Queueing Theory: Empirical distributions.

EBB074A05

Nicky D. van Foreest

2020:12:16

1 General info

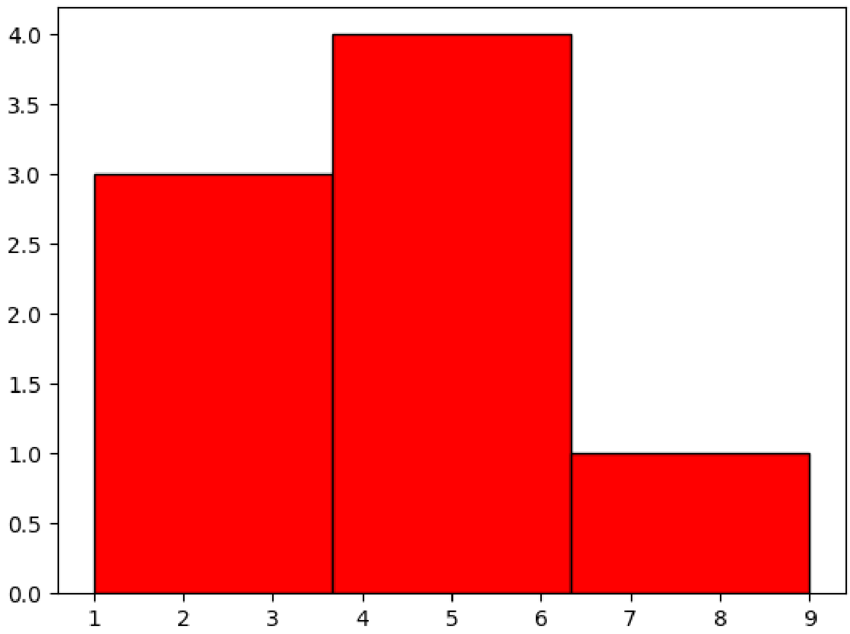
This file contains the code and the results that go with this youtube movie: `https://youtu.be/aKfv908uWqM`

2 Empirical distribution, how to make and plot

We want to know the fraction of periods the queue length is longer than some value q, say. For this we will make the empirical distribution of the queue lengths.

2.1 Plotting a PDF/histogram

```
import matplotlib.pyplot as plt

x = [2, 5, 2, 1, 9, 5, 5, 5]

plt.clf()
plt.hist(x, bins=3, facecolor='red', edgecolor='black', linewidth=1)
plt.savefig('emp0.png')
'emp0.png'
```



2.1.1 DONE Change the number of bins from 3 to 7.

you can remove the `bins` argument altogether.

2.2 First naive idea

Given a set of measurements $x_1, \ldots, x_n$, the empirical CDF is defined as

$$F(x) = \frac{1}{n} \sum_{i=1}^{n} I_{x_i \leq x}$$

This is a clean mathematical definition, but as if often the case with mathematical definitions, you should stay clear from using it to *compute* the CDF: the numerical performance is absolutely terrible.

```
x = [2, 5, 2, 1, 9, 5, 5, 5]

def F(y):
    tot = 0
    for xi in x:
        tot += xi <= y

    return  tot/len(x)

print(F(5.5))
```

```
0.875
```

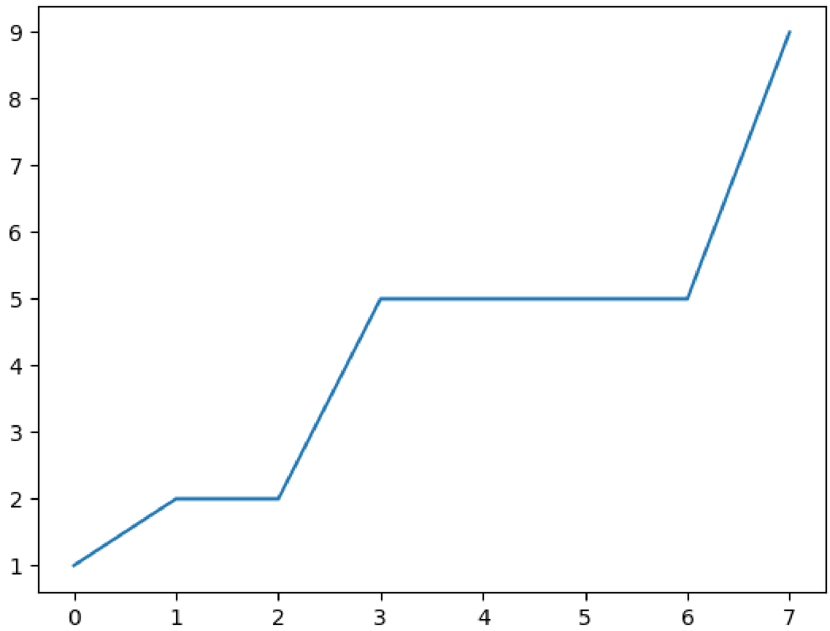
Ex 2.1. Explain in your own words why this way to compute the empirical CDF is a not so smart (i.e., pretty dumb) idea.

