Almlöf et al. *EPJ Quantum Technology* (2023) 10:1
https://doi.org/10.1140/epjqt/s40507-022-00158-7

EPJ Quantum Technology
a SpringerOpen Journal

RESEARCH Open Access

Creating and detecting specious randomness

Jonas Almlöf[1*], Gemma Vall Llosera[1], Elisabet Arvidsson[2] and Gunnar Björk[2]

*Correspondence: jalml@kth.se
[1] Ericsson AB, Stockholm, Sweden
Full list of author information is available at the end of the article

Abstract

We present a new test of non-randomness that tests both the lower and the upper critical limit of a χ^2-statistic. While checking the upper critical value has been employed by other tests, we argue that also the lower critical value should be examined for non-randomness. To this end, we prepare a binary sequence where all possible bit strings of a certain length occurs the same number of times and demonstrate that such sequences pass a well-known suite of tests for non-randomness. We show that such sequences can be compressed, and therefore are somewhat predictable and thus not fully random. The presented test can detect such non-randomness, and its novelty rests on analysing fixed-length bit string frequencies that lie closer to the *a priori* probabilities than could be expected by chance alone.

Keywords: Randomness; Randomness test; Data compression

1 Introduction

Randomness is a resource that is getting increasingly important, such as in scientific or engineering simulations, in statistical sampling, in probabilistic computation and in cryptography (both classical and quantum). Randomness is typically generated from hardware processes, which are considered nondeterministic, or by algorithms which are pseudorandom [1, 2]. A pseudorandom number generator uses a short input string, a "seed", to produce a longer, seemingly random output string by applying a one-way function and an iterative algorithm. However, the process produces a deterministic sequence, as every specific seed produces a specific outcome.

Another means of generating randomness is to use a so-called random number generator (RNG), or entropy source. These are typically based on physical devices exhibiting some stochastic output, such as the thermal noise in electronic devices [3] or the amplitude in chaotic oscillators [4]. In this manner, RNGs are nondeterministic.

The ultimate source of randomness is believed to be found in quantum devices. As far as we know, the collapse (by measurement) of an equal superposition between two quantum states will result in a fundamentally unpredictable outcome (between the two possibilities). In fact, some of the random generators on the market are based on this principle, where a singe light particle (a photon) is prepared in a superposition between taking one path or another, e.g. by passing through a balanced beam splitter. Subsequently the path (reflection or transmission) is measured by a photo detector in the path, and the outcome determines the random bit value [5]. For each passing photon a random bit is generated.

A third way of generating random strings, having elements of both computational and physical randomness, are e.g. the built-in pseudorandom number generator of the Linux operating system. It uses physical, real-time, operating system events (such as mouse and keystrokes) as a seed of randomness and then applies a hash-function on the seed to provide a pseudo random string and provide feedback to update the seed. However, questions have been voiced over how secure this apparent randomness is when used in cryptographic protocols [6].

Unfortunately, physical devices are susceptible to imperfections, so even with the best intention and with the use of quantum physics, a seemingly random string may not be completely random. Therefore, a number of tests, a so-called test suite, for non-randomness of binary strings have been devised by the US National Institute of Standards and Technology (NIST), see [7]. As the world is predominantly digital when it comes to data handling and communication, the NIST test suite only considers randomness in the context of binary strings (ex. 10101000), and so will we in what follows. A sufficiently long[1] sample string from a good source of random binary digits should pass the whole suite of tests. However, it was pointed out in e.g., [8] that constituent tests should ideally be independent.

The NIST tests investigate different aspects of non-randomness of long binary strings, such as predominance of certain sub-strings (in the simplest such case, that the number of zeros and ones in the string are not as equal as one would expect) or if the string has periodic features. At least one of these tests, *Maurer's "Universal Statistical" Test* utilises the fact that the data bits generated should not be possible to compress, i.e., their entropy is maximal.

Indeed, an important quality of a series of random numbers that the NIST test suite allegedly tests is that each symbol is independent of the others, so that the next bit cannot be predicted from the previous one, or the following ones. Thus, unpredictability is an important quality of randomness, moreover it is also essential for the inability to compress data, i.e., incompressibility.

