Simulating Poisson processes

University of Toronto, ACT 350

Prof. S. Pesenti

19/10/2019

How to simulate a Poisson process

Recall that a Poisson process with rate $\lambda > 0$ is a counting process $\{N(t),\ t \geq 0\}$ that has the following properties

1. *starts at 0*: $N(0) = 0$ with probability 1
2. *is increasing*: $N(s) \leq N(t)$ for all $s \leq t$
3. *has Poisson distributed increments*: $P(N(s+t) - N(s) = n) \sim \text{Pois}(\lambda t)$ for all $s, t \geq 0$.

However, this definition does not help if we want to simulate a Poisson process. Thus, to simulate a Poisson process we use its alternative definition. That is, for independent random variables $X_1, X_2, \dots$ that are Exponentially distributed with rate $\lambda > 0$, a Poisson process with rate $\lambda > 0$ can be written as

$$N(t) = \max\Big\{ n \in \mathbb{N} \;\Big|\; \sum_{i=1}^{n} X_i \leq t \Big\}, \quad \text{for all } t \geq 0.$$

Thus, for simulating a Poisson process we first need to simulate exponential random variables, add them up and check whether they are smaller or equal to t. However, we will run into problems when we want to store the value of $N(t)$ for every $0 \leq t < \infty$. The way out is to implement $N(t)$ as a function of t in the following way. First generate the jump points of the Poisson process. Second, we know that a Poisson process can only jump one unit at a time. Thus, we can define the Poisson process as a step function that jumps exactly one unit at every jump point.

1. Generating jump times

This is an algorithm for generating the jump times of a Poisson process

1. Set $\tau_0 = 0$.
2. For $k \geq 1$ do
   a) generate $X \sim\text{Exp}(\lambda)$
   b) set $\tau_k = \tau_{k-1} + X$

The $\tau_k,\ k = 0, 1, \dots$ are the time points, where the Poisson process jumps.

Numerical implementation: R code

```
# set the seed for generating random variables (for reproducability)
# set.seed(2019)

# set the rate of the Poisson process/Exponential distribution
lambda <- 2
# number of jumps
n <- 50
# initialise the jump times
```

```
jump_times <- rep(0, lenthought = n)

# note that the first value of the jump_times is aleady 0
for(i in 2:n){
  jump_times[i] <- jump_times[i - 1] + rexp(1, lambda)
}
jump_times
```

```
##  [1]  0.0000000  0.5444887  0.5454424  0.8809832  1.4505266  1.6869048
##  [7]  1.7059031  1.8607671  2.0729266  2.2589896  2.5371803  2.5877545
## [13]  2.6749528  2.8895166  3.2522176  3.6014219  4.5135901  4.7005667
## [19]  5.2435343  5.8228627  6.5839718  7.7345873  7.7684337  8.7862417
## [25]  9.5079461 10.9345292 11.3409176 11.9732488 14.2617515 14.7938901
## [31] 15.8470452 16.3802866 16.5961629 17.3402180 17.5102764 18.0994127
## [37] 18.7045654 19.8956381 20.5781453 21.2622799 22.5534571 22.8030085
## [43] 23.3603107 23.4186866 23.4839794 24.1003521 25.9850091 26.5601430
## [49] 27.3181185 27.3528258
```

A more efficient implementation, avoiding the for loop, is

```
jump_times_2 = c(0, cumsum(rexp(n-1, lambda)))
# note that the two generated sample_paths are different,
# since the Exponential random variables are different.
jump_times_2
```

```
##  [1]  0.0000000  0.3539839  0.7162121  0.7826041  1.8028482  2.1987581
##  [7]  2.8221569  3.2853485  3.9301713  4.3394103  4.6710097  4.8448800
## [13]  4.8702180  4.9519977  5.1722801  5.4666618  5.9636471  6.3157308
## [19]  6.9063829  7.0085068  7.6099604  8.5552378  8.7247318  9.4616709
## [25]  9.5507656  9.6876916 10.6696000 10.9187058 11.3672931 11.3914625
## [31] 11.5488119 12.2686591 12.3355676 12.6426100 12.6899426 12.7803271
## [37] 13.1920945 13.5216800 13.8321773 15.1126203 15.4269006 17.8920100
## [43] 18.2823616 18.3507779 18.4745869 20.2826124 20.6589388 21.0927695
## [49] 21.1050071 22.0848946
```

