Network Analysis and Simulation - Homework 2

Michele Polese, 1100877

April 18, 2015

1 Exercise 1

A *Linear Congruential Generator* (LCG) is a pseudorandom number generator, characterized by the parameters a, c, m, x_0. Generally the sequence $\{x_n\}$ of random numbers is generated by iterating: $x_n = (ax_{n-1} + c)\text{mod}(m)$ and x_0 is the starting seed. If $c = 0$ then the LCG is a multiplicative LCG and the maximum period of the sequence is $m - 1$ because $x_n = 0$ would be a standpoint for the generator and it is never reached, unless $x_0 = 0$ (but this would be a very bad choice). Figure 1 shows in different ways the randomness of a $U[0, 1]$ sequence generated with a LCG by normalizing the x_n sequence: by comparing with a $U[0, 1]$ generated by MATLAB Mersenne Twister rng and by showing the lack of correlation between samples with the autocorrelation function and lag plots. The parameters of this LCG are $a = 16807, m = 2^{31} - 1, c = 0$.
A LCG however must be handled carefully when dealing with parallel streams. In Figure 2 there are two lag plots at lag 1 which show that the behavior of a LCG depends on the initial seed. If the two seeds depend one on the other or are not randomly chosen, for example with an hardware random number generator, then there's a strong correlation between the two streams. This happens in the first plot of Figure 2 where $x_0^{\text{LGC}_1} = 1$ and $x_0^{\text{LGC}_2} = 2$, therefore up to the wrap around $x_i^{\text{LGC}_2} = 2x_i^{\text{LGC}_1}$. Instead, if the seed of the second stream is the last element of the first sequence and the total number of samples generated doesn't exceed the period of the LCG then the two sequences are uncorrelated, as in Figure 2b. More details on LCGs will be given in Sections 4 and 5.
In Figure 3 there are two distributions generated with rejection sampling. This technique allows to compute a random variable with a certain density distribution which is not completely known (i.e. missing normalization factor) by comparing uniform random variables with the expected values. Figure 3a plots the empirical histogram of 2000 samples drawn from a distribution with pdf $f_y(y) = K\frac{\sin^2(y)}{y^2}\mathbf{1}_{-a\leq y\leq a}$ with $a = 10$ and K an unknown normalization factor, which is hard to compute analytically. It is easier to compute a bound M on the non normalized density pdf $f_y^n(y) = \frac{\sin^2(y)}{y^2}\mathbf{1}_{-a\leq y\leq a}$, which is $M = 1$, and then draw $X \sim U[-a, a], U \sim U[0, M]$ and compare U with $f_y^n(X)$ until $U \leq f_y^n(X)$. In Figure 3b, instead, there are 2000 samples of a random vector (X_1, X_2) whose distribution is in the unit square $[0, 1] \times [0, 1]$ and has a density proportional to $|X_1 - X_2|$. It is possible to easily generate samples of this distribution by generating 3 uniforms U, X_1, X_2 in $[0, 1]$ until $U \leq |X_1 - X_2|$.

