Introduction to simulations and Monte Carlo methods
Project 1

Weronika Szota & Mateusz Szczepański

The main purpose of this report is to examine the properties of some PRNGs (Pseudo Random Numbers Generators). We will consider: Mersenne-Twister, LCG, GLCG, RC4(32), Visual Basic's default PRNG and QCG. Apart from presenting the outputs of mentioned generators, we will also perform both theoretical (Kolmogorov-Smirnov Test, χ^2 Test) and statistical tests (Longest Run Test, Birthday Spacing Test, Pair Test) in order to verify randomness of obtained sequences. For most of our cases we will apply second-level testing procedure. We will determine if binary expansions of $\pi, e, \sqrt{2}$ can be used as PRNGs.

1. Description of used tests

Let us begin with description of tests that will be used. Sometimes it is necessary to convert between fromats of outputs to perform selected tests thus we will use following transformations:

$$y_i = \lfloor M u_i \rfloor, u_i = \frac{y_i}{M},$$

where:

- $u_1, u_2, ...$ are numbers with a distribution on the interval $(0, 1)$
- $y_1, y_2, ...$ are numbers with a discrete distribution on the set $\{0, 1, ..., M - 1\}$
- $b_1, b_2, ...$ is a sequence of bits, i.e. numbers with values $\{0, 1\}$

1.1 Kolmogorov - Smirnov Test (K-S)

Suppose that a random variable X has a continuous distribution with cdf $F_X(x)$. We also have n observations $X_1, X_2, ..., X_n$ and we want to test whether they are independent and come from $F_X(x)$. Let us denote empirical distribution function as $\hat{F}_n(x)$. Using the Gliwienko-Cantelli theorem, statistics

$$\hat{D}_n = \sqrt{n} \cdot \sup |\hat{F}_n(x) - F(x)|,$$

converges, when $n \to \infty$, to a known distribution

$$Pr(\hat{D}_n \leq t) \to H(t) = 1 - \sum_{i=1}^{\infty} (-1)^{i-1} e^{-2i^2 t},$$

H is the cdf of the Kolmogorov-Smirnov distribution. For a discrete distribution, the way D_n is computed is given in [3]. In order to perform this test we will transform outputs (if needed) to numbers with distribution on the interval $(0, 1)$ and verify if output of the generator come from $U(0, 1)$.

1.2 χ^2 Test

Suppose we have n observations (PRNG-generated sequence), each is from one of the k possible categories. Let Y_s be the number of observations from the s category, and let p_s be the probability of falling into the s category. For large n we expect that

$$Y_s \approx np_s$$

Assuming that the observations are independent and each indeed falls into the s category with probability p_s, statistics

$$\hat{\chi}^2 = \sum_{i=1}^{k} \frac{(Y_i - np_i)^2}{np_i}$$

has χ^2 distribution with $k-1$ degrees of freedom. In order to perform this test we will transform outputs (if needed) to numbers with distribution on the interval $(0,1)$ and verify if output of the generator come from $U(0,1)$.

1.3 Frequency Monobit Test

Frequency Monobit Test measures if there is not too many ones or zeros in sequence of bits. Suppose we have a bit sequence $b_1, b_2, ..., b_N \in \{0,1\}$. We convert the sequence to $x_1, x_2, ..., x_N \in \{-1,1\}$ by transformation $x_i = 2b_i - 1$ for every i. Our test statistic is

$$S_N = \frac{1}{\sqrt{N}} \sum_{i=1}^{N} x_i.$$

If the x_i sequence is independently and uniformly distributed then S_N has approximately $N(0,1)$ distribution. P-value is given as

$$p_{val} = 2(1 - \phi(|S_N|)),$$

where ϕ is the cdf of the normal distribution.

1.4 Longest Run Test

The test focuses on the longest run of ones within M-length blocks in bit sequence. The purpose of this test is to determine, whether the length of the longest run of ones within the tested sequence is consistent with the length of the longest run of ones that would be expected in a random sequence. Irregularity in ones implies irregularity in zeros, so in general, it is enough to test only ones. Whole description of the test can be found in [1].

1.5 Birthday spacing test

The test is based on the K statistics defined as follows. A year has k days (aka boxes) and we record r birthdays (aka balls) $Y_1, Y_2, ..., Y_r \in \{1, 2, ..., k\}$ and arrange them in the non-decreasing order $Y_{(1)} \leq Y_{(2)} \leq ... \leq Y_{(r)}$. Then we define the so-called spacings by

$$S_1 = Y_{(2)} - Y_{(1)}, ..., S_{r-1} = Y_{(r)} - Y_{(r-1)}, S_r = k - Y_r + Y_{(1)},$$

which we sort again in the non-decreasing order $S_{(1)}, S_{(2)}, ...$. Then K is the number of equal spacings

$$K = \#\{j \in \{1, 2, ..., r\} : S_{(j-1)} = S_{(j)}\}$$

We will use the following assumption, that for large r, k if the value $\lambda = r^3/(4k)$ is small, then assuming randomness hypothesis H_0, K has approximately Poisson distribution with parametrer λ. Thus, if we observe $K(obs)$ of equal spacing, then p-value is

$$Pr(K \geq K(obs)|H_0) \approx 1 - \sum_{j=0}^{y-1} \frac{\lambda^j}{j!} e^{-\lambda}$$

This test will not be performed for every generator due to the computational issues.

