# The Mersenne Twister Output Stream Postprocessing

Yurii Shcherbyna [1], Nadiia Kazakova[2], Oleksii Fraze-Frazenko[3]

[1] *National University "Odesa Law Academy", Rishelievska st., 28, Odesa, 65011, Ukraine*
[2,3] *Odesa State Environmental University, 15 Lvivska str., Odesa, 65016, Ukraine*

**Abstract**

Today, pseudo-random number sequence generators are actively used to solve a large number of applied problems of statistical and simulation modeling in such areas as telecommunications networks, automated control systems for production processes and infrastructure, security systems and others. Such generators have serious requirements for the sequence of numbers that they generate at their outputs. These are, first of all, the requirements for their randomness. The original sequences should be almost indistinguishable from the truly random ones. And, most importantly, they must also ensure a high uniformity of probability distribution of the original numbers. It is shown that the non-uniformity of numbers at the output of the primary generator significantly affects the quality of modeling of stochastic processes that take place in systems for which computer models are built. Tests on a linear congruent generator and a Mersenne twister (MT) generator have shown that the flow of decimal real numbers at their outputs does not fully meet the needs of modern computer modeling. The vast majority of tests of such flows using the Pearson chi-square test gives an unsatisfactory result. Based on the analysis of post-processing methods of numerical sequences, it is proposed to perform preliminary thinning of the input in relation to the model of the numerical flow by removing elements that do not fit into the uniform distribution. The expected sum of random real numbers to be included in each of the segments of the random number distribution histogram is chosen as the thinning criterion. It is shown that the use of this method of post-processing of the primary generator does not require extra computing resources of the system.

**Keywords**

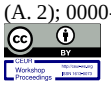
Simulation, linear congruent generator, Mersenne twister generator, inverse function method, Pearson chi-square test, post-processing of numerical flow.

## 1. Introduction

If in the second half of the last century modeling was considered a secondary stage in the design of complex systems, today the modern development of computer technology significantly increases its importance in the study of stochastic processes that occur in modern production, infrastructure management and economic activity.

Usually the modeling of random processes takes place in two stages: first a sequence of random variables evenly distributed on the interval [0, 1] is created, and only then a sequence of numbers is formed from them, which corresponds to the given probability distribution law. Because computing devices are deterministic automata, they can only output pseudo-random numbers (PRN) with a limited repetition period of $T$. For efficient modeling, PRN generation algorithms must provide high speed, long repetition periods, and good statistics. [1].

Libraries of modern programming languages already contain PRN generators with a uniform distribution law, which return the number $U_i$ from

*III International Scientific And Practical Conference "Information Security And Information Technologies", September 13–19, 2021, Odesa, Ukraine*
EMAIL: shcherbinayura53@gmail.com (A. 1); kaz2003@ukr.net (A. 2); frazenko@gmail.com (A. 3)
ORCID: 0000-0003-3885-6747 (A. 1); 0000-0003-3968-4094 (A. 2); 0000-0002-2288-8253 (A. 3)


the finite set $\{0, 1, \dots, T-1\}$. It is also possible to connect external libraries offered by different developers. Most traditional PRN generators are well described by Donald Knuth in [2], where he concludes that they are of insufficient quality and unsuitable for research needs.

The vast majority of PRN cryptographic generators developed in recent decades have been described in detail by Bruce Schneier in [3], but they are hardly suitable for computer simulation. First, their use requires significant computing resources, which significantly reduces their efficiency, and secondly, they ensure uniform distribution at the binary level. As shown in [4], the transformation of a binary sequence into a decimal format and its subsequent scaling leads to a significant loss of uniformity.

Recently, an algorithm known as the Mersenne Twister (MT) has been proposed for modeling purposes, which provides an extremely long repetition period $2^{19937} - 1$ [5]. It, together with the linear congruent generator (LCG) [6], is part of the libraries of almost all known specialized software environments designed to solve research and engineering problems.

The two-stage modeling scheme is very sensitive to the uniform distribution of numbers at the output of the selected generator. As shown by checking the flow of real numbers generated by LCG and MT using Pearson's $\chi^2$-test, up to half of the samples, regardless of their size, are not tested for uniformity.

Since the choice of PRN generator is extremely limited for researchers, this problem should be solved by upgrading the numerical flow at the output of the PRN generator.

