Introduction to Monte Carlo simulation and methods - Project I

Svenja Mikłaszewska

1 Introduction

Pseudorandom number generators (PRNG) are widely used in simulations and various types of calculations.

In this project, we want to compare several such generators. We are interested in how the drawn numbers are randomly distributed, that is, how evenly they are dispersed.

We are considering generators:

- LCG(M,a,c)

  $x_n = (ax_{n-1} + c)$ mod M

- GLCG (M,a_i, i =1,...,k)

  $x_n = (a_1 x_{n-1} + ... + a_k x_{n-k})$ mod M

- RC4(32)

  The initialization process of the RC4 algorithm involves two main phases: initialization of the S array and permutation of this array based on the key.

To be able to compare the generators among themselves, we will look at their p-values. We will obtain them by applying the slugging statistical tests.

Tests considered:

- χ^2 test

  It is a statistical test of concordance, used to check whether the distribution of observed frequencies differs from the expected distribution (in our case, a uniform distribution).

- KS test (Kolmogorov Smirnov)

  It is a non-parametric statistical test used to compare two distributions or to compare an empirical distribution with a theoretical distribution.

- Frequency Monobit test

  Is a fundamental tool in assessing the quality of random number generators The test counts the number of ones and the number of zeros in the given binary sequence.

$$s_b(obs) = \frac{1}{\sqrt(n)} \sum_{i=1}^{n} x_i$$

2 Choosing

In addition to the above generators and tests shown in the lecture, we need to apply one selected generator and test as part of the project.

2.1 Generator

Python uses the Mersenne Twister (MT) generator. It was chosen for this project because we know that it has a long cycle time and is fast at the same time. Therefore, the numbers drawn by it are very close to a uniform distribution.

Brief MT description:

- The Mersenne Twister algorithm uses a large internal array of 624 words (each word has 32 bits). This array is initialized with a starting value called a "seed."
- MT generates pseudorandom numbers using linear recursion with a large period equal to 2^19937-1. This period means that before the algorithm starts repeating a sequence of numbers, it will generate a huge number of different values.
- The internal array is updated using special bit operations that combine existing values in the array into a new sequence.
- The process involves bit shifting operations, XOR operations and bit masking to produce highly random number sequences.
- When the internal array is updated, the numbers are extracted from the array and transformed into a pseudo-random number through additional bit operations.

2.2 Test

From a range of differential tests, we chose the Linear Complexity Test (LCT)

Brief description:

The focus of this test is the length of a generating feedback register. The purpose of this test is to determine whether or not the sequence is complex enough to be considered random. Random sequences are characterized by a longer feedback register. A short feedback register implies non-randomness.

3 Comparing generators

Before we start drawing, let's establish a few things, the number of draws i.e. n = 10^5 and x_0 is arbitrarily chosen by us.

According to the terms of the task, we consider

- LCG(13,1,5,x_0,n)
- GLCG(2^{10},{3,7,68})
- RC4(32)
    1. Initialize with one key - some function of current time
    2. Fixed key length (two versions)

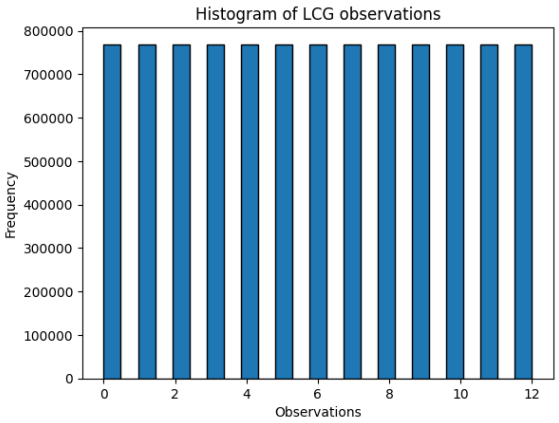

Figure 1: LCG

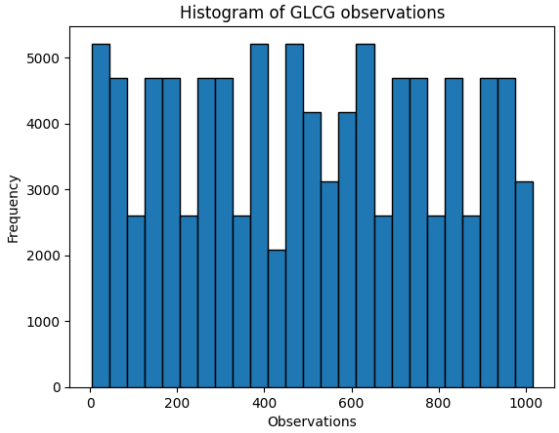

Figure 2: GLCG

From the charts above, we can see that values from 0 to 12 (which is also the length we specified) were drawn at the LCG. The number of these values is evenly distributed among each other.