In this work, it is shown that a particular type of string passes the NIST test suite, and in particular Maurer's test, but at the same time the bit string is constructed to have some degree of predictability, or equivalently, compressibility. An alternative test to demonstrate that such types of strings are compressible, and hence not perfectly random, is presented.

2 Specious randomness

Data compression, i.e., the act of representing a given string of data bits so that they take up less space without losing any of the information from the original representation, can be performed if patterns are present in the data. As a simple pattern model, let us consider so-called *n*-grams, see [9], i.e., blocks of bit symbols with a fixed length *n*. For illustration, first consider the string 1010101001 and count the number of 1-grams, i.e., the number of "0" and "1" in the string. We find that the sequence has exactly the same observed number

[1]Common tests of randomness have different sample length requirements, typically between 10^5 – 10^9 bits, depending on the complexity of the patterns studied.

Table 1 Frequencies and estimated a priori probabilities for some small n-grams for the string 1010101001. When there is a large variation between the frequencies for each n-gram, the original string can be compressed using Huffman coding

n	n-gram	frequency	p_n
1	0	5	0.5
	1	5	0.5
2	00	0	0.0
	01	1	0.2
	10	4	0.8
	11	0	0.0

of occurrences, namely five. Our best guess is that if the *a priori* probabilities for each of these blocks are equal, the string cannot be compressed, see [9]. However, if we consider 2-grams instead, a well-known compression algorithm called Huffman coding [10] would analyse the frequencies for the four 2-grams according to Table 1. As a final step, the algorithm will perform the replacements "10" → "0", and "01" → "1". Then, after replacement, the original string can be encoded as "00001" using the new symbols, thus the string has become smaller, while maintaining the original information. Here, we ignored the fact that the code book also requires space, but in a long data string, for small n, the size of the code book can be ignored, as its relative size asymptotically approaches zero as the data string grows. The example illustrates that although a string can be incompressible for a small n, it may be compressible for a larger n.

The reasoning we used in the example assumed that the frequencies we observe can be used as best point estimates of the *a priori* probabilities, and thus we should abandon the idea of compression when these are equal. But what if, by some mechanism, *the observed frequencies are abnormally equal?* In fact, there are many examples of this in Nature but also in processes where one, artificially, want to mimic randomness, e.g., in the shape of a deck of cards. The randomness we can extract from this deck is artificial in the sense that when almost all the cards in the deck are drawn, the remaining cards are predictable. They are predictable because the observed frequencies of values and colours in the whole deck of cards are abnormally equal. We will in what follows call this type of artificial randomness *specious randomness*, alluding to the deceitful properties which will lead many entropy estimators into believing that the sequence is random.

2.1 A method for generating specious randomness

2.1.1 Binary number systems

Let g_N denote the set of all binary sequences of length N, i.e.,

$$g_N = \{e_i\}, \quad i \in 1, \dots, 2^N. \tag{1}$$

We will denote an *ordered* sequence of the elements in g_N a *number system*, S_N^i, where i can take any value $1, \dots, 2^N!$.

For example, S_1^i denotes the number systems of block length 1,

$$S_1^1 = (0, 1), \qquad S_1^2 = (1, 0), \tag{2}$$

i.e., S_1^i denotes the most primitive number systems.

2.1.2 Kronecker concatenation

A new set g_2 can be generated by a so-called *Kronecker concatenation* of two g_1 sets, i.e.,

$$g_2 = K(g_1, g_1) = \{0,1\} \otimes \{0,1\} = \{00, 01, 10, 11\}. \tag{3}$$

Since the defining property of a number system is that each element occurs exactly once, we note that any number system stemming from a g_{MQ} set, i.e.,

$$g_{QM} = K\big(\overbrace{g_M, g_M, \dots}^{Q \text{ times}}\big), \tag{4}$$

is *M-skip-balanced*. This notion means that if we take an arbitrary number system stemming from a g_{MQ} set and concatenate all elements (forming an $MQ2^{MQ}$ bit sequence), then non-overlapping blocks of length M will occur an equal amount of times. We shall also interest ourselves in another related property; if we instead count *all* possible sub-strings (even overlapping) of fixed length n and find that such strings are equally frequent, we will call such sequences *n-balanced*. Note that an n-balanced string does not need to be *n-skip-balanced* and vice versa.

