Queueing Theory: Psychiatrists doing intakes

EBB074A05

Nicky D. van Foreest

2022:01:19

1 General info

This file contains the code and the results that go with the YouTube movies mentioned in the README file in this directory.

I included a number of exercises to help you think about the code. Keep your answers short; you don't have to win the Nobel prize on literature.

2 Base situation

Five psychiatrists do intakes. See the queueing book for further background.

2.1 Load standard modules

We need some standard libraries for numerical work and plotting.

Python Code

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

style.use('ggplot')

np.random.seed(3)
```

2.2 Simulate queue length

Python Code

```python
def computeQ(a, c, Q0=0):  # initial queue length is 0
    N = len(a)
    Q = np.empty(N)  # make a list to store the values of  Q
    Q[0] = Q0
    for n in range(1, N):
        d = min(Q[n - 1], c[n])
        Q[n] = Q[n - 1] + a[n] - d
    return Q
```

Ex 2.1. Compute the queue lenghts when

Python Code

```python
a = [1,2,5,6,8,3,7,3]
c = [2,2,0,5,4,4,3,2]
```

and include the results in your report.

2.3 Arrivals

We start with run length 10 for demo purpose; later we extend to longer run times

Python Code

```python
a = np.random.poisson(11.8, 10)
print(a)
```

```
[12  9  7 13 14  9  9 11 12 10]
```

2.4 Service capacity

Python Code

```python
def unbalanced(a):
    p = np.empty([5, len(a)])
    p[0, :] = 1.0 * np.ones_like(a)
    p[1, :] = 1.0 * np.ones_like(a)
    p[2, :] = 1.0 * np.ones_like(a)
    p[3, :] = 3.0 * np.ones_like(a)
    p[4, :] = 9.0 * np.ones_like(a)
    return p

p = unbalanced(a)
print(p)
```

```
[[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [3. 3. 3. 3. 3. 3. 3. 3. 3. 3.]
 [9. 9. 9. 9. 9. 9. 9. 9. 9. 9.]]
```

Ex 2.2. Which psychiatrist does the most intakes per week?

2.5 Include holidays

Python Code

```python
def spread_holidays(p):
    for j in range(len(a)):
        psych = j % 5
        p[psych, j] = 0

spread_holidays(p)
print(p)
```

```
[[0. 1. 1. 1. 1. 0. 1. 1. 1. 1.]
 [1. 0. 1. 1. 1. 1. 0. 1. 1. 1.]
 [1. 1. 0. 1. 1. 1. 1. 0. 1. 1.]
 [3. 3. 3. 0. 3. 3. 3. 3. 0. 3.]
 [9. 9. 9. 9. 0. 9. 9. 9. 9. 0.]]
```

Ex 2.3. What is the long-run time average of the weekly capacity?

2.6 Total weekly service capacity

Python Code

```python
s = np.sum(p, axis=0)
print(s)
```

```
[14. 14. 14. 12.  6. 14. 14. 14. 12.  6.]
```

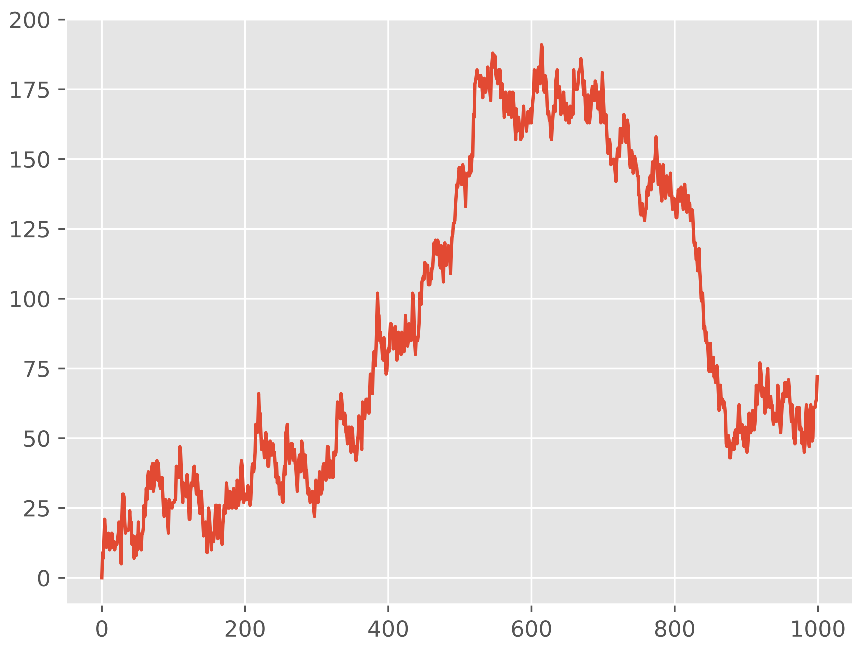
Ex 2.4. Explain why we need to take the sum over `axis=0` to compute the average weekly capacity for the intakes.

2.7 Simulate the queue length process

Python Code

