Rounding error type random number generator designed by subspace projection

Jer-Ming Tsai
Dept. of Information & Communication
Kun Shan University
Email: tjm@fhl.net

I-Te Chen
Dept. of Healthcare Administration & Medical Informatics Kaohsiung Medical University
Email: itchen@kmu.edu.tw

Jengnan Tzeng*
Dept. of Mathematical Science
National Cheng-Chi University
Email:jengnan@math.nccu.edu.tw

***Abstract* — Since the limitation of real number representation of digital number, rounding error has existed almost everywhere in numerical computation. Some mathematical algorithm that should be finite steps will become infinity iterations due to the existence of rounding error. Using this property, we designed a very simple algorithm for random number generator by subspace projection.**

I. INTRODUCTION

Cryptography is fundamental theory in network security; furthermore, random numbers play an essential role in cryptography. The Random numbers are also very useful in simulation, chaos theory, game theory, information theory, pattern recognition, probability theory, quantum mechanics, statistics, and statistical mechanics. The random numbers could be generated from two major ways – true random number generators (TRNGs) and pseudo-random number generators (PRNGs). TRNGs often generate random numbers from nature phenomena such as dice, coin flipping, flip-flop circuit, oscillator, electromagnetic wave, thermal noise, atmospheric noise, etc. On the other hand, PRNGs often generate random numbers from mathematical functions such as linear congruential to simulate real randomness.

A. PRNGs and TRNGs

A linear congruential random number generator [2] represents one of the best-known PRNGs and was firstly broken by Jim Reeds [3] and then by Joan Boyar [4]. Researchers develop the feedback shift register since then [5]. Tzeng et al. proposed the random numbers generated from divergence of scaling function [6] in 2009. The algorithm of [6] generates a random-like real number sequence first and then extracts a specific bit of every random-like real number as the binary random bits. Furthermore, the random-like real number sequence is produced by the divergence of scaling function in wavelet theorem; and the sequence is determined by its scaling coefficients. In 2010, He Debiao, Chen Jianhua, and Hu Jin, proposed "A Random Number Generator Based on Isogenies Operations."[7] They used character of elliptic curves to generate random numbers. In 2012, Xing-yuan Wang and Xue Qin proposed "A new pseudo-random number generator based on CML and chaotic iteration,"[8] which combined the couple map lattice (CML) and chaotic iteration.

However, all random numbers have to pass the statistical test such as NIST SP-800-22 Rev.1a 15 statistical tests [1] to verify the randomness. We briefly describe statistical test as next section.

B. Statistical Test Suite for Random Number Generators

Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone proposed five statistical tests - Frequency (Monobit) Test, Serial test (two-bit test), Poker test, Runs test, and Autocorrelation test - of random sequences - in Handbook of Applied Cryptography, 1996 [9]; but this verification method is not enough to approve the randomness. Therefore, NIST published Special Publication SP-800-22 (A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications) in 2001, and revised in August 2008 and April 2010 [1]. The SP-800-22 Rev.1a developed 15 statistical tests to verify the randomness of random numbers produced by either PRNGs or TRNGs. A qualified random number generator should pass all 15 tests listed in the following:

1. The Frequency (Monobit) Test
2. Frequency Test within a Block
3. The Runs test
4. Test for the longest-Run-of-Ones in a Block
5. The Binary Matrix Rank Test
6. The Discrete Fourier Transform (spectral) Test
7. The Non-overlapping Template Matching Test
8. The Overlapping Template Matching Test
9. Maurer's "Universal Statistical" Test
10. Linear Complexity Test
11. The Serial test (two-bit test)
12. The Approximate Entropy test
13. The Cumulative Suns (Cusum) test
14. The Random Excursions test
15. The Random Excursions Variant test

In addition, the NIST issues the 140 Publication Series to coordinate the requirements and standards for cryptographic modules. The FIPS Pub 140-1 and 140-2 were published in 1994 and 2004 respectively [10]. Revised draft FIPS Pub 140-3 adding on new security features that reflect recent advances in technology and security methods was published in 2009 [11]. FIPS Pub 140 serials recommend some statistical tests and FIPS pub 800-90a recommend random number generator using Deterministic Random Bit Generator and providing validation system [12]. For more other statistical tests please refer to [10-13].

