Common Defects in Initialization of Pseudorandom Number Generators

MAKOTO MATSUMOTO
Hiroshima University
ISAKU WADA
Tempstaff Technologies
and
AI KURAMOTO and HYO ASHIHARA
Kumamoto University

We demonstrate that a majority of modern random number generators, such as the newest version of rand.c, ranlux, and combined multiple recursive generators, have some manifest correlations in their outputs if the initial state is filled up using another linear recurrence with similar modulus. Among 58 available generators in the GNU scientific library, 40 show such defects. This is not because of the recursion, but because of carelessly chosen initialization schemes in the implementations. A good initialization scheme eliminates this phenomenon.

Categories and Subject Descriptors: G.3 [**Mathematics of Computing**]: Probability and Statistics—*Random number generation*

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Difference collision, interstream correlation, Monte-Carlo simulation, nearly affine dependence, pseudorandom number generator

ACM Reference Format:

Matsumoto, M., Wada, I., Kuramoto, A., and Ashihara, H. 2007. Common defects in initialization of pseudorandom number generators. ACM Trans. Model. Comput. Simul. 17, 4, Article 15 (September 2007), 20 pages. DOI = 10.1145/1276927.1276928 http://doi.acm.org/10.1145/1276927.1276928

This study is partially supported by JSPS/Ministry of Education Grant-in-Aid for Scientific Research No. 18654021, 16204002, and JSPS Core-to-Core Program No. 18005.
Authors' addresses: M. Matsumoto, Department of Mathematics, Hiroshima University, Kagamiyama 1-3-1, Higashi-Hiroshima, Hiroshima, Japan; email: m-mat@math.sci.hiroshima-u.ac.jp; I. Wada; email: isaku@pb4.so-net.ne.jp; A. Kuramoto and H. Ashihara, Department of Computer Science, Kumamoto University, Kurokami 2-39-1, Kumamoto, Japan; email: conpeitou_a_k@yahoo.co.jp, ashihara@cs.kumamoto-u.ac.jp.

1. INTRODUCTION

Pseudorandom number generators (PRNGs) are widely used in the area of Monte-Carlo methods. This article warns that a majority of modern standard PRNGs have common defects in the initialization scheme, in a way that correlated seeds yield fatally correlated sequences. Users of PRNGs should be aware of such dependences, and designers of PRNGs must pay more attention to the initialization scheme.

For simplicity, in this article we assume that a PRNG takes an initial seed s of w-bit integer, and outputs a sequence of uniform pseudorandom integers in a certain range $\{0, 1, 2, \ldots, M-1\}$. The output sequence generated from a seed s is denoted by $x_0(s), x_1(s), x_2(s), \ldots$.

Ideally, for any given s, these sequences should behave as if they were independent random streams. However, we show that there are many PRNGs whose nth output can be well approximated by an affine function

$$x_n(s) = a_n s + c_n + (\text{small error}) \bmod M$$

for any s. We call this type of dependence on the initial seed *nearly affine dependence*. It is observed in linear congruential generators, lagged Fibonacci generators, and subtract-with-borrow generators when the state space is initialized by linear congruential generators. Section 2 shows a typical example, and Section 3 analyzes the reasons.

We also report a more general type of dependence named *difference collision*, where the approximation

$$x_n(s+1) - x_n(s) = x_n(t+1) - x_n(t) + (\text{small error}) \bmod M$$

holds for any n and for many different pairs (s, t). It occurs for a class of RNGs that includes (combined) multiple recursive generators when their initialization scheme depends on a linear congruential generator. Section 4.2 shows typical examples, and Section 4.3 analyzes the reason behind this behavior.

2. SYMPTOM 1: AFFINE DEPENDENCE.

We start with a PRNG standardly used in several software libraries in the C language under the name of `random`. There are several versions of `random`, and here we treat its most recent (as of January 2007) version introduced about 1997 and included in the GNU `glibc2` library (for both BSD and Linux operating systems) as the default PRNG. We call this `random97` to distinguish it from its older versions.

The recursion of `random97` is

$$x_{j+31} := x_{j+28} + x_j \mod 2^{32} \quad (j = 0, 1, \ldots). \tag{1}$$

We need to set $x_0, \ldots, x_{30}$ in the initialization. In `glibc2`, these values are computed from a 32-bit integer seed s as follows.

$$x_0 := s, \quad x_{j+1} := 16807 x_j \mod (2^{31} - 1) \quad (j = 0, 1, \ldots, 29) \tag{2}$$

This generator belongs to the class of lagged Fibonacci generators (LFGs) and generates 32-bit integer sequences.

Table I lists the second most significant bit of the $(n+1)$-st output $x_n(s)$ of `random97` (as a 31-bit integer), where $n = 0, 1, 2, \ldots, 19$ in the horizontal axis

Table I. Second Most Significant Bit of the nth Output $x_n(s)$ of `random97` with Seed s, for $0 \leq n \leq 19, 0 \leq s \leq 20$

$x_n(s)$	2nd MSB ($0 \leq n \leq 19$)	
$x_n(0)$	1111101110	1010110000
$x_n(1)$	1111101110	1010110000
$x_n(2)$	0100110000	1110111111
$x_n(3)$	0011100110	0100110101
$x_n(4)$	1001011001	0000110000
$x_n(5)$	1010001111	0110111111
$x_n(6)$	0111010011	0111100001
$x_n(7)$	1100001000	0011100110
$x_n(8)$	1110101110	0101101001
$x_n(9)$	0001110000	0101100101
$x_n(10)$	0000100101	0011101010
$x_n(11)$	1010111011	0010101100
$x_n(12)$	1101001001	0110110011
$x_n(13)$	0110010110	0000111110
$x_n(14)$	1101000000	0000111000
$x_n(15)$	1001111100	0110110111
$x_n(16)$	0010100011	0011111000
$x_n(17)$	0001100101	0011110100
$x_n(18)$	1110111111	0101100011
$x_n(19)$	1100001010	0001101100
$x_n(20)$	0001010100	0011100010

and $s = 0, 1, \ldots, 20$ in the vertical axis. We see a column of pure 1s at the 15th (i.e., $n = 14$) output for $s = 0, 1, \ldots, 20$. Dependence on the seed is observed not only at 1 bit: Figure 1 shows $x_n(s)$ for $n = 10$ and $n = 14$, where s is varied from 1 to 100. One can see that if the most significant bit of the 31-bit integer $x_{10}(s)$ is removed, then the resulting 30-bit integer is approximated by an affine function of s.

This dependence was found through a simulation of baseball games. One of the authors used `random97` to simulate the scores gained by one team in a game. The $(n + 1)$-st batter's result is decided depending on the value of $x_n(s)$: If $x_n(s)/2^{31}$ exceeds one minus the hit-ratio of the $(n + 1)$-st batter, then the batter is considered to hit. The hit-ratios (of typical batters) are around 1/4. In the simulation of the sth team, the seed is chosen to be s. Then, it was noticed that the 15th batter hits with probability more than 1/2 for the first 25 teams, namely, for $1 \leq s \leq 25$. The reason is clear from Figure 1: The graph indicated by the symbol $\times$ shows that $x_{14}(s)/2^{31} > 3/4$ occurs for more than half the values of s among $1 \leq s \leq 25$. The opposite phenomenon is observed for $n = 10$: As indicated by the symbol $+$, $x_{10}(s)/2^{31} < 3/4$ holds for $3 \leq s \leq 25$. We shall analyze these phenomena in following sections.

3. ANALYSIS OF NEARLY AFFINE DEPENDENCES

3.1 Terminology

We consider PRNGs of the following type. Let Z be a finite set. Typically, Z is the residue ring of integers modulo M, that is, $Z = \mathbb{Z}/M := \{0, 1, 2, \ldots, M - 1\}$.

