# Queueing Theory: Psychiatrists doing intakes

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

## 2 Base situation

5 psychiatrists do intakes. See my queueing book for further background.

### 2.1 Load standard modules

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

style.use('ggplot')

np.random.seed(3)
```

### 2.2 Simulate queue length

```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
```

### 2.3 Arrivals

We start with run length 10 for demo purpose.

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

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

### 2.4 Service capacity

```
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.]]
```

### 2.5 Include holidays

```
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.]]
```

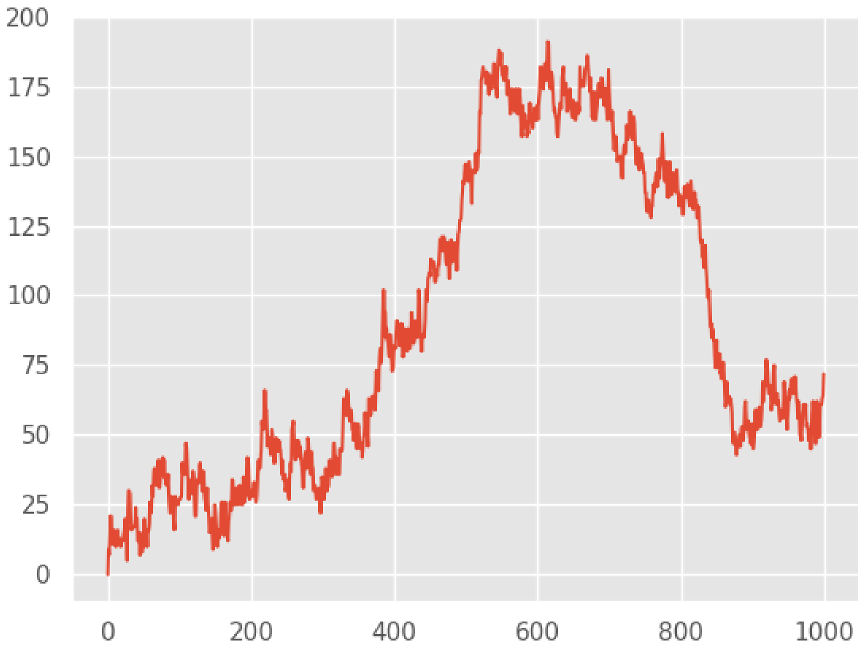
### 2.6 Total weekly service capacity

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

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

### 2.7 Simulate the queue length process

```
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.png")
"psych1.png"
```



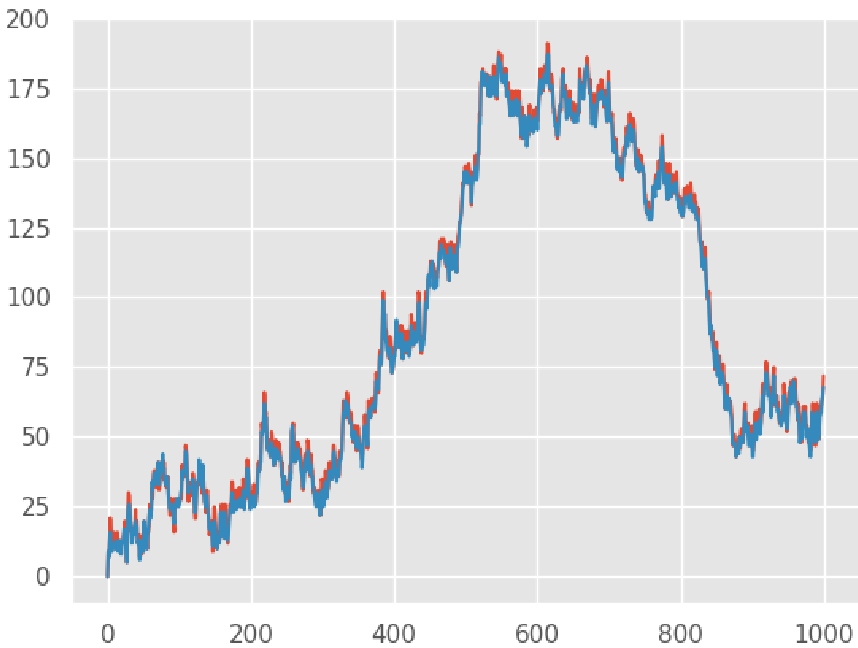

## 3 Evaluation of better (?) plans

### 3.1 Balance the capacity more evenly over the psychiatrists

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

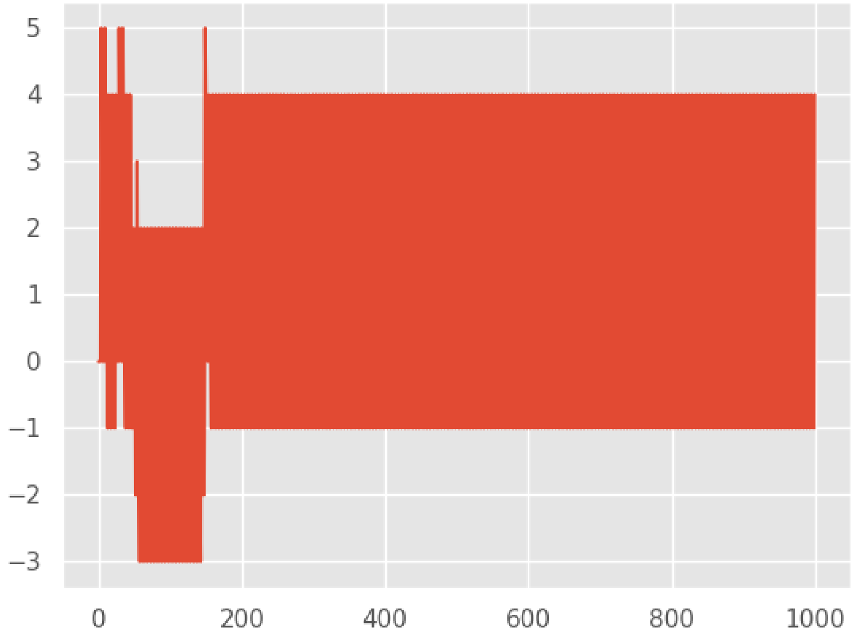
```
def balanced(a):
    p = np.empty([5, len(a)])
    p[0, :] = 2.0 * np.ones_like(a)
    p[1, :] = 2.0 * np.ones_like(a)
    p[2, :] = 3.0 * np.ones_like(a)
```

