Randomness Tests: Theory and Practice

Alexander Shen(✉)

LIRMM, University of Montpellier, CNRS, Montpellier, France
alexander.shen@lirmm.fr

To Yury Gurevich, with whom we have had a lot of interesting and sometimes heated discussions on many topics, including randomness.

Abstract. The mathematical theory of probabilities does not refer to the notion of an *individual* random object. For example, when we toss a fair coin n times, all 2^n bit strings of length n are equiprobable outcomes and none of them is more "random" than others. However, when testing a statistical model, e.g., the fair coin hypothesis, we necessarily have to distinguish between outcomes that contradict this model, i.e., the outcomes that convince us to reject this model with some level of certainty, and all other outcomes. The same question arises when we apply randomness tests to some hardware random bits generator.

A similar distinction between random and non-random objects appears in algorithmic information theory. Algorithmic information theory defines the notion of an individual random sequence and therefore splits all infinite bit sequences into random and non-random ones. For finite sequences there is no sharp boundary. Instead, the notion of *randomness deficiency* can be defined, and sequences with greater deficiency are considered as "less random" ones. This definition can be given in terms of randomness tests that are similar to the practical tests used for checking (pseudo)random bits generators. However, these two kinds of randomness tests are rarely compared and discussed together.

In this survey we try to discuss current methods of producing and testing random bits, having in mind algorithmic information theory as a reference point. We also suggest some approach to construct robust practical tests for random bits.

1 Testing a Statistical Hypothesis

Probability theory is nowadays considered as a special case of measure theory: a random variable is a measurable function defined on some probability space that consists of a set Ω, some σ-algebra of the subsets of Ω, and some σ-additive measure defined on this σ-algebra. A random variable determines a probability distribution on the set of possible values.

Supported by ANR-15-CE40-0016-0 RaCAF project.

A. Blass et al. (Eds.): Gurevich Festschrift, LNCS 12180, pp. 258–290, 2020.
https://doi.org/10.1007/978-3-030-48006-6_18

When probability theory is applied to some "real world" case, it provides a statistical hypothesis that is a mathematical model of the process. For example, for n trials and the fair coin the corresponding model is the uniform distribution on the set $\mathbb{B}^n$ of all possible outcomes, i.e., on all n-bit binary strings. Each string has probability 2^{-n}. The set $\mathbb{B}^n$ can be considered as a probability space, and ith coin tossing is represented by a random variable ξ_i defined on this space: $\xi_i(x_1x_2\ldots x_n) = x_i$.

Having a mathematical model for a real-life process, we need some way to check whether this model is adequate or not. Imagine that somebody gives us a coin, or a more advanced random bits generator. This coin can be asymmetric, and the bit generator can be faulty. To check whether this is the case, we need to perform some experiment and look at its results. Assume, for example, that we toss a coin and get 85 heads in a row, as it happened to Rosencrantz and Guildenstern in Stoppard's play [34, Act 1]. Should we reject the fair coin hypothesis? Probably we should—but how can we justify this answer? One can argue that for a fair coin such an outcome is hardly possible, since its probability is negligible, namely, equals 2^{-85}. However, any other sequence of 85 heads and tails has the same negligible probability—so why this reasoning cannot be applied to any other outcome?

To discuss this problem in a more general case, let us introduce suitable terminology. Consider some set X of possible outcomes. We assume that X is finite. Fix some *statistical model* P, i.e., a hypothetical probability distribution on X. A *randomness test* is an event $T \subset X$ that has small probability according to P. If an experiment produces an outcome that belongs to T, the test is not passed, and we may reject P. This approach has two main problems. The first problem, mentioned earlier, is that in most cases every individual outcome $x \in X$ has negligible probability, so the singleton $\{x\}$ is a test that can be used to reject P. We discuss this problem later. Now let us comment on the other problem: how to choose the threshold value for the probability, i.e., how small should be the probability of T to consider T as a valid randomness test.

In practice, the statistical model is often called the *null hypothesis*. Usually we have some experimental data that, as we hope, exhibit some effect. For example, we may hope that a new drug increases the survival rate, and indeed we see some improvement in the experimental group. However, this improvement could be just a random fluctuation while in fact the survival probability remains unchanged. We formulate the null hypothesis based on the old value of the survival probability and then apply some test. The rejection of the null hypothesis means that we do not consider the data as a random fluctuation, and claim that the new drug has at least some effect.[1]

In this approach, the choice of the threshold value obviously should depend on the importance of the question we consider. Any decision based on statistical considerations is inherently unreliable, but an acceptable level of this unreliability depends on the possible consequences of a wrong decision. The more important the consequences are, the smaller threshold for statistical tests is

[1] Of course, in practice we want also to be convinced that this effect is beneficial.

needed. The choice of the threshold value is often debated. For example, there is a paper [2] signed by 72 authors that proposes "to change the $\langle\ldots\rangle$ threshold for statistical significance from 0.05 to 0.005 for claims of new discoveries". It may look ridiculous—obviously both the old threshold and the new one are chosen arbitrarily—but it reflects the existing situation in natural sciences. Other people point out that fixing a threshold, whatever it is, is a bad practice [1].

2 Randomness Tests

Now let us address the other problem mentioned above. After the experiment is made, we can find a set T of very small measure that contains the actual outcome of the experiment, and declare it to be a randomness test. For example, Rosencrantz could toss a coin 85 times, write down the sequence x of heads and tails obtained, and then try to convince Guildenstern that the fair coin hypothesis should be rejected, because the set $T = \{x\}$ is a test that has probability 2^{-85} according to the hypothesis and still the actual outcome is in T.

This argument is obviously wrong, but it is not that easy to say what exactly is wrong here. The simplest—and rather convincing—answer is that the *test should be fixed before the experiment*. Indeed, if Rosencrantz showed some sequence of heads and tails to Guildenstern and *after that* the coin tossing gave exactly the same sequence, this would be a very convincing reason to reject the null hypothesis of a fair coin.

Still this answer is not universal. Imagine that we buy a book called "A Million Random Digits"[2]. We open it and find that it is filled with zeros. Do we have reasons to complain? If we have said "Look, this book is suspicious; may be it contains only zeros" *before opening the book*, then we definitely do—but what if not? Or what if we find out that the digits form the decimal expansion of 8.5π? Note that this hypothesis hardly can come to our mind before we study the book carefully.

Sometimes the experiment is already in the past, so we look at the data already produced, like Kepler did when analyzing the planet observations. Probably, in this case we should require that the test is chosen without knowing the data, but it is (a) more subjective and (b) rarely happens in practice, usually people do look at the data before discussing them.

On the other hand, even the test formulated before the experiment could be dubious. Imagine that there are many people waiting for an outcome of the coin tossing, and each of them declares her own statistical test, i.e., a set of outcomes that has probability at most ε. Here ε is some small number; it is the same for all tests. After the experiment it is found that the outcome fails one of the tests, i.e., belongs to the small set declared by one of the observers. We are ready to declare that the null hypothesis is rejected. Should we take into account that there were many tests? One can argue that the probability to fail at least one of the N tests is bounded by $N\varepsilon$, not ε, so the result is less convincing than the

[2] By the way, one can still buy such a book [27] now (August 2019) for 50.04 euro, or 903.42 euro, if you prefer the first edition, but agree to get a second-hand copy.

same result with only one observer. This correction factor N is often called the *Bonferroni correction* and is quite natural. On the other hand, the observer who declared the failed test could complain that she did not know anything about other observers and it is a completely unacceptable practice if actions of other people beyond her control and knowledge are considered as compromising her findings. And it is difficult to answer in a really convincing way to this complaint.

In fact, this is not only a philosophical question, but also an important practical one. If a big laboratory with thousand researchers uses threshold value 0.05 for statistical significance, then we could expect dozens of papers coming from this lab where this threshold is crossed—even if in fact the null hypothesis is true all the time.

There are no universally accepted or completely convincing answers to these questions. However, there is an important idea that is a philosophical motivation for algorithmic information theory. We discuss it in the next section.

3 "Remarkable" Events as Tests

Recall the example with zeros in the table of random numbers and the corresponding singleton test that consists of the zero sequence. Even if we have not explicitly formulated this test before reading the table, one can say that this test is so simple that it *could* be formulated before the experiment. The fact that all outcomes are heads/zeros is remarkable, and this makes this test convincing.

This question is discussed by Borel [5]. He quotes Bertrand who asked whether we should look for a hidden cause if three stars form an equilateral triange. Borel notes that nobody will find something strange if the angle between two stars is exactly $13°42'51.7''$, since nobody would ask whether this happens or not before the measurement ("car on ne se serait jamais posé cette question précise avant d'avoir mesuré l'angle"). Borel continues:

> La question est de savoir si l'on doit faire ces mêmes réserves dans le cas où l'on constate qu'un des angles du triangle formé par trois étoiles a une valeur *remarquable* et est, par exemple, égal à l'angle du triangle équilatéral ⟨...⟩ Voici ce que l'on peut dire à ce sujet : on doit se défier beaucoup de la tendance que l'on a à regarder comme *remarquable* une circonstance que l'on n'avait pas précisée *avant l'expérience*, car le nombre des circonstances qui peuvent apparaître comme remarquables, à divers points de vue, est très considérable [5, p. 112–113][3]

This quotation illustrates a trade-off between the probability of the event specified by some randomness test and its "remarkability": if there are N events

[3] The question is whether we should have the same doubts in the case where one of the angles of a triangle formed by tree stars has some *remarkable* value, for example, is equal to the angle of an equilateral triangle. ⟨...⟩ Here we could say the following: one should resist strongly to the tendency to consider some observation that was not specified *before the experiment* as remarkable, since the number of circumstances that may look remarkable from different viewpoints is quite significant.

of probability at most ε that are "as remarkable as the test event" (or "more remarkable"), then the probability of the combined test event is at most $N\varepsilon$. In other words, we should consider not an individual test, but the union of all tests that have the same probability and the same (or greater) "remarkability".

The problem with this approach is that we need to quantify somehow the "remarkability". The algorithmic information theory suggests to take into account the Kolmogorov complexity of the test, i.e., to count the number of bits needed to specify the test. More remarkable tests have shorter descriptions and smaller complexity. The natural way to take the complexity into account is to multiply the probability by $O(2^n)$ if the complexity of the test is n, since there is at most $O(2^n)$ different descriptions of size at most n bits and therefore at most $O(2^n)$ tests of complexity at most n.