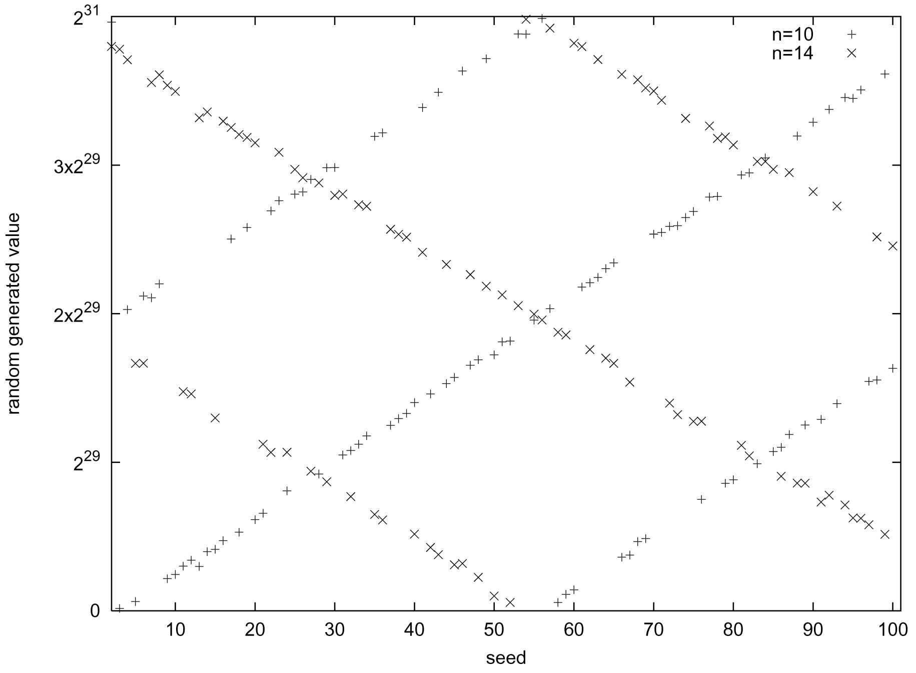

Fig. 1. The nth ($n = 10, 14$) output values of `random97` for seeds $s = 1, 2, \ldots, 100$.

We generate a sequence $z_0, z_1, \ldots \in Z$ by a pth-order recursion formula

$$z_{n+p} = f(z_{n+p-1}, z_{n+p-2}, \ldots, z_n) \quad (n = 0, 1, 2, \ldots) \tag{3}$$

by a suitable function $f : Z^p \to Z$. Let O be the set of the numbers to be produced. We assume that O the set of w-bit integers. The output of the PRNG is

$$o(z_{n_0}), o(z_{n_1}), \ldots \in O,$$

where $o : Z \to O$ is the output conversion function and n_0 the number of initial steps for which the output values are discarded (called the *warm-up*).

Before starting the generation, the PRNG must be given the values of $z_0, z_1, \ldots, z_{p-1}$. This tuple $(z_0, z_1, \ldots, z_{p-1}) \in Z^p$ is called the *initial value* for the PRNG. Often, p is rather large (say 100), and therefore it is costly to specify all of them. For this reason, the library of the PRNG usually prepares another PRNG, called an *initializer* in this article, which takes one word-integer s and produces the initial value $(z_0(s), z_1(s), \ldots, z_{p-1}(s))$. The step to produce the initial value from the seed is called *initialization*, and s is called the *seed*. After initialization, the recursion formula

$$z_{n+p}(s) = f(z_{n+p-1}(s), z_{n+p-2}(s), \ldots, z_n(s)) \quad (n = 0, 1, 2, \ldots)$$

produces the output sequence $x_n(s) := o(z_n(s))$ for $n = n_0, n_0 + 1, \ldots$.

We define an absolute value on the closed interval $[0, M]$ by

$$|x|_M := \min\{x, M - x\} \in [0, M/2] \text{ for } x \in [0, M].$$

This is applied to $x \in \mathbb{Z}/M$ by regarding $x \in \{0, 1, \ldots, M-1\}$. We extend this absolute value to arbitrary real numbers by putting

$$|x|_M := |x \bmod M|_M \text{ for } x \in \mathbb{R},$$

where $x \bmod M$ denotes the smallest nonnegative real number such that $x - (x \bmod M)$ is an integer multiple of M.

3.2 Nearly Affine Dependence

Let G be a PRNG and $x_n(s) \in \{0, 1, \ldots, M-1\}$ its nth output from the seed s, as in Section 1.

Definition 3.1. Let δ be a real number. If there are real constants a_n and c_n independent of s such that

$$|x_n(s) - (a_n s + c_n)|_M \le \delta \tag{4}$$

for all s, then we say that the nth output of G has a *nearly affine dependence* modulo M with defect (at most) δ, or with defect ratio δ/M. This implies the existence of a function $e_n(s)$ such that $|e_n(s)| \le \delta$ and

$$x_n(s) = a_n s + c_n + e_n(s) \mod M \tag{5}$$

hold for all s. If we can take $\delta = 0$, namely, if

$$x_n(s) = a_n s + c_n \bmod M \tag{6}$$

holds for all s, then we say that the nth output has an *affine dependence.*

For example, Figure 1 shows that the 10th and 14th outputs of `random97` have nearly affine dependence (with defect ratio roughly 1/20, if we look closely). A nearly affine dependence is *persistent* if observed for all n with a fixed δ, and *transient* if not observed for large enough n. The nearly affine dependence of `random97` turns out to be transient (see Section 3.5).

3.3 Linear Congruential Generators

A linear congruential generator (LCG) generates a sequence of pseudorandom numbers by the recursion

$$x_0(s) := s \mod M_{\mathrm{LCG}}, \quad x_{j+1}(s) := a x_j(s) + c \mod M_{\mathrm{LCG}}, \tag{7}$$

where a, c, M_{LCG} are integer constants and s is the seed. It is known (e.g., Knuth [1997]) that

$$x_n(s) = a^n s + c(a^n - 1)/(a-1) \mod M_{\mathrm{LCG}}$$

if $a \ne 1$, and thus we have

THEOREM 3.2. *An LCG* (*Eq.* 7) *has the persistent affine dependence* (*Eq.* 6).

Although LCGs are now obsolete, they have been widely used and are still in some software.

Among other drawbacks of LCGs, the following two are known to be serious.

(1) Its period is at most $M_{\rm LCG}$, so it is rapidly exhausted by modern computers for small values of $M_{\rm LCG}$, such as $M_{\rm LCG} \le 2^{32}$ (e.g., L'Ecuyer [1992]).
(2) If $M_{\rm LCG}$ is a power of two, then the least significant k bits of the generated sequence have period length at most 2^k (e.g., Knuth [1997, p.13]).

A typical improvement to avoid these problems is: (1) choose $M_{\rm LCG} \ge 2^{48}$ and (2) discard the least significant several bits. For example, for the `drand48` generator in the GNU C-library, $M_{\rm LCG} = 2^{48}$ and outputs only the most significant 32-bit of x_j by a shift-right operation. However, such an improvement does little for the affine dependence, as shown next. Consider the "shift-right by v-bit" operation

$$T : \mathbb{Z}/2^w \to \mathbb{Z}/2^{w-v}, \quad x \mapsto \lfloor x/2^v \rfloor.$$

For a pseudorandom w-bit integer sequence $x_0, x_1, \ldots,$ we consider the $(w-v)$-bit integer sequence $T(x_1), T(x_2), \ldots$.

THEOREM 3.3. *If a function $x(s) \in \mathbb{Z}/2^w$ of s has an affine dependence* (*Eq.* 4) *modulo 2^w, then $T(x(s))$ has a nearly affine dependence* (*Eq.* 4) *modulo 2^{w-v} with defect at most* 1.

