Available online at www.sciencedirect.com
ScienceDirect
Mathematics and Computers in Simulation 79 (2009) 3328–3338
MATHEMATICS AND COMPUTERS IN SIMULATION
www.elsevier.com/locate/matcom

Polynomial pseudo-random number generator via cyclic phase

A. Marchi, A. Liverani, A. Del Giudice *

Catholic University of Sacred Heart, Economics and Business Administration department, 7 Necchi St., 20123 Milan, Italy

Received 12 December 2005; received in revised form 16 September 2008; accepted 12 May 2009
Available online 23 May 2009

Abstract

Fast and reliable pseudo-random number generator (PRNG) is required for simulation and other applications in scientific computing. In this work, a polynomial PRNG algorithm, based on a linear feedback shift register (LFSR) is presented. LFSR generator of order k determines a $2^k - 1$ cyclic sequence period when the associated polynomial is primitive. The main drawback of this generator is the cyclicality of the shifted binary sequence. A non-linear transformation is proposed, which eliminates the underlying cyclicality and maintains both the characteristics of the original generator and the feedback function. The modified generator assures a good *trade off* between fastness and reliability and passes both graphical and statistical tests.

Keywords: Monte carlo simulation; Random number; Pseudo-random number generator shift register

1. Introduction

Monte Carlo methods are of great importance in simulation, computational finance, numerical integration and many other fields of research. The statistical quality of PRNG is becoming even more important than that in the past because the actual simulations might require huge random number sequences. Small correlations and other deficiencies in PRNG could easily lead to spurious effect and invalidate the result of the computation [2].

Although applications require random numbers with various distributions (e.g. Poisson, normal, exponential), the algorithms used to generate these random numbers require a good uniform PRNG. A good PRNG must have long period and fast run, should not waste memory, be repeatable and portable (able to reproduce the same sequence in different software hardware environment), and allows efficient jumping ahead in order to obtain multiple streams and sub streams.

Given the computers work in binary arithmetic, a fast uniform PRNG must be defined by few operations on bit strings [7,8], such as shifts, rotations, exclusive-or's (XOR) and bit mask.

A Linear Feedback Shift Register (LFSR) is a uniform PRNG. The number of values that an LFSR cycles through before repeating is its period. LFSRs that are maximum period cycle through 2^{n-1} values before repeating, where n is the width of the register.

To obtain a maximum period LFSR, the choice of the bits used for the XOR mask is critical. The mask bits correspond to the terms of a primitive polynomial modulo 2. LFSR of order k determines a $2^k - 1$ cyclic sequence period when

* Corresponding author Tel.: +39 02 72342912; fax: +39 02 72342670.
E-mail addresses: angelo.marchi@unicatt.it (A. Marchi), antonio.liverani@unicatt.it (A. Liverani), alfonso.delgiudice@unicatt.it (A. Del Giudice).

0378-4754/$36.00 © 2009 IMACS. Published by Elsevier B.V. All rights reserved.
doi:10.1016/j.matcom.2009.05.006

the associated polynomial is primitive. The main drawback of this generator is the cyclicality of the shifted binary sequence. A non-linear transformation of the original characteristics is proposed, which eliminates the underlying cyclicality and maintains both the characteristics of the generator and the feedback function.

In this paper a LSFR is defined using a 96th degree primitive polynomial. Introducing rows and columns perturbations, different algorithms are created in assembly code. The computations of both the original and the perturbed algorithms allow to compare the trade off between cyclicality and speed of generators. Perturbed LSFRs eliminate cyclicality and require a time generation increase of 28%.

2. Mathematical framework

The following polynomials $f(x)$:

$$f(x) = f_0 + f_1 x + f_2 x^2 + \cdots + f_n x^n; \quad f_i \in \mathbb{GF}(p^n) \tag{1}$$

are assigned to a Galois field $\mathbb{GF}(\circ)$. If a generic algebra $A(n)$ is defined in $\mathbb{GF}(\circ)$ and modular arithmetic modulo p [12] is used, the algebraic field $\mathbb{Z}_p$ is the ring of quotients modulo the prime p (at least for the multiplication operator).

For example, addition of two polynomials in $\mathbb{GF}(2)$ and $\mathbb{Z}_3(x)$ modulo $(x^3 + x + 1)$ is:

$$f_1(x) = x^3 + 1; \quad f_2(x) = x^2 + x \tag{2}$$

$$f(x) = \sum_i f_i(x) = f_1(x) + f_2(x) = x + 1 \tag{3}$$

The polynomial in Eq. (3) lies into a $\mathbb{GF}(2)$ [5]. The multiplication of two polynomials in $\mathbb{GF}(2)$ and $\mathbb{Z}_3(x)$ modulo $(x^3 + x + 1)$ is:

$$f_1(x) = x^3 + 1; \quad f_2(x) = x^2 + x \tag{4}$$

$$\begin{aligned} f(x) &= \prod_i f_i(x)(\text{mod} x^3 + x + 1) \\ &= f_1(x) \times f_2(x)(\text{mod} x^3 + x + 1) \\ &= x^3 \end{aligned} \tag{5}$$

The polynomial $(x^n - 1)$ can be factored into all $(x^d - 1) \in \mathbb{Q}(x)$ polynomials, where $\mathbb{Q}(x)$ is the field of quotient polynomials (irreducible in $\mathbb{Q}(x)$). The quotient of $(x^n - 1)$ for all these factors d (with $d < n$) is called the n-th cyclotomic polynomial Φ_n.

