Homework 2: Pseudo Random Number Generators

Nicole Zattarin

Abstract

Random Number Generators (RNG) are a fundamental, yet quite delicate aspect of simulation. RNGs are necessary in most of scientific simulations to generate random variables which correspond to the real event that we aim to simulate. Nevertheless, it is impossible to generate sequences of real random numbers without exploiting an external source of randomness, e.g. Quantum RNGs, because the behaviour of our algorithms is essentially deterministic. Therefore, we refer to random numbers generators based on an algorithmic approach as Pseudo RNGs. In this report we explore both all the limitations connected to the generation of pseudo random numbers and different methods to design PRNGs.

1. Linear congruence

Let us consider a first example of PRNG: given a seed s and two parameters a and m, computes a sequence of pseudo random number as follows:

$$x_n = \frac{sa^n \mod m}{m}, \tag{1}$$

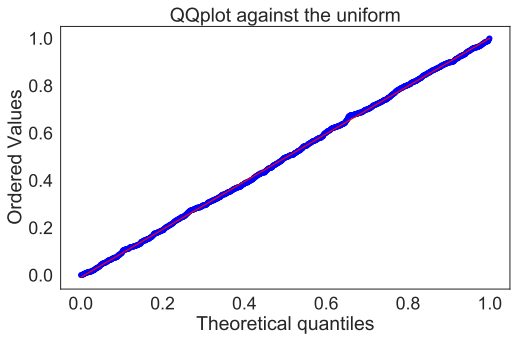
where the period depends on the parameters involved.

1.1. Independent sample

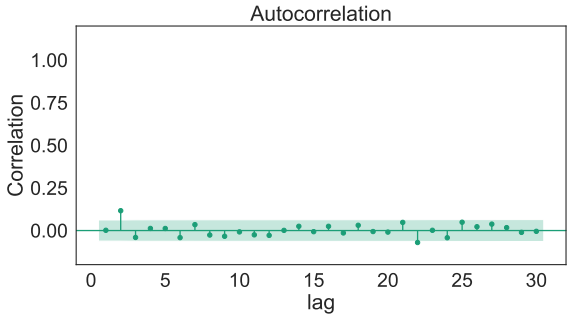
In order to test it, we generate 1000 successive numbers with parameters $a = 16807,\ m = 2^{31} - 1$, results are shown in Figure 1. Figure 1a shows the QQplot against a uniform distribution, from which we deduce that the PRNG provides a uniform sample. Coherently the autocorrelation plot, Figure 1b, shows that there are not significant correlations, independence is proved also by lag plots, see Figure 1c.