C. Rounding Errors in numerical computing

In numerical computation, the floating point is an approximated representation of real numbers. The numbers are usually represented as a fixed number of significant digits and scaled using an exponent by a certain base. The base of digital numbers is normally 2.

The IEEE 754 is a standard for binary floating-point numbers. The Double precision is the most used format in numerical computation, which is a binary format that occupies
64 bits and its significance has a precision of 53 bits. The non-representational error is in the scale of 10-16. Since the digital number using finite digits to represent real numbers, the unexpected consequences are derived from the following types:

- The associative law might be not followed for floating-point addition and multiplication.
- The distributive law might be not followed.
- Subtracting a number from another nearly equal number may result in loss of significance.
- $a(1/a)$ might not equal to 1 for some floating-point number a.

Hence, the more computations above occur, the more rounding errors appear. In numerical computing, we try to avoid these rounding errors. Contrarily, we will try to involve these rounding errors to make the computation results un-expected.

For example, the following two equations are equivalent.

$$y_1 = x_1^2 - x_2^2,$$

and

$$y_2 = \exp(\log\frac{(x_1 + x_2)}{(x_1 - x_2)} + 2\log(x_1 - x_2))$$

Given the same x_1 and x_2, the results of y_1 and y_2 should be the same. If we set x_1 = 0.02873 and x_2 = 0.028699997 in Matlab, then $y_1 = 1.7230721999909 * 10^{-6}$ and
$y_2 = 1.723072199990839 * 10^{-6}$. The unnecessary computation makes a mere rounding error. In the next section, we will propose a linear projection method to produce a sequence of random numbers.

II. SUBSPACE PROJECTION RANDOM NUMBER GENERATOR

Some mathematical algorithms should have only finite steps theoretically, for example, the Gram-Schmidt process, conjugate gradient method, and etc. However these mathematical algorithm will go more steps even infinity steps due to the existence of rounding errors. When the unexpected steps occur, people usually stop the algorithm and ignore the outputs. If the number of the unexpected steps becomes infinity and the outputs look like randomly, it is probably using this properties to design the random number generator.

Gram-Schmidt process is one of such algorithms. Given an m-by-n matrix A, where $n > m$, Gram-Schmidt process looks for most m orthogonal columns that derived from the columns of A. Because there are at most m independent vectors in R^m space, Gram-Schmidt process obtains all zero vectors after processing $m + 1$ columns. The $m + 1$ steps should completely be projected in the column space that spanned by the previous m columns, if those columns are independent. But we will see that there is still none zero vectors obtained after $m + 1$ steps in practical. Those unexpected vectors are produced by the effect of rounding error. Using this practical property, we can design a new random number generator. In next subsection, we will propose a linear projection method that includes many rounding error effects to design a simple random number generator.

A. *Main Methodology*

Let $A \in M_n(Z)$ be an integer square matrix and Q, R are derived from the QR decomposition of A. That is $A = QR$, where Q is a unitary matrix in $M_n(R^1)$ and R is a triangular matrix. If A is full rank, then the columns of Q are an orthogonal basis of R^n. In general, if each element of A is randomly chosen from Z, A will be full rank and Q is an orthogonal basis of R^n.

If $Q = [q_1, q_2, \ldots, q_n]$, where q_i is the i-th column of Q, we set $x_0 = q_1$ and $Q = [q_2, \ldots, q_n]$.

Since q_1 is perpendicular to q_i for $i = 2, \ldots, n$, we have:

$$QQ^T x_0 = 0. \quad (1)$$

We can rewrite the equation (1) by:

$$x_1 = x_0 - \sum_{i=2}^{n} (q_i^T x_0) q_i. \quad (2)$$

$x_0 - (q_i^T x_0)\, q_i$ means x_0 removes the factor in q_i direction. If we normalize x_0 to be a unit vector in each time that we remove the factor in q_i direction for $i = 1, \ldots, n$, the result should be the same, say x_0, theoretically. However, this might not be true in numerical computing, that is $x_1 \neq x_0$ in the numerical sense.

