3.32979481 5.10201723 4.5585035 ]
```

2.6 KPIs and plot

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

```
3.177509575645523 1.7638308028443408
```

```
plt.clf()
plt.plot(W)
plt.savefig("wait.png")
"wait.png"
```



2.7 Is the queue stable?

Let's do a longer simulation

```
num = 1000
labda = 3
X = np.random.exponential(scale=labda, size=num)
A = np.zeros(len(X) + 1)
A[1:] = X.cumsum()
mu = 1.2 * labda
S = np.random.exponential(scale=mu,size=len(A))
D = np.zeros_like(A)

for k in range(1, len(A)):
    D[k] = max(D[k-1], A[k]) + S[k]

```

```
W = D - A

plt.clf()
plt.plot(W)
plt.savefig("wait2.png")
"wait2.png"
```

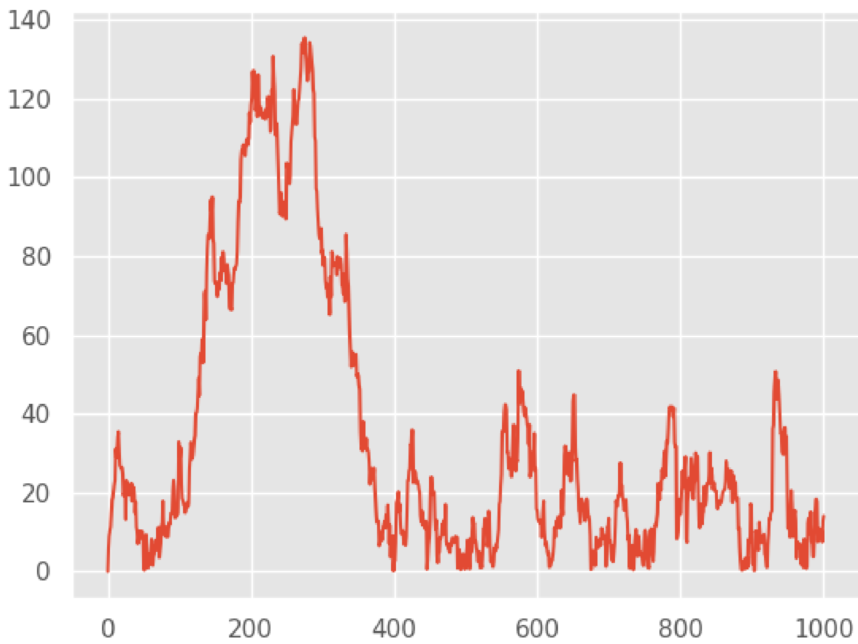
It's increasing. Let's try for another mu.

```
num = 1000
labda = 3
X = np.random.exponential(scale=labda, size=num)
A = np.zeros(len(X) + 1)
A[1:] = X.cumsum()
mu = 0.9 * labda  # changed 1.2 to 0.9
S = np.random.exponential(scale=mu,size=len(A))
D = np.zeros_like(A)

for k in range(1, len(A)):
    D[k] = max(D[k-1], A[k]) + S[k]

W = D - A

plt.clf()
plt.plot(W)
plt.savefig("wait3.png")
"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!

```
num = 10
X = np.random.exponential(scale=labda, size=num)
A = np.zeros(len(X) + 1)
A[1:] = X.cumsum()
mu = 0.9 * labda  # changed 1.2 to 0.9
S = np.random.exponential(scale=mu,size=len(A))
D = np.zeros_like(A)

for k in range(1, len(A)):
    D[k] = max(D[k-1], A[k]) + S[k]

L = np.zeros_like(A)
for k in range(1, len(A)):
    l = k - 1
    while D[l] > A[k]:
        l -= 1
    L[k] = k - l

print(L)
```

```
[0. 1. 1. 2. 3. 2. 3. 4. 3. 1. 1.]
```

A longer run.

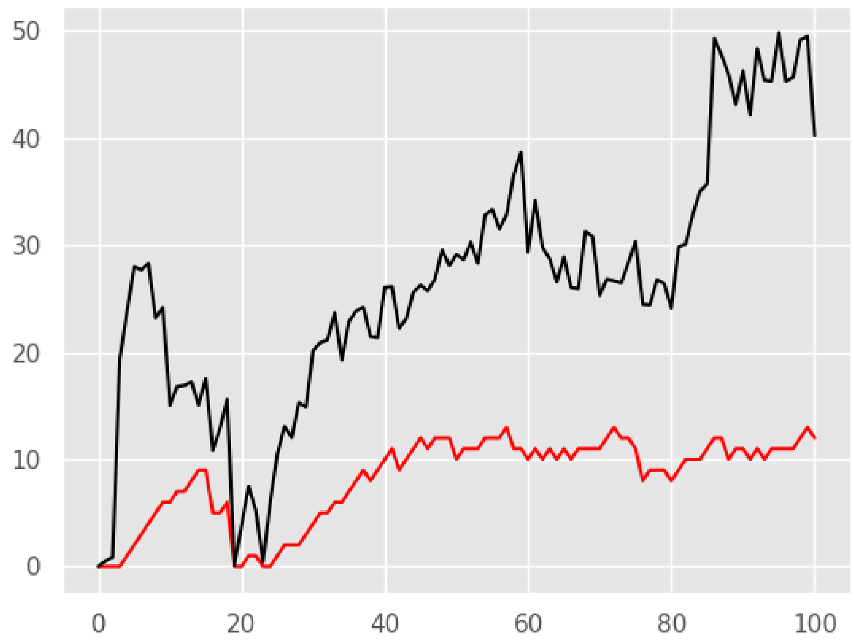
```
num = 100
X = np.random.exponential(scale=labda, size=num)
A = np.zeros(len(X) + 1)
A[1:] = X.cumsum()
mu = 0.9 * labda  # changed 1.2 to 0.9
S = np.random.exponential(scale=mu,size=len(A))
D = np.zeros_like(A)

for k in range(1, len(A)):
    D[k] = max(D[k-1], A[k]) + S[k]

W = D - A

L = np.zeros_like(A)
for k in range(1, len(A)):
    l = k
    while D[l] > A[k]:
        l -= 1
    L[k] = k - l - 1

plt.clf()
plt.plot(L, color='red')
plt.plot(W, color='black')
plt.savefig("wait4.png")
"wait4.png"
```



3 Multiserver queue in continuous time

```python
m = 3
N = 10

one = np.ones(m, dtype=int)  #  vector with ones

X = np.ones(N + 1, dtype=int)
S = 5 * np.ones(N, dtype=int)
w = np.zeros(m, dtype=int)

for k in range(1, N):
    s = w.argmin()  # server with smallest waiting time
    w[s] += S[k]  # assign arrival too this server
    w = np.maximum(0, w - X[k + 1] * one)

print(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)
```