Introduction to simulations and Monte Carlo methods 25.10.2024

Project nr 1

Paweł Lorek

DEADLINE: TBA (in Moodle)

1 Description of the Project

A description of the terms used below is provided in the following sections. We will denote the pseudorandom number generator as PRNG. Many of the concepts discussed here appear (possibly with slight differences in wording) in the manuscript [3], parts of which are available on Moodle.

- We will consider the results of the following PRNGs (most for self-implementation, described in Section 4). If the seed has not been specified - the choice is yours.
  - $\text{LCG}(13, 1, 5)$
  - $\text{LCG}(2^{10}, 3, 7)$
  - $\text{GLCG}(2^{10}, \{3, 7, 68\})$
  - Excell's one:
  $$u_i = (0.9821u_{i-1} + 0.211327) \bmod 1$$
    a) with seed $u_0 = 0$
    b) with seed $u_0 = 1812433253$
  - RC4(32), consider two versions
    i) Initialize with one key (e.g., some function of current time etc.) and then test the PRGA result, which should be as long as you need.
    ii) Concatenation of shorter (fix a length) PRGA results [1] $K = 0, 1, 2, \dots$

  Additionally, we will consider binary expansions of $\sqrt{2}, e$ and π, the files are available at (a simple file `read_pi_e_sqrt2.py` that "reads" these files is available in Moodle):

  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

[1] it is related to the so-called "key-related attacks", see https://en.wikipedia.org/wiki/Related-key_attack

- **TASK**.
    1) For RC(32) and a minimum of 3 other generators (including min. one that is not described in this document) perform the selected statistical tests. Consider using the χ^2 test (computing the $\hat{\chi}^2$ statistic), the Kolmogorov-Smirnov test (computing thebibliography $\hat{D}_n$ statistic), serial test, birthday spacing test, collision test. Compare the generators by comparing the appropriate p-values.
    2) For number $\pi, e, \sqrt{2}$ perform at least the *frequency monobit test*
    3) In both of the above, be sure to perform also a **second-level testing**, described in Section 3.2.

  In point 1) above, use at least 3 tests, including min. 1 test that is not described in this document. Consider one of the tests available in the `DieHarder` package, described at:

  https://sites.google.com/site/astudyofentropy/background-information/the-tests/dieharder-test-descriptions

  or they are contained in `Nist Test Suite`, described in [5] available also at

http://www.math.uni.wroc.pl/~lorek/teaching/files/nistspecialpublication800-22r1a.pdf

Describe everything in a report in .pdf format, attach also a working implementation. The report should explain all the things used (so that someone who did not attend the lecture could understand it) and conclusions.

2 On two statistics

We will briefly describe two tests, χ^2 and the Kolmogorov-Smirnov test. The pseudo-random number generator will be abbreviated as PRNG.

2.1 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 prob. 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, which we will denote as $\chi^2(k-1)$. Built-in libraries can be used to compute p-values.

2.2 Kolmogorov-Smirnov test

Suppose that a random variable X has a continuous distribution with cdf $F_X(x)$. Also suppose we have n observations $X_1, \ldots, X_n$ and we want to test whether they are independent and come from $F_X(x)$. Define *empirical distribution function* of $F_n(x)$ as

$$\hat{F}_n(x) = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}(X_i \leq x).$$

The Gliwienko-Cantelli theorem states that if $X_1, \ldots, X_n$ is a sample from distribution F_X, then

$$\sup_{x\in\mathbb{R}} |\hat{F}_n(x) - F(x)| \to 0, \quad n \to \infty,$$

with probability 1. Moreover, statistics

$$\hat{D}_n = \sqrt{n} \sup_{x\in\mathbb{R}} |\hat{F}_n(x) - F(x)|$$

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].

3 Testing pseudorandom number generators via empirical tests

There are many tests for PRNGs. They can be divided into two categories: **empirical tests** and **theoretical tests**. Theoretical tests require knowledge about the way the generator operates, but the sequence of numbers itself does not have to be generated at all, the good ones are usually hard to find. In turn, empirical tests are performed on the sequences obtained from the PRNG, no knowledge of how they were actually genereted is needed.

Different tests make different assumptions about the input numbers:

a) these are numbers with a distribution on the interval (0,1) (with some fixed precision), let's denote such numbers by $u_1, u_2, \ldots$,

b) these are numbers with a discrete distribution on the set $\{0, 1, \ldots, M-1\}$ (usually M is a power of 2), let's denote such numbers by $y_1, y_2, \ldots$,

c) it is a sequence of bits, i.e., numbers with values $\{0, 1\}$, let's denote such numbers by $b_1, b_2, \ldots$.

