Monte Carlo Methods Appl. Vol. No. (), pp. 1–12
DOI 10.1515 / MCMA.2007.
© de Gruyter

# A New Nonrecursive Pseudorandom Number Generator Based on Chaos Mappings

Hirotake Yaguchi and Izumi Kubo

**Abstract.** We introduce a new pseudorandom number generator SSR (the Simplified Shift-Real random number generator) which generates the *k*-th random number nonrecursively (directly) based on chaos mappings on the interval [1,2). We investigate properties of SSR random numbers and give the theoretical background of generation of random numbers. A practical integral(all-integer) version SSI of SSR, which is suitable for parallel computation, is also provided.

**Keywords.** Random number, Pseudorandom number, Nonrecursive, Chaos, Statistical test, Parallel computation.

**AMS classification.** 11K45, 65C10.

## 1. Introduction

The aim of this paper is to investigate a nonalgebraic and nonrecursive pseudorandom number generator(RNG) which has different algorithm from other RNGs. Random number generators which are used nowadays, for example, feed back shift register RNGs, Mersenne Twister RNG and so on, are designed so that they generate random numbers(RNs) sequentially on a single computer. Therefore it is not so easy to adjust their algorithm to recent parallel computers for parallel generation of RNs to be generated. To solve this problem we here introduce new random number generators SSR and its integral version SSI. They generate RNs nonrecursively by making use of chaotic behavior of cancellation errors in numerical computation. Since SSR and SSI RNGs generate the *k*-th random number directly, they are quite suitable for parallel computations.

One of the authors proposed a new pseudorandom number generator SR(the Shift-Real RNG) in [9] [10]. There, although nonalgebraic and nonrecursive RNGs were given, the theoretical background of the randomness was not clear because of complexity of the SR-algorithm. We here improve the algorithm of SR and introduce the SSR-algorithm which has a simple mathematical structure and makes us possible to analyze randomness theoretically. (Of course, the simplicity of SSR-algorithm

This work was supported by JSPS KAKENHI (17540111,17540138) and the ISM Coopretative Research Program (2007-ISM·CRP-4202).

also improves the speed of generation of RNs.) In the next section, following [9], we give a brief summary of the cancellation error which is the origin of our RNG. It will be seen that a cancellation error can be think of as a shift of the mantissa of a floating-point variable. In Sect. 3 we construct a new nonrecursive RNG SSR by making use of simple multiplication of floating point variables and an artificial shift of mantissa. In Sect. 4 we investigate properties of SSR RNs by applying NIST's suit of statistical tests of randomness. In Sect. 5 we argue the theoretical background of the randomness of SSR RNG by analyzing chaos mappings which correspond to SSR computations. There a little bit of ergodic theory is needed. In Sect. 6, using the knowledge obtained in Sect. 5, we improve randomness and efficiency of SSR RNG. The improvement is realized as the K-improvement and an extension of SSR computation. Since a floating-point system is dependent on a computer system, we introduce in Sect. 7 the SSI random number generator which has the same algorithm as SSR and uses no floating-point computations. The randomness of SSIK(K-improved SSI) RNG is tested by NIST's suit, TestU01 and a $\chi^2$ test concerning the distribution of 211-tera SSRK RNs generated by parallel computations. Finally, in the appendix, we give a code of essential part of SSIK RNG.

## 2. The origin of a new random number generator

The origin of our new pseudorandom number generator is cancellation error of numerical computation [9]. If we compute

$$\begin{aligned}\sin x &\approx x - \tfrac{x^3}{3!} + \tfrac{x^5}{5!} - \tfrac{x^7}{7!} \cdots + (-1)^{n-1}\tfrac{x^{2n-1}}{(2n-1)!} \\ &= x\left(1 - \tfrac{x^2}{2\cdot 3}\left(\cdots\left(1 - \tfrac{x^2}{(2n-4)(2n-3)}\left(1 - \tfrac{x^2}{(2n-2)(2n-1)}\right)\right)\cdots\right)\right)\end{aligned}$$

by Horner's method

$$\begin{aligned}G_0 &= 1, \\ G_k &= 1 - \tfrac{x^2}{(2n-2k)(2n-2k+1)} \times G_{k-1}, \qquad k = 1, 2, \cdots, n-1, \\ G_n &= x \times G_{n-1},\end{aligned}$$

for

$$x_i = \tfrac{30}{19999} \times i, \qquad i = 0, 1, \cdots, 19999,$$

under *single precision* with $n = 30$, we have a strange graph given in Fig. 1(left). Further if we compute $y_i = G_{30}(x_i)$ in the same way for

$$x_i = 22 + \tfrac{4}{19999} \times i, \qquad i = 0, 1, \cdots, 19999,$$

and drop the first 3 digits of $y_i$ and take out the succeeding four digits, say $u_i$, such as

$$y_0 = -6.66\mathbf{8604}37393188E + 000 \qquad \Longrightarrow \qquad u_0 = 8604,$$

then the graph of $(0.u_{2j}, 0.u_{2j+1})$ , $j = 0, \cdots$ ,9999, suggests us that $u_0$, $u_1$, $u_2$, $\cdots$ , are random numbers (Fig. 1 right).