It follows that, any number system S_N^i is n-skip-balanced when n divides N, e.g., all 12-bit number systems are 1-, 2-, 3-, 4-, 6- and 12-skip-balanced.

2.2 Existing tests do not detect specious randomness

We have used the NIST test suite [7] to test different binary sequences by concatenating elements of 13-, 16-, 17-, 24- and 25-bit number systems in random order. The number of used bits for each file were 100 kbit, 1 Mbit, 1 Mbit, 700 Mbit and 700 Mbit respectively.

The NIST test suite uses so called null hypothesis significance testing, where p-values are reported, i.e., the probability of obtaining test results at least as extreme as the results actually observed, under the assumption that the null hypothesis is correct. The null hypothesis is that the bit sequence being tested is random, conversely, the alternative hypothesis is that the sequence is not random.

For every test the bit sequence was tested against, the NIST tests determine acceptance or rejection of the null hypothesis, in our case a critical value of $p = 0.01$ was chosen. In other words, a passing sequence can be considered random with a probability of 0.99.

The NIST tests report both p-values and a proportion of passing sequences. Table 2 shows the mean p-values and mean proportion values for the different binary sequences tested here (13-, 16-, 17-, 24- and 25-bit random number systems). The mean p-value and the mean success proportion is calculated by averaging the individual p-values and the passing rates, obtained after the computation of all the tests in the NIST suite, for which

Table 2 Mean p-values and mean proportion of pass events (out of a maximum of 10) for all data sources

Data source	<p-value>	<proportion>
25 bit	0.5045	9.86
24 bit	0.51	9.88
17 bit	0.4789	9.81
16 bit	0.5117	9.82
13 bit	0.5150	9.82

Table 3 Proportion of passes for each individual test in the NIST test suite. The minimum pass rate is 8

	13 bit	16 bit	17 bit	25 bit	2 × 24 bit**
Frequency	10/10	10/10	10/10	9/10	10/10
Block Frequency	9/10	10/10	10/10	10/10	10/10
Cumulative Sums 1	10/10	10/10	10/10	9/10	10/10
Cumulative Sums 2	10/10	10/10	10/10	9/10	10/10
Runs	10/10	10/10	10/10	10/10	10/10
Longest Run	10/10	10/10	10/10	9/10	10/10
Rank	10/10	10/10	10/10	10/10	9/10
FFT	10/10	10/10	10/10	10/10	9/10
Non Overlapping Template	9.82/10*	9.80/10*	9.79/10*	9.89/10*	9.88/10*
Overlapping Template	10/10	10/10	10/10	8/10	10/10
Universal	0/10	0/10	0/10	10/10	10/10
Approximate Entropy	9/10	8/10	5/10	10/10	10/10
Random Excursions	–	–	–	9/9*	9.00/9*
Random Excursions Variant	–	–	–	9/9*	9.00/9*
Serial 1	10/10	10/10	10/10	10/10	10/10
Serial 2	10/10	10/10	10/10	10/10	10/10
Linear Complexity	9/10	10/10	10/10	10/10	10/10

*Average of multiple versions of the specific test.

**Two 24-bit systems with different seeds were concatenated.

the p-value was larger than 0.01. Note that the *Universal Maurer's* test did not execute for the 13-, 16-, 17-bit random number systems due to non-matching input size requirements.

Table 3 shows the proportion of sequences that pass each constituent statistical test in the NIST test suite.

The proportion of passed tests is between 8-10/10 thus the null hypothesis is favoured for the 24- and 25-bit man-made random number sequences. For the 13-, 16- and 17-bit number system sequences, some tests failed, in particular the *Approximate Entropy* test, and other did not execute due to an insufficient sequence length.

3 Compressibility of specious randomness

3.1 Number system compressibility

Suppose we have a number system S_N^i, that is, a string of binary digits, $N2^N$ digits long, which if it is divided into 2^N sequential sub-strings, each N binary digits long, every possible sub-string will occur exactly once.