In turn, different PRNGs may also output numbers in one of the three above forms as an output. Therefore, it is sometimes necessary to convert between these formats. Suppose $M = 2^k$, the numbers $u_1, u_2, \ldots$ are binary expansions of k, then the conversion between a) and b) is as follows

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

In turn, e.g., converting y_i into a sequence of bits: from y_i we get k bits - it is simply a binary expansion of the number. Similarly - from any k bits we will write the number $y_i \in \{0, 1, \ldots, M-1\}$.

3.1 p-value

In context of $\hat{\chi}^2$: we may test the following hypothesis:

H_0 the observations are independent, each is from category s with probability p_s,

H_1 the observations are not sampled from such distribution.

What is the prob. that if the observations are indeed independent and come from this distribution we would observe something $\geq \hat{\chi}^2(obs)$? It is

$$p = Pr(\chi^2(k-1) > \hat{\chi}^2(obs))$$

(what we read from the tables). The small p-values mean that observing $\hat{\chi}^2(obs)$, or something larger, is small. Most often in testing hypotheses, after setting the confidence level to α (typically 0.05 or 0.01), we reject the hypothesis H_0 if $p \leq \alpha$. In PRNG testing, we usually do not reject the hypothesis, but compare p-values of different PRNGs.

3.2 Empirical tests – second-level testing

In the tests described below, we get one p-value. 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 subsequencess, each of length n. For each subsequence, we calculate the appropriate statistic and its corresponding p-value. Assuming $\mathcal{H}_0$ (that the sequence is from the uniform distribution, depending on the format: on $(0,1)$, on $\{0, \ldots, M-1\}$ or on $\{0,1\}$), the obtained p-values $p_1, p_2, \ldots, p_R$ are independent random variables with the distribution $\mathcal{U}(0,1)$. We are now testing this hypothesis. Following NIST recommendations, we split $[0,1]$ into $s = 10$ equal parts $\mathcal{P} = \{P_i\}_{s \in \{1,\ldots,s\}}$, where

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

Then for $i \in \{1, \ldots, s\}$ we define $E_i = \dfrac{R}{s}$ (expected number of p -values in P_i) and O_i (observed number of p-values in P_i):

$$O_i = |\{j : p_j \in P_i,\ 1 \leq j \leq R\}|.$$

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-value

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

where $\chi^2(s-1)$ is a random variable with the χ^2 distribution with $s-1$ degrees of freedom.

It is recommended to generate $R = 1000$–10000 of seqeunces of length $n = 2^{20}$–2^{32}.

3.2.1 Frequency monobit test

Suppose we have a bit sequence $b_1, b_2, \ldots, b_i \in \{0.1\}$. We convert the sequence to $x_1, x_2, \ldots$ with values $\{-1, 1\}$ via $x_i = 2b_i - 1$. Our test statistic is

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

If $b_1, b_2, \ldots$ are realizations of independent random variables uniformly distributed on $\{0, 1\}$, then $x_1, x_2, \ldots$ are realizations of independent random variables uniformly distributed on $\{-1, 1\}$, thus the $s_n(obs)$ statistic has (from CLT) approximately $\mathcal{N}(0, 1)$ distribution. Thus, the p-value is

$$p = Pr(|N| > |s_n(obs)|) = 2(1 - \phi(|s_n(obs)|),$$

where N is a standard normal $\mathcal{N}(0, 1)$ random variable, and ϕ is its cdf.

3.2.2 Frequency of pairs test

This test checks the distribution of pairs of elements. Suppose $y_i \in \{0, 1, \ldots, M-1\}$, we split the data into m pairs:

$$(y_1, y_2), (y_3, y_4), \ldots, (y_{2m-1}, y_{2m}).$$

Then we count the number of occurrences of each of the pairs $(q, r), 0 \leq q, r < M$ and we use the χ^2 test with $M^2 - 1$ degrees of freedom.

The test can easily be extended to test any tuples (see manuscript [4] and/or [3]).

3.2.3 Birthday spacing test

Let's write down the birthdays of the n individuals $Y_1, \ldots, Y_n \in \{1, \ldots, k\}$ and put them in a non-decreasing order $Y_{(1)} \leq \ldots \leq Y_{(n)}$. Let's define the spacings

$$S_1 = Y_{(2)} - Y_{(1)}, \ldots S_{n-1} = Y_{(n)} - Y_{(n-1)}.$$

Let K be the number of equal spacings (i.e., the number of j's such that $S_{(j)} = S_{(j-1)}$, where $S_{(1)}, S_{(2)}, \ldots$ a sequence $S_1, S_2, \ldots$ sorted non-decreasingly).