With GLCG, values from 0 to 1000 have been drawn, with some values occurring as frequently as others, and some less frequently.

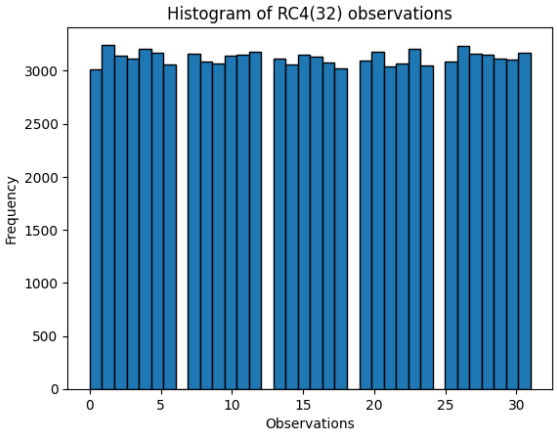

Figure 3: Key - current time

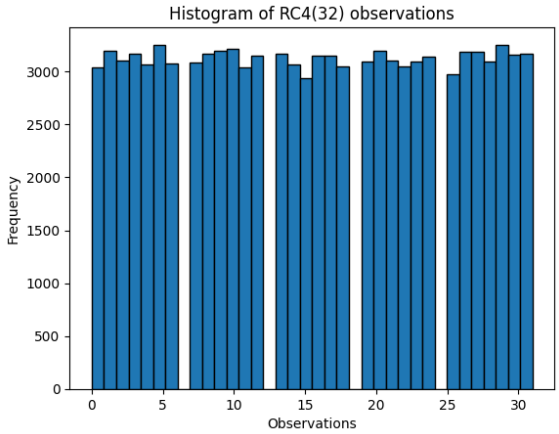

Figure 4: Fixed key (1)

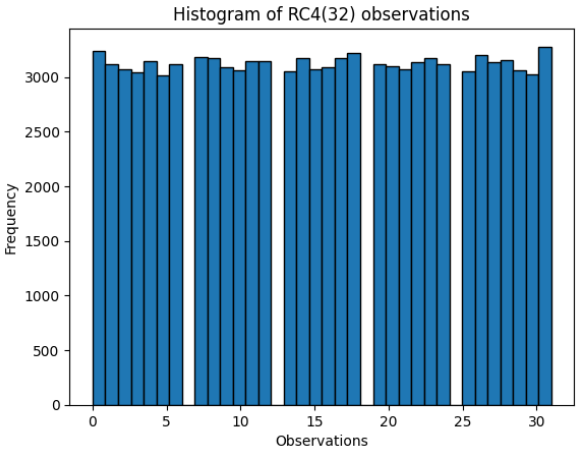

Figure 5: Fixed key (2)

The histograms for the RC4(32) generator with different keys show identical frequencies of occurrence of the observed numbers. It is easy to see that the choice of the key or the function determining the key values for this generator is not significant, since we get a similar number of numbers.

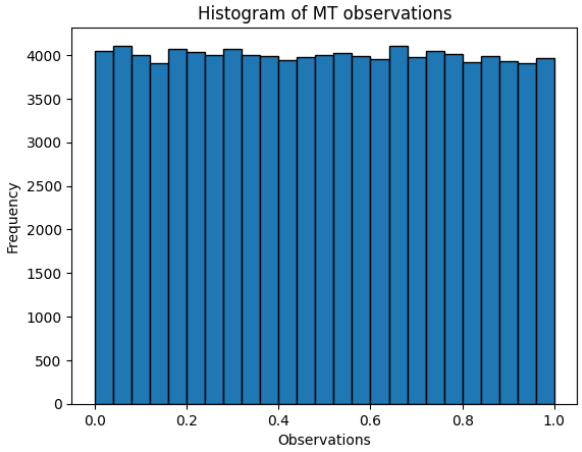

Figure 6: Mersenne Twister

Since the Mersenne Twister gnereator is commonly used in Python, it is obvious to us to expect a mostly even distribution of observation frequencies.