```python
np.random.seed(3)

a = np.random.poisson(11.8, 1000)
p = unbalanced(a)
spread_holidays(p)
s = np.sum(p, axis=0)

Q1 = computeQ(a, s)

plt.clf()
plt.plot(Q1)
plt.savefig("psych1.pdf")
```

```
figures/None
```



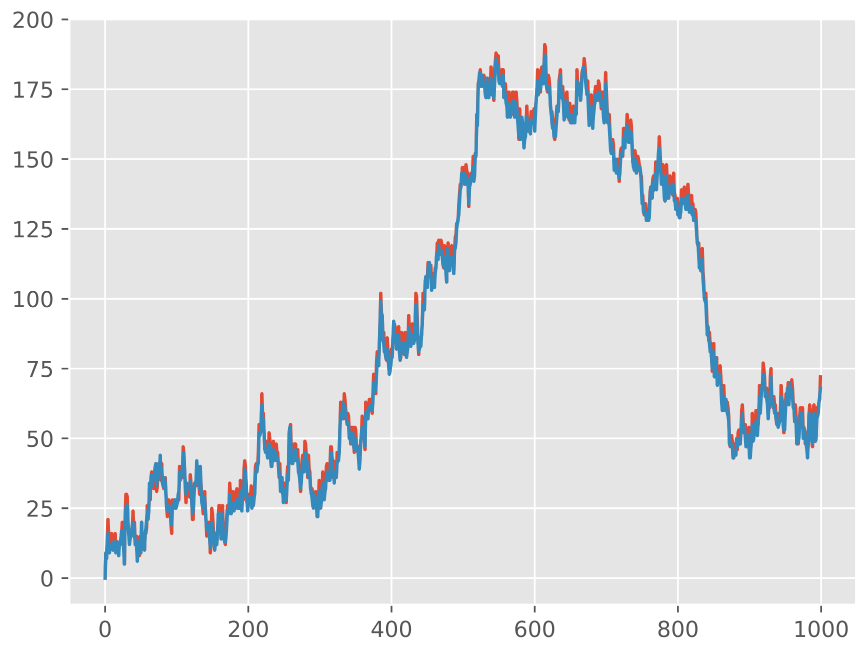
3 Evaluation of better (?) plans

3.1 Balance the capacity more evenly over the psychiatrists

I set the seed to enforce a start with the same arrival pattern.

Python Code

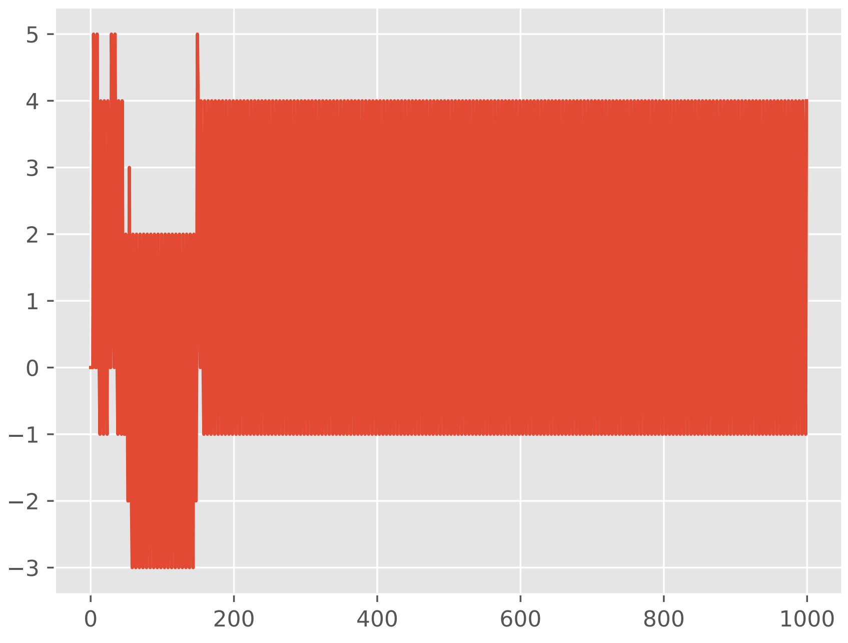
```python
def balanced(a):
    p = np.empty([5, len(a)])
    p[0, :] = 2.0 * np.ones_like(a)
    p[1, :] = 2.0 * np.ones_like(a)
    p[2, :] = 3.0 * np.ones_like(a)
    p[3, :] = 4.0 * np.ones_like(a)
    p[4, :] = 4.0 * np.ones_like(a)
    return p

