ADBA Chaos and Fractals
Cutting-Edge Scientific Solutions

e-ISSN: 3023-8951
RESEARCH ARTICLE
Vol.3 / No.1 / 2026 / pp.21-28
https://doi.org/10.69882/adba.chf.2026014

A Comparative Performance Analysis of Linear Congruential and Combined Linear Congruential Generators

Ahmet Esad Eldoğan *,1 **and Abdullah Sevin** α,2
*Institute of Science, Computer Engineering Program, Sakarya University, Sakarya, Türkiye, αFaculty of Computer and Information Sciences, Computer Engineering, Sakarya University, Sakarya, Türkiye.

ABSTRACT The generation of high-quality randomness is a fundamental prerequisite for the integrity of stochastic modeling and numerical simulations across various scientific disciplines. As the complexity of computational experiments grows, the robustness of the underlying Pseudo-Random Number Generators (PRNGs) becomes a decisive factor in preventing systematic biases. This study examines the performance and reliability of PRNGs, focusing on the widely utilized Linear Congruential Generator (LCG) and the Combined Linear Congruential Generator (CLCG), which was developed to address the inherent limitations of the former. Using a simulation environment implemented in Python, both algorithms were evaluated based on visual distribution analysis (histograms and 2D/3D scatter plots), statistical fitness (Chi-Square and autocorrelation tests), and computational efficiency (execution speed). The empirical findings demonstrate that while the LCG operates approximately 2.1 times faster than the CLCG, it exhibits a distinct "lattice structure" in multi-dimensional space, leading to significant structural correlation errors. In contrast, the CLCG method, despite its higher computational overhead, achieved a high p-value of 0.92 in the Chi-Square goodness-of-fit test, indicating near-perfect alignment with the theoretical uniform distribution and minimized sequential dependency. Consequently, this study concludes that CLCG is a more reliable choice for high-precision simulations, whereas LCG remains viable for simple applications where computational speed is the primary constraint.

KEYWORDS
Pseudo random number generators
LCG
CLCG
Simulation
Performance analysis

INTRODUCTION

In the field of computer engineering, modeling and simulation studies play a pivotal role in analyzing complex real-world systems. The integrity of these simulations heavily relies on the quality of the randomness utilized; if the random number generator lacks sufficient statistical robustness, the simulation outcomes may be erroneous or misleading. Therefore, selecting an appropriate Pseudo-Random Number Generator (PRNG) is not merely a coding preference but a critical necessity for ensuring the validity of the simulation (Johnston 2018; L'Ecuyer 2011; Banks 2005).

The historical evolution of simulation modeling is rooted in the Monte Carlo methods developed by Von Neumann and Ulam during World War II, which rely heavily on random sampling to solve probabilistic problems. Briefly defined, the Monte Carlo method is a numerical simulation technique used to obtain approximate solutions for problems involving uncertainty through repeated random sampling. However, since the hardware-based generation of truly random numbers is complex and costly, PRNG are generally preferred in practical applications. In simulation modeling, the ability to reproduce the same sequence of numbers is of critical importance for model validation and the comparative analysis of different system designs. Particularly in Monte Carlo simulations, where a vast number of random samples must be generated reliably, the quality of the PRNG directly influences the accuracy of the results (Kroese and Rubinstein 2012; Zio 2012). Law *et al.* (2007) emphasize that random number generators failing to satisfy the properties of statistical independence and uniform distribution can lead to systematic errors in simulation outcomes. Therefore, the selection of a PRNG must be addressed as a fundamental design decision in any simulation study.

The Linear Congruential Generator (LCG) is one of the oldest and most widely utilized methods in pseudo-random number generation. Due to its simple architecture, low computational cost, and ease of implementation, it has been preferred as the default generator in numerous simulation software packages for decades. The significance of the LCG in the literature was further solidified by the seminal work of (Park and Miller 1988). In their study titled "Random Number Generators: Good Ones Are Hard to Find," the authors demonstrated that a carefully parameterized LCG could serve as a "minimum standard," replacing the complex but flawed generators commonly used at the time. The literature indicates that when appropriate parameters are selected, LCG-based generators can exhibit acceptable statistical properties for a wide range of

Manuscript received: 11 September 2025,
Revised: 28 December 2025,
Accepted: 20 January 2026.

[1]esadeldogan@gmail.com
[2]asevin@sakarya.edu.tr (**Corresponding author**)

simulation applications.