However, the word "description" is too vague. One should fix a "description language" that determines which test corresponds to a given description, i.e., to a given sequence of bits. Algorithmic information theory does not fix a specific description language; instead, it defines a class of description languages and proves that there are *optimal* description languages in this class. Optimality is understood "up to $O(1)$ additive term": a language L is optimal if for any other language L' in the class there exist a constant c (depending on L') with the following property: if a test T has description of length k via L', it has a description of length at most $k+c$ via L. This implies that two different optimal languages lead to complexity measures that differ at most by $O(1)$ additive term.

Probably this $O(1)$ precision is the best thing a mathematical theory could give us. However, if one would like to define "the gold standard" for valid use of statistical tests, this is obviously not enough, and one should fix some specific description language. It looks like a difficult task and there are no serious attempts of this type. Still one could expect that this language should be domain-specific and take into account the relations and constants that are "naturally defined" for the objects in question. This is discussed in details by Gurevich and Passmore [9]. The authors note that questions about statistical tests and their validity do arise in courts when some statistical argument is suggested as evidence, but there are no established procedures to evaluate statistical arguments.[4] One could also mention a similar (and quite important) case: statistical "fingerprints" for falsified elections. There are many examples of this type (see the survey [31] and references within). Let us mention two examples that illustrate the problem of "remarkable post factum observations". Figure 1 (provided by Kupriyanov [15]) presents the official results of the "presidential elections" in Russia in 2018 and is constructed as follows: for every polling station where both the reported participation rate and the fraction of votes for de facto president of

[4] In this paper some thought experiments and one real story are considered as examples. One of the thought experiments is as follows: the wife of a president of a state lottery turns out to be its winner. Recently I learned that this example is not so far from the real life as one could think: in 2000 BBC reported that "Zimbabwean President Robert Mugabe has won the top prize [about $2600] in a lottery organised by a partly state-owned bank" (http://news.bbc.co.uk/2/hi/africa/621895.stm).

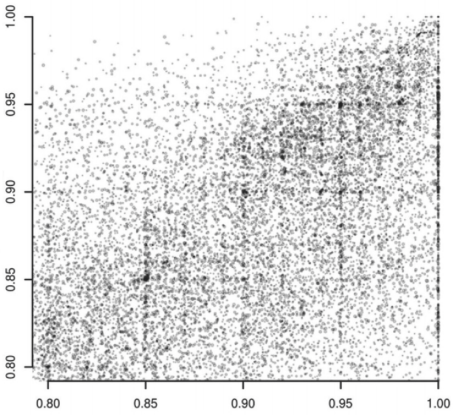


Fig. 1. "Putin's grid" in 2018 [15]. Note that vertical and horizontal lines are formed by data points, they are not the added grid lines.

Russia (Putin) exceed 80%, a corresponding grey point is shown. If several points coincide, a darker/bigger point appears. Looking at the "grid lines" formed by these points (for integer percentages; more visible lines appear for the percentages that are multiples of 5), one probably agrees that such a remarkable grid has negligible probability to appear naturally for any kind of elections. However, it is far from obvious which statistical test should be considered here and how can we quantitatively estimate its complexity and its probability.

Related example is provided by "referendum results" in Crimea (Ukraine). As noted by Alexander Kireev [13], the "official results" in Sevastopol (Crimea) include the following data: total numbers of registered voters (306258), total number of ballots (274101) and the number of "yes for the annexation" votes (262041). The two main ratios (the participation rate and "yes" rate) are suspiciously round: 274101/306258 = 0.895**000**294 and 262041/274101=0.956**00**162. Indeed, in both cases the numerator can be obtained by multiplying the denominator by the integer number of promilles and rounding to the closest integer. Probably most statisticians would agree that this coincidence is remarkable and has very small probability, but it is difficult to agree on a specific quantitative estimate of the corresponding probability after a suitable Bonferroni correction.[5]

[5] A rough estimate is attempted in the survey mentioned above [31, p. 49–50]. It takes into account other information about the case. Both anomalies, the integer grid and round percentages, appeared in earlier "elections", so it is not really fair to call them "post factum observations". For the Crimea's "referendum results" the upper bound for the probability after the correction is estimated as 0.1%.

4 Randomness as Incompressibility

Algorithmic information theory (also called Kolmogorov complexity theory) is outside the scope of this survey[6], but let us mention a few results that have philosophical importance and should be kept in mind when discussing randomness at any level.

Roughly speaking, algorithmic information theory says that *randomness is incompressibility*. More precisely, a binary string looks plausible as an outcome of a fair coin (does not convince us to reject the fair coin hypothesis) if it is incompressible, i.e., if there is no program that produces this string and is much shorter than the string itself. In other words, we

- define Kolmogorov complexity of a string as the minimal length of a program that produces it; in this definition we use some optimal programming language that makes complexity minimal up to an $O(1)$ additive term;
- note that all n-bit strings have complexity at most $n + O(1)$, since a trivial program "print x" has almost the same size as x;
- note that at most 2^{-c} fraction of n-bit strings have complexity less than $n - c$, so this is a very small minority for non-negligible values of c; we treat members of this minority as non-random strings.

This approach is consistent with what we said above about valid tests for randomness as simple sets of small probability. Namely, we consider a test that consists of highly compressible strings, and note that this test is universal in some sense, i.e., it is as sensitive as any other test, up to an $O(1)$-constant.

Technically speaking, there is a result that relates complexity to the randomness deficiency in terms of tests. We state this result for a simple case (uniform distribution on the set of strings of given length; see the textbook [33, Sect. 14.1] for a more general statement). It uses the notion of *conditional complexity* $\mathrm{C}(x|u)$ of a string x given some u (an integer) defined as the minimal length of a program that produces x given u as an input.

Consider some integer function $d(x)$ defined on bit strings. Call it a *deficiency function* if it satisfies two requirements:

- $d(x)$ is *lower semicomputable*, i.e., $d(x)$ can be presented as a limit of a non-decreasing computable sequence of integers (uniformly in x), and
- for every k, the fraction of n-bit strings such that $d(x) > k$, is $O(2^{-k})$.

Such a deficiency function determines, for every n, a series of tests for uniformly distributed n-bit strings, where the kth test set for n-bit strings consists of strings of length n such that $d(x) > k$. In terms of the next section, $2^{d(x)}$ is a probability bounded test up to a constant factor. The second requirement guarantees that the test sets have small probability according to the uniform

[6] The short introduction can be found in the lecture notes [30] or in the introductory part of the textbook [33]. The algorithmic statistics, the part of algorithmic information theory that deals specifically with the statistical hypotheses and their testing, is discussed in two surveys [36,37].

distribution. The first one means, informally speaking, that the test is "semi-effective": if x has some peculiar property that makes it non-random, then we will ultimately discover this property and $d(x)$ will become large, but we never can be sure that x does *not* have properties that make it non-random, since $d(\cdot)$ is not required to be computable.

Proposition 1. *Among the deficiency functions there is a maximal one up to $O(1)$, i.e., a deficiency function d such that for every other deficiency function d' we have $d(x) \geqslant d'(x) - c$ for some c and all x. This maximal function is equal to $n - C(x|n) + O(1)$ for n-bit strings x.*

This result shows that the difference between length and complexity is the "universal measure of non-randomness" that takes into account all regularities that make a string x non-random. It is easy to prove also that if a string x belongs to a simple small set, then its deficiency is large, thus confirming the informal idea that a small set exhibits non-randomness of its elements.

There are many results about randomness deficiencies in this sense, but we cannot go into the details here and return instead to some other topics that are important for practical randomness tests.

5 Families of Tests and Continuous Tests

The law of large numbers says that for independent Bernoulli trials, e.g., for fair coin tossing, the number of successful trials is with high probability close to its expectation, i.e., to $n/2$ for n coin tossings. Therefore, large deviation is a rare event and can be used as a randomness test: as the deviation threshold increases, the probability of the event "the deviation exceeds this threshold" decreases, usually rather fast.

Instead of fixing some significance level and the corresponding threshold one could consider a family of tests: for every significance level ε we consider a set T_ε of measure at most ε. It consists of the outcomes where deviation exceeds some threshold that depends on ε. As ε decreases, the threshold increases, and the set T_ε and its measure decrease.

Such a family of tests can be combined into one non-negative function $t(x)$ defined on the set of possible outcomes, if we agree that T_ε is the set of outcomes where $t(x) \geqslant 1/\varepsilon$. Here we use $c = 1/\varepsilon$ as the threshold instead of ε to simplify the comparison with expectation-bounded tests discussed below. In this language the bound for the probability of T_ε can be reformulated as

$$\Pr[t(x) \geqslant c] \leqslant 1/c \quad \text{for every } c > 0 \qquad (*)$$

Informally speaking, $t(x)$ measures the "rarity", or "randomness deficiency" of an outcome x: the greater $t(x)$ is, the less plausible is x as an outcome of a random experiment.

Functions t that satisfy the condition $(*)$ are called *probability-bounded randomness tests* [3]. Sometimes it is convenient to use the logarithmic scale and

replace t by $\log t$. Then the condition $(*)$ should be replaced by the inequality $\Pr[t(x) \geqslant d] \leqslant 2^{-d}$. Note that a similar condition was used for deficiency functions in Sect. 4.

The condition $(*)$ is a consequence of a stronger condition $\int t(x)\, dP(x) \leqslant 1$ where P is the probability distribution on the space of outcomes. In other terms, this stronger requirement means that the expected value of t (over the distribution P) is at most 1, and the condition $(*)$ is its consequence, guaranteed by Markov's inequality. The functions t that satisfy this stronger condition are called *expectation-bounded randomness tests* [3].

In fact these two notions of test are rather close to each other: if t is a probability-bounded test, then $t/\log^2 t$ is an expectation-bounded test up to $O(1)$-factor. Moreover, the following general result is true:

Proposition 2. *For every monotone continuous function* $u\colon [1, +\infty] \to [0, \infty]$ *such that* $\int_1^\infty u(z)/z^2\, dz \leqslant 1$ *and for every probability-bounded test* $t(\cdot)$ *the composition* $u(t(\cdot))$ *is an expectation-bounded test.*

The statement above [7] is obtained by applying Proposition 2 to $u(z) = z/\log^2 z$.

6 Where Do We Get Randomness Tests?

As we have mentioned, different classical results of probability theory can be used as randomness tests. Take for example the law of large numbers. It says that some event, namely, a large deviation from the expected value, has small probability. This event can be considered as a test set. Mathematical statistics provides a whole bunch of tests of this type for different distributions, including χ^2-test, Kolmogorov–Smirnov test, and others.

Another source of statistical tests, though less used in practice, is provided by the probabilistic existence proofs. Sometimes we can prove that there exists an object with a given combinatorial property, say, a graph with good expansion properties, and the proof goes as follows. We consider a probabilistic process that constructs a random object. In our example this object is a graph. We prove that with high probability this random object satisfies the combinatorial property. Now we use the bit source that we want to test as a source of random bits for the algorithm. If we find out that the object constructed by the algorithm does *not* have the combinatorial property in question, we conclude that a rare event happened, and our source of random bits failed the test.

