Network Analysis and Simulation - Homework 2

Michele Polese, 1100877

April 15, 2015

1 Exercise 1

Figure 1 shows the randomness of a U[0,1] sequence generated with a *Linear Congruential Generator* (LCG) in different ways: by comparing with a U[0, 1] generated by MATLAB Mersenne Twister rng, by showing the lac of correlation between samples with the autocorrelation function and lag plots.
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 entropy extraction, as in the first plot where $x_0^{\text{LGC}_1} = 1$ and $x_0^{\text{LGC}_2} = 2$, then there's a strong correlation between the two streams (actually 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.
In Figure 3 there are two distributions generated with rejection sampling. This technique allows to compute a random variable with a certain probability density distribution which is not completely known (i.e. missing normalization factor) by comparing uniform random variables with the expected values.

2 Exercise 2

A Binomial random variable ($\text{Bin}(n, p)$) can be generated in three different ways. The first is the CDF inversion, which can be computed in an iterative way. Since the CDF of a $\text{Bin}(n, p)$ is $F(r) = \sum_{k=0}^{r} \frac{n!}{(n-k)!k!}(1-p)^{n-k}p^k$ cannot be inverted in a close form, it is possible to compute it 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, U = U[0,1], 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
A more efficient 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 $G(p)$. Thus
The algorithms are implemented in the attached MATLAB code in order to compare their performances. Note that the average number of iterations for Algorithm **??** 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. The generation of geometric strings of zeros has a complexity which is proportional to np too, since on average the strings have $\frac{1-p}{p}$ zeros and a 1, thus are long $1/p$: therefore the number of iterations needed to reach n are on average $\frac{n}{1/p} = np$ (the comparisons are $1 + np$).
The n Bernoulli trials method is the slowest, especially when n grows. The relation between the three methods can be seen in Figure 4. CDF inversion and geometric method should perform approximately in the same way. Actually when