While the LCG was historically accepted as a standard, the increasing computational power of modern computers has exposed significant structural deficiencies. The primary constraint of this method is that its period length is strictly limited by the modulus (m) value; in today's high-speed simulations, this period can be exhausted within seconds, causing the sequence to repeat itself prematurely. To address these limitations, L'ecuyer (1988), in his study titled "Efficient and Portable Combined Linear Congruential Generators," proposed the Combined Linear Congruential Generator (CLCG). L'Ecuyer developed a method to mathematically combine the outputs of multiple LCG components, each having distinct moduli. This approach not only expands the generator's period to massive dimensions but also mitigates the planar distribution issues inherent in individual LCGs, thereby ensuring superior statistical randomness. Furthermore, Gentle (2003), in his work "Random Number Generation and Monte Carlo Methods," noted that numbers generated by an LCG tend to form a "lattice structure" in multi-dimensional space. This phenomenon causes data points to align on specific hyperplanes, leading to severe correlation errors in high-precision simulations.

In addition to the LCG and CLCG methods, the literature offers various pseudo-random number generation techniques developed to meet the requirements of diverse application areas. Currently, the Mersenne Twister algorithm, developed by Matsumoto and Nishimura (1998), is widely utilized in numerous scientific analysis and simulation environments due to its exceptionally long period and robust statistical properties. Furthermore, more recent studies highlight the Permuted Congruential Generator (PCG) family, which combines the computational efficiency of classic linear methods with modern permutation techniques to offer both high speed and low memory consumption (O'neill 2014). Similarly, Xorshift-based methods are preferred in performance-oriented applications and parallel simulations. However, since every method possesses distinct advantages and limitations, the selection of a pseudo-random number generator must be tailored to the specific requirements of the application domain (Marsaglia 2003).

The performance of pseudo-random number generators is evaluated not only through their theoretical mathematical properties but also via empirical tests that measure the statistical randomness of the generated sequences. The most prevalent approaches in the literature include uniformity tests (such as Chi-Square and Kolmogorov–Smirnov) to examine the alignment with theoretical distributions, and serial/correlation tests to measure the independence between successive values—particularly critical for linear-based generators. Beyond these fundamental tests, various comprehensive test batteries have been developed to assess PRNG performance more rigorously. One of the most significant standards in this field is the NIST SP 800-22 test suite, published by the National Institute of Standards and Technology. As detailed by Rukhin *et al.* (2001), this suite encompasses numerous statistical analyses, including frequency, block frequency, and runs tests, to evaluate the uniformity and independence of numerical sequences. Furthermore, while the Diehard test battery developed by Marsaglia (1996) serves as a historically significant reference, the TestU01 library, introduced by L'ecuyer and Simard (2007), offers a more contemporary and demanding evaluation environment. TestU01 is specifically designed to uncover structural flaws of PRNGs in multi-dimensional space. Collectively, these frameworks provide a standardized and objective basis for the comparative analysis of different generators.

The primary objective of this study is to conduct a comparative analysis of the LCG, one of the most established methods in the literature, and the CLCG, which was developed to mitigate the inherent deficiencies of the former. Within the scope of this research, the operational principles of both algorithms are examined, and their performances are evaluated within a simulation environment implemented in the Python programming language. To this end, the LCG and CLCG methods are assessed based on the following fundamental criteria:

- Visual Analysis: Evaluation through scatter plots and histograms
- Descriptive Statistics: Comparison of mean and variance values
- Goodness-of-Fit: Statistical validation via the Chi-Square Test
- Independence: Analysis of sequential correlation using Autocorrelation tests
- Computational Performance: Comparison of execution time and speed

The novelty of this study lies in its systematic approach to bridging the gap between theoretical PRNG weaknesses and practical simulation reliability. Unlike existing literature that often focuses solely on mathematical proofs, this work provides a controlled benchmarking framework using large-scale datasets (5 million samples) to quantify the 'speed-versus-quality' trade-off. Furthermore, the study offers a unique pedagogical contribution through high-fidelity 3D lattice structure visualizations, providing empirical evidence of how subtle algorithmic flaws can compromise multi-dimensional stochastic models. By synthesizing visual, statistical, and computational performance metrics, this study serves as a comprehensive decision-support guide for researchers in selecting appropriate PRNGs for high-precision simulations.

PSEUDO-RANDOM NUMBER GENERATORS (PRNGS)

By their very nature, computers are deterministic systems that consistently produce the same output for a given input. This characteristic stands in direct contradiction to the concept of randomness, which necessitates unpredictability. Consequently, it is impossible for computers to generate truly random numbers in a strictly physical sense. However, since randomness is essential for applications such as modeling, simulation, and statistical analysis, PRNGs have been developed to fulfill this requirement. PRNGs utilize specific mathematical algorithms to produce sequences of numbers that exhibit the appearance of randomness. These algorithms require an initial starting value, known as a seed, to commence the generation process. Each generated number serves as the input for the subsequent step, thereby forming a numerical sequence. One of the fundamental characteristics of PRNGs is that the same seed value will invariably yield the identical sequence of numbers. This property provides a significant advantage in simulation studies, as it ensures the reproducibility of experiments (Bhattacharjee and Das 2022; AbdELHaleem *et al.* 2024; Knuth 2014).