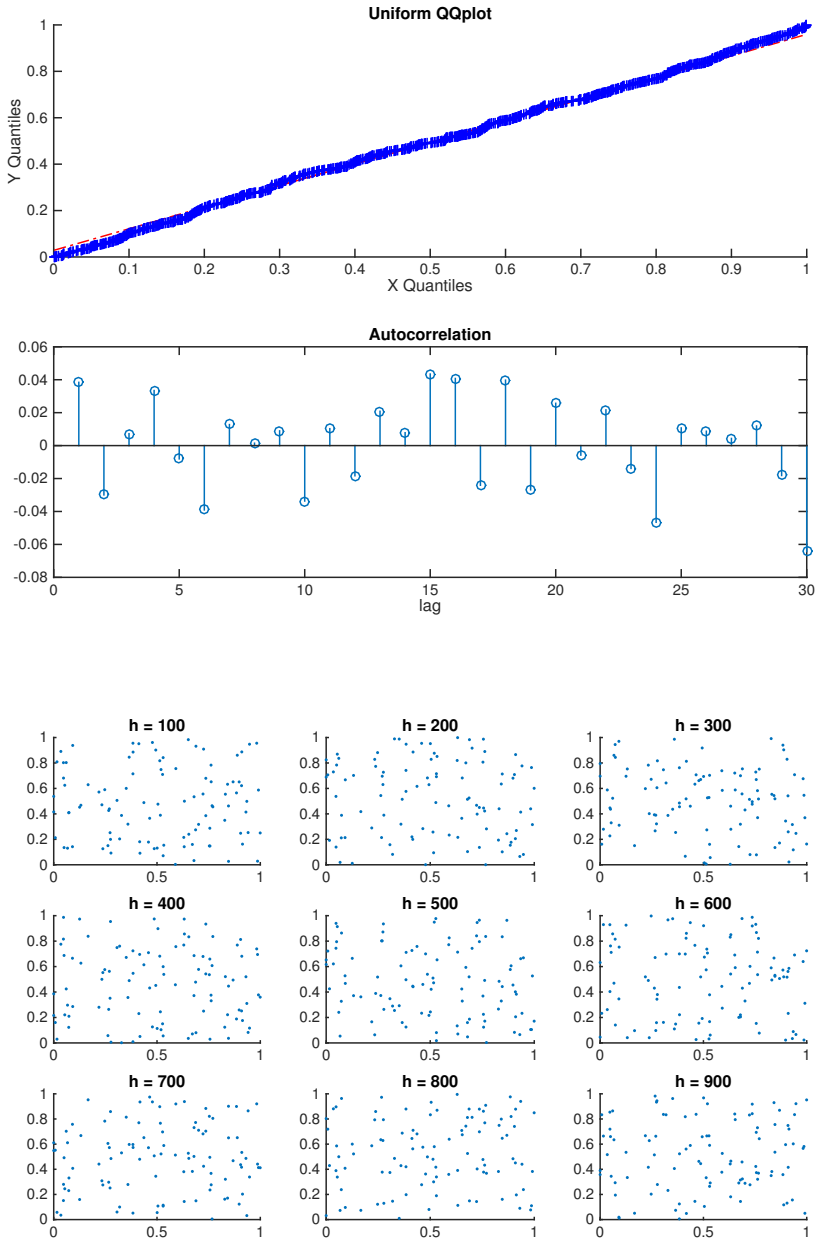
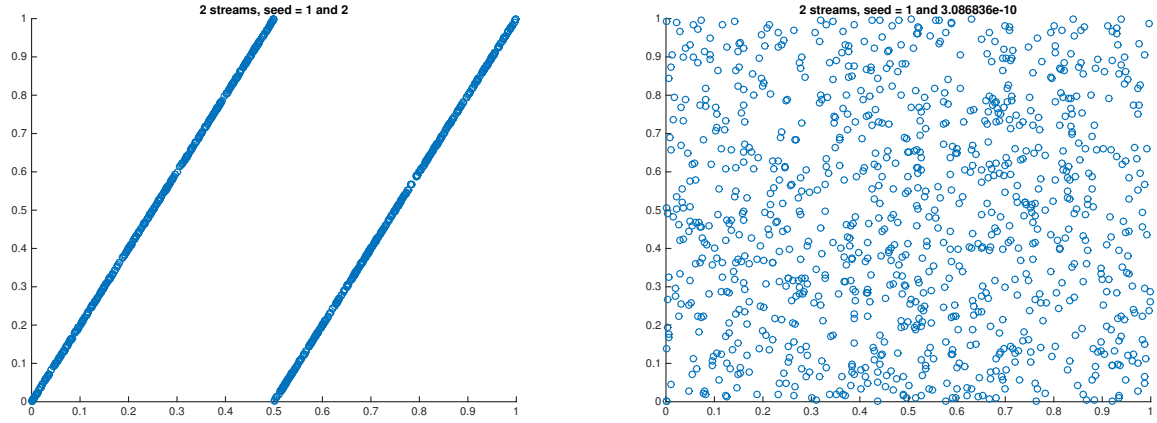
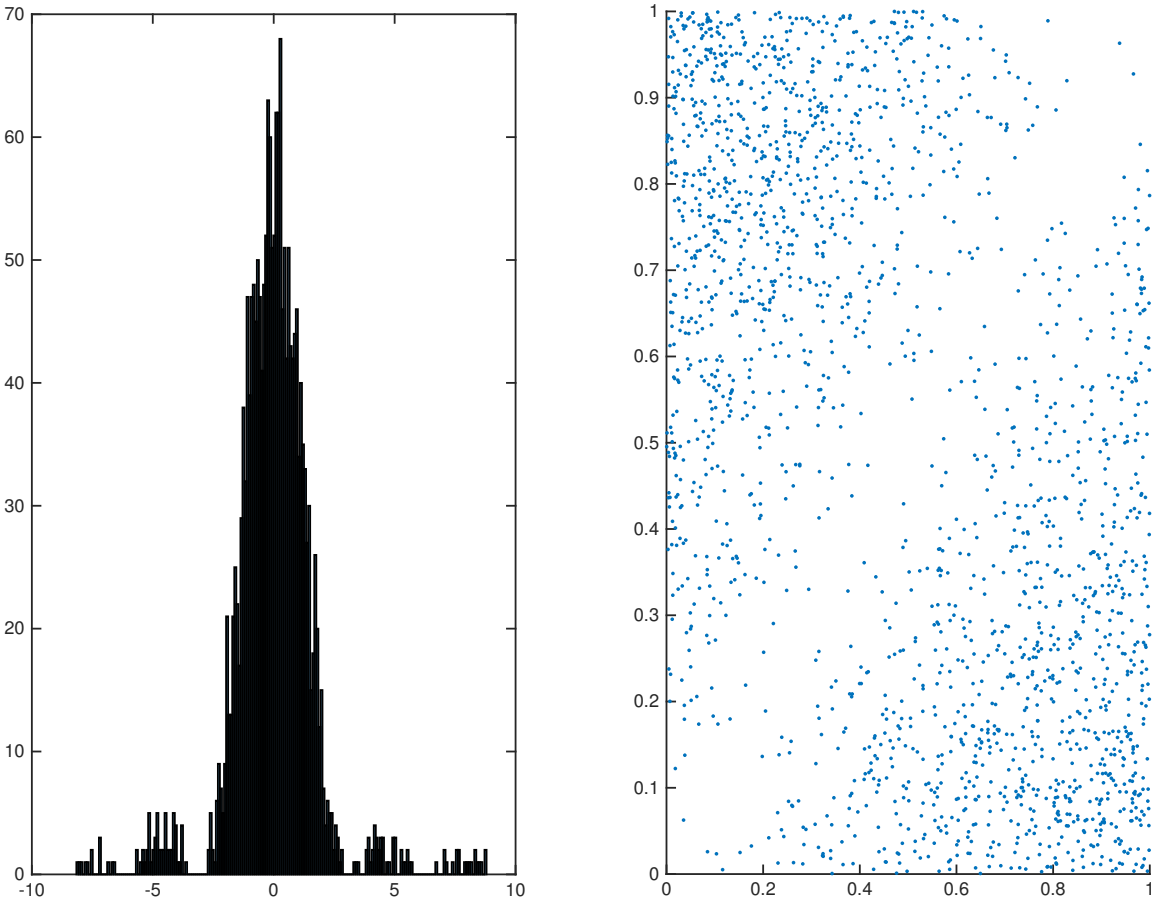


