Automation of statistical tests on randomness to obtain clearer conclusion

Hiroshi Haramoto

Abstract Statistical testing of pseudorandom number generators (PRNGs) is indispensable for their evaluation. A common difficulty among statistical tests is how we consider the resulting probability values (p-values). When we observe a small p-value such as 10^{-3}, it is unclear whether it is due to a defect of the PRNG, or merely by chance. At the evaluation stage, we apply some hundred different statistical tests to a PRNG. Even a good PRNG may produce some suspicious p-values in the results of a battery of tests. This may make the conclusions of the test battery unclear. This paper proposes an adaptive modification of statistical tests: once a suspicious p-value is observed, the adaptive statistical test procedure automatically increases the sample size, and tests the PRNG again. If the p-value is still suspicious, the procedure again increases the size, and re-tests. The procedure stops when the p-value falls either in an acceptable range, or in a clearly rejectable range. We implement such adaptive modifications of some statistical tests, in particular some of those in the Crush battery of TestU01. Experiments show that the evaluation of PRNGs becomes clearer and easier, and the sensitivity of the test is increased, at the cost of additional computation time.

1 Introduction

Pseudorandom number generators (PRNGs) are computer programs whose purpose is to produce sequences of numbers that seem to behave as if they were generated randomly from a specified probability distribution. Here we consider the case that the outputs of the PRNG imitate independent random variables from the uniform distribution over the interval $[0,1)$ or over the integers in an interval $\{0,1,2,\ldots,N\}$.

Hiroshi Haramoto
Department of General Education, Kure College of Technology, Hiroshima, Japan, e-mail: haramoto@hiroshima-u.ac.jp

Since PRNGs have a deterministic and periodic output, it is clear that they do not produce independent random variables in the mathematical sense, and that they cannot pass all possible statistical tests of uniformity and independence. But some of them have huge period lengths and turn out to behave quite well in statistical tests that can be applied in reasonable time. On the other hand, some PRNGs, which are known to be defective, fail very simple tests [7].

Many statistical tests for PRNGs are proposed. Widely used examples are: DIEHARD by Marsaglia [16], the test suite of the National Institute of Standards and Technology (NIST) [24], and TestU01 by L'Ecuyer and Simard [11]. When we use such a test suite for a PRNG, the result is a long list of p-values, each value corresponding to each test. It is often difficult to judge whether or not the existence of a few suspicious but not definitive p-values (say, $10^{-10} < p < 10^{-4}$) implies the defectiveness of the PRNG.

The aim of this paper is to eliminate this uncertainty, by proposing an adaptive modification of (essentially any) statistical test for PRNGs. The adaptive version of a statistical test means that we increase the sample size again and again, until we observe a definitely small p-value, or an acceptable normal p-value. This method is not novel; it is commonly used by hand, with heuristics. Our proposal is to automate this process. The rest of this paper is organized as follows. In Section 2, we review the statistical tests for PRNGs. We give a detailed description of an adaptive statistical test in Section 3. In Section 4, we show results of some adaptive statistical tests on some well known PRNGs.

2 Statistical tests

Let $(a_1,\ldots,a_n)$ be a sequence in $[0,1)$ generated by some method, i.e., by a PRNG, which imitates a uniform independent random sequence. Let Y_n be a (test) function of n variables from $[0,1)^n$ to $\mathbb{R}$. A statistical test of $(a_1,\ldots,a_n)$ by Y_n is a function

$$T_{Y_n} : [0,1)^n \to [0,1], \quad (a_1,\ldots,a_n) \mapsto P(Y_n(X_1,\ldots,X_n) \le Y_n(a_1,\ldots,a_n)) \qquad (1)$$

where $X_1,\ldots,X_n$ are random variables with identical, independent distribution (i.i.d.) uniform in $[0,1)$, and $P(Y_n(X_1,\ldots,X_n) \le Y_n(a_1,\ldots,a_n))$ is the probability that $Y_n(X_1,\ldots,X_n) \le Y_n(a_1,\ldots,a_n)$ holds. This probability is called the p-value of the test. If this value is too close to 0 or too close to 1, the null hypothesis that $a_1,\ldots,a_n$ are uniform i.i.d is deemed suspicious. (As the editor pointed out, there are tests where the number n varies according to the values of a_i, but here we treat only the above type of statistical tests.)

