Pseudorandom Number Generators

Seminar 1

Mathew Thiel

Scientific Computing, 01-22-2024

Contents

1 Background

1.1 What is a Pseudorandom Number Generator (PRNG)?

A formal definition of a PRNG is "a pseudo-random number generator (PRNG) is "A deterministic computational process that has one or more inputs called "seeds", and it outputs a sequence of values that appears to be random according to specified statistical tests. A cryptographic PRNG has the additional property that the output is unpredictable, given that the seed is not known." [9]. PRNGS are not truly random, but are an attempt at a close approximation given speed and domain constraints.

1.2 Common Testing Methods

1. TestU01

   TestU01 is a widely used and rigorous testing library that contains a collection of tests for empirical statistical testing of random number generators. The original version is implemented in C and contains 3 notable tests, nicknamed "Small Crush", "Crush", and "Big Crush". Detailed implementation can be found in the user guide [14]

2. Chi-Square Test

   Chi-Square is a common statistical test that essentially provides an approximate probability of how likely a given observation is, and from this randomness can be determined. This is a foundational test for many other empirical tests, and may be considered one of the most used tests for RNGs. [3]

3. Kolmogorov-Smirnov Test

This test is useful as a counterpart to Chi-Square when Chi-Square cannot be applied. This test essentially creates a continuous Cumulative Density Function (CDF) and compares the tested RGN to it. This is another foundational test and is widely used. [3]

2 Common PRNG's

2.1 Linear Congruential Generator

The Linear congruential generator produces a sequence of integers between zero and parameter m-1. The statistical properties of the generator can be drastically different based on the selection of these parameters. Along with being one of the oldest PRNG's, this method is generally considered the simplest and easiest to understand PRNG. This method has been used in various software packages and hardware applications like C++ 11, Visual Basic, and Java's random function. [21]

The math for this PRNG is as follows [30]:

$$X_{i+1} = (a * X_i + c) \bmod m \qquad (i = 1, 2, \ldots, m \text{ -1})$$

with parameters:

X_0 = initial seed

a = constant multiplier

c = increment parameter

m = "modulus" or remainder parameter

Advantages:

1. Small state

2. Stepping forward is easy

3. Can obtain the same results with any number of threads

Disadvantages:

1. Restricted output set

2. Reliant on proper parameter choice for good properties and maximal period

3. Outdated compared to newer methods

2.2 Mersenne Twister

The Mersenne Twister is a PRNG algorithm initially proposed by Matsumoto and Nishimura [15] . It "has the period $2^19937 - 1$ and a 623-dimensional equidistribution property" [15] which seems to be one of the best at the time. This PRNG is implemented in C++ 11 and is the default random number generator in Python.

The general equation for the Mersenne Twister is the linear occurrence:

$$\mathbf{x}_{k+n} := \mathbf{x}_{k+m} \oplus (\mathbf{x}_k^u | \mathbf{x}_{k+1}^l) A$$

And it is explained more in-depth in the work *The Mersnne Twister* [10]

Advantages:

1. Passes most statistical tests

2. Creates 32-bit or 64-bit numbers

3. Large period

4. Large dimensionality

Disadvantages [17]:

1. Not cryptographically secure

2. Relatively slow

3. Not very memory efficient

4. Output is uneven

2.3 Philox

Published as one of the RNG's proposed in the 2011 paper *Parallel Random Numbers: As Easy as 1, 2, 3* [22], Philox is a counter-based PRNG. These types of counters are generally used for Monte-Carlo and other statistical simulations that require large parallel random number generations. It is implemented in C++ 11, along with the GPU-optimized libraries cuRAND and rocRAND and Python implementation in NumPy.

Philox uses an "Advance Randomization System (ARS) which iterates bijection with rounds of the Feistel function and a couple of XOR operations." [13]) and the specific math is explained more in depth in *Parallel Random Numbers: As Easy as 1,2,3* [22]

Advantages:

1. Small state

2. Long period

3. Cryptographically secure

4. Supports parallel simulations and is easy to vectorize and parallelize

5. Suited to a wide variety of hardware

6. Passes statistical tests

Disadvantages:

1. Might not work in applications that can't afford wasting memory to store state and initialization parameters [20]

2.4 Splitmix

Splitmix is an object-oriented and splittable PRNG. It is capable of 9 64-bit operations bet 64-bits generated. A splittable PRNG has the property that each generation has another operation that replaces the original object with two independent PRNG objects. This allows for easy use with multi-threaded programs. This method is suitable for GPUs and SIMD programming and is widely implemented in programming languages such as Java and C++. [25]

As this method is object-oriented, the algorithm is better defined in pseudo-code than math:

```
uint64 state                                  /* The state can be seeded
                                                 with any (upto) 64 bit
                                                 integer value. */


 next_int() {
```

```
    state += 0x9e3779b97f4a7c15              /* increment the state
                                               variable */
    uint64 z = state                         /* copy the state to
                                               a working variable */
    z = (z ^ (z >> 30)) * 0xbf58476d1ce4e5b9 /* xor the variable with
                                               the variable right bit
                                               shifted 30 then multiply
                                               by a constant */
    z = (z ^ (z >> 27)) * 0x94d049bb133111eb /* xor the variable with
                                               the variable right bit
                                               shifted 27 then multiply
                                               by a constant */
    return z ^ (z >> 31)                     /* return the variable xored
                                               with itself right bit shifted 31 */
}

next_float() {
    return next_int() / (1 << 64)            /* divide by 2^64 to return
                                               a value between 0 and 1 */
}
```

Source: *Pseudo-random numbers/Splitmix64* [2]

Advantages:

1. Good enough for some use cases

2. Fast to calculate

3. Passes several statistical tests

4. Works with parallelization

Disadvantages:

1. Not cryptographically secure

2. Not recommended for demanding RGN requirements

2.5 Xoroshirto128+

Xoroshiro128+ is a version of F2 linear PNGR that is high-speed and has fairly low memory requirements at only a few hundred bits. This PRNG is relatively recent and considered fairly effective. Used in web browsers such as Google Chrome, Firefox, Safari, and Microsoft Edge, and is used in programming languages such as Java, Julia, and C. [23]

The base xoroshiro linear transformation is defined in a $2wx2w$ matrix:

$$X_{kw} = \begin{pmatrix} R^a + S^b + I & R^c \\ S^b + I & R^c \end{pmatrix} \tag{1}$$

"We denote with S the wxw matrix on $Z/2Z$ that effects a left shift of one position on a binary row vector (i.e., S is all zeroes except for ones on the principal subdiagonal) and with R the wxw matrix on $Z/2Z$ that effects a left rotation of one position (i.e., R is all zeroes except for ones on the principal subdiagonal and a one in the upper right corner)." [7]

Advantages:

1. Recent (2016)
2. Fast to calculate
3. Passes all tests at upper bits
4. Efficient in hardware

Disadvantages:

1. Lower bits fail some linearity tests
2. Test fails after 5 TB of output (can switch to higher bit versions like xoshiro256+)
3. Not cryptographically secure

2.6 Other Generators

1. Wichmann-Hill, ASA183 [28]

   It was found that combining weak PRNGs results in an overall stronger PRNG, an example of which is the Wichmann-Hill generator.

2. XorShift [27]

   semi-large period shift PRNG

3. O'Neil PCG Family Generators [18]

   https://www.pcg-random.org/pdf/toms-oneill-pcg-family-v1.02.pdf

4. Sponge-Based PRNG [4]

   A recent method of PRNGs, although efficacy needs more research

5. Chaotic Maps [16]

6. Middle Square [6]

7. MELG-64 [12]

   Based on F2-linear generators similar to Philox

8. Squares [29]

   A recent method of PRNGs, although efficacy needs more research

9. ThreeFry [22]

   Developed by the D.E Shaw along with Philox, Philox may be more advanced

10. PRNG created from Reinforcement Learning [19]

2.7 Areas of Further Research

1. *A search for good pseudo-random number generators: Survey and empirical studies* [5]

2. *An Empirical Study of Non-Cryptographically Secure Pseudorandom Number Generators*[24]

References

[1] Random123: a Library of Counter-Based Random Number Generators.

[2] Pseudo-random numbers/Splitmix64, 06 2023.

[3] Dan Beibighauser. Testing Random Number Generators. *REU Summer 2000*, 2000.

[4] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Sponge-Based Pseudo-Random Number Generators. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010*, Lecture Notes in Computer Science, pages 33–47, Berlin, Heidelberg, 2010. Springer.

