

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

```
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)
```

2.2 Simulate queue length

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

2.3 Arrivals

We start with run length 10 for demo purpose.

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

12	9	7	13	14	9	9	11	12	10
----	---	---	----	----	---	---	----	----	----

2.4 Service capacity

```
1 def unbalanced(a):
2     p = np.empty([5, len(a)])
3     p[0, :] = 1.0 * np.ones_like(a)
4     p[1, :] = 1.0 * np.ones_like(a)
5     p[2, :] = 1.0 * np.ones_like(a)
6     p[3, :] = 3.0 * np.ones_like(a)
7     p[4, :] = 9.0 * np.ones_like(a)
8     return p
9
10 p = unbalanced(a)
11 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

```
1 def spread_holidays(p):
2     for j in range(len(a)):
3         psych = j % 5
4         p[psych, j] = 0
5
6 spread_holidays(p)
7 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

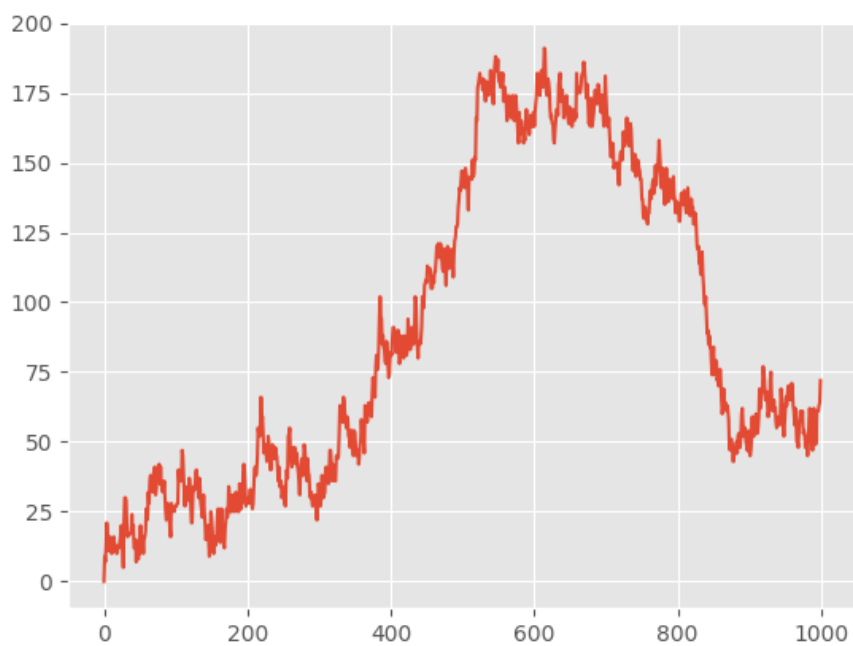
2.6 Total weekly service capacity

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

14	14	14	12	6	14	14	14	12	6
----	----	----	----	---	----	----	----	----	---

2.7 Simulate the queue length process

```
1 np.random.seed(3)
2
3 a = np.random.poisson(11.8, 1000)
4 p = unbalanced(a)
5 spread_holidays(p)
6 s = np.sum(p, axis=0)
7
8 Q1 = computeQ(a, s)
9
10 plt.clf()
11 plt.plot(Q1)
12 plt.savefig("psych1.png")
13 "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.

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

```

6     p[3, :] = 4.0 * np.ones_like(a)
7     p[4, :] = 4.0 * np.ones_like(a)
8     return p
9
10    np.random.seed(3)
11    a = np.random.poisson(11.8, 1000)
12
13
14    p = balanced(a)
15    spread_holidays(p)
16    s = np.sum(p, axis=0)
17    Q2 = computeQ(a, s)
18
19    plt.plot(Q2)
20    plt.savefig("psych2.png")
21    "psych2.png"

```

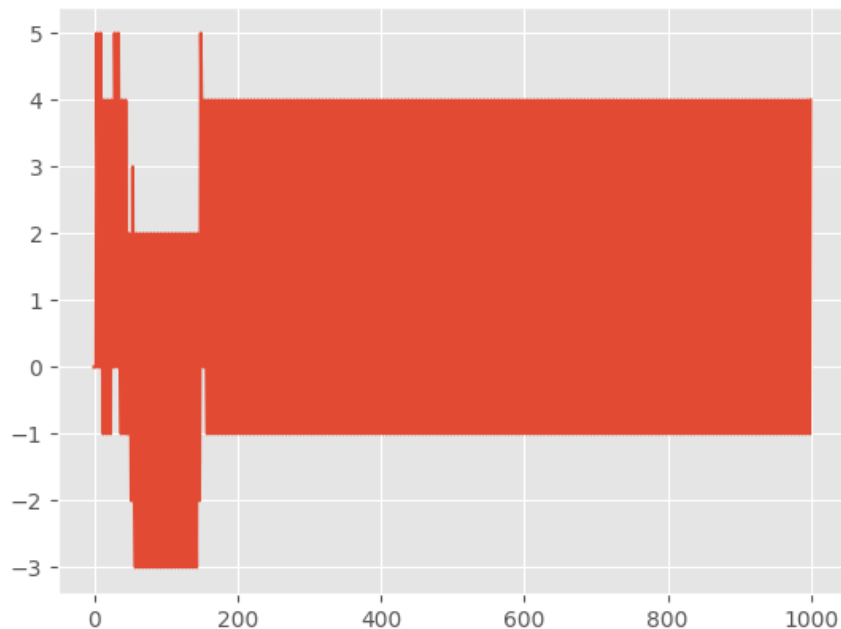


What is the effect?

