

Network Analysis and Simulation - Homework 2

Michele Polese, 1100877

April 16, 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: $x_n = (ax_{n-1} + c) \bmod m$. 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 the randomness of a $U[0,1]$ sequence generated with a LCG by normalizing the x_n sequence to m in different ways: by comparing with a $U[0, 1]$ generated by MATLAB Mersenne Twister rng, by showing the lack 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 closed form, it is possible to compute it in an iterative way with the following algorithm:

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

```
1: procedure
2:   Let  $U$  be a number, generated from a  $U[0, 1]$  distribution
3:   Let  $X = 0, pr = (1 - p)^n, F = pr, i = 0$ 
4:   while  $U \geq F$  do
5:      $X = X + 1$ 
6:      $pr = \frac{n-i}{i+1} \frac{p}{1-p} pr$ 
7:      $F = F + pr$ 
8:      $i = i + 1$ 
9:   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 $G(p)$. 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

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

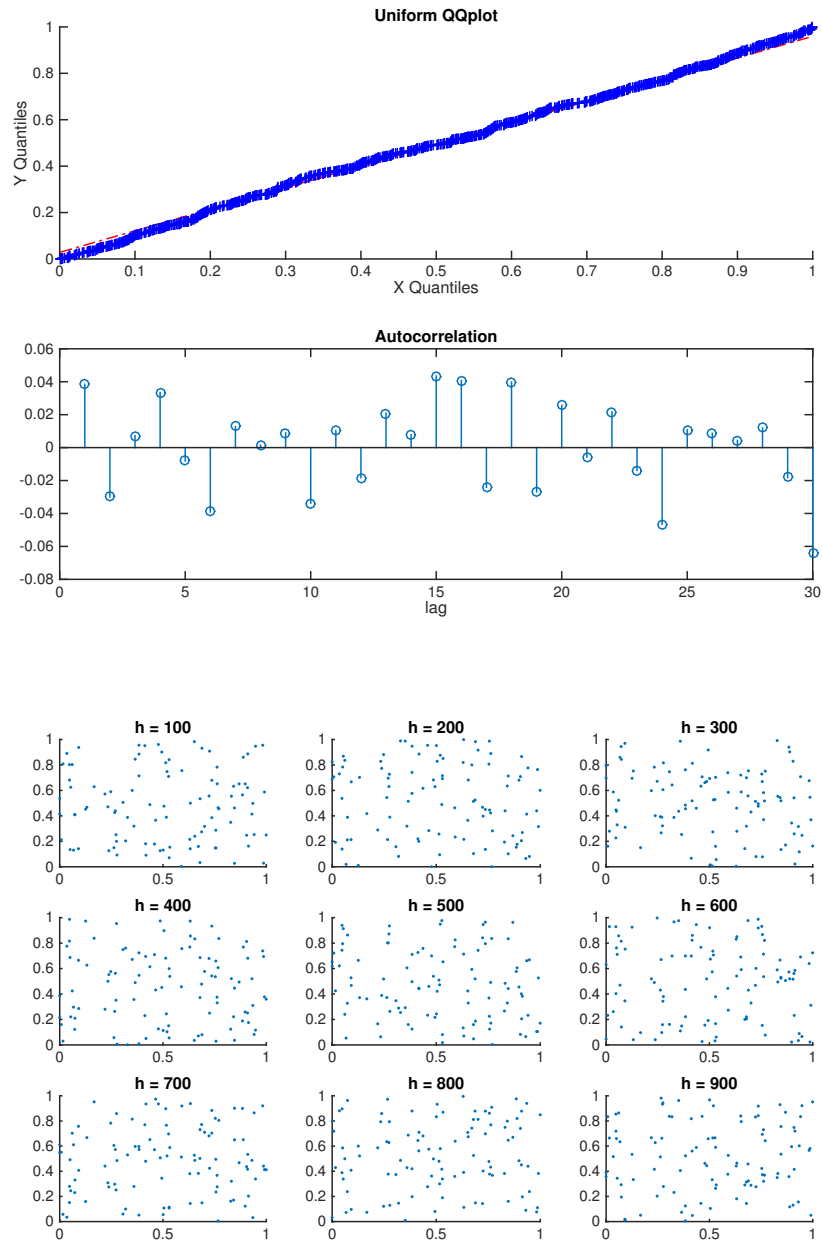


Figure 1: Figure 6.5 in [1]

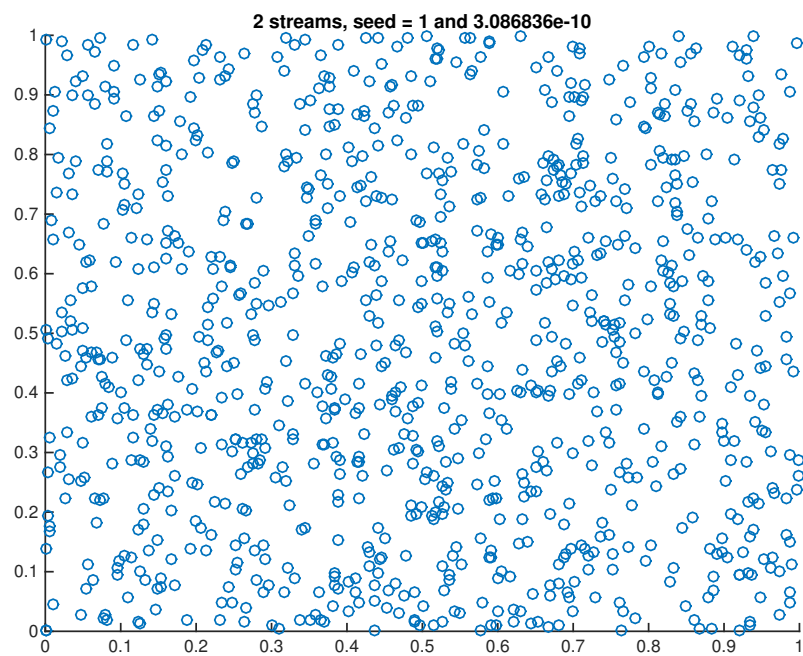
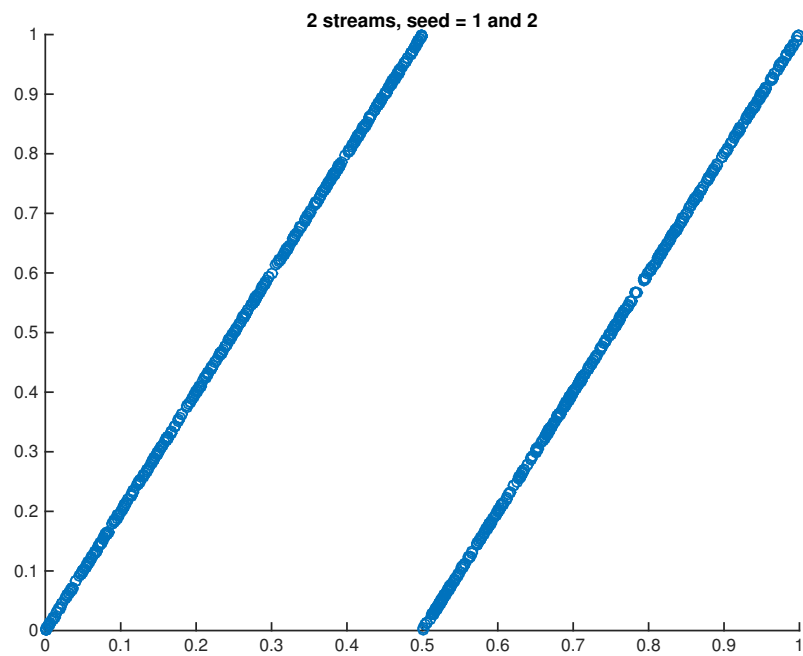


Figure 2: Figure 6.7 in [1]

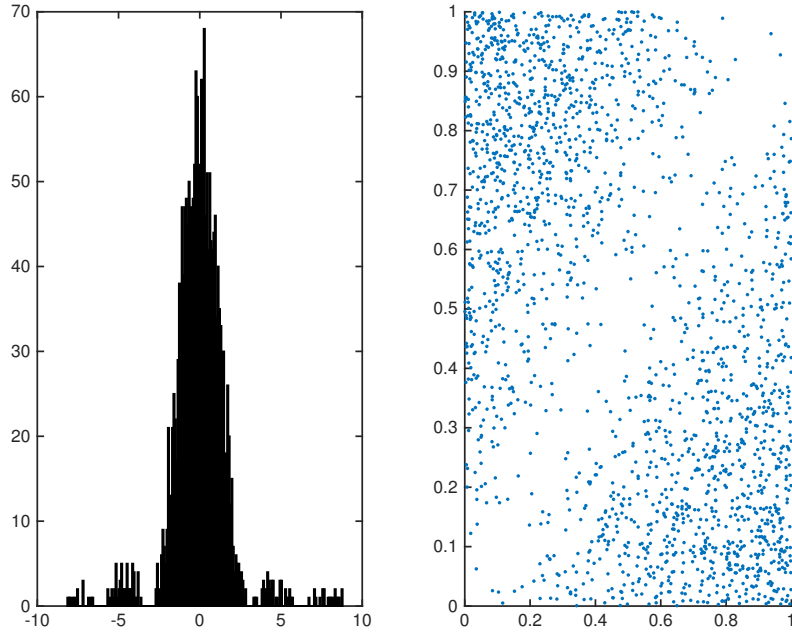


Figure 3: Figure 6.10 in [1]

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