[5] Kamalika Bhattacharjee and Sukanta Das. A search for good pseudo-random number generators: Survey and empirical studies. *Computer Science Review*, 45:100471, August 2022.

[6] Anirban Bhowmick, Nishith Sinha, R Vijaya Arjunan, and B Kishore. Permutation-Substitution architecture based image encryption algorithm using middle square and RC4 PRNG. In *2017 International Conference on Inventive Systems and Control (ICISC)*, pages 1–6, January 2017.

[7] David Blackman and Sebastiano Vigna. Scrambled Linear Pseudorandom Number Generators, March 2022. arXiv:1805.01407 [cs].

[8] Pavel Dyakov, Ilya Burylov, Ruslan Arutyunyan, Andrey Nikolaev, and Alina Elizarova. Philox as an extension of the C++ RNG engines, January 2023.

[9] CSRC Content Editor. PRNG - Glossary | CSRC.

[10] Fionn Fitzmaurice. The Mersenne Twister.

[11] James Hanlon and Stephen Felix. A Fast Hardware Pseudorandom Number Generator Based on xoroshiro128. *IEEE Transactions on Computers*, 72(05):1518–1524, May 2023. Publisher: IEEE Computer Society.

[12] Shin Harase and Takamitsu Kimoto. Implementing 64-bit Maximally Equidistributed F $_2$ -Linear Generators with Mersenne Prime Period. *ACM Transactions on Mathematical Software*, 44(3):1–11, September 2018.

[13] S. Y. Jun, P. Canal, J. Apostolakis, A. Gheata, and L. Moneta. Vectorization of random number generation and reproducibility of concurrent particle transport simulation. *Journal of Physics: Conference Series*, 1525(1):012054, April 2020. Publisher: IOP Publishing.

[14] Pierre L'Ecuyer and Richard Simard. A Software Library in ANSI C for Empirical Testing of Random Number Generators.

[15] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1):3–30, January 1998.

[16] Rasika B. Naik and Udayprakash Singh. A Review on Applications of Chaotic Maps in Pseudo-Random Number Generators and Encryption. *Annals of Data Science*, January 2022.

[17] M. E. O'Neill. Specific Problems with Other RNGs, August 2014.

[18] Melissa E O'Neill and Harvey Mudd College. PCG: A Family of Simple Fast Space-Efficient Statistically Good Algorithms for Random Number Generation. *ACM Transactions on Mathematical Software*.

[19] Luca Pasqualini and Maurizio Parton. Pseudo Random Number Generation: a Reinforcement Learning approach. *Procedia Computer Science*, 170:1122–1127, January 2020.

[20] Jonathan Passerat–Palmbach, Claude Mazel, and David R. C. Hill. TaskLocalRandom: a statistically sound substitute to pseudorandom number generation in parallel java tasks frameworks. *Concurrency and Computation: Practice and Experience*, 27(13):3383–3398, September 2015.

[21] MJ Rutter. Linear Congruential Random Numbers.

[22] John K. Salmon, Mark A. Moraes, Ron O. Dror, and David E. Shaw. Parallel random numbers: as easy as 1, 2, 3. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12, Seattle Washington, November 2011. ACM.

[23] Sebastiano Vigna. xoshiro / xoroshiro generators and the PRNG shootout.

[24] Mehul Singh, Prabhishek Singh, and Pramod Kumar. An Empirical Study of Non-Cryptographically Secure Pseudorandom Number Generators. In *2020 International Conference on Computer Science, Engineering and Applications (ICCSEA)*, pages 1–6, March 2020.

[25] Guy L. Steele, Doug Lea, and Christine H. Flood. Fast splittable pseudorandom number generators. In *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications*, pages 453–472, Portland Oregon USA, October 2014. ACM.

[26] Ying Tan. Mersenne Twister - an overview | ScienceDirect Topics.

[27] Sebastiano Vigna. An Experimental Exploration of Marsaglia's xorshift Generators, Scrambled. *ACM Transactions on Mathematical Software*, 42(4):30:1–30:23, June 2016.

[28] B. A. Wichmann and I. D. Hill. Algorithm AS 183: An Efficient and Portable Pseudo-Random Number Generator. *Applied Statistics*, 31(2):188, 1982.

[29] Bernard Widynski. Squares: A Fast Counter-Based RNG, March 2022. arXiv:2004.06278 [cs].

[30] Meng Xiannong. Linear Congruential Method, October 2002.