Quantum Random Number Generation with the Superconducting Quantum Computer IBM 20Q Tokyo

Kentaro Tamura[1] and Yutaka Shikano[2,3][0000−0003−2107−7536]

[1] Department of Applied Physics and Physico-Informatics, Keio University, 3-14-1 Hiyoshi, Kohoku, Yokohama, 223-8522 Japan
`cicero@keio.jp`

[2] Quantum Computing Center, Keio University, 3-14-1 Hiyoshi, Kohoku, Yokohama, 223-8522 Japan
`yutaka.shikano@keio.jp`

[3] Institute for Quantum Studies, Chapman University, 1 University Dr., Orange, CA 92866 USA

Abstract. Quantum random number generators (QRNGs) produce theoretically unpredictable random numbers. A typical QRNG is implemented in quantum optics [Herrero-Collantes, M., Garcia-Escartin, J. C.: Quantum Random Number Generators. Rev. Mod. Phys. **89**, 015004 (2017)]. Quantum computers become QRNGs when given certain programs. The simplest example of such a program applies the Hadamard gate on all qubits and performs measurement. As a result of repeatedly running this program on a 20-qubit superconducting quantum computer (IBM 20Q Tokyo), we obtained a sample with a length of 43,560. However, statistical analysis showed that this sample was biased and correlated. One of the post-processed samples passed statistical tests. To show the effectiveness of post-processing, a larger sample size is required. The present study of quantum random number generation and statistical testing may provide a potential candidate for benchmarking tests of actual quantum computing devices.

Keywords: true number generator· quantum random number generator· IBM Q· randomness extraction· min-entropy.

1 Introduction

Random numbers are a fundamental resource in various fields of science and industry [1]. This is for two reasons: Firstly, random numbers are used to simulate stochastic phenomena such as Brownian motion. Secondly, random numbers are a source of unpredictability. Examples of such applications are below.

Cryptography. Binary information can be encrypted with the use of random numbers. By XORing the binary representation of a message with a random number sequence of equal length, an information-theoretically secure encrypted message is obtained.

Simulation. Random numbers are used to simulate random events such as Brownian motion.

Gambling. Various situations in gambling require randomness. To simulate a fair shuffle of a deck of cards, for example, random numbers are essential.

Sampling. In situations where samples need to be extracted from a group, random numbers enable unbiased and uncorrelated extraction.

Given the wide variety of applications above, the generation of random numbers is still an ongoing field of research.

There are three main types of random number generators (RNGs): pseudorandom number generators (PRNGs), classical physical random number generators, and quantum physical random number generators (QRNGs). PRNGs adopt deterministic functions that produce seemingly random bits when given an initial value called a seed. There are several advantages to PRNGs. Firstly, they can be easily implemented on the classical computer. Secondly, one needs only to store the function and the seed in order to reproduce the output. Finally, the generation rate is generally faster than other types of random number generators. The problem with PRNGs is that the output can be predicted with knowledge of the seed and function. While PRNGs use deterministic functions to produce pseudorandom numbers, physical random number generators apply measurement to physical phenomena and convert the resulting values into bits. In the case of classical physical random number generators, the output can be predicted from the initial condition of the physical phenomena. QRNGs, on the other hand, produce theoretically unpredictable bits. The reason behind this is that under the axioms of quantum physics, the measurement outcome is probabilistic.

Quantum computers can be taken as QRNGs when assigned certain programs. In this study, we experimented with a program that applies the Hadamard gate and performs measurement on all qubits available. A sample sequence with a length of 43,560 was obtained by repeatedly running this program on IBM's superconducting computer (IBM 20Q Tokyo). It became clear through statistical tests that this sample was biased and correlated. In order to see whether the bias and correlation can be corrected, we applied several post-processors. Among the ones we applied, the combination of Samuelson's extractor and von Neumann's extractor produced a sample that passed the six statistical tests. As the combination of post-processing significantly reduced the sample size, the effectiveness is of the post-processor is debatable.

