Randomness from Deterministic Chaos: A Novel Algorithm for Random Number Generation

Sajad Ahmad Mir(✉) and Puneet Sharma

Indian Institute of Technology Jodhpur, Jodhpur 342030, India
{mir.1,puneet}@iitj.ac.in

Abstract. In this article, we address the problem of generation of pseudo-random sequences using deterministic chaotic maps. Using a set of three deterministic chaotic maps, we propose an algorithm for generation of pseudo-random sequences. We establish that the sequences generated through the algorithm are random in accordance with the criteria prescribed by the NIST 800-22 Test suite. We also establish the independence of output sequences with the sequences generated by any of the generator maps. The results indicate that the output sequence exhibits sufficiently high degree of randomness when the perturbations are non-dissipative.

Keywords: Random number generation · Chaotic Maps · Correlation and S Plots · Perturbation analysis · Time Series Analysis · NIST 800-22 Test suite

1 Introduction

In modern day world, the field of random number generation caters to a wide range of requirements arising across various state of the art applications. While true random number generators are computationally infeasible, pseudo-random number generators are capable of generating sufficiently long "nearly random" sequences that share many properties with truly random sequences and can be applied to address many modern-day technological challenges. The field has found applications in several areas, such as cryptography, communication, online gaming and many more. In particular, pseudo-random number generation has found applications in fields such as multimedia encryption schemes, system security, and generation of one-time passwords [2]. As instant generation of pseudo random sequences using algorithms of lower complexity can have a meaningful impact on current state of the art facilities, it is not only important to generate sequences with sufficient degree of randomness, it is critical to generate such sequences using algorithms with lower time complexity. While "randomness" broadly refers to the unpredictability of the system, chaos theory also investigates the unpredictability in a deterministic system over a long periods of time. It is interesting to determine whether chaotic maps can be used to generate a pseudo-random sequence [1].

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2025
O. Gervasi et al. (Eds.): ICCSA 2025, LNCS 15648, pp. 21–35, 2025.
https://doi.org/10.1007/978-3-031-97000-9_2

It is worth noting that chaotic maps are aperiodic sensitive deterministic mathematical functions capable of generating arbitrary patterns and hence can be used to generate pseudo-random sequences with sufficient degree of unpredictability. In [3], the author provides an efficient way to combine two or more Multiplicative Linear Congruential Generators (MLCGs) to propose a class of more efficient and portable random number generators. In [4], the authors propose a discrete time chaotic system-based TRNG (true random number generator) using the Henon map and the logistic map. The design utilizes the two dimensional structure of the Henon map with the one dimensional logistic map to propose a true random number generator. The proposed algorithm produces high-entropy random bit sequences without post-processing and passed all NIST 800-22 statistical tests [5]. In [6], the authors describe a portable set of software utilities for uniform generation. The utilities describe for multiple generators (streams) running simultaneously with each generator having its numbers partitioned into many long disjoint contiguous substreams. In [7], the authors introduce a software library offering a collection of utilities for the empirical statistical testing of uniform random number generators (RNGs). The investigation provides tools to perform systematic studies of the interaction between a specific test and the structure of the point sets produced by a given family of RNGs and provides a survey and a classification of statistical tests for RNGs.

In [8], the authors proposed random number generators using the Linear congruential map. In [9], the authors present good parameter sets to combine parallel multiple recursive sequences to implement random number generators with long periods and good structural properties. The author investigated the problem to combine multiple recursive generators of different sizes and also provided a code faster than previous implementations of similar generators. In [10] the authors used Ikeda maps and XOR operations over Arduino platform to generate a sequence of random numbers. In [11], the authors propose a TRNG using zigzag map for random bit generation. They investigate key chaotic properties and attributes of the map under discussion. The authors show that with certain modifications, a TRNG based on the zigzag map passes all NIST 800-22 statistical randomness tests using simple post-processing of output data. In [12], Moysis et al. created a pseudo-random bit generator utilizing multiple-digit comparison. They employed several maps to create the pseudo-random bit after comparing the outcomes of each bit produced by each map. In [13], the authors discuss various algorithms to generate pseudo-random sequences using maps such as Logistic, sine, linked sine, Renyi, Chebyshev (and many more) for various applications.