PROOF. By definition of an affine dependence, for some real constants a, c, we have $x(s) = as + c \bmod 2^w$, and hence $x(s) = as + c + k2^w$ for some $k \in \mathbb{Z}$. Since we have $T(x(s)) = \lfloor (as + c + k2^w)/2^v \rfloor$ and

$$(as+c)/2^v + k2^{w-v} - 1 < \lfloor (as+c)/2^v + k2^{w-v} \rfloor \le (as+c)/2^v + k2^{w-v},$$

it follows that

$$|T(x(s)) - (as+c)/2^v|_{2^{w-v}} < 1.$$

This implies that $T(x(s))$ is approximated by an affine function $\frac{a}{2^v}s + \frac{c}{2^v}$ modulo 2^{w-v}, with defect at most 1. $\square$

3.4 Lagged Fibonacci Generators

A lagged Fibonacci generator (LFG) [Knuth 1997] generates a sequence of integers in $\{0, \ldots, M_{\rm LF} - 1\}$ by the recursion of degree p

$$x_{j+p} := x_{j+q} + x_j \mod M_{\rm LF} \quad (j = 0, 1, \ldots). \tag{8}$$

The integers $q < p$ are chosen so that the generated sequence (x_n) has a long period: If $M_{\rm LF}$ is a power of two, then its period is at most $M_{\rm LF}(2^p - 1)/2$, and if $M_{\rm LF}$ is a prime then it is at most ${M_{\rm LF}}^p - 1$ [Knuth 1997, p. 29]. (The plus in the righthand side of (8) may be minus; in that case it is called a subtractive LFG.)

An LFG needs initial values $(x_0, x_1, \ldots, x_{p-1})$ and an initializer, as explained in Section 3.1. In some implementations, such as in the GNU Scientific Library (GSL) [GNU 2004], an LCG is used as the initializer. If the modulus $M_{\rm LCG}$ of the LCG is the same as the modulus $M_{\rm LF}$ of the LFG, then the LFG has persistent affine dependence. This is because $x_i(s)$ $(0 \le i \le p-1)$ are all affine functions of s modulo $M_{\rm LCG}$ (Theorem 3.2), and any other $x_i(s)$ $(i \geq p)$ are linear combinations of the previous functions $x_j(s)$ $(j < i)$ modulo $M_{\rm LF} = M_{\rm LCG}$. Since linear combinations of affine functions are again affine, we have the result.

LEMMA 3.4. *The lagged Fibonacci generator* (*Eq.* 8) *with initialization given by* (*Eq.* 7) *has a persistent affine dependence* (6) *if the modulus $M_{\rm LCG}$ of the initializer is equal to the modulus $M_{\rm LF}$ of the LFG.*

The first implementation of `random` on SunOS 4.1 and FreeBSD (of around 1994) is such an example (this is still the standard on some platforms, including GNU-C in CYGWIN). The chosen parameters are $p = 31$, $q = 28$, and $M_{\rm LCG} = M_{\rm LF} = 2^{32}$, so persistent affine dependences are observed.[1]

The following improvements were adopted in the initialization of `random` in the mid 1990's.

—If $s = 0$, then set $s = 1$.
—Discard the least significant bit of the output by shift-right.
—After initialization, the first 310 outputs are discarded.

However, this does not remove the affine dependence (Theorem 3.3).

3.5 Lagged Fibonacci Generators with a Hetero-Modulus Initializer

The most recent version `random97` explored in Section 2 adopted the modulus $M_{\rm LCG} = 2^{31} - 1$ for the initialization LCG of Eq. (7), different from $M_{\rm LF} = 2^{32}$. In spite of this, we still observe a transient nearly affine dependence, as seen in Figure 1 for $n = 10, 14$. Such an obvious dependence fades away for $n > 40$, but statistical tests reveal some residual correlations for $n < 83$.

The nearly affine dependence of `random97` survives through the aforementioned improvements because of the following two unfortunate factors (see the analysis to follow):

—The chosen modulus $M_{\rm LCG} = 2^{31} - 1$ of the initializing LCG is too similar to the modulus $M_{\rm LF} = 2^{32}$ of the LFG.
—The largest eigenvalue of the recursion (8) is too close to 1.

Let G be a lagged Fibonacci generator (8), with initializer LCG (7). To analyze G, we introduce an auxiliary generator G', which is obtained from G by replacing the modulus $M_{\rm LF}$ of the generating recursion (8) with $M_{\rm LCG}$. Thus, the initialization and generation of G' share a common modulus $M = M_{\rm LCG}$ and, as seen in Section 3.4, its output $x'_n(s)$ has a persistent affine dependence modulo $M_{\rm LCG}$. We show that the output $x_n(s)$ of G is often approximated by $x'_n(s)$ with a small error term.

We define the error term by

$$e_n(s) := x_n(s) - x'_n(s) \mod M_{\rm LF}, \tag{9}$$

[1]The period of this version of `random` is about 2^{63}, with $M_{\rm LF} = 2^{32}$, $p = 31$, and $q = 28$. Note that the description of `random` in the Free BSD manual `man` seems to contain mistakes: This says "non-linear additive generator," but it is actually linear, and "period is $16(2^{31} - 1)$," but it is $2^{32}(2^{31} - 1)$.

and claim that $e_n(s)$ takes a small value if n is not large and if

$$d := |M_{\mathrm{LF}} - M_{\mathrm{LCG}}| \tag{10}$$

is small.

Since G and G' share the same initializing LCG, we have

$$e_0(s) = e_1(s) = \cdots = e_{p-1}(s) = 0 \quad \text{for all } s. \tag{11}$$

We shall prove that

$$|e_{p+j}(s)|_{M_{\mathrm{LF}}} \leq |e_{q+j}(s)|_{M_{\mathrm{LF}}} + |e_j(s)|_{M_{\mathrm{LF}}} + d \text{ for all } s \text{ and } j. \tag{12}$$

Since

$$x_{p+j}(s) = x_{q+j}(s) + x_j(s) \mod M_{\mathrm{LF}}$$

and

$$x'_{p+j}(s) = x'_{q+j}(s) + x'_j(s) \quad \text{or} \quad x'_{p+j}(s) = x'_{q+j}(s) + x'_j(s) - M_{\mathrm{LCG}}$$

hold, we have

$$e_{p+j}(s) = e_{q+j}(s) + e_j(s) + D_j(s) \mod M_{\mathrm{LF}}, \tag{13}$$

where $D_j(s) = 0$ or $\pm d$ (if $x'_{q+j}(s) + x'_j(s) < M_{\mathrm{LCG}}$ then $D_j(s) = 0$, otherwise $+d$ or $-d$, according to whether $M_{\mathrm{LF}} > M_{\mathrm{LCG}}$), and hence

$$|e_{p+j}(s)|_{M_{\mathrm{LF}}} \leq |e_{q+j}(s)|_{M_{\mathrm{LF}}} + |e_j(s)|_{M_{\mathrm{LF}}} + d. \tag{14}$$

The following lemma immediately follows.

LEMMA 3.5. *Let E_n be the integer sequence defined by the linear recursion*

$$E_{p+j} = E_{q+j} + E_j + d \tag{15}$$

with initial values $E_0 = E_1 = \cdots = E_{p-1} = 0$. Then,

$$|e_n(s)|_{M_{\mathrm{LF}}} \leq E_n$$

for all $n = 0, 1, \ldots$ and $s \in \mathbb{Z}$.

LEMMA 3.6. *One has*

$$E_n = \left(\frac{d}{(\alpha - 1) f'(\alpha)} + o(1) \right) \alpha^n$$

for $n \geq p$, where α is the unique positive real root of the characteristic polynomial

$$f(t) = t^p - t^q - 1 \tag{16}$$

and $f'(t)$ is the derivative function of $f(t)$.