1.6 Pair Test

The test checks the distribution of pairs in sequence of integers. Suppose we have a sequence of integers less than 10 (we do not want to have more than 10 boxes in χ^2 test). In case of bigger integers we use modulo transformation. We split the initial sequence into pairs

$$(y_1, y_2), (y_2, y_3), ..., (y_{N-1}, y_N).$$

Then we can create a matrix of pairs:

$$\begin{bmatrix} (0,0) & (0,1) & (0,2) & (0,3) & (0,4) & (0,5) & (0,6) & (0,7) \\ (1,0) & (1,1) & (1,2) & (1,3) & (1,4) & (1,5) & (1,6) & (1,7) \\ (2,0) & (2,1) & (2,2) & (2,3) & (2,4) & (2,5) & (2,6) & (2,7) \\ (3,0) & (3,1) & (3,2) & (3,3) & (3,4) & (3,5) & (3,6) & (3,7) \\ (4,0) & (4,1) & (4,2) & (4,3) & (4,4) & (4,5) & (4,6) & (4,7) \\ (5,0) & (5,1) & (5,2) & (5,3) & (5,4) & (5,5) & (5,6) & (5,7) \\ (6,0) & (6,1) & (6,2) & (6,3) & (6,4) & (6,5) & (6,6) & (6,7) \\ (7,0) & (7,1) & (7,2) & (7,3) & (7,4) & (7,5) & (7,6) & (7,7) \end{bmatrix} \tag{M}$$

where each unique tuple will be replaced with the number of hits in our sequence of pairs. For these tests we consider each row as a box and then we perform χ^2 test. Assuming that if whole matrix should have uniformly distributed values - sums of each row should have them either. This type of considering boxes in χ^2 has many drawbacks what will be shown later.

1.7 Second-level testing

After performing each test we get one p_{val}. The idea behind second-level testing is as follows. Suppose we have a sequence (a result of PRNG) of length $R' = Rn$. We split it into R subsequences, each of length n. For each subsequence, we calculate the appropriate statistic and its corresponding p_{val}. Assuming H_0 (that the sequence is from the uniform distribution, depending on format: on $(0,1)$, on $\{0,...,M-1\}$ or on $\{0,1\}$, the obtained p_{val} : $p_1, p_2, ..., p_R$ are independent random variables with the distribution $U(0,1)$. We will follow NIST recommendations and split $[0,1]$ into $s = 10$ equal parts P_i where:

$$P_i = \left[\frac{i-1}{s}, \frac{i}{s}\right), 1 \le i \le s.$$

Then for $i \in \{1, 2, ..., s\}$ we define $E_i = \frac{R}{s}$ (expected number of p_{val} in P_i) and O_i as observed number of p_{val} in P_i. Ultimately we calculate the chi-square statistic

$$\hat{\chi}^2(obs) = \sum_{i=1}^{s} \frac{(O_i - E_i)^2}{E_i}$$

and the final p_{val}

$$p_{final} = P(\chi^2(s-1) > \hat{\chi}(obs)^2)$$

where $\chi^2(s-1)$ is a random variable with the χ^2 distribution with $s-1$ degrees of freedom. We will generate $R = 1000$ of sequences of length $n = 2^{16}$. For second level testing procedure we will perform K-S Test, χ^2 Test, Monobit Test, Pair Test and Longest Run Test.

2. Considering $\sqrt{2}$, π and Euler's Number as PRNG's.

2.1 Description

Consider binary representation of $\sqrt{2}, \pi, e$ as PRNG's. Below are performed tests how good (in pseudo random sense) generators of bits, these numbers are. Binary representations can be found under following links:

π	:	`http://www.math.uni.wroc.pl/~rolski/Zajecia/data.pi`
e	:	`http://www.math.uni.wroc.pl/~rolski/Zajecia/data.e`
$\sqrt{2}$	:	`http://www.math.uni.wroc.pl/~rolski/Zajecia/data.sqrt2`.

2.2 Testing Results

We perform Frequency Monobit Test and Longest Run test in order to verify null hipothesis about randomness of binary expansion for each of considered numbers. The results of those two test are shown in table below.

The Frequency Monobit test result:

Number	Test Statistic	P-value
$\sqrt{2}$	-0.23	0.82
π	-0.50	0.613
e	0.09	0.927

The Longest Run test result:

Number	Test Statistic	P-value
$\sqrt{2}$	16.32	0.012
π	14.51	0.024
e	3.69	0.719

We can notice that for testing Euler's Number we do not reject the null hipothesis in both cases, but we do according to Longest Run Test for each of remaining numbers. It seems as Euler's number looks like a good PRNG's output.

2.3 Extra visualization

Now we consider our binary representations in a different form - we take every 3 consecutive bits and create an integer from the set $\{0, 1, 2, 3, 4, 5, 6, 7\}$[1]. Below we can see histograms of each 3bit combined binary representation.