The existence of bias and correlation in the quantum random numbers generated by the quantum computer carries information about the device. Our next task is to determine the reason behind these flaws.

The rest of this paper is organized as follows. Section 2 briefly reviews randomness, random number generators, tests for randomness, and post-processing. We present the procedure for quantum random number generation on the IBM 20Q Tokyo and show the test results for the sample before and after post-processing in Sec. 3. An interpretation for the results are also discussed. Section 4 is devoted to the summary. In Sec. 5, we discuss some open questions on QRNGs.

2 Preparation

2.1 Randomness

The key to understanding randomness is to acknowledge the distinction between the following two types of randomness.

Product randomness. The apparent randomness that can be statistically identified by examining the output of a random process is called product randomness. Product randomness is practically important, as the length of a random number sequence is always finite. To check whether a coin toss yields a random number sequence, one can toss the coin a finite number of times and see if the heads and tails outcomes are unbiased and independent. This is the idea of product randomness.

Process randomness. The inherent randomness of the process of generation is called process randomness, which is an important factor of a true random number generator. In the example of a coin toss, process randomness corresponds to a symmetrically designed coin and a fair tossing process. By making sure that the process is random, one can expect the output to have product randomness as well.

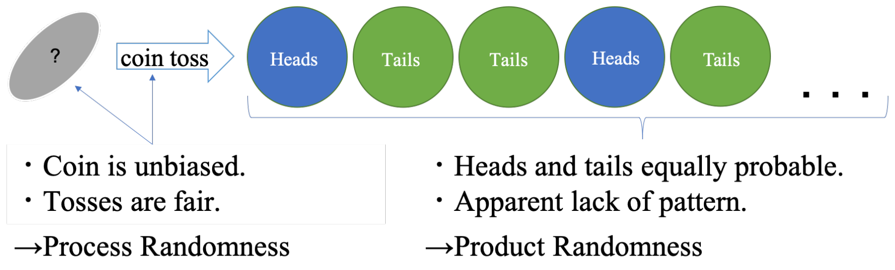

Fig. 1. Product randomness and process randomness of a coin toss.

While the goal of random number generation is to identify whether a generator is capable of producing random numbers, one can only observe finite samples of outputs. In practice, therefore, a generator is characterized from the samples obtained. For example, to check whether a coin toss is fair, one must toss the coin repeatedly and look for signs of bias and correlation.

2.2 Essential Characteristics of Random Number Generators

There are three main qualities that a random number generator must possess in order to qualify as legitimate.

Uniformity. A coin (random variable) with uniformity yields heads "1" and tails "0" with equal probability.
Independence. Independent coin tosses (trials) are such that any toss does not affect any of the other tosses. In other words, the tosses are not correlated.
Unpredictability. Unpredictability means that it is impossible to predict any of the future outcomes.

Ideally, every random number generator should possess all qualities shown above. However, only quantum random number generators possess all three qualities. Because one can only ever hope to examine a random number sequence of finite length, identifying the qualities above can only be done by applying randomness tests to the output.

2.3 Types of Random Number Generators

Random number generators can be classified into the following groups. Each group has its advantages and disadvantages, and the means of generation is chosen in light of what application the user has in mind. Only the final group can be considered true random number generators, which produce uniform, independent, and unpredictable bits.

Pseudorandom Number Generators The present computer (classical computer) is deterministic due to the deterministic principles of classical physics. As a result, classical computers are incapable of producing true random numbers. In order to overcome this problem, pseudorandom numbers have been used as a substitute for true random numbers.

The idea behind pseudorandom numbers is that as far as product randomness is concerned, a deterministic function that produces seemingly random outputs suffices for some applications. For instance, random numbers used in simulation need not be unpredictable.

While pseudorandom numbers are predictable and do not possess process randomness, advantages do exist: pseudorandom numbers are reproducible. This means that one needs only to remember the initial value and generation process to obtain the same sequence, which is not the case with true random numbers. In other words, pseudorandom numbers can be compressed. Under circumstances where the same random number sequence is required multiple times, pseudorandom numbers are convenient. Another advantage is that the generation of pseudorandom numbers is generally faster than physical random numbers [1]. This is convenient for simulation purposes where long sequences of random numbers are required.