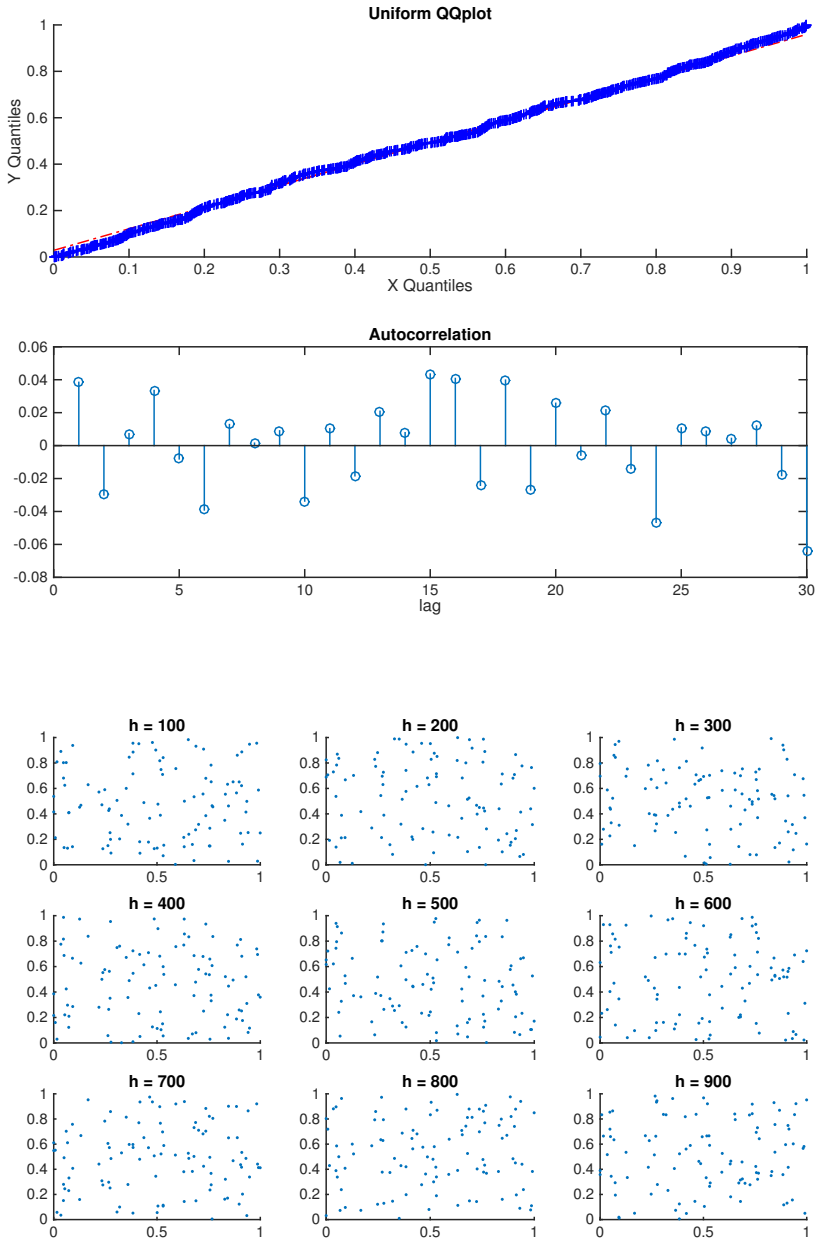


Figure 1: Figure 6.5 in [1]

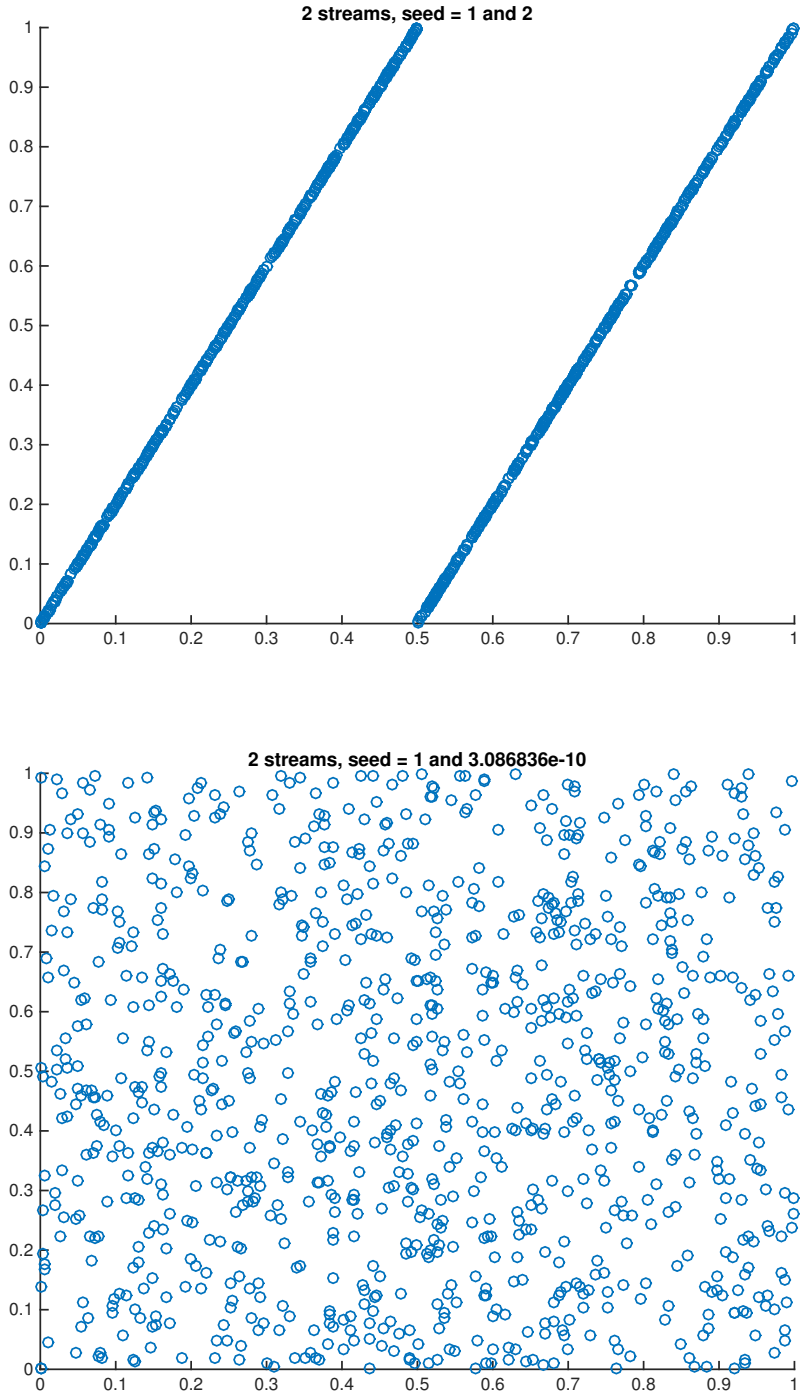


Figure 2: Figure 6.7 in [1]