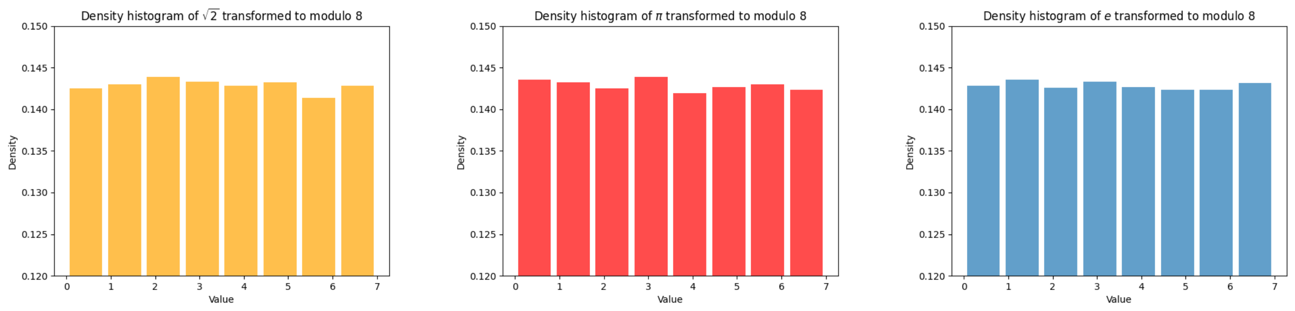


Figure 1: Density of transformed binary representations

2.4 Summary

On the one hand we can see that each binary representation passes Frequency Monobit Test with above 60% p-value. This is enough not to reject null hypothesis for all the significance levels considered. On the other hand only Euler's number passes Longest Run Test for 5% significance level. If we were to consider the significance level of 1%, all the numbers would pass the test. Visualizations of 3 bit sequences doesn't show us any improper and suspicious behaviours. Potentially a pair and birth test for bit generators can be performed by combining several bits into a single number (as in the visualization), but this would reduce the sample size for the given generators, and the tests themselves did not give interesting results, so this is not shown.

Based on the results given above, we can conclude that the Euler's number's binary expansion can be treated as an output of quite good PRNG.

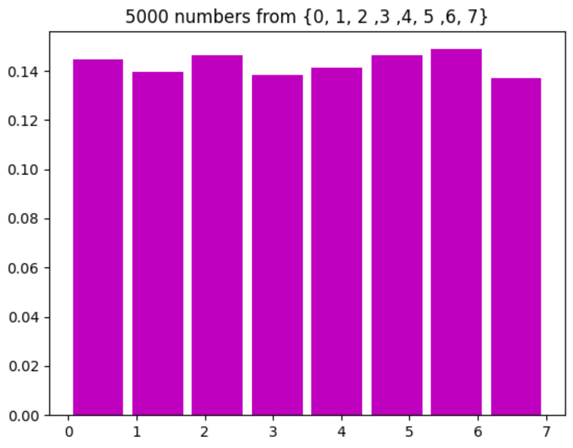
[1]For example if we get 3 consecutive bits 1, 1, 0 we combine them into an integer $x = 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 6$.

3. Testing PRNG's and performing second level testing

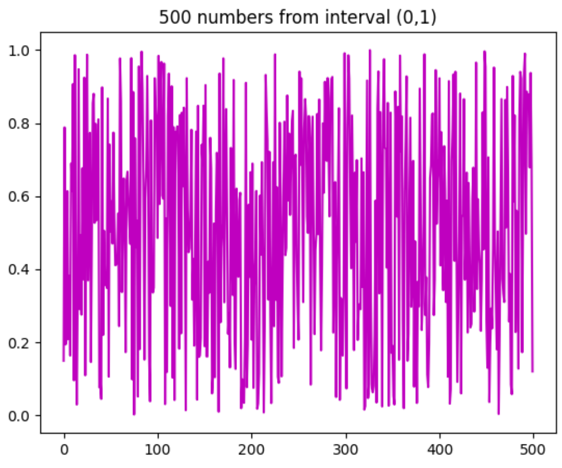
3.1 Mersenne - Twister (M-T)

In this part we consider default Python's PRNG. The whole description can be found in [2]. K-S test executed with numbers generated by `random.random()`. The rest generated by `random.randint()` for specified integer bounds. For tests on bits we generate 2^{16} bits with `random.randint(a=0, b=1)`. For other tests we generate 2^{16} non-negative integers less than 8. In every case seed has been set to 0. We expect that this generator will pass most of the tests.

Below we have a histogram and a variation plot for less number of observations to have good-looking plots.



(a) $n = 5000$ for integers from {0,1,2,3,4,5,6,7}



(b) $n = 500$ floats from $(0, 1)$

Figure 2: Mersenne-Twister visualizations

We can see no weird behavior in histogram and no special patterns in variation plot. Nextly we check how good Mersenne-Twister is based on selected tests.

Test	K-S	χ^2	Pair Test	Birthday spacing	Frequency Monobit	Longest Run
P-value	0.750	0.913	0.914	–	0.494	0.120

Results of single tests are as expected. All p_{val} are hardly over 5%, no pattern, no zeros and ones. It looks like a good PRNG.

Now we will consider second level testing for Frequency Monobit test and χ^2 test.Frequency Monobit test gives $p_{final} = 0.272$, when χ^2 test gives $p_{final} = 0.390$. Results prove that sequence generated with M-T generator is well random.