## 2. The aim of the study

To check the quality of pseudo-random number generators, a large number of test packets were created [7,8] and all of them perform the analysis of the output stream at the binary level. This is due to the fact that they are mainly intended for testing cryptographic generators focused on the performance of quenching operations, which are performed bit by bit. Divided into bytes and converted to a decimal sequence of real numbers, a binary sequence does not necessarily remain evenly distributed. In most cases, it is necessary to perform its post processing [9], choosing a method that would give a satisfactory simulation result and, at the same time, was effective in terms of the use of computing resources. In view of this, the aim of the study is to select and justify an additional method of converting a sequence of pseudo-random numbers at the output of the MT generator to ensure a given level of uniformity of their distribution.

## 3. Methods of post-processing

The general idea of additional processing of numbers at the output of the generator was formulated long ago, when the main source of random numbers were physical noise occurring in electronic devices, such as electronic lamps, quantum generators and the like. Its essence is to sacrifice a certain number of numbers at the output of the generator for the sake of obtaining an output stream that would satisfy the conditions. Later, von Neumann remarked on the inadmissibility of using physical generators in computer technology, because due to technical difficulties the possibility of re-implementing the obtained sample of random numbers at that time was absent and, therefore, proposed algorithms for generating pseudo-random numbers as the method of mean squares [10 ] and the linear congruent method. But, as shown by D. Knuth [4], they also did not provide the necessary uniformity of the formed numerical flow. Since it is almost impossible to make an ideal generator, the idea of post-processing for both real random number generators and PRN generators remains relevant.

At the moment, we can identify the following four methods of post-processing: [9].

1. Ad hoc simple correctors.
2. Whitening with hash functions.
3. Extractor algorithms.
4. Resilient functions.

The general requirement for all methods of post-processing is the minimization of resources for their implementation.

An example of a simple corrector is the corrector described by von Neumann in [10] where he proposes to combine a pair of bits obtained from independent sources on the principle: if the bits match (00 or 11), the bits are canceled, the combination of bits 01 corresponds to 0-th the output bit, and the combination 10 corresponds to the 1st output bit. The maximum efficiency of such an algorithm is on average 4 input bits per 1 output bit. It was in this work that von Neumann emphasized the difficulty of generating random decimal numbers.

Later, other, more advanced versions of similar correctors were proposed, but they also work at the bit level.

Whitening is a method that reduces the correlation of symbols at the output of the entropy source and increases the homogeneity and uniformity of the distribution of symbols in the output stream. It is usually performed using hash algorithms, such as MD5, SHA-1, SHA-2, SHA-256 or SHA-512. This processing is a deterministic algorithm that converts input blocks of characters of arbitrary length into a fixed size string. In [11] it was shown that bleaching does not increase entropy and therefore the main task of ensuring randomness should be solved by the PRN generator, and not by the post-processing algorithm. It should be noted that the term randomness means the absence of a noticeable analytical relationship between the symbols at the output of the PRN generator. But, in contrast to cryptographic problems, in modeling it is important to ensure the uniformity of the distribution of the original numerical flow.

Randomity extractors are algorithms that convert a low-quality stream of input values into an almost uniform stream of numeric characters with a small number of guaranteed random bits. Formally, the method of such a transformation is described in the work of Luke Trevisan [12]. To characterize weak sources of chance, the author introduces the concept of minimum entropy, which characterizes the non-uniform distribution of the quantity $X$ in the range $\{0,1\}^n$, where $n$ is a binary combination at the source output. In the case of a perfectly uniform distribution, all combinations will be equally probable and the entropy will be maximum, otherwise it will be smaller. If the minimum entropy of such a source has a value of at least $k$, then for each $x \in \{0,1\}^n$ the condition $\Pr[X = x] \leq 2^{-k}$ is fulfilled. The task of the extractor is to convert the flow $X$ into almost uniform. To quantify the output flow, the concept of statistical difference $\epsilon$ between two random variables $X$ and $X$ in the range $\{0,1\}^n$ is used, which is defined as:

$$|p[T(X) = 1]| - |p[T(Y) = 1]| \leq \epsilon \qquad (1)$$

In the general case, the $(k, \epsilon)$ -extractor converts the flow of random variables $X$ into an almost uniform flow by the rule:

$$Ext : \{0,1\}^n \times \{0,1\}^t \to \{0,1\}^m, \qquad (2)$$

where the random variable $X$ has a minimum entropy $k$, and $U_t$ is a uniformly distributed quantity on $\{0,1\}^t$. The mechanism of operation of the randomness extractor is shown in Figure 1.