PROOF. Since (15) is equivalent to the linear recursion $(E_{p+j} + d) = (E_{q+j} + d) + (E_j + d)$, we have

$$E_n + d = \sum_{i=0}^{p-1} c_i \alpha_i^n, \tag{17}$$

where $\alpha_0, \ldots, \alpha_{p-1}$ are the roots of the characteristic polynomial (16). Let $\alpha := \alpha_0$ be the root with maximum absolute value. The Perron-Frobenius theorem say that α_0 is a positive real root with multiplicity one. By dividing (17) by α^n and by taking the limit, we have

$$E_n/\alpha^n \to c_0 \quad (n \to \infty).$$

Now c_0 is obtained by solving the linear equation (17) for $n = 0, \ldots, p-1$. Cramer's formula states that c_0 is d times the van der Monde determinant of $(1, \alpha_1, \ldots, \alpha_{p-1})$ divided by that of $(\alpha_0, \alpha_1, \ldots, \alpha_{p-1})$, thus

$$c_0 = d \cdot \frac{f(1)}{1-\alpha} \cdot \frac{1}{f'(\alpha)} = \frac{d}{(\alpha-1)f'(\alpha)}. \qquad \square$$

THEOREM 3.7. *A lagged Fibonacci generator* (8) *with initialization given by an LCG* (7) *has a nearly affine dependence* (5) *at the nth output with defect at most*

$$|e_n(s)|_{M_{\mathrm{LF}}} \le \left(\frac{|M_{\mathrm{LCG}} - M_{\mathrm{LF}}|}{(\alpha-1)f'(\alpha)} + o(1) \right) \alpha^n,$$

where α is the unique positive real root of the characteristic polynomial $f(t) = t^p - t^q - 1$ and $f'(t)$ is the derivative of $f(t)$.

PROOF. This follows from the approximation in Lemma 3.6, the inequality in Lemma 3.5, and the fact that $d = |M_{\mathrm{LF}} - M_{\mathrm{LCG}}|$. $\square$

By the comment following Eq. (13), either one of $0 \le e_n(s) < E_n$ (if $M_{\mathrm{LF}} > M_{\mathrm{LCG}}$) or $0 \ge e_n(s) > -E_n$ (if $M_{\mathrm{LF}} < M_{\mathrm{LCG}}$) holds. Thus (5) can be rewritten as

$$x_n(s) = a_n s + (c_n + E_n/2) + e'_n(s) \mod M_{\mathrm{LF}}, \tag{18}$$

where $|e'_n(s)|_{M_{\mathrm{LF}}} < E_n/2$.

Example 3.8. Recall that `random97` has $M_{\mathrm{LCG}} = 2^{31} - 1$ and $M_{\mathrm{LF}} = 2^{32}$. If we remove the most significant bit of each output value, then it amounts to setting $M_{\mathrm{LF}} = 2^{31}$. Now $|M_{\mathrm{LCG}} - M_{\mathrm{LF}}| = 1$, $\alpha \approx 1.06054$ for $p = 31$ and $q = 28$, and Theorem 3.7 implies $E_n \sim 0.376259(1.06054)^n$. By computation, it follows that $E_n < 2^{31}$ (consequently $|e'_n(s)|_{M_{\mathrm{LF}}} < 2^{30}$) holds for $n < 383$, and thus the lower 31 bits of $x_n(s)$ have the nearly affine dependence with defect ratio less than 1/2 for $n < 383$. Since the first 341 outputs are discarded for a warm-up, the nth output of `random97` is x_{341+n} $(n = 0, 1, 2, \ldots)$, and the nearly affine dependence is proved detectable for at least the first $382 - 341 = 41$ outputs.

One may obtain a more precise estimation as follows (the reader not interested in such details may skip the rest of this section). We define a sequence of nonnegative integers by

$$F_{n+p} = F_{n+q} + F_n, \quad F_0 = 1, F_1 = \cdots = F_{p-1} = 0,$$

then it follows easily by induction that for $n \ge p$

$$E_n = d \sum_{j=0}^{n-p} F_j, \tag{19}$$

and hence

$$F_{n-p} = E_n - E_{n-1} \simeq \frac{d}{f'(\alpha)} \alpha^n. \tag{20}$$

We assume $M_{\rm LF} > M_{\rm LCG}$ (the converse case can be treated in the same manner). Then, in the recursion (13), $D_j(s) = 0$ if $x'_{q+j}(s) + x'_j(s) < M_{\rm LCG}$, and $D_j(s) = d$ otherwise. By the linearity, we have

$$e_n(s) = \sum_{j=0}^{n-p} F_{n-j} D_j(s). \tag{21}$$

Our approximative assumption is that $D_j(s)$ behaves as a random variable taking the value 0 or d with probability 1/2 for each. This is justified by the fact that the $x'_j(s)$ should behave as uniform random variables. Thus, from (21) we have the expectation

$$E(e_n(s)) = \frac{1}{2} E_n. \tag{22}$$

In addition, for $j \le j'$, we assume that $D_j(s)$ and $D_{j'}(s)$ are independent, unless $j' = j$, $j' = j + q$, $j' = j + p$, or $j' = j + p - q$. In these four cases, the correlations are approximated by

$$E(D_j(s)D_{j'}(s)) - E(D_j(s))E(D_{j'}(s)) \simeq \begin{cases} d^2/4 & (j' = j), \\ d^2/12 & (j' = j + q), \\ -d^2/12 & (j' = j + p), \\ -d^2/12 & (j' = j + p - q). \end{cases}$$

These correlations come from the following. Let $0 \le x, y, z < 1$ be uniformly and independently distributed random variables. Then, $\mathrm{Prob}(x + y > 1) = \mathrm{Prob}(y + z > 1) = 1/2$, and $\mathrm{Prob}(x + y > 1 \text{ and } y + z > 1) = 1/3$ (by the volume computation). Thus, the correlation between the indicator functions of the two events $x + y > 1$ and $y + z > 1$ is $1/3 - 1/2 \cdot 1/2 = 1/12$. This means that the correlation between $D_{j+q}(s)$ and $D_j(s)$ is $d^2/12$. Other cases follow from similar computations. The variance of (21) is now approximated by

$$\begin{aligned} V(e_n(s)) \approx\ & \frac{1}{4} d^2 \sum_{j=0}^{n-p} F_{n-j}^2 \\ & + 2\frac{1}{12} d^2 \left(\sum_{j=0}^{n-p-q} F_{n-j}F_{n-j-q} - \sum_{j=0}^{n-2p} F_{n-j}F_{n-j-p} \right. \\ & \left. - \sum_{j=0}^{n-2p+q} F_{n-j}F_{n-j-p+q} \right). \end{aligned}$$

By substituting F_n using (20) and taking the square root, we obtain an approximation of the standard deviation of e_n.

$$\sigma_{e_n} \simeq \frac{E_n}{4} \sqrt{\frac{\alpha - 1}{\alpha + 1} \left(1 - \frac{2}{3} \left(\frac{1}{\alpha^{p-q}} - \frac{1}{\alpha^q} + \frac{1}{\alpha^p} \right) \right)}$$

Example 3.9. For `random97`, the preceding approximation leads to $\sigma_{e_n} \sim 0.03E_n$. This implies that the correlation is observable for $n \le 341 + 78$, where $2\sigma_{e_n} \le 2^{31}$. Figure 2 shows the distribution of $e_{341+n}(s)/2$ (we divide by 2 because the output of `random97` is shifted to the right, and the subscript is $341 + n$