Figure 1: Figure 6.5 in [1]: analysis of the randomness of LCG with $a = 16807, m = 2^{31} - 1, c = 0$



(a) (b)

Figure 2: Figure 6.7 in [1]



(a) (b)

Figure 3: Figure 6.10 in [1]

2 Exercise 2

A Binomial random variable ($\text{Bin}(n,p)$) can be generated in three different ways. The first method is the CDF inversion, which is performed in an iterative way. The CDF of a $\text{Bin}(n,p)$ is $F(r) = \sum_{k=0}^{r} \frac{n!}{(n-k)!k!}(1-p)^{n-k}p^k$ and it cannot be inverted in a close form, but it is possible to compute the inverse in an iterative way with the following algorithm:

Algorithm 1 CDF inversion for $\text{Bin}(n,p)$

```
procedure
    Let U be a number, generated from a U[0,1] distribution
    Let X = 0, pr = (1-p)^n, F = pr, i = 0
    while U >= F do
        X = X + 1
        pr = (n-i)/(i+1) * p/(1-p) * pr
        F = F + pr
        i = i + 1
    return X
```

The second algorithm exploits the nature of the binomial distribution, which represents the number of success in n Bernoulli trials with probability of success p. Therefore

Algorithm 2 Generation of a $\text{Bin}(n,p)$ with n Bernoulli trials

```
procedure
    Let X = 0, i = 1
    while i <= n do
        Let U be a number, generated from a U[0,1] distribution
        if U <= p then
            X = X + 1
        i = i + 1
    return X
```

A variant of this method involves the generation of strings of 0 (unsuccessful Bernoulli trials with $P_{\text{succ}} = p$) followed by a 1, which is the first successful Bernoulli trial. These strings are distributed according to a geometric random variable $G(p)$. A geometric random variable can be generated with CDF inversion in a closed form, using $G = \lfloor \frac{\log(U)}{\log(1-p)} \rfloor$ with U a uniform sample in $[0,1]$. Thus

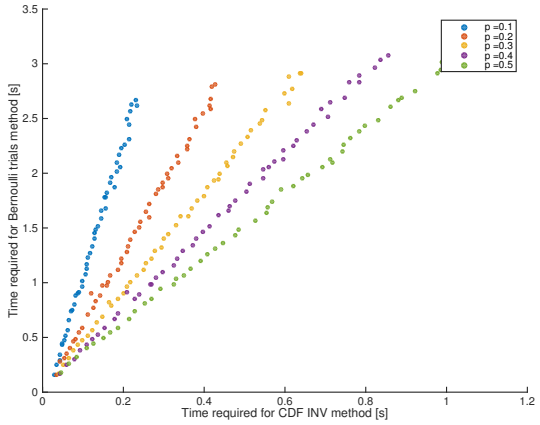
Algorithm 3 Generation of a $\text{Bin}(n,p)$ with geometric strings of 0

```
procedure
    Let X = 0
    Let U be a number, generated from a U[0,1] distribution
    Let G = ⌊log(U)/log(1-p)⌋ the length of a string of zeros
    Let i = G + 1 a string of G zeros and a 1
    while i <= n do
        X = X + 1
        Let U be a number, generated from a U[0,1] distribution
        Let G = ⌊log(U)/log(1-p)⌋ the length of a string of zeros
        i = i + G + 1
    return X
```