As we could expect, Mersenne-Twister generator is very immune to tests. No weird behaviors, periods, too big or too small p_{val}. Based on considered tests we can conclude that M-T is a very good PRNG.

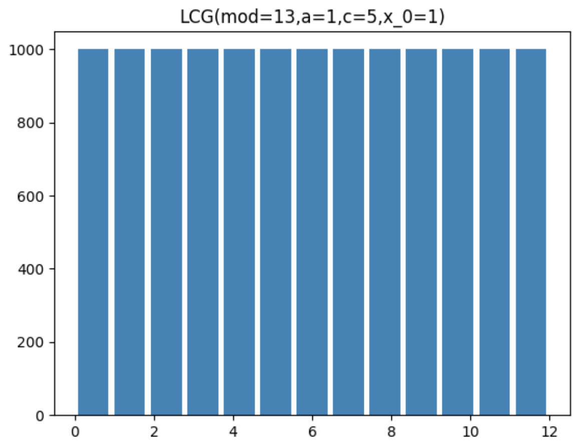
3.2 LCG & GLCG

3.2.1 LCG

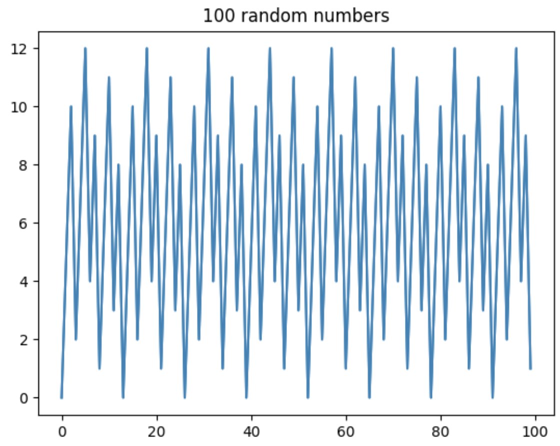
Let us consider **L**inear **C**ongruent **G**enerator given by

$$x_{n+1} = (ax_n + c) \bmod m,$$

with initial state x_0 and constants a, c. In our case $x_0 = 1$. We can expect that the PRNG's period will be short and for that reason numbers will be cyclical, no matter how good we determine parameters a, m, c, x_0.
Firstly, we will examine results of generating 2^{16} numbers using LCG with: $m = 13, a = 1, c = 5, x_0 = 1$.



(a) $n = 13000$ for integers from {0,1,2,3,...,12}



(b) first $n = 100$ random numbers

Figure 3: LCG(13,1,5,1) visualizations

Figure 3 shows the histogram of the whole sample as well as plot of the first 100 generated numbers. We can see that the period of generator is short as we expected. The plot shows that the variation of numbers looks like there is a visible pattern. On the other hand the histogram looks ideally what may give rise to our doubts.

Now we are going to verify our observations with the results of some theoretical and empirical tests performed on generated numbers. The following table shows p_{val} obtained after performing selected tests.

Test	K-S	χ^2	Pair Test	Birthday spacing	Frequency Monobit	Longest Run
P-value	0	0	0.999	0.999	0	0

As we can see above we do reject null hipothesis according to most of tests. As mentioned before, we will also perform second-level testing. We obtain 1000 single p_{val} which lead us to the final ones. Performing second-level testing leads us to five final p_{val} all equal to 0. Based on that we conclude that we should reject null hipothesis about $p - values$ uniformity.

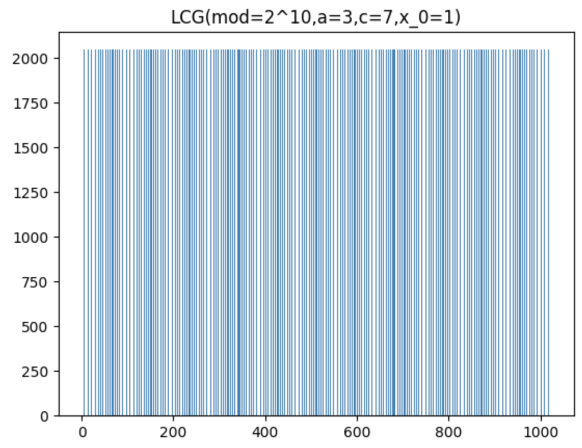
Now let us consider Linear Congruent Generator with different parameters: $m = 2^{10}, a = 3, c = 7, x_0 = 1$. Figure 4 presents the histogram of 2^{20} obtained numbers and the first 100 random numbers. The histogram looks ideally as in the previous example. Based on the plot it is more difficult to spot the pattern in comparison to the prevoius example.

In this case results based on selected tests are more diverse.