The reason of disorder of the graph is the cancellation error caused by missing of leading significant zeros in floating-point computations

$$G_k = 1 - \frac{x^2}{(2n-2k)(2n-2k+1)} \times G_{k-1} = (-)0.0d_2d_3d_4d_5d_6\cdots$$

for $k =$ 26, 27, 28 and the near. Since the drop of leading zeros can be regarded as a shift of the mantissa of single precision variable such as

$$\overset{\text{mantissa}}{\boxed{00d_2d_3d_4d_5d_6}}?? \xrightarrow{\text{shift left}} 00\overset{\text{mantissa}}{\boxed{d_2d_3d_4d_5d_6??}},$$

if we can cause a shift of mantissa artificially, then we could have a new method of generating random numbers.

## 3. The pseudorandom number generator SSR

We introduce here a new nonrecursive pseudorandom number generator SSR (the *S*implified *S*hift-*R*eal RNG) whose base is an artificial shift of mantissa of floating-point variable. The SSR is an improved and simplified version of SR in [10], and enables us to analyze its randomness mathematically because of its simplicity.

Let $w_0$ and $x$ be in [1,2) and

$$g_{w_0}(x) = w_0 \circ \underbrace{x \circ x \circ x \circ \cdots \circ x \circ x}_{24\ times}.$$

We compute $g_{w_0}(x)$ under the *double precision* (IEEE 754 format) such as

$$w_j = w_{j-1} \circ x,\ j = 1, 2, \cdots, 24, \qquad \text{and}\ \ g_{w_0}(x) = w_{24}.$$

Here the meaning of $\circ$ in $w_j = w_{j-1} \circ x$ is the following(Fig. 2):

*The SSR computation.*

(i) Compute $w_{j-1} \times x$ as a usual floating-point multiplication,

(ii) set the exponent of $w_{j-1} \times x$ so that it represents $\cdots \times 2^0$($\rightarrow$ $w_{j-1} \times x$ is in [1,2) again), and

(iii) shift all the bits of the mantissa of $w_{j-1} \times x$ to the left by 1 ($\rightarrow$ the most significant bit of the mantissa of $w_{j-1} \times x$ is dropped and hence a cancellation error of 1 bit is caused artificially).

Then we construct a pseudorandom number generator SSR by varying $x$ in [1,2) uniformly as follows:

*The SSR random number generator*.
Let

$p$=49933453, $q$=22801201, $r$=491377, $s$=47513, $a$=1920000, $b$=48060000
and set $(r_k, s_k) \equiv (rk \bmod p, sk \bmod q)$ , $k = 1, 2, \cdots$ . For

$$x_k = \begin{cases} 1.0 + \frac{r_k}{a+s_k} & \text{if } (a+s_k) > r_k \\ 1.0 + \frac{r_k-(a+s_k)}{b-s_k} & \text{if } (a+s_k) \le r_k, \end{cases}$$

we compute $g_1(x_k)$ and discard the first three digits of $g_1(x_k)$ and take out the succeeding four digits $\alpha_k$ as the $k$-th random number.

We note that the sequence $\{x_k\}$ is designed so that it spreads uniformly in the interval [1,2). Further the period of RNs $\{\alpha_k\}$ is $pq$ =1,138,542,698,477,053 because $p$, $q$, $r$ and $s$ are prime numbers.

The SSR computation $w_j = w_{j-1} \circ x$ can be treated mathematically in the following way:

*Mathematical representation of the SSR computation* $w_{j-1} \circ x$.
Let

(i) $\varphi_x : [1,2) \to [1,4)$ be $\varphi_x(t) = tx$ , which corresponds to the multiplication of $t \in [1,2)$ by $x \in [1,2)$ in the SSR computation,

(ii) $\varphi_e : [1,4) \to [1,2)$ be $\varphi_e(v) = v$ if $1 \le v < 2$ and $= v/2$ if $2 \le v < 4$, which corresponds to the substitution of the exponent of floating-point variable $v$ by ... $\times\ 2^0$ , and

(iii) $\varphi_s : [1,2) \to [1,2)$ be $\varphi_s(w) = 2w - 1$ if $1 \le w < 1.5$ and $= 2w - 2$ if $1.5 \le w < 2$ , which corresponds to the shift of the mantissa of floating-point variable $w$ to the left by 1.

Then $w_j = w_{j-1} \circ x$ is represented by $w_j = \varphi_s(\varphi_e(\varphi_x(w_{j-1})))$ .

If we set $\Phi_x(t) = \varphi_s(\varphi_e(\varphi_x(t)))$ , then $w_j = \Phi_x(w_{j-1})$ and $\Phi_x$ is a piece-wise affine function which maps [1,2) onto [1,2) and is written explicitly by

if $1 \le x < 1.5$ if $1.5 \le x < 2$

$$\Phi_x(t) = \begin{cases} 2xt - 1 & (1 \le t < \frac{3}{2x}) \\ 2xt - 2 & (\frac{3}{2x} \le t < \frac{2}{x}) \\ xt - 1 & (\frac{2}{x} \le t < 2) \end{cases} , \qquad \Phi_x(t) = \begin{cases} 2xt - 2 & (1 \le t < \frac{2}{x}) \\ xt - 1 & (\frac{2}{x} \le t < \frac{3}{x}) \\ xt - 2 & (\frac{3}{x} \le t < 2) \end{cases} .$$

The graphs of $\Phi_x$ and $\Phi_x^2$, $\Phi_x^4$ for $x$=1.28 are as in Fig. 3.

