Pseudo random generator based on irrational numbers

Vesna Dimitrievska Ristovska

Faculty of Computer Science and Engineering,
Ss. Cyril and Methodius University
16, Rugjer Boshkovikj str., 1000 Skopje, Macedonia,
vesna.dimitrievska.ristovska@finki.ukim.mk

Abstract. In this paper we propose an algorithm for designing a pseudo-random number generator (PRNG) using digits of irrational numbers, like π, e, $\sqrt{2}$, $\sqrt{3}$, $\sqrt{5}$. In order to check the randomness of the generated sequences we use statistical tests from Diehard battery. In our algorithm, we use about $3 \cdot 10^7$ digits of an irrational number. Many experiments are made and after analyzing of the results from statistical tests we conclude that proposed PRNG passes the Diehard tests very well.

Keywords: PRNG, irrational numbers, statistical tests, Diehard battery, randomness

1 Introduction

Pseudo-random number generator (PRNG) is a mechanism which produces a sequence of numbers $s_1, s_2, \ldots$ with a given distribution which is supposed to be uniform, where $s_1, s_2, \ldots$ are elements of a given set of numbers. A sequence of random numbers $s_1, s_2, \ldots$ should have two important properties: uniformity (i.e. they are equally probable everywhere) and independence (i.e. the current value of a random variable has no relation with the previous values). In practice, we cannot construct a perfect PRNG, since the way we are building the mechanism is not a random one, which affects the uniformity and independence of the produced sequences. That is why the word "pseudo" is used and we have to measure the randomness of the obtained sequences.

Also, a good random number generator should have some additional qualities as simplicity and large period.

In the following subsection there are some historical facts recall on L'Ecuyer (2017) [8] about PRNG's which use irrational numbers.

1.1 Background and overview of related works

The idea of using successive digits of π, e or other transcendental numbers in order to generate a random sequence appeared very early. Metropolis et al. (1950) [10] succeeded to compute 2000 decimals of π and e, and found that this sequence

passes elementary statistical tests. This testing was extended to the first 10000 decimals by Pathria (1962) [11] and to 100000 decimals by Esmenjaud-Bonnardel (1965) [3], and no problem was found with passing elementary statistical tests.

Later, longer sequences of digits of π have been obtained and tested and many papers have elaborated and discussed this idea. Till now, the world record (in 2016) was 22459157718361 decimal digits of π, computed in about four months by Peter Trueb using an algorithm of Chudnovsky and Chudnovsky (1989) [2], Bellards formula, and the Y-Cruncher multi-threaded software (Yee 2017) [14].

However, to justify that the successive digits of π (or any other given irrational number) in a given base b can be taken as random, it would be good to know that this sequence of digits is uniformly distributed in base b, i.e., that each of the b possible digits appears with frequency $\frac{1}{b}$ (on average) in the infinite sequence.

Empirical counting for π, over several digits suggests that this is true, but there is no known proof of it. Also, this uniform distribution of the digits is far from sufficient; we need to have the uniform distribution of the pairs, triplets, and so on.

Beginning with Borel (1909) [1], by Weyl (1916) [13], B. Korobov (1948) [6], Kuipers and Niederreiter (1974) [7] the independence of the successive digits is captured by some stronger properties, introduced definitions and concepts.

In the latest researches, there are some trials to design a PRNG using digits of any irrational number since irrational numbers have decimal expansions that neither terminate nor become periodic, practically their decimal expansion has infinite period. In [12], the authors proposed an algorithm for pseudo-random 5-digit numbers using the digits of π and made some visual and statistical analyses for goodness of proposed generator.

In this paper, the main idea is using digits of any irrational number (π, e, $\sqrt{2}$,$\sqrt{3}$, $\sqrt{5}$, ...) to generate random numbers, and practically this idea is a generalization of the idea in [12]. Our investigation shows that our new proposed algorithm is efficient and produces sequences with large period.