3.1 Tests and p-values

So far we have looked at the graphs of the generated numbers. We now turn to analyzing the results of the obtained p-values from the tests discussed in 2.2.

Table 1 shows the results for the χ^2 test.

LCG, GLCG have very high values for the χ^2 statistic and a p-value of 0.0, indicating that there is a strong deviation from the expected random distribution. This suggests that these sequences are not random.

RC4(1), RC4(2) are RC4(32) with a current time key and a fixed key, respectively.

RC4(1), RC4(2) and the Mersenne Twister have p-values much higher - thus passing the χ^2 test.

Sequence	Chi-squared statistic	P-value
LCG	355030.4142	0.0
GLCG	233.6903	0.0
RC4(1)	8.2883	0.9117
RC4(2)	22.0422	0.1067
Mersenne Twister	0.8682	0.9291

Table 1: Chi-square test

In table 2, only the Mersenne Twister passes the Kolmogorov-Smirnov test. We observe, a drastic difference in p-values for RC4 (between table 1, and table 2), because the Kolmogorov-Smirnov test compares the empirical sample distribution with the reference probability distribution. And in our case, RC4, has a distribution close to a step function, and the test takes the distribution of the Normal distribution.

Sequence	KS statistic	P-value
LCG	0.0769	0.0
GLCG	0.0234	0.0
RC4(1)	0.0315	0.0
RC4(2)	0.0323	0.0
Mersenne Twister	0.0019	0.8523

Table 2: Kolmogorov-Smirnov test

In Table 3, LCG and GLCG have a p-value of 0.0 suggests that these sequences have low linear complexity, meaning no randomness.

RC4(1), RC4(2), Mersenne Twister have p-values much higher than 0. This means that they have high linear complexity, suggesting that their results are closer to the expected random distribution.

Sequence	P-value
LCG	0.0
GLCG	0.0
RC4(1)	0.4733
RC4(2)	0.9066
Mersenne Twister	0.9397

Table 3: Linear Complexity Test

Mersenne Twister scored positively in all statistical tests, suggesting that it is the most random generator of all those analyzed. RC4 generators, on the other hand, passed two out of three tests, placing it second in terms of randomness.

3.2 Second-level testing

"Second level testing" involves evaluating the distribution of p-values obtained from previous statistical tests to see if they conform to the expected uniform distribution.

Key steps of second level testing:

- Data Partitioning:

  The first step involves dividing the larger data set into smaller subsets, and then running a test (e.g., a monobit test) on each of these subsets. The result of this step is p-values for each subset.

- Calculation of Observed and Expected Frequencies:

  The collected p-values are then divided into intervals, and their abundance (observed frequency) is compared with the expected frequency, assuming a uniform distribution across these intervals.

- Test χ^2:

  A chi-square test is then performed to see if the differences between the observed and expected frequencies are statistically significant. If the differences are small, it means that the p-values are evenly distributed, suggesting a good quality of randomness in the original data set.

3.2.1 LCG

The charts for second-level testing for the LCG generator are shown below.