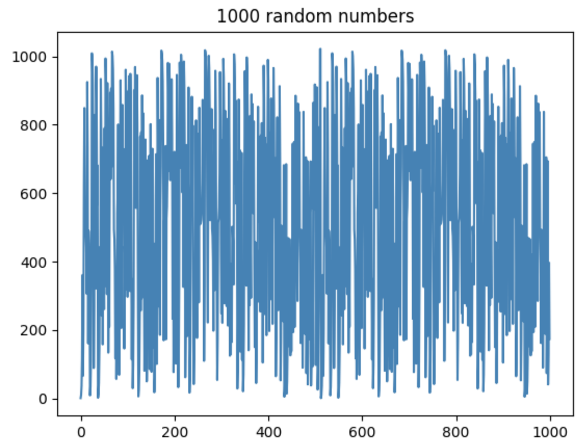
Test	K-S	χ^2	Pair Test	Birthday spacing	Frequency Monobit	Longest Run
P-value	0.00068	0	0.999	0.999	1	0

We reject null hipothesis about the randomness of generated numbers according to χ^2 Test, K-S Test and Longest Run Test. In other cases we do not reject the null hipothesis.

Performing second-level testing leads us to five final p_{val} all equal to 0. Based on that we conclude that we should reject null hipothesis about p_{val} uniformity.



(a) $n = 2^{20}$ for integers from $\{0,1,2,3,...,2^{10}-1\}$



(b) first $n = 1000$ random numbers

Figure 4: LCG(2^{10},3,7,1) visualizations

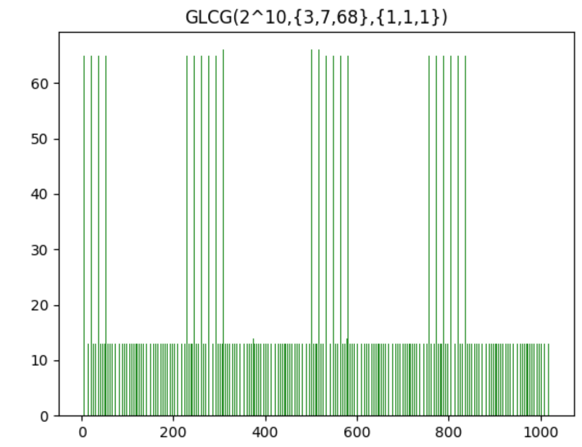
3.2.2 GLCG

The next generator we consider is **G**eneralized **L**inear **C**ongruent **G**enerator given by

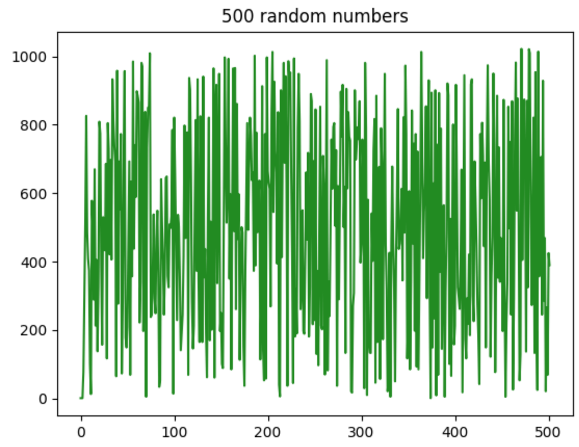
$$x_{n+1} = (a_1 x_{n-1} + a_2 x_{n-2} + ... + a_k x_{n-k}) \bmod m,$$

with initial state given by sequence $(x_1, x_2, ..., x_k)$ and sequence of constants $(a_1, a_2, ..., a_k)$. We can think of this generator as the more general form of previously considered LCG. The obtained results could be better than previous ones depending on the choice of parameters. In the following simulations we will set seed equals to $(1, 1, 1)$.

As shown in Figure 5 we have more diverse results. Based on histogram we can see that some of numbers occur more often. It is difficult to spot the pattern while examing plot of the first 500 numbers.



(a) $n = 10000$ integers



(b) first $n = 500$ random numbers

Figure 5: GLCG(2^{10},{3,7,68},{1,1,1}) visualizations

Studying results obtained from selected tests we can notice that null hipothesis should be rejected based on the Longest Run Test's result. Other tests' results proves randomness of generated numbers.

Test	K-S	χ^2	Pair Test	Birthday spacing	Frequency Monobit	Longest Run
P-value	0.9531	0.3641	0	–	0.9194	0

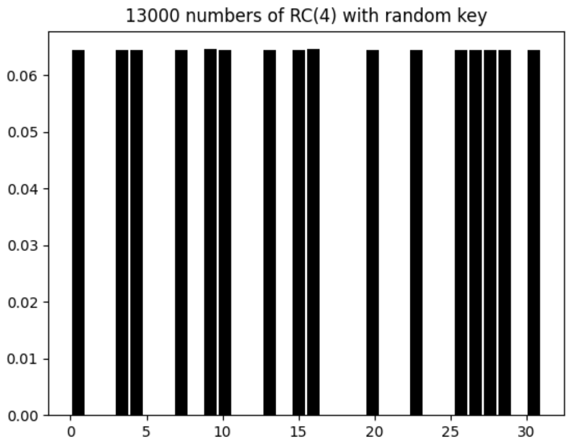
Performing second-level testing leads us to five final p_{val} all equal to 0. Based on that we conclude that we should reject null hipothesis about p_{val} uniformity.