If we reset $Q = [q_2, q_3, \ldots, q_n, x_k]$ for every time we obtain the new xk and repeat the following equation:

$$x_k = x_{k-1} - \sum_{i=2}^{n} (q_i^T x_{k-1}) q_i,$$

we can get a vector sequence $\{ x_k \}$.

Please note that Q is always a unitary matrix, because its columns are orthogonal to one other. x_k is almost parallel to q_1 with some modification by rounding error. The cyclic permutation of columns of Q makes the output more random. After the random vector sequence $\{ x_k \}$ be obtained, we can easily fabricate the randomly real number or randomly binary number from the element of x_k. The algorithm 1 is the basis algorithm of our method.

In the step 8 of algorithm 1, the normalization

makes more rounding error. In the step 15 of algorithm 1, $f(x)$ is a function that transfers the i-th element x_k to the certain format the we want. For example, if $f(x) = x$, we assign the i-th element as the output of random number and the final output R becomes a randomly real number sequence. If $f(x)$ transfers x to a binary representation and then extract one of the bit from the significance part of binary representation, then the output R will become a randomly binary sequence.

We restrict matrix A in $M_n(Z)$ space, because we hope A can be determined by the user input password in practice. Then Q is determined by A, the initial x_0 is determined by Q and the result R is also determined. Actually the initial x_0 can be not determined by Q, we can release this criterion for further applications. Another important fact is that this rounding error type of projection is not linear, because we have insert the normalization in each steps.

Algorithm 1 Rounding error random number generator

Require: An integer $n \geqq 2$ and $N > 0$
$A \in M_n(Z)$
$QR = A$
$k = 0$
$R = []$
Ensure: A is full rank.
1: while $k < N$ do
2: $x_k = Q(:, 1)$
3: $\hat{x} = x_k$
4: $Q = Q(:; 2 : n)$
5:
6: for i=1:n-1 do
7: $x_k = x_k - Q(:; 2)*(Q(:; 2)^{'} * x_k)$
8: $x_k = x_k / \| x_k \|$
9: end for
10: $x_k = \hat{x} - x_k$
11: $x_k = x_k / \| x_k \|$
12: $Q = [Q| x_k]$
13:
14: for i=1:p do
15: $r = f(x_k(i))$
16: $R = [R|r]$
17: $k = k+1$
18:
19: if $k = N$ then
20: break
21: end if
22: end for
23: end while
24: return R

III. EXPERIMENTAL RESULT

We have try the square matrix with size for 3, 5, 8, 10, 16, 32, 64 and 128. For the fixed matrix size, we randomly chosen matrix A from $M_n(Z)$, where the element of A belongs to [−100, 100]. The function $f(x) = b(x, t)$ transfers real number x to the significance part of IEEE 754 representation and then extract the t-th bit. Then the result R is a randomly binary sequence. The computer specification of experience is AMD Phenom II X2 555(3.2G Hz, 2 Cores), 4G RAM; and the MATLAB 2013a. We are generating 1,000,000 random bits spend around 3 minutes.

For the fixed matrix size, we repeat 100 times to obtain R with length 1000000. Then we use NIST SP800-22 Rev.1a to check the randomness of our random number. The results of the previous experiments are shown in Figure 1 to Figure 8. The y-axis is the pass rate of each test. The x-axis is the bit location in the significance part of IEEE 754. The red bar is the average of the pass rate of 100 experiments and the blue bar is the minimal pass rate.

We can see that the previous bits of the significance part is not useful. This is natural that we normalize the vector for each steps in our algorithm and the previous parts of the significance part is related to the average size of elements in the one norm vector. If the vector belongs to R^n, the average of each element is $\sqrt{1/n}$.

We also see that the small size matrices are not useful. We recommend that the matrix size is greater than 8-by-8 and the bit locations are among 10 to 40.