In this article, we investigate the problem of random number generation using chaotic maps such as logistic map, sawtooth map and the baker map. Using this fixed set of chaotic maps, we propose a scheme to generate a pseudo-random sequence using a linear congruence map as the decision map. The algorithm was executed and implemented in python environment. The algorithm generates a pseudo random sequence using a simple deterministic mechanism and hence generates pseudo random sequences in a fast and efficient manner (with lower time complexity). As a result of simple mechanism and lower time complexity, the algorithm can be used in various applications

such as neural networks, graphics and simulations, online gaming, OTP generation and encryption schemes. Further, as the algorithm utilizes a sufficiently long periodic sequence for map selection, the algorithm indicates a possible relation between chaotic orbits and pseudo random sequences. While some of the existing works use multidimensional chaotic maps to generate pseudo random sequences [4], the proposed algorithm uses a collection of one dimensional maps and hence provides a simple mechanism to generate the pseudo random sequences. Further, some of the works combine MLCGs to generate the desired sequences [3], such algorithms use integer keyspace and hence do not utilize the sensitivity of the underlying system. As the proposed algorithm utilizes a collection of maps with sensitive dependence on initial conditions, the generated sequences exhibit sensitivity to small changes in the keyspace.

The proposed scheme generates a pseudo-random sequence in accordance with the criteria prescribed by the NIST 800-22 Test suite. We also conduct a qualitative analysis of the sequences generated. The results indicate that if the sequence of perturbations in the generating parameters is non-dissipative ($\sum_{i=1}^{\infty} p_i = \infty$) the sequence generated exhibits sufficient degree of randomness. Further, the results indicate that if the sequence of perturbations in the generating parameters is dissipative ($\sum_{i=1}^{\infty} p_i < \infty$), the sequences generated fails to qualify the NIST 800-22 Test suite criteria for randomness. We also establish the independence of the sequence generated with sequences generated by any of the generator maps. As a result of this work, an R package **PRNG** was developed and published in the CRAN platform [14].

2 Basic Tools and Setup

2.1 Linear Congruence Map

Let $\mathbb{N}$ be the set of natural numbers $a, b \in \mathbb{N}$ and p be a **prime**. For any $z_0 \in \mathbb{N}$,

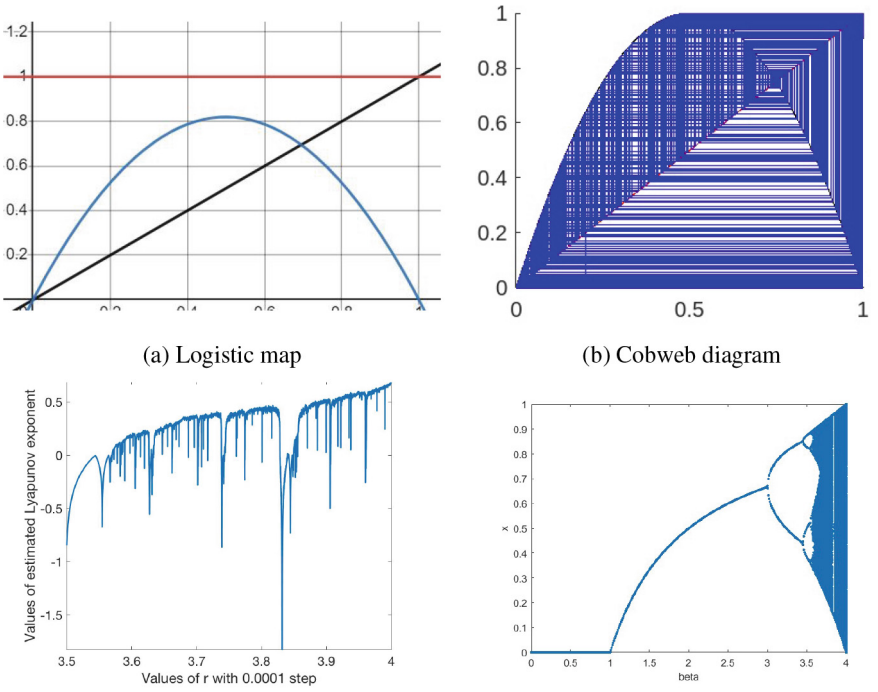
$$z_{n+1} = (az_n + b) mod(p) \tag{1}$$

generates a sequence of natural numbers. As any pseudo-random sequence generated using linear congruence is periodic in nature, we choose the values a, b and p so that the pseudo-random sequences generated are of sufficient large length. In this work, we use $a = 7^5$, $b = 0$, $p = 2^{31} - 1$(a large **prime**). In [8], Lehmer's method exemplifies the charm of simplicity to select the parameters of the linear congruence map to ensure sufficiently long sequences. The procedure involves the meticulous selection of two predetermined integer parameters a (the multiplier) and the prime p.