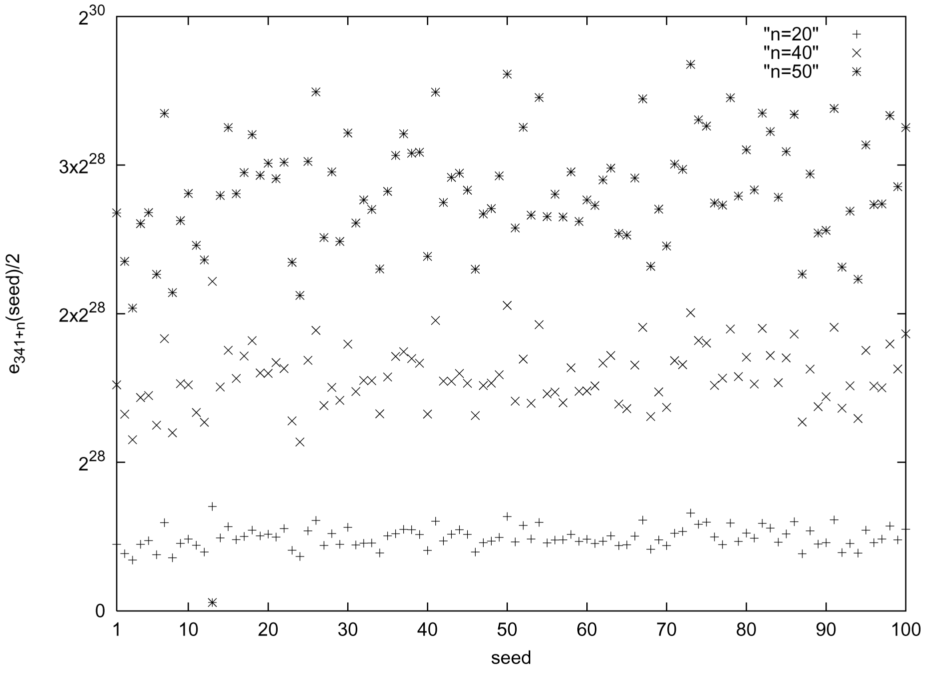

Fig. 2. The distribution of the error terms $e_{341+n}(s)/2$ for the lower 30 bits of the nth output of `random97`, for $s = 1, 2, \ldots, 100$, $n = 20, 40, 50$.

because the first 341 outputs are discarded). This distribution agrees with the previous analysis. Note that in this case $E_{341+20} = 6.130 \times 10^8$ ($\sim 1.14 \times 2^{29}$), $E_{341+40} = 2.018 \times 10^9$ ($\sim 1.88 \times 2^{30}$), and $E_{341+50} = 3.594 \times 10^9$ ($\sim 1.67 \times 2^{31}$). For example, the $(341+40)$-th error term $e_{341+40}/2$ has the average value 4.24×10^8 over $s = 1, 2, \ldots, 100$, and the expected error $E_{341+40}/4$ is 5.05×10^8.

3.6 Analysis of the ranlux Generator

The generator `ranlux` [Lüscher 1994] is based on a subtract-with-borrow generator (SWB). It generates a sequence of 24-bit integers by a recursion

$$x_{j+24} := x_{j+14} - x_j + c_{j+24} \mod M_{\mathrm{LF}} \quad (j = 0, 1, \ldots), \tag{23}$$

with $M_{\mathrm{LF}} = 2^{24}$ and some initial values $x_0, \ldots, x_{23}$ of 24-bit integers. The term c_{j+24} is called a carry (or borrow). It is set to 1 or 0, according to whether the previous step required the borrow, that is, according to whether $x_{j+13} - x_{j-1} + c_{j+23}$ is negative. In a standard initialization due to James [1994], the initializer is an LCG with modulus $M_{\mathrm{LCG}} = 2^{31} - 85$.

If we neglect the term c_{j+24}, we have an LFG. The purpose of this term is to lengthen the period, and this has little effect on breaking the nearly affine dependence: Clearly, (13) holds for $D_j(s) = 0, 1, \pm d$ or $1 \pm d$, where $d = |M_{\mathrm{LCG}} - M_{\mathrm{LF}}|$. Thus, the error estimation for SWB is almost the same as that of LFG, except that we need to replace d with $d + 1$ in (14) and the eigenvalue α is a complex number. The almost-equivalence between the SWB and LCG was first observed and studied by Tezuka et al. [1994].

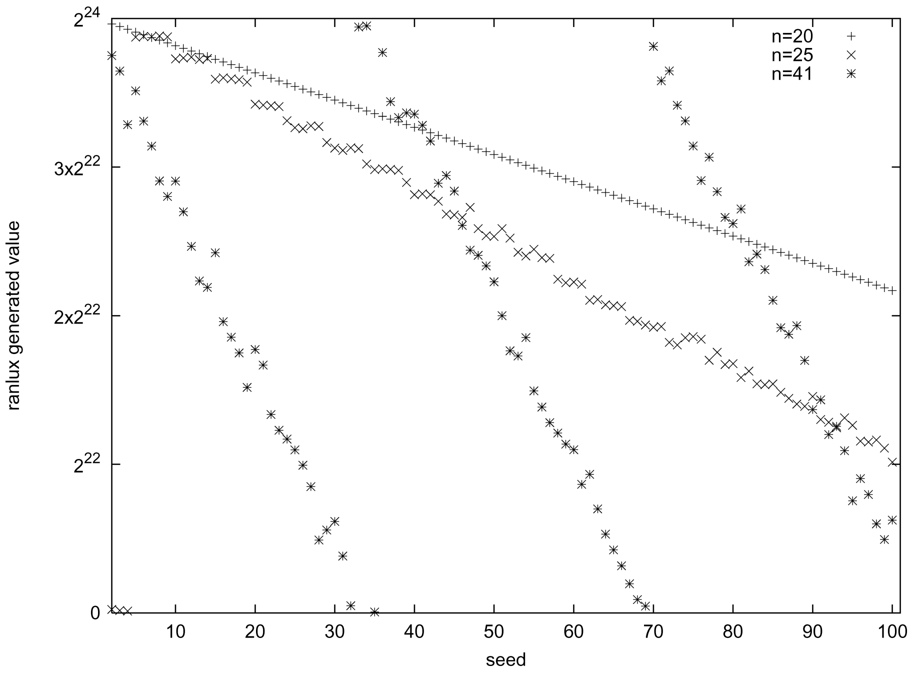

Fig. 3. The nth ($n = 20, 25, 41$) output $x_n(s)$ of `ranlux` for seeds $s = 1, 2, \ldots, 100$.

The generator `ranlux` is obtained by simply discarding a large part of the outputs of an SWB: `ranlux` generates 24 words by SWB and outputs them, and then discards the next r words of the outputs of SWB. Then it outputs the next 24 outputs of SWB again, and discards the next r. It iterates this, and so on.

In the standard implementation by James [1994], r is 199 by default, and the largest value that the user can choose is 365. These values are claimed to assure no correlation, using a discussion based on chaos theory [Lüscher 1994]. However, Figure 3 shows obvious correlations even after skipping 199 words. This implies that Lüscher [1994] misused chaos theory: He neglected the nearly affine property of the transition function of SWB. Actually, it turns out that for every fixed $n < 48$, the nth output of the default `ranlux` is a nearly affine function of s with defect ratio $< 1/2$. Thus, similar phenomena are observed for such n whenever s changes linearly. This dependence fades away for $n \geq 48$, so `ranlux` has a transient nearly affine dependence.

Another type of deviation of `ranlux` is reported by Matsumoto and Nishimura [2003].

4. SYMPTOM 2: DIFFERENCE COLLISIONS

4.1 Difference Collisions

The nearly affine dependence can be rephrased by using the difference equation

$$x_n(s+1) - x_n(s) = a_n + \text{small error}$$

Table II. Difference $x_n(s+1) - x_n(s) \bmod 2^{31} - 1$ Between the nth Output of `mrg` with Seed $s+1$ and That with Seed s (in hexadecimal notations: for example, `0xff` denotes 255)

n	s = 1	s = 6	s = 3	s = 10
$x_0(s+1) - x_0(s)$	0x0ef1bc75	0x0ef1bc75	0x5e240b4b	0x5e240b4b
$x_1(s+1) - x_1(s)$	0x1e145efa	0x1e145efa	0x095a16e1	0x095a16e1
$x_2(s+1) - x_2(s)$	0x569876c8	0x569876c8	0x0f3e4c65	0x0f3e4c65
$x_3(s+1) - x_3(s)$	0x1f7dfaba	0x1f7dfaba	0x7d41feab	0x7d41feab
$x_4(s+1) - x_4(s)$	0x129d6b03	0x129d6b03	0x06ec1ff8	0x06ec1ff8
$x_5(s+1) - x_5(s)$	0x7c0e50d1	0x7c0e50d1	0x1a7c458d	0x1a7c458d
$x_{10}(s+1) - x_{10}(s)$	0x6dffd0a3	0x6dffd0a3	0x53d428ef	0x53d428ef
$x_{100}(s+1) - x_{100}(s)$	0x56f28400	0x56f28400	0x1628f237	0x1628f237
$x_{1000}(s+1) - x_{1000}(s)$	0x27a58ffe	0x27a58ffe	0x71cf9f1f	0x71cf9f1f

The differences coincide for $s = 1$ and $s = 6$, as well as for $s = 3$ and $s = 10$.

for all s, or equivalently

$$x_n(s+1) - x_n(s) = x_n(t+1) - x_n(t) + \text{small error}$$

for any pair of seeds (s, t). The following is a weaker type of dependence.