If a number n is factored into primes $p_1, \ldots, p_d$, from Euler's Φ function Eq. (6) is obtained:

$$\phi(n) = n \left(1 - \frac{1}{p_1}\right) \cdots \left(1 - \frac{1}{p_d}\right) \tag{6}$$

For the n-th cyclotomic polynomial, the notation is:

$$\Phi_n = \prod (x - \xi) \tag{7}$$

and the equation is:

$$x^n - 1 = \prod_{d|n} \Phi_d(x) \tag{8}$$

The n-th cyclotomic polynomial is obtained by the quotient of the polynomial $x^n - 1$ divided by all the associated polynomials of degree $d \in \mathbb{Q}(x)$ [10].

For example, the following polynomials are considered:

$$\begin{aligned}&\Phi_1(x) = x - 1\\&\Phi_2(x) = x + 1\\&\Phi_3(x) = x^2 + x + 1\end{aligned} \tag{9}$$

The 6-th cyclotomic polynomial can be determined as follows:

$$\Phi_6(x) = \frac{x^6 - 1}{(x-1)(x+1)(x^2+x+1)} = x^2 - x + 1 \tag{10}$$

If the vector $u' = (u_{n-1}, u_0, \ldots, u_{n-2})$, obtained by a cyclic shift of the elements, is a cyclic vector in C for each vector of n elements $u = (u_0, \ldots, u_{n-2}, u_{n-1}) \in C$, then the code C is cyclic [9].

A cyclic code vector can be represented by a state polynomial $s(x)$ and a generator polynomial $g(x)$:

$$u(x) = s(x)g(x) \tag{11}$$

If the generator polynomial $g(x)$ is $r < n$, then a remainder from the division is obtained and the cycle is non-maximum.

The scope is to generate a pseudo-random number (PRN) sequence with maximum period for the generator polynomial used. This requires a primitive polynomial of degree n [17].

3. Linear feedback function

The feedback function in an LFSR has several names: XOR, odd parity, sum modulo 2 [14]. Whatever the name, the function is a feedback on the PRN.

The LFSR is based on the following linear recurrence:

$$x_n = a_1x_{n-1} + a_2x_{n-2} + \cdots + a_kx_{n-k}(\text{bmod}2) \tag{12}$$

where $k > 1$ is the order of the recursion, $a_k = 1$ and $a_j \in \{0, 1\} \forall j$.

We obtain a cyclic recursion with maximum cycle $2^k - 1$ if the characteristic associated polynomial $P(z) = -\sum_{i=0}^{k} a_i z^{k-i}$ (with $a_0 = -1$) is primitive in $\mathbb{GF}(2)$ [6]. Eq. (13) shows the feedback function which defines the new input for the generator [3,6]:

$$f(x) = c_0 + c_1x + c_2x^2 + \cdots + c_{n-1}x^{n-1} = \sum_{i=0}^{n-1} c_i x^i \tag{13}$$

where $f(x)$ is the feedback function and the coefficients $c_i \in \mathbb{GF}(2) \forall i$ with $c_n = 1$ by definition [16].

4. Columns perturbation

A shift register [1] is the shift of one or more positions of the elements of the vector $\vec{a}$, which is called register. We can have a left or right shift. The shifted elements fall off the end step by step.

In the present work the following conditions are considered: a one position (1 bit) left shift is allowed; the Linear Feedback function determines the new entry in the empty position.

Changing the form of the notation, Eq. (14) is obtained:

$$f(\vec{s}^{\ k}) = f(s_0^k, \ldots, s_{n-1}^k) = \sum_{i=0}^{n-1} c_i s_i^k \tag{14}$$

where $(s_0^k, \ldots, s_{n-1}^k)$ is the initial state vector of length n at the k th step and $c_i \in \mathbb{GF}(2) \forall i$.

The initial value of the function (which is recursive in k steps) is the initial state. Parameter k identifies the next step of PRN generation code. Since $\mathbb{GF}(2)$ arithmetic is adopted, the length of the initial state vector must be n to obtain a maximum period generator. Since the initial state $\vec{s} = (0, \ldots, 0)$ cannot be used, the maximum period $2^n - 1$ is obtained from a primitive polynomial of degree n associated to an initial state vector of length n [15,18].

For a primitive polynomial in $\mathbb{GF}(2)$, the associated coefficients c_i of the function (2) are obtained; the polynomial is considered primitive normal [19].

For example, a primitive polynomial of 3rd degree $(x^3 + x^2 + 1)$is:

$$\begin{aligned}
(x^3 + x^2 + 1) &= (c_2 x^3 + c_1 x^2 + c_0 x + 1) \\
&= (1 \cdot x^3 + 1 \cdot x^2 + 0 \cdot x + 1) \\
\leftrightarrows \vec{c} &= [c_2, c_1, c_0] \\
&= [1, 1, 0]
\end{aligned} \tag{15}$$

Once the feedback function is determined, the shift register can be defined. The initial state $\vec{s}^0$ is both the generator's seed and the initial value (the register or vector).