2.2 The Logistic Map

Let $I = [0, 1]$ be the unit interval,then for each $\mu \in [0, 4]$, $f : I \to I$ defined as

$$f_\mu(x) = \mu x(1 - x) \tag{2}$$



(a) Logistic map (b) Cobweb diagram

(c) Lyapunov exponent of logistic map (d) Bifurcation diagram

Fig. 1. Dynamical Analysis of Logistic Map.

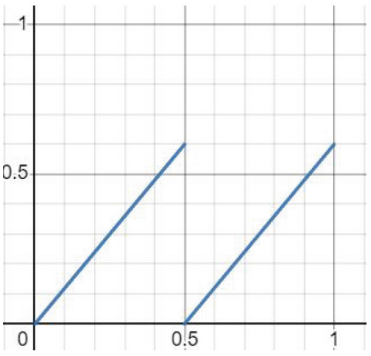
is called the logistic map with parameter μ. The map is known to be chaotic for $\mu \in [3.57, 4]$. The map is known to exhibit sensitive dependence on the initial conditions, as assessed by the **Lyapunov exponent** λ and has periodic points of all orders [1]. The long-term behavior can be predicted through the cobweb diagram.(see Fig. 1)

2.3 Baker Map

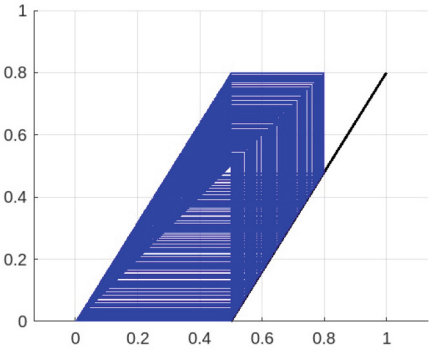
Consider the unit interval $I = [0, 1]$. Then for each $\mu \in (0.5, 1)$, the map $B_\mu : I \to I$ defined as

$$B_\mu(x) = \begin{cases} 2\mu x, & \text{if } 0 \leq x \leq \frac{1}{2}, \\ \mu(2x-1), & \text{if } \frac{1}{2} \leq x \leq 1. \end{cases} \tag{3}$$

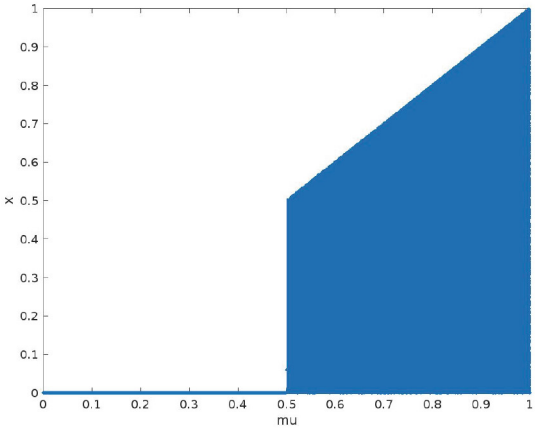
is called the Baker map. In [1] the map is found to be sensitive to initial conditions for $\mu \in (0.5, 1)$. The map along with its cobweb diagram and bifurcation diagram are shown in Fig. 2.



(a) Baker map



(b) Cobweb diagram



(c) Bifurcation diagram

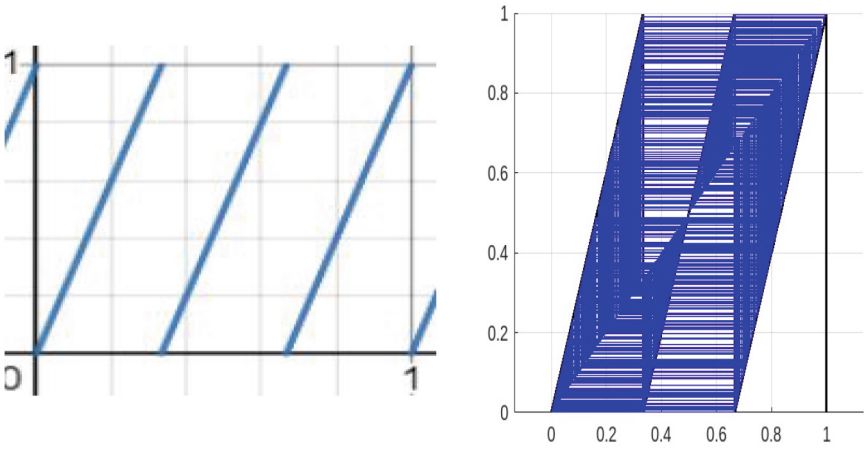
Fig. 2. Dynamics of Baker Map.