If the p-value is extremely small (e.g., less than 10^{-10}), then it is (more or less) clear that the PRNG fails the test. If the p-value is suspicious but does not clearly indicate rejection ($p = 10^{-4}$, for example), it is difficult to judge. When we apply several tests to a PRNG, p-values smaller than 0.01 or larger than 0.99 are often

observed (since such values appear with probability 0.02). Therefore, users of test packages for PRNGs are often troubled by the interpretation of suspicious p-values.

In order to avoid such difficulties, a two-level test is often used, see [1] [4] [5] [15]. In a two-level test, we fix a test function Y_n. At the first level, we apply the test T_{Y_n} to the PRNG to be tested, consecutively k times. Then we obtain k of p-values, $p_1, \ldots, p_k$. At the second level, we test these k values under the null hypothesis of the uniform i.i.d. in the $[0,1]$ interval, by some statistical test such as Kolmogorov-Smirnov test. The resulting p-value is the result of the two-level test. A merit of the two-level test is that it tends to give a clearer result, by accumulating the possibly existing deviation k times. Even if the first-level tests report moderate p-values, the two-level test may give a definitive p-value such as 10^{-8}. However, the possibility of getting a suspicious but not definitive p-value still remains.

Moreover, one may suffer from accumulated approximation error in computing p-values. We often compute p-values by using approximation formula: for example, the p-value of χ^2-test is computed by using an approximation. Therefore, some computing error exists in every p-value. Thus, if the p-values of the first level tests has 1% error in the same direction, and if the second level test uses a large number of these p-values (say, $k = 10000$ times), then the second level may detect the systematic computing error, which may lead to a false rejection [5] [13].

3 Adaptive statistical test

An adaptive statistical test requires a test function of variable sample size. That is, one type of test function Y_n of n variables, where n may vary. This is usually the case: most test statistics for testing PRNGs allow any sample size.

An adaptive statistical test for a PRNG based on Y_n is as follows. Fix a moderately large n. We generate n samples $a_0, \ldots, a_{n-1}$ using the PRNG, and compute the corresponding p-value $p_1 := T_{Y_n}(a_0, \ldots, a_{n-1})$. If p_1 lies in the pre-fixed admissible interval, say, in $[0.1, 0.9]$, then the test ends and does not reject the null hypothesis. Otherwise, we double the sample size, and generate $2n$ new samples $a_n, \ldots, a_{3n-1}$ using the PRNG, and compute the p-value $p_2 := T_{Y_{2n}}(a_n, \ldots, a_{3n-1})$. If p_2 lies in the admissible interval, then we accept. Otherwise, we double the sample size again, namely, we generate the next $4n$ samples using the PRNG, and compute the p-value p_3 for these $4n$ samples using Y_{4n}. We iterate this process, until reaching one of the following three cases:

Rejection: the p-value reaches to a prefixed value for the definitive rejection (e.g. $p = 10^{-8}$),
Acceptance: the p-value falls in the admissible interval,
Give up: the number of iterations reached to a prefixed number (say, 6) to stop the test (considering some limitation of memory and/or computation time).

Merits of the adaptive test are:

1. In most cases, the user obtains clear conclusions. The test takes care of suspicious p-values, until we conclude that they were obtained only by chance, or that they expose a systematic deviation. A final suspicious p-value is obtained only in the "give up" case, which occurs rarely.
2. The approximation errors in computing p-values are not accumulated, contrary to the the two-level tests described in the previous section. In the adaptive test, the larger sample size usually results in the smaller approximation error.

Note that such adaptive tests are appropriate for testing PRNGs, but not for general statistical tests such as census of population, where the sample size is often fixed or limited.

As claimed in the introduction, it is not novel to increase the sample size to resolve the suspicious p-values, and a number of studies exist, which treat delicate issues arising in adaptive tests. There are statistical tests where one need to change the parameters and/or the approximation formula of the distribution, according to the increase of the sample size n. In [9] and [12], the number of the cells in a classical serial test is kept proportional to n. In [10], $n^3/(4k)$ is kept constant. In both cases, these changes are necessary to keep an asymptotic approximation formula. In [12, p. 658], the asymptotic formula is changed according to the sample size n.

Below in §4.1 and §4.2, we treat some toy examples, where we may change the sample size n with keeping other parameters constant. In §4.3, we show a more serious implementation based on TestU01 [11], where appropriate choices of the parameters and approximation formulas are processed in the TestU01 library.

Another difficulty is the choice of the first sample size for the adaptive test. Every PRNG is rejected if the sample size is large enough, but the size depends on the interaction of the type of the PRNG and the test, and there are some thumb-nail rules [10], but we do not discuss here. In §4.1, we treat the case where a risky sample size is known in advance. In §4.3, we depends on a set of sample sizes proposed by TestU01; see below.