```

1: procedure
2:   Let  $X = 0, i = 1$ 
3:   while  $i \leq n$  do
4:     Let  $U$  be a number, generated from a  $U[0, 1]$  distribution
5:     if  $U \leq p$  then
6:        $X = X + 1$ 
7:      $i = i + 1$ 
8:   return  $X$ 

```

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

```

1: procedure
2:   Let  $X = 0$ 
3:   Let  $U$  be a number, generated from a  $U[0, 1]$  distribution
4:   Let  $G = \lfloor \frac{\log(U)}{\log(1-p)} \rfloor$  the length of a string of zeros
5:   Let  $i = G + 1$  a string of  $G$  zeros and a 1
6:   while  $i \leq n$  do
7:      $X = X + 1$ 
8:     Let  $U$  be a number, generated from a  $U[0, 1]$  distribution
9:     Let  $G = \lfloor \frac{\log(U)}{\log(1-p)} \rfloor$  the length of a string of zeros
10:     $i = i + G + 1$ 
11:  return  $X$ 

```

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 relation between the three methods can be seen in Figures 4, 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 than the simple CDF inversion. Figure 4c has on x and y axis the time needed to generate 10^5 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 dependence, 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 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 $p \in [10^{-9}, 10^{-8}, 10^{-7}, 10^{-6}, 10^{-5}]$ and in Figures . In this case the geometric strings method has a weak dependence on p , probably due to the generation of the geometric random variable. 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. This can happen for $p \approx 1/2$ and $n > 1000$.

Another observation is that despite the dependence 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 are less, their complexity is higher, therefore for values of p which are close to 0.5 then the Bernoulli becomes faster.

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 link 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)

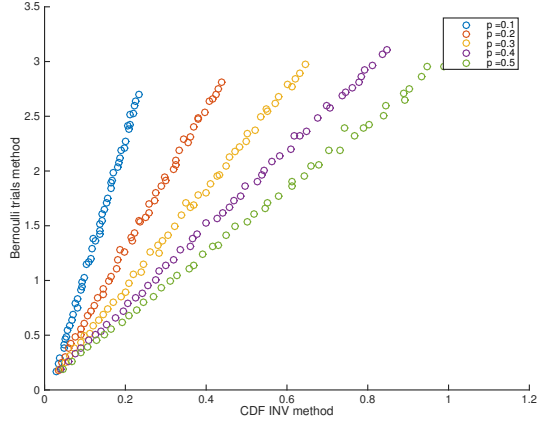
```