There are a lot of tests for such measurements and all of them measure the difference between the generated pseudo-random sequences and the theoretically supposed ideal random sequence. We say that a PRNG passes a test if the random sequences produced by that PRNG pass the test with a probability near to 1. We can classify PRNGs depending of the tests they have passed. So, for obtaining a better classification we should have many different tests.

This paper is organized as follows. In Section 2 we give an explanation for our new generator of pseudo random numbers and next in Section 3 we present the algorithm for the generator. In Section 4 we explain statistical tests from Diehard battery. In Section 5 and Section 6 are given the tabulate results and some conclusions.

2 Some ideas for designing a PRNG using digits of irrational numbers

Diehard tests require precise format of the numbers whose randomness they test. Namely, the file of the numbers should be a binary file of a hexadecimal numbers with approximately 11 MB size. There should be ten numbers in each row, about 2 870 000 numbers in the file and the maximum number in the file should be $max = 2^{32} - 1$.

Let explain the ideas for designing a PRNG using digits of an irrational number by the example in which we use the digits of π. In this example we will use sequential 10-tuples of digits from the decimal expansion of π. Using some database we will take about $3 \cdot 10^7$ digits of π. The next step is generating decimal 10-digit number from every 10-tuple.

We consider 4 ideas for this generators.

1. We check the obtained 10-digit number and if its value is greater than max, we ignore this number and go to the next 10-tuple until we generate a number that is smaller than $max + 1$. These steps will be repeated until we obtain 2 870 000 numbers that are smaller than $max + 1$.
2. We will obtain a faster algorithm if we check the first digit instead checking the whole decimal number. If the first (left most) digit c in the 10-tuple is greater than 4, then the digit c is ignored and we consider 10-tuple which start with the next smaller than 5 digit in the decimal expansion of π.
3. It is easy to check that the algorithm extremely increases its speed if we transform the first digit in the 10-tuple which is greater than 4 instead of ignoring it. If the first digit in the 10-tuple is greater than 4, then we use the operation mod 5 and this is how we obtain the digit 0, 1, 2, 3 or 4 instead of 5, 6, 7, 8 or 9. If the new obtained number is greater than max, (its first digit is 4) we set the second digit to be 1 and the obtained number is smaller than $max + 1$. Unfortunately, on this way we can obtain sequences which distribution is not uniform since the number of 10-tuples which start with (4,1) is greater than number of 10-tuples which start with other pair of digits.
4. The next idea for improving the proposed algorithm is to scale the obtained 10-digit number $0 \leq number < 10^{10}$ in the range $0 \leq number' < 2^{32}$. In this case there is no need to check if the last number is greater than max.

3 The new algorithm for pseudo random number generator using digits of an irrational number

In our algorithm we will use the last idea given in Section 2 since if the obtained 10-tuples (using digits of an irrational number) are uniformly distributed, the scaling will not corrupt that uniformity. Also, very important for a PRNG is to produce different numbers each time we started generating of a new sequence. Therefore, we must initialize the beginning pointer to an arbitrary digit of the

chosen irrational number. The position of the beginning pointer will be an input parameter. Also, the input parameter will be the maximal length n of digits for generating of each number in the sequence (in the previous example, we chose $n = 10$). Let stress that we compute the digits of any irrational number using package *Mathematica*.

Algorithm for pseudo random number generator using digits of an irrational number

1. Choose an irrational number which digits will be used for generating random numbers.
2. Set the maximal length n of digits for generating of each number in the sequence, the position s of the beginning pointer (the first digit where the generating starts) and the maximum of the generated numbers max.
3. Let $counter = 0$.
4. Until $counter <= 2870000$ do
   4.1. Use slice size of n digits to generate a number r.
   4.2. Scale $r \leftarrow \left[\frac{r}{10^n - 1} \cdot max\right]$
   4.3. Put pointer position $s \leftarrow s + n$.
   4.4. $counter = counter + 1$.

We note that software realization of this algorithm and many experiments was done using package *Mathematica*.