2.4 The Sawtooth Map

Consider the function $h : I \rightarrow I$ defined by

$$h(x) = 3x(mod\ 1) \tag{4}$$

where $x \in (0, 1)$. The map is called the sawtooth map and is known to be sensitive to initial conditions [1]. The map is piecewise linear, consists of linear segments or "teeth" (where each tooth represents a linear map) and is known to be topologically mixing (see Fig. 3).



(a) Sawtooth map (b) Cobweb diagram

Fig. 3. Dynamics of Sawtooth Map.

3 Proposed Algorithm

In this algorithm, we propose a pseudo-random generator using a set of three deterministic chaotic maps. We consider logistic map, Baker map and saw-tooth map as the generating maps for the proposed scheme. We use Linear congruence map as a decision map. Parameter Space:

$n_0 =$ initial seed for the Linear congruence map
$x_1 =$ the initial input for logistic map $0 < x_1 < 1$
$a_1 =$ the initial value of the parameter of logistic map $3.57 \leq a_1 \leq 4$
$x_2 =$ the initial input for baker map $0 < x_2 < 1$
$a_2 =$ the initial value of the parameter of baker map $a_2 \geq 0.5$
$x_3 =$ the initial input for saw-tooth map $0 < x_3 < 1$
$N =$ the required length of random sequence

Algorithm

INPUT: **key** $[n_0, x_1, a_1, x_2, a_2, x_3, N]$
OUTPUT: Random sequence of length N

$RanSeq \leftarrow [\,]$
for $i \in \{1 \cdots N\}$ **do**
  $n \leftarrow$ **Linear-con**(n_0)
  $n_1 \leftarrow n \mod 3$
  if $n_1 = 0$ **then**
    $x \leftarrow$ **logistic map**(x_1, a_1)
    $x_1 \leftarrow x$
    $a_1 \leftarrow a_1 + \frac{1}{\sqrt{i}}$ ▷ Update parameter a_1
  else if $n_1 = 1$ **then**
    $x \leftarrow$ **Baker map**(x_2, a_2)
    $x_2 \leftarrow x$
    $a_2 \leftarrow a_2 + \frac{1}{\sqrt{i}}$ ▷ Update parameter a_2
  else
    $x \leftarrow$ **saw-tooth**(x_3)
    $x_3 \leftarrow x$
  end if
  $RanSeq$.**append**(x)
  $n_0 \leftarrow n$
end for
return $RanSeq$

This sequence of x(i)'s$\in (0, 1)$ are the required random numbers returned as a list. Inorder to generate random bits we consider the decision $w_i = 1$ for $x_i > 0.5$ and 0 otherwise. The time complexity of the algorithm is $\mathcal{O}(N)$. Since we need to store the key only for reproducing the sequences. Therfore the space complexity of the algorithm is $\mathcal{O}(1)$

4 Testing and Analysis

4.1 Histogram Analysis

The purpose of the histogram test is to identify any bias that the algorithm may have towards a particular region in the output space. The analysis of the sequences generated by the proposed algorithm shows the uniformity in the bin heights and thus demonstrates the unbaisedness in the generated sequence. We observe that the generated sequences are uniformly distributed in the output region (Fig. 4).

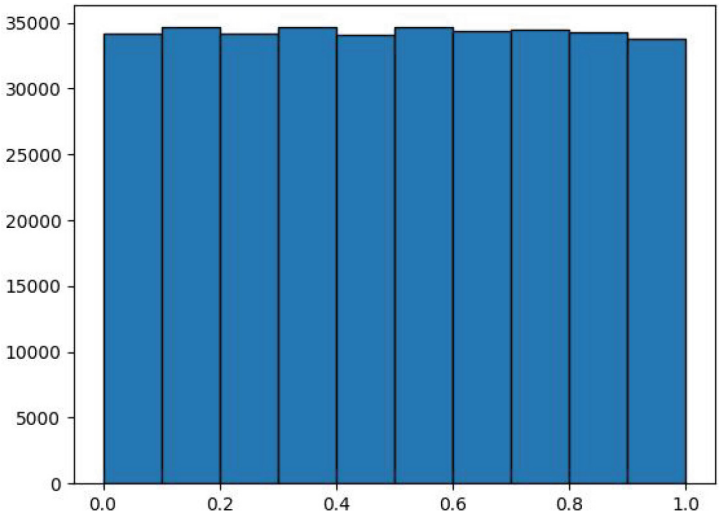


Fig. 4. Histogram analysis.