Definition 4.1. A PRNG generating integers in the interval $[0, M-1]$ has a *difference collision at the seeds* s, t with defect (at most) δ if

$$|(x_n(s+1) - x_n(s)) - (x_n(t+1) - x_n(t))|_M \leq \delta$$

for all n.

4.2 Examples of Difference Collisions

In GSL, there are more advanced generators, such as multiple recursive generators (MRGs) [L'Ecuyer 1993] and combined multiple recursive generators (CMRGs) [L'Ecuyer 1996] (we postpone their descriptions to Section 4.4), which are optimized with respect to high-dimensional distributions and passed stringent tests. However, they have difference collisions because of the carelessly chosen initializers in these implementations. (Actually, one should use the optimized initializers for the multiple stream generations studied in L'Ecuyer et al. [2002] and L'Ecuyer and Andres [1997] for which such dependences should not be observed.)

Table II shows that we have difference collisions with $\delta = 0$ at the seeds 1, 6 (left) and 3, 10 (right) of an MRG named `mrg` in GSL, with modulus $M' = 2^{31} - 1$. Such collisions can be observed rather densely among pairs of seeds. Actually, for 2^{32} possible seeds, only 32 possible patterns of the difference exist; see Section 4.4.

Table III shows the difference collision of a CMRG named `cmrg` in GSL. The symptom is similar to that of `mrg`, except for a few irregularities.

4.3 Analysis of Difference Collisions

In the rest of this section we analyze the difference collision: The nth output of a PRNG (generating integers between 0 and $M-1$) has a difference collision at initial seeds s, t if

$$x_n(s+1) - x_n(s) = x_n(t+1) - x_n(t) \mod M. \tag{24}$$

Table III. A Similar Table to Table II Obtained from `cmrg`

n	s = 1	s = 6	s = 3	s = 10
$x_0(s+1) - x_0(s)$	0x54220df5	0x54220df5	0xf0835f95	0xf0835f95
$x_1(s+1) - x_1(s)$	0x0cec35ad	0x0d0abad5	0x29c28e2f	0x29c28e2f
$x_2(s+1) - x_2(s)$	0x7cc9c4e9	0x7cab3fc1	0x79116af0	0x79116af0
$x_3(s+1) - x_3(s)$	0x7c1059f7	0x7c1059f7	0x0f48a4bb	0x0f6729e3
$x_4(s+1) - x_4(s)$	0x25319211	0x25319211	0x359ee410	0x359ee410
$x_5(s+1) - x_5(s)$	0x4fe67406	0x4fe67406	0x6d03daf2	0x6d03daf2
$x_{10}(s+1) - x_{10}(s)$	0x5021c770	0x5021c770	0x5c130ffb	0x5c130ffb
$x_{100}(s+1) - x_{100}(s)$	0x7e079cd8	0x7e079cd8	0x4a0118e5	0x4a1f9e0d
$x_{1000}(s+1) - x_{1000}(s)$	0x1ce7d959	0x1ce7d959	0x2afb70eb	0x2afb70eb

The differences for $s = 1$ and $s = 6$ coincide for $n = 0, 3, 4, 5, 10, 100$, and 1000 in this table. For $n = 1, 2$, they do not coincide but are close. Similar phenomena are observed for $s = 3$ and $s = 10$.

The probability of such a collision should be $1/M$. We shall see examples where: (a) For densely distributed pairs (s, t), the collision (24) holds for all n (*everlasting difference collision* at s and t, e.g., Table II); and (b) for sparsely distributed pairs (s, t), (24) holds for all n (*sparse difference collision*, e.g., see Section 4.5), and their weak versions (i.e., Eq. (24) holds with a small error term, e.g., Table III) and transient versions (i.e., the error term is increasing for increasing n, e.g., see Section 4.5).

4.4 Dense Difference Collisions

A multiple recursive generator (MRG) [L'Ecuyer 1993] is based on a pth-order linear recursion

$$x_{j+p} := a_{p-1}x_{j+p-1} + \cdots + a_1 x_{j+1} + a_0 x_j \mod M_{\mathrm{MRG}} \quad (j = 0, 1, \ldots), \qquad (25)$$

for a fixed modulus M_{MRG} and suitable constants $a_{p-1}, \ldots, a_0$. Thus, this class includes lagged Fibonacci generators (8). A combined multiple recursive generator (CMRG) [L'Ecuyer 1996] combines two distinct MRGs by computing a linear combination of the two output streams.

Consider an MRG (25) with the initializer an LCG (7). If $M_{\mathrm{MRG}} = M_{\mathrm{LCG}}$, then the MRG has the affine dependence, which we have already analyzed. Assume that $M_{\mathrm{MRG}} \neq M_{\mathrm{LCG}}$. Differently from the case of LFGs, the nearly affine dependence is not observed even for very small $|M_{\mathrm{MRG}} - M_{\mathrm{LCG}}| > 0$. This is because the characteristic polynomial of the recursion (25) has a root of large absolute value in general, and consequently, the error in the affine approximation increases rapidly (see the discussion in Section 3.5).

However, if p is small, we observe difference collisions.

THEOREM 4.2. *Suppose that an MRG* (25) *of order p is initialized by a generator with affine dependence* (6) *with seed s. Then, we have*

$$\#\{(x_n(s+1) - x_n(s) \bmod M_{\mathrm{MRG}})_{n=0,1,2,\ldots} \mid s \text{ goes through all possible values}\} \leq 2^p.$$

This implies that the set of seeds is classified into 2^p categories, so that if s and t belong to one and the same category, then $x_n(s+1) - x_n(s) = x_n(t+1) - x_n(t) \bmod M_{\mathrm{MRG}}$ holds for all n.

Proof. Let us denote the first p outputs of the initializer by

$$X_0(s), \ldots, X_{p-1}(s).$$

By definition of affine dependence, we have

$$X_i(s) = a_i s + c_i \bmod M \quad (i = 0, 1, \ldots, p-1)$$

for some M and a_i with $0 \le a_i < M$. This implies that

$$X_i(s+1) - X_i(s) = a_i \text{ or } a_i - M, \tag{26}$$

depending on s.

Let us denote the output of the MRG (25) initialized with seed s by $x_0(s), x_1(s), \ldots$. It follows that

$$x_i(s) := X_i(s) \mod M_{\text{MRG}} \quad (i = 0, 1, \ldots, p-1). \tag{27}$$

Now the sequence of differences

$$x_i(s+1) - x_i(s) \quad (i = 0, 1, 2, \ldots)$$

satisfies the linear recursion (25), and its initial values are

$$(x_0(s+1) - x_0(s), x_1(s+1) - x_1(s), \ldots, x_{p-1}(s+1) - x_{p-1}(s)). \tag{28}$$

By Eqs. (27) and (26), there are at most 2^p values for (28). Consequently, there are at most 2^p different sequences $(x_n(s+1) - x_n(s))_{n=0,1,2\ldots}$. □

Example 4.3. GSL has an MRG named `mrg` with $p = 5$, and its initializer is an LCG. Thus, the set of seeds is classified into $2^5 = 32$ categories. Computations show $s = 1$ and $s = 6$ belong to one category, and $s = 3$ and $s = 10$ to another. Table II shows the difference collision at the seeds 1, 6, and 3, 10.