For each step k starting from an initial vector $(1,\ 0,\ 1)$ with $\vec{c} = [1,\ 1,\ 0]$, the first number on the left will be dropped out, and, at the same time, the feedback function result will be entered on the last right empty position. The results from Eq. (14) are showed in the following matrix.

It is possible to see the cyclicality associated to the polynomial used to determine the PRN sequence. For $\vec{c} = [1,\ 1,\ 0]$:

$$\vec{s} = \begin{bmatrix} 1 & 0 & 1 & f(\vec{s}^0) = 1 \\ 0 & 1 & 1 & f(\vec{s}^1) = 1 \\ 1 & 1 & 1 & f(\vec{s}^2) = 0 \\ 1 & 1 & 0 & f(\vec{s}^3) = 0 \\ 1 & 0 & 0 & f(\vec{s}^4) = 1 \\ 0 & 0 & 1 & f(\vec{s}^5) = 0 \\ 0 & 1 & 0 & f(\vec{s}^6) = 1 \end{bmatrix} \text{;decimal vector associated} \begin{bmatrix} 5 \\ 3 \\ 7 \\ 6 \\ 4 \\ 1 \\ 2 \end{bmatrix} \tag{16}$$

The columns are cyclical mutual shifts. For example, the second column is obtained by 3 positions shift from the first; the third is obtained by 1 position shift from the second. Fig. 1(a) represents the binary matrix associated to the PRNG.

Diagonal bands represent the one position shift columns; the vertical band up on the left points out the initial state; the vertical band down on the right points out the complementary of the initial state $\mathbb{GF}(2)$. For the generic primitive polynomial (normal) of degree n the vertical bands are of length n. Starting from the n-th position, the same shift is adopted for each polynomial function and each state. For example, a 4-th degree primitive polynomial $(x^4 + x + 1)$, the associated coefficients c_i, $[c_3,\ c_2,\ c_1,\ c_0] = [1,\ 0,\ 0,\ 1]$ and initial state: $\vec{s} = (s_3,\ s_2,\ s_1,\ s_0) = (0,\ 0,\ 0,\ 1)$ are considered. Fig. 1 (b) represents the columns.

The examples displayed in Fig. 1 can be generalized. This band configuration can be obtained for all the polynomials of degree n. The columns have two main properties: each column is a shift of the preceding (and vice versa); the columns are linearly independent vectors [19].

The second property is very interesting: since the columns are linearly independent, it is possible to change their position in the matrix. If the columns are exchanged, the same numbers are produced in different positions. Exchanging columns implies that it is possible to obtain a perturbed sequence from the original sequence. Since the columns are independent, column permutations are adopted. All the possible matrix permutations define a perturbation on the original sequence. The associated permutations of an n element series are $n!$.

If the example of Fig. 1(a) is considered, all possible initial state are $2^n - 1 = 7$. Two series are obtained: the first starts from $\vec{s}^0 = (1,\ 0,\ 1)$ and the second from $\vec{s}^0 = (0,\ 1,\ 1)$. The second series is a shift of the first. All possible permutations of the $\vec{s}^0$ matrix columns are computed. The possible permutations are: $[1,\ 2,\ 3]$; $[1,\ 3,\ 2]$; $[2,\ 1,\ 3]$; $[2,\ 3,\ 1]$; $[3,\ 1,\ 2]$; $[3,\ 2,\ 1]$ (Figs. 2 and 3).

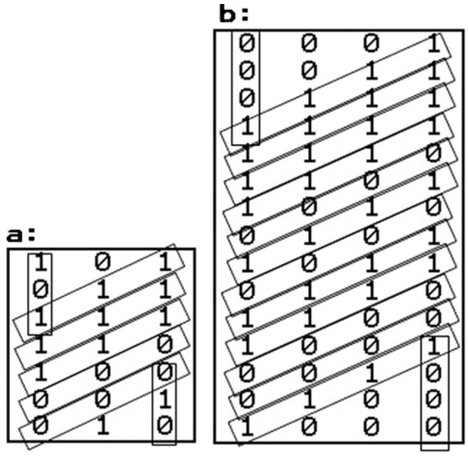

Fig. 1. Cyclical binary matrix for polynomials (a) $x^3 + x^2 + 1$ and (b) $x^4 + x + 1$.

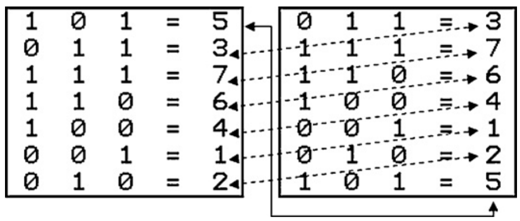

Fig. 2. Output associated to states $\vec{s}^{\,0}$, $\vec{s}^{\,1}$.