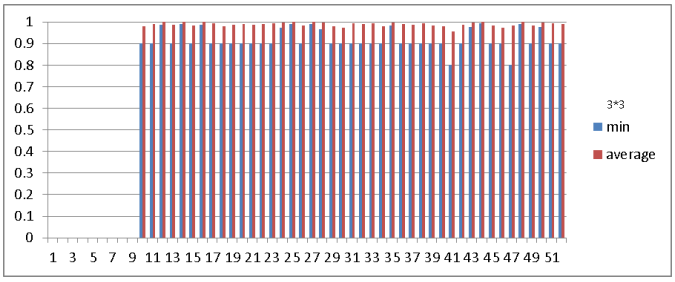

Fig. 1. Test Results of 3-by-3 matrix

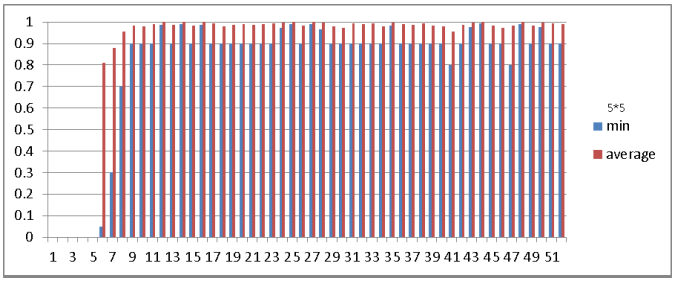

Fig. 2. Test Results of 5-by-5 matrix

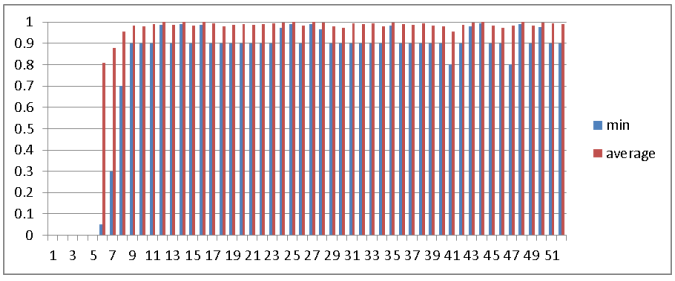

Fig. 3. Test Results of 8-by-8 matrix

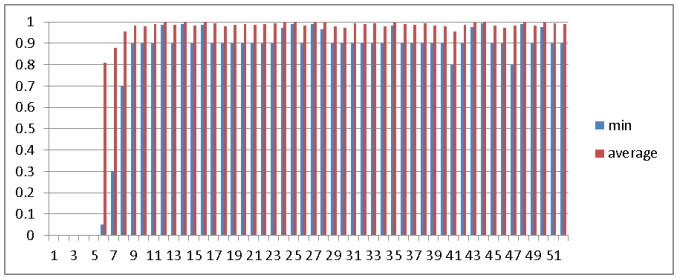

Fig. 4. Test Results of 10-by-10 matrix

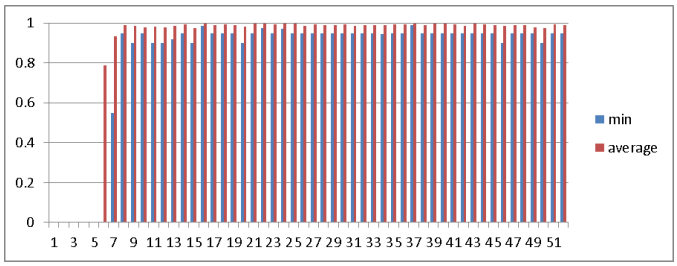

Fig. 5. Test Results of 16-by-16 matrix

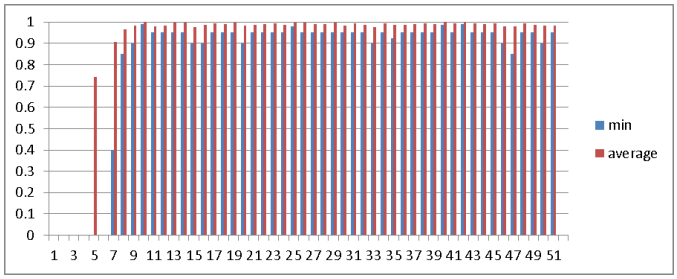

Fig. 6. Test Results of 32-by-32 matrix

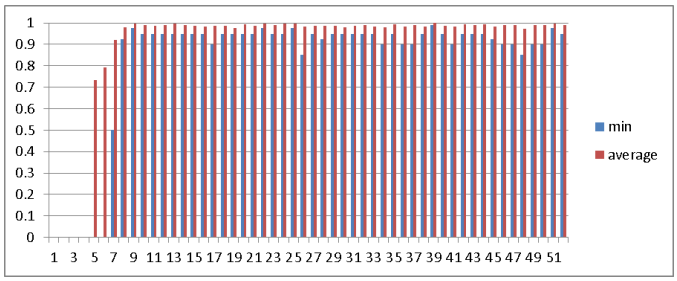

Fig. 7. Test Results of 64-by-64 matrix

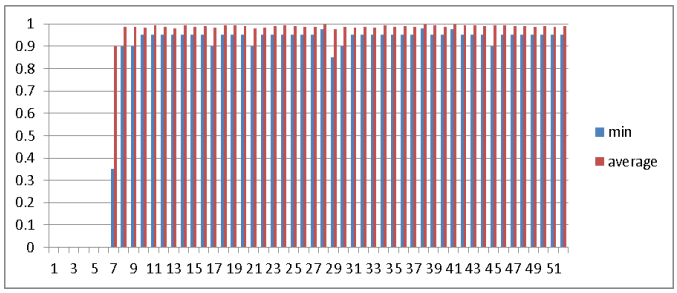

Fig. 8. Test Results of 128-by-128 matrix

IV. CONCLUSION

We propose a rounding error type of random number generator designed by the linear projection. The main idea is using the experiment when serious rounding error can be produced in numerical computing. The special design of projection makes this method is fast and simple. Since the special projection is not a linear operation, this make the random output can pass the NIST SP800-22 Rev.1a.

ACKNOWLEDGMENT

This work was supported in part by the National Science Council under the Grants NSC 102-2218-E-168-001-.

REFERENCES

[1] NIST, FIPS Special Publication 800-22 Rev.1a “A Statistical Test Suite for the Validation of Random Number Generators and Pseudo Random Number Generators for Cryptographic Applications”, April 2010.

[2] J.B Plumstead, “Inferring a Sequence Generated by a Linear Congruence,” Proceedings of the 23th IEEE symposium on the Foundations of Computer Science, pp.153-159, ISSN: 0272-5428, 1982.