4 The results of tests

4.1 Weight distribution test on FSRs

The weight distribution test is a test on the distribution of 1's in a pseudorandom bit sequence $x_1, x_2, \ldots$. We cut the sequence into subsequences of fixed length (here we choose the length 94), and count the number of 1's in each subsequence (i.e., the Hamming weight of each subsequence). The number should conform to the binomial distribution $B(94, 1/2)$. Let n be the sample size, namely, the number of subsequences of length 94 generated to be tested. We categorize the 95 observable values into several categories by merging, and apply the χ^2-test for the goodness-of-fit of the observed values to the binomial distribution. Note that this test is a variant

of the Hamming test treated in the next section, and related tests are included in TestU01 [11, Section 5.2.1].

The tested generator is a trinomial based feedback shift register (FSR) generator, defined by the recurrence

$$x_{j+89} := x_{j+38} + x_j$$

over the two-element field $\mathbb{F}_2$ (i.e., every operation is done modulo 2).

This generator is not an excellent generator; it is a toy-model example to explain how our adaptive test works. Matsumoto-Nishimura [22] computes *risky sample size* of such kind of generators, which means that if the sample size is larger than this size, then a simple weight distribution test will probabilistically reject a generated sequence with significance level 0.01. Thus, we can know an appropriate sample size for the test in advance. The risky sample size of the above generator is reported to be 1.16×10^5, so we choose the initial sample size $n = 50000$ in our adaptive test. Table 1 lists the results of the adaptive test described above (i.e., the acceptable interval is $[0.1, 0.9]$ and the rejection corresponds to the p-value outside $[10^{-8}, 1 - 10^{-8}]$). We apply the same adaptive test to the same generator with three randomly chosen initial values.

Table 1 Weight distribution test on the generator $x_{j+89} = x_{j+38} + x_j$

sample size	50000	100000	200000	400000	result
1st	4.3×10^{-4}	7.4×10^{-3}	1.1×10^{-3}	1.4×10^{-14}	reject
2nd	3.6×10^{-2}	8.6×10^{-3}	1.6×10^{-5}	1.9×10^{-9}	reject
3rd	2.0×10^{-1}				accept

For example, in the first experiment, the p-value with sample size 50000 is 4.3×10^{-4}. It is suspicious, but not in the clear-rejection area ($< 10^{-8}$). Accordingly, the sample size is doubled, and the same test is applied to the new $100,000$ samples, obtaining the p-value 7.4×10^{-3}. After four iterations, the p-value reaches 1.4×10^{-14}, and the bias of the weight distribution of the PRNG becomes clear. The result of the second experiment is similar. The first p-value of the third experiment lies in the acceptance interval, and hence there is no rejection, this time (could be regarded as a false-acceptance).

Table 2 shows the result of the same test on a similar generator based on a 5-term relation $x_{j+89} = x_{j+57} + x_{j+23} + x_{j+15} + x_j$ over $\mathbb{F}_2$. The risky sample size is known to be 6.99×10^7, so we choose the initial sample size to be 7×10^7 [22].

Table 2 Weight distribution test on the generator $x_{j+89} = x_{j+57} + x_{j+23} + x_{j+15} + x_j$

sample size	7×10^7	1.4×10^8	2.8×10^8	5.6×10^8	result
1st	1.8×10^{-2}	4.2×10^{-5}	1.3×10^{-9}		reject
2nd	7.4×10^{-4}	1.1×10^{-5}	2.4×10^{-10}		reject
3rd	3.4×10^{-2}	2.2×10^{-5}	1.8×10^{-5}	4.4×10^{-33}	reject

4.2 Hamming weight test on LCG

Here we treat a classical linear congruential generator (LCG), defined by the recurrence

$$x_{j+1} = 110351245x_j + 12345 \pmod{2^{31}}.$$

The outputs (x_j) of this LCG are considered as 31-bit integers. In the Hamming test, these 31-bit integers are concatenated to be a single bit stream. The bit stream is divided to consecutive subsequences of 60 bits, and the number of 1's in each subsequence is counted. Then, the χ^2-test on the null hypothesis of the binomial distribution $B(60,1/2)$ is applied, in the same way as the previous section. Unlike the previous generator, there seems no theory to detect the risky sample size. We choose the initial sample size n to be $1,000,000$. Table 3 shows the results of three experiments, for randomly chosen initial values.

