Chi-square of Pseudorandom Number Generator of Normal Distribution in C++17

Pavel Tomášek, Hana Tomášková, Jakub Rak

Tomas Bata University in Zlín, nám. T. G. Masaryka 5555, 760 01 Zlín, Czech Republic

***Abstract* – High quality pseudorandom number generators were needed in many software solutions throughout the history of programming. Nowadays, these generators play an even more significant role in software development. Generally, these generators bring a certain level of coincidence in some algorithms which need it. This work focuses on the statistical evaluation of one of the representatives of the generators using Pearson's Chi-square goodness of fit test. The generator of pseudorandom numbers under test is the specific implementation in the modern standard of the programming language of C++ (the standard of C++17). Results presented in this paper inform whether the numbers generated by the selected generator follow the desired probability distribution (normal).**

***Keywords* – chi-square, pseudorandom number generator, C++, normal distribution**

1. Introduction

Numbers obtained from pseudorandom generators are used in many software solutions. Reasons for the usage of these pseudorandom numbers vary from implementation of a type of (pseudo) coincidence (pseudorandom selection from a set of values, testing purposes), utilization in stochastic methods, evolutionary algorithms to approximation of some natural behaviour in natural sciences and engineering or utilization in sensitivity analyses. Therefore, the findings presented in this paper may be relevant in the areas of mathematics, computer science (including artificial intelligence, software development), economy, physics, engineering, and natural sciences as well.

The numbers generated by a pseudorandom number generator (PRNG) must usually fulfil some criteria. These criteria may be simply represented as a set of boundaries of a desired interval (lower and upper limits) and the PRNG can be set to a specific probability distribution. Further parameters of the target probability distribution must be specified in such a case.

Some software applications do not rely on a precise randomness or on the probability distribution of the generated numbers. However, at least a lower fraction does [1]. Therefore, in these applications relying on high-quality PRNGs, generators must be tested. And this paper focuses on one specific subset of the aforesaid group of PRNGs. It focuses on the testing of the PRNG of normal distribution available in the modern standard of the programming language of C++.

DOI: 10.18421/TEM104-01
https://doi.org/10.18421/TEM104-01

Corresponding author: Pavel Tomášek,
Tomas Bata University in Zlín, Czech Republic
Email: tomasek@utb.cz

Received: 23 July 2021.
Revised: 15 September 2021.
Accepted: 22 September 2021.
Published: 26 November2021.

2. Problem Formulation

High quality PRNGs following a desired probability distribution are needed in many software solutions [1], as mentioned in the abstract and in the introduction of this paper.

C++ is a widely spread programming language which includes an implementation of a PRNG with variants of probability distributions of the generated numbers (details below). Its implementation of the generator is being tested in this paper. At the end, this paper concludes whether the numbers generated by the selected generator follow a desired probability distribution.

Subsequent subsections introduce the areas of PRNGs, probability distributions, probability density functions, and C++.

a. Literature overview

The evaluation of the quality of pseudorandom numbers is a difficult problem which has no unique solution [2]. Tests may be aimed at the most fundamental properties. There exists a type of standardized set of statistical tests (presented in [3]), such as the uniformity test [4], the serial test [4], the gap test [5], the maximum t test [3], the collision test [6], the run test and the park test [7], bit level tests [8], the spectral test [3] and visual tests. These tests are well described in the comparative study of many PRNGs in [9], [10].

Another set of experiments may be performed using empirical testing (blind statistical tests with Diehard battery of tests [11], TestU01 library [12]) and NIST statistical test suite [13] and graphical tests (lattice test and space-time diagram test). Results are presented in [14].

None of the research mentioned above analysed the PRNGs using the chi-square test, what is an opportunity for this paper to fill the gap. This approach is described in the section of the problem solution.

b. Pseudorandom Numbers Generator

PRNGs behave efficiently but also deterministically. Therefore, the numbers generated by a PRNG are only pseudorandom in contrary to the true number generators (TRNGs, like it is expected in lotteries). The functioning of PRNGs' algorithms is based on the specific deterministic mathematical model [15].