From a monic (normal) primitive polynomial it is possible to obtain $n!$ PRN perturbed series as results of simple permutations of the binary matrix. Considering a binary mode, LFSR is cyclic by construction. A permutation on the binary matrix columns cannot eliminate the cyclicality (see Appendix A).

Unfortunately residual cycles persist [4,11]; it is necessary to add another permutation tool on the binary matrix rows.

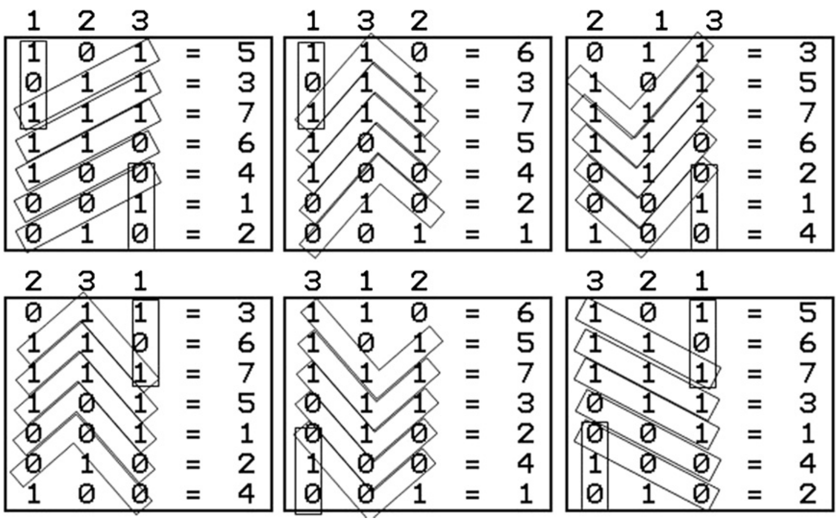

Fig. 3. Cyclical matrix referred to all permutations on $\vec{s}^{\,0}$.

Table 1
First hexadecimal digit in a 20 bit register.

Case **FF**	BIN	HEX
REGISTERCHARGED	[1111 1001 1101 0010 1010]	F 9 D 2 A
SHIFT LEFT1BIT	1111 0011 1010 0101 0101]	F 3 A 5 5
Case **FE**	BIN	HEX
REGISTERCHARGED	1111 0011 1101 0010 1010]	F 3 D 2 A
SHIFT LEFT1BIT	[1110 0111 1010 0101 0101]	E 7 A 5 5

5. Rows perturbation

The cycle on the binary series modifies the decimal and hexadecimal series too. Passing from binary to hexadecimal series an interesting property is noted. In a not permutated shift register each new number is obtained by shifting one bit left from the preceding binary string. Each hexadecimal digit is made up by 4 bits, yet each following number is only one bit shifted.

If all couples of the first digits from hexadecimal number series are considered, some of these couples cannot exist. The occurrence probability of an hexadecimal couple is 256^{-1}.

For example, the consecutive couple **F0** (two hexadecimal digit) cannot exist ($\mathbf{F} := [1\ \ 1\ \ 1\ \ 1]$ and $\mathbf{0} := [0\ \ 0\ \ 0\ \ 0]$). In fact, if the next number is obtained by one binary position shift of the preceding, starting from the number $\mathbf{F} := [1\ \ 1\ \ 1\ \ 1]$ the first hexadecimal digit of the next number has two possible states:

$$\mathbf{F} := [1\ \ 1\ \ 1\ \ 1] \quad \text{if queued bit is1}$$
$$\mathbf{E} := [1\ \ 1\ \ 1\ \ 0] \quad \text{if queued bit is0}$$

In this condition, the possible couples are **FF** or **FE**. For example in a 20 bit register[1] the first hexadecimal digit could be reported as in Table 1.

The row and column permutations together can solve this problem. The row permutation is obtained using $\mathbf{T}'$, the transposed matrix of $\mathbf{T}$ in (17), a 64 cyclic distance matrix with distances length 8; the distances are computed with respect to the generator dimension. While $\mathbf{T}$ and $\mathbf{T}'$ identifies memory allocations order, $\mathbf{T}'$ is a closed cyclical perturbation on print locations.

$$\mathbf{T}_{(8\times 8)} = \begin{pmatrix} 1 & 2 & \dots & 7 & 8 \\ 9 & 10 & \dots & 15 & 16 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 49 & 50 & \dots & 55 & 56 \\ 57 & 58 & \dots & 63 & 64 \end{pmatrix} \tag{17}$$

Once the counter value reaches 64, the matrix is complete and all the numbers are cyclically distributed. The pseudo-random numbers are reported in independent closed cycles of length 64. This cyclic matrix is used in the generator presented in this paper.

Smaller cyclic matrices can be derived, for example a 16 dimension matrix with distances length 4.

Using $\mathbf{T}'$ the first generated number 1 is printed in the location number 1, while the second is printed in location number 9 and so on.

The cycle allows the binary shift to work for 8 digits and the cyclic print allows to outdistance the generated numbers so that each number is completely independent. No additional computing is required to the generated numbers.

[1] In this example the scope is to show the limits shifting 1 bit left in a register, independently of the new bit input in the queue of the register.

Table 2
Register rotation (columns perturbation).