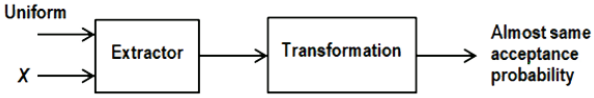



**Figure 1**: The mechanism of the extractor

In [12], another method of amplifying the randomness of the output flux of the PRN generator, which is based on its postfiltration through some deterministic process, is considered. His idea is to use the use of elastic functions to divide the original characters into random and "not random enough". In [13], the elastic function $F$ is defined as the $(n, m, k)$ -function $f : F^n \to F_m$, which forms each output $k$-bit combination of fixed input bits directly, and the others $n - k$ bits are selected randomly. Such functions were created exclusively for the needs of cryptographic transformations.

From the above we can conclude that the work on creating generators of random and pseudo-random numbers is mainly focused on cryptographic needs. At the heart of such generators is a complex computational process, the implementation of which requires significant computing resources. For modeling purposes, either LCG or MT generators are typically used, which have unsatisfactory uniformity in the distribution of the source symbols, but can be subject to post-processing methods such as combining streams from multiple sources and thinning them by removing symbols that look like " not random enough ".

## 4. Post-processing of a numerical stream from the MT generator

Computer simulation of random processes such as request flows in telecommunication systems [1], flows of attacks on information system resources [14], or failures of technical equipment in computer systems, involves the use of procedures containing elements of randomness implemented using built based on number theory and numerical analysis of optimally selected deterministic systems. Such systems are understood as arithmetic generators of pseudo-random numbers, which are based on recurrent relations. This means that each subsequent number at the output of the generator is

determined by one or more pre-formed numbers and the flow of such numbers will be repeated regularly with period $T$. Despite this dependence, the numbers generated by the generator should look independent throughout the period. only in the case of their absolutely uniform distribution. Such numbers, evenly distributed on the interval $[0,\ 1]$, are most often used for modeling purposes. They must meet the following requirements:

1. the sequence must have the properties of uniform distribution of random numbers in the interval $[0,\ 1]$ throughout the repetition period $T$;
2. each fragment of the sequence within the period $T$, from the output of the generator must have the properties of uniform distribution.

The first condition is not met by the definition of PRN, but this shortcoming developers are trying to compensate by creating algorithms for generating numbers with too long a repetition period. Both LCG and MT generators have fairly long periods. The problem for them is the need to initialize them with a real random number, but it is quite simply solved by forming such a number from the current time.

The second condition can be formally described as follows. The general sequence $x_1,\ x_2, \ldots$ can be considered completely uniformly distribute (CUD), if for any $s \geq 1$ part of this sequence $(x_n, x_{n+1}, \ldots, x_{n+s+1})\quad n = 1, 2, \ldots$ will also be evenly distributed.

In [6] it was shown that the LCG developers tried to provide satisfactory generator characteristics with the optimal ratio of the coefficients $a,\ c,\ m$ of the recurrent ratio

$$X_{n+1} = (aX_n + c)\ mod\ m. \qquad (3)$$

As practice shows, the function **rand()** is built into most modern programming environments and uses as a module $m$ 32-bit machine bit word, which provides a period of repetition of numbers T at the output of the generator, which does not exceed the value of $m = 2^{32}$. As for the uniformity of the distribution of the output stream, it remains extremely low.

Figure 2 shows an example of a histogram of the distribution of numbers at the output of the LCG, obtained using the function **rand()**, which is part of the library C ++. The value of the sample $N$ is 1024 numbers, and the number of intervals of the histogram is 16.

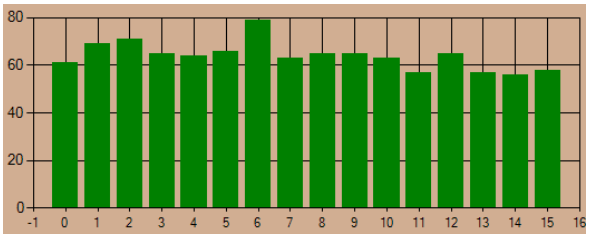



**Figure 2**: Histogram of the distribution of PRN obtained by the function **rand()**

A check of the quality of the distribution using the $\chi^2$ −test showed that more than two-thirds of the samples do not meet the requirements of uniformity. Figure 3 shows the distribution of the number of hits in each of the 16 intervals of the histogram, Figure 2. Here are the limits of the intervals of the histogram (*Xmin, Xmax*), the probability of hitting the number in the interval (*Pi*), the number of hits in the interval (*Ni*) and components indicator $\chi^2$ −-test (*Hi*), for each specific interval of the histogram.