There are plenty of PRNGs in the history of software engineering. Few representatives are mentioned below [15], [16]:

- LCG,
- Unix Random,
- KnuthB,
- SplitMix,
- Mersenne Twister,
- MRG32k3a,
- Ranlux48,
- MixMax,
- Arc4Random,
- Ran,
- XorWov,
- XorShift.

Further details about the mentioned PRNGs are written, for instance, in [15].

c. Probability distribution

There is a need of various probability distributions of generated numbers in various utilizations of PRNGs. The following continuous probability distributions may belong to the set of mainly known and used [17]:

- Uniform distribution,
- Normal distribution (also known as Gaussian distribution, see Figure 1.),
- Lognormal distribution,
- Student T-distribution.

The normal distribution is probably one of the most used distributions. In many cases, the inspiration comes from the nature and natural behaviour.

Moreover, one of the key characteristics in SI measurements is jitter. It represents variations in the data signal. Typically, these variations behave statistically in nature. Random jitter usually follows a normal distribution [18].

Classical statistical methods using mean and standard deviation of a dataset are usually applied to measurement intercomparisons [19] (assuming the data in the dataset follow a normal distribution).

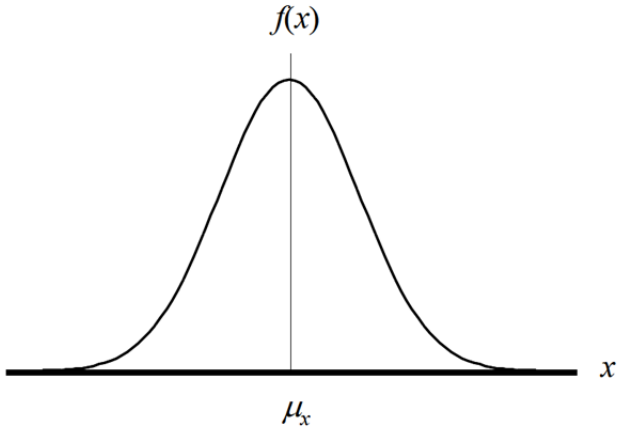

Figure 1. Normal distribution: illustration of the probability density function [18]

d. C++, the selected Programming Language

The reasons for the choice of C++ were the authors' preference, the efficiency and speed of the software written in this programming language, and simple portability between different architectures and operating systems. C++ is a modern, object-oriented and performance-oriented programming language [20].

The name of this language has appeared in 1983 for the first time and despite its age, it is still a modern and preferred language not only in low-level highly effective programming. Its current standard comes from 2017 (usually marked as C++17) and a fresh C++20 is coming soon [20].

Pseudorandom number generator ranlux48 available in C++ (available since a standard called C++11) has been selected. This generator implements all the basic distributions mentioned in the section of 2.c and much more [21].

The histogram of a test case with 10,000 numbers generated by the PRNG of C++ is depicted in Figure 2. The setup of the PRNG was the following:

- normal distribution,
- mean μ = 0.0,
- standard deviation σ = 1.0.

Simplified source code of a program written in C++ generating pseudorandom numbers which should follow the normal distribution (μ = 0.0, σ = 1.0) follows in Table 1.

Table 1. Simplified source code

```
#include <iostream>
#include <random>
#include <time.h>

int main (void) {
  double mean = 0.0;
  double std_dev = 1.0;

  // The desired total count of numbers
  // generated by PRNG
  unsigned total_generated = 100;

  // Counter of iterations
  unsigned iterations = 0;

  // Initialization of the PRNG
  std::ranlux48_base generator(time(0));

  // Configuration of the desired distrib.
  std::normal_distribution<double>
    normal_distrib(mean, std_dev);

  // Getting a set of real numbers from PRNG
  while (iterations < total_generated) {
    std::cout << normal_distrib(generator)
       << std::endl;
    ++iterations;
  }
  return 0;
}
```

3. Problem Solution

This section presents the way of testing of the selected PRNG. A statistical evaluation based on the Pearson's Chi-square goodness of fit test has been performed. This is just one of many possible approaches for testing PRNGs. Another approach like GGRTest or OPERM can be found, for instance, in [22].