3.3 RC4(32)

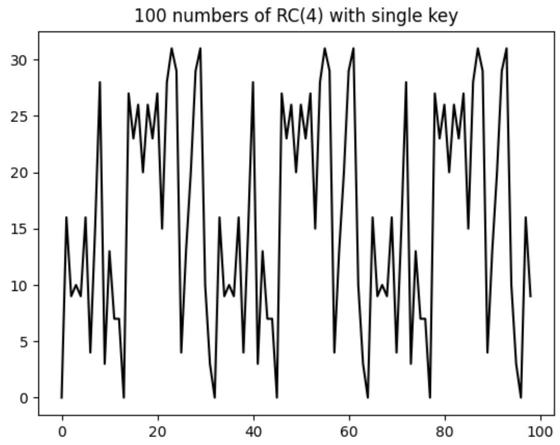
We consider 3 different types of PRNG's based on RC4 stream cipher algorithm. Based on results we will try conclude which is the best. Description of used algorithm can be found in [3].
In all cases we identify **key** as our seed.

3.3.1 RC4(32) with single key

Single **key** means that our **key** is fixed single element list like **key = [1]**. In this case we generate $n = 9 \cdot 10^6$ numbers with this version. For these tests let the fixed key be **key = [7]**.
Below we can find histogram and variation plot of this modification.



(a) $n = 13000$ integers



(b) first $n = 100$ numbers

Figure 6: RC4(32) single key

We can see period and pattern, then we can expect that Pair Test and χ^2 will be cheated.
And results of tests:

Test	K-S	χ^2	Pair Test	Birthday spacing	Frequency Monobit	Longest Run
P-value	0	0	1	–	0	0

Second level testing examine in almost the same results then we reject null hypothesis.

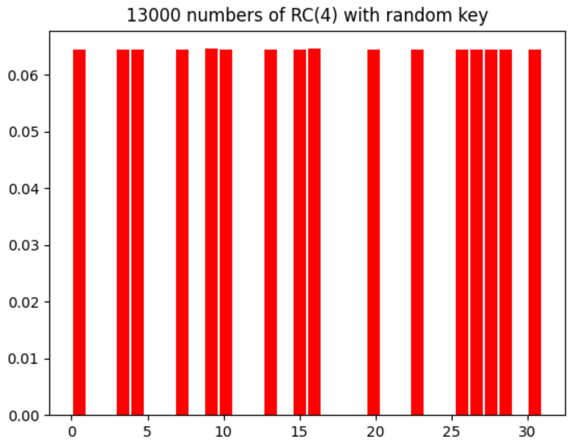
3.3.2 RC4(32) with random key

Random key is generated at start as n element list of integers - generating is done by M-T.
Example **key = [1,6,2,5]**. In this case we generate 3 times $n = 3 \cdot 10^6$ numbers with 3 generated **keys** with length

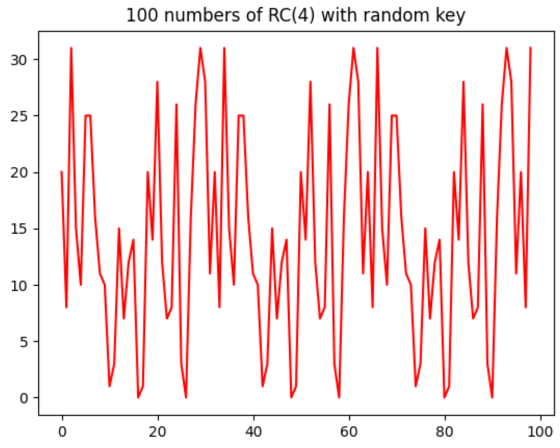
4, 8 and 16. **Keys** in single tests are:

$$\begin{aligned}\mathbf{key}_1 &= [\mathbf{24, 26, 2, 16}]\\ \mathbf{key}_2 &= [\mathbf{31, 25, 19, 30, 22, 13, 8, 18}]\\ \mathbf{key}_3 &= [\mathbf{8, 6, 16, 9, 19, 6, 4, 21, 30, 6, 22, 27, 20, 13, 30, 28}]\end{aligned}$$

Below we can find histogram and variation plot of this modification.



(a) $n = 13000$ integers



(b) first $n = 100$ numbers

Figure 7: RC4(32) random key

Results of tests:

Test	K-S	χ^2	Pair Test	Birthday spacing	Frequency Monobit	Longest Run
P-value	0	0	0	–	0	0

Same situation as in previous modification.

3.3.3 RC4(32) with ordered key

Ordered key is the n element list of ordered integers like **key** = [**1,2,3,4,5**].
Let fix our **key** in this case as **key** = [**5,6,7,8,9,10,11,12**].
Below we can find histogram and variation plot of this modification.
Again the same situation. Then changing key showed no differences.
And results of tests:

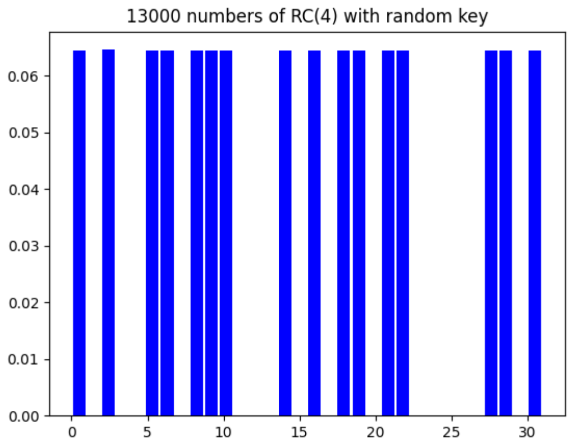
Test	K-S	χ^2	Pair Test	Birthday spacing	Frequency Monobit	Longest Run
P-value	0	0	1	–	0	0

In summary we can say that RC4(32) is periodic and this allows us to reject null hypothesis about randomness and independence.
Second level testing doesn't give any other results so it will not be performed in this subsection.