We now consider a special case of CMRG as follows. One component MRG has order p and modulus M_{MRG}, and the other component MRG has order p' and modulus $M_{\text{MRG}}{}'$. Assume that both MRGs are initialized by an initializer with affine dependence.

Let $x_n(s)$ be the nth output from the seed s of the former MRG, and $x'_n(s)$ be the latter. The output of the CMRG is given by

$$y_n(s) := x_n(s) - x'_n(s) \bmod M_{\text{MRG}}{}'. \tag{29}$$

Theorem 4.4. *Consider the CMRG given in Eq.* (29). *If two seeds s and t belong to the same category with respect to both MRGs in the sense of Theorem* 4.2*, then*

$$|(y_n(s+1) - y_n(s)) - (y_n(t+1) - y_n(t))|_{M_{\text{MRG}'}} = 0 \textit{ or } |M_{\text{MRG}}|_{M_{\text{MRG}'}}.$$

Proof. By the assumption,

$$x_n(s+1) - x_n(s) = x_n(t+1) - x_n(t) \bmod M_{\text{MRG}} \text{ and}$$
$$x'_n(s+1) - x'_n(s) = x'_n(t+1) - x'_n(t) \bmod M_{\text{MRG}}{}'.$$

By considering the range of $x_n(s)$, we have

$$(x_n(s+1) - x_n(s)) - (x_n(t+1) - x_n(t)) = 0, \pm M_{\text{MRG}}.$$

A straightforward elimination gives

$$\begin{aligned}&(y_n(s+1)-y_n(s))-(y_n(t+1)-y_n(t))\\ =\ &(x_n(s+1)-x_n(s))-(x_n(t+1)-x_n(t)) \bmod M_{\mathrm{MRG}}{}',\end{aligned}$$

and hence the conclusion of the theorem follows. □

Example 4.5. The CMRG generator `cmrg` in GSL is as follows: One component MRG has parameters $p=3$ and $M_{\mathrm{MRG}}=2^{31}-1$, and the other component MRG has $p=3$ and $M_{\mathrm{MRG}}{}'=2^{31}-2000169$. Both MRGs share the same LCG as `mrg`, and hence $s=1$ and $s=6$ belong to the same category, for example. If two seeds s,t belong to the same category, then $|y_n(s+1)-y_n(s)|_{M_{\mathrm{MRG}'}}$ and $|y_n(t+1)-y_n(t)|_{M_{\mathrm{MRG}'}}$ coincide or differ by $|M_{\mathrm{MRG}}|_{M_{\mathrm{MRG}'}}=2000168=$ `0x1e8528`. Table III shows this phenomenon. For example, for $n=1,2$, the difference for $s=1$ does not coincide with that for $s=6$, but the values differ by `0x1e8528`.

Remark 4.6. Let c be a positive integer. The definition of difference collisions (24) can be generalized to

$$x_n(s+c)-x_n(s)=x_n(t+c)-x_n(t) \quad \bmod M_{\mathrm{MRG}}.$$

In this generalization, Theorems 4.2 and 4.4 still hold if we replace $s+1$ with $s+c$ and $t+1$ with $t+c$ in the statements. Their proofs are analogous to the preceding proofs, and we omit them here.

4.5 Sparse Difference Collision

In the case of `random97` ($p=31$) or `ranlux` ($p=24$), 2^p appears to be large enough to avoid such collisions, but actually is not. We shall compute the probability that (24) holds for $n=0,\dots,p-1$ under a random choice of s,t.

Assume that $X_n(s)$ has an affine dependence

$$X_n(s)=a_ns+c_n \quad \bmod M \quad (-M/2<a_n\le M/2).$$

In the case of LCG, $a_n=a^n \quad \bmod M$. Since the LCG is a fairly good PRNG, we may consider $|a_n|$ as a random variable uniformly distributed in the interval $[1,M/2]$.

We have two possibilities, depending on s:

$$X_n(s+1)-X_n(s)=\begin{cases}a_n & \text{(A)}\\ a_n\mp M & \text{(B)},\end{cases}$$

where the sign $\mp$ is the converse to that of a_n. Under the uniformly random choice of s, (A) occurs with probability $(M-|a_n|)/M$ and (B) with probability$|a_n|/M$. Thus, under the random choice of s and t, the probability that (24) occurs for this n is at least

$$P_n:=((M-|a_n|)/M)^2+(|a_n|/M)^2.$$

The collision probability for a random choice of s,t is the average of the product $E(P_1\cdots P_p)$ over $a_1,\dots,a_p$ varying. Here $E(P_1\cdots P_p)$ is the probability that

(24) holds for all $n = 1, \ldots, p$. Under the assumption that $a_1, \ldots, a_p$ are independent, $E(P_1 \cdots P_p) = E(P_1) \cdots E(P_p)$ holds, and we have

$$E(P_i) = 2\int_0^{1/2} (1-x)^2 + x^2 dx = 2[x - x^2 + 2/3x^3]_0^{1/2} = 2(1/2 - 1/4 + 1/12) = 2/3.$$

Thus, the collision probability is approximated by $(2/3)^p$. This value is 3.477×10^{-6} for `random97` ($p = 31$) and 5.940×10^{-5} for `ranlux` ($p = 24$).

For fixed s ($s = 1, 2, 3, 4, 5$), we searched for t satisfying (24) for all n, with error term ± 1 for `random97` (± 1 introduced because `random97` discards the least significant bit). The following list shows the values of t in the ascending order.

$$\begin{aligned} s &= 1, t = 6441, 48467, 55121, \ldots \\ s &= 2, t = 176077, 183610, 369789, \ldots \\ s &= 3, t = 255319, 940971, 1054113, \ldots \\ s &= 4, t = 671577, 1031941, 1040006, \ldots \\ s &= 5, t = 80166, 349265, 451966, \ldots \\ &\ldots \end{aligned}$$

Similarly, `ranlux` has many pairs (s, t) with difference collision (24) for $1 \le n \le 24$ with small error terms. However, the errors introduced by the carry in (23) are accumulated as n grows (see Section 3.6), and the error in (24) increases. The speed of the decay depends on s, but experiments show that the initial seeds $s = 1, 65, 1714$ are correlated up to $n = 95$, that is, the three values $x_n(2) - x_n(1)$, $x_n(66) - x_n(65)$, $x_n(1715) - x_n(1714)$ are very close for $n < 96$ (here $x_n(s)$ means the nth outputs of `random97` with the default discarding). Moreover, we have $x_n(2) - x_n(1) = x_n(77044) - x_n(77043)$ for almost all n (we checked this by computation for $n < 10^9$. This implies that the "highest luxury level", namely, discarding 365 words, does not suffice. This again makes the use of chaos theory in Lüscher [1994] doubtful.

5. RESULTS OF EXTENSIVE EXPERIMENTS

We have investigated 58 generators available in the GNU Scientific Library [GNU 2004] regarding the defects aforementioned. Table IV shows the result. Among the generators, 40 showed some kind of defect. Some of them have been declared obsolete, but some are still widely used.

There are 18 generators which passed both tests. Among them, there is one LFG (i.e., `random256-glibc2`) with initializer an LCG, which is essentially `random97` but the first 630 values are discarded. `random256-glibc2` also has difference collisions, but the collisions seem sufficiently sparse because of its huge state space.

Other generators do not have affine dependence because of the nonlinearity introduced in the recurrence or initialization. Note that some of these 18 are known to be defective for other reasons.