In the following we sometimes use the word "SSR value" when we want to refer to the value of $g_1(x_k)$=$\Phi_{x_k}^{24}(1)$ (more generally, $g_{w_0}(x_k)$=$\Phi_{x_k}^{24}(w_0)$). A 4-digit number $\alpha_k$ obtained from SSR value $\Phi_{x_k}^{24}(1)$ after dropping the first 3 digits will be called the $k$-th "SSR random number". Since the $k$-th SSR RN $\alpha_k$ is obtained directly through a computation $\Phi_{x_k}^{24}(1)$, RNs $\{\alpha_k\}$ can be generated in parallel. Therefore SSR RNG is suitable for parallel computing. Moreover, by varying numbers $r$ and $s$ in the definition of $x_k$ and also by varying $w_0$ in $\Phi_{x_k}^{24}(w_0)$, we can obtain many different sequences of random numbers. Thus the SSR RNG is a very flexible RNG.

## 4. Statistical test of SSR random numbers

We show, using the suit of statistical tests supplied by NIST(http://csrc.nist.gov/rng/), that SSR RNs $\alpha_k$, $k = 1, 2, \cdots$, are uniform random numbers. It will be known that results for SSR RNs are of the same level with other well-known RNGs.

NIST's suit(Version 1.8) consists of 188 tests of the following 15 kinds: Frequency; block frequency; cumulative sums; runs; longest run; rank; fft; nonperiodic templates; overlapping templates; universal; approximate entropy; random excursions; random excursions variant; serial; and linear complexity tests. We applied all the tests to a file consisting of one-giga bits of $b_7 \cdots b_{22}$ obtained from the mantissa $b_1 \cdots b_{52}$ of $\Phi^{24}_{x_k}(1)$. All the parameters we used were default's values except that the block length of the block-frequency test is 20000. The number of bit-streams and the length of bit are 1000 and 1000000 respectively. Under these circumstances, each test of NIST's suit generated a finalAnalysisReport and gave us a value of $\chi^2$ test concerning the distribution of 1000 *p*-values derived from 1000 bit-streams and also a proportion of *p*-values which are greater than 0.01. In order to exclude exceptional values, we repeated NIST's test 16 times by varying $r$ and $s$ in the definition of $\{x_k\}$ such as

$r$ =491377, 492299, 493177, 404041, $s$ =47513, 47699, 47869, 48049,

and averaged the results in the finalAnalysisReports. Since it is tedious to cite all averages, we give below the maximum and the minimum of averages together with the results for the feedback shift register method (FSR) $Y_n = Y_{n-32}$ xor $Y_{n-521}$ and Mersenne Twister (MT [5]): (In the results, $[i = kk]$ is the serial number of the test.)

**[SSR]**
AvMax of $\chi^2$ test of *p*-value $=$ 0.666747 at nonperiodic-templates [*i*=102]
AvMin of $\chi^2$ test of *p*-value $=$ 0.338575 at nonperiodic-templates [*i*=153]
AvMax of proportion of $(p > 0.01)$ =0.992112 at random-excursions-variant [*i*=183]
AvMin of proportion of $(p > 0.01)$=0.987188 at fft [*i*=7],

**[FSR]**
AvMax of $\chi^2$ test of *p*-value $=$ 0.685129 at nonperiodic-templates [*i*=58]
AvMin of $\chi^2$ test of *p*-value $=$ 0.292520 at nonperiodic-templates [*i*=34]
AvMax of proportion of $(p > 0.01)$=0.991750 at nonperiodic-templates [*i*=44]
AvMin of proportion of $(p > 0.01)$=0.986788 at random-excursions [*i*=166],

**[MT]**
AvMax of $\chi^2$ test of *p*-value $=$ 0.721409 at rank [*i*=6]
AvMin of $\chi^2$ test of *p*-value $=$ 0.304160 at random-excursions-variant [*i*=179]
AvMax of proportion of $(p > 0.01)$=0.992250 at nonperiodic-templates [*i*=138]
AvMin of proportion of $(p > 0.01)$=0.987500 at fft [*i*=7].

From these results we think that concerning NIST's tests properties of SSR RNs are of the same level with other RNs.

## 5. Theoretical analysis of the SSR computation

We consider here the mathematical background why the SSR computation $\Phi_x$ generates random numbers, and also consider the distribution (probability density function) of SSR values $\Phi^{24}_{x_k}(1)$.

The base of the randomness of SSR values is, in a short word, the chaotic property of $\Phi_x$ which is characterized by the following (i), (ii) and (iii) ([1], [8]).

(i) Since $\frac{d}{dt}\Phi_x(t) = x$ or $2x$ and is greater than one, $\Phi_x$ is an expansive function. This implies that the value of $\Phi^n_x(t)$ is extremely sensitive to the initial value $t$ in the sense that there is an $r > 0$ such that for each point $t$ and for each $\varepsilon > 0$ there is a point $t'$ with $|t - t'| < \varepsilon$ and a $k$ such that $|\Phi^k_x(t) - \Phi^k_x(t')| \geq r$ .