1.2. Seeding PRNGs

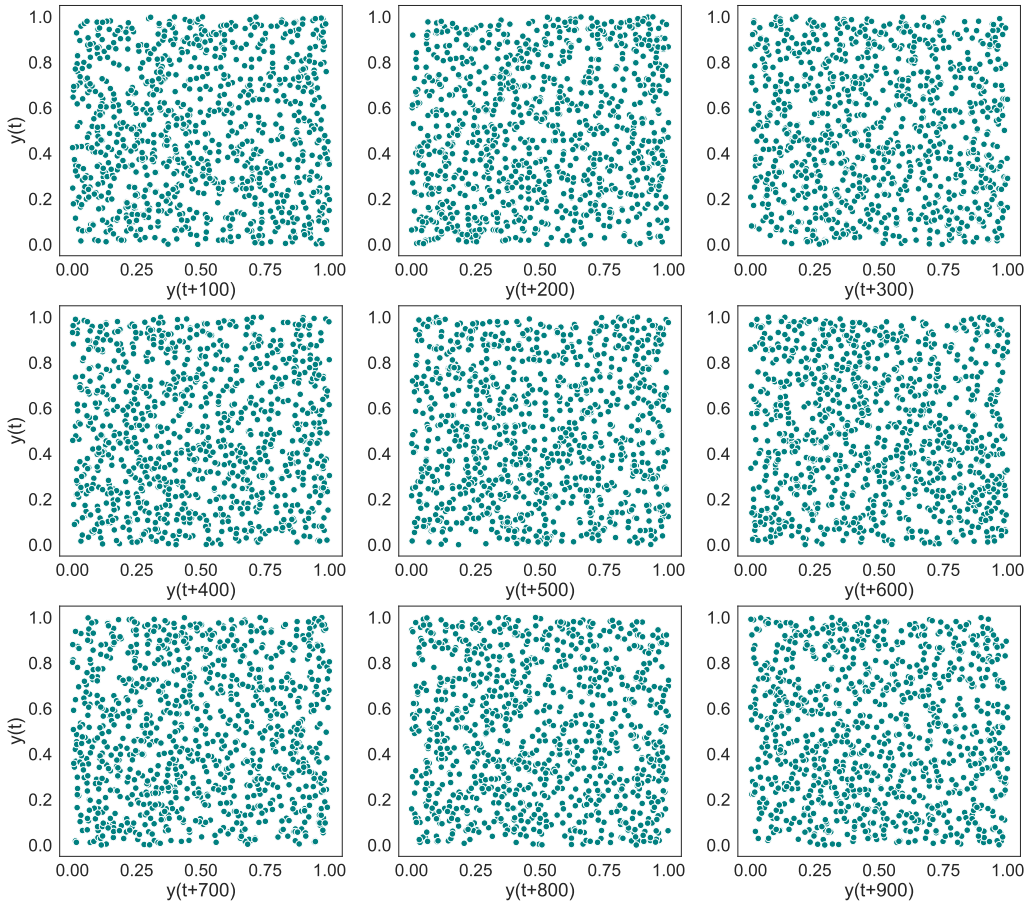
An important point in generating random numbers is choosing a proper seed, in order to avoid the generation of periodic data and to have control over the simulation. For instance, if we pick the last element of a first sample as seed for the next sample, we are sure that we won't generate a sequence with common elements, at least before reaching the periodicity. To show the consequences of such choice we generate three samples of data: one with seed 1, one with seed 2, one with seed equal to the last element of the first sequence. In Figure 2, the left panel shows sequence with seed 1 vs seed 2, in the right panel $seed = 0.6628$ is the last value of the sequence generated with seed 1. It is clear that the first choice produces samples which are correlated, while in the second case we observe no correlation.



(a) *The QQplot against the uniform distribution in (0, 1) shows match.*



(b) *Autocorrelation plot does not highlight any correlation for all the lag considered.*



(c) *Lag plots show that the sample is indeed independent.*

Figure 1: *Sample of 1000 successive numbers uniformly distributed in* $(0, 1)$ *with* $a = 16807$, $m = 2^{31} - 1$.

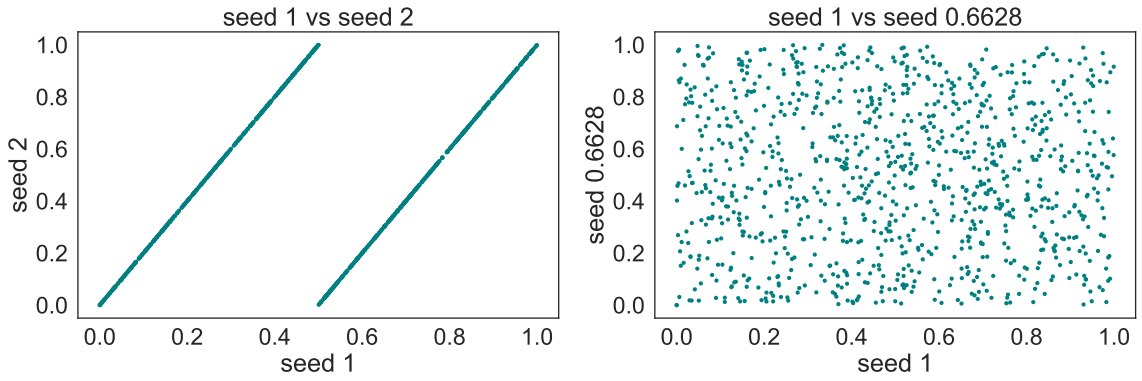


Figure 2: *Comparison of two samples generated with the same LCGs, but different seeds. The left panel shows results for seed 1 vs seed 2, the right panel shows seed vs a sequence generated with seed equal to last value of first stream.*