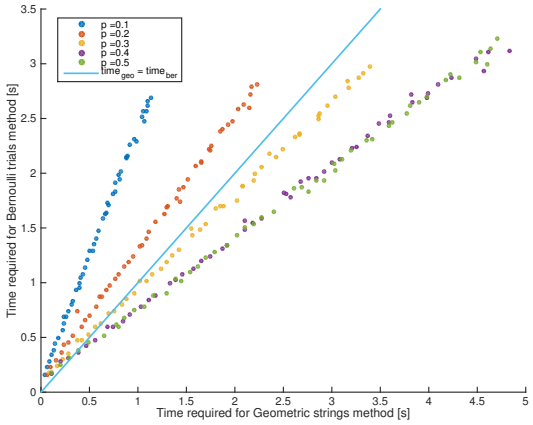
The algorithms are implemented in the attached MATLAB code in order to compare their performances. Note that the average number of iterations that Algorithm 1 has to perform is one more than the value of the random variable it generates, so on average $1 + np$. Algorithm 2 instead performs always n iterations. Algorithm 3 has a complexity which is proportional to np too, since on average the strings have $\frac{1-p}{p}$ zeros and a 1, so their length is $1/p$: it follows that the number of iterations needed to reach n are on average $\frac{n}{1/p} = np$ (the comparisons are $1 + np$).
The relation between the three methods can be seen in Figure 4, where a comparison between the execution time of the three method is plotted. The number of variates generated for every pair of (n,p) is $N = 10^5$, $n \in [20, 10^3], p \in$

$[0.1, 0.2, 0.3, 0.4, 0.5]$. CDF inversion and geometric method should perform approximately in the same way. Actually when there are many iterations the complex operations that Algorithm 3 has to perform in each iteration (a logarithm, a division, an extraction of random number) make it slower that the simple CDF inversion. Figure 4c has on x and y axis the time needed to generate 10^5 variates with CDF inversion and geometric strings, respectively (each point represents the time to generate 10^5 $\text{Bin}(n, p)$ with the same n and p). It can be seen that the two methods have a linear dependance, which means that they share the same complexity (as expected) but the time required to execute each iteration differs by a constant. Instead when np is small and the number of iterations is on average lower than 1 then the two methods perform approximately in constant time. This can be seen in Figure 6 where it is plotted the time required to generate 10^5 binomial random variables as a function of $n \in [20, 10^4]$ (increased by a step of 20), with $p \in [10^{-9}, 10^{-8}, 10^{-7}, 10^{-6}, 10^{-5}]$ and in Figure 5 where the execution times of the two methods are plotted one against each other. In this case the geometric strings method has a weak dependence on p, due to the generation of the geometric random variable, and performs slightly better than the CDF inversion. Note also that the CDF inversion method is based on iterations, thus the values it computes are subject to approximation errors. Moreover it must be taken into account the limit of the finite precision of a computer, and since the lowest positive number which can be represented in MATLAB is $\delta = 4.9407e-324$ then for values of n and p such that $(1-p)^n < \delta$ the CDF inversion cannot be performed.

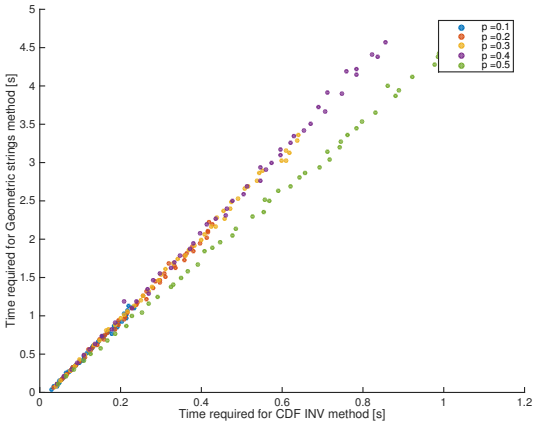
Another observation is that despite the dependance on n and not on np of the Bernoulli strings method there are some cases in which it performs better than the geometric strings method. Indeed if in principle the number of iterations of the second method is smaller, the complexity of each of them is higher, therefore the Bernoulli becomes faster, as it can be seen in Figure 4b and in Figure 7. The closer the p is to 0.5 the smaller is the n which makes the Bernoulli method perform faster than the geometric method, for example for $p = 3$ then $n = 186$, $p = 0.4$ then $n = 84$.



(a) Bernoulli trials vs CDF inversion



(b) Bernoulli trials vs geometric strings



(c) Geometric strings vs CDF inversion

Figure 4: Comparison between time required to generate $N = 10^5$ binomial rv, $n \in [20, 10^3]$, $p \in [0.1, 0.2, 0.3, 0.4, 0.5]$