1: procedure
2:   Let  $U$  be a number, generated from a  $U[0, 1]$  distribution
3:   Let  $X = 0, pr = e^{-\lambda}, F = pr, i = 0$ 
4:   while  $U >= F$  do
5:      $X = X + 1$ 
6:      $pr = \frac{\lambda}{i+1} pr$ 
7:      $F = F + pr$ 
8:      $i = i + 1$ 
9:   return  $X$ 
```

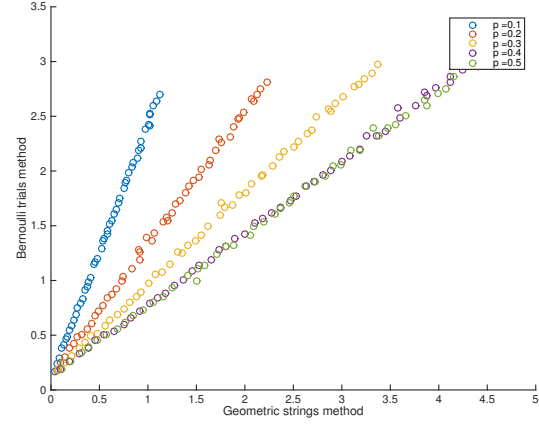
The second algorithm is based on the following fact. In a Poisson process with intensity λ the number of events in a time interval t is distributed according to a Poisson distribution with mean λt . The time between each event is an exponential with mean $\frac{1}{\lambda}$. The algorithm counts the 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 $X = \arg \max_n \frac{-1}{\lambda} \sum_{i=0}^n \log(U_i) \leq 1 = \arg \max_n \frac{-1}{\lambda} \log(\prod_{i=0}^n U_i) \leq 1$ and finally $X = \arg \min_n \prod_{i=0}^n U_i > e^{-\lambda}$. Therefore the third algorithm is described by the following pseudocode.

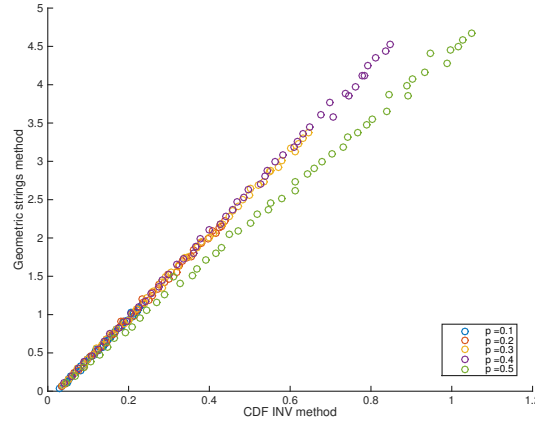
All three methods have a computational complexity which is proportional to λ , as it can be seen in Figure 6. However the implementation of the third algorithm is much more expensive. This can be seen in



(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

4 Exercise 4

Let the first linear congruential generator (LCG1) have $a = 18, m = 101, c = 0$ 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.

Figures 7 and 8 contain the lag plot for lag 1 of these generators. It can be seen that the samples 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 times the sample n . Moreover, since the number of possible values is small, the randomness of the sequence is limited.

References

- [1] Y. Le Boudec, Performance Evaluation of Computer and Communications Systems, EPFL, 2015
- [2] M. Pinsky, S. Karlin, An Introduction to Stochastic Modeling, 4th edition, Elsevier, 2011

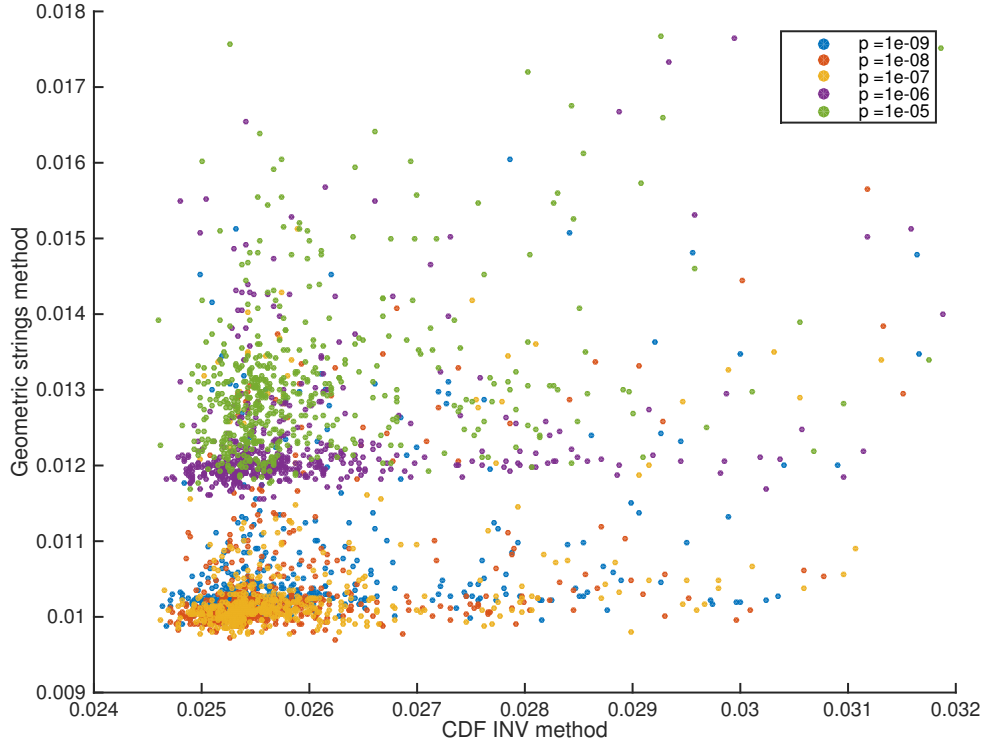


Figure 5: Geometric strings vs CDF inversion methods' execution time for $np < 1$

Algorithm 5 Generation of a $\text{Poisson}(\lambda)$ with exponential interarrival times