np.random.seed(3)
a = np.random.poisson(11.8, 1000)


p = balanced(a)
spread_holidays(p)
s = np.sum(p, axis=0)
Q2 = computeQ(a, s)

plt.plot(Q2)
plt.savefig("psych2.pdf")
```



Python Code

```python
plt.clf()
plt.plot(Q1-Q2)
plt.savefig("psych22.pdf")
```

```
figures/None
```



Ex 3.1. How can we see that the effect of balancing capacity is totally uninteresting?

Ex 3.2. Change the capacities of the psychiatrists but keep the average weekly capacity the same. Include a graph of your result.

3.2 Synchronize holidays

What is the effect of all psychiatrists taking holidays in the same week?

Python Code

```python
a = np.random.poisson(11.8, 10)


def synchronize_holidays(p):
    for j in range(int(len(a) / 5)):
        p[:, 5 * j] = 0  # this

p = unbalanced(a)
synchronize_holidays(p)
print(p)
```

```
[[0. 1. 1. 1. 1. 0. 1. 1. 1. 1.]
 [0. 1. 1. 1. 1. 0. 1. 1. 1. 1.]
 [0. 1. 1. 1. 1. 0. 1. 1. 1. 1.]
 [0. 3. 3. 3. 3. 0. 3. 3. 3. 3.]
 [0. 9. 9. 9. 9. 0. 9. 9. 9. 9.]]
```

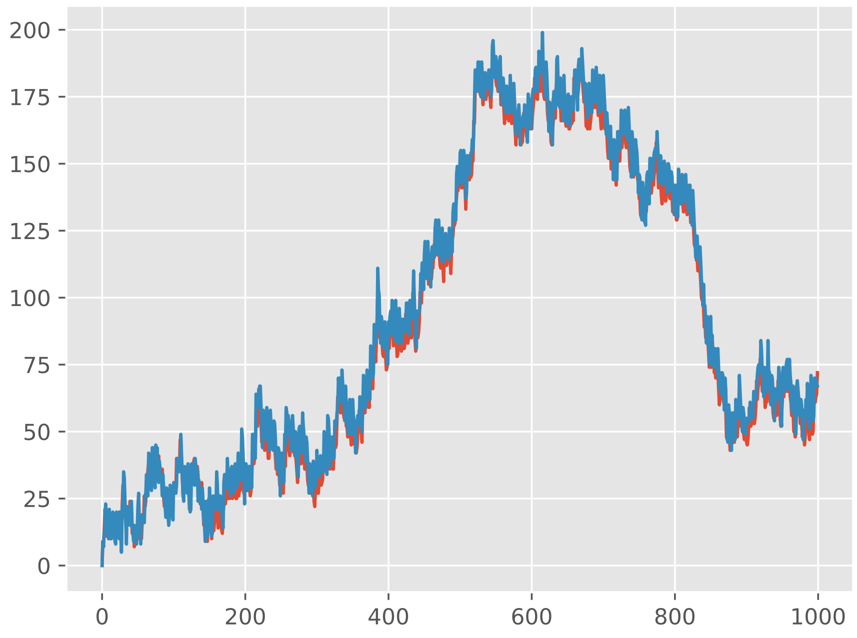
Ex 3.3. Explain how the code works. Specifically, what does the line marked as `this`.

Ex 3.4. Change the code such that psychiatrists go on holiday every 6 weeks. Include your code, and check with a sum (over an appropriate axis) that the average weekly capacity is still the same afte your changes.

Python Code