In such a number system string we can always predict the last sub-string given that we know the $2^N - 1$ sub-strings preceding it. Hence, in transmitting a number-system string, we need only to transmit $2^N - 1$ of the 2^N sub-strings to fully specify the whole string. Thus, it can be compressed into a shorter string. The length ratio η between the two strings will be

$$\eta_1 = \frac{2^N - 1}{2^N} = 1 - 2^{-N}. \tag{5}$$

However, we should be able to do substantially better than that. Having no other information about the string than that it is a number system, we get, on average, the information

$$i[m, N] = -\log_2\left(\frac{1}{2^N + 1 - m}\right) \tag{6}$$

when receiving the *m*th sub-string. This is because by then, the choice of possible (and equally likely) sub-strings is only $2^N + 1 - m$. Hence, the average information obtained over all 2^N sub-strings is

$$I_{Av} = 2^{-N} \sum_{m=0}^{2^N-1} \log_2\left(2^N - m\right). \tag{7}$$

A more relevant measure is the average, obtained information per binary digit which is

$$\eta_2 = \frac{1}{N2^N} \sum_{m=0}^{2^N-1} \log_2\left(2^N - m\right). \tag{8}$$

In principle, η_2 denotes the compressibility of the string. However, the fact that in principle the string can be compressed to η_2 times its original length gives no clue as to how such a compression algorithm should be designed.

3.2 A simple number-system compression algorithm

A possible compression algorithm, inspired by Huffman coding [10], that works on any arbitrary $N2^N$ binary digits long number-system string is the following: Transmit the first 2^{N-1} sub-strings, each one N binary digits long, just as they are. Since we know that the whole string to be sent is a number system, we order all the 2^{N-1} "missing" sub-strings in ascending order, and renumber them using the $N-1$ binary number long strings $0\ldots0$, $0\ldots1$, …, $1\ldots1$. Subsequently send the first 2^{N-2} of these relabeled (and shortened) $N-1$ binary number sub-strings. Then order all the 2^{N-2} still "missing" original sub-strings in ascending order and relabel them with $N-2$ binary number long strings. Repeat the process until there are only two of the original sub-strings left to transmit. Order these in ascending order and call the smallest of them 0 and the other 1. Send the bit corresponding to the second to last sub-string. The last sub-string need not be sent since it can be deduced from the received string.

Let us exemplify the compression algorithm through a $N = 3$ number sequence 24 binary digits long. We choose the string to be compressed to be

$$S = 000\ 111\ 101\ 001\ 011\ 100\ 010\ 110, \tag{9}$$

where we have separated the string into eight three-digit sub-strings, partly to highlight that the string is a number sequence (the numbers 0 to 7 in binary notation, and unsorted). We start the algorithm from the left. The first half of the compressed string is identical to the leftmost half of S, namely 000 111 101 001.

To compress the second half of S we order the remaining four sub-strings in ascending order and associate them with their order number using two bits:

$$010 \leftrightarrow 00, \tag{10}$$

$$011 \leftrightarrow 01, \tag{11}$$

$$100 \leftrightarrow 10, \tag{12}$$

$$110 \leftrightarrow 11. \tag{13}$$

Table 4 Compression ratios η_1, η_2, and η_3 are listed for some N

N	η_1	η_2	η_3
1	0.5	0.5	0.5
2	0.75	0.573	0.625
3	0.825	0.637	0.708
4	0.937	0.691	0.766
5	0.969	0.735	0.806
10	0.999	0.856	0.900
100	1.000	0.986	0.990

In the string S, the sub-strings 011 and 100 appear at position five and six from the left. In the compressed string we replace them with the numbers 01 and 10 according to the table above. The appended compressed string will thus become 000 111 101 001 01 10.

At position seven and eight in S we find the sub-strings 010 and 110. We assign to them their ascending order number with one bit, hence $010 \leftrightarrow 0$ and $110 \leftrightarrow 1$. We append the bit value associated to the seventh sub-string in S to the end of C and get

$$C = 000\ 111\ 101\ 001\ 01\ 10\ 0. \tag{14}$$

This compressed string uniquely encodes S. The eighth sub-string of S is the only three binary digit string not represented in C. Therefore, it can be omitted from C without losing any information.