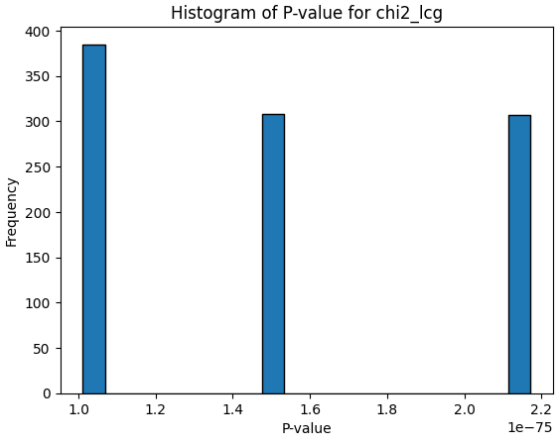

Figure 7: χ^2

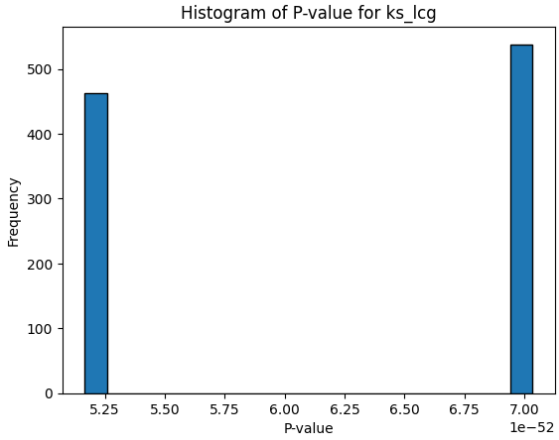

Figure 8: Komgorov-Smirnov

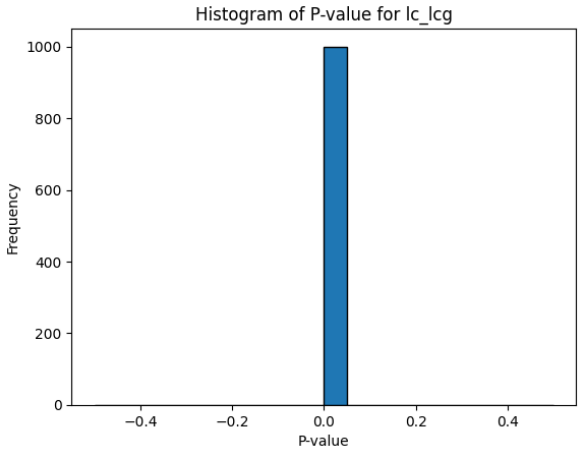

Figure 9: Linear Complexity Test

We observe that the p-values are not uniformly distributed, reaching mostly zero values. Which tells us that drawing using LCG is not the best method to generate pseudo-random numbers.

3.2.2 GLCG

Histograms of second-level results for the GLCG generator are shown below.