4.2 Autocorrelation Test

In order to assess the presence of any autocorrelation in the output sequences, the autocorrelation test for generated sequences of varying lengths was conducted. The analysis indicates that the sequences exhibit no significant autocorrelation, as depicted in the autocorrelation plot (see Fig. 5, conducted for sequences with a fixed length of 10^5).

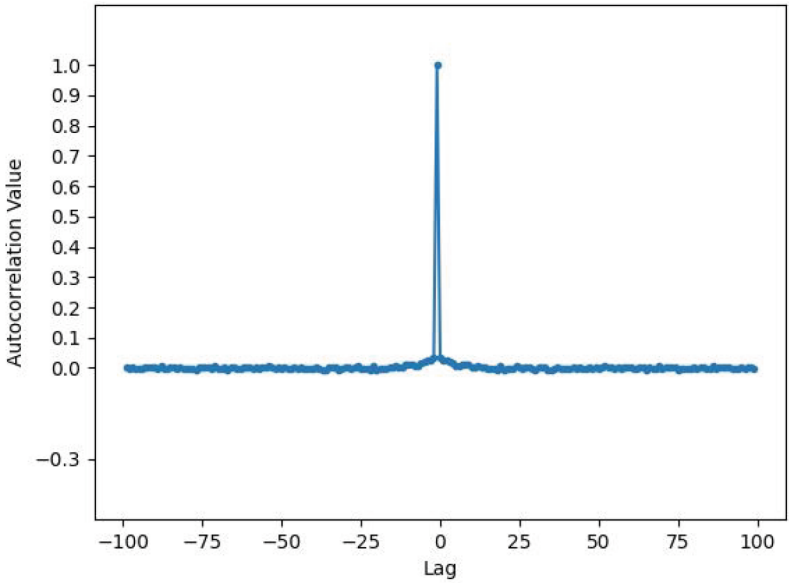


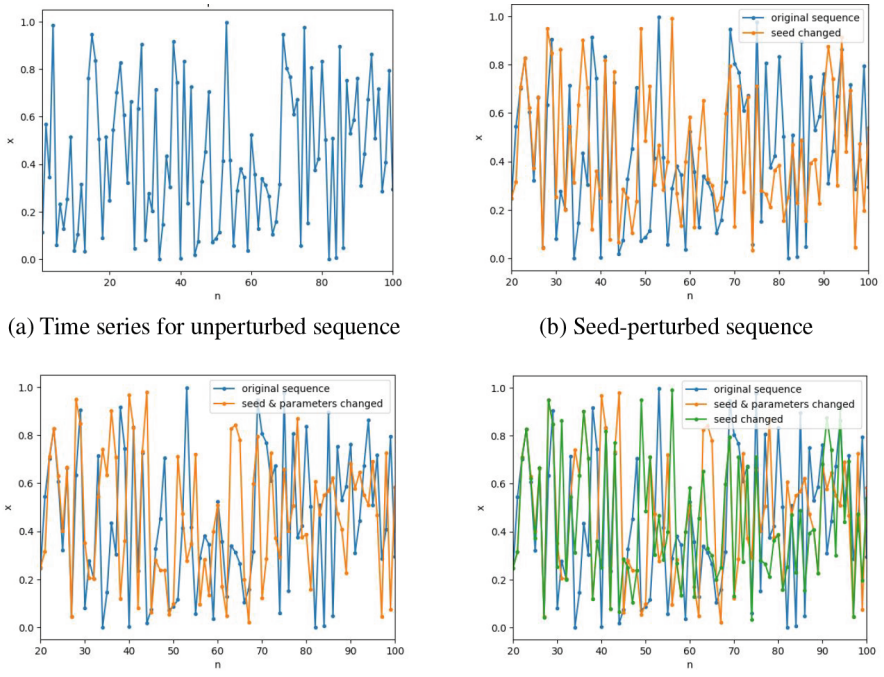
Fig. 5. Autocorrelation plot.

4.3 Time Series Analysis

The time series plots are used to determine the variations in the output sequence under small perturbations in the seed values or the parameter values. The plots for the proposed algorithm indicate a significant variation in the output sequence when seed values or parameter values (or both) are subject to small perturbations (Fig. 6). We also examine the variation between the originally generated sequence and the sequence generated under perturbed values using Granger causality test. We perform the Granger causality test for sequences of fixed length (10000 each) with up to 10 lags under a sufficiently small perturbation parameter ($\epsilon \in [0.0001, 0.001]$). The results for the Grager causality indicated for p-values exceeding 0.05 is shown in the Fig. 7. The test confirms that the perturbed sequences significantly differs from the original sequence and cannot be used to predict the original sequence.