Pseudo-random number generators are widely employed in various fields, including modeling and simulation, statistical sampling, computer gaming, optimization problems, and network simulations. The objective in such applications is to sufficiently represent real-world uncertainties within a computational environment. However, if the utilized PRNG lacks adequate quality, the generated sequences may exhibit certain patterns or dependencies, which can lead to erroneous or misleading simulation results. The evolution of PRNGs gained momentum alongside the proliferation of computers (Johnston 2018; L'Ecuyer 2011). Although the first algorithmic approaches emerged in the 1940s, many of

these early methods possessed short periods and weak statistical properties. The development of linear congruential methods in the 1950s provided simpler and faster solutions; in subsequent years, generators with longer periods and superior statistical robustness were introduced. The LCG, proposed by Hamming (1952), was accepted as the industry standard for many years due to its simple architecture and speed. Today, as simulations become increasingly complex, modern algorithms with even longer periods and higher statistical independence have been developed to meet contemporary demands (Banks 1998). This study examines the LCG, a fundamental approach to pseudo-random number generation, and its improved version, CLCG.

Linear Congruential Generator (LCG)

The LCG is recognized as one of the oldest and most fundamental methods in the field of pseudo-random number generation. First proposed by Hamming (1952), this method has been extensively utilized in both academic research and practical applications for decades, owing to its simple architecture and low computational overhead. Although more sophisticated PRNGs have been developed since its inception, the LCG remains significant as an instructional model and a baseline reference for comparative performance analyses. The core operational logic of the LCG involves generating a new number in each step by applying a simple mathematical recurrence relation to the preceding value, starting from an initial seed. However, it is important to note that due to its predictable nature, its use is strictly discouraged in security-sensitive or cryptographic applications.

The mathematical model of the LCG is built upon a linear equation and modular arithmetic. The fundamental recurrence relation is defined as follows (1):

$$X_{n+1} = (a \cdot X_n + c) \ (mod\ m) \tag{1}$$

In this equation, X_n represents the current value, while X_{n+1} denotes the value to be generated in the next step. The parameters are defined as the multiplier (a), the increment (c), and the modulus (m). The algorithm begins with a designated starting value, X_0 (the seed); at each iteration, the value is multiplied by (a), the increment (c) is added, and the remainder of the division by (m) is taken to yield the new value. The resulting raw integers are typically normalized to the [0, 1) interval using the operation $U_n = X_n/m$ (Hamming 1952).

Combined Linear Congruential Generators (CLCG)

CLCG are pseudo-random number generation architectures developed to mitigate the inherent limitations of a single LCG. While a standalone LCG is computationally efficient and straightforward to implement, it suffers from several weaknesses, including relatively short periods, dependencies between successive values, and the emergence of regular patterns known as lattice structures in multi-dimensional scatter plots. The CLCG approach is based on the strategic combination of multiple LCGs to overcome these deficiencies. The conceptual framework of CLCG was systematically formalized through the research conducted by Pierre L'Ecuyer starting in the late 1980s. L'ecuyer (1988) demonstrated that by appropriately combining two or more LCGs defined with distinct parameters, the resulting generator can achieve a period significantly longer than that of a single component and exhibit substantially improved statistical properties. These advancements have established the CLCG as a reliable alternative, particularly for high-fidelity simulations and Monte Carlo analyses. The operational principle of a CLCG involves running multiple sub-LCGs each with its own multiplier, increment, and modulus in parallel. In each iteration, the values produced by these sub-generators are merged using a simple mathematical operation, typically addition or subtraction, to produce a single output value. A common CLCG structure involving two LCGs can be expressed as follows 2:

$$Z_n = \left(X_n^{(1)} - X_n^{(2)}\right) (mod\ m_1 - 1) \tag{2}$$

In this formulation, $X_n^{(1)}$ and $X_n^{(2)}$ represent the n^{th} values generated by two independent LCGs. The resulting Z_n value is subsequently normalized to the [0, 1) interval. This synthesis effectively reduces the periodicity and dependency issues observed in a single LCG, yielding numerical sequences with vastly extended periods, enhanced distribution balance, and minimized correlation (L'Ecuyer 2011).

METHODOLOGY

In this study, the LCG, a foundational approach in pseudo-random number generation, and its enhanced variant, the CLCG, were tested on identical hardware under uniform initial conditions (seeds) and shared evaluation criteria. The pseudo-random sequences generated within the simulation were analyzed at different scales to examine visual patterns, evaluate statistical accuracy, and measure time/speed performance.

