Pseudo-Random Numbers Generators

Anne GILLE-GENEST

February 18, 2016

Premia 18

Contents

1 Introduction **2**
1.1 Definitions 2
1.2 Good generators 3

2 Linear Methods **4**
2.1 Linear Congruential Generators (LCG) 5
2.2 Multiple Recursive Generator (MRG) 5
2.3 Combined LCGs and MRGs (CMRG) 6
2.4 Matrix LCGs and MRGs 7
2.5 Multiply-with-Carry Generator (MWC) 8

3 Shift-Register or Digital Methods **8**
3.1 Linear Feedback Shift Register (LFSR) or Tausworthe Generator 8
3.2 Generalized Feedback Shift Register (GFSR) 10
3.3 Equidistribution Properties 10

4 Nonlinear Methods **11**
4.1 Inversive Congruential Generators (ICG) 11
4.2 Inverse Generators 11
4.3 Quadratic Congruential Generators 13
4.4 Blum, Blum & Shub generator (BBS) 13

5 Use of the Random Number Generators **13**

Implementation of the Pseudo-Random Numbers Generators

Monte Carlo simulation method consists in approximating $E[\psi(X)]$ by $\frac{1}{N}\sum_{n=1}^{N}\psi(x_n)$, where x_n are $i.i.d$ to X. Thus, as explained in the section about simulation of random variables, we first need sequences of uniform independent numbers. These random numbers u_n can next be transformated into variables x_n with specific distributions.
Randomness is simulated by a pseudo-random number generator whose output is assumed to be a sequence of independent and identically distributed $U(0,1)$ random variables. An other approach for simulation concerns Quasi-Monte Carlo simulatio

This part deals with the presentation of various pseudo-random generators. We first introduce the notion of generator and the properties we require for a good generator. The problem is not obvious and a bad generator can lead to false results for a considered simulation. In the following sections, we briefly describe different methods to construct random number generators: linear methods, shift register methods and nonlinear methods. Literature about Random Number Generator is abundant. We refer the reader to general articles like [5], [4], [3], [7], [8] or [1]. Specific references will be given in the presentation. For a more complete bibliography, we suggest consulting the pages *http://www.iro.umontreal.ca/ lecuyer* or *http://www.random.mat.sbg.ac.at*.
Implementation of the generators is described in the implemented part.

1 Introduction

We first give a definition for a pseudo-random number generator and some properties about it.

1.1 Definitions

A *pseudo-random number generator* is a structure $\mathcal{G} = (S, s_0, T, U, G)$ where S is a finite set of states, $s_0 \in S$ is the *initial state*, the mapping $T : S \to S$ is the *transition function*, U is a finite set of *outputs symbols*, and $G : S \to U$ is the *output function*.
This definition was introduced by L'Ecuyer in [5] or [4] for instance.

Since S is finite, the sequence of states is periodic. The *period* is the smallest positive integer ρ such that for some integer $\tau \geq 0$ and for all $n \geq \tau$, $s_{\rho+n} = s_n$. The smallest τ with this property is called the *transcient*. When $\tau = 0$, the sequence is said to be *purely periodic*.

The *resolution* of a generator is the largest number x such that all output values are multiples of x. It determines the maximal number of different values we can obtain with the generator.

1.2 Good generators

We summarize in this point the properties required for a good random numbers generator.

- *Large period length.*
  The applications of simulations require more and more random numbers and computers now allow more iterations steps than before. Thus the random number generator must have a very large period.

  Note that the period can be larger than the resolution x. In fact the generator produces only x different values but not always in the same order. For problems in dimension $D \geq 2$, we are interested in the period rather than the resolution.