4.4 Bits to Pixel Analysis

The generated sequence (of numbers) was converted in the binary form and was examined for any possible patterns present. The transformed image shows no apparent trends in the pixels, indicating a certain degree of randomness in the output sequence (Fig. 8).



(a) Time series for unperturbed sequence

(b) Seed-perturbed sequence

(c) Seed- and parameter-perturbed sequence

(d) Combined plot for different perturbations

Fig. 6. Output sequences under various perturbations.

4.5 I.I.D Sequences

The proposed algorithm was used in conjunction with the Inversion Principle method to generate sequences using various distributions. The generated sequences were tested for independence and their distributions were examined for possible correlation using Ljung-Box test (conducted on 1000 sequences of 1000 length each). The results indicate that only 0.8% and 6.2% of the P-values are less than 0.01 for the normal distribution and uniform distribution respectively, thus providing a strong evidence that the sequences generated are indeed independent and identically distributed (are IID sequences). The distribution of the P-values obtained are plotted in Fig. 9.

4.6 Correlation and Scatter Plots

The sequences generated from each of the generators was examined for possible correlation with the output (pseudorandom) sequence generated using the heat map and scatter plots. If the generated sequence is of sufficiently large length, the plots are observed to be densely populated exhibiting no apparent patterns. The plots establish low levels of correlation between the sequences generated by each of the generating maps and the proposed algorithm (Fig. 10).

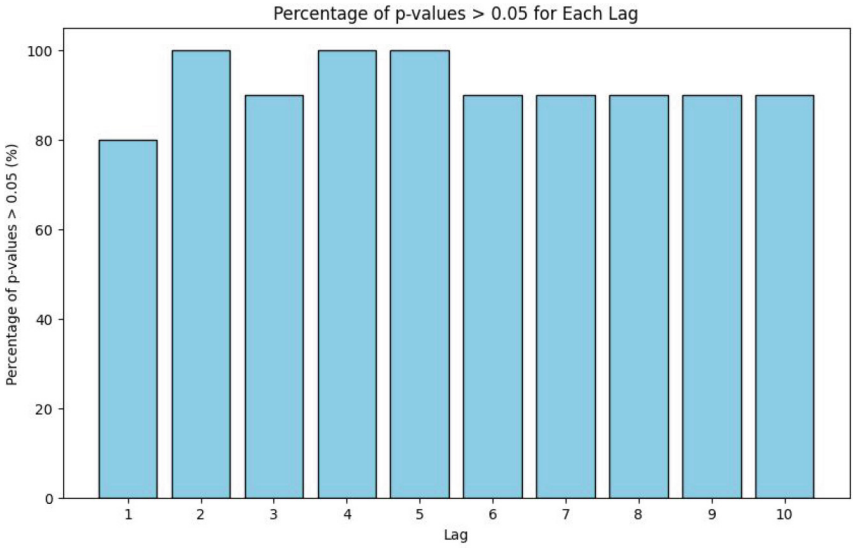


Fig. 7. Success plot for Granger causality test.

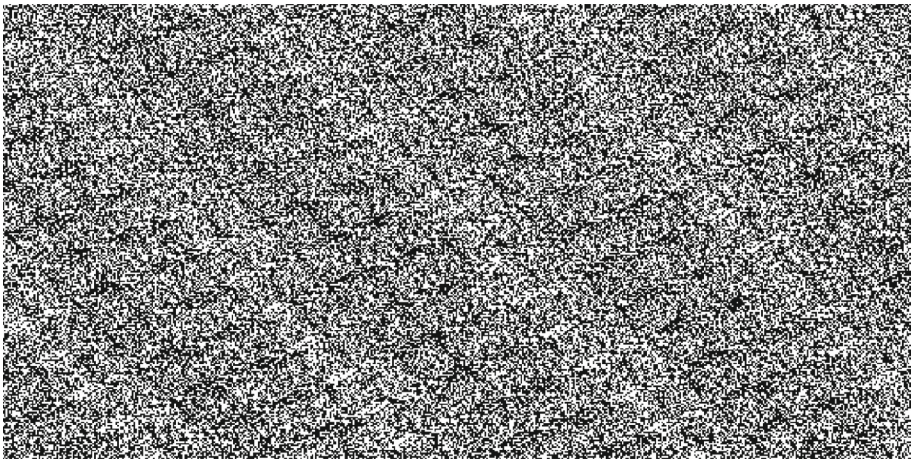
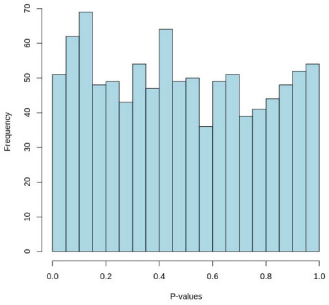
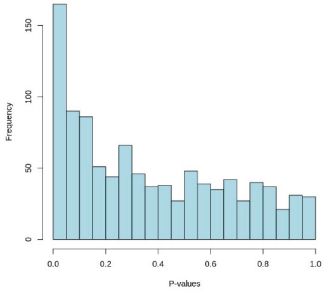


Fig. 8. Pixel analysis.