One should also mention tests inspired by algorithmic information theory (see, e.g., a 1992 paper by Maurer [20]) Each file compressor (like `zip`, `bzip`, etc.) can be considered as a random test. Assume that we have a sequence of bits, considered as a file, i.e., a sequence of bytes. If this file can be compressed by n bytes for some non-negligible n, say, by a dozen of bytes, then this bit sequence fails the test and can be considered as non-random one. Indeed, the probability of this event is at most 256^{-n}, up to a factor close to 1. The Bonferroni correction here says that we should multiply this probability by the number of popular

compressors, but even if we assume that there are thousands of them in use, it usually still keeps the probability astronomically small.

There is also a general way to construct probability-bounded tests. It is called "p-values", and the two previous examples of randomness tests can be considered as its special cases. Consider an arbitrary real-valued[7] function D defined on the space of outcomes. The value $D(x)$ is treated as some kind of "deviation" from what we expect, so we use the letter D. Then consider the function

$$p_D(x) = \Pr[\{y\colon D(y) \geqslant D(x)\}].$$

(defined on the same set of outcomes). In other words, for every threshold d we consider the set

$$T_d = \{y\colon D(y) \geqslant d\}$$

of all outcomes where the deviation is at least d, and measure the probability of this event, thus "recalibrating" the deviation function. In this language, $p_D(x)$ is the probability of the event $T_{D(x)}$, the chance to have in a random experiment of the given type the same deviation as it happened now, or a larger one.

Proposition 3.

(a) *For every $c \geqslant 0$, the probability of the event $p_D(x) \leqslant c$ is at most c.*

(b) *If each value of function D has probability at most ε, then the probability of the event $p_D(x) \leqslant c$ is between $c - \varepsilon$ and c.*

Proof. The probability $p_d = \Pr[T_d]$ decreases (more precisely, does not increase) as d increases. The function $d \mapsto p_d$ is left-continuous since the inequalities $D \geqslant d'$ for all $d' < d$ imply $D \geqslant d$. However, it may not be right-continuous, and a similar argument shows that the gap between the value of p_d and the right limit $\lim_{d' \to d+0} p_{d'}$ is the probability of the event $\{x\colon D(x) = d\}$. Now we add to this picture some threshold c, see Fig. 2. It may happen (case 1) that c is among the values p_d. This case is shown on the left and right pictures (Fig. 2). On the right picture the function p_d is constant on an interval where there are no values

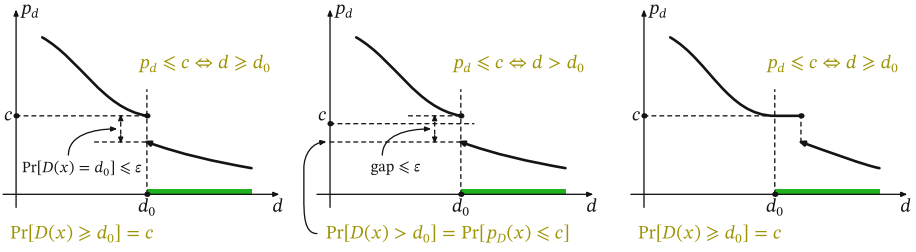


Fig. 2. Proof of Proposition 3

[7] This trick in a more general situation where the values of D are elements of some linearly ordered set, is considered in a paper by Gurevich and Vovk [10].

of D or these values have probability 0. Case 2: the threshold c may fall in a gap between some value of p_d and the right limit in the same point (denoted by d_0), as shown in the middle picture. The size of the gap is the probability of the event $D(x) = d_0$ and is at most ε according to our assumption. For the left and right pictures the inequality $p_D(x) \leqslant c$ means that $D(x) \geqslant d_0$, and the probability of the event $p_D(x) \leqslant c$ is exactly $p_{d_0} = c$, so both statements (a) and (b) are true. In this case the value of ε does not matter. For the middle picture $p_D(x) \leqslant c$ when $d > d_0$, and the probability of this event is not the value of p_d when $d = d_0$ but the right limit of p_d as $d \to d_0 + 0$. Still the difference does not exceed ε according to the assumption in (b), and again both statements are true.

Remark 1. This proof is given for the general case (X may be infinite); in the finite case the function p_d has only finitely many values, and the graph is a finite family of horizonal lines.

Remark 2. Proposition 3 obviously implies that the function $1/p_D(x)$ is a probability-bounded test. This observation allows us to construct many probability-bounded tests, starting from almost any random variable D. For example, we get a test from a probabilistic existence proof if we let D be the function that appears in the combinatorial statement, e.g., the second eigenvalue for the probabilistic proof that expander graphs exists. The only caveat is that we need to compute the function $d \mapsto p_d$, and this is usually not so easy. This function is often replaced by some its approximation, and this may lead to problems; see the discussion below.

Remark 3. If we apply this procedure to a function D that is already a probability-bounded test, then by definition we get some new test $t = 1/p_D$ such that $t(x) \geqslant D(x)$ for all x. In general, the function t could exceed D if the inequality in the condition $(*)$, Sect. 5, is strict.

7 Secondary Tests

There is an important type of randomness tests that can be called "secondary tests". Tests of this type appeared already in the classical book of Knuth [14, Sect. 3.3.1, B] and are extensively used in practical test suites [17]. Recall Proposition 3 and assume for a while that every individual value of D has very small probability, so we may assume that there are (almost) no gaps in the graph of p_d. Then Proposition 3 says that the random variable p_D is uniformly distributed on $[0, 1]$. Repeat the test N times, using fresh random bits (from the generator we are testing) for each repetition. Assuming that the null hypothesis is true, we get N independent reals that are uniformly distributed in $[0, 1]$. Then we may apply any test for the independent uniformly distributed variables, e.g., the Kolmogorov–Smirnov test for this distribution.

This procedude converts any p-value test for a random bits generator that has negligible probabilities of individual values into a "secondary test" that could be much more sensitive. Knuth describes a similar trick, but he does not use

the recalibration using p-values and applies the Kolmogorov–Smirnov (KS) test directly to the values of D and the distribution that should appear if the null hypothesis is true:[8]

> ... We should observe that the KS test may be used in conjunction with the χ^2 test... Suppose we have made, say, 10 independent χ^2 tests of different parts of a random sequence, so that values $V_1, V_2, \ldots, V_{10}$ have been obtained. It is not a good policy simply to count how many of the V's are suspiciously large or small. This procedure will work in extreme cases, and very large or very small values may mean that the sequence has too much local nonrandomness; but a better general method would be to plot the empirical distribution of these 10 values and to compare it to the correct distribution... This would give a clearer picture of the results of the χ^2 tests, and in fact the statistics K_{10}^+ and K_{10}^- [from KS test] could be determined as an indication of the success or failure... [Speaking about an example discussed earlier:] Notice that *all 20 observations in Fig. 4 (c)* [a figure from Knuth's book that we do not reproduce] *fall between the 5 and 95% levels*, so we would not have regarded *any* of them as suspicious, individually; yet collectively the empirical distribution shows that these observations are not at all right [14, Sect. 3.3.1, p. 50–51].

We return to the use of secondary tests in practical test suites in the next section.

8 Testing (Pseudo)randomness in Practice

There are several suits of randomness tests that are often used. The early history of randomness tests (as well as pseudorandom number generators) is described by Knuth [14, Sect. 3.3]. He starts with χ^2 and Kolmogorov–Smirnov tests, explains secondary testing (see the quote in the previous section) and also describes several *ad hoc* tests.

8.1 Diehard

Later George Marsaglia developed a `diehard` series of tests that were included (as C and Fortran sources) in a CD that he prepared [17]. That CD also included a collection of files with "random" bits, constructed by combining the output of hardware random bits generators with some deterministic pseudorandom sequences, see below Sect. 10. The description of the tests could be found in Marsaglia's papers [16,19]; see also the file `tests.txt` in the source code of the tests [17].

However, there are some problems with these tests. They heavily use the secondary test approach but not always in a correct way. First, one of the tests computes p-values for data that are not independent, as the following description, copied verbatim from the source code, shows:

[8] This is an equivalent approach since KS-test gives the same result after any monotone recalibration of the empirical values and theoretical distribution.

```
            This is the BIRTHDAY SPACINGS TEST
Choose m birthdays in a year of n days.  List the spacings
between the birthdays.  If j is the number of values that
occur more than once in that list, then j is asymptotically
Poisson distributed with mean m^3/(4n).  Experience shows n
must be quite large, say n>=2^18, for comparing the results
to the Poisson distribution with that mean.  This test uses
n=2^24 and m=2^9,  so that the underlying distribution for j
is taken to be Poisson with lambda=2^27/(2^26)=2.  A sample
of 500 j's is taken, and a chi-square goodness of fit test
provides a p value.  The first test uses bits 1-24 (counting
from the left) from integers in the specified file.
      Then the file is closed and reopened. Next, bits 2-25 are
used to provide birthdays, then 3-26 and so on to bits 9-32.
Each set of bits provides a p-value, and the nine p-values
provide a sample for a KSTEST.
```

As we see from this description, the different p-values use overlapping bits (2–25, 3–26, etc.) of the same numbers. There is no reason to expect that they are independent, contrary to the requirements of the Kolmogorov–Smirnov test. This description also exhibits another problem that appears in many tests from `diehard` suite. We use some asymptotic approximation, in this case the Poisson distribution, instead of the true distribution, ignoring the approximation error for which we have no upper bounds. Moreover, even if the error can be upper-bounded for the primary test, this upper bound does not translate easily into a bound for an error in the secondary test where we use the approximate distribution for recalibrating the deviations. Sometimes even the parameters of approximate distribution are only guessed. For example, in the description of one of the tests (named OQSO) Marsaglia writes about the distribution: "The mean is based on theory; sigma comes from extensive simulation". For the other one (called "parking lot test") even the mean is based on simulation: "Simulation shows that k should average 3523 with sigma 21.9 and is very close to normally distributed. Thus $(k - 3523)/21.9$ should be a standard normal variable, which, converted to a uniform variable, provides input to a KSTEST based on a sample of 10". Here KSTEST is the Kolmogorov–Smirnov test for uniform distribution. The arising problem is described by Marsaglia as follows:

> NOTE: Most of the tests in DIEHARD return a p-value, which should be uniform on $[0, 1)$ if the input file contains truly independent random bits. Those p-values are obtained by $p = F(X)$, where F is the assumed distribution of the sample random variable X—often normal. But that assumed F is just an asymptotic approximation, for which the fit will be worst in the tails. Thus you should not be surprised with occasional p-values near 0 or 1, such as .0012 or .9983. When a bit stream really FAILS BIG, you will get p's of 0 or 1 to six or more places. By all means, do not, as a Statistician might, think that a $p < .025$ or $p > .975$ means that

> the RNG has "failed the test at the .05 level". Such p's happen among the hundreds that DIEHARD produces, even with good RNG's. So keep in mind that "p happens".

