A multi-server queue with changing interarrival rates

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://studio.youtube.com/video/nV4MwjMe3vQ/edit`

1.1 DONE Set theme and font size

Set the theme and font size so that it is easier to read on youbute
By the way, this is emacs lisp; you cannot run it in python.

1.2 Load standard modules

```
import numpy as np
from numpy.random import poisson, uniform, exponential
import matplotlib.pylab as plt
import seaborn as sns; sns.set()

np.random.seed(3)
```

2 Supermarket with one fast server.

In the simulations models we considered up to now the arrival rate was constant. For supermarkets we know that this is not true. In this simulation we learn how to simulate a (simple) queueing system with time changing arrival rates.

2.1 Make data

We take the average number of arrivals per minute constant during one hour. Thus, `labda = 1` means that during the first hour 1 customer arrives per minute, on average.

```
labdas = np.array([1, 2, 3, 3, 2.5, 2, 2, 2, 2, 3, 4, 3, 2.5, 2, 1])
# labdas = np.array([1, 2])  # Short case for testing
```

For ease, the shop opens at hour 1, and is open for this amount of hours.

```
print(len(labdas))
```

```
15
```

We now generate the number of arrivals in an hour. Let the number N_i of customers in hour i be $\sim$ Pois(60λ); recall λ is number of arrival per minute.

```
N = np.zeros(len(labdas), dtype=int)

for i, labda in enumerate(labdas):
    N[i] = poisson(labda * 60)
print(N)
```

```
[ 61 113 162 150 155 126 112 113 118 182 231 176 169 116  63]
```

The total number of arrivals is

```
print(N.sum())
```

```
2047
```

The average service time per customer is $\mathsf{E}[S]$, in minutes.

```
ES = 2.5  # expected service time  in minutes.
```

Finally, we need the service times. Let's take it as exponential.

```
S = exponential(ES, size=N.sum())
print(S.mean(), S.std()) # check
```

```
2.506528761470696 2.4599902225851897
```

Here is an exercise. Do it after you have read the entire document.

Ex 2.1. Change the service time distribution for another distribution (take any distribution to your liking.). Analyze the impact on the KPIs in the section below.

We need some extra capacity, for otherwise the queue is critically loaded. Take the slack s; initially I set it to 1. The number of servers planned for the ith hour is then $c_i = \lfloor \rho + s \rfloor$. Hence, c is at least as large as the loads.

```
loads = labdas * ES
print(loads)
```

```
[ 2.5   5.    7.5   7.5   6.25  5.    5.    5.    5.    7.5  10.    7.5
  6.25  5.    2.5 ]
```

```
slack = 1
num_servers = np.array(loads + slack, dtype=int)
print(num_servers)
```

```
[ 3  6  8  8  7  6  6  6  6  8 11  8  7  6  3]
```

2.2 First set of KPIs

The amount of planned capacity (hence payed fees to personnel) is $\sum_{i=1}^{n} c_i$.

```
total_offered_service_capacity = num_servers.sum()  # in hours
print(total_offered_service_capacity)
```

```
99
```

For the total amount of required service, add up all services and divide by 60 to get the time in hours.

```
total_offered_service = S.sum() / 60  # in hours
print(total_offered_service)
```

```
85.51440624550858
```

```
idle = 1 - total_offered_service / total_offered_service_capacity
print(idle)
```

```
0.13621811873223655
```

2.3 Make arrival times.

There is an interesting theorem that relates the Poisson and exponential distribution in the following way. Suppose that the inter-arrival times or jobs are $\sim \text{Exp}(\lambda)$. Then, given N arrivals during the interval $[0, T]$, the arrival times of the N jobs are uniform on $[0, T]$.

```
A = np.zeros(N.sum() + 1)
start = 1
for n in N:
    A[start : start + n] = A[start - 1] + sorted(uniform(0, 60, size=n))
    start += n

print(A[:10])
```

```
[0.         1.04562269 1.13263927 1.28932131 2.17167152 2.90243072
 3.98007067 4.24975945 4.96445375 6.54453062]
```

Ex 2.2. While building the model the code was initially like this.

```
A = np.zeros(N.sum())
start = 1
for n in N:
    A[start : start + n] = A[start - 1] + uniform(0, 60, size=n)
    start = n
```

Explain which lines contain errors (there are 3) and why these are wrong.

2.4 A fast server

While you are *still in queue* of a multi-server queue, the rate at which jobs are served is the same whether there are c servers or just one server working at a rate of c, i.e., c times as fast as a server in the multi-server. As a first simple case, we just model the system as if there is one fast server.

```
W = np.zeros_like(S)
hour = 0
for k in range(1, len(W)):
    X = A[k] - A[k - 1]
    c = num_servers[int(A[k] / 60)]
    W[k] = max(W[k - 1] + S[k - 1] / c - X, 0)
```