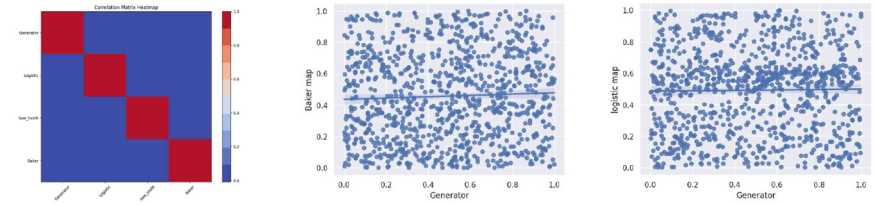


(a) P-values for normally distributed data



(b) P-values for uniformly distributed data

Fig. 9. Distribution of P-values from the Ljung-Box test.



(a) Heat map of correlation matrix (b) Generator vs Baker map (c) Generator vs Logistic map

(d) Generator vs Sawtooth map (e) Logistic vs Baker (f) Sawtooth vs Baker

Fig. 10. Heat map and scatter plots.

4.7 NIST Test Suite Report

The NIST 800-22 test suite is used to examine the randomness in the output sequences. The tests determine existence of any possible patterns in the output sequence. The results establish high level of randomness in the output sequences produced(See Table 1).

Table 1. Results From The NIST Test Module

Statistical Test	Proportion	P-value	Result
Frequency	10/10	0.350485	Success
Block Frequency	10/10	0.534146	Success
Cumulative Sums	10/10	0.991468	Success
Runs	10/10	0.534146	Success
Longest Run	9/10	0.350485	Success
Rank	10/10	0.350485	Success
FFT	9/10	0.350485	Success
Nonoverlapping Template	10/10	0.911413	Success
Overlapping Template	9/10	0.534146	Success
Universal	10/10	0.447934	Success
Approximate Entropy	8/10	0.350485	Success
Random Excursions	2/2	—	Success
Random Excursions Variant	2/2	—	Success
Serial	10/10	0.534146	Success
Linear Complexity	10/10	0.911413	Success

5 Conclusions and Observations

The above work proposes an algorithm to generate pseudo random sequences with sufficiently high degree of randomness using a fixed set of chaotic maps. The algorithm uses the linear congruence map for map selection and hence provides a simple and robust mechanism with lower time complexity to generate arbitrarily long pseudo random sequences. The generated sequences passes all the tests in NIST 800-22 test suite. Consequently, chaotic maps can be used (in conjunction with sufficiently long periodic sequences) to generate sequences with sufficiently high degree of randomness. The generated sequences are observed to be independent of the sequences generated by any of the component maps. The sequences generated by any subset of proposed chaotic maps fail to qualify the standards of the NIST 800-22 test suite. The perturbation analysis and parameter dependence for the proposed algorithm was performed. It is observed that if the perturbations in the parameter are too small (dissipative), the algorithm fails to achieve sufficient degree of randomness in the generated sequences. It is interesting to investigate the nature of the sequences generated when the generating maps have positive entropy or possesses Li-Yorke sensitivity. It is also interesting to investigate randomness in the sequences generated using a smaller collection of generating maps (possibly with decision map of higher complexity). The pointwise summary of observations and results is included below to provide better understanding of the working and functionality of the algorithm.