This note combines two warnings. One is quite general and is related to the question of many tests applied to one sequence, see the discussion of the Bonferroni correction above, Sect. 2. The other one that should be separated from the first (but is not) is that `diehard` tests are not really tests in statistical sense, since they use the approximate distribution for recalibration and therefore small p-values could appear more often than they should.

Some other tests (not included in `diehard`) were later suggested by Marsaglia and Tsang [18].

8.2 Dieharder

A decade later Robert Brown [6] produced an extended version of the `diehard` test suite, called `dieharder`. The code was rewritten and published under GNU public license, integrated with GNU statistical library and parametrized, so now one can vary the sample size and the number of p-values easily. New tests were added and other improvements made. The resulting package is supported by mainstream Linux distributions. The package involves an extensive documentation. In particular, the `man` page says:

> A failure of the distribution of p-values at any level of aggregation signals trouble. ⟨...⟩ The question is, trouble with what? Random number tests are themselves complex computational objects, and there is a probability that their code is incorrectly framed or that roundoff or other numerical—not methodical—errors are contributing to a distortion of the distribution of some of the p-values obtained.

In this quote two problems are noted: the coding errors in the tests, and the problems related to the mathematical flaws in the approximate data used to construct the tests. The suggested solution for both problems is the same: testing these tests on "reference" random number generators. The `man` page says:

> There are a number of generators that we have theoretical reasons to expect to be extraordinarily good and to lack correlations out to some known underlying dimensionality, and that also test out extremely well quite consistently. By using several such generators and not just one, one can hope that those generators have (at the very least) different correlations and should not all uniformly fail a test in the same way and with the same number of p-values. When all of these generators consistently fail a test at a given level, I tend to suspect that the problem is in the test code, not the generators, although it is very difficult to be certain...
> Tests (such as the diehard `operm5` and `sums` test) that consistently fail at these high resolutions are flagged as being "suspect" ⟨...⟩ and they are strongly deprecated! Their results should not be used to test random

number generators pending agreement in the statistics and random number community that those tests are in fact valid and correct so that observed failures can indeed safely be attributed to a failure of the intended null hypothesis.

Unfortunately, `dieharder-3.31.1`, the last version available as of September 2019, also has some problems. One of them, affecting almost all tests, is the incorrect code that computes the Kolmogorov – Smirnov statistic. This code produces incorrect values, sometimes even impossibly small values, and in this case the computation of p-value gives 1. Indeed, in this case with probability 1 the deviation will be bigger than this impossibly small value. This is (correctly) interpreted as the test failure. Fortunately, it seems that for larger sample sizes the error in the statistics computation becomes less important.

8.3 NIST Test Suite

In 2000 the National Institute of Standards published a description of a test suite for randomness, including the source code. Now there exists an updated version [21].[9]

The description starts with some general words about randomness: "For example, a physical source such as electronic noise may contain a superposition of regular structures, such as waves or other periodic phenomena, which may appear to be random, yet are determined to be non-random using statistical tests" (p. 1–2). Then the authors speak about two types of possible errors: Type I (rejecting a good generator) and Type II (accepting a bad one) and about probabilities of these errors. However, their comments are misleading. Authors explain that a Type I error probability is a probability for a random sequence to get into the rejection set under a null hypothesis H_0—and this is correct. But then they say something confusing about the Type II errors: "Type II error probability is $\langle\ldots\rangle$ $P(\text{accept}H_0|H_0\text{is false})$" [21, p. 1–4]. While H_0 is a statistical hypothesis (model), namely, the assumption that the bits are independent and uniformly distributed, the words "H_0 is false" do not define any distribution, so one cannot speak about this conditional probability. The authors acknowledge this by saying "The probability of a Type II error is denoted as β. $\langle\ldots\rangle$ Unlike α [the probability of a Type I error], β is not a fixed value. $\langle\ldots\rangle$ The calculation of Type II error β is more difficult than the calculation of α because of the many possible types of non-randomness"—but still one could conclude from what the authors say that Type II error probability is well defined and is some number, though difficult to compute. This is a gross misunderstanding.

Then the authors explain the meaning of p-values, but again their explanations sound confusing, to say the least: "If a *P-value* for a test is determined to be equal to 1, then the sequence appears to have perfect randomness" (page

[9] The original version contained 16 randomness tests. In 2004 some errors in two tests were pointed out [12]. Correcting these errors, the revised version (2010) deleted one of the tests (the Lempel – Ziv test) and corrected the other one (the Fourier spectral test).

1–4). In reality the value 1 is not much better than the value 0, since the correctly computed p-values have uniform distribution. Even more strange is the following remark: "For a *P-value* ≥ 0.001, a sequence would be considered to be random with a confidence of 99.9%. For a *P-value* < 0.001, a sequence would be considered to be non-random with a confidence of 99.9% [21, p. 1–4, line 6 from below]. The second part could be interpreted in a reasonable way, though one should be cautious here, especially in the case of many tests. But the first part is completely misleading. Of course, one test that did not fail convincingly does not mean that the sequence is random with high confidence!

General remarks about tests constitute Part I of [21]. Parts II and III consist of the description and commentary for 15 tests. Some are similar to the tests in `diehard` while some other are different. The final Part IV, "Testing strategy and the Result Interpretation", recommends two ways to analyze the results of the tests. When several runs of a test produce a sequence of p-values, two forms of analysis of this sequence are recommended: "Proportion of Sequences Passing a Test" (4.2.1) and "Uniform Distribution of *P-values*" (4.2.2). Both are some variants of secondary tests: assuming that the distribution of p-values is uniform in $[0, 1]$, authors recommend to look at the proportion of values exceeding some threshold and to compare it with the Bernoulli distribution (4.2.1), or divide the interval $[0, 1]$ into some number of bins and apply χ^2-test (4.2.2). This approach replaces the Kolmogorov–Smirnov test used by Marsaglia in `diehard`, and in `dieharder`.

As for the case of `dieharder`, many tests from the NIST collection use approximations for computing p-values. The only warning about the consequences of this approach appears in the last section (p. 4-3):

> In practice, many reasons can be given to explain why a data set has failed a statistical test. The following is a list of possible explanations. The list was compiled based upon NIST statistical testing efforts.
> (a) An incorrectly programmed statistical test. ⟨...⟩
> (b) An underdeveloped (immature) statistical test.
> There are occasions when either probability or complexity theory isn't sufficiently developed or understood to facilitate a rigorous analysis of a statistical test. Over time, statistical tests are revamped in light of new results. Since many statistical tests are based upon asymptotic approximations, careful work needs to be done to determine how good an approximation is.
> (c) An improper implementation of a random number generator. ⟨...⟩
> (d) Improperly written codes to harness test input data. ⟨...⟩
> (e) Poor mathematical routines for computing *P-values*. ⟨...⟩
> (f) Incorrect choices for input parameters.

It is hardly surprising that with such a relaxed approach to statistical testing ("over time, statistical tests are revamped") the authors have included two bad tests in the original version of the document, see the paper [12] where the errors are noted. It is instructive to look at the errors in these tests. The first

error, in the Fourier spectral test, happened because the expectation and variance of the approximating normal distribution were computed incorrectly. The second is more interesting, since two different (and quite predictable) problems with the p-values approach appeared at the same time. First, the distribution of the test statistics based on the Lempel–Ziv compression algorithm was not computed exactly but was approximated using some presumably good pseudo-random number generator as reference. The experiments with other generators made in the paper [12] showed that this approximation is dubious. The other problem could be related to the non-negligible probability of individual values. As we have mentioned, in this case the distribution of the p-values differs from the uniform one. The results of numerical experiments described in the paper suggest that this could be the reason for the rejection of truly random sequences. In the current version of the NIST document[10] this test is excluded (it contains 15 tests instead of 16).

9 How to Make a Robust Test

There are different ways to deal with errors in statistical tests. Finding and correcting the coding errors is a general problem for all software, and tests are no exceptions here. One may argue that, since errors are anyway possible for many reasons (see the list above), we should not insist on the mathematical correctness of tests, just deleting the tests when they are discovered to be incorrect. Still the other approach is to do whatever we can to avoid errors that could be avoided. In this section we explain, following the technical report [32], how one could avoid problems related (a) to the approximation errors while computing p-values, and (b) to the non-negligible probabilities of individual outcomes. This will be done in two steps.

Step 1. Let us still assume that a reference generator that is truly random is available. But instead of using the reference generator to find the approximate distributions or to look for suspicious tests, as suggested by the authors of `dieharder` and NIST tests, we use it directly. Recall that the we used the Kolmogorov–Smirnov test to check that the distribution of p-values is consistent with the uniform distribution, and our problem was that due to approximation errors and non-zero probabilities of individual deviation values the distribution of p-values is not exactly uniform under the null hypothesis. However, there is a version of the Kolmogorov–Smirnov test that deals with *two* samples. Here the null hypothesis is that the two samples are formed by independent random variables with the same distribution, but nothing is assumed about this distribution.

[10] Unfortunately, the current version of the NIST report [21] was not checked carefully either. For example, the description of the serial test (Sect. 2.11.4, (5)) contains conflicting instructions for computing p-values: the example includes division by 2 that is missing in the general formula. The C code follows the example, not the general formula. Also the values of `igamc` function in this section are incorrect, while the correct values do appear few lines later, in Sect. 2.11.6.

Therefore, we may proceed as follows. The first sample of p-values is constructed using the random numbers generator that we test, as before. The second sample is constructed exactly in the same way but using the reference generator. Then we apply the Kolmogorov–Smirnov test for two samples. This procedure remains valid even if the formulas used to convert the deviations into p-values are only approximately true, or even completely wrong, since even completely wrong formulas would be the same for both generators, the one we test and the reference one. So we can omit the recalibration step completely and just consider the samples of deviations.

Remark 4. In fact, only the ordering is important, so the Kolmogorov–Smirnov test for two samples can be presented as follows. We have two arrays of reals, $x_1, \ldots, x_n$ (one sample) and $y_1, \ldots, y_m$ (another sample). Then we combine them into one array of length $n + m$ and sort this array, keeping track of the origin: elements that came from the first and second samples are marked by letters X and Y respectively. In this way we get a sequence of $n + m$ letters X and Y that contains n letters X and m letters Y, and consider some test statistic for the following null hypothesis: *all* $\binom{n+m}{m}$ *sequences of this type are equiprobable.* The Kolmogorov–Smirnov test uses some specific statistic, namely, the maximal difference between the frequencies of X's and Y's in all prefixes, but the same approach can be used with any other test for this distribution.

