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

```r
# 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  0.4767655  1.0581041  1.8578236  3.2559489  3.5394439
##  [7]  3.9906094  4.1307497  4.2866726  4.3992743  5.0108379  6.0654886
## [13]  6.2149066  6.2303050  7.1849729  8.1933875  8.7753199  8.9484147
## [19]  9.2087433  9.3263074  9.4383098  9.7045452 10.8761676 11.3973042
## [25] 11.4074462 11.9643543 12.3000737 12.3072904 12.4983924 13.7908779
## [31] 14.6747610 14.6754677 14.8844742 14.9548258 15.4500550 15.5023878
## [37] 15.7508640 16.2612631 16.4405782 16.7284608 17.3694169 17.7220491
## [43] 18.4199368 18.6186752 19.1562800 19.1664464 19.5583147 19.9685487
## [49] 20.8510205 20.9302502
```

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  0.1142011  0.4496699  1.7086030  2.0097590  2.5269483
##  [7]  2.7085649  3.3147742  3.3952648  3.8675995  5.1157780  5.4036021
## [13]  8.1685501  8.1760947  8.5146537  9.1459960  9.3345537  9.5853237
## [19] 10.1212488 10.1565996 10.3555413 10.3651231 10.6938204 11.5283052
## [25] 11.9005555 13.0766322 13.5692754 13.6825197 14.4026515 14.6386976
## [31] 14.6576369 14.9392504 15.0516467 15.0614125 15.6172854 15.7309394
## [37] 16.3628387 16.7120088 16.8609103 17.0117249 17.0678618 17.7134498
## [43] 17.7559488 18.5740632 19.5641118 19.7950294 19.8895788 20.5698014
## [49] 20.5940748 20.6303171
```

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, seq(0, n, 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] 4
```

```
poisson_process(10)
```

```
## [1] 22
```

```
poisson_process(15)
```

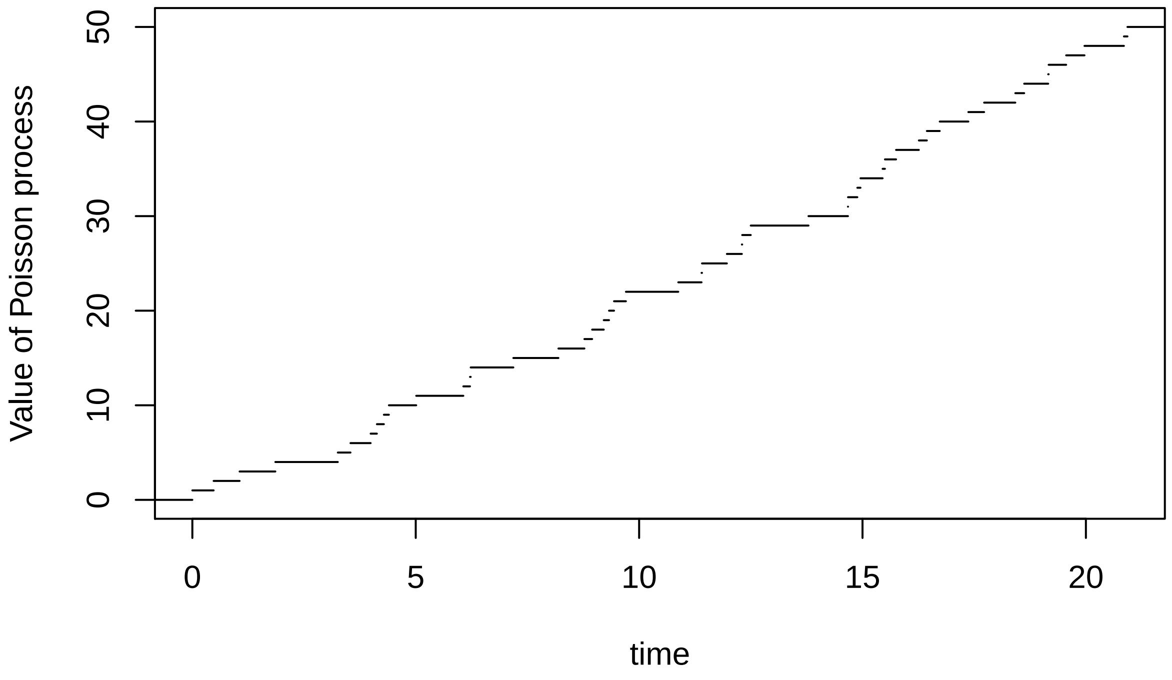
```
## [1] 34
```

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*.