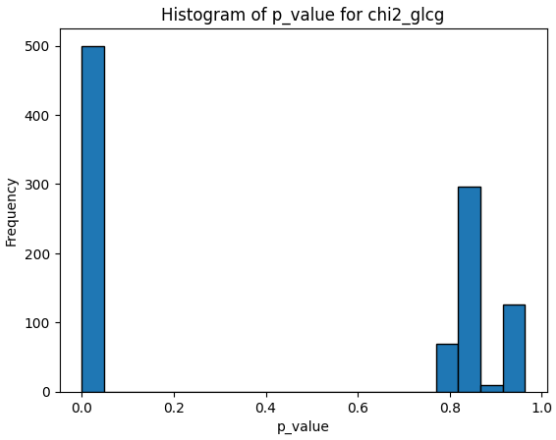

Figure 10: χ^2

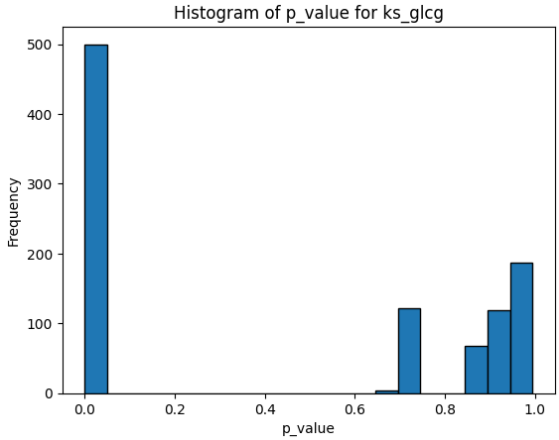

Figure 11: Komgorov-Smirnov

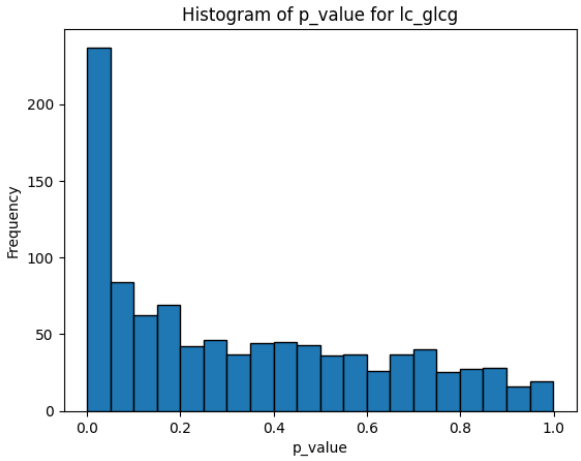

Figure 12: Linear Complexity Test

All three histograms show significant deviations from the expected uniform distribution. This suggests that the GLCG generator is not perfectly random and exhibits some regularities that can be detected by various statistical tests.

The Linear Complexity Test histogram indicates low complexity, further emphasizing the lack of randomness in the data

3.2.3 RC4(1) and RC4(2)

The following graphs show the p-value distribution of the RC4 generator for both cases under consideration. They are similar (between generators) hence the conclusions are indentical for both of them.

Hence, we are able to observe that for the χ^2 and LCT tests show an even distribution, indicating a high degree of randomness of the data in these tests. In contrast, the p-value histograms for the Kolmogorov-Smirnov test show significant deviations, suggesting that the data may show some regularity that does not follow the expected random distribution.