Remark 5. In this way we get a test that does not depend on the approximations to the distributions that we do not know how to compute. However, there is some price for it. Since now the reference generator is an additional source of random variations, we need more samples to get the same sensitivity of the test. This increase, however, is rather modest.

Step 2. We constructed a randomness test that does not rely on unproven assumptions about distributions that we cannot compute exactly. However, it uses a reference generator that is assumed to be truly random, and this is crucial. Obviously, if we use a faulty reference generator to test a truly random one, the test will fail (the procedure is symmetric, so we get exactly the same result when testing a faulty random generator against a truly random reference). In some sense, we constructed only a "randomized test of randomness". This is unsatisfactory, but can be easily avoided using the following trick.

Let us consider n deviation values $d_1, \ldots, d_n$ obtained by using bits from the generator we are testing. Then construct the other sample, $d'_1, \ldots, d'_m$, but this time let us use not the reference generator but the bitwise `xor` of the bits from the reference generator and fresh bits from the generator we are testing. Then we apply the Kolmogorov–Smirnov test to these two samples. If

1. the generator that we are testing is truly random, and
2. the reference generator and the test generator are independent,

then the probability to fail the test is guaranteed to be small due to Kolmogorov–Smirnov's result.

Remark 6. Of course, if the reference generator is not independent with the one that we want to test, the correctness claim is no more true. For example, if during the second part the reference generator produces the same bits as the generator we are testing, the `xor` bits will be all zeros. However, the independence condition looks much easier to achieve. For example, the reference bits could be produced in different place, or in advance, or can even be an output of a fixed deterministic pseudorandom generator.

Remark 7. If the reference generator produces truly random bits that are independent from the output of the generator we are testing, then `xor`-bits are also truly random. So our new test is as sensitive as the previous one (the comparison with the reference generator) if the reference generator is truly random, but the correctness of the new test does not depend on the assumption of true randomness for the reference generator.

Remark 8. There is one small problem that we have not mentioned yet: while sorting the array of deviations, we may have ties. If some value from the first sample coincides with some value from the second sample, then the letter (X or Y in our notation, see above) is not well defined. However, we can break the ties randomly, using the bits of the generator we are testing. In this way we may assume that these bits are truly random when bounding the probability of Type I error.

Remark 9. The same idea can be used even for "informal" tests. For example, imagine that we construct some image based on the bits we test, and then people look at this image and decide whether it looks similar to the pictures of the same type that use reference generator or there are some visible differences[11]. Recalling the interactive non-isomorphism proof and using the same trick as before, we can make a robust test. Take $2n$ disjoint bit blocks from the generator under testing. Use n of them to create images, and do the same for other n blocks but use the bitwise `xor` of these blocks and n blocks of the same size from some other origin. As a source of these auxiliary blocks one may use a reference generator, or the binary representation of π, or any other source. The only requirement is that the auxiliary blocks should be fixed before sampling our generator. If a human expert (or a machine-learning algorithm), looking at the resulting $2n$ images, can correctly classify them into two groups according to their origin, then the generator fails the test, and the probability of this for a true random bits generator is 2^{-2n}, up to a $\mathrm{poly}(n)$-factor.

One can also consider a more advanced version of this test. Several experts say how "random" the images are. Then the images are ordered according to their approval ratings and Kolmogorov–Smirnov test for two samples is used.

[11] This is not a purely theoretic possibility: the documentation for some hardware random bit generators contains pictures of this type.

10 Hardware Random Generators

Randomness is ubiquitous—from the coin tossing and cosmic rays to the thermal noise in audio and video recordings, Brownian motion, quantum measurements and radioactive decay. So one may think that constructing a good randomness generator is an easy task. However, if we require that the output distribution is guaranteed with high precision, the problem becomes much more difficult. Coins may be biased, the independence between the two consecutive coin tosses may be not absolute, the circuit with the noise is affected also by some undesirable signals that may not be random, etc. In addition, some technical errors could happen.

For an illustration one may look at the sequences of bits that are included in the CD prepared by Marsaglia [17]. Among them there are two bit sequences that he got from two devices he bought (one from Germany, one from Canada) and a third bit sequence produced (as Marsaglia says) by some hardware random generator in California. The names are `canada.bit`, `germany.bit` and `calif.bit`. The device makers claimed that the bits produced by their devices are perfectly random. However, applying `diehard` tests to these sequences, Marsaglia found that they are far from being random. In fact, looking at two of them (Canada and Germany), one could guess one of the reasons for their non-randomness [8]. Namely, splitting the bit sequences into bytes (integers in $0 \ldots 255$ range) and searching for the two-byte substring 10 10, we find that this substring does not appear at all (at least among the first 10^6 bytes I tested), while the expected number of occurrences is several dozens. To get an idea why this happens, one may also count the substrings of the form 13 x and find out that one of them appears much more often than the others: the substring 13 10 occurs more than four thousand times (instead of expected few dozens).

If the reader worked a lot with Unix and MSDOS computers in 1990s, she would immediately see a plausible explanation: the file was converted as a text file from Unix to MSDOS encoding. In Unix the lines of a text file were separated just by byte 10, while in MSDOS they were separated by 13 10. Converting 10 to 13 10, we make substrings 10 10 impossible and drastically increase the number of substrings 13 10.

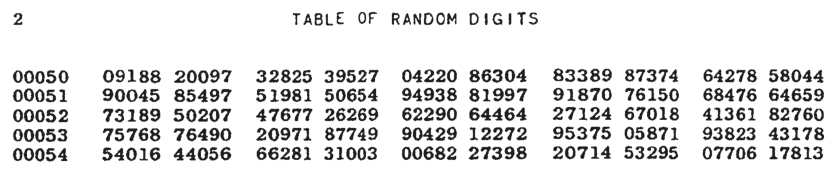
The third file `calif.bit` probably had some other history that did not involve Unix to MSDOS conversion, but still fails the tests for the other reasons.[12]

Marsaglia solved this problem by combining (`xor`ing) the output of the hardware random generators with pseudorandom sequences obtained by some deterministic generators [17, file `cdmake.ps`]

> The sixty 10-megabyte files of random numbers are produced by combining two or more of the most promising deterministic generators with sources of random noise from three physical devices (white noise), for those who feel that physical sources of randomness are better than deterministic sources.

[12] We also tested the corrected files, replacing groups 13 10 by 10 in `canada.bit` and `germany.bit`. They still fail many tests in the `dieharder` suite.

Some of the files have white noise combined with black noise, the latter from digital recordings of rap music. And a few of the files even had naked ladies thrown into the mix, from pixel files on the network. The last two, digitized music and pictures, are thrown in to illustrate the principle that a satisfactory stream of random bits remains so after combination [xor-ing] with the bits of any file.



2 TABLE OF RANDOM DIGITS

00050	09188	20097	32825	39527	04220	86304	83389	87374	64278	58044
00051	90045	85497	51981	50654	94938	81997	91870	76150	68476	64659
00052	73189	50207	47677	26269	62290	64464	27124	67018	41361	82760
00053	75768	76490	20971	87749	90429	12272	95375	05871	93823	43178
00054	54016	44056	66281	31003	00682	27398	20714	53295	07706	17813

Fig. 3. A fragment of table of random digits from [27]

A similar combination of the hardware source of somehow random bits and post-processing was used for the table of random digits published in 1955 [27] (see Fig. 3 for a small fragment of it):

> The random digits in this book were produced by rerandomization of a basic table generated by an electronic roulette wheel. Briefly, a random frequency pulse source, providing an average about 100,000 pulses per second, was gated about once per second by a constant frequency pulse. Pulse standardization circuits passed the pulses through a 5-place binary counter. In principle the machine was a 32-place roulette wheel which made, on the average, about 3000 revolutions per trial and produced one number per second. A binary-to-decimal converter was used which converted 20 of the 32 numbers (the other twelve were discarded) and retained only the final digit of two-digit numbers; this final digit was fed into an IBM punch to produce finally a punched card table of random digits.
>
> Production from the original machine showed statistically significant biases, and the engineers had to make several modifications and refinements of the circuits before production of apparently satisfactory numbers was achieved. The basic table of a million digits was then produced during May and June of 1947. This table was subjected to fairly extensive tests and it was found that it still contained small but statistically significant biases. ⟨. . .⟩
>
> [Comparing the results of tests before and after one month of continuous operations:] Apparently the machine had been running down despite the fact that periodic electronic checks indicated that it had remained in good order.
>
> The table was regarded as reasonably satisfactory because the deviations from expectations in the various tests were all very small—the largest being less than 2%—and no further effort was made to generate better numbers

> with the machine. However, the table was transformed by adding pairs of digits modulo 10 in order to improve the distribution of the digits. There were 20,000 punched cards with 50 digits per card; each digit on a given card was added modulo 10 to the corresponding digit of the preceding card to yield a rerandomized digit. It is this transformed table which is published here ⟨...⟩
> These tables were reproduced by photo-offset of pages printed by the IBM model 856 Cardatype. Because of the very nature of the table, it did not seem necessary to proofread every page of the final manuscript to catch random errors of the Cardatype. All pages were scanned for systematic errors, every twentieth page was proofread ⟨...⟩

We see that the same scheme was used here. However, the post-processing algorithms used in both cases are far from perfect. The sum modulo 10 used by RAND is almost reversible (if we know the resulting table and the first card, then we can reconstruct all the cards), so it cannot significantly change the entropy or the Kolmogorov complexity of the data string. This entropy is probably insufficient if simple tests fail on the string. The same can be said about `xor`-ing with a (deterministic) pseudorandom sequence used by Marsaglia. In the latter case, the rap music and naked ladies could save the day, assuming that these strings have enough complexity (generating processes have enough entropy) and are independent from the data from the electronic devices. But obviously one would like to have less frivolous and more regular procedure.