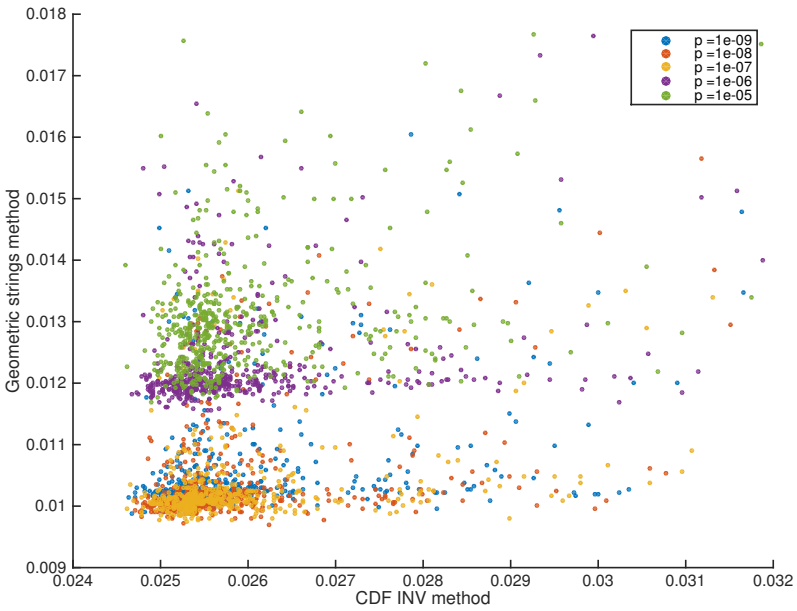
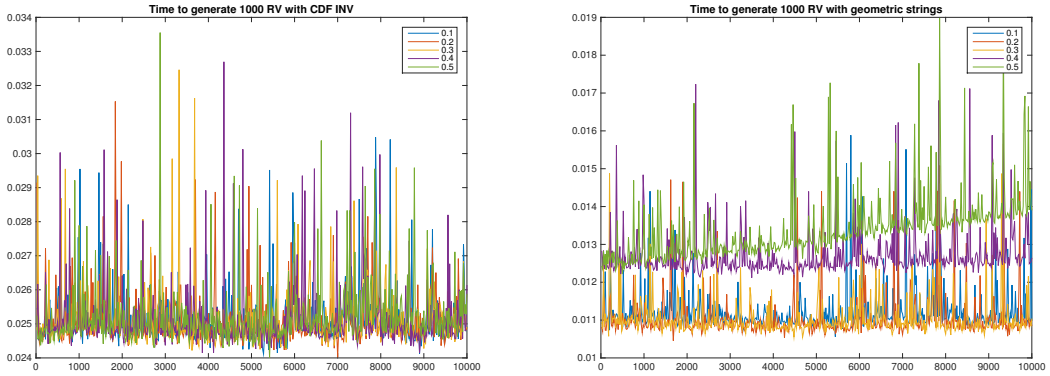


Figure 5: Geometric strings vs CDF inversion methods' execution time for $np < 1$, $N = 10^5$, $n \in [20, 10^4]$ (increased by a step of 20) and $p \in [10^{-9}, 10^{-8}, 10^{-7}, 10^{-6}, 10^{-5}]$



(a) CDF inversion

(b) Geometric strings

Figure 6: Execution time as a function of n for $np < 1$, $N = 10^5$, $n \in [20, 10^4]$ (increased by a step of 20) and $p \in [10^{-9}, 10^{-8}, 10^{-7}, 10^{-6}, 10^{-5}]$

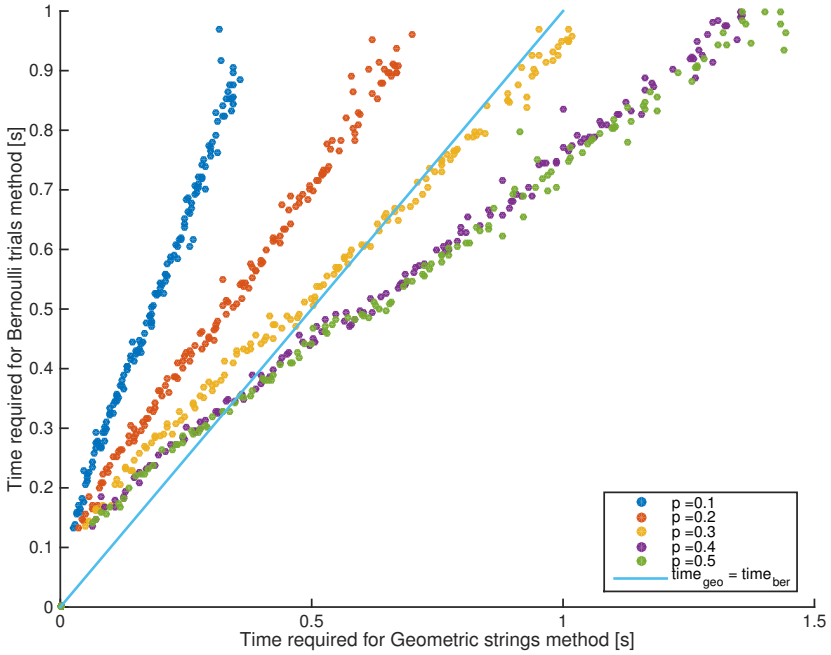


Figure 7: Bernoulli trials vs geometric strings methods' execution time, $N = 10^5, n \in [10, 300], p \in [0.1, 0.2, 0.3, 0.4, 0.5]$