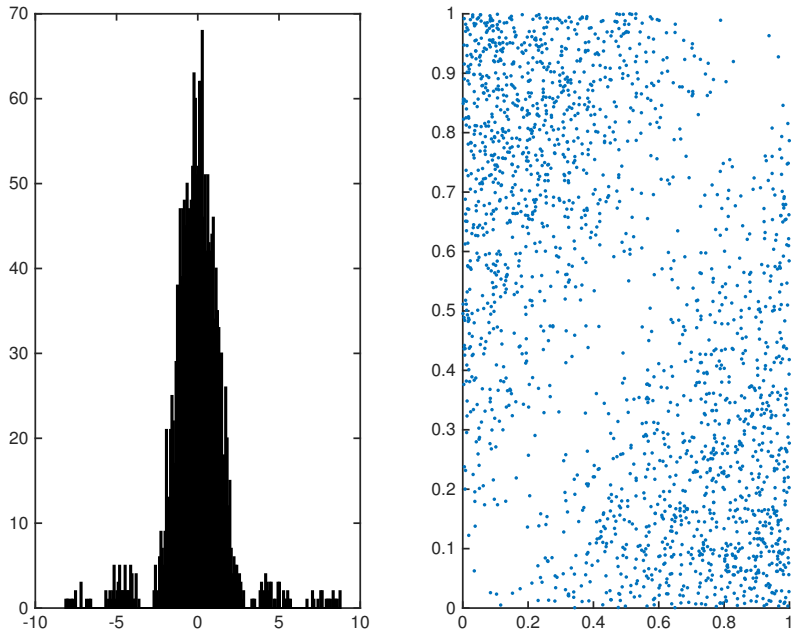


Figure 3: Figure 6.10 in [1]

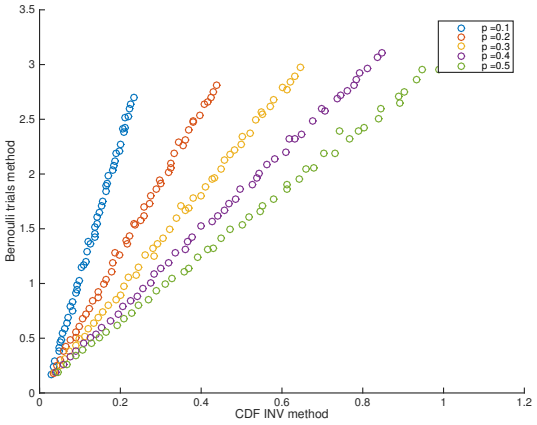
Algorithm 2 Generation of a Bin(n, p) with n bernoulli trials

procedure
    Let $X = 0, i = 1$
    while $i \leq n$ **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

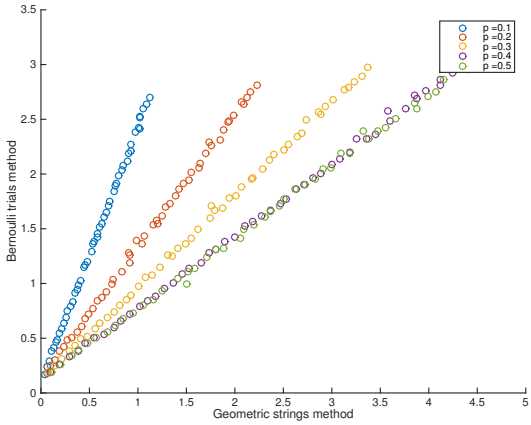
Algorithm 3 Generation of a Bin(n, p) with geometric strings of 0

procedure
    Let $X = 0$
    Let U be a number, generated from a $U[0, 1]$ distribution
    Let $G = \lfloor \frac{\log(U)}{\log(1-p)} \rfloor$ the length of a string of zeros
    Let $i = G + 1$ a string of G zeros and a 1
    while $i \leq n$ **do**
        $X = X + 1$
        Let U be a number, generated from a $U[0, 1]$ distribution
        Let $G = \lfloor \frac{\log(U)}{\log(1-p)} \rfloor$ the length of a string of zeros
        $i = i + G + 1$
    return X