We mention three generators showing behaviors other than those mentioned earlier: `coveyou` generates the same sequence for the seed $s = 4i, 4i + 2, 4i + 3$ for any integer i. The dependence of tt800 is observed only on the first three

Table IV. Seed Dependence of 58 GSL Generators

Generator	Affine	Collision	Generator	Affine	Collision
borosh13	×	×	cmrg	◯	×
coveyou	◯	×	fishman18	×	×
fishman20	×	×	fishman2x	×	×
gfsr4	◯	◯	knuthran	◯	◯
knuthran2	×	×	lecuyer21	×	×
minstd	×	×	mrg	◯	×
mt19937	◯	◯	mt19937_1999	◯	◯
mt19937_1998	◯	◯	r250	△	◯
ran0	×	×	ran1	◯	◯
ran2	◯	◯	ran3	×	×
rand	×	×	rand48	×	×
random8-glibc2	×	×	random32-glibc2	△	△
random64-glibc2	△	△	random128-glibc2	△	△
random256-glibc2	◯	◯	random8-libc5	×	×
random32-libc5	×	×	random64-libc5	×	×
random128-libc5	×	×	random256-libc5	×	×
random8-bsd	×	×	random32-bsd	×	×
random64-bsd	×	×	random128-bsd	×	×
random256-bsd	×	×	randu	×	×
ranf	×	×	ranlux	△	△
ranlux389	△	△	ranlxd1	◯	◯
ranlxd2	◯	◯	ranlxs0	◯	◯
ranlxs1	◯	◯	ranlxs2	◯	◯
ranmar	◯	◯	slatec	×	×
taus	◯	◯	taus2	◯	◯
taus113	◯	◯	transputer	×	×
tt800	△	◯	uni	×	×
uni32	△	×	vax	×	×
waterman14	×	×	zuf	◯	◯

Column affine:
×:persistent nearly affine dependence observed.
△:transient nearly affine dependence observed.
◯:affine dependence not observed.
Column collision:
×:dense difference collision observed.
△:sparse difference collision observed.
◯:difference collision not observed.
(We do not mean ◯ is recommendable.)

outputs. The state array of tt800 is initialized by an LCG, and the first three outputs are simple transformations of the output of the LCG. The initialization of uni32 neglects the least significant bit of the seed.

6. IMPROVED INITIALIZATION

A simple solution to these defects is to replace the initializer LCG with some nonlinear one. For example, the 2002 version of the initializer of Mersenne Twister (at the URL shown in Matsumoto and Nishimura [1998]) adopts the recursion

$$x_{j+1} := a(x_j \oplus \text{shift-right}(x_j, s)) + j \mod M, \tag{30}$$

where $M = 2^{32}$, $a = 1812433253$, $s = 30$, $\oplus$ denotes the bitwise exclusive-or, and shift-right means the 32-bit integer x_j is shifted to the right by s (=30) bits. This transformation is invertible. According to experiments, this seems to solve these problems, although it is highly heuristic. Note that this recursion is only for the initialization: We do not know its period nor its distribution property. This initialization gives only at most 2^{32} distinct streams of random numbers. The URL given in Matsumoto and Nishimura [1998] proposes an initializer whose seed is an array of integers of arbitrary length.

7. RELATED WORK

The danger of interstream correlation has often been pointed out (e.g., Mascagni and Srinivasan [2004]) in the context of possible erroneous results in iterative or parallel simulations.

A known cause of correlation is overlapping: In a generator of relatively small state space, two initial states may produce two overlapping sequences after some generations. This is usually detected after long and careful statistical tests.

The phenomena we report here are of a different nature: They are caused by resonance between the generating recursion and the initialization scheme. This is observed in huge-state-space generators such as `ranlux` and `cmrg`, where the probability of overlapping is negligible. These phenomena destroy the simulation from the beginning, not after substantial use of the PRNG.

In a parallelized situation, a possible solution is *parameterizing* [Mascagni and Srinivasan 2000, 2004], which uses different parameters instead of different seeds. Dynamic Creator [Matsumoto and Nishimura 2000] is of this type. However, this is costly in a single-processor system. Another possibility is to apply a theoretical test on the multiple streams of random numbers to select an optimized initialization [L'Ecuyer et al. 2002; L'Ecuyer and Andres 1997].

The fact that `random97` and many other generators have a warm-up in their initialization indicates that the programmers of these generators are aware of some kind of dependences on the seed. Nevertheless, the implemented warm-up is sometimes invalid (e.g., the old BSD `random`) or insufficient to remove the nearly affine dependence (e.g., the present GNU `random`), seemingly because of the lack of mathematical analysis.

We remark that these dependences are well studied in the area of cryptography; see the linear cryptanalysis of Matsui [1994] and the differential cryptanalysis of Biham and Shamir [1991].

8. CONCLUSION

We demonstrated that many modern random number generators have some manifest patterns when initial seeds are chosen via a linear recurrence. These phenomena can be avoided by a nonsystematic choice of seeds, or a more careful initialization scheme such as those in L'Ecuyer and Andres [1997] and L'Ecuyer et al. [2002].

The observations so far suggest that the designer of a PRNG should pay more attention to the initialization. When one proposes a practical PRNG, one should specify an initialization scheme, and test the randomness of the two-dimensional array of outputs $x_n(s)$ with n, s both varying.

REFERENCES

BIHAM, E. AND SHAMIR, A. 1991. Differential cryptanalysis of DES-like cryptosystems. *J. Cryptol. 4*, 1, 3–72.

GNU. 2004. GNU scientific library version 1.6. http://www.gnu.org/software/gsl/.

JAMES, F. 1994. Ranlux: A fortran implementation of the high-quality pseudorandom number generator of Lüscher. *Comput. Phys. Commun. 79*, 1 (Feb.), 111–114.

KNUTH, D. E. 1997. *The Art of Computer Programming, vol. 2. Seminumerical Algorithms*, 3rd ed. Addison-Wesley, Reading, MA.

L'ECUYER, P. 1996. Combined multiple recursive random number generators. *Oper. Res. 44*, 5, 816–822.

L'ECUYER, P. 1992. Testing random number generators. In *Proceedings of the IEEE Winter Simulation Conference* (Arlington VA) . 305–313.

L'ECUYER, P. 1993. A search for good multiple recursive random number genarators. *ACM Trans. Model.Comput. Simul. 3*, 2 (Apr.), 87–98.

L'ECUYER, P., SIMARD, R., CHEN, E. J., AND KELTON, W. D. 2002. An object-oriented random number package with many long streams and substreams. *Oper. Res. 50*, 6, 1073–1075.

L'ECUYER, P. AND ANDRES, T. H. 1997. A random number generator based on the combination of four LCGs. *Math. Comput. Simul. 44*, 44, 99–107.

LÜSCHER, M. 1994. A portable high-quality random number generator for lattice field theory simulations. *Comput. Phys. Commun. 79*, 1 (Feb.), 100–110.

MASCAGNI, M. AND SRINIVASAN, A. 2004. Parameterizing parallel multiplicative lagged-Fibonacci generators. *Parallel Comput. 30*, 7 (Jul.), 899–916.

MASCAGNI, M. AND SRINIVASAN, A. 2000. Algorithm 806: SPRNG: A scalable library for pseudorandom number generation. *ACM Trans. Math. Softw. 26*, 3, 436–461. http://sprng.cs.fsu.edu/.

MATSUI, M. 1994. Linear cryptanalysis method for DES cipher. In *Proceedings of the International Workshop on Theory and Applications Advances in Cryptology* (*EuroCrypt*) (Lofthus, Norway). Lecture Notes in Computer Science, 765. Springer. 386–397.

MATSUMOTO, M. AND NISHIMURA, T. 1998. Mersenne twister: A 623-dimensionally equidistributed uniform pseudorandom number generator. *ACM Trans. Model. Comput. Simul. 8*, 1 (Jan.), 3–30.

MATSUMOTO, M. AND NISHIMURA, T. 2003. Sum-Discrepancy test on pseudorandom number generators. *Math. Comput. Simul. 62*, 3–61, 431–442.

MATSUMOTO, M. AND NISHIMURA, T. 2000. The dynamic creation of pseudorandom number generators. In *Monte Carlo and Quasi-Monte Carlo Methods*. Springer, 56–69.

TEZUKA, S., L'ECUYER, P., AND COUTURE, R. 1994. On the add-with-carry and subtract-with-borrow random number generators. *ACM Trans. Model. Comput. Simul. 3*, 4, 315–331.

Received July 2005; revised February 2007; accepted March 2007