Register	Part of register	Rotation	Bits involved
1	DWORD	↺	20
1	lower WORD	↻	15
1	lower BYTE	↺	4
2	DWORD	↺	12
2	lower WORD	↻	8
2	lower BYTE	↺	3
3	DWORD	↺	18
3	lower WORD	↻	5
3	lower BYTE	↺	7

Table 3
Generation run time.

Generator	Numbers	Time (s)
Classic	35,000,000	1.054688
Perturbed	25,000,000	1.062500

This operation is executed in assembly code with minimum time consuming because of the direct allocation of the numbers in predefined positions.

6. The generators

Three dynamic libraries (in DLL format) are created in order to generate a Linear Feedback Shift Register using 686 assembly code on a Pentium 4 PC-Desktop. A 96th degree polynomial ($x^{96} + x^{49} + x^{47} + x^2 + 1$) is used.

The polynomial is primitive and allows a $2^{96} - 1$ generation period [13]. The generators are:

1. "`LFSR_96`" *Classic generator*: one bit left shift generator.
2. "`LFSR_96_C`" *Columns perturbation generator*: the program executes a bitwise bit rotation inside the registers in order to perturb the generated binary strings columns at each generation. This is an extremely fast way, but it is possible to build different perturbation methods.
3. "`LFSR_96_CR`" *Rows and columns perturbation generator*: rows permutation is executed together with the columns permutation. The algorithm generates numbers at determined cyclic distances.

The row perturbations do not require computations but only a different print allocation. The generated number is transferred into DLL code; run time depends by the code.

Table 2 reports the register's rotations applied to generators "`LFSR_96_C`" and "`LFSR_96_CR`", where:[2] ↺ := rotate left; ↻ := rotate right.

Table 3 shows the pure generation timing summary.

Classic generation use a classic LSFR without permutations while perturbed generation use a LSFR with rows and columns perturbation. The perturbed generator slows generation time (+28%). Time values reported in Table 3 are just the times for the pseudo-random number generation.

7. Graphical and statistical tests on generator "LFSR_96_CR"

We use 2D lattices to report on the output of generators.

In Fig. 4 the three lattice are made for distance length respectively of $d = 1$; $d = 6$; $d = 7$; using the *Classical Generator* "`LFSR_96`". The plot shows uniform distribution only at seven distance length.

[2] Every routine associated to generators use three registers of 32 bits for computing every number of $2^{96} - 1$ generation.

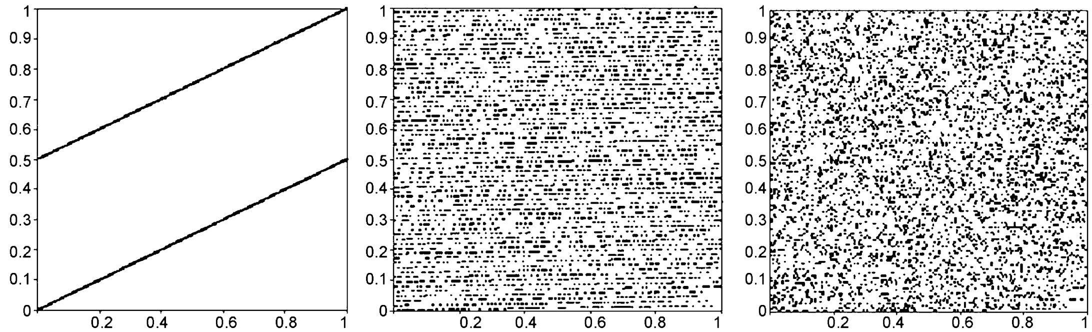

Fig. 4. "LFSR_96" generation for $d = 1$; $d = 6$; $d = 7$.

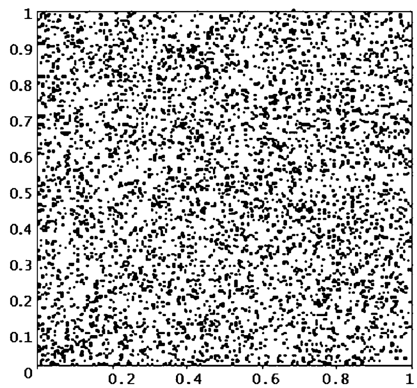

Fig. 5. "LFSR_96_CR" generation for $1 \leq d \leq 7$.

Table 4
Scores criteria of statistical tests.

Score	Legend	**V**
S	Pass test	$0.1 < V < 0.95$
A1	Warning	$0.05 \leq V \leq 0.1$
A2	Warning	$0.95 \leq V \leq 0.99$
F	Fail test	$V > 0.99 \wedge V < 0.05$

In Fig. 5 all the 2D lattice, for distances from $d = 1$ to $d = 7$ are the same, using *Rows and columns perturbation generator* "LFSR_96_CR": we obtain soon uniform distribution on $d = 1$, which is maintained in forward distances $d = 2, \ldots, 7$.

The"LFSR_96_CR" generator passed the most important statistical tests [5].

In Fig. 6 are reported the tests made on the decimal series obtained by the generator,[3] for different sample length; any box display the output of test respectively: serial, coupon collector, gap, frequency, Kolmogorov–Smirnov, run.