Table 3 Hamming weight test on the generator $x_{j+1} = 1103515245x_j + 12345 \pmod{2^{31}}$

sample size	10^6	2×10^6	4×10^6	8×10^6	result
1st	0.0×10^{0}				reject
2nd	2.5×10^{-2}	1.6×10^{-8}	8.1×10^{-7}	0.0×10^{0}	reject
3rd	7.0×10^{-2}	3.0×10^{-3}	7.2×10^{-10}		reject

4.3 Crush in TestU01 and the Adaptive Crush

TestU01 by L'Ecuyer and Simard [11] is a strong comprehensive suite of statistical tests for uniform random numbers. TestU01 has flexible parameters, and hence is best suitable to implement the adaptive version of statistical tests, unlike DIEHARD and NIST where the sample size is fixed.

There are three related batteries of tests in TestU01, using different sample sizes. These are the Small Crush, Crush, and Big Crush batteries. Big Crush is the most serious test battery, containing several statistical tests whose computation time and the memory consumption is near the limit of our computer, so partly it does not fit the adaptive version where the sample size is doubled iteratively. So, we choose the Crush battery as the basis for a battery of adaptive versions.

Among the 144 tests in Crush, 48 are not suitable to create adaptive versions. More precisely, (1) some of 48 are unable to create adaptive versions due to the lack of our computing resources, and (2) the others among 48 are two-level-test versions of other tests in Crush. We implement adaptive versions of the rest 96 tests, and call them the *Adaptive Crush* battery.

We apply the Adaptive Crush to the following three generators: an LCG [2] based on the recurrence

$$x_{j+1} = 950706376 x_j \pmod{2^{31}-1}, \tag{2}$$

a subtract with borrow (SWB) [18] based on the recurrence

$$x_i = (x_{i-22} - x_{i-48} - c_{i-1}) \pmod{2^{32}-5}, \tag{3}$$

$$c_i = \lfloor (x_{i-22} - x_{i-48} - c_{i-1})/(2^{32}-5) \rfloor, \tag{4}$$

and TT800 ([20]).

Table 4, 5, and 6 list the tests for which

- the original Crush (namely the first step of the adaptive version) gives a p-value in the interval $[10^{-8}, 1-10^{-8}]$,
- but the p-value given by the Adaptive Crush lies outside $[10^{-8}, 1-10^{-8}]$ (The fourth column shows the number of iteration of doubling sample size until the p-value reached outside $[10^{-8}, 1-10^{-8}]$).

Table 4 Result on $x_{j+1} = 950706376 x_j \pmod{2^{31}-1}$

test name	initial sample size	1st p-value	# of iteration	final p-value
Gap	5×10^6	0.0125	2	$< 10^{-300}$
MaxOft	10^7	0.9863	2	$> 1-10^{-15}$
GCD	10^8	0.9805	3	$> 1-10^{-15}$
PeriodsInStrings	3×10^8	$1-6.6 \times 10^{-8}$	2	$> 1-10^{-15}$
PeriodsInStrings	3×10^8	0.9999	2	$1-7.3 \times 10^{-9}$

Table 5 Result on a subtract with borrow

test name	initial sample size	1st p-value	# of iteration	final p-value
SimpPoker	10^7	0.0505	3	6.1×10^{-13}
SimpPoker	10^7	4.9×10^{-4}	3	3.8×10^{-14}
CouponCollector	10^7	0.0217	4	1.5×10^{-11}
CouponCollector	10^7	0.0328	4	6.0×10^{-15}

Table 6 Result on TT800

test name	initial sample size	1st p-value	# of iteration	final p-value
Gap	5×10^6	1.6×10^{-4}	4	8.9×10^{-10}
RandomWalk1 J	10^6	6.1×10^{-5}	3	5.2×10^{-14}
HammingIndep	10^7	8.5×10^{-3}	2	9.5×10^{-12}

4.4 Comparison of the sensitivity

In order to compare the sensitivity between the original Crush and the adaptive Crush, we apply the same 96 tests as in §4.3 to several PRNGs. Table 7 gives the number of tests which report a p-value outside $[10^{-8}, 1-10^{-8}]$ by the original Crush and the adaptive Crush, respectively.