| N | Xmin | Xmax | Ni | Hi |
|---|---|---|---|---|
| 1 | 0 | 0,0625 | 61 | 0,1406 |
| 2 | 0,0625 | 0,125 | 69 | 0,3906 |
| 3 | 0,125 | 0,1875 | 71 | 0,7656 |
| 4 | 0,1875 | 0,25 | 65 | 0,0156 |
| 5 | 0,25 | 0,3125 | 64 | 0 |
| 6 | 0,3125 | 0,375 | 66 | 0,0625 |
| 7 | 0,375 | 0,4375 | 79 | 3,5156 |
| 8 | 0,4375 | 0,5 | 63 | 0,0156 |
| 9 | 0,5 | 0,5625 | 65 | 0,0156 |
| 10 | 0,5625 | 0,625 | 65 | 0,0156 |
| 11 | 0,625 | 0,6875 | 63 | 0,0156 |
| 12 | 0,6875 | 0,75 | 57 | 0,7656 |
| 13 | 0,75 | 0,8125 | 65 | 0,0156 |
| 14 | 0,8125 | 0,875 | 57 | 0,7656 |
| 15 | 0,875 | 0,9375 | 56 | 1 |
| 16 | 0,9375 | 1,0625 | 58 | 0,5625 |

**Figure 3**: Boundaries of histogram intervals and distribution of sample numbers between them

The total quality index according to Pearson's $\chi^2$ −-test is calculated by the formula

$$\chi^2 = \sum_{i=1}^{k} \frac{(n_i - N_i^*)^2}{N_i^*}, \qquad (4)$$

where $k$ – this is the number of segments of the histogram, $n_i$ and $N_i^*$ – the number of random numbers of the output stream that actually fell in the i-th interval and their expected number, respectively. The expected number with uniform

distribution $N_i^*$ is defined as $N/k$ and for the given example is 64.

Similar tests were performed for the MT generator. Unlike the LCG generator, it has a much longer repetition period, which is equal to $T = 2^{19937} - 1$ bits, and the algorithm embedded in it provides very little correlation between two samples from the original sequence of numbers. The developers of the generator claim that it passes the tests of the DIEHARD package [8]. However, all the declared positive qualities of such a generator are valid for a binary sequence. Tests of real numbers distributed in the interval $[0,\ 1]$ using the $\chi^2$ –test showed that only $10 \div 15$ percent of samples from the output stream from the MT generator give a positive test result.

Figure 4 shows an example of a histogram of the distribution of numbers at the output of MT, obtained using the function **uniform_real_distribution<> mersenne(0, 1)** from the library C ++ <cmath.h>

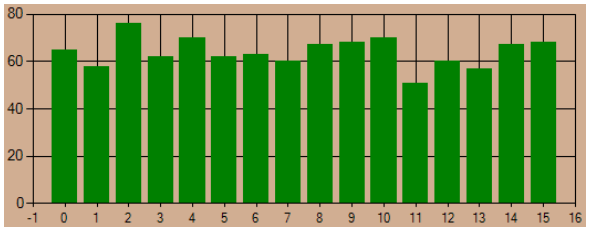



**Figure 4**: Histogram of the PRN distribution obtained using the function **uniform_real_distribution<> mersenne(0, 1)**

The sample size *N*, as in the case of the LCG generator, was equal to 1024 real decimal numbers, and the number of intervals of the histogram k was also equal to 16.

To model stochastic processes, the method is most often used, the essence of which is that on the basis of the conditions of inverse functions and the theorem according to which a continuous random variable $x$, with an arbitrary distribution having a probability distribution function $F(x)$, determines a continuous uniformly distributed on the interval $[0,\ 1]$, the random variable $\gamma = F^{-1}(\gamma)$ [15]. This method works well when the process can be described analytically and the inverse function $F^{-1}(x)$ exists for it.

To evaluate the numbers distribution uniformity influence at the output of the MT generator on the quality of the simulation, consider the example of creating a numerical flow, which is described by the Weibull function with two parameters. It looks like this:

$$F(x, \alpha, \beta) = 1 - e^{-(x/\beta)^{\alpha}}. \quad (5)$$

where $\alpha$ – is a scale parameter, $\beta$ – form parameter, a $x$ – variable. $\alpha$ і $\beta$ – are fixed values for which the conditions $\alpha > 0,\ \beta > 0$ must be met. In case if $\beta = 1$, Weibull's distribution coincides with the exponential distribution.

The inverse function looks like:

$$F^{-1}(x) = \beta[-\ln(1 - x)]^{1/\alpha}. \quad (6)$$

We assume that at the output of the MT generator there is a flow of numbers $y_1,\ y_2, \ldots$ which express the probability of events of the random Weibull process. Then, using relation (6), we can obtain a stream of numbers $x_1,\ x_2, \ldots$, which in accordance with this principle, is determined by relation (5) and expresses the argument of the distribution function.