4 Diehard tests

George Marsaglia has developed Diehard tests as a battery of statistical tests for measuring the quality of a random number generator. This battery was developed over several years and first published in 1995. It consists of 15 statistical tests, and it is a comprehensive set of statistical tests for PRNG and serves as some kind of litmus for checking and certification of PRNG. If a PRNG passes Diehard statistical tests, then it can be used in deeper scientific researches.

In the next, we give an overview of these tests [5], [9] and [4].

1. **Birthday Spacings Test.** For this test we choose m birthdays in a year of n days. The spacing between the birthdays needs to be listed. If the number of values that occurred more than once in the list is the variable J, than it has asymptotic Poisson distribution with mean $\frac{m^3}{4n}$. For a sample of 500 s a chi-square test is performed to provide a p value. At the end a Kolmogorov Smirnov (KS) test is taken for the acquired 9 p-values.
2. **Overlapping 5-Permutation Test (OPERM-5).** The overlapping 5 permutation test gets its name since it takes 5! possible orderings of five consecutive integers. These tests are overlapping m-tuple tests for which elements of the overlapping 5-tuples are not independent, or even successive states of a Markov chain.

3. **Binary rank tests.** Binary rank tests use some of the characteristics of the matrices and their ranks. N-dimensional cube is taken using the columns of a matrix as axes.
   3.1. 31×31 **Binary Matrix.** A 31×31 binary matrix is generated formed of 31 integers in each row. 31 bits are being used (only the last bit is not taken) of those 31 integers.
   3.2. 32×32 **Binary Matrix.** A 32×32 binary matrix is random generated formed of 32 integers in each row. If the rank of the matrix is r then $0 \leq r \leq 32$. But in practice ranks are rarely less than 29, so if that is the case those kinds of ranks are being combined with those for rank 29.
   3.3. 6×8 **Binary Matrix.** A 6×8 binary matrix is generated from 6 random integers and 8 bits from those integers. If the rank of the matrix is r then $0 \leq r \leq 6$. But in practice ranks are rarely less than 4, so if that is the case those kinds of ranks are being combined with those for rank 4.
4. **Bitstream Test.** The file that is tested is considered as a stream of bits. So called 20 letter words are taken, overlapping between each other. The first word is from the 1st to 20th bit, then the second from 2nd to 21st bit etc.
5. **Test OPSO (Overlapping pairs sparse occupancy test)**. In this test a 2-letter words from an alphabet of 1024 letters are being considered. Each of the 2 letters are determined by a specified 10 bits from a 32 bit integers in the sequence to be tested.
6. **Test OQSO (Overlapping quadruples sparse occupancy test).** Similar as in OPSO, in this test a 4-letter words from an alphabet of 32 letters are being considered. Each of the 4 letters are determined by a specified 5 bits from a 32 bit integers.
7. **Test DNA.** Like the previous 2 tests, in this test a 10-letter words from an alphabet of 4 letters C, G, A, T are being considered. Each of the 10 letters are determined by a specified 2 bits from a 32 bit integers.
   8.1 **Count the 1's Test on a stream of bytes.** As the tests name suggests the number of 1's in a stream of bytes is counted. Each byte can contain from 0 to 8 1's with probabilities: $\frac{0}{256}, \frac{8}{256}, \frac{28}{256}, \frac{56}{256}, \frac{70}{256}, \frac{56}{256}, \frac{28}{256}, \frac{8}{256}, \frac{0}{256}$.
   8.2. **Count the 1's Test for specific bytes.** In this test as its name suggests the number of 1's in specific bytes from each 32 integer are being counted. From each integer, a specific byte is chosen, say the left-most: bits 1 to 8. Each byte can contain from 0 to 8 1s with different probabilities: $\frac{1}{256}, \frac{8}{256}, \frac{28}{256}, \frac{56}{256}, \frac{70}{256}, \frac{56}{256}, \frac{28}{256}, \frac{8}{256}, \frac{1}{256}$. The letters are determined by the number of 1's.
9. **Parking test.** In this test we park a car (it is a circle with radius 1) in a square of side 100 (so the square is with 100×100 size). Then we do the same with the second, third car and so on. If a crash occurs when we try to park a car, the process for that particular car is repeated from the beginning choosing different random location for parking.
10. **Minimum Distance Test.** Again we take a square but now with a side of 10 000 choosing 8 000 random points in it. If we denote the minimum distance between $\frac{n(n-1)}{2}$ pairs of random points with d, and if the points

are independent and uniformly distributed, then d^2 should be exponentially distributed with mean 0.995.