[3] J.A. Reeds, “Cracking Random Number Generator,” Cryptologia, Vol.1, No.1, pp.20-26, 1997.

[4] J. Boyar, “Inferring sequences produced by pseudo-random number generators,” Journal of the ACM Vol. 36, Issue 1, pp.129-141, 1989.

[5] P. Alfke, “Efficient Shift Registers, LFSR, Counters, and Long Pseudo-Random Sequence Generators,” XAPP 052, (Version 1.1), 1996. http://www.xilinx.com/support/documentation/application_notes/xapp052.pdf

[6] Jengnan Tzeng, I-Te Chen*, and, Jer-Min Tsai “Random Number Generator designed by the divergence of scaling functions,” International Conference on Intelligent Information Hiding and Multimedia Signal Processing (IIHMSP 2009), pp.1038-1041, September 12-14, 2009.

[7] He Debiao, Chen Jianhua, and Hu Jin, “A Random Number Generator Based on Isogenies Operations.” http://eprint.iacr.org/2010/094, 2010.

[8] Xing-yuan Wang and Xue Qin, "A new pseudo-random number generator based on CML and chaotic iteration," Nonlinear Dynamics, Vol.70, No.2, pp-1589-1592, 2012.

[9] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone, Handbook of Applied Cryptography, CRC Press, pp. 169-190, October 16, 1996. ISBN: 0849385237.

[10] NIST, FIPS PUB 140-2, “Derived Test Requirements for FIPS PUB 140-2, Security Requirements for Cryptographic Modules”, Federal Information Processing Standards Publication, March 2004.

[11] NIST, Revised draft FIPS 140-3, December 11, 2009.

[12] NIST, Revised draft FIPS 800-90a, Jan. 2012.

[13] George Marsaglia, “DIEHARD: a battery of tests of randomness,” the preceding description of the DIEHARD executable program that explains the significance of the results, 1995.