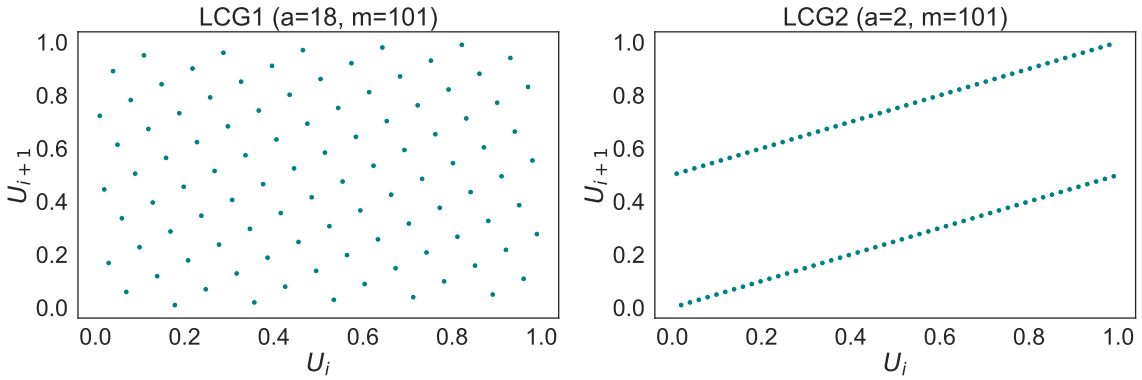


Figure 3: *Scatterplot of U_i vs $U_i + 1$ for different LCGS. The left panel shows that the LCG with a=18 and m=101 exhibits a regular pattern. The right panel shows the correlation between consequent elements in a LCG with a=2 and m=101.*

1.3. Periodicity

As we already pointed out, a problem related with LCGs is that these are characterized by a periodicity. The period can be at most m, but in general it depends on the combination of parameters, we refer to full period LCGs if the period $T = m$.

Let us consider a first example of LCG with $a = 18$ and $m = 101$ and another one with $a = 2$ and $m = 101$. We can manually deduce if these are full period or not by going through m elements of the samples and check if the value has already been observed. We conclude that both of them are full period with period $m = 101$. To draw conclusion regarding the generation of pseudo random numbers with such tools, in Figure 3 we show all pairs (U_i, U_{i+1}) for both of them. It is possible to observe that the samples exhibit a regular behaviour with some clear patterns. The different arrangement of samples has to do with the value of a, since the parameter m is shared among the two. Moreover, both of the LCGs are full period, thus plotting the first m values we are displaying the entire space that can be generated from such generators. Indeed, the main issue that is possible to highlight from the plots is that the sequences are not independent.

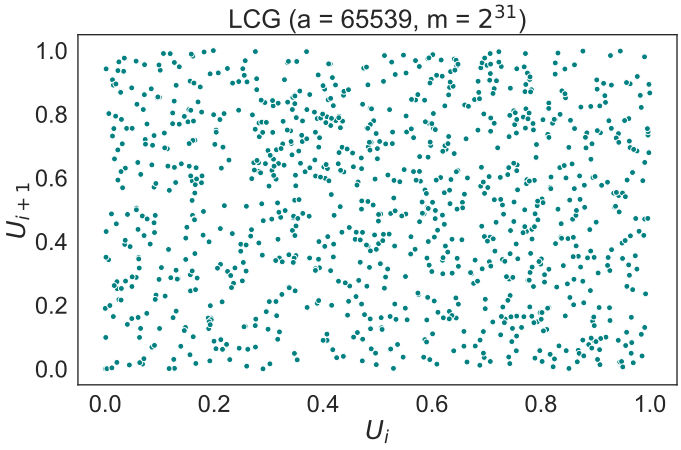


Figure 4: *Scatterplot of U_i vs $U_i + 1$ for a LCG with $a = 65539$ and $m = 2^{31}$, it shows that the sample is independent.*

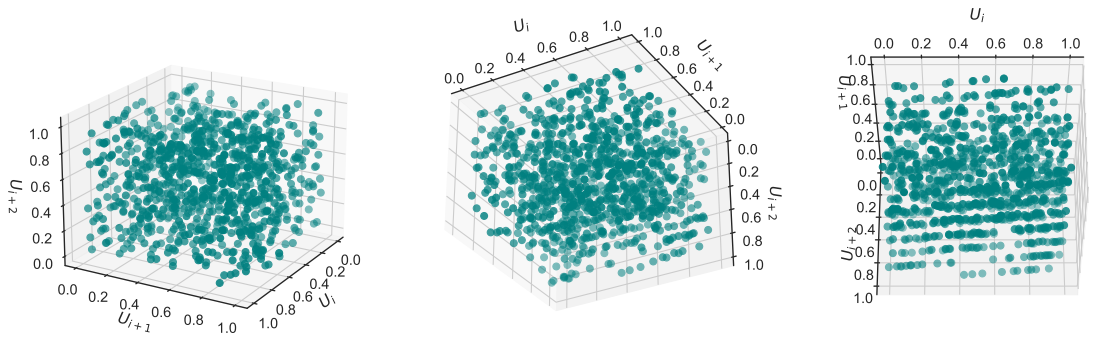


Figure 5: *3-dimensional representation of the sample generated with LCG with $a = 65539$ and $m = 2^{31}$, points are the triples (U_i, U_{i+1}, U_{i+2}). It is possible to highlight that dependencies arise once one consider the third dimension.*