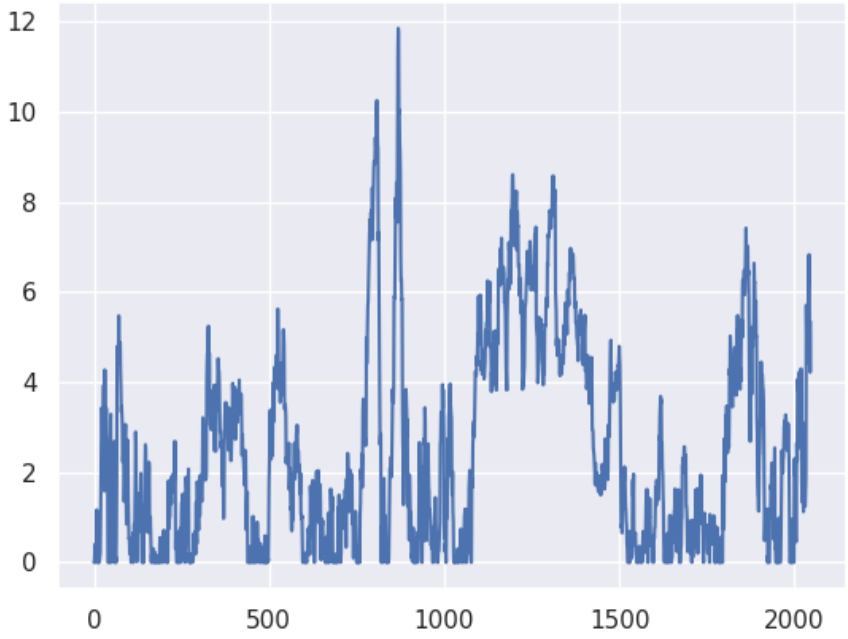
This is a bit tricky: `A[k]` is the arrival time of job k. Then `A[k]/60` is the hour in which job k arrives. Then we look up the number of servers planned for that hour.

Here are some KPIs.

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

```
2.4888178640316547 2.2978630075196977 11.854734178715747
```

```
plt.clf()
plt.plot(W)
plt.savefig('multi0.png')
'multi0.png'
```



Ex 2.3. Change the slack and see what happens to the queue length process and the KPIs.

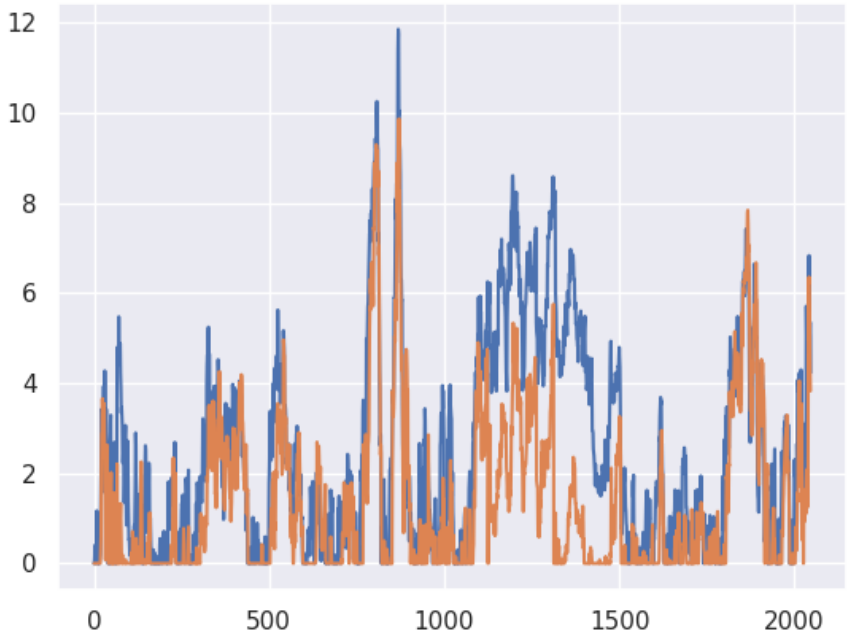
3 A real multiserver queue

```
m = max(num_servers)

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

w = np.zeros(m)
W = np.zeros_like(S)
server = np.zeros_like(S)

# explain why W[k] has to be above w[s]. And how I found this error.

for k in range(1, len(W)):
    X = A[k] - A[k - 1]
    c = num_servers[int(A[k] / 60)]
    s = w[:c].argmin()
    server[k] = s
    W[k] = w[s]
    w[s] += S[k]  # store waiting time
    w = np.maximum(0, w - X * one)


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

```
1.3141839485857727 1.7606254417452725 9.866951032209062
```

Let's plot it.

```
plt.plot(W)
plt.savefig('multi1.png')
'multi1.png'
```



Here I don't use `plt.clf()` because I want to compare the graphs of both models.

Ex 3.1. Explain line 14 below. Hint, compare the code of multi server we discussed earlier.

Ex 3.2. While building the model the code was initially like this.

```
for k in range(1, len(W)):
    X = A[k] - A[k - 1]
    c = num_servers[int(A[k] / 60)]
    s = w[:c].argmin()
    server[k] = s
    w[s] += S[k]
    W[k] = w[s]
    w = np.maximum(0, w - X * one)
```

Explain which line contain an errors. What is wrong? How do you think I discovered the mistake (i.e,. what tests did I do so that I found the problem)?

Ex 3.3. Make some graphs, 4 or so, in which you analyze different aspects of the setting. Recall, the aim is that you play with the code, you don't have to do anything special, or bright. Just changing a few of the parameters and making graphs is fine. (Do put labels, and explain what you see, and whether this is in line with what you expected before you did the experiments.)

Ex 3.4. Explain how a supermarket could use this simulator. What is the data they have to provide?

Ex 3.5. Finally, point out one aspect in which the simulator can be improved.

4 Restore my emacs settings