Despite such advantages, there exist applications where pseudorandom numbers cannot be used as a substitute for true random numbers. These applications require unpredictability or fairness. In cryptography, for example, it is crucial that the future values are unpredictable. This cannot be achieved by pseudorandom numbers alone. Furthermore, pseudorandom numbers are not fair. This means that with knowledge of the initial value called a seed along with the function, one can easily manipulate the output. This is why pseudorandom numbers should not be used for choosing winning lottery tickets.

In areas where unpredictability or fairness is required, true random numbers cannot be replaced with pseudorandom numbers.

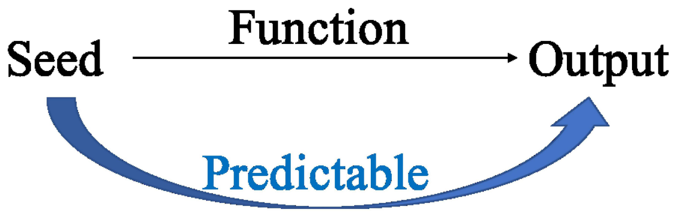

Fig. 2. Conceptual diagram of pseudorandom number generation.

As shown in Fig. 2, pseudorandom number generators are functions that when given an initial value called a seed, the rest of the output is automatically determined in the form of a recurrence formula. Random numbers produced in this manner have patterns (the generation formula itself) and periods. It is thus crucial that the user understands the limitations of pseudorandom number generators. For a better grasp of this concept, several examples of such functions or algorithms are presented in this section.

Middle-Square Method. The middle-square method is one of the first methods of pseudorandom number generation. It produces pseudorandom numbers based on the following algorithm. In this example, the seed is a decimal integer with 4 or more digits. The resulting sequence consists of 4-digit decimal integers.

1. Square the seed.
   e.g.) If seed = 12345, seed^2 = 152399025.
2. Add zeroes at the beginning of the outcome until the number of digits becomes an even number of 4 or larger.
   e.g.) 152399025 has 9 digits, so a single 0 is added. The number becomes 0152399025.
3. Extract the middle four digits and assign it as the new seed.
   e.g.) 015**2399**025 $\rightarrow$ **2399**. New seed = 2399.
4. Go back to initial step.

The problem with this method is that firstly, once the seed degenerates to zero, the following numbers will all be zero and secondly, that the sequence can end up in a cycle (e.g. 6100 $\rightarrow$ 2100 $\rightarrow$ 4100 $\rightarrow$ 8100 $\rightarrow$ 6100) [2].

Pseudorandom numbers are predictable, periodic, and correlated. While pseudorandom numbers produced by more recent methods are less predictable and have longer periods, they still cannot be considered true random numbers.

Classical Physical Random Number Generators The simplest form of classical random number generation is through coin tosses: when the coin yields heads the generator outputs 1, and when the coin yields tails it outputs 0. This example captures the essence of classical physical random number generation, which is that a classical physical phenomena is measured and encoded to binary numbers.

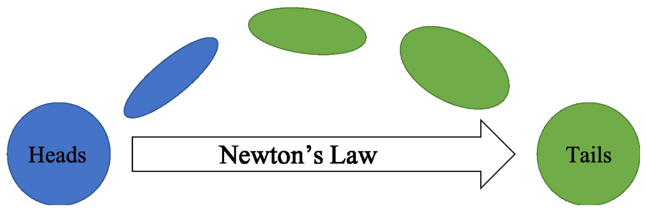

Fig. 3. Conceptual diagram of classical physical sources.

In classical physical random number generation, the initial condition of the physical phenomena is the seed. The problem is that in classical physics, all future measurement values are determined by this initial condition. Therefore, like pseudorandom numbers, classical physical random numbers are deterministic and thus predictable in theory [1].