Next, we construct a histogram, on the basis of which we calculate the quality index by the $\chi^2$-test. The choice of this criterion is determined by the fact that, firstly, its use is not limited to the type of distribution and, secondly, if this criterion is not met, then all other criteria, too, are unlikely to be met.

A separate issue in the special literature is the choice of the number of segments of the histogram. They should be sufficient so that the shape of the histogram in its form as close as possible to the form of the Weibull distribution density function described by the expression:

$$p(x) = \frac{\alpha}{\beta^{\alpha}} x^{\alpha-1} e^{-(x/\beta)^{\alpha}}. \quad (7)$$

On the other hand, the number of segments should not be too large so as not to lose the filtering capabilities of the histogram, as is the case with signal quantization. Today there are several different ways to determine their number and the most popular is the formula proposed in 1926 by Sturges [16]

$$k = 1 + [log_2 N] \quad (8)$$

where $k$ number of histogram segments, and $N$ – the number of characters in the random number sample. This formula is derived from the binomial distribution and implicitly assumes work with the normal distribution.

There are other formulas that also allow you to determine the approximate number of segments. A good option is the formula proposed in 1981 by Freedman and Diaconis [17], which gives the length of the segment $h$, expressed in terms of

interquartile range (the distance between the end of the first and the beginning of the last quartile of the *IQ* sample)

$$h = 2 \cdot (IQ) \cdot N^{-1/3}, \quad (9)$$

where the number of segments can be defined as

$$k = \frac{y_{max} - y_{min}}{h}. \quad (10)$$

where $y_{max}$ and $y_{min}$ are, respectively, the maximum and minimum values of the members of the sample variation series.

All the proposed methods for calculating the number of segments of the histogram were determined based on the problem of finding the type of distribution based on the accumulated statistical material and, each time, the researchers proceeded from the features of the process to be evaluated. That is why there are so many ways to determine the value of *k*. As for the simulation, the inverse problem is solved here, when the method of distribution of random variables is known and, therefore, the value of the number of segments is not so critical and can be determined arbitrarily.

If the inverse function method converts a sequence of random numbers from the MT generator into a random Weibull process with the parameters $\alpha = 1.3$, $\beta = 0.1$, it will look like Figure 5.

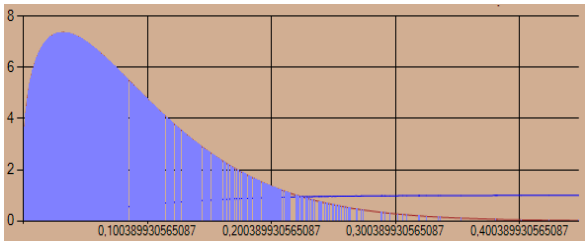



**Figure 5**: The result of modeling a random Weibull process by the inverse function method

An additional problem that arises when estimating an asymmetric random process is that, taking into account the peculiarities of formula (4), very few random numbers $n_i$ fall into the last segments of the "tails" of the distribution and, therefore, these components of the indicator $\chi^2$-test contributes the lion's share to the error, which brings its value closer to the critical value $\chi^2_{кр}$. In [2] D. Knuth points out that the sample size *N* must ensure that each interval of the histogram hits at least 5 random numbers. To avoid this problem, and since the histogram segments do not have to be the same size, we will combine the last intervals with less than 6 numbers into one common interval. For the example shown in Figure 5, 16 segments of the histogram will be filled as follows

The table in Figure 6 shows the boundaries of the intervals of the histogram (*Xmin, Xmax*), the probability of hitting the number in the interval (*Pi*), the number of hits in the interval (*ni*), the expected number of hits in the interval (*Ni*) and the component indicator $\chi^2$-test (*Hi*).

| N | Xmin | Xmax | Pi | ni | Ni | Hi |
|---|---|---|---|---|---|---|
| 1 | 0 | 0,0331 | 0,211148 | 193 | 216 | 2,449074 |
| 2 | 0,0331 | 0,0661 | 0,231187 | 248 | 237 | 0,510549 |
| 3 | 0,0661 | 0,0992 | 0,185829 | 196 | 190 | 0,189474 |
| 4 | 0,0992 | 0,1322 | 0,134426 | 135 | 138 | 0,065217 |
| 5 | 0,1322 | 0,1653 | 0,091082 | 83 | 93 | 1,075269 |
| 6 | 0,1653 | 0,1984 | 0,058813 | 62 | 60 | 0,066667 |
| 7 | 0,1984 | 0,2314 | 0,036541 | 38 | 37 | 0,027027 |
| 8 | 0,2314 | 0,2645 | 0,02198 | 31 | 23 | 2,782609 |
| 9 | 0,2645 | 0,2975 | 0,012855 | 11 | 13 | 0,307692 |
| 10 | 0,2975 | 0,3306 | 0,007332 | 17 | 8 | 10,125 |
| 11 | 0,3306 | 0,3636 | 0,004089 | 3 | 4 | 0,25 |
| 12 | 0,3636 | 0,3967 | 0,002234 | 2 | 2 | 0 |
| 13 | 0,3967 | 0,4298 | 0,001197 | 1 | 1 | 0 |
| 14 | 0,4298 | 0,4628 | 0,00063 | 1 | 1 | 0 |
| 15 | 0,4628 | 0,4959 | 0,000326 | 0 | 0 | не число |
| 16 | 0,4959 | 0,562 | 0,000166 | 2 | 0 | ∞ |