The compressed string C is 17 binary digits long compared to S that is 24 digits long. The relative length of C is thus $17/24 \approx 0.708$.

Thus, instead of sending a whole $N2^N$ binary number long string, a re-coded string will "only" have the length

$$1 + \sum_{m=0}^{N-2}(N-m)2^{N-1-m} = 2^N(N-1) + 1 \quad \forall N \geq 2, \tag{15}$$

and the length one for $N = 1$. The relative length, which is a measure of the obtained compression is thus

$$\eta_3 = \frac{(N-1)2^N + 1}{N2^N} = 1 - \frac{1}{N} + \frac{1}{N2^N} \quad \forall N \geq 2. \tag{16}$$

The tabulated compression ratios η_1, η_2, and η_3 for some N are found in Table 4.

4 A new statistical test of non-randomness

The test suite from NIST [7] tests *non-randomness*, i.e., these tests checks a binary sequence for different abnormal properties. Since there are many such abnormalities that could be present in a binary sequence, a truly random binary string should, in principle, pass all such tests. Since there is no known universal test for randomness, the best we can do is to try to test for non-randomness, i.e., our test hypothesis is formulated in such a way that a probability for such abnormalities can be calculated.

Below we will present a new test that performs a double sided χ^2 test to the uniform distribution, i.e., also checking the lower limit. It was understood by R. Fisher [11] that statistical experiments are unlikely to yield observed frequencies too close to the expected ones. He used this method to show that Mendel's genetic experiments with garden peas [12, 13] showed observed relative frequencies closer to the *a priori* (expected) probabilities than were likely by chance alone. In the presented method, the lower range of the test statistics checks if the sub-strings in the sample follows the *a priori* probability closer than expected by chance. If so, the test fails since it gives rise to predictability. The upper limit checks if the observed frequencies are compatible with the hypothesis that all *a priori* probabilities are equal.

4.1 The chi-squared test

The χ^2 test is often used to verify if a given expected distribution is compatible with observations. Its probability density function is

$$f(k,x) = \frac{1}{2^{k/2}\Gamma(k/2)} x^{k/2-1} e^{-x/2}, \tag{17}$$

where k is the number of degrees of freedom and Γ is the Gamma function [14].

For a discrete uniform distribution, we can calculate the χ^2 *statistic*

$$\chi^2 = \sum_i \frac{(\nu_i - \mathcal{N}/m)^2}{\mathcal{N}/m}, \tag{18}$$

where ν_i is the number of observations in bin i, $\mathcal{N}$ is the total number of observations and $m = 2^n$ is the number of possible n-grams of length n. The assumed *a priori* probabilities for each of these is $1/2^n$. Thus the test is specific for a particular $n \in 1, 2, 3, \dots$.

In order to determine the probability to get some lower value a than the test statistic χ^2 we calculate the cumulative distribution function

$$P\bigl(k, a \leq \chi^2\bigr) = \int_{-\infty}^{\chi^2} f(k,x)\,dx. \tag{19}$$

We now denote p as the probability that a distribution was deemed not random, while in fact it was random (this can happen by chance due to statistical variance). Then we define a lower critical bound $\chi^2_{p/2}$ as the value of the test statistic χ^2 when the probability $P(k, a \leq \chi^2) = p/2$ and an upper critical bound $\chi^2_{1-p/2}$ as the value of χ^2 when the probability $P(k, a \leq \chi^2) = 1 - p/2$.

The degrees of freedom k in this case is $2^n - 1$, and thus $\chi^2_{p/2}$ ($\chi^2_{1-p/2}$) indicates the lower (higher) limit for the test statistic χ^2 when we accept the hypothesis that the data is random. If the value is lower than $\chi^2_{p/2}$ we will reject that the data is random on the basis that its distribution is too close to the a priori uniform probabilities. If the value is higher than $\chi^2_{1-p/2}$, we reject the hypothesis because the observations are not compatible with the assumed uniform a priori distribution. The total probability that a random series will fail the test is thus p.