Any test return a specific output score in relation to the statistic V associated to the distribution; Table 4 shows the scores criteria.

Serial, collector, gap and frequency tests are made on sub-sequences obtained considering parts of the decimal digits composing the numbers: consecutive single digits, couples, triples. K–S and run test are made on the full float number by construction of the test. The generator seems to have a good behavior on short series ($n \leq 10,000$) and long series ($40,000 \leq n \leq 50,000$), failing in a very few times.

[3] The sequences $s_i \in [0, 1]$: generation output numbers are standardized (are considered 15 digit floating-point numbers).

GENERATOR CRITERIA				SCORE PER DIGIT													
TEST	CONS. DIGIT	SAMPLE	SERIAL NUMBERS	D01	D02	D03	D04	D05	D06	D07	D08	D09	D10	D11	D12	D13	D14
SERIAL	1	10,000	PAIR	S	S	S	S	S	S	S	S	S	S	S	S	S	S
SERIAL	1	50,000	PAIR	S	S	S	S	S	S	S	S	S	S	S	S	S	S
SERIAL	1	10,000	TRIPLES	S	S	S	S	S	S	S	S	S	S	S	S	S	S
SERIAL	1	50,000	TRIPLES	S	S	S	S	S	S	S	S	S	S	S	S	S	S
SERIAL	2	10,000	PAIR	S	S	S	S	S	S	S							
SERIAL	2	50,000	PAIR	F	S	S	S	S	S	A1							

GENERATOR CRITERIA				SCORE PER DIGIT													
TEST	CONS. DIGIT	SAMPLE	SET	D01	D02	D03	D04	D05	D06	D07	D08	D09	D10	D11	D12	D13	D14
COLLECTOR	1	10,000	10	S	S	S	S	S	S	S	S	S	S	S	S	S	S
COLLECTOR	1	10,000	10	S	S	S	S	S	S	S	S	S	S	S	S	S	S
COLLECTOR	1	10,000	10	S	S	S	S	S	S	S	S	S	S	S	S	S	S
COLLECTOR	1	50,000	10	S	S	S	S	S	S	S	S	S	S	S	S	S	S
COLLECTOR	1	50,000	10	S	S	S	S	S	S	S	S	S	S	S	S	S	S
COLLECTOR	1	50,000	10	S	S	S	S	S	S	S	S	S	S	S	S	S	S

GENERATOR CRITERIA				SCORE PER DIGIT													
TEST	CONS. DIGIT	SAMPLE	ALFA; BETA	D01	D02	D03	D04	D05	D06	D07	D08	D09	D10	D11	D12	D13	D14
GAP	1	10,000	00; 05	S	S	S	S	S	A2	S	A1	S	S	A1	S	S	S
GAP	1	10,000	05; 10	S	S	S	A2	S	S	S	S	S	S	S	S	S	S
GAP	1	10,000	02; 07	S	S	F	S	S	S	S	S	S	S	S	S	S	S
GAP	1	50,000	00; 05	S	S	S	S	S	S	A2	S	S	S	S	S	S	S
GAP	1	50,000	05; 10	S	S	S	S	S	S	S	S	S	S	S	S	S	S
GAP	1	50,000	02; 07	A1	S	S	S	S	S	S	S	S	A1	S	S	S	S

GENERATOR CRITERIA				SCORE PER DIGIT													
TEST	CONS. DIGIT	SAMPLE	FREQUENCY	D01	D02	D03	D04	D05	D06	D07	D08	D09	D10	D11	D12	D13	D14
FREQUENCY	2	10,000	100	S	S	S	S	S	S	S							
FREQUENCY	2	10,000	100	S	S	S	S	S	S	S							
FREQUENCY	4	10,000	100	S	S	S											
FREQUENCY	2	50,000	100	S	S	S	S	S	S	S							
FREQUENCY	2	50,000	100	S	S	S	S	S	S	S							
FREQUENCY	4	50,000	100	S	S	S											

GENERATOR CRITERIA									SCORE	
TEST	DIGIT	SAMPLE	FREQUENCY	K_PLUS	K_MINUS	FUNCTION	LAMBDA	GAMMA_R	K_PLUS	K_MINUS
K-S	ALL	10,000	100	0.81	0.41	UNIFORM			S	S
K-S	ALL	10,000	100	2.62	0.23	EXP	4		F	S
K-S	ALL	10,000	100	0.30	0.24	GAMMA	4	5	S	S
K-S	ALL	50,000	100	1.01	0.58	UNIFORM			S	S
K-S	ALL	50,000	100	1.90	0.08	EXP	4		F	A2
K-S	ALL	50,000	100	0.24	0.44	GAMMA	4	5	S	S

GENERATOR CRITERIA					SCORE
TEST	DIGIT	SAMPLE	OSCILLATION	RUNS	
RUN	ALL	30,000	11.527	UP	S
RUN	ALL	30,000	14.806	DOWN	A1
RUN	ALL	40,000	16.088	UP	A1
RUN	ALL	40,000	11.085	DOWN	S

Fig. 6. Serial, collector, gap, frequency, K–S, run tests for “LFSR_96_CR”.