Detailed information about the Chi-square, experimental setup and results follow. Selected statistical method primarily leads to a conclusion about the selected PRNG, whether its generated numbers follow a desired distribution.

a. Chi-square

The histogram presented in Figure 2. seems to correspond to the normal distribution (see Figure 1.). However, this hypothesis should be verified. For this purpose, the chi-square (χ^2) test has been applied to the pseudorandomly generated data. This test is used to confirm or reject a null hypothesis. In this case, the null hypothesis relates to the kind of distribution of the input data, whether these data follow the normal distribution. The input data must be transformed into a histogram of a specified number of sections for the purpose of further processing.

χ^2 can be quantified in the following way [23]:

$$\chi^2 = \sum_{j=1}^{k} \frac{(O_j - E_j)^2}{E_j} \qquad (1)$$

where k stands for the number of histogram sections, E_j is the expected count (also known as the *theoretical* value) of the j-th section of the histogram, and O_j is the observed count (also known as the *empirical* value) of the j-th section of the histogram.

The empirical (observed) counts are created from the input data from a selected PRNG (as can be seen in Figure 2.). In contrary, the theoretical (expected) counts are precisely computed using mathematical formulas. In this case, the expected histogram is prepared exactly by the normal distribution's probability density function [24]:

$$f(x, \mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \qquad (2)$$

where $|x| < \infty$, $|\mu| < \infty$ and $\sigma > 0$. A normal distribution is called standard when μ is equal to 0.0 and σ is equal 1.0.

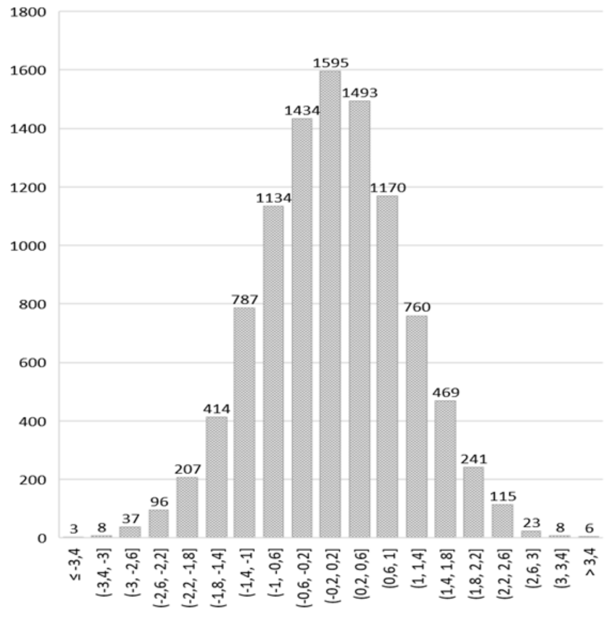

Figure 2. Histogram of 10,000 numbers generated in one run by selected PRNG (normal distribution, mean: 0.0, standard deviation: 1.0)

χ^2 follows the Pearson probability distribution with n degrees of freedom:

$$n = k - r - 1 \tag{3}$$

where r represents the number of parameters of the probability density function. The χ^2 along with the number of degrees of freedom is then used to compute the probability using the Pearson probability distribution F_p:

$$p = F_p\,(\chi^2, n) \tag{4}$$

The probability p is then compared to the significance level α (commonly set to 0.05 [25]). If $p > 1 - \alpha$, then the tested hypothesis is rejected with the significance level α [23].

b. Experiment

This subsection describes the experiment of testing the probability distribution of the selected PRNG.

Working with the observed data generated by the selected PRNG, the null and the corresponding alternative hypotheses have been defined in the following way:

- H_0: The observed data from the selected PRNG are normally distributed.
- H_A: The observed data from the selected PRNG are not normally distributed.