Let us now consider another example, with $a = 65539$ and $m = 2^{31}$. In Figure 4 we show the pairs (U_i, U_{i+1}): from a two dimensional representation of data we would conclude that the sample consists of independent and uniformly distributed elements. Nevertheless, in Figure 5, a 3-dimensional plot shows that, once we consider triples (U_i, U_{i+1}, U_{i+2}), dependencies arise.

2. Generic distribution

Since we already discussed methods and problems related to the generation of uniform random numbers, we now aim to exploit such observations to generate specific distributions. To this end, many approaches can be exploited, often based on the generation of a uniform value in $(0, 1)$. here we propose a strategy based on rejection sampling.

Let us consider a sample Y, which takes values in a finite interval $[a, b]$. If its density is given by $f_Y = K\, f_Y^n$, where K in un unknown normalization constant, thus f_Y^n is the non normalized density, that we assume to be easy to compute. Let us also consider the case in which it is possible to find a bound $M : f_Y^n < M$. The algorithm 1 shows the process in the 1-dimensional case, the

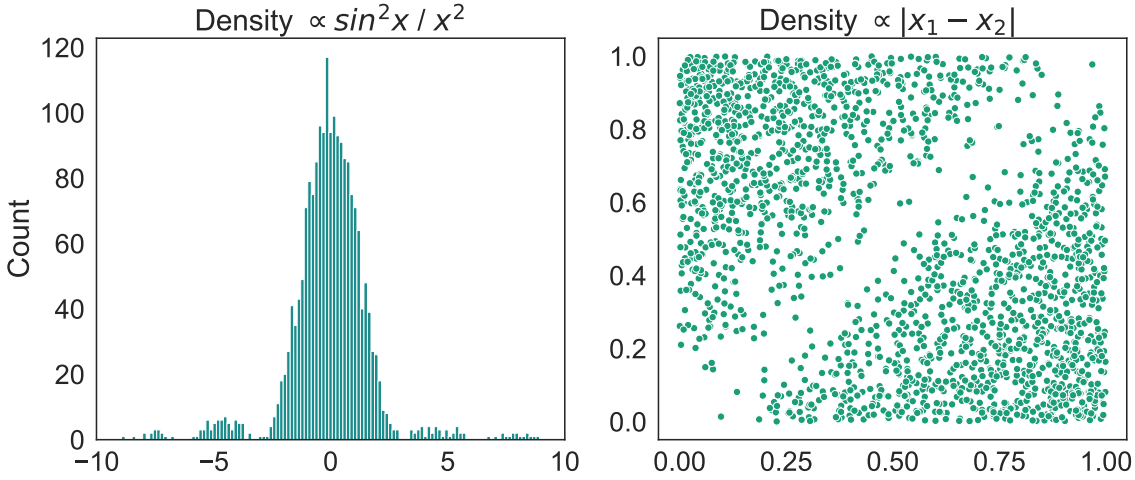


Figure 6: *Generated samples with arbitrary distribution. On the left panel we show 2000 samples of the distribution with* $f_Y^n = \frac{sin^2 x}{x^2}$*, while in the right panel we show the 2-dimensional distribution with* $f_{Y_1,Y_2}^n = |Y_2 - Y_1|$

generalization to the 2-dimensional is trivial.

Algorithm 1 Generation of RV with arbitrary distribution

Take $U \sim (0,1)$, $y \sim (0, M)$
while $U \geq f_Y^n$ **do**
  Take $U \sim (0,1)$, $y \sim (0, M)$
end while
return Y

In Figure 6 we show two examples, in the left panel we provide the histogram of 2000 samples of the distribution with $f_Y^n = \frac{sin^2 x}{x^2}$, while in the right panel we show the 2-dimensional distribution with $f_{Y_1,Y_2}^n = |Y_2 - Y_1|$. Note that this method allows to generate a sequence without knowing the value of the normalization constant M.