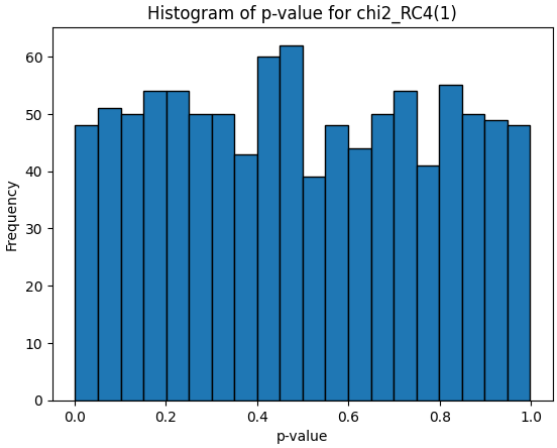

Figure 13: χ^2

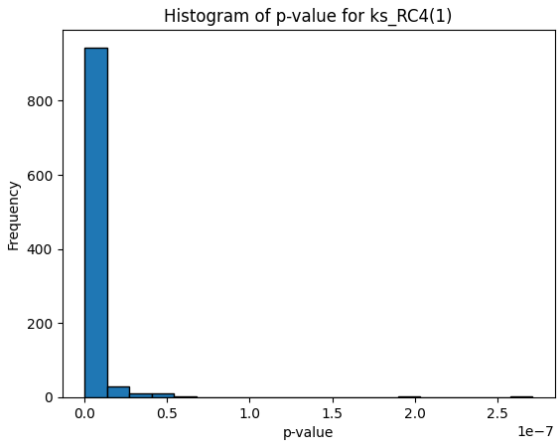

Figure 14: Komgorov-Smirnov

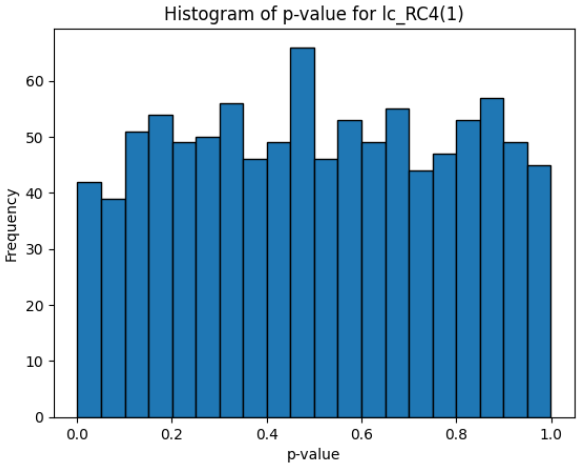

Figure 15: Linear Complexity Test

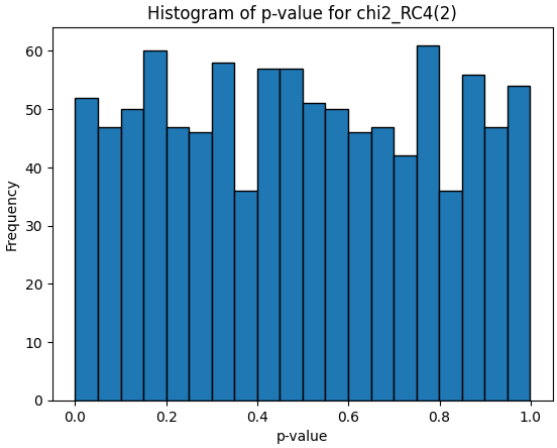

Figure 16: χ^2

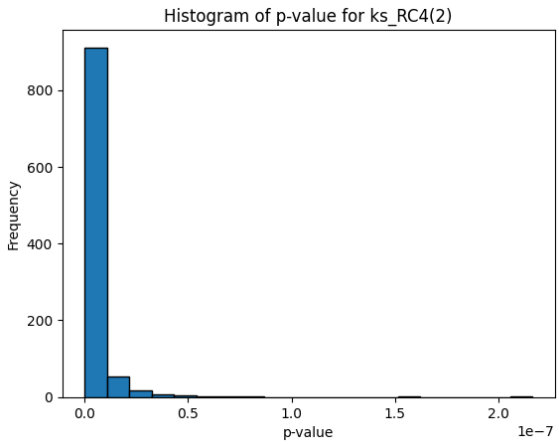

Figure 17: Komgorov-Smirnov

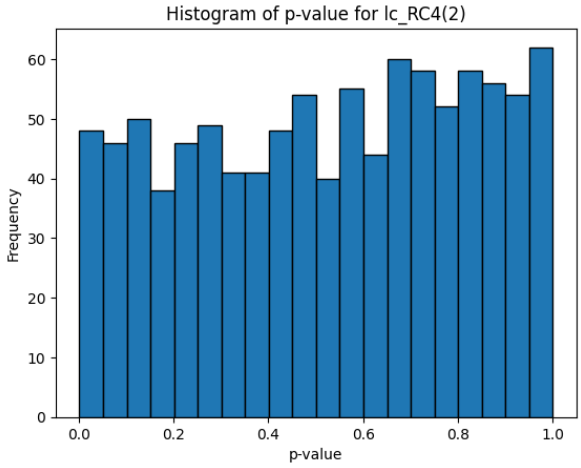

Figure 18: Linear Complexity Test

3.2.4 Mersenne Twister

The histograms below show the results of second level testing. As we can see, the distribution of p-values for each test is very close to a uniform distribution.