2.3 A better idea

```
print(sorted(x))
```

```
[1, 2, 2, 5, 5, 5, 5, 9]
```

```
plt.clf()
plt.plot(sorted(x))
plt.savefig("emp00.png")
"emp00.png"
```



2.4 Yet better idea

```
def cdf_better(x):
    x = sorted(x)
    n = len(x)
    y = range(1, n + 1)
    y = [z / n for z in y]   # normalize
    return x, y


x = [2, 5, 2, 1, 8, 5, 5]
x, F = cdf_better(x)
print(F)
```

```
[0.14285714285714285, 0.2857142857142857, 0.42857142857142855, 0.5714285714285714, 0.714285714285
```

2.4.1 DONE Explain

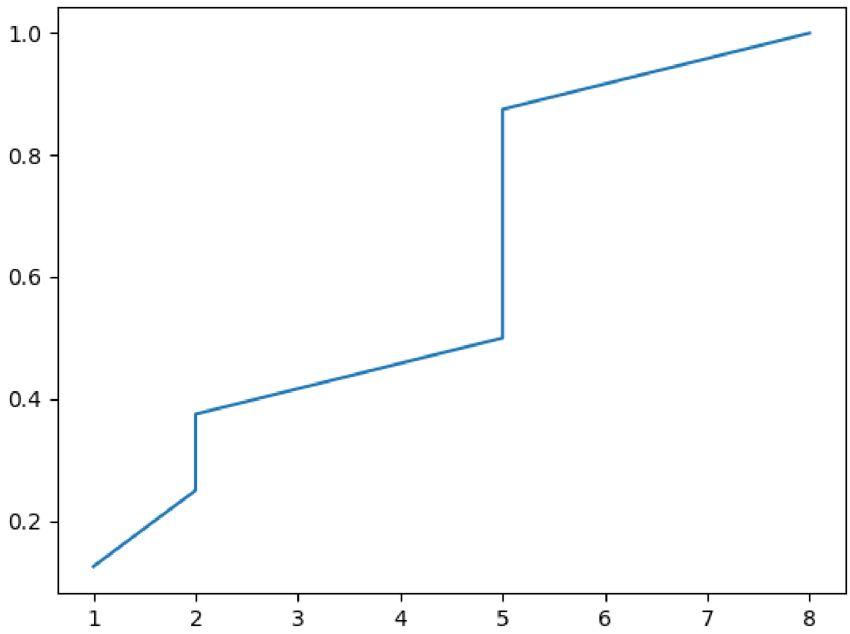
Why the ~n = len(x)

2.4.2 TODO Explain

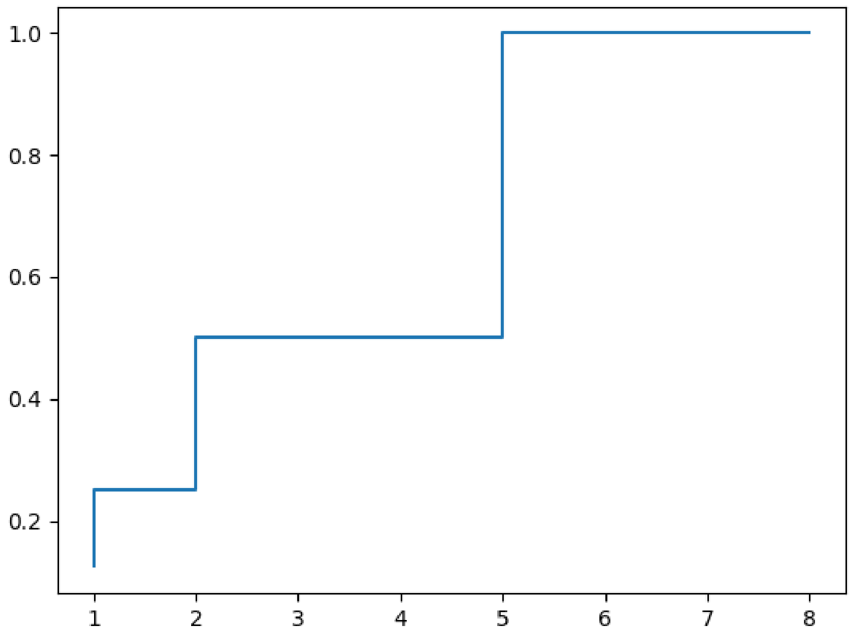
You should know that `for` loops in R and python are quite slow. We use this in the list comprehension in the line in which we `#normalize`. For larger amounts of data it is better to use `numpy`. This we do below.