In Table 5 we list the results from the testing of two concatenated 24-bit number systems, i.e., our example of specious randomness from Sect. 2.2. The file is $2 \times 24 \cdot 2^{24}$ bits ($\approx$800 Mbit) long and different random seeds were used for the two systems to determine their respective element order. A file with the entire string can be found at [15]. From the Table, we see that from the 48 values tested, only 2 will pass the test, i.e. have an observed χ^2 value between $\chi^2_{p/2}$ and $\chi^2_{1-p/2}$, where $p = 0.01$. All the other 46 values indicate

Table 5 For the specious randomness file based on two concatenated 24-bit number systems, the probability to get some value a, lower than the cumulative distribution function Eq. (19) is tabulated for different n-grams ≤ 24, using non-overlapping counting (column 2). We know that for $n = 1, 2, 3, 4, 6, 8, 12$ and 24, all n-grams occur equally frequently, therefore the sequence is skip-balanced for those n-values. In column 3 we tabulate the corresponding result when overlapping n-gram counting was used

n	$P(2^n - 1, a \leq \chi_1^2)$ (non-overlapping)	$P(2^n - 1, a \leq \chi_2^2)$ (overlapping)
1	0	0
2	0	0.00050
3	0	0.000034
4	0	0.0000000080
5	0.13	$<10^{-15}$
6	0	$<10^{-15}$
7	0.0092	$<10^{-15}$
8	0	$<10^{-15}$
9	0.000091	$<10^{-15}$
10	0.00024	$<10^{-15}$
11	0.034	$<10^{-15}$
12	0	$<10^{-15}$
13	0.0041	$<10^{-15}$
14–24	$<10^{-6}$	$<10^{-15}$

Table 6 For the specious randomness file based on a single 25-bit number system, the probability to get some value a, lower than the cumulative distribution function Eq. (19) is tabulated for different n-grams ≤ 25, using non-overlapping (column 2) and overlapping (column 3) counting of n-grams. We know that for $n = 1, 5$ and 25, all n-grams occur equally frequently, therefore the sequence is skip-balanced for those n-values

n	$P(2^n - 1, a \leq \chi_1^2)$ (non-overlapping)	$P(2^n - 1, a \leq \chi_2^2)$ (overlapping)
1	0	0
2	0.62	0.0061
3	0.020	0.000085
4	0.067	0.000000037
5	0	0.00000000010
6	0.37	$<10^{-15}$
7	0.048	$<10^{-15}$
8	0.012	$<10^{-15}$
9	0.18	$<10^{-15}$
10	$<10^{-15}$	$<10^{-15}$
11	0.010	$<10^{-15}$
12	0.0026	$<10^{-15}$
13	0.11	$<10^{-15}$
14	0.000025	$<10^{-15}$
15	$<10^{-15}$	$<10^{-15}$
16–25	$<10^{-10}$	$<10^{-15}$

non-randomness. To put the number $p/2 = 0.005$ in perspective, recall that it is half of the p-value used when performing the NIST tests in Sect. 2.2, since we now examine both the lower and upper critical value $\chi^2_{p/2}$ and $\chi^2_{1-p/2}$. In Table 5 we can also see that the observed test statistics for $n = 1 - 24$ are all lower than the upper-critical value $\chi^2_{1-p/2}$, both for non-overlapping and overlapping n-gram counting. In Table 6 a single 25-bit number system was tested with similar result for overlapping n-grams, however with a larger proportion of "pass" results for non-overlapping n-grams.

If this test is to be used together with other tests, it was pointed out in [8] that combined tests should be independent. The novelty of our test lies in testing the lower critical limit, while the upper critical limit is examined in other tests, e.g., in the *Frequency* sub-test of the NIST suite. The presented test could in light of this, easily be modified so that it only examines a lower critical value χ_p^2, i.e., performs a one-sided test. Such a modification could help make the presented test independent, or even better; *anti-correlated*, of other tests in a test suite, making the combined set of tests better at detecting non-randomness.

5 Conclusions

We have demonstrated a method for constructing binary strings $N2^N$ digits long, consisting of one instance of every binary number with N digits, ordered in a random fashion. We have shown that if N is sufficiently large, which in practice means $N > 12$, such a string will typically pass the NIST suite of tests for randomness (or more accurately expressed, fail the tests for non-randomness).