11. **3D Spheres Test.** Here we take a cube with side 10 000 and choose 4 000 random points in it. At each point, center a sphere large enough to reach the next closest point. Then the distribution of the volume of the smallest such a sphere is found and it is approximately exponentially distributed with mean $\frac{120\pi}{3}$.
12. **Squeeze Test.** In this test random integers are floated to get uniform distributions on [0, 1).
13. **Overlapping Sums Test.** Let $m \geq 100$ be a fixed integer. Take a sequence of independent and identically distributed $U(0,1)$ random variables $U_1, U_2, \ldots$ and form the overlapping sums $S_1 = U_1 + U_2 + \ldots + U_m$, $S_2 = U_2 + U_3 + \ldots + U_{m+1}$, and so on. The random variables $S_i, i = 1, 2, \ldots, m$ are virtually normal.
14. **Runs Test.**The RUNS test counts the number of runs up and run downs in a sequence of 10 000 uniform variables [0,1) acquired by floating the 32-bit integers from the specified file.
15. **Craps Test.** This test is somehow connected with the Craps game. The test plays $n \geq 200000$ games of craps and counts the number of wins and the number of throws necessary to end each game.

We will note that the 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 cumulative distribution function 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. Therefore $p < 0.025$ or $p > 0.975$ means that the RNG has "failed the test at the 0.05 level".

5 Randomness results obtained from Diehard tests

In the next three tables we will present some of the obtained results from the our new proposed algorithm and applied Diehard tests. In the Table 1 there is a brief preview of the results: in the third column there is the number of the idea from Section 2, in the sixth column is the time (in seconds) for constructing file in corresponding format (about 11MB size) for Diehard testing.

In Table 2 and Table 3 we have shown detailed information for the sequences presented in Table 1, for all tests in Diehard battery.

Table 1. Results- brief preview

irr. number	seq. name	idea	n	s	time (in sec.)	success of Diehard tests
π	Seq.1	3	10	1	506	64 %
π	Seq.2	4	10	1	400	92 %
e	Seq.3	4	10	1	577	95 %
$\sqrt{2}$	Seq.4	4	10	1	678	98 %
$\sqrt{3}$	Seq.5	4	10	6	636	97 %
$\sqrt{3}$	Seq.6	4	12	2	653	92 %
$\sqrt{5}$	Seq.7	4	10	3	668	96 %

Table 2. Results from tests in Diehard battery, part 1

		Birth-Test	OPERM-5	Binary-31	Binary-32	Binary-6x8	Count-Stream	Parking	Minim. D	3D Spheres
Irrat. number	Seq. name	p-value								
π	Seq.1	0.929	0.631 0.004	0.417	0.462	1	1 1	0.881	0.668	0.832
π	Seq.2	0.251	1 0.203	0.635	0.780	0.098	0.005 0.734	0.384	0.697	0.685
e	Seq.3	0.262	0.535 0.341	0.340	0.324	0.486	0.946 0.986	0.410	0.182	0.752
$\sqrt{2}$	Seq.4	0.368	0.076 0.566	0.778	0.321	0.537	0.641 0.683	0.730	0.812	0.744
$\sqrt{3}$	Seq.5	0.397	0.911 0.805	0.321	0.861	0.823	0.244 0.235	0.308	0.57	0.92
$\sqrt{3}$	Seq.6	0.23	0.294 0.704	0.684	0.643	0.838	0.365 0.285	0.185	0.004	0.278
$\sqrt{5}$	Seq.7	0.823	0.935 0.259	0.558	0.675	0.215	0.220 0.136	0.050	0.346	0.761

6 Conclusion

From the realized experiments (with different parameters) and from presented results in the previous tables, we can conclude that all generated sequences (with the fourth idea) with the new PRNG, independent from the initial irrational number very well pass the statistical tests from the Diehard battery.