8. Conclusion

In this paper the characteristics of the Linear Feedback Shift Register are presented. It is possible to optimize the generation unchanging the original structure of the generator.

LFSR can be programmed in assembly code to have high performances and fast executions. The perturbed LSFR algorithms reduce the speed of execution (28% increase in time consuming) but eliminate the cyclicality of the output. This methods allow to produce high quality random number sequences with good time performances. The trade off between fastness and quality is balanced, and the perturbed algorithms can be both of theory and practical application interest.

Appendix A.

In Fig. A.1 are reported three samples of sequences generated by the generators "LFSR_96", "LFSR_96_C", "LFSR_96_CR". Sequences, in hexadecimal and binary digits, show immediately the cyclical behavior of the standard LFSR "LFSR_96".

LFSR_96 HEXADECIMAL	LFSR_96 BINARY
D72B8B4B1D58DEAABD4506B4	110101110010101110001011010010110001110101011000110111101010101010111101010001010000011010110100
AE5716963AB1BD557A8A0D68	101011100101011100010110100101100011101010110001101111010101010101111010100010100000110101101000
5CAE2D2C75637AAAF5141AD0	010111001010111000101101001011000111010101100011011110101010101011110101000101000001101011010000
B95C5A58EAC6F555EA2835A0	101110010101110001011010010110001110101011000110111101010101010111101010001010000011010110100000
72B8B4B1D58DEAABD4506B40	011100101011100010110100101100011101010110001101111010101010101111010100010100000110101101000000
E5716963AB1BD557A8A0D680	111001010111000101101001011000111010101100011011110101010101011110101000101000001101011010000000
CAE2D2C75637AAAF5141AD01	110010101110001011010010110001110101011000110111101010101010111101010001010000011010110100000001
95C5A58EAC6F555EA2835A02	100101011100010110100101100011101010110001101111010101010101111010100010100000110101101000000010
2B8B4B1D58DEAABD4506B404	001010111000101101001011000111010101100011011110101010101011110101000101000001101011010000000100
5716963AB1BD557A8A0D6808	010101110001011010010110001110101011000110111101010101011110101000101000001101011010000000001000
AE2D2C75637AAAF5141AD010	101011100010110100101100011101010110001101111010101010101111010100010100000110101101000000010000
5C5A58EAC6F555EA2835A021	010111000101101001011000111010101100011011110101010101011110101000101000001101011010000000100001
B8B4B1D58DEAABD4506B4042	101110001011010010110001110101011000110111101010101010111101010001010000011010110100000001000010
716963AB1BD557A8A0D68084	011100010110100101100011101010110001101111010101010101111010100010100000110101101000000010000100
E2D2C75637AAAF5141AD0108	111000101101001011000111010101100011011110101010101011110101000101000001101011010000000100001000
C5A58EAC6F555EA2835A0211	110001011010010110001110101011000110111101010101010111101010001010000011010110100000001000010001

LFSR_96_C HEXADECIMAL	LFSR_96_C BINARY
6B4BA81A8DEAAEA12D2F39E5	011010110100101110101000000110101000110111101010101011101010000100101101001011110011100111100101
D68751141BD55D535A5AF2CA	110101101000011101010001000101000001101111010101010111010101001101011010010110101111001011001010
AD0FA22837AAB2A7B4B1E195	101011010000111110100010001010000011011110101010101100101010011110110100101100011110000110010101
5A0E45706F55655E6962C72B	010110100000111001000101011100000110111101010101011001010101111001101001011000101100011100101011
B40D8AC0DEAAC2BDD2C50B57	101101000000110110001010110000001101111010101010110000101011110111010010110001010000101101010111
680A14B1BD558D7AA58F16AE	011010000000101000010100101100011011110101010101100011010111101010100101100011110001011010101110
D01528437AAA1BF54B1FAC5C	110100000001010100101000010000110111101010101010000110111111010101001011000111111010110001011100
A02A50A6F55536EA963A59B8	101000000010101001010000101001101111010101010101001101101110101010010110001110100101100110111000
4044A06DEAAB6CD52C74B671	010000000100010010100000011011011110101010101011011011001101010100101100011101001011011001110001
808841DAD557D8AB58E969E2	100000001000100001000001110110101101010101010111110110001010101101011000111010010110100111100010
010183A5AAAFB956B1D6D2C5	000000010000000110000011101001011010101010101111101110010101011010110001110101101101001011000101
0212065B555E7BAC63A9A58B	000000100001001000000110010110110101010101011110011110111010110001100011101010011010010110001011
04250D86AABDF648C756CB16	000001000010010100001101100001101010101010111101111101100100100011000111010101101100101100010110
084A1A0D557AED818EAD172D	000010000100101000011010000011010101010101111010111011011000000110001110101011010001011100101101
1084350AAAF5D3131D5B2E5A	000100001000010000110101000010101010101011110101110100110001001100011101010110110010111001011010
21186B0455EAAF263AB358B4	001000010001100001101011000001000101010111101010101011110010011000111010101100110101100010110100