- It turns out that if n is large and $\lambda = n^3/(4k)$ is small, then assuming the uniform distribution of Y_i (hypothesis $\mathcal{H}_0$) the variable K has approximately Poisson distribution with parameter λ. Thus, if we observe $K(obs)$ of equal spacing, then p-value is
$$Pr(K \geq K(obs)|\mathcal{H}_0) \approx 1 - \sum_{j=0}^{y-1} \frac{\lambda^j}{j!} E^{-\lambda}.$$
- Knuth [1] suggests $k = 512 = 2^9$ and $n = 2^{25}$. The probabilities of the number of equal spacings are given in Fig. 1. Using these probabilities a chi-square test can be applied.

s	0	1	2	≥ 3
$Pr(K = s)$	0.368801	0.369035	0.183471	0.078692

Figure 1: Probabilities for the birthday spacing test, $n = 2^{25}, k = 2^9$

3.2.4 Collision test

See [4] or [3].

3.2.5 Poker test

See [4] or [3].

4 Pseudorandom number generators

4.1 LCG(M, a, c)

Linear congruential generator (LCG) changes its state according to the recurrence

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

Initial state: x_0

4.2 GLCG$(M, \{a_i\}_{i=1}^k)$

Generalized LCG changes its state according to the recurrence

$$x_n = (a_1 x_{n-1} + \ldots a_k x_{n-k}) \bmod M$$

Initial state: $x_0, x_1, \ldots, x_{k-1}$

4.3 RC4(m)

RC4 (we will omit (m) in the description) is the so-called stream cipher[2], which can be used as a PRNG. Let's denote $[m] := \{0, 1, \ldots, m-1\}$.
Its so-called *internal state* is (S, i, j), where S is a permutation of $[m]$ and $i, j \in [m]$ are indices. As input, the algorithm takes the key K: L numbers, each from $[m]$ (we can think that the key is an additional parameter of our PRNG). Next:

- KSA (*Key Scheduling Algorithm*) initializes S with an identity permutation, then uses K to change S to another permutation (intended to be textsl random).
- PRGA (*Pseudo Random Generation Algorithm*) returns the items of this permutation while updating it.

KSA and PRGA are shown in Fig. 2. In the original RC4 we have $m = 256$, while the numbers returned by PRGA are " XORed " with the message giving the cryptogram. RC4 used in PRNG mode simply returns PRGA output.

```
KSA($K$)

for $i := 0$ to $m-1$ do
    $S[i] := i$
end for

$j := 0$
for $i := 0$ to $m-1$ do
    $j := j + S[i] + K[i \bmod L]$
    swap($S[i], S[j]$)
end for
$i, j := 0$
```

```
PRGA

while $r \in \mathcal{N}_+$ do
    $i := i + 1$
    $j := j + S[i]$
    swap($S[i], S[j]$)
    $Y_r \leftarrow S[S[i] + S[j]]$
end while
```

Figure 2: RC4: KSA i PRGA. All additions are computed mod m.

In a **second-level testing** approach and in version ii) (where one is supposed to use keys $K = 0, 1, \ldots$) it may be done in the following way: we run the algorithm R times for consecutive keys $K = 0, 1, \ldots, R-1$, each returning a sequence of length n.
(fixed key K should be the sequence $K[0], \ldots, K[L]$ of numbers from $\{0, \ldots, 31\}$ (for $m = 32$)).

Remark. *RC4 with $m = 256$ was used heavily until recently. If in KSA we replace the line $j := j + S[i] + K[i \bmod L]$ with $j :=$ **random**(m), then this algorithm can be thought of as card shuffling. In the i-th step, we swap the card in position i with a card in a random position. This shuffling scheme is known and it is called Cyclic-To-Random shuffle. It is known that $O(m \lg m)$ steps are needed to get a uniform permutation. However, the KSA only makes m steps, which is one of its serious shortcomings.*

[2]https://en.wikipedia.org/wiki/Stream_cipher

References

[1] D. E. Knuth. *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms,*. Addison-Wesley, 1997.

[2] P. L'Ecuyer, R. Simard, TestU01: A c library for empirical testing of random number generators, *ACM Trans. Math. Sofw.* 33(4), 2007.

[3] P. Lorek, T. Rolski, *Theory and practice of Monte Carlo methods*, *(manuscript under preparation)*, 2023.

[4] T. Rolski, *Symulacje stochastyczne i teoria Monte Carlo*, skrypt UWr, 2017. `http://www.math.uni.wroc.pl/~rolski/Zajecia/sym.pdf`

[5] 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, `https://www.nist.gov/publications/statistical-test-suite-random-and-pseudorandom-number-generators-cryptographic`