Experimental setup: the following list contains the description of the experimental setup selected for this paper:

- The PRNG has been set up to produce numbers following the normal distribution of desired configuration.
- Histogram is portioned into $k = 20$ segments.
- The significance level in this test has been set to $\alpha = 0.05$ (common threshold [25]).
- The sum of the empirically counted numbers is equal to the sum of the generated numbers assigned to the histogram, what is nearly 100,000 (it is exactly only 99,992, it is not rounded to thousands due to the nature of the normal distribution and the fact that the histogram is segmented into a specific number of segments).

Histograms of the expected data and a test set of observed data generated by the selected PRNG are presented in Table 2. Repeating the process of generation of a new test set of pseudorandom numbers gives little bit different results all the time, but the results are usually similar to the following ones.

Table 2. Histograms of the expected and observed random data (generated in one run by the selected PRNG) of standard normal distribution, nearly 100,000 numbers, 20 segments

Interval	Expected count	Observed count
[-3.60, -3.24]	41	45
[-3.24, -2.88]	133	148
[-2.88, -2.52]	375	392
[-2.52, -2.16]	929	948
[-2.16, -1.80]	2023	2099
[-1.80, -1.44]	3867	3886
[-1.44, -1.08]	6495	6516
[-1.08, -0.72]	9581	9640
[-0.72, -0.36]	12417	12227
[-0.36, 0.00]	14135	14163
[0.00, 0.36]	14135	13953
[0.36, 0.72]	12417	12395
[0.72, 1.08]	9581	9564
[1.08, 1.44]	6495	6553
[1.44, 1.80]	3867	3834
[1.80, 2.16]	2023	2080
[2.16, 2.52]	929	988
[2.52, 2.88]	375	357
[2.88, 3.24]	133	151
[3.24, 3.60]	41	53

Results and discussion: the experiment resulted in the following numbers:

- $n = 17$,
- $\chi^2 = 24.961$,
- $p = 0.929$.

Evaluation of these numbers is the following subsection. H_0 is accepted with the significance level alpha ($p \leq 1 - \alpha$, where $\alpha = 0.05$) with respect to the results presented above.

This test has been repeated and the hypothesis H_0 was accepted in 9 from 10 cases (10 test sets of pseudorandom numbers). Therefore, the numbers generated by the selected PRNG follow the normal distribution.

Further research may aim at the analysis of other kinds of distributions and generator of pseudorandom numbers in newer standards of C++ and other generators.

4. Conclusion

High quality PRNGs generating numbers following desired probability distributions are fundamental in many software applications. Therefore, their testing for quality is important. The work presented in this paper focused on the statistical testing of the probability distribution of numbers generated by the selected PRNG.

The tested distribution in this study was the normal distribution and the selected PRNG was the built-in generator available in C++ since the standard of

C++11. Pearson's Chi-square goodness of fit test was the statistical method applied in the test.

Results presented in the experiment described above indicate that the selected PRNG follows the desired normal distribution. The null hypothesis of the experiment stated that the observed numbers generated by the selected PRNG are normally distributed. This hypothesis has been confirmed.

Acknowledgements

This work has been funded by the project of Applied informatics in population protection (RVO/FLKR/2020/04, RVO excelent/MSM) and by Department of Population Protection of Faculty of Logistics and Crisis Management at Tomas Bata University in Zlín.

References

[1]. James, F., & Moneta, L. (2020). Review of high-quality random number generators. *Computing and Software for Big Science*, *4*(1), 1-12.

[2]. Janke, W. (2002). Pseudo random numbers: Generation and quality checks. *Lecture Notes John von Neumann Institute for Computing*, *10*, 447.

[3]. Knuth, D. E. (2014). *Art of computer programming, volume 2: Seminumerical algorithms*. Addison-Wesley Professional.

[4]. L'Ecuyer, P., Simard, R., & Wegenkittl, S. (2002). Sparse serial tests of uniformity for random number generators. *SIAM Journal on scientific computing*, *24*(2), 652-668.

[5]. Jhajharia, S., Mishra, S., & Bali, S. (2013, August). Public key cryptography using neural networks and genetic algorithms. In *2013 Sixth International Conference on Contemporary Computing (IC3)* (pp. 137-142). IEEE.