- *Good equidistribution properties and statistical independence of successive pseudorandom numbers.*
  Generated sequences should behave as sequences of independent random variables, uniformly distributed over $[0, 1]$. The generator should pass statistical tests for uniformity and independence. Several empirical statistical tests are detailed in Knuth [1] or in L'Ecuyer [4]. We just enumerate the main ones: general tests like chi-square or Kolmogorov-Smirnov tests; specific tests like equidistribution test, serial test, gap test, partition test, permutation test, run test, collision test, test on subsequence or correlation test. All these tests are empirical tests, i.e *a posteriori* tests based on a realization of the generated sequence.
  To increase its power, a given test can be replicated N times on disjoint part of the sequence. It allows to test the local behaviour of generators. This procedure is called multilevel-test.
  But since generated sequences are deterministic, we can always find a test the sequence will fail. L'Ecuyer introduced the notion of *PT-perfect* (polynomial-time perfect) generator but no generator has been proven PT-perfect until now.

  Theoretical tests are *a priori* tests. They tell us in advance the properties of the full period of the sequence.

- *Little intrinsic structure.*
  Successive values produced by some of the described generators have a lattice structure in any given dimension.
  Let $T_t = \{U_n = (u_n, \ldots, u_{n+t-1})/n \geq 0\}$. The lattice structure means that all the points of T_t lie in a family of equidistant parallel hyperplans. The smaller the distance between hyperplanes is, the better the generator is, because this means thinner empty slices of space.
  Lattice tests help to know the number of hyperplanes and the distance between them.

- *Efficiency, fast generation algorithm.*
  It is important, especially if we use many generators together or in parallel, that the algorithm allows a fast generation of random numbers and that it does not require too much memory space.

- *Repeatability.*
  Being able to reproduce exactly the same sequence can be necessary and very usefull for practical applications. Thus we do not use the current time (computer clock) to initialize the generators.

- *Portability*
  It means that the generator will produce exactly the same sequence on different computers or with different compilers. This parameter is important to obtain the same results if we realize simulations on various machines

- *Unpredictability*
  It means that we can not predict the next generated value by the algorithm from the previous ones. This property is parlicularly important for cryptographic applications.

2 Linear Methods

A first group of pseudo-random numbers generators corresponds to linear methods. The principle is based on a linear recurrence relation to compute the next value from the previous ones. First of all we describe the simplest of them: the Linear Congruential Generator based on a recurrence of order 1. The others linear generators are based on a variant of it: recurrence of order k and combination between generators.

2.1 Linear Congruential Generators (LCG)

The state at step n is an integer x_n and the transition function T is defined by the recurrence :

$$x_n = (ax_{n-1} + c) \mod m$$

where $m > 0, a > 0$ and c are integers called the *modulus*, the *multiplier* and the *additive constant* respectively. The set S is $\{0, \ldots, m-1\}$.
The output function G is defined by $G(x) = x/m$. The output is:

$$u_n = G(x_n) = \frac{x_n}{m} \in [0, 1]$$

The maximal period for a LCG is m. It is reached if and only if c is relatively prime to m, $a - 1$ is a multiple of p for every prime factor p of m and if m is a multiple of 4, then $a - 1$ is also a multiple of 4.

• When $c = 0$, this generator is called a *multiplicative linear congruential generator* (MLCG). The recurrence relation is given by :

$$x_n = (ax_{n-1}) \mod m$$

The maximal period for a MLCG is $m - 1$, since 0 is an absorbing state.
You can jump ahead from x_n to $x_{n+\nu}$ with the relation :

$$x_{n+\nu} = a^\nu x_n \mod m = (a^\nu \mod m) x_n \mod m$$

• Properties of this method:
- The modulus m and the period are bounded in terms of the word length of the machine,
- Such generators produce a lot of regularity in sequences,
- And an unfavorable lattice structure.