11 Random Source and Post-processing

Noise sources are cheap and easy to find. A classical example is a Zener diode which costs few cents; the noise generated by it is strong enough to be captured by an inexpensive audio card that has microphone inputs, and one can then try to convert this noise to a high-quality random bits ("white noise") using some processing called "conditioning" or "whitening". Many commercial devices uses this scheme (usually with a higher frequency and a lower precision than used in typical audio cards); here is the description of one of devices of this type:

> The TrueRNG Hardware Random Number Generator uses the avalanche effect in a semiconductor junction to generate true random numbers. The avalanche effect has long been used for generation of random number/noise and is a time-tested and proven random noise source. The semiconductor junction is biased to 12 volts using a boost voltage regulator (since USB only supplies 5V), amplified, then digitized at high-speed. The digitized data is selected and whitened internal to the TrueRNG and sent over the USB port with more than 400 kilobits/second of throughput. ⟨...⟩
> The new entropy mixing algorithm takes in 20 bits of entropy and outputs 8 bit to ensure that maximum entropy is maintained. The algorithm uses multiplication in a Galois field similar to a cyclic redundancy check to mix the ADC inputs thoroughly while spreading the entropy evenly across all bits [35].

On the other hand, one should be careful here, since the properties of Zener diodes are not guaranteed[13], as a simple experiment shows (Fig. 4). It is quite possible that noise properties may change over time and depend on the environment (exact voltage and current, temperature etc.) The post-processing should somehow be robust enough to convert these varying types of noise into random bits with the same uniform distribution.

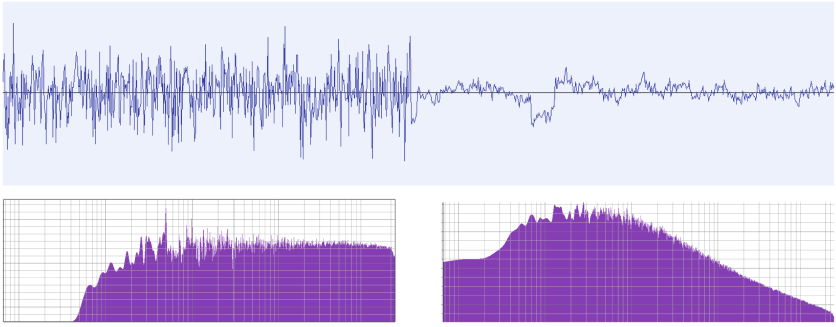


Fig. 4. The noise signal and its spectrum for two Zener diodes from the same roll, digitized by the same sound card (Behringer 1204usb) and analyzed by the same program (`audacity`).

The scheme of such a hardware random number generator is shown in a picture from NIST publication [23] (Fig. 5). In addition to the analog source and the conditioning block this scheme also provides a "health tests" block, with the obvious goal to raise an alarm when, for example, the analog noise source becomes broken for some reason, or drastically changed its parameters. The circuit is called an "entropy source", not a random bits generator, and the conditioning block is optional, since in [23] a more complicated scheme is considered: the output of this block is subjected to the next layer of conditioning before being sent to the customer. See below Sect. 13.

12 What We Would Like to Have

The ideal situation can be described as follows. There exist

- some mathematical property (E) of the output distribution of the (digital) noise source; its informal meaning is that "there is enough randomness in the output of the noise source";
- some hardware device for which the physicists guarantee (E) unless the device is visibly broken;

[13] The manufacturers of Zener diodes do not care much about the noise since the primary purpose of Zener diodes is different (and somehow opposite): to produce a stable voltage.

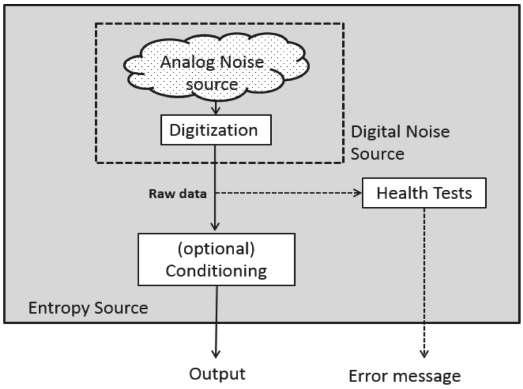


Fig. 5. A general scheme of a hardware entropy sources [23, p. 5]

– a deterministic transformation ("conditioning") and a mathematical theorem that guarantees that the output of this transformation is distributed almost uniformly if its input has the property (E).

Unfortunately, the current practice is rather far from this ideal. The NIST publication mentioned above suggests the property "min-entropy is large" as (E). This means that each individual outcome has small probability (by definition, the min-entropy of a distribution is at least k if every outcome has probability at most 2^{-k}). Here is what they say:

> The central mathematical concept underlying this Recommendation is entropy. Entropy is defined relative to one's knowledge of an experiment's output prior to observation, and reflects the uncertainty associated with predicting its value—the larger the amount of entropy, the greater the uncertainty in predicting the value of an observation. There are many possible measures for entropy; this Recommendation uses a very conservative measure known as min-entropy, which measures the effectiveness of the strategy of guessing the most likely output of the entropy source [23, p. 4].

However, min-entropy, being a very important notion, is still not enough to guarantee the good distribution after any (deterministic) conditioning transformation. Namely, for any transformation $T\colon \mathbb{B}^n \to \mathbb{B}$ that maps n-bit strings into bits, there is a random variable ξ with values in $\mathbb{B}^n$ that has min-entropy at least $n-1$ (almost maximal) such that $T(\xi)$ is a constant. Indeed, one of the preimages $T^{-1}(0)$ and $T^{-1}(1)$ has size at least 2^{n-1}, and we may let ξ be uniformly distributed in this preimage.

Moreover, even a stronger requirement than high min-entropy, introduced long ago by M. Santha and U. Vazirani [29], is not enough. This requirement, for a sequence of random Boolean variables $\xi_1, \xi_2, \ldots, \xi_n$, says that

$$\Pr[\xi_m = 1 \mid \xi_1 = x_1, \ldots, \xi_{m-1} = x_{m-1}] \in \left(\frac{1}{2} - \delta, \frac{1}{2} + \delta\right)$$

for every $m \leqslant n$ and for every $(m-1)$-bit string $x_1 \ldots x_{m-1}$. Here $\delta \in (0, 1/2)$ is some constant. Assume for example that $\delta = 1/6$; then the requirement says that whatever bits we have observed, the conditional probabilities to have 0 and 1 as the next bit differ at most by factor 2, being in the interval $(1/3, 2/3)$. This implies that min-entropy (and Shannon entropy) grows linearly with n, but is a much stronger condition, saying that in no circumstances we may predict the next bit reliably. Still, as proven in [29], this condition is not enough to extract even one "whitened" bit: there is no whitening algorithm that is better than the trivial one (taking the first bit). This claim can be slightly generalized: no way to extract k bits is better than the trivial one (just taking the first k bits). Here is the exact statement that implies this result.

Proposition 4. *Let $A \subset \mathbb{B}^n$ be a subset that has uniform probability p and let $\delta \in (0, 1/2)$. Then there exists a sequence of n random Boolean variables $\xi_1, \ldots, \xi_n$ that satisfies the Santha–Vazirani condition for this δ, such that*

$$\Pr[\xi_1 \ldots \xi_n \in A] \geqslant p^\alpha,$$

where α is a number such that $(1/2)^\alpha = \left(\frac{1}{2} + \delta\right)$.

Therefore, if the uniform probability of A is $(1/2)^k$ for some k, then the probability of the event $\xi_1 \ldots \xi_n \in A$ guaranteed by Proposition 4 (for some Santha–Vazirani source) is at least $(1/2 + \delta)^k$. This means that no transformation $T\colon \mathbb{B}^n \to \mathbb{B}^k$ can be better (in terms of extracting min-entropy from Santha–Vazirani source) than taking the first k bits. Indeed, one of the points in $\mathbb{B}^k$ has T-preimage in $\mathbb{B}^n$ of uniform probability at least $(1/2)^k$, and applying Proposition 4 to this preimage we conclude that for some Santha–Vazirani source $\xi_1, \ldots, \xi_n$ the min-entropy of $T(\xi_1 \ldots \xi_n)$ is not better than just for $\xi_1 \ldots \xi_k$: probability of some point in the image distribution is at least $(1/2 + \delta)^k$.

Proof. We need to construct a distribution on $\mathbb{B}^n$ that satisfies Santha–Vazirani condition and assigns large probability to A. We do it inductively, following the suggestion by Ruslan Ishkuvatov. The case $n = 1$ is obvious. For $n > 1$, we split A into two parts $0A_0$ and $1A_1$ according to the first bit, where A_0 and A_1 are subsets of $\mathbb{B}^{n-1}$ that can be considered as two faces of the Boolean cube. Let p_0 and p_1 be the probabilities of A_0 and A_1 according to the uniform distribution in $\mathbb{B}^{n-1}$, so their average is p. Assume that $p_0 \leqslant p_1$, so $p_0 = p - x$ and $p_1 = p + x$ for some $x \in [0, p]$. The induction assumption gives two Santha–Vazirani distributions on $\mathbb{B}^{n-1}$ that give large probabilities to A_0 and A_1, namely, at least p_0^α and p_1^α. They can be combined into one distribution on $\mathbb{B}^n$, we only need to chose the probability (between $1/2 - \delta$ and $1/2 + \delta$) for the first bit, and then use the two Santha–Vazirani distributions provided by the induction assumption as conditional distributions. To maximize the resulting probability, we should put maximal allowed weight on the face where the probability is greater. It remains to prove then that

$$\left(\frac{1}{2} - \delta\right) p_0^\alpha + \left(\frac{1}{2} + \delta\right) p_1^\alpha = \left(\frac{1}{2} - \delta\right) (p - x)^\alpha + \left(\frac{1}{2} + \delta\right) (p + x)^\alpha \geqslant p^\alpha.$$

For $\alpha \leqslant 1$, the function $t \mapsto t^\alpha$ is concave, therefore the left hand side is a concave function of x, and it is enough to check this inequality for endpoints $x = 0$ (where it is obvious), and $x = p$. In the latter case we need to prove that $(1/2 + \delta)(2p)^\alpha \geqslant p^\alpha$, and this follows directly from the definition of α.

Remark 10. A very simple proof for the case of 1-bit output [28] goes as follows. If $p \geqslant 1/2$, then for some Santha–Vazirani distribution with parameter δ one can achieve probability at least $1/2 + \delta$. Why? It is enough to spread the probability $1/2 + \delta$ uniformly on a subset $A' \subset A$ with uniform probability $1/2$, and spread the remaining probability $1/2 - \delta$ uniformly on the complement of A'.

So the situation is far from ideal: large min-entropy and even stronger Santha–Vazirani condition are not enough to guarantee the correct distribution after whitening, for any fixed whitening function. Still some practical solutions, i.e., some common sense recommendations that help us to avoid obviously faulty generators, are needed, even if no theoretical guarantees are provided. In the next section we look at the NIST approach to this problem.