3. Discrete random variables

In this section we explore two specific discrete distributions: binomial and the poisson RVs.

3.1. Binomial distribution

A powerful approach to generate a specific distribution of RV is represented by CDF inverse method: the idea is that by generating samples uniformly in $(0,1)$ we are able to reconstruct a generic distribution F by exploiting the properties of the cumulative density function. Formally, given $U \sim \mathcal{U}(0,1)$, then $\forall F$ continuous distribution function, the RV X defined as $X = F^{-1}(U)$ is distributed as F. Such result can be extended to the discrete case and it can be applied efficiently to the generation of binomial samples because of the specific relationship that occurs between $B(n,k)$ and $B(n,k+1)$, the pseudo code is reported in Algorithm 2.

Algorithm 2 Generation of binomial RV

Take U uniformly in $(0, 1)$
$c = \frac{p}{1-p}$
$i = 0$
$pr = (1-p)^n$
$F = pr$
while U$\geq F$ **do**
  $pr = pr\ c(n-i)/(i+1)$
  $F = F + pr$
  $i = i + 1$
end while
$X = i$
return X

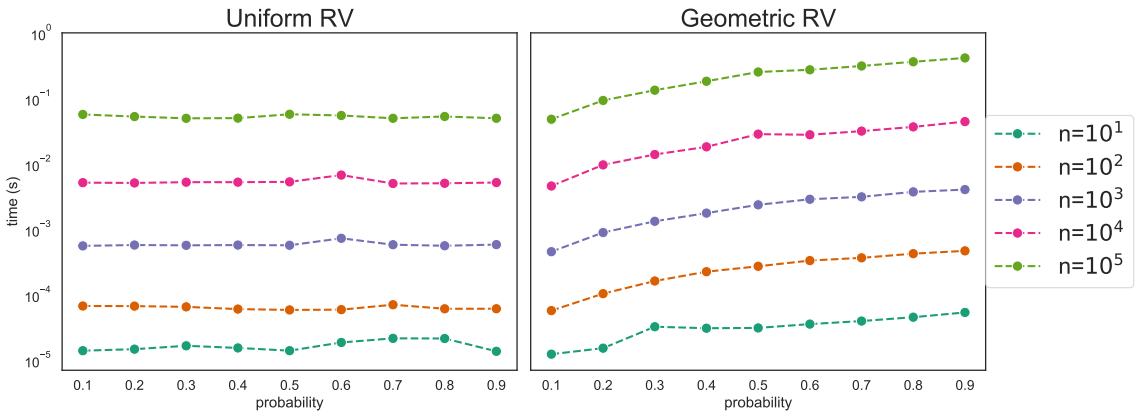


Figure 7: *Time necessary to generate a binomial RV with the method that exploits uniform distribution (left panel) and geometric distribution (right panel). We provide results in logarithmic scale, for different values of n, as a function of p.*

Another method to generate binomial RV with parameters n, p is to pick a value $u \sim \mathcal{U}(0, 1)$ n times, and count the number k of successes of the event $u < p$. k corresponds indeed to the definition of binomial RV.

Finally, we could also consider a binomial process as a sequence of extractions of geometric RV. Indeed, a geometric RV with parameter p represents the number of trials one has to go through before observing a successful event. We can then count the number k of geometric RV that it is necessary to generate in order to collect n trials. Still, k is a binomial RV.

To test the performances or our generator, we generate 100 RV for different values of n and p, compute the time for each generation and take the average over N. In Figure 7 we show the result, in logarithmic scale, for different values of n and p. It is possible to conclude that the time needed to sample RV with the method based on uniform RV increases with n, while it is constant with p. We would expect such result since time is connected only to the number of uniform RV that must be extracted to generate a single binomial variable, n, but not on the probability, which is used only to check if the event occurs. On the other hand, using geometric RV time increases both with n and p. This behaviour can be explained if we consider that the bigger n, the more geometric RV we have to generate, moreover if p increases, the sequence of trials becomes shorter thus we need