```python
np.random.seed(3)

a = np.random.poisson(11.8, 1000)
p = unbalanced(a)
spread_holidays(p)
s = np.sum(p, axis=0)
Q3 = computeQ(a, s)

plt.clf()
plt.plot(Q3)

p = balanced(a)
synchronize_holidays(p)
s = np.sum(p, axis=0)
Q4 = computeQ(a, s)

plt.plot(Q4)
plt.savefig("psych3.pdf")
```

`figures/None`



Ex 3.5. Just to improve your coding skills (and your creativity), formulate another vacation plan. Implement this idea in code, and test its success/failure. Make a graph to show its effect on the dynamics of the queue length. (I don't mind whether your proposal works or not; as long as

you 'play' and investigate, all goes.) Include your code—if you ported all this code to R , then include your R code— and comment on thex difficult points.

Most probably, your proposals will also not solve the problem. We need something smarter.

4 Control capacity as a function of queue length

4.1 Simple on-off strategies

Let's steal an idea from supermarkets: dynamic control.

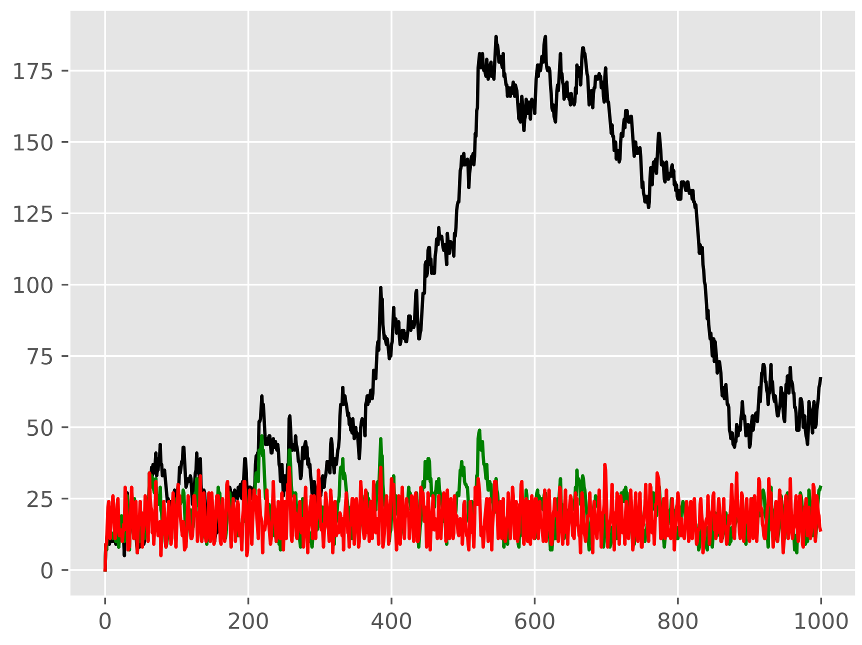
Python Code

```python
lower_thres = 12
upper_thres = 24

def computeQExtra(a, c, e, Q0=0):  #  initial queue length is 0
    N = len(a)
    Q = [0] * N  # make a list to store the values of  Q
    Q[0] = Q0
    for n in range(1, N):
        if Q[n - 1] < lower_thres:
            C = c - e
        elif Q[n-1] >= upper_thres:
            C = c + e
        d = min(Q[n-1], C)
        Q[n] = Q[n-1] + a[n] -d
    return Q


np.random.seed(3)
a = np.random.poisson(11.8, 1000)
c = 12
Q = computeQ(a, c * np.ones_like(a))
Qe1 = computeQExtra(a, c, 1)
Qe5 = computeQExtra(a, c, 5)