**Figure 6**: Boundaries of histogram intervals and their filling for Weibull distribution

From the table shown in Figure 6, it is seen that the segments 12 to 16 do not allow to calculate the corresponding components of the indicator $\chi^2$, and therefore they are combined into one interval, for which $n_i = 9$, and $n_i^* = 8$. At the level of five percent error $(\lambda = 0.05)$ for the given example, its value is $\chi^2 = 17.723577$, which is more than the critical value $\chi^2_{кр} = 7.290644$, and this means dissatisfaction with the simulation result. This result is confirmed in the vast majority of subsequent tests and, thus, it can be concluded that it is necessary to correct the numerical flux at the output of the MT generator by post-processing.

As can be seen from the above analysis of post-processing methods, the vast majority of them were developed for cryptography and, therefore, are unacceptable for the correction of the numerical flow at the output of the MT generator due to their excessive complexity. The solution to the problem should not burden the computer system with significant additional resources.

Given the admissibility of such operations as combining numerical streams, their "bleaching" or "sieving", as well as the use of Pearson's $\chi^2$-test to assess the uniformity of number distribution,

we will try to "align" it by removing from its composition "extra" elements .

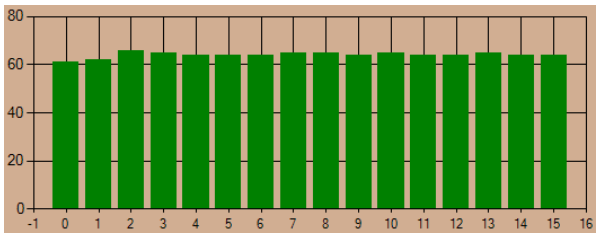



**Figure 7**: Histogram of the PRN distribution obtained at the output of the MT generator after post-processing

It is expected that in the case of uniform distribution in each segment of the histogram should fall the same number of random numbers equal to $N_i^* = N/k$. The mathematical expectation of a quantity to be included in a segment bounded by the conditions $x_{min} \le x_i < x_{max}$ is defined as $m_i = (x_{min} + x_{max})/2$. Then the number of numbers that fall into the $i$-th segment $S_i$, will be approximately equal to the value of $S_i^* = N_i^* m_i$. Of course, each time this sum will be either less than $S_i^*$, or more than $S_i^*$, but there will be no significant difference. This makes it possible to formulate such an algorithm. If the sum of the numbers $S_i$ that fall into the i-th segment of the histogram exceeds the value of $S_i^*$, then all other numbers that fall into it are skipped. Of course, the number of numbers in the segments will remain different, but the unevenness of the sample will be smaller.

| N | Xmin | Xmax | Ni | Hi |
|---|---|---|---|---|
| 1 | 0 | 0,0625 | 61 | 0,1406 |
| 2 | 0,0625 | 0,125 | 62 | 0,0625 |
| 3 | 0,125 | 0,1875 | 66 | 0,0625 |
| 4 | 0,1875 | 0,25 | 65 | 0,0156 |
| 5 | 0,25 | 0,3125 | 64 | 0 |
| 6 | 0,3125 | 0,375 | 64 | 0 |
| 7 | 0,375 | 0,4375 | 64 | 0 |
| 8 | 0,4375 | 0,5 | 65 | 0,0156 |
| 9 | 0,5 | 0,5625 | 65 | 0,0156 |
| 10 | 0,5625 | 0,625 | 64 | 0 |
| 11 | 0,625 | 0,6875 | 65 | 0,0156 |
| 12 | 0,6875 | 0,75 | 64 | 0 |
| 13 | 0,75 | 0,8125 | 64 | 0 |
| 14 | 0,8125 | 0,875 | 65 | 0,0156 |
| 15 | 0,875 | 0,9375 | 64 | 0 |
| 16 | 0,9375 | 1 | 64 | 0 |

**Figure 8**: Boundaries of histogram intervals and their filling with numbers from the output of the MT generator after post-processing

Figure 7 shows the test results of a sample of length $N = 1024$ real decimal pseudo-random numbers at the output of the MT generator after post-processing as described, and the filling of histogram segments is shown in Figure 8 Now, for $\lambda = 0.05$ $\chi^2 = 0.3438$, which is less than critical value $\chi^2_{\text{кр}} = 7.2609$..