(ii) Periodic points with the period $n$ are given by the intersection of the graphs of $y = \Phi^n_x(t)$ and $y = t$, $t \in [1, 2)$. Then as the graphs of $\Phi_x, \Phi^2_x$ and $\Phi^4_x$ in Fig. 3 indicate, $\Phi_x$ has dense periodic points.

(iii) $\Phi_x$ is transitive, that is, the orbit of some point of an invariant set of $\Phi_x$ is dense. This is known by the following consideration. The probability density function $h_x$ of the invariant measure of $\Phi_x$ is given by

$$h_x(t) = \lim_{n\to\infty}\{\tfrac{1}{n}\textstyle\sum_{k=0}^{n-1}\mathcal{L}^k_x\mathbf{1}\}(t)$$

([2], [6]), where $\mathcal{L}_x : L^1[1, 2) \to L^1(1, 2)$ is the Perron-Frobenius(P-F) operator corresponding to $\Phi_x$ which is defined by

$$\text{if } 1 \leq x < 1.5,\quad (\mathcal{L}_x g)(t) = \begin{cases} \frac{1}{2x}g(\frac{t+2}{2x}) + \frac{1}{x}g(\frac{t+1}{x}) & (1 \leq t < 2x-1) \\ \\ \frac{1}{2x}g(\frac{t+1}{2x}) + \frac{1}{2x}g(\frac{t+2}{2x}) & (2x-1 \leq t < 2) \end{cases},$$

$$\text{if } 1.5 \leq x < 2,\quad (\mathcal{L}_x g)(t) = \begin{cases} \frac{1}{x}g(\frac{t+1}{x}) + \frac{1}{x}g(\frac{t+2}{x}) & (1 \leq t < 2x-2) \\ \\ \frac{1}{2x}g(\frac{t+2}{2x}) + \frac{1}{x}g(\frac{t+1}{x}) & (2x-2 \leq t < 2) \end{cases}.$$

(The Perron-Frobenius operator $\mathcal{L}$ corresponding to $\Phi$ is the mapping from $L^1 \ni g$ to $L^1 \ni \mathcal{L}g$ where $\mathcal{L}g$ is the function satisfying $\int f(\Phi(x))g(x)dx = \int f(t)(\mathcal{L}g)(t)dt$ for every bounded measurable function $f$ . Under appropriate conditions, $(\mathcal{L}g)(t)$ is represented such as $(\mathcal{L}g)(t) = \sum_{y\in\Phi^{-1}(t)}\frac{g(y)}{|\Phi'(y)|}$ ([6], [7]). As an example of $h_x(t)$ we show in Fig. 4(left) the graphs of $h_{1.28}$ which is obtained by numerical computation.) Since the distribution of $\{\Phi^n_x(t)\}_{n=1,2,\cdots}$ is approximated by the invariant measure with density $h_x$ for almost $t \in [1, 2)$ (Fig. 4(right)), SSR values $\Phi^n_x(t)$, $n = 1, 2, \cdots$, spread nearly uniformly in [1,2). This implies that for almost $t \in [1, 2)$, if positive $\varepsilon$ and $\tilde{t} \in [1, 2)$ are given, there exists $m \in N$ such that $|\Phi^m_x(t) - \tilde{t}| < \varepsilon$ (i.e., $\tilde{t}$ is arbitrarily approximated by some $\Phi^m_x(t)$), which means the transitivity of $\Phi_x$. From these facts we see that $\Phi_x$ is a chaos mapping.

What we want to stress here is that

the probability density function $H(t)$ of $\{\Phi^{24}_{x_k}(1)\}_k$ can be described
by the average of $h_x$ over $x \in [1,2)$, that is, $H(t) = \int_1^2 h_x(t)dx$.

The intuitive meaning of the assertion above would be obvious if one sees the graphs of $H(t)$ and the distribution of $\{\Phi^{24}_{x_k}(1)\}_{k=1}^{10^7}$ (Fig. 5). As the reason why $H(t)$ is described by $\int_1^2 h_x(t)dx$, we consider the following necessary condition for $H(t)$. Suppose that if $q$ is sufficiently large, the distribution of $\{\Phi^{q}_{x_k}(1)\}_k$ is described by $H(t)$ which is independent of $q$, and for a bounded measurable function $f$ it holds that $\frac{1}{N}\sum_{k=1}^N f(\Phi^q_{x_k}(1)) \approx \int_1^2 f(t)H(t)dt$ for sufficiently large $N$. Then averaging over $q$=$Q$, $Q$+1, $\cdots$, $Q$+$R$-1 we have

$$\frac{1}{N}\sum_{k=1}^N \frac{1}{R}\left\{f(\Phi^Q_{x_k}(1)) + \cdots + f(\Phi^{Q+R-1}_{x_k}(1))\right\} \approx \int_1^2 f(t)H(t)dt.$$