LCG(m,a,c) means the generator which obeys the recurrence $x_{j+1} := (ax_j + c) \pmod m$. LFib$(m,r,k,\mathtt{op})$ uses the recurrence $x_j := x_{j-r} \mathbin{\mathtt{op}} x_{j-k} \pmod m$, where $\mathtt{op}$ is an operation which can be $+$ (addition), $-$ (subtraction), $*$ (multiplication), $\oplus$ (bitwise exclusive-or). The ran3 generator of Press and Teukolsky [23] is essentially an LFib$(10^9, 55, 24, -)$. Unix-randoms are PRNGs that are LFib$(2^{32}, 7, 3, +)$, LFib$(2^{32}, 15, 1, +)$, LFib$(2^{32}, 31, 3, +)$, with the least significant bit of each random number dropped. Knuth-ran_array2 [4] is LFib$(2^{30}, 100, 37, -)[100, 1009]$, where $[100, 1009]$ indicates that from each 1009 successive terms x_j from the recurrence, the first 100 outputs are retained and the others are discarded.

The notation GFSR(k,r) means the GFSR generator, with recurrence of the form $x_j := x_{j-r} \oplus x_{j-k}$. T800 and TT800 are twisted GFSR generators proposed by Matsumoto-Kurita [19] [20]. MT19937 is the Mersenne Twister of Matsumoto-Nishimura [21]. LFSR113 and LFSR258 are the combined Tausworthe generators of L'Ecuyer [6] designed for 32-bit and 64-bit computers, respectively. Marsaglia [17] recommends the 3-shift PRNGs Marsa-xor32 and Marsa-xor64.

SWB(m,r,k) means a subtract with borrow generator, employing the recurrence $x_j := (x_{j-r} - x_{j-k} - c_{j-1}) \pmod m$, where $c_j := \lfloor (x_{j-r} - x_{j-k} - c_{j-1}) \rfloor$. SWB$(2^{24}, 10, 24)[24, l]$ is called RANLUX with luxury level ℓ [14] [3]. In its original form, it returns 24-bit output values. For our tests, we use a version with 48-bit of precision obtained by concatenating every pair of the outputs to have a 48-bit integer. All these generators are copied from the library of TestU01 [11], except for the above-mentioned modification for SWB.

5 Conclusion

We introduced the notion of an adaptive statistical test. This method clarifies the conclusion from the test: Suspicious p-values are resolved by doubling the sample size iteratively. Experiments showed that this method works well in almost all cases. The sensitivity of the test increased, at the cost of additional computational time. In the experiments shown in Table 7, the Adaptive Crush consumed 3–5 times longer time than the original Crush for most cases (of course, it heavily depends on the number of iterations.)

Among the PRNGs tested in this way, we could not find a generator that passes all the Crush tests but fails one of the adaptive Crush tests. This means that the sample size of the original Crush is very well chosen.

Table 7 The number of rejections in the tests

	original crush	adaptive crush
LCG($2^{31}, 65539, 0$)	86	92
LCG($2^{32}, 69069, 1$)	72	74
LCG($2^{32}, 1099087573, 0$)	75	81
LCG($2^{46}, 5^{13}, 0$)	15	19
LCG($2^{48}, 252144903917, 11$)	8	8
LCG($2^{48}, 5^{19}, 0$)	8	12
LCG($2^{31}-1, 16807, 0$)	10	19
LCG($2^{31}-1, 2^{15}-2^{10}, 0$)	29	33
LCG($2^{31}-1, 397204094, 0$)	6	14
LFib($2^{31}, 55, 24, +$)	9	10
LFib($2^{31}, 55, 24, -$)	11	11
LFib($2^{48}, 607, 273, +$)	3	3
ran3	10	10
Unix-random-32	86	86
Unix-random-64	38	52
Unix-random-128	13	14
Knuth-ran_array2	2	2
GFSR(250, 103)	8	10
GFSR(521, 32)	6	6
T800	29	33
TT800	13	17
MT19937	2	2
LFSR113	6	6
LFSR258	6	6
Marsa-Xor32	78	88
Marsa-Xor64	8	8
SWB($2^{24}, 10, 24$)	26	28
SWB($2^{24}, 10, 24$)[24,48]	3	4
SWB($2^{24}, 10, 24$)[24,97]	0	0
SWB($2^{24}, 10, 24$)[24,389]	0	0
SWB($2^{32}-5, 22, 43$)	8	11
SWB($2^{31}, 8, 48$)	10	10

Acknowledgements I would like to thank Professor Makoto Matsumoto who helped and encouraged me constantly, Professor Pierre L'Ecuyer who gave useful comments, Professor Hirokazu Yanagihara who pointed out the importance of the power of tests, and the anonymous referee for deep and valuable comments. This research has been supported in part by JSPS Grant-In-Aid #19204002, #18654021, #21654017 and JSPS Core-to-Core Program No.18005.