2.2 Multiple Recursive Generator (MRG)

The recurrence relation for a MRG is defined by :

$$x_n = (a_1 x_{n-1} + \cdots + a_k x_{n-k}) \mod m$$

where the *order* k and the *modulus* m are positive integers, while the coefficients $a_1, \ldots, a_k$ are in $\{-(m-1), \ldots, m-1\}$.
Let $\mathbb{Z}_m = \{0, 1, \ldots, m-1\}$. The state at step n of the MRG is the vector $s_n = (x_n, \ldots, x_{n-k+1}) \in \mathbb{Z}_m^k$.

The output function is defined by $u_n = G(s_n) = x_n/m$, which gives a value in $[0, 1]$.
We define the characteristic polynomial P of the MRG by :

$$P(z) = z^k - a_1 z^{k-1} - \cdots - a_k$$

The maximal period of a MRG is $\rho = m^k - 1$. It is reached if and only if m is prime and P is a primitive polynomial over $\mathbb{Z}_m^k$.

The resolution of a MRG is $1/m$. It means that only m different values are produced by this generator but not always in the same order (the period is greater than m).

For large values of k, the disadvantages incurred by the lattice structure become less pronounced.

2.3 Combined LCGs and MRGs (CMRG)

To increase the period length and improve the statistical behavior of a generator, an idea is to combine different generators. Well known combination methods are :

1. *Shuffling* one sequence with another one or with itself.
   The principle is to shuffle the outputs to remove low-order serial correlations. The shuffling algorithm is due to Bayes and Durham.
   The first step is to fill up a table $t[\ ]$ of size d with the first output values from the generator.
   Then each time we need a new random number, we generate an index j and use the value $t[j]$. Finally we replace it by the next value from the generator.
   The choice of the index j can be made with an other generator or with the same one.
   Nevertheless, effects of the shuffle procedure are not wellknown from the theoretical point of view.

2. Adding several integer sequences modulo some integer m_0, or adding sequences in $[0, 1]$ modulo 1, or adding binary fraction bitwise modulo 2.
   We consider J MRGs defined by :

$$x_{j,n} = (a_{j,1}x_{j,n-1} + \cdots + a_{j,k}x_{j,n-k}) \mod m_j$$

for $j = 1, \ldots, J$, assumed to be purely periodic with period ρ_j and the moduli m_j to be pairwise relatively prime.
We define two combinations

$$z_n = (\sum_{j=1}^{J} \delta_j x_{j,n}) \mod m_1 \quad ; \quad u_n = \frac{z_n}{m_1}$$

and

$$\omega_n = (\sum_{j=1}^{J} \delta_j \frac{x_{j,n}}{m_j}) \mod 1$$

where $\delta_1, \ldots, \delta_J$ are arbitrary integers such that δ_j and m_j have no common factor.

Results :
The sequences $\{u_n\}$ and $\{\omega_n\}$ have period length $\rho = lcm(\rho_1, \ldots, \rho_J)$.
We can find coefficients a_i independent of δ_j such that the sequence ω_n satisfies the relation :

$$x_n = (a_1 x_{n-1} + \cdots + a_k x_{n-k}) \mod m \ ; \ \omega_n = \frac{x_n}{m}$$

Resolution of such CMRG is $1/m$.

In [6], L'Ecuyer suggests some values for the parameters which lead to satisfying properties for the generators.

2.4 Matrix LCGs and MRGs

To simulate a sequence of independent random vector variables with the uniform distribution on I^l, LCGs and MRGs can be generalized. Then we consider recurrence relations for vectors with matrix coefficients.

- *Matrix method*:

$$X_{n+1} = X_n A \mod M$$

where A is a $l \times l$ matrice and $U_n = \frac{X_n}{M}$ for each coordinate of X_n. The generator is full-period if the matrice A is non singular modulo M, that is $\gcd(\det(A), M) = 1$.

- *Multiple Recursive Matrix Method*:

$$X_n = (A_1 X_{n-1} + \cdots + A_k X_{n-k}) \mod M$$

where $(A_1, \ldots, A_k)$ are $l \times l$ matrix and X_n is a l-dimensional vector.