If we let $R \to \infty$ in the above, the LHS approaches to $\frac{1}{N}\sum_{k=1}^N \int_1^2 f(t)h_{x_k}(t)dt$ because the distribution of $\Phi^n_{x_k}(1), n = 1, 2, \cdots$, is approximated by $h_{x_k}$, and hence $\int_1^2 f(t)\left(\frac{1}{N}\sum_{k=1}^N h_{x_k}(t)\right)dt \approx \int_1^2 f(t)H(t)dt$. Since $x_1, \cdots, x_N$ spread uniformly in [1,2) as $N \to \infty$, the term $\frac{1}{N}\sum_{k=1}^N h_{x_k}(t)$ approaches to $\int_1^2 h_x(t)dx$ as $N \to \infty$. So we could conclude that $\int_1^2 f(t)\left(\int_1^2 h_x(t)dx\right)dt = \int_1^2 f(t)H(t)dt$ for every bounded measurable function $f$. This means that $H(t) = \int_1^2 h_x(t)dx$.

To ensure that the distribution of $\{\Phi^{24}_{x_k}(1)\}_k$ is described by $H(t)$, we apply the following statistical test.

Put $z_k = \Phi^{24}_{x_k}(1)$. Then we repeat for $j = 0, 1, \cdots, 9999$ the Kolmogorov-Smirnov test [3] for $z_{1+20000\times j}$, $z_{2+20000\times j}$, $\cdots$, $z_{20000+20000\times j}$ and obtain two statistics $\kappa_j^+$ and $\kappa_j^-$. It is known that the distributions of $\kappa_0^+$, $\kappa_1^+$, $\cdots$, $\kappa_{9999}^+$ and $\kappa_0^-$, $\kappa_1^-$, $\cdots$, $\kappa_{9999}^-$ are closely approximated by the function $1 - \exp(-2x^2)$. So we can employ $\chi^2$ test(Fig. 6). The results of $\chi^2$ test for the distributions of $\kappa^+$ and $\kappa^-$ are 0.20098 and 0.66702 respectively. Hence the hypothesis that the distribution of $\{\Phi^{24}_{x_k}(1)\}_k$ is described by $H(t)$ is not rejected at the level of 0.05.

## 6. Improvement of the SSR random number generator

Although SSR RNs are created by dropping the first 3 digits of SSR values, as the graph of $H(t)$ in Fig. 5 suggests, the distribution has a slight asymmetry which NIST's tests can not detect (Fig. 7 left). Further the size of SSR RNs is four digit which is not so long.

In this section we give two improvements of SSR RNG. One is the K-improvement which improves randomness of SSR RNG, and another is an extension of SSR computation which improves efficiency of generation of RNs.

*The K-improvement of SSR (SSRK).*
Let $1.e = 1.2718281828459$ and $1.\tilde{\pi} = 1.8141592653589$. Then, instead of the SSR value $\Phi^{24}_{x_k}(1)$, we use $\Phi^{24}_{x_k}(1.e) - \Phi^{24}_{x_k}(1.\tilde{\pi})$ (mod [1,2)), which we call an SSRK value. (The meaning of mod [1,2) is to transfer the value in (-1,1) to the one in [1,2) by adding 1 or 2.) An SSRK random number is created in the same way as to SSR random number by discarding the first 3 digits and taking out the succeeding 4 digits.

The K-improvement is making use of the fact that $\Phi_x$ is a chaos mapping, and $\Phi^{24}_{x_k}(w_0)$ is extremely sensitive to the initial value $w_0$. Therefore if we assume that the distributions of $\{\Phi^{24}_{x_k}(1.e)\}_k$ and $\{\Phi^{24}_{x_k}(1.\tilde{\pi})\}_k$ are mutually independent, the density function $I(y)$, $y \in [1, 2)$, of SSRK values is given by the convolution

$$I(y) = \int_1^y H(t)H(t-y+2)dt + \int_y^2 H(t)H(t-y+1)dt.$$

The right graph of Fig. 7 is the theoretical distribution of SSRK RNs which is derived from $I(y)$. We see that the effect of convolution is marvelous.

The next improvement enable us to generate an 8-digit RN at one time.

*The SSRex computation (An extension of SSR computation).*
Let $\tilde{\Phi}_x$ be a modification of $\Phi_x$ such that the size of the left-shift of mantissa is *two*. Then, instead of $\Phi^{24}_{x_k}(1)$, we use $\tilde{\Phi}^{26}_{x_k}(1)$ which we call an SSRex value. An SSRex random number is the succeeding 8 digits of SSRex value after dropping the first 5 digits.

The SSRex technique brings us an important information that the size of the shift of mantissa in the SSR computation is variable. In the next section we will make use of this fact in realizing an integral(=all-integer) version of SSR.

## 7. SSI – A practical integral version of SSR

The SSR computation essentially relies upon floating-point computations, and the floating-point system usually depends on a computer system (especially on MPU). So SSR occasionally generates slightly different RNs if a computer system is different. To avoid this problem we introduce the SSI random number generator which has the same mechanism of random number generation as to SSR and uses no floating-point computations. The base of SSI RNG is the following SSI computation.