13 What We Have

There are three documents produced by NIST that cover different aspects of random bits generation. The first, SP 800-90A [22], deals with algorithmic pseudorandom bits (or numbers) generators, called there *deterministic random bits generators*[14]. Here the relevant mathematical theory is not the algorithmic information theory but the complexity theory where the notion of (cryptographically strong) pseudorandom number generator was introduced by Manuel Blum, Silvio Micali and Andrew Yao [4,38]. This theory goes far beyond the scope of our survey.

Roughly speaking, such a generator is a polynomial-time algorithm that maps a truly random seed into a (much longer) sequence of bits that is "indistinguishable" from a random one by polynomial-size circuits. The indistinguishability means that no polynomial-size circuit can have significantly different probabilities of (a) accepting the output of the generator for a truly random seed, and (b) accepting a sequence of truly random bits. An equivalent definition says that there is no way to predict by a polynomial-size circuit the next bit of the output sequence (for a random seed) significantly better than by guessing.

The existence of generators with these properties is equivalent to the existence of one-way functions [11], and this existence is an unproven assumption. This assumption implies P $\neq$ NP, while the reverse implication is not known. For this reason we do not know any cryptographically strong pseudorandom number generator for which this property can be proven. Moreover, since the constructions from [11] are quite complicated, practical pseudorandom generators may use stronger assumptions, like hardness of factoring, or just have no theoretical justification at all. In fact, NIST [22] not only recommends but also insists

[14] Note an oxymoron.

on using "allowed" methods to generate random bits from the seed, and these methods are far from being justified mathematically, even in a very weak sense. For example, one of the methods uses hash values for consecutive bit strings (see Fig. 6). As explained in [22, page 37], "mechanisms specified in this Recommendation have been designed to use any **approved** hash function and may be used by consuming applications requiring various security strengths, providing that the appropriate hash function is used and sufficient entropy is obtained for the seed". On the next page a list of these "approved" hash functions is provided that includes SHA-1, SHA-224, SHA-512/224, SHA-256, SHA512/256, SHA-384, SHA-512. According to NIST [25]:

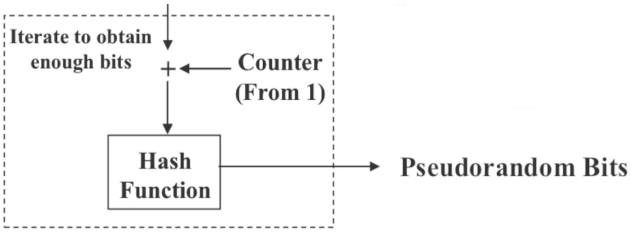


Fig. 6. Part of Fig. 8 on p. 39 in [22] related to the random bit generation using a hash function (the initialization part is omitted).

An approved hash function is expected to have the following three properties:

1. Collision resistance: It is computationally infeasible to find two different inputs to the hash function that have the same hash value. That is, if *hash* is a hash function, it is computationally infeasible to find two different inputs x and x' for which $hash(x) = hash(x')$. Collision resistance is measured by the amount of work that would be needed to find a collision for a hash function with high probability. If the amount of work is 2^N, then the collision resistance is N bits $\langle \ldots \rangle$
2. Preimage resistance $\langle \ldots \rangle$
3. Second preimage resistance $\langle \ldots \rangle$

Obviously, for a specific function like SHA-1 or any other mentioned in the list above, the collision resistance requirement makes no sense if understood literally: computation infeasibility means high complexity of some function, and here we have no function. If somebody comes with a collision pair x, x', the collision resistance in the naïve sense disappears starting from this moment, so it is not a mathematical property of a hash function (a mathematical property cannot suddenly become false), but some property of the current state of art (still measured in bits!). Moreover, the hash functions mentioned above are obtained by a complicated *ad hoc* construction and there are no reasons to believe that something similar to collision resistance can be proven. Finally, as it is mentioned [22, p. 89],

> Hash_DRBG's [the random generator based on hash functions] security depends on the underlying hash function's behavior when processing a series of sequential input blocks. If the hash function is replaced by a random oracle, Hash_DRBG is secure. It is difficult to relate the properties of the hash function required by Hash_DRBG with common properties, such as collision resistance, pre-image resistance, or pseudorandomness.

Indeed, it is impossible to relate the "required" properties with "common" properties, since a function with no known collisions and high preimage resistance still may have much more 1s than 0s in most of its outputs, or have the last bit always equal to 1, therefore being completely unsuitable for Hash_DRBG.[15] So the reference to the security properties of the allowed hash functions can only create a false feeling of security.

The second NIST publication, SP 800-90B [23], describes the allowed constructions of the "entropy source" (see Fig. 5 above) while the third one [24] "addresses the construction of RBGs from the mechanisms in SP 800-90A and the entropy sources in SP 800-90B" [23, p. 1]. The idea here is that the whitening (conditioning) process is splitted into two stages. The first stage, described in SP 800-90B, does only some "rough" conditioning and may not produce a distribution that is very close to the uniform one. We hope only that the entropy of its output is close to the output length or at least is a significant fraction of the length. Then the second stage that may involve deterministic random bits generators or not is used for "fine-tuning".[16]

But what is meant by "entropy" in this description? As we have said, the NIST recommendations claim to use min-entropy. Still there are some problems with this approach.

- There is no way to get a reliable lower bound for the min-entropy of a physical source. If there is some isolated value that appears with probability 2^{-k}, the min-entropy is at most k, but one needs to make $\Theta(2^k)$ trials to have a reasonable chance to see this value at least once.[17]

[15] And the claim about the random oracle model is obviously true and obviously irrelevant.

[16] The final stage may also use pseudorandom bit generators to provide additional "backup" layer if the physical source stops working. For example, one may follow Marsaglia and produce `xor` of the bit sequences from physical and deterministic sources. Note that this operation, hiding the problems with physical source, makes the testing of the output sequence almost useless; testing should be done before this last step.

[17] Things are much better for independent identically distributed (i.i.d.) variables [23, p. 11]; there are also some physical sources where i.i.d. assumptions are reasonable, and some tests that can detect some violations of i.i.d. property.

– Quite often the NIST recommendations treat the notion of entropy informally, as some mystical substance that can be present in a binary string (and not in a random variable) and even can be accumulated and/or condensed[18]. For example, it is written [23, p. 11] that "in all cases, the DRBG [deterministic random bits generator] mechanism expects that when entropy input is requested, the returned bitstring will contain at least the requested amount of entropy." The closest approximation to this interpretation is Kolmogorov complexity, but it is (a) non-computable and (b) defined up to a constant, and different reasonable optimal programming languages easily can give the values that differ by several thousands, so it does not make sense to ask whether the complexity exceeds (say) 512 or not.
– Moreover, in some cases even more enigmatic explanations are given: "For the purposes of this Recommendation, an n-bit string is said to have full entropy if the string is the result of an approved process whereby the entropy in the input to that process has at least $2n$ bits of entropy (see [ILL89] and Sect. 4.2)" [24, p. 11]. Here [ILL89] is the preliminary version of [11] and neither says anything about approved processes nor justifies the requirement about $2n$ bits of entropy.
– As mentioned above for a similar situation, the properties of the functions used for conditioning, in particular the standard requirements for the security of a hash function, do not guarantee, even informally, that the output distribution for the hash function applied to an input source of high min-entropy, is close to the uniform distribution. However, this is implicitly assumed in the recommendations when an approved process of obtaining a string of full entropy is described.
– The recommendations encourage the designer to use different combinations of "approved" constructions (see, e.g., Fig. 7); even if some good properties of one-stage construction are plausible, the claim that the composition of several stages will still have good properties, is much less founded.

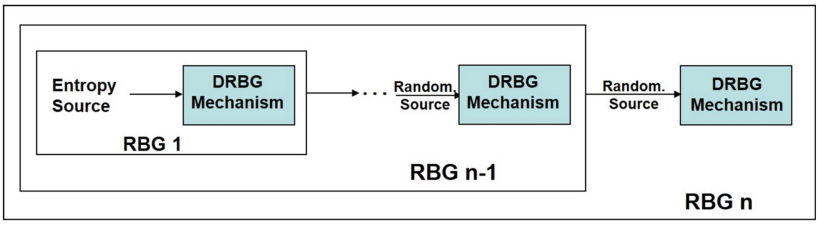


Fig. 7. The construction of a random bits generator with several layers [24, p. 19].

[18] "When the entropy [string?] produced by the entropy source(s) is very long (e.g., because the entropy rate of the entropy source(s) is very low), and the entropy bits may need to be condensed into a shorter bitstring, the **Get_Entropy** function in Sect. 10.3.1.1 or Sect. 10.3.1.2 shall be used to condense the entropy bits without losing the available entropy in the bit string." [23, Sect. 10.3.1, p. 44].

All these critical remarks do not mean that NIST recommendations are unnecessary: they reflect the current state of technology, the existing practice and prevent the appearance of completely bogus generators, therefore playing a very important role. However, one should keep in mind that they are not based on any "hard science"; they sometimes use mathematical notions and results but only as hints and sources of inspiration.

Could we have better recommendations? This is a difficult question. One can hope for the *security through obscurity*: if a long sequence of different mathematical operations is performed, this could make an attack much more difficult. However, the idea that *random actions give random results* does not look as a good plan for designing random bits generators. It could be that the careful choice of a noise source plus one-layer conditioning procedure that is based on something more suitable than just hash functions, would give a better result than a complicated multi-layer approach using cryptographic primitives.

14 Final Remarks

The space limitations do not allow us to discuss other interesting questions related to theory and practice of random bit generators.

On the theory side, there is a lot of knowledge about randomness extractors—from a complexity-theoretic viewpoint, they are much closely related to the practical task of conditioning raw randomness than hash functions. It is not completely clear to what extent we can achieve the goal: to have a clear and reasonable assumption about raw distribution that provably guarantees that the output distribution is close to the uniform one. Still we may hope that some constructions inspired by this theory could be practically useful. In particular, there are results about extractors with many sources that could be easier to use (literally or as a source of inspiration), since independence appears more often in the "real world" than uniform distributions.

From the viewpoint of physics there is a difference between a "random noise" that comes from statistical mechanics, say, the thermal noise in a resistor, or the Brownian motion, or some chaotic dynamical systems with external noise, and more "refined" randomness that comes out of quantum mechanical systems where the events related to individual microscopic objects can be observed, for example, experiments with individual photons. However, one can argue that

- there is no clear distinction between two categories: how do we classify Geiger counter events for a macroscopic piece of slightly radioactive material? how do we classify the noise in a PN junction (definitely related to some quantum effects but in multi-particle systems)?
- from the practical viewpoint, it is not clear if one can construct an experimental device that is "clean" enough to avoid the conditioning step. And if we use conditioning, do we really have an advantage using a delicate quantum-mechanical experiment instead of cheaper alternatives? One can argue that it is better to have "true randomness" instead of "mere chaos", or something like this. It definitely sounds good for philosophers or for a sales brochure, but are there more essential advantages?