We have also shown a method to compress such a string. This shows that the string has a certain amount of compressibility, or equivalently, predictability or non-randomness.

A test to reveal the non-randomness of the string is subsequently proposed. It consists of looking at the frequency of n-grams for different n:s. This is already done in some of the NIST-suite tests, but evidently these tests look for unusually large deviations from a nominally even distribution of frequencies. In our test we also examine the "opposite", namely if the frequencies are suspiciously similar (or even identical). The extremes of either case signal that the tested string is non-random.

Acknowledgements
The authors would like to thank Andrew Rukhin, Meltem Sonmez Turan (National Institute of Standards and Technology), Johan Håstad (Royal Institute of Technology,KTH) for helpful comments about randomness tests and Remi Robert (Ericsson AB) for providing valuable feedback.

Funding
Not applicable.

Availability of data and materials
Files with specious random data can be found online, see [15].

Declarations

Ethics approval and consent to participate
No humans or animals were involved in the research which is purely theoretical. Hence, ethical approval and consent to participate are irrelevant.

Consent for publication
No human subjects were involved in the research which is purely theoretical. All authors and their employers agree to publish the results.

Competing interests
The authors declare no competing interests.

Author contributions
All authors wrote the main manuscript text. J.A. and G.B. constructed the specious randomness files, G. V.-L. and E. A. prepared Tables 2–3. E.A. executed the NIST test suite programs. J.A. suggested the new randomness test, G.B. suggested the compression method in section "Number system compressibility". All authors reviewed the manuscript.

Author details
[1]Ericsson AB, Stockholm, Sweden. [2]Department of Applied Physics, KTH Royal Institute of Technology, Stockholm, Sweden.

Received: 19 October 2022 Accepted: 28 December 2022 Published online: 30 January 2023

References

1. Yao AC. Theory and applications of trapdoor functions. In: 23rd IEEE symposium on foundations of computer science. 1982. p. 80–91.
2. Blum M, Micali S. How to generate cryptographically strong sequences of pseudo-random bits. SIAM J Comput. 1984;13:850–64.
3. Millenson JR, Sullivan GD. A hardware random number generator for use with computer control of probabilistic contingencies. Behav. Res. Meth. Instrum.. 1968;1:194–6.
4. Reidler I, Aviad Y, Rosenbluh M, Kanter I. Ultrahigh-speed random number generation based on a chaotic semiconductor laser. Phys Rev Lett. 2009;103:024102.
5. Jennewein T, Achleitner U, Weihs G, Weinfurter H, Zeilinger A. A fast and compact quantum random number generator. Rev Sci Instrum. 2000;71:1675.
6. Gutterman Z, Pinkas B, Reinman T. Analysis of the Linux random number generator. In: 2006 IEEE symposium on security and privacy (S&P'06). 2006. p. 15–385.
7. Rukhin A, Soto J, Nechvatal J, Smid M, Barker E, Leigh S, Levenson M, Vangel M, Banks D, Heckert A, Dray J, Vo S. A statistical test suite for random and pseudorandom number generators for cryptographic applications. https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-22r1a.pdf. 2010. Accessed 25 Jun 2020.
8. Sönmez Turan M, Doğanaksoy A, Boztaş S. On independence and sensitivity of statistical randomness tests. Berlin: Springer; 2008. p. 18–29.
9. Shannon CE. A mathematical theory of communication. Bell Syst Tech J. 1948;27:379–423, 623–656.
10. Huffman DA. A method for construction of minimum redundancy codes. Proc IRE. 1952;40:1098–101.
11. Fisher RA. Has Mendel's work been rediscovered? Ann Sci. 1936;1:115–37.
12. Mendel JG. Versuche über pflanzenhybriden. Verhandlungen des naturforschenden Vereines in Brünn. 1866;Bd. IV für das Jahr 1865:3–47.
13. Druery CT, Bateson W. Experiments in plant hybridization. J. R. Hortic. Soc.. 1901;26:1–32.
14. Wikipedia contributors. Gamma function—Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Gamma_function. 2021. Accessed 30 Jun 2020.
15. Almlöf J, Vall Llosera G, Arvidsson E, Björk G. Specious randomness data sequences for various number systems. 2021. http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-298306.