For visual analysis, datasets of 2,000 and 3,000 samples were utilized to generate scatter plots and detect the potential lattice structure in multi-dimensional space. Statistical evaluations, including histograms, mean-variance analysis, Chi-Square goodness-of-fit tests, and autocorrelation analyses, were conducted using sequences of 100,000 samples. Finally, to reveal the differences in computational overhead between the algorithms, execution time and speed performance comparisons were performed on large-scale datasets consisting of 5,000,000 samples.

While comprehensive batteries such as TestU01 or NIST SP 800-22 are frequently used for validating cryptographic-grade generators, the current study utilizes a targeted set of statistical and visual benchmarks. This selection is justified by the study's focus on the structural transition from LCG to CLCG. Given that the LCG (specifically the RANDU parameters) is historically known to fail complex randomness batteries, our methodology prioritizes visual lattice detection and Lag-1 autocorrelation to provide a transparent analysis of the deterministic dependencies inherent in these algorithms. This focused approach allows for a clearer interpretation of the speed-versus-quality trade-off without the redundant complexity of cryptographic test suites.

Experimental Environment

All simulations in this study were developed using the Python programming language, which provides an efficient environment for both statistical and visual analysis of PRNGs. All experiments were conducted on the same hardware configuration to ensure that the performance comparisons were consistent and reproducible. The NumPy library was employed for generating numerical sequences and performing fundamental mathematical operations. For statistical analyses, including mean-variance calculations, Chi-Square goodness-of-fit tests, and autocorrelation, the SciPy library was utilized. Visualizations of the results, such as histograms and 2D/3D scatter plots, were created using the Matplotlib library. Specifically, 3D visualizations were used to clearly observe the lattice structure inherent in linear congruential methods.

The computational performance and execution speed of the algorithms were measured using Python's standard timing mod-

ules. In these analyses, the duration required for each algorithm to produce a specific volume of pseudo-random numbers was calculated using large-scale datasets. This allowed for a comprehensive evaluation of the LCG and CLCG algorithms, not only in terms of statistical quality but also regarding their computational efficiency.

In the first phase, the LCG was implemented using formula. To clearly demonstrate the structural weaknesses and the "lattice structure" effect of the LCG, the infamous RANDU parameters were intentionally selected. RANDU is a historical reference point known for its poor statistical performance in multi-dimensional spaces. The parameters are defined as follows:

- Modulus (m): 2^{31}
- Multiplier (a): 65539 ($2^{16} + 3$)
- Increment (c): 0
- Initial Seed (X_0): 123456789

The second phase involved the Combined Linear Congruential Generator (CLCG), based on the dual-component architecture proposed by (L'ecuyer 1988). This method combines two independent LCGs to achieve a significantly longer period and superior randomness. The selected parameters (Table 1) are based on prime numbers near $2^{31} - 1$, with multipliers identified as primitive roots to ensure full periods:

Table 1 CLCG parameters

Parameter	LCG 1 (X_n)	LCG 2 (Y_n)
Multiplier (a)	40014	40692
Modulus (m)	2147483563	2147483399
Increment (c)	0	0

The outputs are combined using the following rule 3:

$$Z_n = \left(X_n^{(1)} - Y_n^{(2)}\right) (mod\ m_1 - 1) \tag{3}$$

The resulting Z_n is normalized to the [0, 1) interval. This configuration provides a theoretical period of approximately 2.3×10^{18}, offering a massive improvement in periodicity and correlation over standalone LCGs.

Evaluation Criteria and Statistical Testing

The performance of the pseudo-random number generators was evaluated through a multidimensional approach encompassing distributional uniformity, statistical independence, structural correlation, and computational cost. The following criteria and testing methods were employed:

- Histogram (Visual Distribution Analysis): Used to visualize the frequency distribution of generated numbers within the [0, 1) interval. An ideal PRNG should produce a histogram with bars of approximately equal height, indicating no significant clustering or gaps in any specific sub-interval.
- Descriptive Statistics (Mean and Variance): The numerical sequences were checked against theoretical expectations. For a continuous uniform distribution on [0, 1), the theoretical mean is 0.5, and the variance is $1/12 \approx 0.0833$. Simulation outputs were assessed based on their proximity to these reference values.
- Scatter Plots: Utilized to visualize the relationship between consecutive pairs (X_n,X_{n+1}) or triplets of numbers. This is the most effective method for detecting linear dependencies and the lattice structure inherent in congruential methods. Both 2D and 3D scatter plots were utilized to analyze the structural independence of the generators.
- Chi-Square Goodness-of-Fit Test: A formal statistical test to determine whether the generated numbers follow a uniform distribution. A p-value less than 0.05 leads to the rejection of the uniformity hypothesis, while a value closer to 1.0 indicates higher reliability and superior fit.
- Autocorrelation Test: Measures the dependency (correlation) between successive values in the sequence. In an ideal generator, consecutive numbers should be independent. In this study, the Lag-1 autocorrelation coefficient was calculated to examine how closely the correlation approaches zero, as Lag-1 is the primary indicator of sequential dependency.
- Time/Speed Performance Analysis: Computational cost is a critical factor in large-scale simulation studies. In this analysis, the CPU time required for each algorithm to generate a large-scale dataset (5,000,000 samples) was measured and compared to determine their efficiency.