2.5 Multiply-with-Carry Generator (MWC)

We consider a generator based on the following recurrence relations :

$$x_n = (a_1 x_{n-1} + \cdots + a_k x_{n-k} + c_{n-1}) \mod b$$

$$c_n = (a_1 x_{n-1} + \cdots + a_k x_{n-k} + c_{n-1}) \; div \; b$$

$$u_n = \frac{x_n}{b}$$

The state at step n is $s_n = \{x_n, \ldots, x_{n+k-1}, c_n\}$. $x_n \in \mathbb{Z}_b, c_n \in \mathbb{Z}$.

3 Shift-Register or Digital Methods

The principle is to take several successive values of a MRG x_n defined by $x_{n+k} = \sum_{j=0}^{k-1} a_j x_{n+j} \mod m$, to construct each output value u_n. We have:

$$u_n = \sum_{j=1}^{L} x_{ns+j-1} m^{-j}$$

where s and $L \leq k$ are two positive integers.
s is called the step size and L the number of digits in the m-digit expansion.
If $L = s$, the sequence is called *digital multistep sequence*.

If the polynomial P_k connected with x_n is primitive, then the period length for u_n is $m^k - 1$.
The state at step n is the vector $s_n = (x_{ns}, \ldots, x_{ns+k-1})$. The output values are multiple of m^{-L} (it is the resolution) instead of m^{-1} for linear generators.
A particular case for digital sequences is the Tausworthe Generator described in the next point.

3.1 Linear Feedback Shift Register (LFSR) or Tausworthe Generator

The resolution m^{-L} allows to choose a smaller value for m and to keep a large period. LFSR is a special case for Digital Sequence where $m = 2$. It was introduced by Tausworthe.
In this case $\{x_n\}$ is a sequence of bits and the u_n are constructed by juxtaposing L bits from this sequence.
We consider P_k equal to $z^k - z^{k-r} - 1$. Then, the recurrence relation is given by:

$$x_n = (x_{n-r} + x_{n-k}) \mod 2$$

and the output u_n are obtained by splitting up the sequence x_n into consecutive blocks of length L and then by interpreting each block as the digit expansion in base m of a number.

$$u_n = \sum_{j=1}^{L} x_{ns+j-1} 2^{-j}$$

with $s \leq r$ and $2r > k$.
The period length is:

$$\rho = \frac{m^k - 1}{\gcd(L, m^k - 1)}$$

• **Combination**: Better structural properties are obtained by combination of several recurrences of this type.
Let J LFSR generators, with recurrence relations $\{x_{j,n}\}$ and primitive characteristic polynomial $P_j(z)$ of degree k_j, and output sequence $\{u_{j,n}\}$. Assumed that the step size s_j and the period length $\rho_j = 2^{k_j} - 1$ have no common factor.
Then we define u_n as a bitwise exclusive-or (XOR) of $u_{1,n}, \ldots, u_{j,n}$ that is

$$u_n = u_{1,n} \oplus \cdots \oplus u_{J,n}$$

Results : If the polynomials $P_1(z), \ldots, P_J(z)$ are pairwise relatively prime, the period length ρ of the combined sequence $\{u_n\}$ is $\rho = lcm(\rho_1, \ldots, \rho_J)$. When the ρ_j are relatively prime, then $\rho = \prod_{j=1}^{J} \rho_j$.

It is equivalent to consider the combination of the J LFSR generators or a LFSR generator based on a recurrence relation with characteristic polynomial $P(z) = \prod_{j=1}^{J} P_j(z)$.

• Shift Register generators satisfy the following properties:
- Fast algorithm,
- The period is not bounded in terms of the word length of the machine,
- They produce a not so bad lattice structure. Nevertheless there exists a lattice structure in spaces of formal series,
- Strong uniformity property.