```
    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.png")
"psych2.png"
```




What is the effect?

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




The effect of balancing capacity is totally uninteresting.

## 3.2 Synchronize holidays

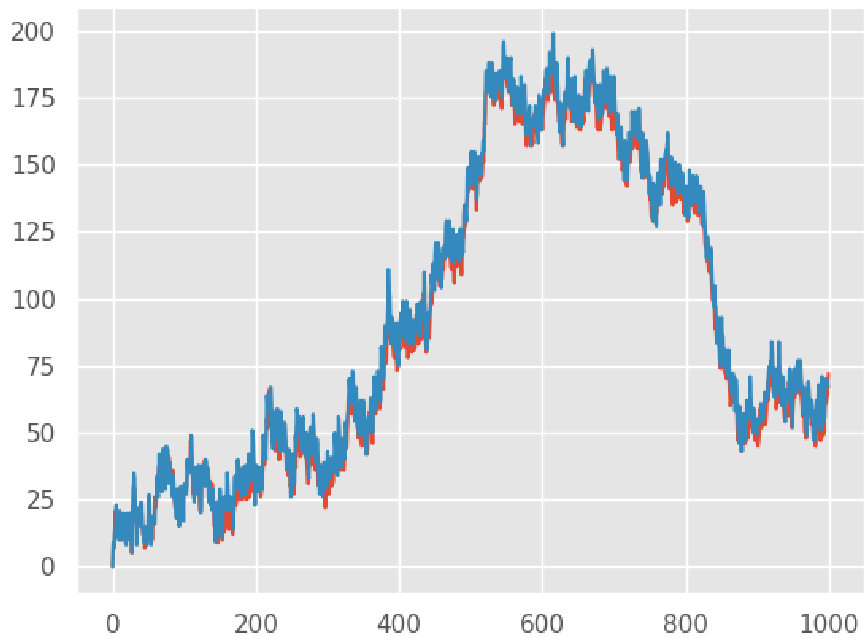
What is the effect of all psychiatrists taking holidays in the same week?

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


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

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.]]
```

```
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.png")
"psych3.png"
```

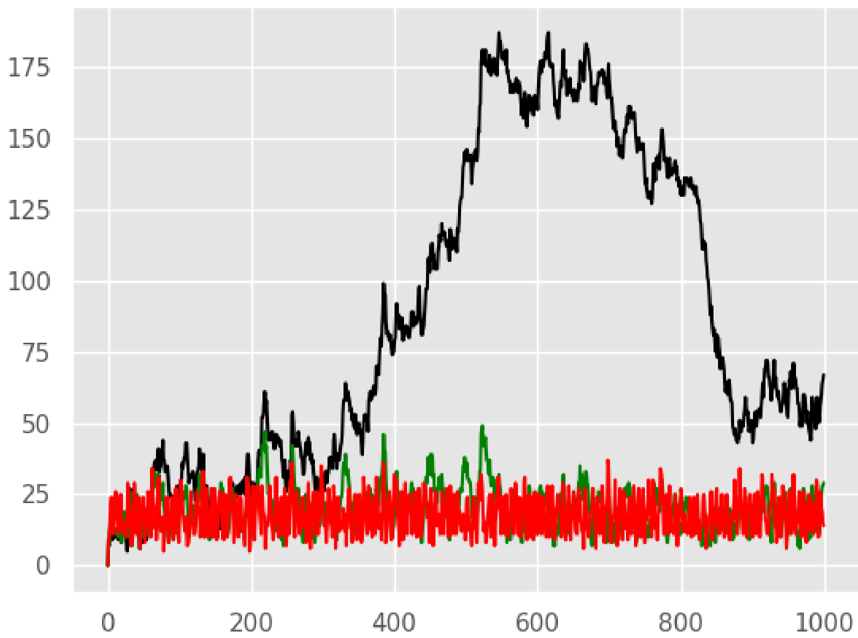



All these proposals will not solve the problem. We need something smarter. For this, we steal an idea from supermarkets: dynamic control.

## 4 Control capacity as a function of queue length

```
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.png")
"psychfinal.png"
```




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

# 5 Restore my emacs settings