*The SSI computation* $\Psi_X(W) = W \bullet X$.
Let $W$ and $X$ be 64-bit integers. Then

(i) compute $W \times X$ as a usual multiplication of 64-bit integers (ignore overflow if it occurs), and

(ii) shift the result of (i) to the right by 32 bits (i.e., bits $b_1 \cdots b_{32}$ are moved to $b_{33} \cdots b_{64}$), and set $b_1 \cdots b_{32} = 0 \cdots 01$ as in Fig. 8. ($\rightarrow W \bullet X$ is used as the next $W$.)

In the real SSI RNG below, the size of overflow ”1YYY” in (i) is adjusted so that it is 4 of 5 bits, which corresponds to the shift of 3 or 4 bits of mantissa in the SSR computation (because the effective drop of bits is ”YYY”).

The next SSIK RNG corresponds to the K-improvement of SSR RNG in Sect. 6.

*The SSIK random number generator.*
Set

*X*=0x88237449a, *P*=0x7ffffffe1, *R*=0x39f750241, *W*=0x18237449a,
*Y*=0xbdda73ad3, *Q*=0x7ffffffcf, *S*=0x32f50fee9, *V*=0x1dda73ad3,

and compute the $k$-th SSIK value such as $(\Psi^{22}_{X_k}(W)) \times X_k - (\Psi^{22}_{Y_k}(V)) \times Y_k$ for $k = 1, 2, \cdots$, where $X_k = (X \text{ xor } (Rk \bmod P))$ and $Y_k = (Y \text{ xor } (Sk \bmod Q))$. Then as the $k$-th SSIK random number we discard the first 16 bits of the $k$-th SSIK value and take out the succeeding 32 bits.

The period of SSIK RNG is $PQ \approx 1.18 \times 10^{21}$ because $P, Q, R$ and $S$ are prime numbers. Results of NIST's test for SSIK RNs are as follows:

**[SSIK]**
AvMax of $\chi^2$ test of *p*-value $=$ 0.661006 at nonperiodic-templates [*i*=38]
AvMin of $\chi^2$ test of *p*-value $=$ 0.320636 at serial [*i*=186]
AvMax of proportion of $(p > 0.01)$=0.991750 at nonperiodic-templates [*i*=33]
AvMin of proportion of $(p > 0.01)$=0.986687 at fft [*i*=7],

Again no particular values are found in the results.

The numbers of hexa-decimal characters 0,1, $\cdots$, F appeared in the hexa-decimal representation $(h_1h_2h_3h_4)_{16}$ of successive 211,106,232,532,992(=0xC00000000000) SSIK RNs are

| | | | | | |
|---|---|---|---|---|---|
| (0) | 52776563456789 | (1) | 52776558944932 | (2) | 52776562922561 |
| (3) | 52776546655991 | (4) | 52776568973476 | (5) | 52776564421452 |
| (6) | 52776553403380 | (7) | 52776560693283 | (8) | 52776561825206 |
| (9) | 52776563077608 | (A) | 52776559739050 | (B) | 52776569167421 |
| (C) | 52776550628626 | (D) | 52776551750245 | (E) | 52776549898173 |
| (F) | 52776544573775, | | | | |

and the $\chi^2$-value is 16.689. Since the $\chi^2$-value at the level 0.05 is 24.9958, the hypothesis of uniformity of appearance is not rejected.

Uniform RNs on [1,2) are made from SSIK RNs by setting the exponent of a floating-point variable to be $\cdots \times 2^0$, and by setting $b_1 \cdots b_{32}$ and $b_{33} \cdots b_{52}$ of mantissa to be 32 bits of the $k$-th SSIK RN and the successive 20 bits of $(\Psi^{22}_{X_k}(W)) \times X_k$ after discarding the first 16 bits respectively. To these RNs we applied BigCrush of TestU01 (V1.2) ([4], http://www.iro.umontreal.ca/-simardr/testu01/ tu01.html), and had the following good result.

```
========= Summary results of BigCrush =========
 Version:          TestU01 1.2
 Generator:        My SSI32 implementation
 Number of statistics:  160
 Total CPU time:   71:24:46.76
 All tests were passed
```

## Concluding Remarks.

(1) The mechanism of random number generation of SSR and SSI is quite different from other RNGs. So in various scenes they can be served as a one-more RNG which has different algorithm of RN generation.

(2) Since SSR and SSI generate the $k$-th RN directly, they are suitable for parallel computations. For example, if the number of RNs required by one job is $n \times m$, then one may split the work into $n$ parts and generate $m$ RNs in the $i$-th subjob $(i = 1, \cdots, n)$ starting from the $(i-1) \times m$-th RN.

(3) Ratios of time for creating a file of one-giga bits for NIST's test are 1(MT), 0.911(FSR) and 3.26(SSIK) under Windows XP x64, Pentium 4 2.8 GHz and Intel's compiler icl 9.0. Since the speed of SSIK is easily covered by parallel computation, we can say that SSIK is a practical random number generator.