1. The sequences generated by the decision map used (linear congruence modulo 3) or any of the generating maps fail to pass the NIST 800-22 test suite. However, the chaotic maps can be used in conjunction with linear congruence map (deterministic periodic sequences of sufficiently large length) to generate pseudo-random sequences with high degree of randomness (passes the NIST 800-22 test suite).
2. The pseudo-random sequence generated is independent of the sequence generated through any of the component generators and cannot be predicted using sequences generated by individual generator maps.
3. The sequence generated by any two maps taken from above collection fails to pass the NIST 800-22 test suite (with decision map being linear congruence modulus 2). The results thus indicate minimal level of complexity required for the proposed algorithm to produce sequences with sufficient degree pf randomness.
4. The results indicate that the degree of randomness in the output sequence depends on the perturbation in the parameter values. If the perturbations are dissipative ($\sum_{i=1}^{\infty} p_i < \infty$), the proposed algorithm fails to produce sequences with sufficient degree of randomness. However, if the perturbations are non-dissipative ($\sum_{i=1}^{\infty} p_i = \infty$), the output sequence generated by the proposed scheme passes all the tests in the NIST 800-22 test suite and hence indicate sufficient degree of randomness in the output sequence. The observations relating the perturbations and randomness of the generated sequence are given below:
   (a) if $p_i = \frac{1}{i}$ **passes all the tests**
   (b) if $p_i = \frac{1}{i^2}$ **fails the frequency monobit test which results failure in other tests**

(c) if $p_i = \frac{1}{\sqrt{i}}$ **passes all the tests**
(d) if $p_i = \frac{1}{i^3}$ **fails the frequency monobit test**
(e) if $p_i = \frac{1}{i^{1/3}}$ **passes all the tests**

6 PRNG R Package

As a conclusion to the results obtained in the article, we have developed `PRNG` R package for the proposed algorithm. The `PRNG` package [14] provides functions for generating pseudo-random numbers and random bits. The package can be found at https://doi.org/10.32614/CRAN.package.PRNG

Acknowledgments. The first author is thankful to Ministry of Education, India, for the financial support.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Elaydi, S.N.: Discrete Chaos-With Applications in Science and Engineering. CRC Press, Hoboken, NJ, USA (2007)
2. Panagiotou, P., Sklavos, N., Darra, E., Zaharakis, I.D.: Cryptographic system for data applications, in the context of Internet of Things. Microprocess. Microsyst. **72**, 102921 (2020). https://doi.org/10.1016/j.micpro.2019.102921
3. L'Ecuyer, P.: Efficient and portable combined random number generators. Commun. ACM **31**(6), 742–749 (1988). https://doi.org/10.1145/62959.62969
4. Magfirawaty, Suryadi, M.T., Ramli, K.: On the design of Henon and logistic map-based random number generator. In: J. Phys. Conf. Ser. **893**(1), 012060 (2017). https://doi.org/10.1088/1742-6596/893/1/012060
5. Bassham, L., Rukhin, A., Soto, J., Nechvatal, J., Smid, M., et al.: A statistical test suite for random and pseudorandom number generators for cryptographic applications. NIST Special Publication **800-22r1a** (2010). https://doi.org/10.6028/NIST.SP.800-22r1a
6. L'Ecuyer, P., Simard, R., Chen, E.J., Kelton, W.D.: An object-oriented random-number package with many long streams and substreams. Oper. Res. **50**(6), 1073–1075 (2002). https://www.jstor.org/stable/3088626
7. L'Ecuyer, P., Simard, R.: TestU01: a C library for empirical testing of random number generators. ACM Trans. Math. Softw. **33**(4), 1–40 (2007). https://doi.org/10.1145/1268776.1268777
8. Park, S.K., Miller, K.W.: Random number generators: good ones are hard to find. Commun. ACM **31**(10), 1192–1201 (1988). https://doi.org/10.1145/63039.63042
9. L'Ecuyer, P.: Good parameters and implementations for combined multiple recursive random number generators. Oper. Res. **47**(1), 159–164 (1999). https://doi.org/10.1287/opre.47.1.159
10. Stoyanov, B., Ivanova, T.: CHAOSA: Chaotic map-based random number generator on Arduino platform. In: AIP Conf. Proc. **2172**(1), 090001 (2019). https://doi.org/10.1063/1.5133578

11. Nejati, H., Beirami, A., Ali, W.H.: Discrete-time chaotic-map truly random number generators: design, implementation, and variability analysis of the zigzag map. arXiv (2012). https://doi.org/10.48550/arXiv.1206.1039
12. Moysis, L., Tutueva, A., Volos, C., Butusov, D.: A chaos-based pseudo-random bit generator using multiple digits comparison. J. Comput. Eng. Inf. Technol. **2**, 58–68 (2020)
13. Naik, R.B., Singh, U.: A review on applications of chaotic maps in pseudo-random number generators and encryption. Ann. Data Sci. 1–26 (2021). https://doi.org/10.1007/s40745-021-00364-7
14. Mir, S.A., Sharma, P.: PRNG: Pseudo-random number generator. R Package Version 0.0.2.1.2 (2024). https://doi.org/10.32614/CRAN.package.PRNG