Quantum Physical Random Number Generators A quantum physical random number generator is similar to its classical counterpart. The only difference is that the source is quantum physical instead of classical. Unlike classical physics, the measurement results in quantum physics cannot be determined by the initial condition [1]. This is due to the stochastic laws of quantum physics which state that the measurement results are not predetermined.

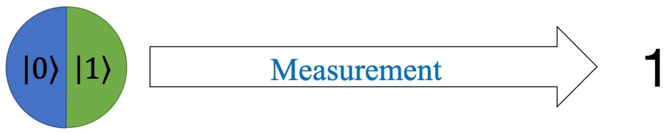

Fig. 4. Conceptual diagram of quantum physical sources.

A qubit is a quantum coin. While a classical coin is either heads or tails at all times, a qubit can be in multiple states at once (superposition). For example, Figure 4 shows a qubit that is initially prepared as a superposition between $|0\rangle$ and $|1\rangle$. Note that $|0\rangle$ is a state that when measured, its measurement value is 0 with certainty. Likewise, $|1\rangle$ always yields 1. One can control the probability of 0s and 1s at will, creating a quantum coin whose output is unpredictable.

Various quantum phenomena can be considered qubits. One example is the two paths of a photon after passing through a beam splitter as shown in Fig. 5.

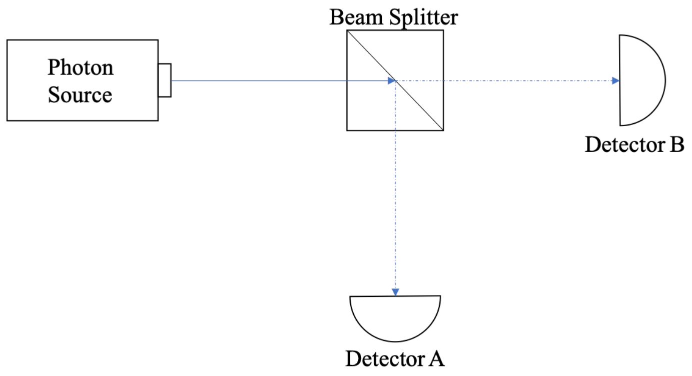

Fig. 5. Quantum random number generation using a photon source [1].

When a photon is detected by detector A, the output is 1. If it is detected by detector B, the output is 0. Optical qubits are a popular means of quantum random number generation. Firstly, optical qubits are easy to prepare compared to other types such as superconducting qubits. Secondly, optical methods tend to achieve a higher generation rate compared to other quantum physical methods. As large samples are required in both application and randomness testing, optical qubits are convenient.

2.4 Tests of Randomness

The goal of randomness testing is to statistically evaluate the generator's capability of producing unbiased and uncorrelated bits. This is done by examining the generator's output. Common test suites are the NIST test suite [3], TestU01 [4], and the dieharder test [5].

Let there be a coin with certain probabilities for yielding heads and tails. By flipping the coin 10 times, one obtains a sequence of heads and tails of length 10. Randomness tests aim to decide whether the sequence in question was obtained from an ideal random number generator that outputs unbiased and uncorrelated bits. This is done with the following steps.

1. Obtain a sample of appropriate length.
   e.g.) 0111111111
2. Characterize the sample with a test statistic.
   e.g.) the number of 1s within the sequence is 9.
3. Calculate the probability (p-value) that an ideal random number generator outputs a sample with such a test statistic.
   e.g.) the probability that an ideal random number generator outputs a sequence of length 10 containing 9 ones is approximately 0.009765625.
4. Decide whether the probability (p-value) is acceptable by comparing it to the level of significance α.
   e.g.) if $\alpha = 0.01$, then p-value $< \alpha$. In this case, the test result is that the sample was not obtained from an ideal random number generator.

One could apply various tests by characterizing the sample using various test statistics. One must, however, keep in mind that the ideal probability distribution of the test statistic must be known beforehand.