[6]. Capó, E. J. M., Cuellar, O. J., Pérez, C. M. L., & Gómez, G. S. (2016, October). Evaluation of input—Output statistical dependence PRNGs by SAC. In 2016 International Conference on Software Process Improvement (CIMPS) (pp. 1-6). IEEE.

[7]. Min, L., Chen, T., & Zang, H. (2013). Analysis of fips 140-2 test and chaos-based pseudorandom number generator. *Chaotic Modeling and Simulation*, *2*(1), 273-280.

[8]. Inayah, K., Sukmono, B. E., Purwoko, R., & Indarjani, S. (2013, November). Insertion attack effects on standard PRNGs ANSI X9. 17 and ANSI X9. 31 based on statistical distance tests and entropy difference tests. In *2013 International Conference on Computer, Control, Informatics and Its Applications (IC3INA)* (pp. 219-224). IEEE.

[9]. Vattulainen, I., Kankaala, K., Saarinen, J., & Ala-Nissila, T. (1995). A comparative study of some pseudorandom number generators. *Computer Physics Communications*, *86*(3), 209-226.

[10]. Vattulainen, I. (1994). New tests of random numbers for simulations in physical systems. *arXiv preprint cond-mat/9411062*.

[11]. Brown, R. G. (2004). Engineering a beowulf-style compute cluster. *Duke University Physics Department*. Retrieved from: https://webhome.phy.duke.edu/~rgb/General/dieharder.php [accessed: 10 June 2021].

[12]. L'ecuyer, P., & Simard, R. (2007). TestU01: AC library for empirical testing of random number generators. *ACM Transactions on Mathematical Software (TOMS)*, *33*(4), 1-40.

[13]. Rukhin, A., Soto, J., Nechvatal, J., Smid, M., & Barker, E. (2001). *A statistical test suite for random and pseudorandom number generators for cryptographic applications*. Booz-allen and hamilton inc mclean va.

[14]. Bhattacharjee, K., Maity, K., & Das, S. (2018). A search for good pseudo-random number generators: Survey and empirical studies. *arXiv preprint arXiv:1811.04035*.

[15]. O'Neill, M. E. (2014). PCG: A family of simple fast space-efficient statistically good algorithms for random number generation. *ACM Transactions on Mathematical Software*.

[16]. Steele Jr, G. L., Lea, D., & Flood, C. H. (2014). Fast splittable pseudorandom number generators. *ACM SIGPLAN Notices*, *49*(10), 453-472.

[17]. Castrup, H. (2001, May). Distributions for uncertainty analysis. In *Proc. Int. Dimensional Workshop* (pp. 1-12).

[18]. Bonaguide, G., & Jarvis, N. (2019). *The VNA Applications Handbook*. Artech House.

[19]. Judish, R. M., & Splett, J. (1999, May). Robust statistical analysis of vector network analyzer intercomparisons. In *IMTC/99. Proceedings of the 16th IEEE Instrumentation and Measurement Technology Conference (Cat. No. 99CH36309)* (Vol. 3, pp. 1320-1324). IEEE.

[20]. Stroustrup, B. (2018). *A Tour of C++*. Addison-Wesley Professional.

[21]. The C++ Resource Network (2021). <random>. *The C++ Resources Network*. Retrieved from: http://www.cplusplus.com/reference/random/ [accessed: 15 June 2021].

[22]. Lorek, P., Słowik, M., & Zagórski, F. (2017, November). Statistical Testing of PRNG: Generalized Gambler's Ruin Problem. In *International Conference on Mathematical Aspects of Computer and Information Sciences* (pp. 425-437). Springer, Cham.

[23]. Rao, C. R. (2002). Karl Pearson chi-square test the dawn of statistical inference. In *Goodness-of-fit tests and model validity* (pp. 9-24). Birkhäuser, Boston, MA.

[24]. Patel, J. K., & Read, C. B. (1996). *Handbook of the normal distribution* (Vol. 150). CRC Press.

[25]. Delorme A (2006) Statistical methods. In: Webster JG (ed) Encyclopedia of medical device and instrumentation, vol 6. Wiley Interscience, Hoboken, NJ, USA, pp 240–264.