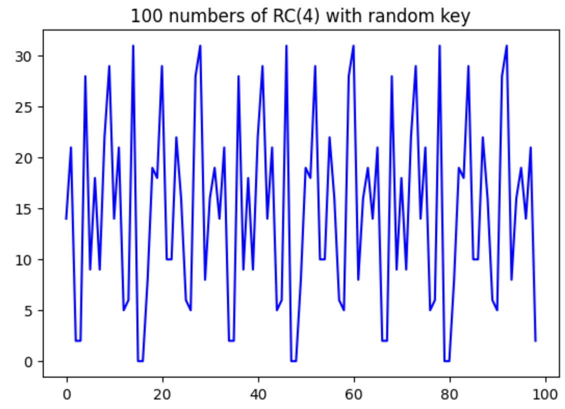
3.4 Visual Basic

Below we consider Visual Basic PRNG. Implementation is as below:

$$x_{n+1} = (1140671485 \cdot x_n + 12820163) \bmod 2^{24},$$



(a) $n = 13000$ integers

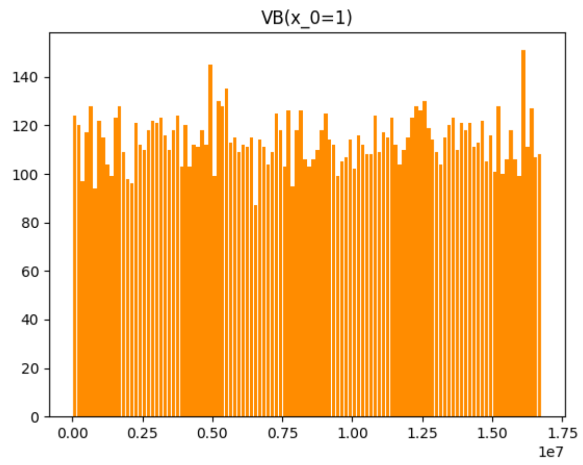


(b) first $n = 100$ numbers

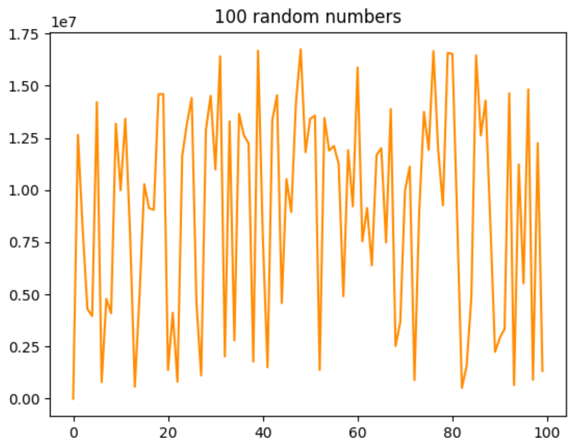
Figure 8: RC4(32) ordered key

with initial value x_0. We will consider $x_0 = 1$.

Firstly, let us examine results based on Figure 6. As shown in histogram in this case we can see irregularity in the number of occurance of each value. There is also no visible pattern that can be spotted in the plot.



(a) $n = 13000$ integers



(b) first $n = 100$ random numbers

Figure 9: Visual Basic ($x_0 = 1$) visualizations

As we can see, the results of selected tests confirm the randomness of generated sequence.

Test	K-S	χ^2	Pair Test	Birthday spacing	Frequency Monobit	Longest Run
P-value	0.8394	0.9227	1	–	0.3593	0.82

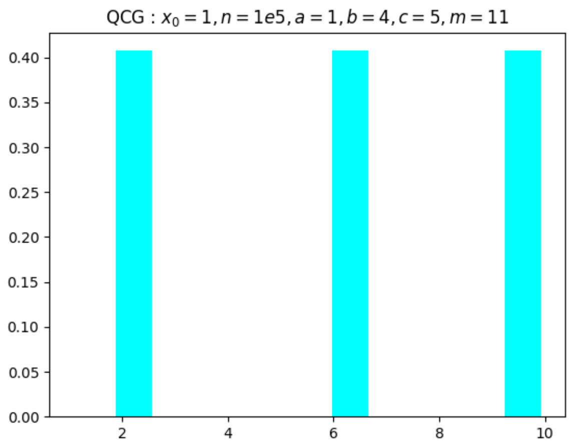
Performing second-level testing lead us to five final p_{val}: 0.3012 for χ^2 Test and 0 for Monobit Frequency Test, Pair Test, Longest Run Test and 0.5201 for K-S Test. It turns out that according to χ^2 and K-S sequence generated using Visual Basic's PRNG is random.