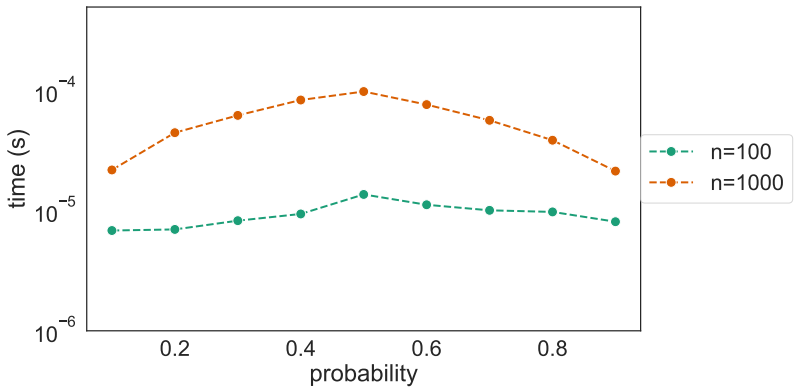


Figure 8: *Time necessary to generate a binomial RV with CDF method.*

to generate more variables as well. Finally, in Figure 8 we provide, for two values of n, the time necessary to generate our sequence of data with CDF method. In this case we extended Algorithm 2, by exploiting a backward approach to address probabilities greater than 0.5. It turns out, as we expect, that the worst case scenario corresponds to $p = 0.5$, since it is the furthest point both starting from zero and n successes. Moreover, we observe that time increases with n as expected.

3.2. Poisson distribution

The CDF method is generic, thus it can also be exploited in order to generate poisson RV, in Algorithm 3 we provide the adaption of CDF algorithm to the generation of poisson RV.

Algorithm 3 Generation of Poisson RV

Take U uniformly in $(0, 1)$
$i = 0$
$p = e^{-\lambda}$
$F = p$
while U$\geq$ F **do**
  $p = p\lambda(i + 1)$
  $F = F + p$
  $i = i + 1$
end while
$X = i$
return X

Nevertheless, it is possible to design other methods to generate poisson RVs, which are based on the generation of exponential RVs, since the distribution of interarrival times in a poisson process is exponential. A poisson RV represents indeed the number of occurrences of an event, i.e. arrivals, in a given interval of time. As a consequence, for a unitary interval, a poisson RV is given by:

$$N = \max_n \{n : \sum_i^n x_i \leq 1\}, \tag{2}$$

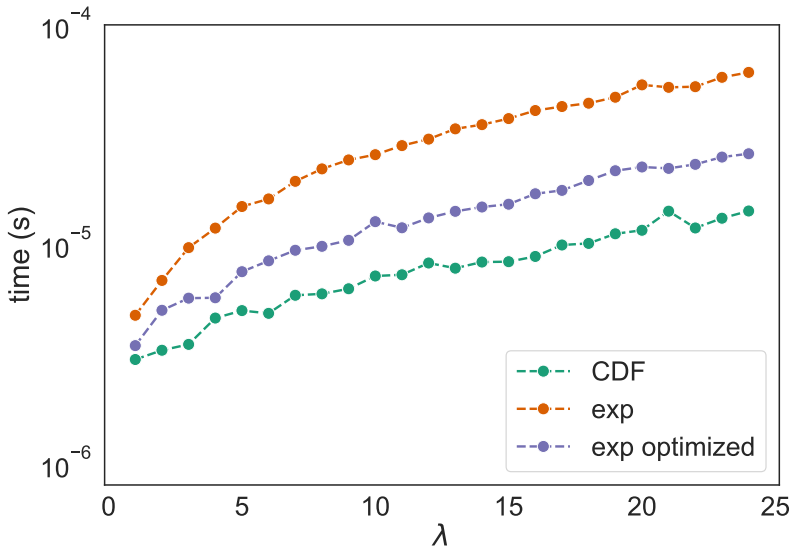


Figure 9: *Time necessary to generate a poisson RV with the methos discussed. We provide results in logarithmic scale, as a function of* λ.

where x_i has exponential distribution with parameter λ, thus it can be generated as follows:

$$x_i = \frac{\log u}{\lambda}, \quad u \sim U(0,1). \tag{3}$$

Finally, the previous method can be optimized by rewriting the previous formula as:

$$N = \max_n \{n : \prod_i^n u_i \leq e^{-\lambda}\}, \quad u_i \sim U(0,1). \tag{4}$$

To test the performances, as we did with the binomial distribution, we generate 100 RV for different values of , compute the time for each generation and take the average over N. In Figure 9 we show results of the simulation: it is possible to observe that time increases with λ for all the considered methods. In particular, the most efficient method is the CDF based, while exponential RV generation requires more time.