FINDINGS

In this section, the findings obtained from the conducted experiments are reported in terms of descriptive statistical measures, distribution analyses, scatter plots, statistical goodness-of-fit tests, autocorrelation results, and computational performance.

Statistical Results (mean and variance)

The fundamental statistical properties of the pseudo-random sequences generated by the LCG and CLCG algorithms were examined. The objective was to evaluate the extent to which both generators align with the theoretically expected characteristics of a uniform distribution. To this end, 100,000 pseudo-random numbers were generated for each algorithm, and the mean and variance values were calculated based on these sequences (Table 2). Theoretically, for a random variable following a uniform distribution on the interval [0, 1), the expected mean value is 0.5, and the variance is $1/12 \approx 0.0833$.

Table 2 Mean-variance results

Metric	Theoretical Value	LCG	CLCG
Mean	0.50000	0.50000	0.49971
Variance	0.0833	0.08313	0.08310

Histogram Analysis Results

To visually inspect the uniformity of the generators, histograms generated from 100,000 data points are presented below. To ensure a robust comparison, both plots were rendered using identical axis scales, and a reference line (dashed black line) representing the ideal theoretical uniform distribution has been superimposed onto the graphs (Figure 1).

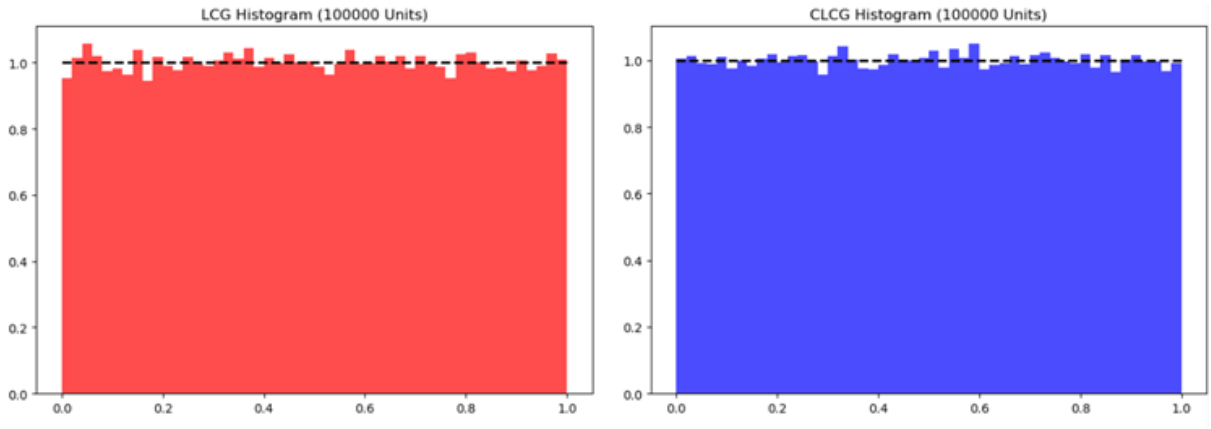

Figure 1 Mean-variance histograms

Scatter Plot Analysis Results (2D)

The independence properties of the pseudo-random numbers sequentially generated by the LCG and CLCG algorithms were investigated through two-dimensional (2D) scatter plots (Figure 2). These plots were constructed by mapping successive pairs of values (X_n, X_{n+1}) onto a coordinate plane, with the objective of visually detecting potential linear patterns or correlation structures. For this analysis, 2,000 pseudo-random numbers were generated for each generator, and 2D scatter plots were derived from these consecutive value pairs.

Scatter Plot Analysis and Lattice Structure (3D)

In this subsection, the distribution characteristics of the pseudo-random sequences generated by the LCG and CLCG algorithms in multi-dimensional space are examined. Three-dimensional (3D) scatter plots (Figure 3) were constructed by mapping triplets of successive values (X_n, X_{n+1}, X_{n+2}) onto a coordinate space. The primary objective was to detect potential correlation structures or regular patterns that may not be visible in lower dimensions. For this analysis, 3,000 pseudo-random numbers were generated for each algorithm. To ensure a consistent and objective comparison, the 3D scatter plots for both LCG and CLCG were rendered using identical viewing angles, axis scales, and visualization parameters.