3.2 Generalized Feedback Shift Register (GFSR)

The GFSR can be viewed as a different way to implement a LFSR generator. It is based on parallel MRG $x_{n,j}$ or on a matrix MRG.

$$u_n = \sum_{j=1}^{L} x_{n,j} m^{-j}$$

Its implementation requires more memory than for the LFSR but may lead to a faster generator.

3.3 Equidistribution Properties

Shift Register Generators verify strong uniformity properties, which can be made explicit in terms of the theory of nets and (t, s) sequences, (see Quasi Monte Carlo methods).
Linear Shift Register Generators have a lattice structure in a space of formal series. This structure can be analyzed via the notion of equidistribution.
To understand the meaning of the lattice structure, we introduce the following definitions.

Definitions:
The cube I^t is partitioned into 2^{tl} cubic cells of equal size and we consider the set $T_t = \{U_n = (u_n, \ldots, u_{n+t-1})/n \geq 0, s_0 \in \{0,1\}^k\}$

- T_t (or U_n) is (t, l)-equidistributed if all the cells of I^t contain exactly the same number of points. We note l_t the largest value of $l \leq L$ for which the sequence is (t, l)-equidistributed. It is called the resolution in dimension t.

- *Maximally equidistributed (ME) generator*
  A generator is called ME if the resolution l_t reaches its largest possible value in all dimensions.

- *collision-free (ME-CF) generator*
  A ME generator with the additional property that for $L \geq l > l_t$ and all t, none of the 2^{tl} cells contains more than a single point is called a ME-CF generator.

ME-CF generators have their sets of points T_t very evenly distributed in I^t.

In [2], L'Ecuyer makes explicit some numerical values in order to obtain ME and ME-CF generators.

4 Nonlinear Methods

Nonlinear methods are the last type of pseudo-random number generators we will present here.
The advantage of nonlinear methods is that they do not produce a lattice structure and their outputs behave much like 'truly' random numbers, even over the entire period.

We are interested in two approaches :

1. Keep the transition function T linear but use a nonlinear function G to produce the output.

2. Use a nonlinear transition function T.

We give an example of the first case, called the ICG and a few examples of the second one.

4.1 Inversive Congruential Generators (ICG)

We consider a full-period MRG sequence $\{x_n\}$ with prime modulus m and we replace the output function $u_n = \frac{x_n}{m}$ by :

$$z_n = (\widetilde{x}_{n+1}\widetilde{x}_n^{-1}) \mod m$$

$$u_n = \frac{z_n}{m}$$

where $\widetilde{x}_i$ is the ith non-zero value in x_n and $\widetilde{x}_n^{-1}$ is the inverse of $\widetilde{x}_n \mod m$.

4.2 Inverse Generators

Those methods are well-described in Niederreiter [8].
• *Recursive inverse generator*
This generator is based on the recursion:

$$x_{n+1} = a\overline{x}_n + b \mod m$$

where $\overline{x}$ is such that $x\overline{x} = 1 \mod m$ if $x \neq 0$ and $\overline{x} = 0$ if $x = 0$.
The output value is given by:

$$u_n = \frac{x_n}{m}$$

If m is prime and if $x^2 - bx - a$ is a primitive polynomial over F_m them the

period length is $\rho = m$.

• *Explicit Inverse Congruential Generator*
This method is due to Eichermann and Herrmann. It is based on the recursion:

$$x_n = \overline{an + c} \mod m = (an + c)^{m-2} \mod m$$

The output value is given by:

$$u_n = \frac{z_n}{m}$$

If m is prime, the period length is $\rho = m$.
This algorithm satisfies the s-dimensional serial test for all $s \leq m - 2$. It has an optimal behaviour under the lattice test. It verifies a strong nonlinearity property like the Recursive Inverse Generator.

• These generators satisfy the following properties:
- No nontrivial lattice structure is observed,
- Strong nonlinearity property,
- The s-dimensional serial test is satisfied, for $s \leq d$ where d is a specific value.
- A large choice of parameters is allowed.
The main disadvantage of those both inverse generators is that a calculation of one random number takes $O(\log m)$ multiplication in F_m so the algorithm is not fast.