2.5 Post-processing

True random number generators have physical sources. The process of physical random number generation begins with the measurement of some observable value of a physical system as shown in Fig. 6. At this stage, the effect of classical noise becomes a matter of concern. The measured values are then encoded into binary information. At this encoding stage, errors must be taken into consideration. As a result of these properties, physical random number generators require post-processing where noise and errors are excluded as much as possible.

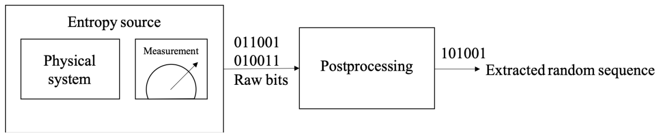

Fig. 6. The process of random number generation based on physical sources [1].

von Neumann Extractor The simplest example of a randomness extractor would be the von Neumann extractor. This extractor interprets the input binary sequence as pairs and translates 01s to 0s and 10s to 1s [6]. This extractor can only be effectively used for certain sources, namely, independent and identically distributed (iid)-bit sources [7]. Iid-bit sources are sources that consist of discrete random variables $X_1, X_2, \cdots, X_n \in \{0, 1\}$ where for any $i \in \{1, \cdots n\}$, $\Pr[X_i = 1] = \delta$ for some unknown constant δ[9, Chapter 6]. A coin toss with a constant bias throughout the tosses is, for instance, an iid-bit source. Such a source should yield 01s and 10s with equal probability, hence the method turns the biased sequence into a uniform one.

Samuelson Extractor The Samuelson extractor is similar to the von Neumann extractor, except that it translates 10s to 0s and 11s to 1s [8]. It is often the case that the use of the Samuelson extractor is followed by the use of the von Neumann extractor. While using both extractors reduces the sequence to 1/4 on average[9, Chapter 6], this is sometimes necessary as the Samuelson extractor targets different sources compared to the von Neumann extractor. This extractor is effective when used on a source where the previous bit affects the probability of the next bit. For example, if a source is more likely to produce $0 \to 1$ than $1 \to 0$, the von Neumann extractor cannot be used. The Samuelson extractor, however, fixes the previous bit to 1, which turns the input into a sequence expected from an iid-bit source.

3 Experimental Procedure and Results

In light of the recent trend of quantum random number generation and quantum computing, we wonder if actual quantum computers are capable of producing random numbers. The following experiment aims to answer this question.

3.1 Random Number Generation

In this section, the procedure for generating random numbers on a quantum computer is introduced. A quantum computer is a device that contains multiple qubits. The IBM 20Q Tokyo device contains 20 qubits. Below are the steps for random number generation.

1. Prepare all qubits in the state of superposition between state $|0\rangle$ and $|1\rangle$, so that both states have equal probabilities. This can be achieved with the use of the Hadamard gate.
2. Measure all qubits once, and collect the result.
   e.g.) $|00101101 \cdots 1\rangle$.
3. Return to step 1.

By repeating the procedure above and concatenating the results in order, a sequence with a length of 43,560 was obtained. This procedure was conducted in July of 2018. It took approximately one month to obtain a sequence with a length of 43,560. Let us call this raw sample rand43560.

When the user specifies how many times which algorithm should be run, the IBM 20Q Tokyo device returns a histogram showing the probability of each measurement outcome. In order to obtain a sequence of measurement outcomes, we set the number of runs at 1 and kept running the same algorithm. This way, the histogram shows the measurement outcome of each run.

3.2 Post-Processing

The three post-processing extractors presented below were used.

1. The von Neumann Extractor: 01 $\to$ 0, 10 $\to$ 1.
   Sample name after post-processing: Neumann
2. The Samuelson Extractor: 10 $\to$ 0, 11 $\to$ 1.
   Sample name after post-processing: Samuelson
3. A combination of the above: Samuelson $\to$ Neumann.
   Sample name after post-processing: SN

The resulting samples were statistically examined along with the raw sample rand43560.