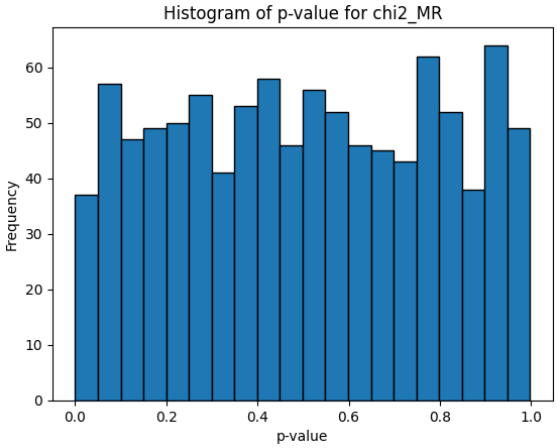

Figure 19: χ^2

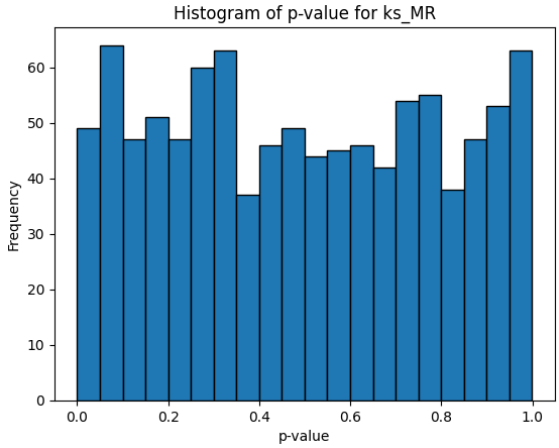

Figure 20: Komgorov-Smirnov

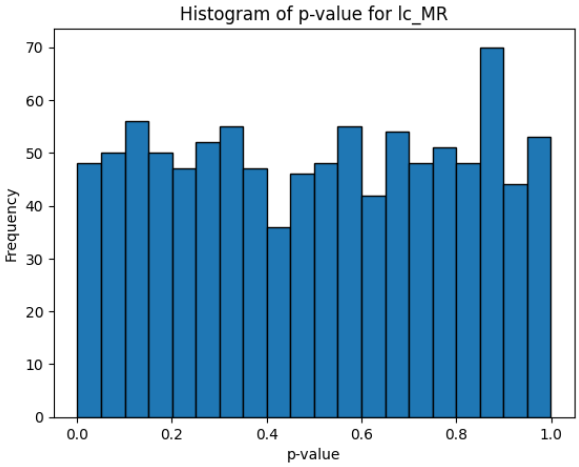

Figure 21: Linear Complexity Test

The histograms indicate the good randomness quality of the Mersenne Twister generator, which confirms its ability to generate numbers with a high degree of randomness. Mersenne Twister passes all statistical tests, suggesting that it is one of the best random number generators available.

3.2.5 Table

The following is a table with the p-values of the χ^2 test calculated from a set of p-values for each of the considered generators under different tests.

Sequence	Chi-squared statistic	P-value
chi2.RC4(1)	11.2	0.7383
ks.RC4(1)	15000.0	0.0
lc.RC4(1)	16.928	0.3232
chi2.RC4(2)	11.008	0.752
ks.RC4(2)	15000.0	0.0
lc.RC4(2)	17.312	0.3006
chi2.MT	6.97	0.1375
ks.MT	3.93	0.4156
lc.MT	2.27	0.6862
chi2.LCG	4000.0	0.0
ks.LCG	4000.0	0.0
lc.LCG	4000.0	0.0
chi2.GLCG	1470.36	0.0
ks.GLCG	1028.76	0.0
lc.GLCG	416.35	0.0

Table 4: χ^2 test

The table shows the results of the χ^2 test in the context of second-level testing of the considered generators.

The results show that the sequences generated by LCG and GLCG did not pass any of the χ^2 tests, as the p-values are 0, indicating a significant deviation from the expected uniform distribution. This ultimately reassures us that these generators are not adequately random.

The RC4(1) and RC4(2) sequences show good agreement with the expected distribution compared to LCG and GLCG. The KS test indicates problems with the uniform distribution. This suggests that the RC4 generator shows some randomness, but is still not perfectly random.

For the Mersenne Twister, all tests indicate that there are no significant deviations from the random distribution, confirming the high randomness quality of this generator.

Overall, Mersenne Twister shows the best results in terms of conforming to the expected random distribution, while LCG and GLCG show significant deviations, suggesting their lower randomness quality. RC4 sequences have mixed results depending on the test, but generally present good results, but with some caveats.

4 Number π,e,$\sqrt{2}$

In the second part of the project, we will look at the binary expansions of π,e and $\sqrt{2}$. We treat their decimal expansions as potential pseudorandom number generators.

We will compare all three "generators" by applying the Frequency Monobit Test.

4.1 Frequency Monobit Test

The table below shows the p-values along with the values of the statistic $S_n(obs)$.

Sequence	S(obs)	P-value
pi	0.5048	0.6137
e	0.0918	0.9269
sqrt(2)	0.2304	0.8178

Table 5: Frequency Monobit Test

All three sequences (π, e, $\sqrt{2}$) passed the Frequency Monobit test. High p-values in each case suggest that the distribution of bits in these sequences follows the expected random distribution. Therefore, we can conclude that the bits in these sequences are distributed in a way that shows no significant deviation from randomness.