Chi-Square Goodness-of-Fit Test Results

In this subsection, the statistical alignment of the pseudo-random sequences generated by the LCG and CLCG algorithms with the theoretically expected uniform distribution was evaluated. To this end, the Chi-Square goodness-of-fit test was applied to the datasets obtained from both generators. The tests were conducted using sequences of 100,000 pseudo-random numbers, and the results are reported in terms of p-values. The calculated p-values serve as a statistical indicator of whether the null hypothesis (H_0), which assumes a uniform distribution, can be accepted according to Table 3.

The p-values calculated for the LCG and CLCG algorithms, along with their acceptance/rejection status based on a 5% significance level (α= 0.05), are presented in the table below:

Table 3 Chi-Square p-values

Algorithm	Calculated p-value	Significance level (α)	Result (H_0 hypothesis)
LCG	0.1324	0.05	Accepted (Uniform)
CLCG	0.9209	0.05	Accepted (Highly Uniform)

Autocorrelation Analysis Results

The level of dependency between consecutive values in the pseudo-random sequences generated by the LCG and CLCG algorithms was examined quantitatively. Autocorrelation analysis was utilized to evaluate whether the generated numbers are independent of one another.

For this analysis, Lag-1 autocorrelation coefficients were calculated using sequences of 100,000 pseudo-random numbers for each generator. The term "Lag-1" refers to the correlation between each number (X_n) in the sequence and the value immediately preceding it (X_{n-1}). The resulting coefficients serve as a quantitative indicator of the sequential relationship between successive values in Table 4.

Table 4 Autocorrelation test results

Algorithm	Calculated Coefficient (Lag-1)	Ideal Value
LCG	0.00729	0.0
CLCG	-0.00118	0.0

Speed and Computational Performance Results

In this subsection, the time and speed performance of the LCG and CLCG algorithms were investigated during large-scale pseudo-random number generation. The objective was to provide a comparative analysis of the computational overhead associated with each generator. To achieve this, 5,000,000 pseudo-random numbers were generated using both the LCG and CLCG algorithms, and the total execution time for each process was measured. All measurements were conducted under identical hardware and operating conditions to ensure consistency. The resulting execution times are presented in the Table 5 below to facilitate a direct speed performance comparison:

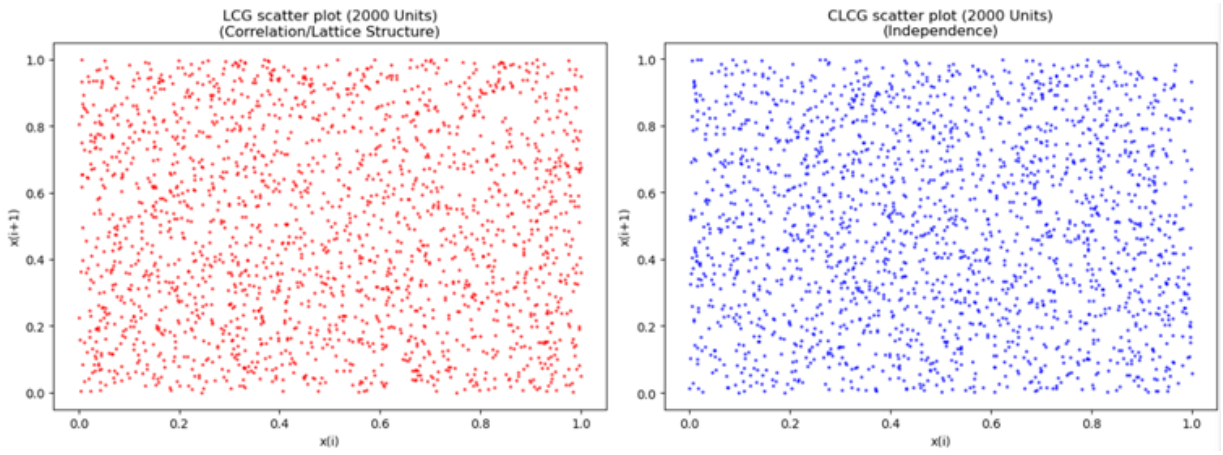

Figure 2 Scatter plots

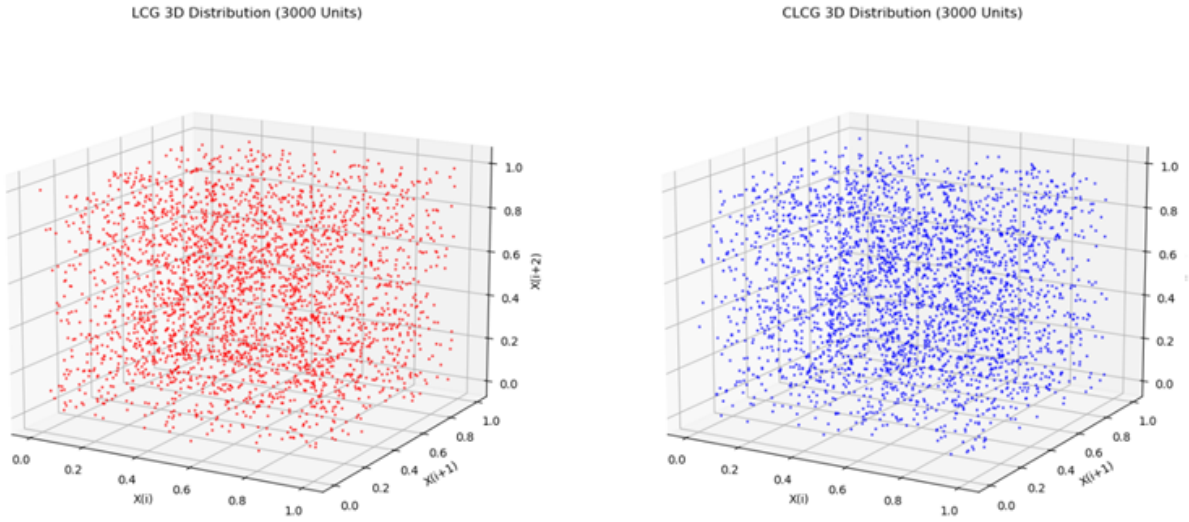

Figure 3 Scatter plots and lattice structure

Table 5 Speed and computational performance results

Algorithm	Quantity Generated	Execution Time (Seconds)	Speed Comparison
LCG	5,000,000	1.7684	Reference (1.0x)
CLCG	5,000,000	3.7544	∼2.12x Slower

The findings presented in this section encompass the visual and numerical results derived from the LCG and CLCG simulations. A comprehensive evaluation and comparative discussion of these findings are provided in the following section.

RESULTS AND DISCUSSION

In this study, the LCG, a widely utilized method in pseudo-random number generation, and the CLCG, an enhanced statistical extension of this method, were comparatively analyzed. The findings have been evaluated to highlight the respective strengths and weaknesses of these generators in terms of randomness quality and practical applicability.

Comparative Analysis of Statistical Quality

Upon reviewing the simulation results, it is observed that both methods meet the fundamental statistical expectations (mean and variance) at a reasonable level. The means of the sequences generated by both the LCG and CLCG are remarkably close to the theoretical value of 0.5, and their variances align closely with 0.0833. This indicates that both algorithms are successful in scaling the generated numbers accurately within the [0, 1) interval. However, a significant difference in quality emerges regarding the "uniformity" of the distributions. When visual histogram analyses and numerical Chi-Square goodness-of-fit results are evaluated together, the CLCG algorithm demonstrates a near-perfect alignment with the theoretical uniform distribution, yielding a high p-value of 0.9209. In contrast, while the LCG algorithm remains within statistically acceptable limits ($p > 0.05$) with a p-value of 0.1324, it exhibits a lower level of fit compared to the CLCG. Consequently, in terms of statistical reliability, it is concluded that the CLCG provides a much more stable and homogeneous distribution than the LCG, making it a superior candidate for high-precision

simulations.

Structural Independence and Correlation Analysis

The most critical factor determining the success of a pseudo-random number generator is the degree of independence between the generated values. The visual analyses conducted in this study revealed a significant structural disparity, particularly in the 3D scatter plots. When triplets of values (X_n, X_{n+1}, X_{n+2}) generated by the LCG algorithm were projected into space, it was observed that the points, rather than forming a random distribution, were aligned on specific parallel hyperplanes. This phenomenon, identified in the literature as the "Lattice Structure," provides empirical proof of the structural correlation inherent in the LCG. In contrast, the 3D plots for the CLCG algorithm demonstrated a much more homogeneous distribution of points, with no detectable planar clustering. This visual superiority is further substantiated by the numerical data. In the autocorrelation analysis, while the LCG coefficient indicated a trend of positive dependency (0.007), the CLCG coefficient (-0.001) was significantly closer to zero in absolute terms. In light of these findings, it is concluded that the CLCG method minimizes sequential dependency and approximates "true randomness" far more effectively than the LCG.