3 Exercise 3

A random variable which follows a Poisson distribution with parameter λ can be generated in three ways: with CDF inversion (in an iterative fashion) and exploiting the property that links a Poisson distribution with the Poisson Process of intensity λ.
CDF inversion is performed in an iterative way. The CDF of a Poisson process is $F(k) = \sum_{i=0}^{k} e^{-\lambda} \frac{\lambda^i}{i!}$ and it can be written as $F(k+1) = \frac{\lambda}{k+1} F(k) + F(k)$ with $F(0) = e^{-\lambda}$. The following algorithm exploits this property:

Algorithm 4 CDF inversion for Poisson(λ)

procedure
  Let U be a number, generated from a $U[0,1]$ distribution
  Let $X = 0, pr = e^{-\lambda}, F = pr, i = 0$
  while $U >= F$ **do**
    $X = X + 1$
    $pr = \frac{\lambda}{i+1} pr$
    $F = F + pr$
    $i = i + 1$
  return X

The second algorithm is based on the following fact. In a Poisson process with intensity λ the number of events in a time interval t is distributed according to a Poisson distribution with mean λt. The time between each event is an exponential with mean $\frac{1}{\lambda}$. The algorithm counts the number of events X in a time interval $t = 1$ by generating exponential random variables until they sum up to 1. The exact procedure is described in Algorithm 5. Note that an exponential random variable can be generated by the direct inversion of its CDF as like as the the geometric rv, using the formula $E = -\frac{1}{\lambda} \log(U)$ with U a uniform $U[0,1]$ random variable.
In the previous the Poisson random variable X is defined as $X = \arg\max_n \sum_{i=0}^{n} E_i \leq 1$ with $E_i == \frac{-1}{\lambda} \log(U_i)$. Therefore $X = \arg\max_n \frac{-1}{\lambda} \sum_{i=0}^{n} \log(U_i) \leq 1 = \arg\max_n \frac{-1}{\lambda} \log(\Pi_{i=0}^{n} U_i) \leq 1$ and finally $X = \arg\min_n \Pi_{i=0}^{n} U_i > e^{-\lambda}$. Therefore the third algorithm is described by the pseudocode 6.

Algorithm 5 Generation of a Poisson(λ) with exponential interarrival times

procedure
  Let $X = 0$
  Let U be a number, generated from a $U[0, 1]$ distribution
  Let $E = \frac{-1}{\lambda} \log(U)$ the time of next event
  Let $i = E$ the time of last event
  while $i \leq 1$ **do**
    $X = X + 1$
    Let U be a number, generated from a $U[0, 1]$ distribution
    Let $E = \frac{-1}{\lambda} \log(U)$
    $i = i + E$
  return X

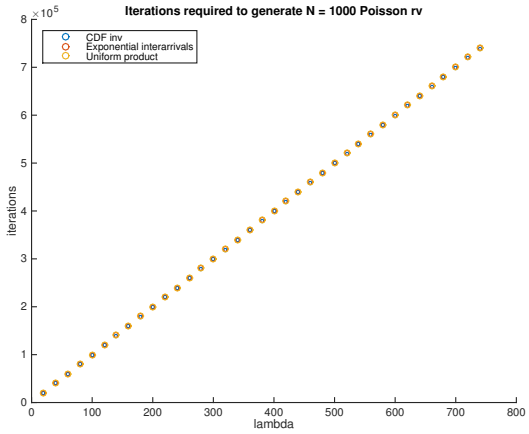
Algorithm 6 Generation of a Poisson(λ) with product of uniforms

procedure
  Let $X = 0$
  Let U be a number, generated from a $U[0, 1]$ distribution
  Let $i = U$
  while $i \leq e^{-\lambda}$ **do**
    $X = X + 1$
    Let U be a number, generated from a $U[0, 1]$ distribution
    Let $i = Ui$
  return X

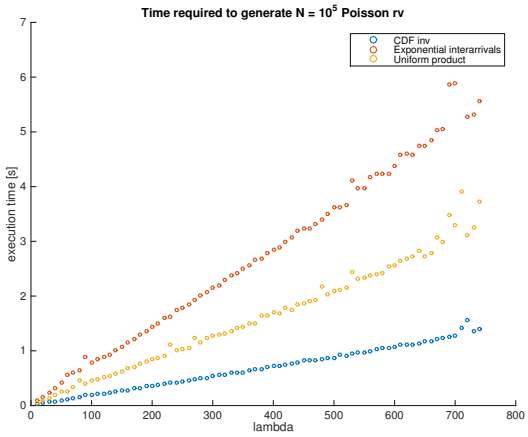
All three methods have a computational complexity which is proportional to λ, as it can be seen in Figure 8a. However the implementation of the second algorithm is much more expensive, since it requires to compute a logarithm and a division at each iteration. This can be seen in Figures 8b and 8c where the time to generate 10^5 Poisson random variables with the three algorithms is plotted against the value of λ. In Figure 8c $\lambda \in [0.001, 10]$ while in Figure 8b $\lambda \in [10, 740]$. For $\lambda > 745.14$ the CDF inversion and algorithm 6 are unfeasible since they require a value (respectively $pr(k = 0)$ and the lower bound on which the product of uniforms is checked) to be initialized to $e^{-\lambda} < \delta$ with $\delta = 4.9407e - 324$ the lowest positive number which can be represented in MATLAB, so it is approximated to 0 and this becomes a standpoint for the algorithm.