```

1    plt.clf()
2    plt.plot(Q1-Q2)
3    plt.savefig("psych22.png")
4    "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?

```

1 a = np.random.poisson(11.8, 10)
2
3
4 def synchronize_holidays(p):
5     for j in range(int(len(a) / 5)):
6         p[:, 5 * j] = 0
7
8 p = unbalanced(a)
9 synchronize_holidays(p)
10 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

```

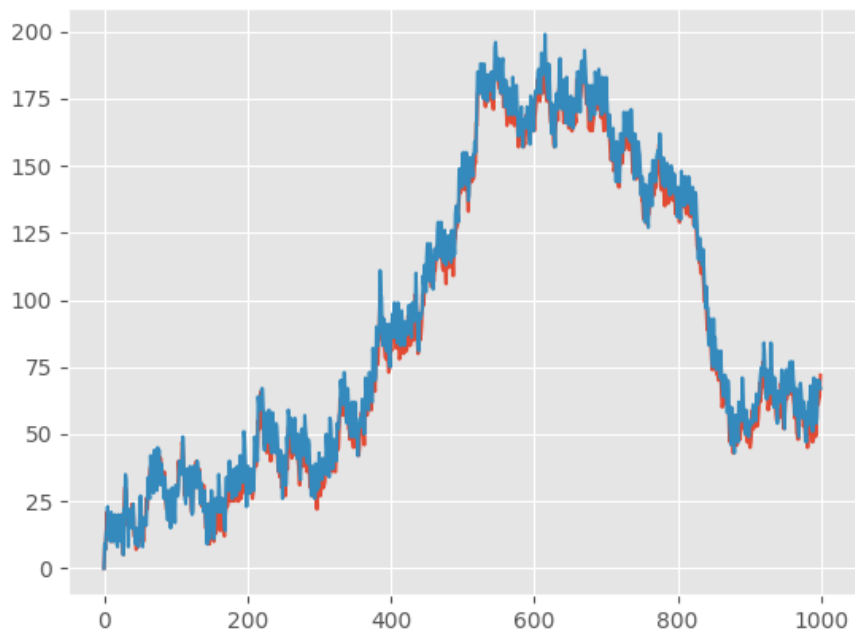
1 np.random.seed(3)
2
3 a = np.random.poisson(11.8, 1000)
4 p = unbalanced(a)
5 spread_holidays(p)

```

```

6  s = np.sum(p, axis=0)
7  Q3 = computeQ(a, s)
8
9  plt.clf()
10 plt.plot(Q3)
11
12 p = balanced(a)
13 synchronize_holidays(p)
14 s = np.sum(p, axis=0)
15 Q4 = computeQ(a, s)
16
17 plt.plot(Q4)
18 plt.savefig("psych3.png")
19 "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

```

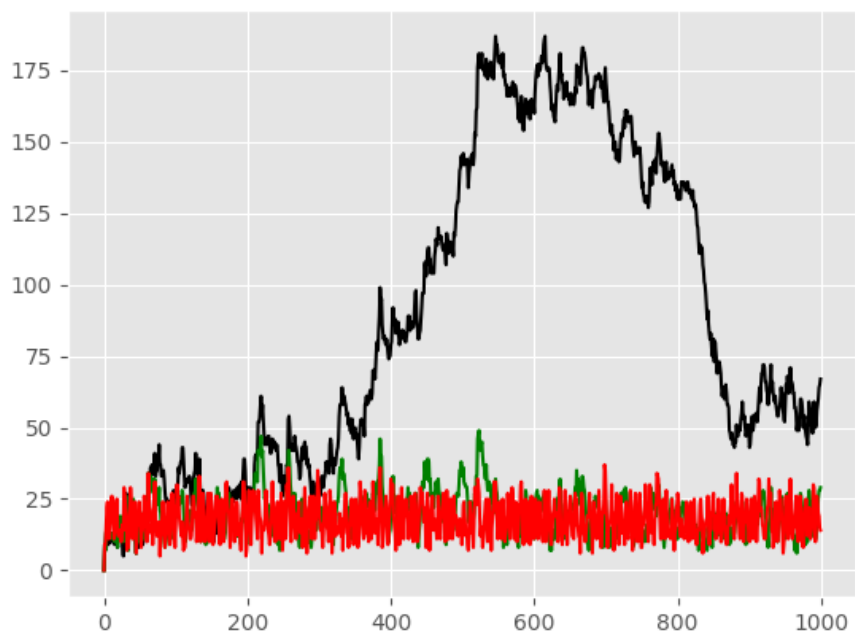
1  lower_thres = 12
2  upper_thres = 24
3
4  def computeQExtra(a, c, e, Q0=0): # initial queue length is 0
5      N = len(a)
6      Q = [0] * N # make a list to store the values of Q
7      Q[0] = Q0

```

```

8     for n in range(1, N):
9         if Q[n - 1] < lower_thres:
10             C = c - e
11         elif Q[n-1] >= upper_thres:
12             C = c + e
13         d = min(Q[n-1], C)
14         Q[n] = Q[n-1] + a[n] - d
15     return Q
16
17
18 np.random.seed(3)
19 a = np.random.poisson(11.8, 1000)
20 c = 12
21 Q = computeQ(a, c * np.ones_like(a))
22 Qe1 = computeQExtra(a, c, 1)
23 Qe5 = computeQExtra(a, c, 5)
24
25 plt.clf()
26 plt.plot(Q, label="Q", color='black')
27 plt.plot(Qe1, label="Qe1", color='green')
28 plt.plot(Qe5, label="Qe5", color='red')
29 plt.savefig("psychfinal.png")
30 "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

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