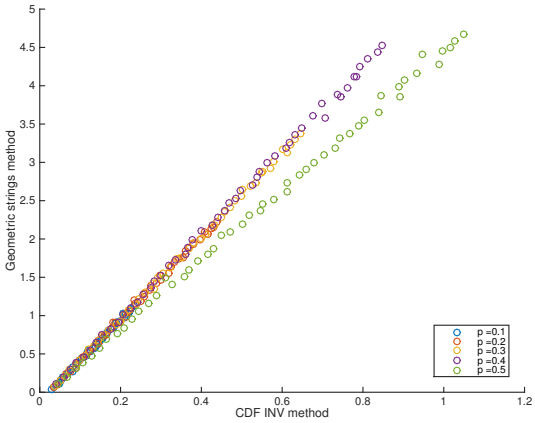
there are many iterations the complex operations that Algorithm 3 has to perform in each iteration (a logarithm, a division, an extraction of random number) make it slower that the simple CDF inversion. Instead when np is small thus the number of iterations is on average lower than 1 then the two methods perform approximately in the same way. This can be seen in Figure 5 where it is plotted the time required to generate 10^5 binomial random variables with $n \in [20, 10^4]$ (increased by a step of 20) and $p \in [10^{-9}, 10^{-8}, 10^{-7}, 10^{-6}, 10^{-5}]$.
This could actually be improved by looking at the values around the mean instead of beginning from 0. Note also that the CDF inversion method is based on iterations, thus the values it computes are subject to 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 $(1-p)^n < e$ the CDF inversion cannot be performed. This can happen for $p \approx 1/2$ and $n > 1000$.



(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

References

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

[2] M. Pinsky, S. Karlin, An Introduction to Stochastic Modeling, 4^{th} edition, Elsevier, 2011

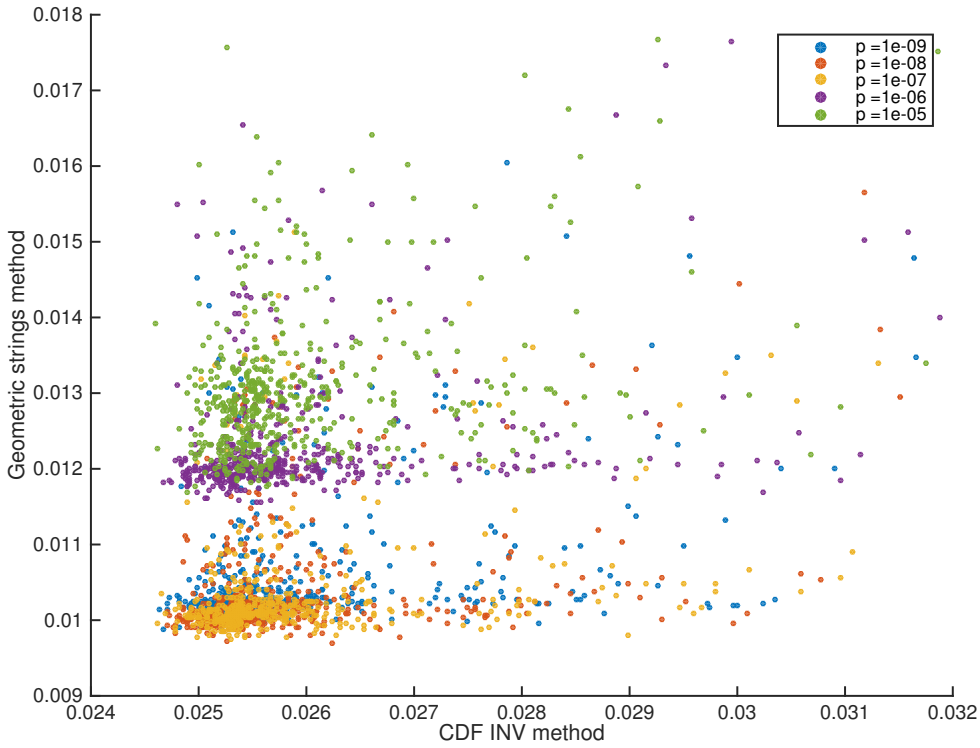


Figure 5: Geometric strings vs CDF inversion methods' execution time for $np < 1$