3.3 Test Results of Post-Processed Samples

The first 6 tests from the NIST test suite were applied to the respective samples. Figure 7 shows the result of the frequency test, which is a test for uniformity. The test statistic z is defined as below, and is known to follow a standard normal distribution.

$$z \equiv \frac{x - np}{\sqrt{np(1-p)}} \tag{1}$$

Note that x is the number of 1s in the sample, n is the length of the sample, and p is the ideal probability of 1s, which is 0.5. z yields 0 for a sample that contains equal proportions of 0s and 1s.

Figure 7 shows the values of z for each sample, as well as the corresponding probability densities. The probability that an ideal random number generator generates a sample with a value of z falling into the regions colored in green is lower than 1 %. When the value of z is in the green region, the sample is not uniform. This decision is based on a level of significance of $\alpha = 0.01$.

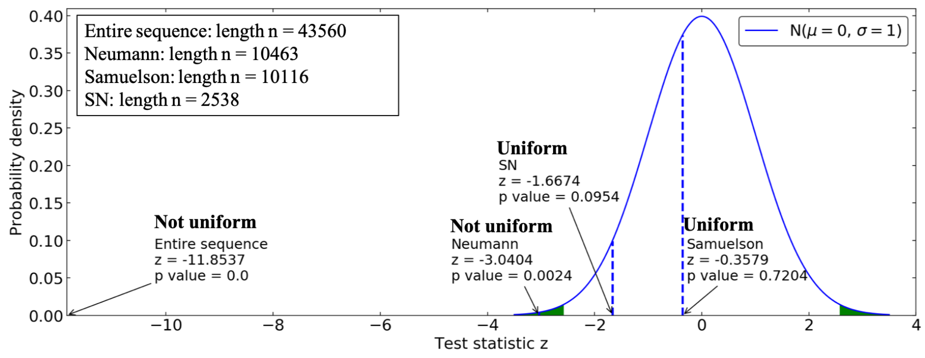

Fig. 7. Uniformity of rand43560 before and after randomness extraction.

We observe that the raw sample rand43560 is not uniform. This points to the possibility that the qubits were not prepared to yield equal probabilities of 0s and 1s, which provides insight into the accuracy of the gates and measurement of the device. Another reason is that the device is unstable. With the device going through calibration on a daily basis, it is questionable whether the device maintains the same properties from one day to another.

The fact that the von Neumann extractor failed to sufficiently correct the bias in the sequence suggests that the 20 qubits cannot be considered an iid-bit source.

The Samuelson extractor and the combination of the Samuelson extractor and the von Neumann extractor, on the other hand, proved effective. The effectiveness of the former extractor implies a Markovian property in the sequence. This means that the output of adjacent qubits tend to be correlated. The latter extractor is not as effective as the former, which suggests that even after the Samuelson extractor is applied, the sequence is still not iid.

Table 1. The NIST test results and corresponding p-values of samples.

Test	Rand43560	Neumann	Samuelson	SN
Length	43560	10463	10116	2538
Frequency	0.0000	0.0024	0.7204	0.0954
Frequency (block)	0.0000 (396)	0.0052 (126)	0.0161 (85)	0.2099 (22)
Runs	0.0000	0.1135	0.2333	0.7388
Runs (block)	0.0000 (128)	0.3125 (128)	0.0325 (128)	0.3952 (8)
Matrix rank	0.5285	0.9523	0.7138	0.7419
DFT	1.0000	0.1493	0.0003	0.4072
Result	Fail	Fail	Fail	Pass

Table 1 shows the p-values corresponding to the 6 tests from the NIST test suite of the respective samples. If a p-value is lower than $\alpha = 0.01$, the test is a failure. Unless the sample passes all six tests, it is not regarded as random. Block means that the sequence was divided into blocks of certain lengths, which are shown next to the p-values in brackets.

According to Table 1, only sample SN passes all 6 tests. This could be due to the fact that the size of the sample is small. In general, smaller samples are more likely to pass these tests.