2. Defining the Poisson process as a function

Next, we can define the Poisson process as a step function that jumps exactly one unit at every value in `jump_times`. Note that the implemented Poisson process is now a *function* of the time t.

```
# define the Poisson process as a function
poisson_process <- stepfun(jump_times[2 : 50], seq(0, n - 1, by = 1))
# the first argument of `stepfun` is where the step function jumps
# the second argument is the hight of the step function

# Let us check some values.
# What is the value of N(3)? (Note that we use jump_times)
poisson_process(3)
```

```
## [1] 13
```

```
poisson_process(10)
```

```
## [1] 24
```

```
poisson_process(15)
```

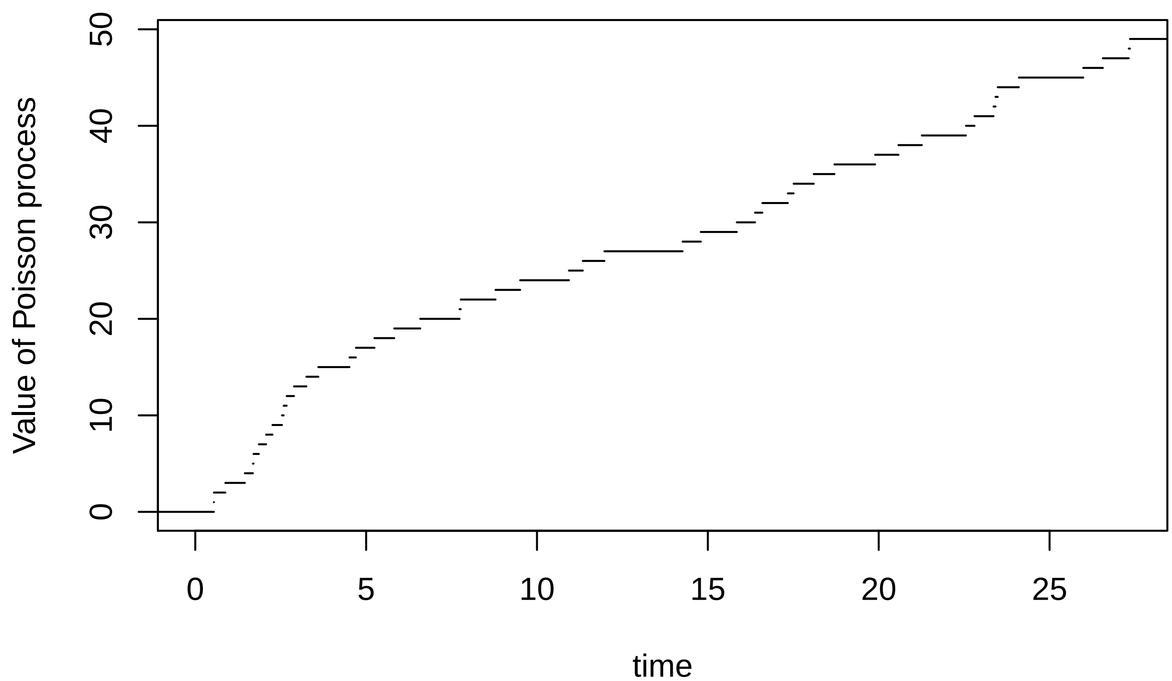
```
## [1] 29
```

Note you can evaluate the Poisson process for any time point t, however, keep in mind that we only simulated 50 jumps of the Poisson process. Thus, evaluating `N(t)` for t large does not make sense.

3. Plotting a sample path

To plot a sample path we have to plot the implemented step function `poisson_process`.

```
plot(poisson_process, xlab = "time", ylab = "Value of Poisson process", main = NULL,
     verticals = FALSE, do.points = FALSE, xlim = c(0,jump_times[n]))
```



Question: Rerun the code, what do you observe?
Question: Rerun the code with a different λ or change the number of jumps. How does the sample path change?
Question: What happens if you replace the Exponential random variables in the definition of the Poisson process with other i.i.d. random variables? For example, Gamma, Normal or LogNormal?
Rerun the code snippets changing `rexp` to `rgamma`, `rnorm` or `rlnorm`.

Note that for random variables that are not Exponentially distributed you still get a counting process. These processes are called *Renewal processes*.