```

1: procedure
2:   Let  $X = 0$ 
3:   Let  $U$  be a number, generated from a  $U[0, 1]$  distribution
4:   Let  $E = \frac{-1}{\lambda} \log(U)$  the time of next event
5:   Let  $i = E$  the time of last event
6:   while  $i \leq 1$  do
7:      $X = X + 1$ 
8:     Let  $U$  be a number, generated from a  $U[0, 1]$  distribution
9:     Let  $E = \frac{-1}{\lambda} \log(U)$ 
10:     $i = i + E$ 
11:  return  $X$ 

```

Algorithm 6 Generation of a $\text{Poisson}(\lambda)$ with product of uniforms

```

1: procedure
2:   Let  $X = 0$ 
3:   Let  $U$  be a number, generated from a  $U[0, 1]$  distribution
4:   Let  $i = U$ 
5:   while  $i \leq e^{-\lambda}$  do
6:      $X = X + 1$ 
7:     Let  $U$  be a number, generated from a  $U[0, 1]$  distribution
8:     Let  $i = Ui$ 
9:  return  $X$ 

```

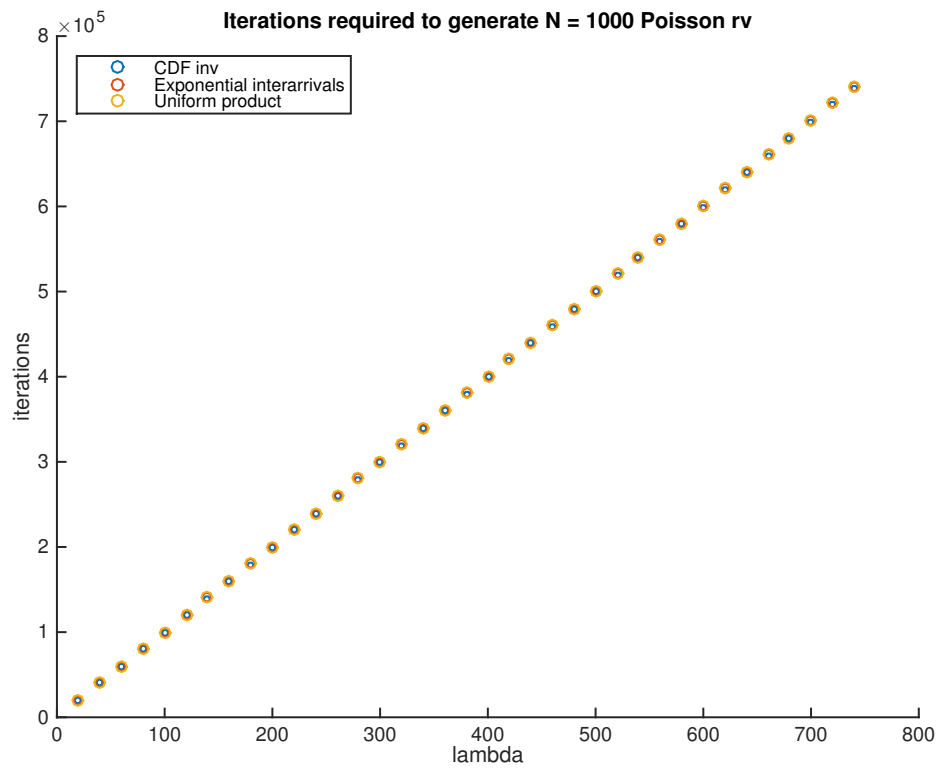


Figure 6: Iterations required to generate $N = 1000$ Poisson random variables with the three methods

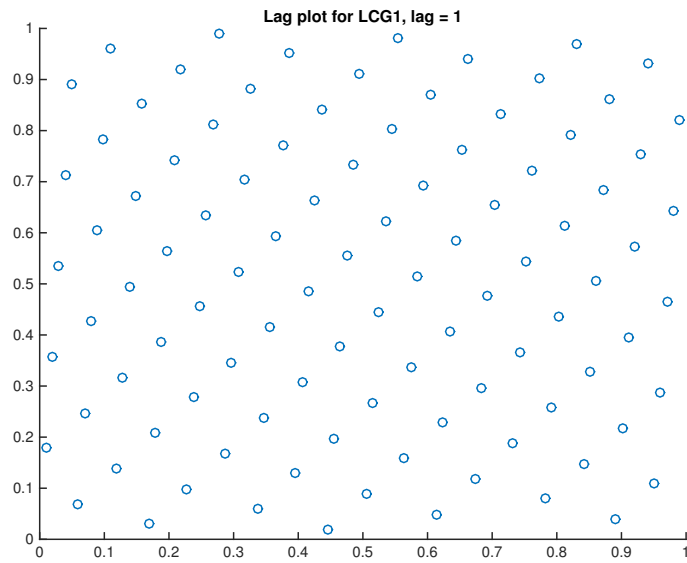


Figure 7: Lag plot for LCG1 at lag 1

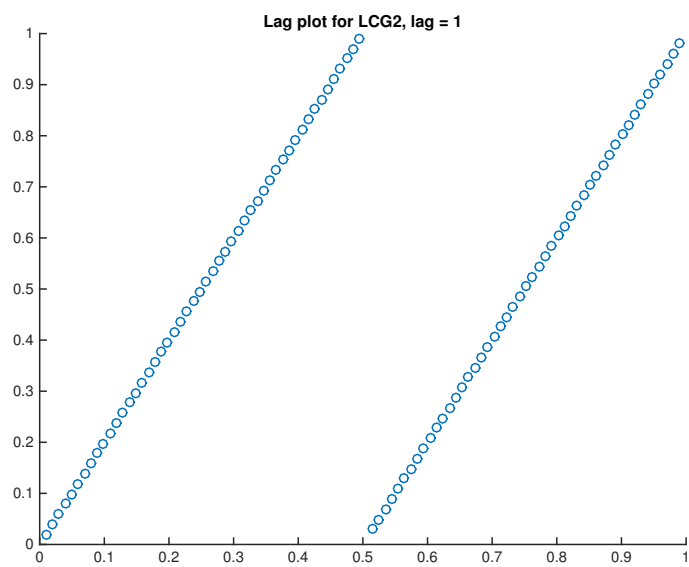


Figure 8: Lag plot for LCG2 at lag 1