• *Digital Inverse Generator*
In order to find a faster inversive generator than the recursive inverse or the explicit inverse ones, a new method called Digital Inverse Method has been recently studied.
For a more detailed description, you can refer to [8].
For a precision k, let F_q be the finite field of order $q = 2^k$ and F_q^* the multiplicative group of non zero elements of F_q.
Then we consider the recursion:

$$\gamma_{n+1} = \alpha\overline{\gamma}_n + \beta$$

with $\alpha \in F_q^*,\ \beta \in F_q,\ \gamma_0 \in F_q$
and we define the digital inversive pseudorandom number by:

$$u_n = \sum_{j=1}^{k} c_n^{(j)} 2^{-j}$$

with $c_n = (c_n^{(1)}, \ldots, c_n^{(k)}) \in Z_2^k$ the coordinate vector of γ_n relative to the ordered basis B.

The period length is $\rho = m$ if and only if $x^2 - \beta x - \alpha$ is an IMP polynomial over F_q. In particular $\rho = m$ if the polynomial is primitive.

4.3 Quadratic Congruential Generators

The transformation T is quadratic instead of linear. We consider the recurrence:

$$x_n = (ax_{n-1}^2 + bx_{n-1} + c) \mod m$$
$$u_n = \frac{x_n}{m}$$

where a, b, c and $x_n \in \mathbb{Z}_m$.
If m is a power of 2, the generator has full period $\rho = m$ if and only if a is even, $(b - a) \mod 4 = 1$ and c is odd.

4.4 Blum, Blum & Shub generator (BBS)

The transformation T is quadratic. We consider the recurrence relation:

$$x_n = x_{n-1}^2 \mod m \quad , x_0 = x^2 \mod m$$

with $x \in \mathbb{N}$ and $gcd(x, m) = 1$. Let $m = pq$ be a Blum integer, that is such that p, q are two distinct primes both congruent to 3 modulo 4.
This generator is conjectured to be Polynomial-time perfect.

5 Use of the Random Number Generators

In this section, we summarize some remarks about the different groups of random number generators in order to compare them and to find an adapted method for different kinds of simulations.

- LCG and Shift Register generators have strong uniformity properties but their lattice structure can make them unsuitable in many applications.
MRCG, MRMM and combined generators have a very large period.
The period length for Shift Register generator is not bounded in terms of the word length of the machine.
- Inversive generators provide true randomness with reference to the s-dimensional serial test and a lack of lattice structure. The most promissing method seems to be the digital inverse method, which combines nonlinear method advantage and fast algorithm.

References

[1] D.E.KNUTH. *The Art of Computer programming, Seminumerical Algorithms*, volume 2. Addison-Wesley, 1981. 2, 3

[2] P. L'ECUYER. Maximally equidistributed combined tausworthe generators. 10

[3] P. L'ECUYER. Random numbers for simulation. *Communications of the ACM*, 33(10), Octobre 1990. 2

[4] P. L'ECUYER. Uniform random number generation. *The Annals of Operations Research*, 53:77–120, 1994. 2, 3

[5] P. L'ECUYER. Random number generation. In *In the Hanbook of Simulation*. 1998. 2

[6] P. L'ECUYER. Good parameters and implementations for combined multiple recursive random number generators. *Shorter version in Operations Research*, 47(1):249–260, 1999. 7

[7] H. NIEDERREITER. Random number generation and quasi-monte carlo methods. *SIAM*, 1992. 2

[8] H. NIEDERREITER. New developments in uniform pseudorandom number and vector generation. In Springer, editor, *In Lecture Notes in Statistics, 106 : Monte Carlo and Quasi-Monte Carlo Methods in Scientific Computing*, volume 106, pages 87–120, 1994. 2, 11, 12