It is a good direction to work deeply over this algorithm and generalize it.

Table 3. Results from tests in Diehard battery, part 2

		Bit stream	OPSO	OQSO	DNA	Count Bytes	Sqeeze	O-SUM	Run Test	Craps Test
Irrat. number	Seq. name	No. of passed tests / total no. of tests					p-value	p-value	p-value	p-value
π	Seq.1	0/20	15/23	19/28	22/31	17/25	0.916	0.004	0.130 0.087 0.042 0.013	0.59 0.776
π	Seq.2	20/20	22/23	26/28	30/31	22/25	0.947	0.674	0.839 0.246 0.841 0.193	0.441 0.764
e	Seq.3	18/20	21/23	25/28	30/31	22/25	0.506	0.494	0.131 0.350 0.544 0.223	0.715 0.705
$\sqrt{2}$	Seq.4	20/20	18/23	28/28	28/31	22/25	0.141	0.501	0.958 0.139 0.340 0.608	0.260 0.406
$\sqrt{3}$	Seq.5	19/20	21/23	28/28	31/31	23/25	0.449	0.031	0.121 0.017 0.823 0.404	0.24 0.031
$\sqrt{3}$	Seq.6	20/20	17/23	25/28	30/31	24/25	0.535	0.293	0.47 0.683 0.781 0.403	0.391 0.919
$\sqrt{5}$	Seq.7	20/20	21/23	25/28	29/31	22/25	0.042	0.664	0.148 0.994 0.415 0.879	0.214 0.545

References

1. Borel, E.: Le continu mathmatique et le continu physique, Rivista di scienza 6, pp. 21–35, 1909
2. Chudnovsky D., and Chudnovsky G.:The computation of classical constants, Proceedings of the National Academy of Sciences of the USA 86, pp. 8178–8182, 1989
3. Esmenjaud-Bonnardel M.: Etude Statistique des Decimales de π, Revue Francaise de Techniques Informatiques 8 (4), pp. 295–306, 1965
4. Gjorgjievski S., Bakeva V., Dimitrievska Ristovska V.: Relation between statistical tests for pseudo-random number generators and diaphony as a measure of uniform distribution of sequences, Kulakov A., Stojanov Gj.(Eds.): Advances in Intelligent Systems and Computing, ICT Innovations 2016, Springer (in print)
5. http://stat.fsu.edu/pub/diehard/

6. Korobov, N. M.: On Functions with Uniformly Distributed Fractional Parts, Doklady Akad. Nauk SSSR 62, pp. 21–22, 1948
7. Kuipers H.L., and Niederreiter. H.: Uniform Distribution of Sequences, New York, NY: John Wiley, 1974
8. L'Ecuyer P.: History of uniform random number generation, Chan W. K. V., DAmbrogio A., Zacharewicz G., Mustafee N., Wainer G., and Page E., eds.: DIRO, GERAD, and CIRRELT, Proceedings of the 2017 Winter Simulation Conference, 2017
9. Marsaglia G., Tsang W. W.: Some difficult-to-pass tests of randomness, Journal of Statistical Software, Volume 7, Issue 3, 2002
10. Metropolis N., Reitwiesner G., and von Neumann J.: Statistical Treatment of Values of First 2000 Decimals Digits of e and p Calculated on the ENIAC, Mathematical Tables and Other Aids to Computation 4, 1950
11. Pathria, R. K.: A Statistical study of randomness among the first 10000 digits of π, Mathematics of Computation 16, pp. 188–197, 1962
12. Rogers I., Harrell G., Wang J.: Using π digits to Generate Random Numbers: A Visual and Statistical Analysis, Int'l Conf. Scientific Computing — CSC'15 —
13. Weyl H.: U ber die Gleichverteilung von Zahlen mod. Eins. Math. Ann. 77, pp. 313-352, 1916
14. Yee A. J.: Y-CruncherA Multi-Threaded Pi-Program, 2017