4 Conclusion

The sequence of length 43,560 was obtained by repeatedly performing a quantum coin toss using all 20 qubits of the IBM 20Q Tokyo device. Following the conventional treatment of physical random number generation, three post-processing extractors were applied to the raw sample. The extractors are the von Neumann extractor, the Samuelson extractor, and the two combined.

As a result of applying the first 6 tests from the NIST test suite, it was revealed that only the combination of the von Neumann and Samuelson extractors succeeded in producing a sample that passes all 6 tests. Due to the limited sample size, however, it was not clear whether this method was effective.

Regarding the generation rate (approximately 43,560 digits per month) and precision of the device, using the IBM 20Q Tokyo device as a QRNG is not a practical idea. However, examining the quality of the output quantum random numbers may lead to benchmarking of actual quantum computing devices.

5 Open Questions

While using the IBM 20Q Tokyo device as a quantum random number generator is impractical, producing quantum random numbers with the device may not be an entirely pointless attempt.

In theory, quantum computers should be capable of producing quantum random numbers without post-processing. Therefore, the reason why the IBM 20Q Tokyo device failed to achieve this is worth pursuing.

A quantum computer is a multipurpose device, and judging its condition in light of a single purpose is of no use. However, there is a need for a comprehensive indicator of the device's condition, such that the user could refer to that indicator when deciding which device to use and when to use it. We believe that the quality of quantum random numbers produced by the device is a potential candidate of such an indicator.

The following are some open questions.

1. How can the quality of quantum random numbers and generators be quantified and compared?
2. What is the relationship between the quality of the random numbers and the device's condition?
3. Given that the quality of the random numbers is poor, what can be done to improve their quality?
4. How does the poor quality of random numbers affect other algorithms run on the same device?
5. What is the relationship between randomness and quantum computers?

Acknowledgement

This work is partially supported by JSPS KAKENHI (Grant Nos. 17K05082 and 19H05156). The results presented in this paper were obtained in part using an IBM Q quantum computing system as part of the IBM Q Network. The views expressed are those of the authors and do not reflect the official policy or position of IBM or the IBM Q team.

References

1. Herrero-Collantes, M., Garcia-Escartin, J. C.: Quantum Random Number Generators. Rev. Mod. Phys. **89**, 015004 (2017)

2. Knuth, D. E.: The Art of Computer Programming, Volume 2: Seminumerical Algorithms. 3rd edn. Addison-Wesley Longman Publishing Co. Inc., Redwood City, CA, USA (1997)
3. Bassham, L. E., Rukhin, A. L., Soto, J., Nechvatal, J. R., Smid, M. E., Leigh, S. D., Levenson, M., Vangel, M., Heckert, N. A., Banks, D. L.: A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic, NIST Special Publication **800**, 163 (2001)
4. L'Ecuyer, P., Simard, R.: TestU01: A C library for empirical testing of random number generators. ACM Trans. Math. Softw. (TOMS) **33**, 22 (2007).
5. Brown, R. G., Eddelbuettel, D., Bauer, D.: `https://webhome.phy.duke.edu/~rgb/General/dieharder.php` (Version 3.31.1, checked by online on April 25, 2019).
6. von Neumann, J.: Various techniques used in connection with random digits. J. Res. Nat. Bur. Stand. Appl. Math. Series **3**, 36 – 38 (1951)
7. Kwok, S. H., Ee, Y. L., Chew, G., Zheng, K., Khoo, K., Tan, C. H.: A Comparison of Post-Processing Techniques for Biased Random Number Generators. In: Ardagna, C. A., Zhou, J. (eds.): Information Security Theory and Practice. Security and Privacy of Mobile Devices in Wireless Communication. WISTP 2011. Lecture Notes in Computer Science **6633**, pp. 175 – 190, Springer, Berlin, Heidelberg (2011)
8. Samuelson, P.: Constructing an Unbiased Random Sequence. J. Am. Stat. Assoc. **63**, 1526 – 1527 (1968)
9. Vadhan, S.: Pseudorandomness. Found. Trends Theor. Comput. Sci. **7**, 1–336 (2012)