3.5 Quadratic Congruent Generator

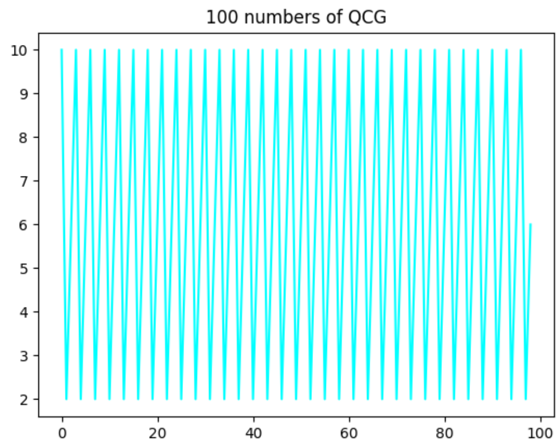
At the end have **Q**uadratic **C**ongruent **G**enerator given by

$$x_{n+1} = (ax_n^2 + bx_n + c) \bmod m,$$

with initial state x_0 and constant integers a, b, c.
For single tests we will examine results of generating 2^{16} numbers using QCG with parameters $m = 11$, $a = 1$, $b = 4$, $c = 5$, $x_0 = 7$. To get uniformly distributed bits m need to be changed to for example 16. Only in this case we can expect good distribution of bits.
Below histogram and variation plot.



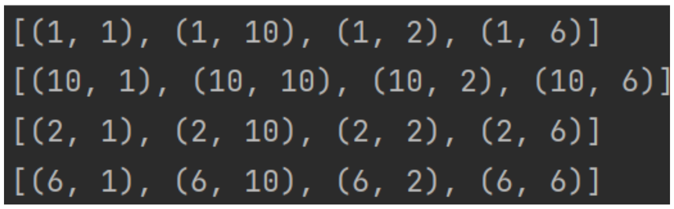
(a) $n = 10000$ integers from {0,1,...,10}



(b) $n = 100$ integers from {0,1,...,10}

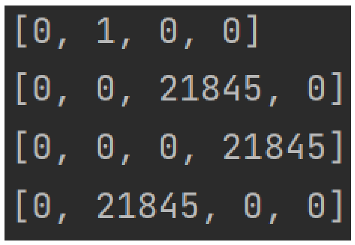
Figure 10: QCG visualizations

As we can see histogram is absolutely not uniformed distributed. Only 3 values are visible in our sequence. Thus we expect that χ^2 and pair test will fail. Similarly on variation plot where obvious seasonality is visible.
Pair test fails due to lack of non-zero values in matrix (M). Matrix in this case looks like on Figure 7a. Figure 7b with specific values of pair hits shows that it doesn't have any chance to pass the test.



```
[(1, 1), (1, 10), (1, 2), (1, 6)]
[(10, 1), (10, 10), (10, 2), (10, 6)]
[(2, 1), (2, 10), (2, 2), (2, 6)]
[(6, 1), (6, 10), (6, 2), (6, 6)]
```

(a) QCG : non-zero elements in pairs in matrix (M)



```
[0, 1, 0, 0]
[0, 0, 21845, 0]
[0, 0, 0, 21845]
[0, 21845, 0, 0]
```

(b) QCG : values of matrix (M)

Figure 11: QCG pair matrix

Results of all tests:

Test	K-S	χ^2	Pair Test	Birthday spacing	Frequency Monobit	Longest Run
P-value	0	0	0	–	0	0

Second level testing is not necesarry in this case. All tests will fail due to constant $p - value = 0$. Other choices of constants a, b, c results in a bit different results but most of them share the same characteristics - period shorter than module, visible pattern in variation plot and failing mosts of tests. In many publications can be found that module should be much bigger like 2^{15}, but we were considering only small modules for this generator.

4. Final summary

Let us briefly summarize conclusions discussed in the previous part.

- We checked that binary expansion of e (Euler's Number) can be treated as an output of a good PRNG.
- There is a difference between results from performing χ^2 based on generated sequence and tranforming this sequence to (0,1).
- Pair test can be cheated easy be many PRNG's by summing rows or columns. Considering each pair as a box in χ^2 can give us better results but performing χ^2 for many boxes is risky because a lot of small differences can make huge statistic value which will lead to deceptive results.
- Second-level procedure should be applied especially when obtained p_{val} are high.
- Obtained results depend on choice of the parameters and the number of generated numbers.
- It turned out that Mersenne-Twister and Visual Basic's PRNG generators were the best from ones we considered in this report.
- GLCG is a good example of generator which is able to pass many tests once but it is weak in second level testing.

Most of scripts done in Python 3.10 which allow to perform all tests can be found in repository under url:
`https://github.com/mszczepanskigit/PRNG-s-Testing.git`

References

[1] Lawrence E. Bassham, Andrew L. Rukhin, Juan Soto, James R. Nechvatal, Miles E. Smid, Stefan D. Leigh, M Levenson, M Vangel, Nathanael A. Heckert, D L. Banks, *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*, 2010

[2] `https://en.wikipedia.org/wiki/Mersenne_Twister`

[3] `http://www.math.uni.wroc.pl/~rolski/Zajecia/sym.pdf`