Computational Efficiency vs. Quality Trade-off

Time and speed performance analyses demonstrate that the LCG algorithm can generate pseudo-random numbers in significantly less time compared to the CLCG. Due to its simple architecture based on a single linear recurrence relation, the LCG offers low computational overhead and high generation throughput. This characteristic makes the LCG an attractive option for scenarios where speed is the primary priority, such as very large-scale simulations or real-time applications. Conversely, the CLCG algorithm incurs a higher computational cost due to its structure based on combining multiple LCG components. However, this additional overhead yields substantial gains in statistical quality and independence. The fact that the CLCG effectively eliminates the lattice structure in visual analyses and produces more robust results in statistical tests clearly illustrates that quality is achieved by sacrificing a degree of speed. In this context, the choice between LCG and CLCG depends heavily on the specific application domain. For simple simulations or preliminary computational processes where speed is critical and the quality of randomness is of secondary importance, the LCG may suffice. However, for long-duration simulations, engineering tasks, and scientific applications where statistical accuracy is paramount, the CLCG provides a much more reliable alternative. Ultimately, this balance established between performance and quality underscores the importance of selecting a pseudo-random number generator that is fit for purpose.

CONCLUSION

This study demonstrates that the selection of a PRNG in simulation projects is not merely a coding detail, but a critical design decision that directly impacts the validity and reliability of the model. The findings confirm that while the basic LCG offers speed advantages, its structural flaws most notably the lattice structure exhibited in multi-dimensional space render it insufficient for modern, high-precision simulations. Conversely, the CLCG approach, despite increasing computational overhead, successfully addresses these deficiencies by providing superior statistical accuracy and independence. In future research, a comparative analysis of the CLCG method against modern, high-performance algorithms such as the Mersenne Twister or Xorshift would provide a contemporary contribution to the literature. Furthermore, exploring the performance of these generators within parallel computing environments and GPU-based simulations remains a promising area for further investigation.

Ethical standard

The authors have no relevant financial or non-financial interests to disclose.

Availability of data and material

Not applicable.

Conflicts of interest

The authors declare that there is no conflict of interest regarding the publication of this paper.

LITERATURE CITED

AbdELHaleem, S. H., S. K. Abd-El-Hafiz, and A. G. Radwan, 2024 Analysis and guidelines for different designs of pseudo random number generators. IEEE Access **12**: 115697–115715.

Banks, J., 1998 *Handbook of simulation: principles, methodology, advances, applications, and practice*. John Wiley & Sons.

Banks, J., 2005 *Discrete event system simulation*. Pearson Education India.

Bhattacharjee, K. and S. Das, 2022 A search for good pseudo-random number generators: Survey and empirical studies. Computer Science Review **45**: 100471.

Gentle, J. E., 2003 *Random number generation and Monte Carlo methods*. Springer.

Hamming, R., 1952 Mathematical methods in large-scale computing units. Math Rev **13**: 495.

Johnston, D., 2018 *Random Number Generators—Principles and Practices: A Guide for Engineers and Programmers*. Walter de Gruyter GmbH & Co KG.

Knuth, D. E., 2014 *The art of computer programming: Seminumerical algorithms, volume 2*. Addison-Wesley Professional.

Kroese, D. P. and R. Y. Rubinstein, 2012 Monte carlo methods. Wiley Interdisciplinary Reviews: Computational Statistics **4**: 48–58.

Law, A. M., W. D. Kelton, and W. D. Kelton, 2007 *Simulation modeling and analysis*, volume 3. Mcgraw-hill New York.

L'ecuyer, P., 1988 Efficient and portable combined random number generators. Communications of the ACM **31**: 742–751.

L'ecuyer, P. and R. Simard, 2007 Testu01: Ac library for empirical testing of random number generators. ACM Transactions on Mathematical Software (TOMS) **33**: 1–40.

L'Ecuyer, P., 2011 Random number generation. In *Handbook of computational statistics: concepts and methods*, pp. 35–71, Springer.

Marsaglia, G., 1996 Diehard: a battery of tests of randomness. https://cir.nii.ac.jp/crid/1571698600935841152 .

Marsaglia, G., 2003 Xorshift rngs. Journal of Statistical software **8**: 1–6.

Matsumoto, M. and T. Nishimura, 1998 Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. ACM Transactions on Modeling and Computer Simulation (TOMACS) **8**: 3–30.

O'neill, M. E., 2014 Pcg: A family of simple fast space-efficient statistically good algorithms for random number generation. ACM Transactions on Mathematical Software **204**: 1–46.

Park, S. K. and K. W. Miller, 1988 Random number generators: good ones are hard to find. Communications of the ACM **31**: 1192–1201.

Rukhin, A., J. Soto, J. Nechvatal, M. Smid, and E. Barker, 2001 A statistical test suite for random and pseudorandom number generators for cryptographic applications. Technical report, NIST-SP-800-02.

Zio, E., 2012 Monte carlo simulation: The method. In *The Monte Carlo simulation method for system reliability and risk analysis*, pp. 19–58, Springer.

How to cite this article: Eldoğan, A. E., and Sevin, A. A Comparative Performance Analysis of Linear Congruential and Combined Linear Congruential Generators. *Chaos and Fractals*, 3(1), 21-28, 2026.