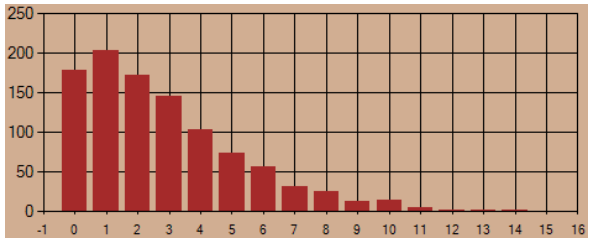



**Figure 9**: Histogram of the Weibull process, based on the table in Figure 8

The Weibull process formed by the method of the inverse function of the numbers from the MT generator after their post-processing gives significantly better results. The histogram constructed on the basis of such flow is shown in figure 9.

The table in Figure 10 shows the distribution of numbers in the segments of the histogram shown in Figure 9 and the components of the $\chi^2$-test.

| N | Xmin | Xmax | Pi | ni | Ni | Hi |
|---|---|---|---|---|---|---|
| 1 | 0 | 0,0275 | 0,17037 | 178 | 176 | 0,022727 |
| 2 | 0,0275 | 0,055 | 0,19828 | 203 | 204 | 0,004902 |
| 3 | 0,055 | 0,0825 | 0,172518 | 172 | 178 | 0,202247 |
| 4 | 0,0825 | 0,11 | 0,136571 | 146 | 141 | 0,177305 |
| 5 | 0,11 | 0,1375 | 0,102121 | 104 | 105 | 0,009524 |
| 6 | 0,1375 | 0,1651 | 0,073283 | 74 | 76 | 0,052632 |
| 7 | 0,1651 | 0,1926 | 0,050908 | 57 | 52 | 0,480769 |
| 8 | 0,1926 | 0,2201 | 0,034422 | 32 | 35 | 0,257143 |
| 9 | 0,2201 | 0,2476 | 0,022739 | 26 | 23 | 0,391304 |
| 10 | 0,2476 | 0,2751 | 0,014715 | 13 | 15 | 0,266667 |
| 11 | 0,2751 | 0,3026 | 0,009348 | 14 | 10 | 1,6 |
| 12 | 0,3026 | 0,3301 | 0,00584 | 5 | 6 | 0,166667 |
| 13 | 0,3301 | 0,3576 | 0,003592 | 2 | 4 | 1 |
| 14 | 0,3576 | 0,3851 | 0,002178 | 2 | 2 | 0 |
| 15 | 0,3851 | 0,4126 | 0,001303 | 2 | 1 | 1 |
| 16 | 0,4126 | 0,4677 | 0,000769 | 0 | 1 | 1 |

**Figure 10**: Boundaries of histogram intervals and distribution of numbers between them for Weibull distribution after post-processing

For the given example for $\lambda = 0.05$ , $\chi^2 = 4.10807$, which is less than the critical value $\chi^2_{\text{кр}} = 7.260944$.

Subsequent tests showed that the use of the proposed method of "thinning" the input stream from the MT generator, gives significantly better simulation results in terms of their reliability.

## 5. Conclusions

The experience of modern computer modeling shows that the use of specialized software packages such as Boost, Glib, C ++, Python, Ruby, R, PHP, MATLAB and Autoit requires significant computing resources and therefore the simulation of stochastic processes is better performed using common tools programming in languages that allow you to create economical program code. Modern C ++ programming environments, such as Visual Studio and QT5, are a good option. They include a large number of additional libraries containing various PRN generators.

Such generators are built on the basis of recurrent algorithms and do not provide a given level of uniformity of distribution of real numbers in the output stream and, therefore, their use for modeling significantly affects its quality.

The problem of improving the quality of modeling can be solved by supplementing the algorithm for calculating operations that provide pre-randomization of the input stream. As such operations, you can use the removal of the original numbers, the presence of which violates the uniformity of the distribution of the primary generator. One way to implement such an algorithm is to limit the number of characters in each segment of the histogram by the value of the expected sum of random numbers, which is determined by the mathematical expectation of the number in the segment and the expected number of numbers in the segment. Tests show that the unevenness of the primary generator with this method of post-processing has almost no effect on the quality of modeling.