References

1. G. S. Fishman. *Monte Carlo*. Springer Series in Operations Research. Springer-Verlag, New York, 1996. Concepts, algorithms, and applications.
2. G. S. Fishman and L. R. Moore, III. An exhaustive analysis of multiplicative congruential random number generators with modulus $2^{31}-1$. *SIAM J. Sci. Statist. Comput.*, 7(3):1058,

1986.
3. F. James. RANLUX: a Fortran implementation of the high-quality pseudorandom number generator of Lüscher. *Computer Physics Communications*, 97:357–357(1), September 1996.
4. D. E. Knuth. *The art of computer programming, volume 2 (3rd ed.): seminumerical algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.
5. P. L'Ecuyer. Testing random number generators. In *WSC '92: Proceedings of the 24th conference on Winter simulation*, pages 305–313, New York, NY, USA, 1992. ACM.
6. P. L'Ecuyer. Tables of maximally equidistributed combined LFSR generators. *Math. Comput.*, 68(225):261–269, 1999.
7. P. L'Ecuyer. Software for uniform random number generation: distinguishing the good and the bad. In *WSC '01: Proceedings of the 33nd conference on Winter simulation*, pages 95–105, Washington, DC, USA, 2001. IEEE Computer Society.
8. P. L'Ecuyer, J. F. Cordeau, and R. Simard. Close-point spatial tests and their application to random number generators. *Operations Research*, 48(2):308–317, 2000.
9. P. L'Ecuyer and P. Hellekalek. Random number generators: selection criteria and testing. In *Random and Quasi-Random Point Sets*, volume 138 of *Lecture Notes in Statistics*, pages 223–266. Springer, 1998.
10. P. L'Ecuyer and R. Simard. On the interaction of birthday spacings tests with certain families of random number generators. *Mathematics and Computers in Simulation*, 55:131–137, 2001.
11. P. L'Ecuyer and R. Simard. TestU01: a C library for empirical testing of random number generators. *ACM Trans. Math. Software*, 33(4):Art. 22, 40, 2007.
12. P. L'Ecuyer, R. Simard, and S. Wegenkittl. Sparse serial tests of uniformity for random number generators. *SIAM Journal on Scientific Computing*, 24(2):652–668, 2002.
13. P. C. Leopardi. Testing the tests: using pseudorandom number generators to improve empirical tests. Talk in MCQMC 2008, July 2008.
14. M. Lüscher. A portable high-quality random number generator for lattice field theory simulations. *Comput. Phys. Comm.*, 79(1):100–110, 1994.
15. G. Marsaglia. A Current View of Random Number Generators. In *Computer Science and Statistics, Sixteenth Symposium on the Interface*, pages 3–10. Elsevier Science Publishers, 1985.
16. G. Marsaglia. DIEHARD: A battery of tests of randomness. 1996. See `http://stat.fsu.edu/~geo/diehard.html`.
17. G. Marsaglia. Xorshift RNGs. *Journal of Statistical Software*, 8(14):1–6, 2003.
18. G. Marsaglia, B. Narasimhan, and A. Zaman. A random number generator for PCs. *Comput. Phys. Comm.*, 60(3):345–349, 1990.
19. M. Matsumoto and Y. Kurita. Twisted GFSR generators. *ACM Trans. Model. Comput. Simul.*, 2(3):179–194, 1992.
20. M. Matsumoto and Y. Kurita. Twisted GFSR generators II. *ACM Trans. Model. Comput. Simul.*, 4(3):254–266, 1994.
21. M. Matsumoto and T. Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, 8(1):3–30, 1998.
22. M. Matsumoto and T. Nishimura. A nonempirical test on the weight of pseudorandom number generators. In *Monte Carlo and quasi-Monte Carlo methods, 2000 (Hong Kong)*, pages 381–395. Springer, Berlin, 2002.
23. W. H. Press and S. A. Teukolsky. *Numerical recipes in C (2nd ed.): the art of scientific computing*. Cambridge University Press, New York, NY, USA, 1992.
24. A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, and S. Vo. A Statistical Test Suite for Random and Pseudorandom number Generators for Cryptographic Applications. NIST Special Publication 800-22, National Institute of Standards and Technology (NIST), Gaithersburg, MD, USA, 2001. See `http://csrc.nist.gov/rng/`.