4 Exercise 4

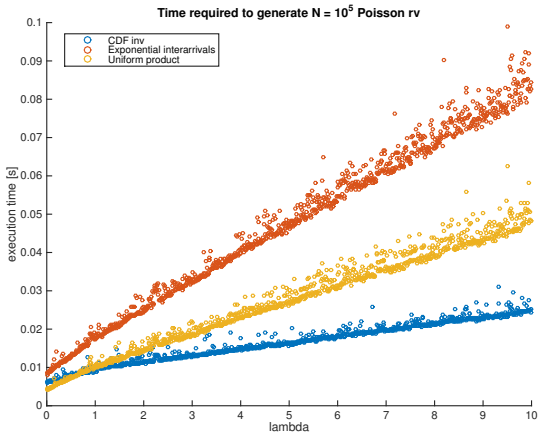
The first linear congruential generator of this exercise (LCG1) has as parameters $a = 18, m = 101, c = 0$ and the second (LCG2) $a = 2$ and the same m, c. They are both full period. Indeed by generating a sequence of $m - 1$ samples there are no repeated samples for both of them. This test is performed in the attached MATLAB code. Note that a period for a multiplicative LCG ($c = 0$) is $m - 1$ as described in Section 1.
Figure 9 contains the lag plot for lag 1 of these generators. From Figure 9c it can be seen that the samples of LCG2 are strongly correlated because of the bad choices of the LCGs parameters. Actually, up to the wrap around, for any choice of x_0, the sample $n + 1$ is $a = 2$ times the sample n. Moreover, since the number of possible values is small, the randomness of the sequence is limited. The other LCG seems to have samples which are uniformly distributed in a 2d space at lag 1, with rows of points which are equally spaced, but by looking at the analysis in three dimensions in Figure 9b it can be clearly seen that they are not well distributed and that they fall into hyperplanes, as it always happens for LCG (Masaglia Theorem from [3]). Moreover Masaglia Theorem states that if n-tuples of subsequent samples of a LCG are considered then there are at most $n!m^{\frac{1}{n}}$ hyperplanes in an n-space (and they have $n-1$ dimensionality and are parallel). The more the number of actual hyperplanes is closer to this limit the better the rng is. For $n = 3$ and $m = 101$ then $n!m^{\frac{1}{n}} \approx 28$, while in Figure 9b there are just 6 hyperplanes, therefore the points generated by a LCG with these parameters are not as well distributed as they could be.



(a) Iterations required to generate $N = 10^5$ Poisson random variables with the three methods

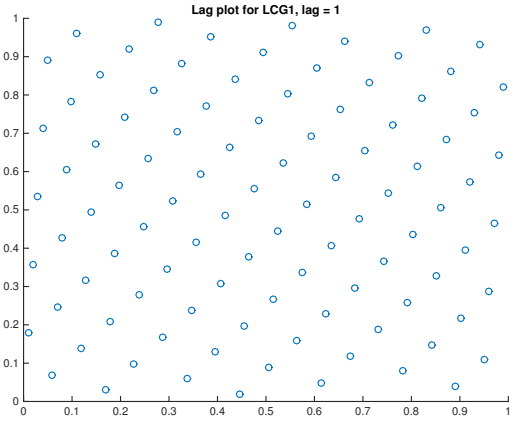


(b) Time required to generate $N = 10^5$ Poisson random variables with the three methods as a function of λ

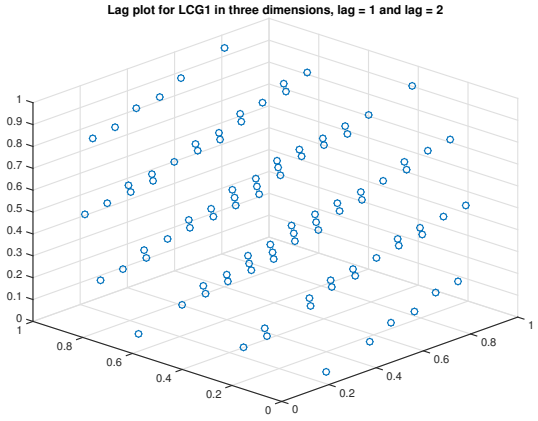


(c) Time required to generate $N = 10^5$ Poisson random variables with the three methods as a function of λ

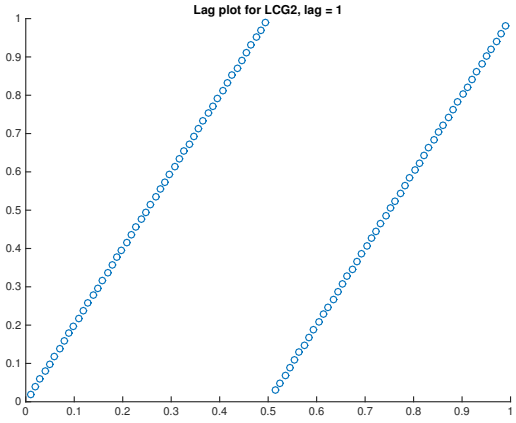
Figure 8: Comparisons between methods for the generation of Possion variates