Last but not least, the notions that appear in this discussion (randomness tests, individual random objects) can be studied from the viewpoint of algorithmic information theory. There are many interesting questions and results of this type [3,26] that are starting points for the "quantitative" theory of randomness. In this approach, roughly speaking, a result of the form "if α is algorithmically random, then β is algorithmically random" is made more precise by proving the upper bound for the randomness deficiency of β in terms of the random deficiency of α. Interesting questions also appear when we try to translate the results about some combinatorial constructions (say, randomness extractors or secret sharing) into the language of algorithmic information theory. But this is a topic for another long survey.

Acknowledgments. The author is grateful to all the people in the RaCAF project and the ESCAPE team (LIRMM, Montpellier), Kolmogorov Seminar (Moscow), Theoretical Computer Science Lab (Moscow, HSE), and all others from whom I learned about randomness including my (late) teacher Vladimir Uspensky, Leonid Levin, Alexander Zvonkin, Nikolay Vereshchagin, Vladimir Vovk, Vladimir Vyugin, Andrei Romashchenko, Bruno Durand, Gregory Lafitte, Laurent Bienvenu, Péter Gács, Wolfgang Merkle, Paul Vitányi, Daniil Musatov, Andrei Rumyantsev, Mikhail Andreev, Gleb Novikov, Bruno Bauwens, Konstantin Makarychev, Yury Makarychev, Ilya Razensteyn, Gleb Posobin, Alexey Vinogradov, Ruslan Ishkuvatov. The work was supported by ANR RaCAF grant ANR-15-CE40-0016-0.

References

1. Wasserstein, R.L., Lazar, N.A.: Editorial: the ASA's statement on p-values: context, process, and purpose. Am. Stat. **70**(2), 129–133 (2016). https://doi.org/10.1080/00031305.2016.1154108
2. Benjamin, D.J., et al.: Redefine statistical significance. Nat. Hum. Behav. **2**, 6–10 (2018)
3. Bienvenu, L., Gáacs, P., Hoyrup, M., Rojas, C., Shen, A.: Algorithmic tests and randomness with respect to a class of measures. Proc. Steklov Inst. Math. **274**, 34–89 (2011). http://arxiv.org/abs/1103.1529
4. Blum, M., Micali, S.: How to generate cryptographically strong sequences of random bits. SIAM J. Comput. **13**(4), 850–864 (1984). https://doi.org/10.1137/0213053. (preliminary version was presented at FOCS 1982 conference)
5. Borel, É.: Le Hasard. Librarire Félix Alcan (1920)
6. Brown, R.G.: Dieharder: a GNU public random generator, version 3.31.1. Technical report, Duke University Physics Department (2006–2018). http://www.phy.duke.edu/~rgb/General/dieharder.php
7. David, A.P., de Rooij, S., Shafer, G., Shen, A., Vereshchagin, N., Vovk, V.: Insuring against loss of evidence in game-theoretic probability. Stat. Probab. Lett. **81**, 157–162 (2011). https://doi.org/10.1016/j.spl.2010.10.013
8. Davies, R.: Hardware random number generators. Technical report, Statistics Research Associates Limited (2000). http://robertnz.net/hwrng.htm. Presented at 15th Australian Statistics Conference, July 2000, and 51st Conference of New Zealand Statistical Association, September 2000

9. Gurevich, Y., Passmore, G.O.: Impugning randomness, convincingly. Studia Logica **100**(1–2), 193–222 (2012). https://link.springer.com/article/10.1007/s11225-012-9375-1. See also https://arxiv.org/pdf/1601.00665.pdf, https://www.cl.cam.ac.uk/~gp351/Gurevich-Passmore-IRC.pdf
10. Gurevich, Y., Vovk, V.: Test statistics and p-values. Technical report, arXiv (2017). Working paper #16, On-line compression modelling project (new series). http://www.alrw.net/articles/16.pdf. See also https://arxiv.org/pdf/1702.02590.pdf
11. Håstad, J., Impagliazzo, R., Levin, L.A., Luby, M.: A pseudorandom generator from any one-way function. SIAM J. Comput. **28**(4), 1364–1396 (1999). https://doi.org/10.1137/S0097539793244708
12. Kim, S.Y., Umeno, K., Hasegava, A.: Corrections of the NIST statistical test suite for randomness. Technical report (2004). https://eprint.iacr.org/2004/018.pdf
13. Kireev, A.: On the falsified results of the "referendum" in Sevastopol (in Russian). Technical report, LiveJournal, November 2014. https://kireev.livejournal.com/1095568.html
14. Knuth, D.: The Art of Computer Programming. Seminumerical Algorithms, vol. 2, 2nd edn. Addison-Wesley, Boston (1981). ISBN 0-201-03822-6
15. Kupriyanov, A.: Gauss against Churov: preliminary conclusions. Technical report, Troitsky variant (Russian newspaper), May 2018. https://trv-science.ru/2018/05/08/gauss-protiv-churova-promezhutochnyj-itog
16. Marsaglia, G.: A current view of random number generators. In: Computer Science and Statistics, Sixteenth Symposium on the Interface, pp. 3–10. Elsevier, North-Holland (1985)
17. Marsaglia, G.: Random numbers CDROM including the Diehard battery of tests of randomness. Technical report, University of Florida (1995). http://stat.fsu.edu/pub/diehard/, was available at http://stat.fsu.edu/pub/diehard/; now (2019) still available as snapshots from https://web.archive.org. Contains the preprint version of [16, 19]
18. Marsaglia, G., Tsang, W.W.: Some difficult-to-pass tests of randomness. J. Stat. Softw. **7**(3) (2002). https://www.jstatsoft.org/article/view/v007i03
19. Marsaglia, G., Zaman, A.: Monkey tests for random number generators. Comput. Math. Appl. **26**(9), 1–10 (1993)
20. Maurer, U.M.: A universal statistical test for random bit generators. J. Cryptol. **5**(2), 89–105 (1992). https://link.springer.com/article/10.1007/BF00193563
21. Rukhin, A., et al.: A statistical test suite for random and pseudorandom number generators for cryptographic applications, revision 1 by Lawrence E. Bassham III. Special Publication 800-22-1a, National Institute of Standards and Technology, Technology Administration, U.S. Department of Commerce (NIST), April 2010. https://www.nist.gov/publications/statistical-test-suite-random-and-pseudorandom-number-generators-cryptographic. Previous version seems to be unavailable at this site, but the review of Elaine B. Barker, ITL Bulletin (December 2000, 3 pp.), is available at https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=151231. The Lempel–Ziv test, criticised in [12], was there (#10) according to the review; it is missing in the updated version
22. Barker, E., Kelsey, J.: Recommendation for random number generation using deterministic random bit generators. Special Publication 800-90A, National Institute of Standards and Technology, Technology Administration, U.S. Department of Commerce (NIST), June 2015. https://csrc.nist.gov/publications/detail/sp/800-90a/rev-1/final. Previous Version: January 2012

23. Turan, M.S., Barker, E., Kelsey, J., McKay, K., Baish, M., Boyle, M.: Recommendation for the entropy sources used for random bit generation. Special Publication 800-90B, National Institute of Standards and Technology, Technology Administration, U.S. Department of Commerce (NIST), January 2018. https://csrc.nist.gov/publications/detail/sp/800-90b/final
24. Barker, E., Kelsey, J.: Recommendation for random bit generator (RBG) constructions (second draft). Special Publication 800-90C, National Institute of Standards and Technology, Technology Administration, U.S. Department of Commerce (NIST), April 2016. https://csrc.nist.gov/CSRC/media/Publications/sp/800-90c/draft/documents/sp800_90c_second_draft.pdf
25. Dang, Q.: Recommendation for applications using approved hash algorithms, revision 1. Special Publication 800-107r1, National Institute of Standards and Technology, Technology Administration, U.S. Department of Commerce (NIST), August 2012. https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-107r1.pdf
26. Novikov, G.: Randomness deficiencies. In: Kari, J., Manea, F., Petre, I. (eds.) CiE 2017. LNCS, vol. 10307, pp. 338–350. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-58741-7_32
27. RAND Corporation: A Million Random Digits with 100,000 Normal Deviates. Free Press (1955). Reissued in 2001 as ISBN 0-8330-3047-7
28. Reingold, O., Vadhan, S., Wigderson, A.: A note on extracting randomness from Santha–Vazirani sources. Technical report, available from Reingold (2014). https://omereingold.files.wordpress.com/2014/10/svsources.pdf
29. Santha, M., Vazirani, U.V.: Generating quasi-random sequences from semi-random sources. J. Comput. Syst. Sci. **33**, 75–87 (1986). https://doi.org/10.1016/0022-0000(86)90044-9
30. Shen, A.: Around Kolmogorov complexity: basic notions and results. In: Vovk, V., Papadopoulos, H., Gammerman, A. (eds.) Measures of Complexity, pp. 75–115. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21852-6_7
31. Shen, A.: Election and statistics: the case of "United Russia", 2009–2018 (in Russian), preprint (2018). https://arxiv.org/abs/1204.0307
32. Shen, A.: Making randomness tests more robust. Technical report, HAL, February 2018. https://hal.archives-ouvertes.fr/hal-01707610
33. Shen, A., Uspensky, V.A., Vereshchagin, N.K.: Kolmogorov Complexity and Algorithmic Randomness. American Mathematical Society (2017). http://www.lirmm.fr/~ashen/kolmbook-eng-scan.pdf
34. Stoppard, T.: Rosencrantz and Guildenstern Are Dead, a Play (1966). Grove Press (1971). ISBN 978-0-8021-3275-8
35. TrueRNG: TrueRNG documentation. Technical report, Ubld.it (2019). http://ubld.it/truerng_v3
36. Vereshchagin, N., Shen, A.: Algorithmic statistics revisited. In: Vovk, V., Papadopoulos, H., Gammerman, A. (eds.) Measures of Complexity, pp. 235–252. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21852-6_17. https://arxiv.org/abs/1504.04950v2
37. Vereshchagin, N., Shen, A.: Algorithmic statistics: forty years later. In: Day, A., Fellows, M., Greenberg, N., Khoussainov, B., Melnikov, A., Rosamond, F. (eds.) Computability and Complexity. LNCS, vol. 10010, pp. 669–737. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-50062-1_41. https://arxiv.org/abs/1607.08077
38. Yao, A.C.: Theory and application of trapdoor functions. In: 23rd Annual Symposium on Foundations of Computer Science (FOCS), pp. 80–91 (1982). http://ieeexplore.ieee.org/document/4568378/