## Appendix

We give below the essential part of the code of SSIK RNG stated in Sect. 7.

```
//The following code requires a 64-bit C-compiler.
#include <stdio.h>
typedef unsigned long long int ulli;
typedef struct   //Intel’s MPU only
    {unsigned int L;  unsigned int H;} uiHL;
typedef union {ulli I; uiHL HL;} uiLLHL;
typedef struct
    {
    ulli p; ulli q; ulli r; ulli s; ulli rk; ulli sk;
    ulli XpBase; ulli XqBase; ulli w0; ulli v0;
    } SIPara64;
SIPara64 SIParaK =
    {
    0x7ffffffe1, 0x7ffffffcf, 0x39f750241, 0x32f50fee9, 0, 0,
    0x88237449a, 0xbdda73ad3, 0x18237449a, 0x1dda73ad3
    };
unsigned int SSI32K(void)
    {
    int i;
    uiLLHL WLL1, XLL1, WLL1_Wk, WLL2, XLL2, WLL2_Wk;
    WLL1.I = SIParaK.w0;       WLL2.I = SIParaK.v0;
    XLL1.I = SIParaK.XpBase;   XLL2.I = SIParaK.XqBase;
    XLL1.I ˆ= (SIParaK.rk);    XLL2.I ˆ= (SIParaK.sk);
    for (i=1; i<=23; i++)
        {
        WLL1_Wk.I = WLL1.I*XLL1.I; WLL1.HL.L = WLL1_Wk.HL.H;
        WLL2_Wk.I = WLL2.I*XLL2.I; WLL2.HL.L = WLL2_Wk.HL.H;
        }
    WLL1_Wk.I -= WLL2_Wk.I; WLL1_Wk.I <<= 16;
    return(WLL1_Wk.HL.H);
    }
void SSI32KplusRS(void)
    {
    SIParaK.rk += SIParaK.r;  SIParaK.sk += SIParaK.s;
    if (SIParaK.rk >= SIParaK.p) {SIParaK.rk -= SIParaK.p;}
    if (SIParaK.sk >= SIParaK.q) {SIParaK.sk -= SIParaK.q;}
    }
unsigned int SSI32KBin32(void)
    {
    unsigned int RanBits;
    SSI32KplusRS(); RanBits = SSI32K();
    return(RanBits);
    }
void SSI32IniK(ulli n)
    {
    // (rk,sk) is set so that SSIK starts from the n-th RN.
    }
int main()
    {
    printf("1st random number = %08x\n", SSI32KBin32());
    printf("2nd random number = %08x\n", SSI32KBin32());
    // Use SSI32IniK(n) to start from the n-th RN.
    return(0);
    }
```

## References

1. A. Berger, *Chaos and Chance*. de Gruyter, Berlin New York, 2001.
2. A. Boyarsky and P. Góra, *Laws of Chaos*. Birkhäuser, Boston Basel Berlin, 1997.
3. D. E. Knuth, *The Art of Computer Programming, Volume 2*. Addison Wesley, MA: Reading, 1997.
4. P. L'Ecuyer and R. Simard, *TestU01: A C Library for empirical testing of random number generators*, ACM Transactions on Mathematical Software 33 (2007).

5. M. Matsumoto and T. Nishimura, *Mersenne Twister: a 623-dimensionally equidistributed uniform pseudorandom number generator*, ACM Transactions on Modeling and Computer Simulation 8 (1998), pp. 3–30.
6. W. De Melo and S. Van Strien, *One-Dimensional Dynamics*. Springer, Berlin Heidelberg New York, 1993.
7. M. Pollicott and M. Yuri, *Dynamical Systems and Ergodic Theory*. Cambridge Univ Pr, Cambridge, 1998.
8. C. Robinson, *Dynamical Systems*. CRC Pr, FL: Boca Raton, 1998.
9. H. Yaguchi, *Randomness of Horner's rule and a new method of generating random number*, Monte Carlo Methods and Applications 6 (2000), pp. 61–76.
10. ———, *Construction of a long-period nonalgebraic and nonrecursive pseudorandom number generator*, Monte Carlo Methods and Applications 8 (2002), pp. 203–213.

Received

**Author information**

Hirotake Yaguchi, Faculty of Education, Mie University, Tsu City, Japan.
Email: `yaguchi@edu.mie-u.ac.jp`

Izumi Kubo, Hiroshima Institute of Technology, Hiroshima City, Japan.

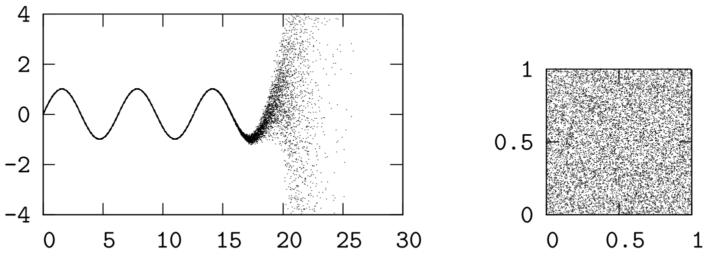


**Figure 1.** A strange graph of $\sin x$ and the graph of pairs of successive $0.u_i$'s

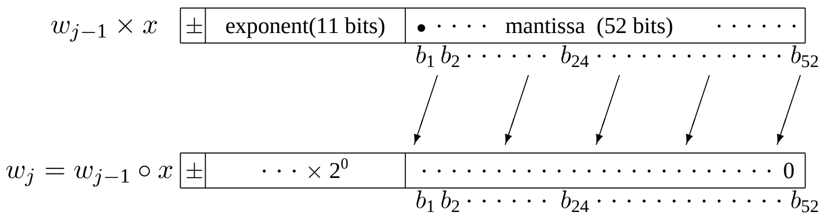

$w_{j-1} \times x$
± exponent(11 bits) • · · · · mantissa (52 bits) · · · · · ·
$b_1\, b_2 \cdots\cdots b_{24} \cdots\cdots b_{52}$
$w_j = w_{j-1} \circ x$
± $\cdots \times 2^0$ $\cdots\cdots$ 0
$b_1\, b_2 \cdots\cdots b_{24} \cdots\cdots b_{52}$