2.5 Plot the cdf

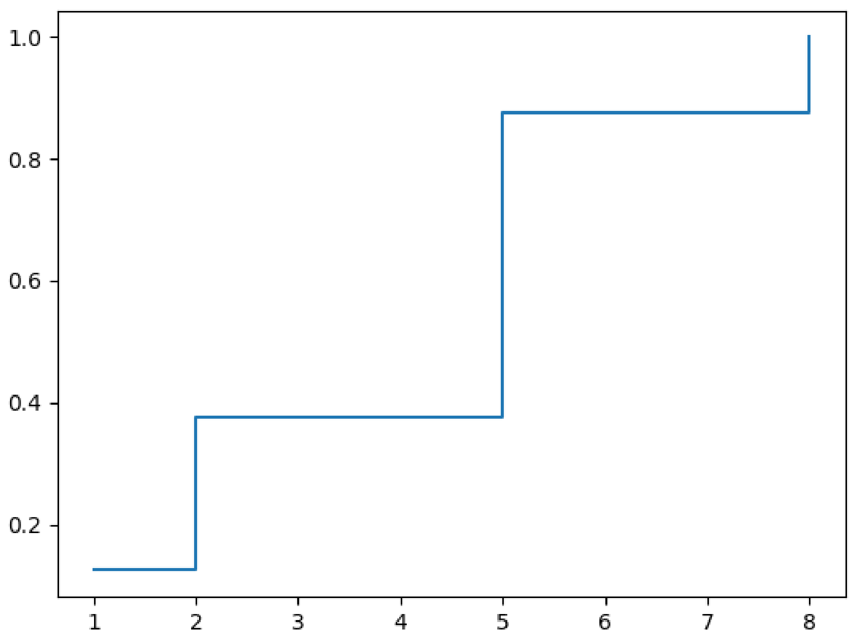
```
x = [2, 5, 2, 1, 8, 5, 5, 5]
x, F = cdf_better(x)

plt.clf()
plt.plot(x, F)
plt.savefig(fname)
fname
```




```
plt.clf()
plt.step(x, F)
plt.savefig(fname)
fname
```




```python
plt.clf()
plt.plot(x, F, drawstyle="steps-post")
plt.savefig(fname)
fname
```



2.6 Faster with numpy

```python
import numpy as np

def cdf(x):
    y = np.arange(1, len(x) + 1) / len(x)
    x = np.sort(x)
    return x, y

x = [2, 5, 2, 1, 8, 5, 5]
x, F = cdf(x)
print(F)
```

```
[0.14285714 0.28571429 0.42857143 0.57142857 0.71428571 0.85714286
 1.        ]
```

2.7 Remove duplicate values

Finally, we can make the computation of the cdf significantly faster with using the following numpy functions.

```python
unique, count = np.unique(np.sort(x), return_counts=True)
print(unique, count)
```

```
[1 2 5 8] [1 2 3 1]
```

```python
print(count.cumsum()/7)
```

```
[0.14285714 0.42857143 0.85714286 1.        ]
```

```python
def cdf_fastest(x):
    # remove multiple occurences of the same value
    unique, count = np.unique(np.sort(x), return_counts=True)
    x = unique
    y = count.cumsum() / count.sum()
    return x, y

x = [2, 5, 2, 1, 8, 5, 5]
x, F = cdf_fastest(x)
print(F)
```

```
[0.14285714 0.42857143 0.85714286 1.        ]
```

Ex 2.2. What does `unique` do? (Look it up on the web.) What happens if you forget to sort the input `x`?

Ex 2.3. Find the CDF for arrival times [2,5,7,8,9,10] and plot it

Ex 2.4. Find some data set on the web—take any data source you like—and compute the empirical CDF. Include a graph of the ecdf, and comment on your findings. You can compare your results with the `EDCF` function of the `statsmodels` module (search on google), if you are interested.

Ex 2.5. Explain what the Kolmogorov-Smirnov test has to do with ecdf's; search on the web for this. Keep your discussion short, but to the point.