plt.clf()
plt.plot(Q, label="Q", color='black')
plt.plot(Qe1, label="Qe1", color='green')
plt.plot(Qe5, label="Qe5", color='red')
plt.savefig("psychfinal.pdf")
```

figures/None

Ex 4.1. Explain how the if statements in the code above work.

Ex 4.2. Explain how this idea relates to what happens in a supermarket if there are still open service desks but queues become very long.



We see, dynamically controlling the service capacity (as a function of queue length) is a much better plan.

Ex 4.3. Use simulation to show that the psychiatrists don't have more work.

Ex 4.4. Choose some other control thresholds (something reasonable of course, but otherwise you are free to select your own values.) Run the simulation with your values, include a graph and explain what you see.

4.2 Hire an extra server for a fixed amount of time

In the real case the psychiatrists hired an extra person to do intakes when the queue became very long, 100 or higher, and then they hired this person for one month (you may assume that a month consists of 4 weeks). Suppose this person can do 2 intakes a day and works for 4 days a week.

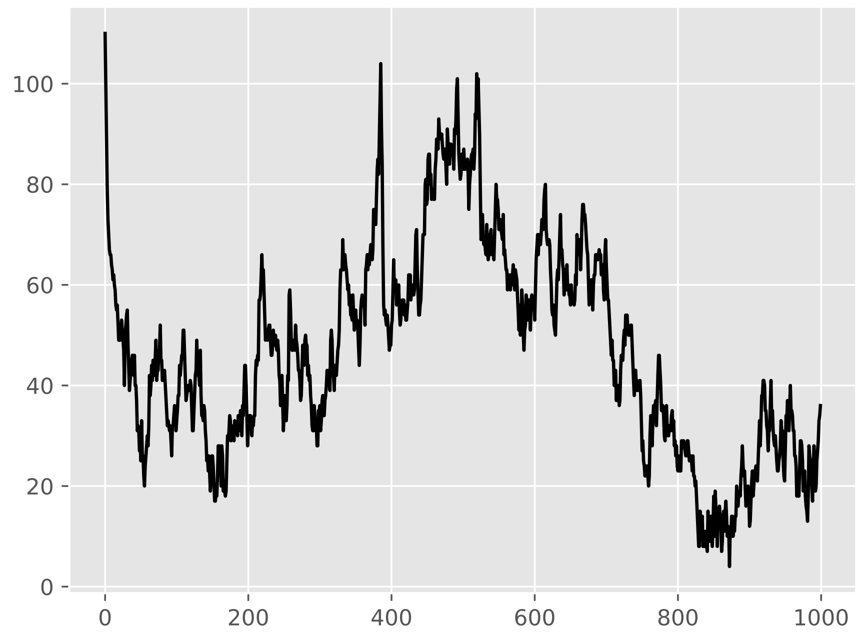
The code below implements this control algorithm.

Ex 4.5. Explain the code below.

Python Code

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

style.use('ggplot')
np.random.seed(3)

extra_capacity = 8  # extra weekly capacity
contract_duration = 4   # weeks


def compute_Q_control(a, c, Q0=0):
    N = len(a)
    Q = np.empty(N)
    Q[0] = Q0
    extra = False
    mark_time = 0
    for n in range(1, N):
        if Q[n - 1] > 100:
            extra = True
            mark_time = n
        if extra and n >= mark_time + contract_duration:
            extra = False
        d = min(Q[n - 1], c[n] + extra * extra_capacity)
        Q[n] = Q[n - 1] + a[n] - d
    return Q


a = np.random.poisson(11.8, 1000)
c = 12
Q = compute_Q_control(a, c * np.ones_like(a), Q0=110)
# print(Q)
plt.clf()
plt.plot(Q, label="Q", color='black')
plt.savefig("psych_extra.pdf")
```

figures/None



Ex 4.6. Do a number of experiments to see the effect of the duration of the contract by making it longer (experiment 1), or shorter (experiment 2). Run the simulation, Include graphs, and discuss the effect of these changes.

Ex 4.7. Now change the number of intakes per day done by the extra person. (For instance, an experienced person can do more intakes in the same amount of time than a newbie. However, this comes at an additional cost of course.) Make a graph, and compare the effect of this change to the previous (changing the duration).

Ex 4.8. If you were a consultant, what would you advice the psychiatrists on how to control their waiting lists?