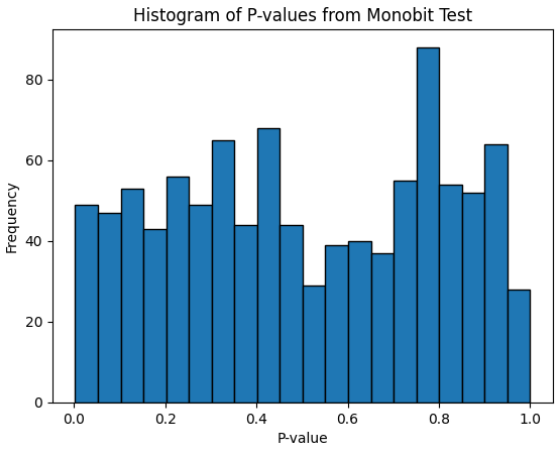

Figure 22: π

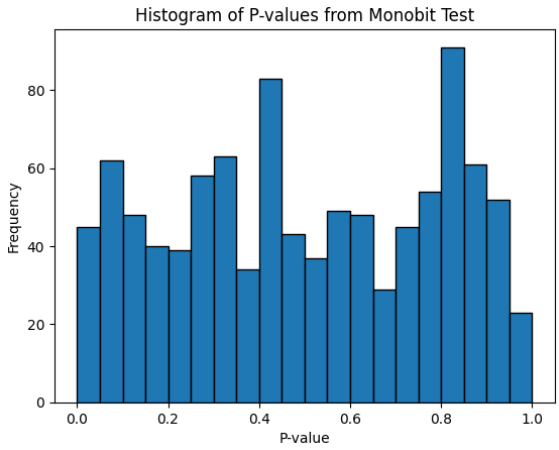

Figure 23: e

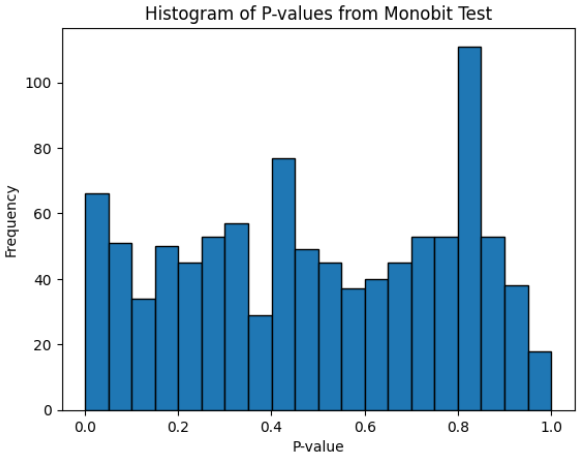

Figure 24: $\sqrt{2}$

Of the above graphs, we observe that the p-value distribution for each sequence is similar.

In Figure 21, we observe an increase in the frequency of p-value values in the range of 0.0-0.4, indicating that most of the tested values had low p-values, which may suggest that the π sequence passes the Frequency Monobit test, but not all values are evenly distributed.

A similar distribution to that of e has e and $\sqrt{2}$ with frequencies increasing from 0.0 to 0.4 and decreasing at 0.5. This may indicate that the e and $sqrt2$ sequence passes the Frequency Monobit test, but with some deviations from the expected random distribution.

Although all three sequences passed the Frequency Monobit test, increases in the frequency of p-value values in the lower ranges may suggest some regularities or patterns that are not perfectly random

4.2 Second-level testing

Sequence	Chi-square test	P-value
pi	16.9861	0.0019
e	7.504	0.1115
sqrt(2)	3.9781	0.409

Table 6: Second-level test

The results of the χ^2 test show that $\sqrt{2}$ is the most random, the e number shows moderate conformity to randomness, and the π number shows the greatest deviation from the expected uniform distribution.

These results suggest that among the analyzed generators, e is closest to perfect randomness, while e is furthest from it.

5 Table with functions

Category	Item	Function
Figure	Fig 1	lcg(13,1,5,10^5)
Figure	Fig 2	glcg(2^{10},[3,7,68],[4,88,0],10^5)
Figure	Fig 3-5	initialize_with_key();KASA(key);PRGA(S)
Figure	Fig 6	random.random()
Figure	Fig 7-9	perform_tests_lcg()
Figure	Fig 10-12	perform_tests_glcg()
Figure	Fig 13-15	perform_tests_rc1()
Figure	Fig 16-18	perform_tests_rc2()
Figure	Fig 19-21	perform_tests_MT()
Figure	Fig 22-24	divide_and_test()
Table	Table 1	chi_squared_test()
Table	Table 2	stats.kstest()
Table	Table 3	linear_complexity_test()
Table	Table 4	append_results()
Table	Table 5	monobit_test()
Table	Table 6	second_level_testing()