LFSR_96_CR HEXADECIMAL	LFSR_96_CR BINARY
6B4BA81A8DEAAEA12D2F39E5	011010110100101110101000000110101000110111101010101011101010000100101101001011110011100111100101
4044A06DEAAB6CD52C74B671	010000000100010010100000011011011110101010101011011011001101010100101100011101001011011001110001
4230D608ABD4575D7562B069	010000100011000011010110000010001010101111010100010101110101110101110101011000101011000001101001
373B8048D4505D6A6379D463	001101110011101110000000010010001101010001010000010111010110101001100011011110011101010001100011
383084E6506BA64B7AA88DAB	001110000011000010000100111001100101000001101011101001100100101101111010101010001000110110101011
34326E076B408244AAF5E91B	001101000011001001101110000001110110101101000000100000100100010010101010111101011110100100011011
3887708640425B30F517A9D5	001110001000011101110000100001100100000001000010010110110011000011110101000101111010100111010101
8EF868074237023B141AD657	100011101111100001101000000001110100001000110111000000100011101100010100000110101101011001010111
D68751141BD55D535A5AF2CA	110101101000011101010001000101000001101111010101010111010101001101011010010110101111001011001010
808841DAD557D8AB58E969E2	100000001000100001000001110110101101010101010111110110001010101101011000111010010110100111100010
8460AD1057A8AEABEAC461D2	100001000110000010101101000100000101011110101000101011101010101111101010110001000110000111010010
6E760190A8A0BAD5C6F629C7	011011100111011000000001100100001010100010100000101110101101010111000110111101100010100111000111
706008DDA0D64587F5549B56	011100000110000000001000110111011010000011010110010001011000011111110101010101001001101101010110
6874DC0ED680058855EA5737	011010000111010011011100000011101101011010000000000001011000100001010101111010100101011100110111
711EE00D8084B660EA2AD7AA	011100010001111011100000000011011000000010000100101101100110000011101010001010101101011110101010
1DE0D02E846E0476283529AF	000111011110000011010000001011101000010001101110000001000111011000101000001101010010100110101111

Fig. A.1. Hexadecimal and binary pattern for "LFSR_96", "LFSR_96_C", "LFSR_96_CR".

The binary pattern became better with columns perturbation (applied in "LFSR_96_C"), and much better with columns and rows perturbations together (applied in "LFSR_96_CR").

References

[1] S.L. Anderson, Random number generators on vector supercomputers and other advanced architectures, SIAM Review 32 (2) (1990) 221–251.
[2] A.M. Ferrenberg, D.P. Landau, Y.J. Wong, Monte Carlo simulations: hidden errors from good random number generators, Physical Review Letters 69 (1992) 3382–3384.
[3] J.E. Gentle, Random Number Generation and Monte Carlo Methods, Springer, New York, 1998.
[4] P. Hellekalek, Good random number generators are (not so) easy to find, Mathematics and Computers in Simulations 46 (1998) 485–505.
[5] E.D. Knuth, The Art of Computer Programming: Seminumerical Algorithms, in: Addison-Wesley, Series in Computer Science and Information Processing, Addison-Wesley Readings, 1981.
[6] P. L'Ecuyer, F. Panneton, A new class of linear feedback shift register generators, in: Proceedings of the 2000 Winter Simulation Conference, Departement d'Informatique et de recherche Operationelle, Universit de Montreal, 2001.
[7] M. Matsumoto, Y. Kurita, Twister GFSR generators II, ACM Transactions on Modelling and Computer Simulation 4 (1994) 254–266.
[8] M. Matsumoto, T. Nishimura, Mersenne Twister: a 623 dimensionally equidistributed uniform pseudo-random number generator, ACM Transactions on Modelling and Computer Simulation 8 (1998) 3–30.
[9] W. Micheal, Arithmetic in a finite field, Mathematics of Computation 35 (152) (1980) 1353–1359.
[10] J.S. Milne, 2003. Fields and Galois theory, http://www.jmilne.org/math/.
[11] H. Niederreiter, C.P. Schnorr, Local randomness in polynomial random number and random function generators, SIAM Journal Computation 22 (4) (1993) 684–694.
[12] J.J. Rotman, Galois theory, Springer, New York, 1990.
[13] W. Stahnke, Primitive binary polynomials, Mathematics of Computation 27 (24) (1973) 977–980.
[14] R.C. Tausworthe, Random numbers generated by linear recurrence modulo two, Mathematics of Computation 19 (1965) 201–209.
[15] S. Tezuka, On optimal GFSR pseudo-random number generator, Mathematics of Computation 50 (182) (1988) 531–533.
[16] S. Tezuka, The *K*-dimensional distribution of combined GFSR sequences, Mathematics of Computation 62 (206) (1994) 809–817.
[17] J.H. Van Linth, F.J. MacWilliams, M.J.A. Sloane, On pseudo-random arrays, SIAM Journal on Applied Mathematics 36 (1) (1979) 62–72.
[18] D. Wang, A. Compagner, On the use of reducible polynomials as random number generators, Mathematics of Computation 60 (201) (1993) 363–374.
[19] M. Willett, Characteristic m sequences, Mathematics of Computation 30 (134) (1976) 306–311.