## 6. References


[1] Averill M. Law. Simulation modeling and analysis, 5th. ed., McGraw-Hill Education, 2 Penn Plaza, New York, 2015. URL: https://industri.fatek.unpatti.ac.id/wp-content/uploads/2019/03/108-Simulation-Modeling-and-Analysis-Averill-M.-Law-Edisi-5-2014.pdf

[2] D. E. Knuth, The Art of Computer Programming, Volume 2: Seminumerical Algorithms, 3rd. ed., Boston, Mass, USA : Addison-Wesley, Longman Publishing, Addison-Wesley, Reading, Mass, 1998. URL: https://doc.lagout.org/science/0_Computer%20Science/2_Algorithms/The%20Art%20of%20Computer%20Programming%20%28vol.%202_%20Seminumerical%20Algorithms%29%20%283rd%20ed.%29%20%5BKnuth%201997-11-14%5D.pdf

[3] Bruce Schneier, Applied Cryptography, Second Edition: Protocols, Algorthms, and Source Code in C, 20. ed., Boston, Mass, USA : Addison-Wesley, Longman Publishing, Addison-Wesley, Reading, Mass, 1998. doi:10.1002/9781119183471 URL: https://lost-contact.mit.edu/afs/adrake.org/usr/rkh/Books/books/Schneier%20-%20Applied%20Cryptography%202ed%20-%20Wiley.pdf.

[4] J. H. Ahrens, U. Dieter, A. Grube, Pseudo-random numbers. Computing 6 (1970) 121-138). URL: https://doi.org/10.1007/BF02241740.

[5] Saito, M. An Application of Finite Field: Design and Implementation of 128-bit Instruction-Based Fast Pseudorandom Number Generator. Dept. of Math. Graduate School of Science, February 9th, 2007. URL: http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/SFMT/M062821.pdf.

[6] Niederreiter H. Quasi-Monte Carlo methods and pseudo-random number. doi: https://doi.org/10.1090/S0002-9904-1978-14532-7. URL: https://www.ams.org/journals/bull/1978-84-06/S0002-9904-1978-14532-7/S0002-9904-1978-14532-7.pdf.

[7] A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. SP 800-22 Rev. 1a, April 2010. URL: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-22r1a.pdf.

[8] The Marsaglia Random Number CDROM including the Diehard Battery of Tests of Randomness, 1995. URL: http://ftpmirror.your.org/pub/misc/diehard/.

[9] Mario Stipčević, True Random Number Generators. Open Problems in Mathematics

and Computational Science, Open Problems in Mathematics and Computational Science 275-315) doi:10.1007/978-3-319-10683-0_12 URL: https://www.researchgate.net/publication/299824248_True_Random_Number_Generators.

[10] J. von Neumann. Various techniques for use in connection with random digits. Applied Math Series, Notes by G. E. Forsythe, in National Bureau of Standards, Vol. 12, 36 – 38, 1951. URL: https://mcnp.lanl.gov/pdf_files/nbs_vonneumann.pdf.

[11] Sunar, B. Martin, W. J. Stinson, D. R. A Provably Secure True Random Number Generator with Built-in Tolerance to Active Attacks doi:10.1109/TC.2007.250627 URL: https://cs.uwaterloo.ca/~dstinson/papers/rng-IEEE.pdf.

[12] Trevisan L. Extractors and Pseudorandom Generators 1999. Journal of the ACM URL: http://theory.stanford.edu/~trevisan/pubs/extractor-full.pdf

[13] Reshef, Yakir. On Resilient and Exposure-Resilient Functions. 2009. URL: https://www.math.harvard.edu/media/reshef.pdf.

[14] Shcherbyna, Y. Analysis of attacks in modern cyberphysical systems , Kazakova, N. , Fraze-Frazenko, O., Parchuts, L. , Schneider, S. CEUR Workshop Proceedings, 2019, 2683, pp. 12-14

[15] Ross S. A First Course in Probability. 8th Edition. 2010. ISBN-13: 978-0136033134 URL: http://julio.staff.ipb.ac.id/files/2015/02/Ross_8th_ed_English.pdf

[16] Sturges, H. (1926) The choice of a class-interval. J. Amer. Statist. Assoc., 21, P. 65–66. URL: https://www.tandfonline.com/doi/abs/10.1080/0162

[17] Freedman, D. and Diaconis, P. (1981) On this histogram as a density estimator: L2 theory. Zeit. Wahr. ver. Geb., 57, 453–476. URL: https://bayes.wustl.edu/Manual/FreedmanDiaconis1_1981.pdf