(a) LCG1



(b) LCG1 in 3 dimensions

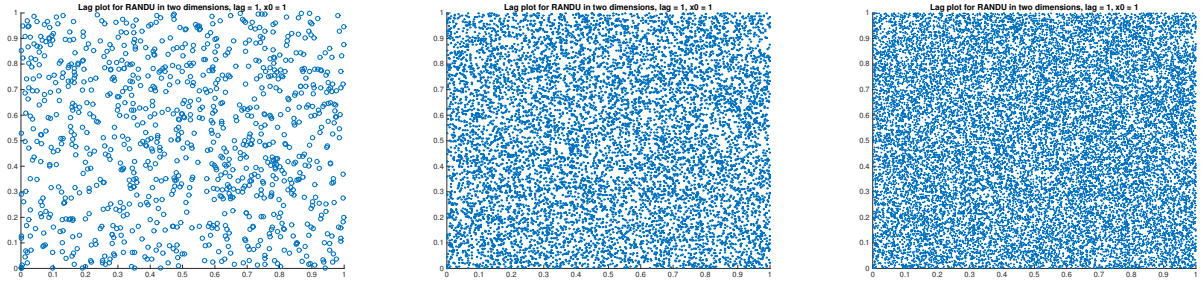


(c) LCG2

Figure 9: Lag plots for LCG1 and LCG2

5 Exercise 5

The third LCG under analysis belongs to the family of LCGs with $m = 2^M$, in particular $m = 2^{31}$ and $a = 65539$, which is a prime number ($c = 0$ as usual). If the seed is an odd number (for example $x_0 = 1$) these are the parameters of the rng called RANDU, a random number generator which was designed by IBM in the 1960s [2]. Apparently, if just 2 dimensions are observed, the samples at lag 1 are equally distributed in the unit square and there are not hyperplanes structures as shown in Figure 10. However if another dimension is taken into account the correlation between subsequent samples is clear, since there are 15 hyperplanes on which the points are distributed, approximately with the same distance one to each other as it can be seen in Figure 11. As previously stated, this is a common result in LCG analysis. Moreover, by recalling Masaglia Theorem, for n=3 and RANDU rng ($m = 2^{31}$) there should be at most $(3!2^{31})^{1/3} \approx 2344$ hyperplanes, but the triplets of this LCG are distributed only on 15 of them and this shows the weakness of RANDU in generating samples which are uncorrelated.



(a) 1000 samples (b) 10000 samples (c) 20000 samples

Figure 10: Lag plots for RANDU, lag = 1, $x0 = 1$

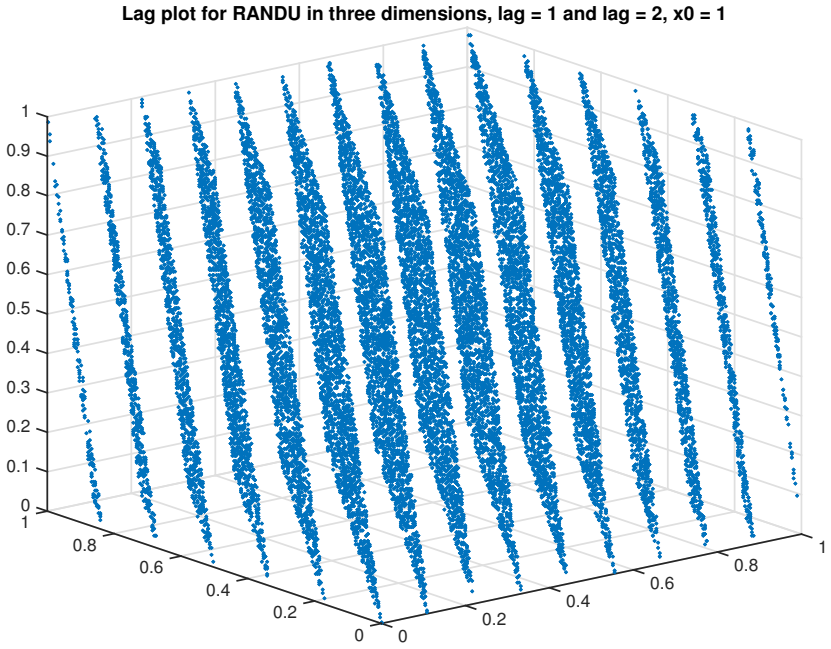


Figure 11: Lag plot for RANDU in three dimension, each point is x_n, x_{n+1}, x_{n+2}, 20000 points

References

[1] Y. Le Boudec, Performance Evaluation of Computer and Communications Systems, EPFL, 2015

[2] D.E. Knuth, The Art of Computer Programming, volume 2: Seminumerical Algorithms, Addison-Wesley, Reading, MA, 2nd edition, 1981.

[3] G. Masaglia, Random numbers fall mainly in the planes, Mathematics Research Laboratory, Boeing Scientific Research Laboratories, 1968