**Figure 2.** The Simplified Shift-Real(SSR) computation

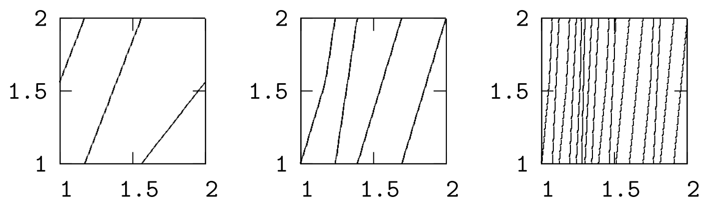


**Figure 3.** Graphs of $\Phi_x$, $\Phi_x^2$ and $\Phi_x^4$ for *x*=1.28

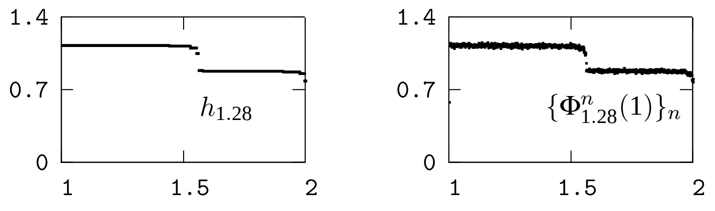


**Figure 4.** Graphs of $h_{1.28}$ and the distribution of $\{\Phi^n_{1.28}(1)\}_n$

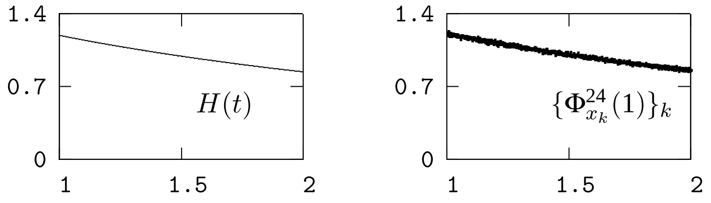


**Figure 5.** Graphs of $H(t)$ and the distribution of $\{\Phi^{24}_{x_k}(1)\}_k$

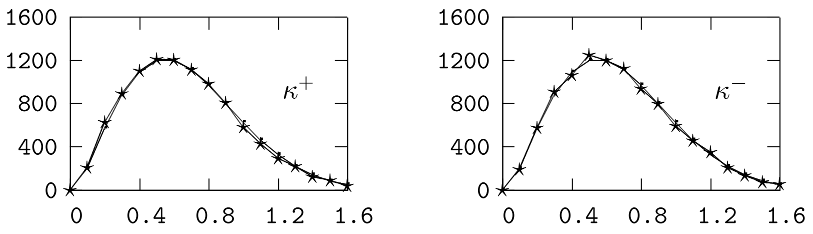


**Figure 6.** Distributions of $\kappa_0^+, \kappa_1^+, \cdots, \kappa_{9999}^+$ and $\kappa_0^-, \kappa_1^-, \cdots, \kappa_{9999}^-$

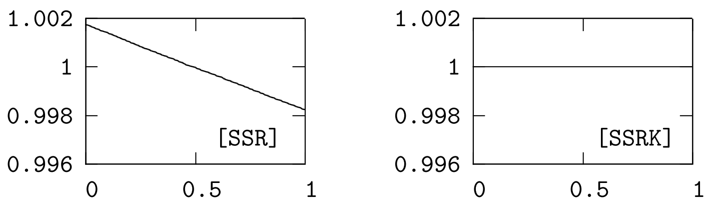
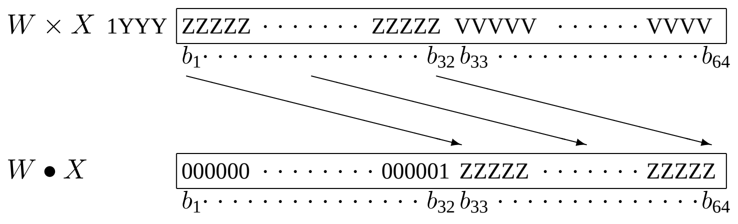


**Figure 7.** Before and after the K-improvement



$W \times X$ 1YYY
ZZZZZ · · · · · · · ZZZZZ VVVVV · · · · · · VVVV
$b_1 \cdots b_{32}\, b_{33} \cdots b_{64}$
$W \bullet X$
000000 · · · · · · · · 000001 ZZZZZ · · · · · · · ZZZZZ
$b_1 \cdots b_{32}\, b_{33} \cdots b_{64}$

**Figure 8.** The Simplified Shift-Integer (SSI) computation