Two applications for estimating Pi using MIXMAX random number generator and Monte Carlo method

J. P. Ortega, Alejandra Pérez, Laura Suárez, Sebastián Torroledo, Juan D. Lara*

Departamento de Matemáticas y Ciencias de la Computación, Universidad Nacional de Colombia, Carrera 45 No. 26-85, Edificio Uriel Gutiérrez, Bogotá D.C., Colombia

Abstract

In this report we review some pseudo-random number generators (PRNGs) based on the theory of mixing in classical mechanical systems. In particular, we explore in detail the statistical characteristics of MIXMAX pseudo-random number generator such as its entropy, period and spectrum. We then apply this generator along with Monte Carlo method to estimate the value of Pi in two different ways, using the relation between approximated areas. We compare these results with those obtained using MATLAB's default PRNG. The results of these simulations show the effectiveness, precision, consistency and viability of the random numbers generated by MIXMAX.

Keywords: Random numbers, Monte Carlo method, MIXMAX, Theory of Mixing, RNG, PRNG, MATLAB.

2020 MSC: 65C10, 65C05, 11K45,

1. Introduction

We refer to *random numbers* as those extracted unpredictably from a set of possible values, each of which has the same probability of being chosen. When we talking about a sequence of random numbers, each number drawn must be statistically independent of the others. Random numbers have a wide variety of applications, such as generating data encryption keys [1], simulating and modeling complex phenomena [2], and selecting random samples from larger data sets in order to make simulations to verify statistical results [3]. Random numbers have also been used in literature and music [4] and are widely used in video games and betting [5]. With the advent of computers, programmers recognized the need for mechanisms of introducing randomness into a computer program. However, a computer follows pre-programmed instructions and is therefore completely predictable, and this is where the difficulty of generating random numbers in a computer lies. As a consequence of this fact, two questions appear: How to generate random numbers using computer programs? And once we establish a method for that purpose, how to test the quality of the random numbers it generates?

There are two main groups of random number generators. The first one consists of the so-called true random number generators (RNG), sometimes referred to as the non-deterministic ones; whereas the second kind consists of the pseudo-random number generators (PRNG), also known as the deterministic ones. Some authors establish additional intermediate types of generators like quasi-random number generators, but these can also be categorized as pseudo-random number generators. For further information, see [6].

A true random number generator uses a non-deterministic entropy source along with the entropy distillation process to produce randomness. This distillation process is needed to overcome any weakness produced

*Corresponding author
Email address: jlara@unal.edu.co (Juan D. Lara*)

in the entropy source which can cause the production of non-random numbers [1]. There is a general consensus that true randomness cannot appear in the classical physical world because any classical system admits a deterministic description and thus appears unpredictable only as a consequence of a lack of knowledge about its fundamental description [7]. It follows from this that true randomness may only emerge from quantum systems [8].

As for PRNGs, they produce deterministic sequences of numbers that have many significant statistical characteristics in common with true random numbers [9]. This sequences are not truly random because they are completely determined by an initial value called the *seed* and some algorithm that can even be public, this is that we can know [10]. The source of entropy in PRNGs is a chaotic deterministic system in order to make it hard to find any information of the numbers produced based on partial knowledge of the generator. This last kind of random number generator may be preferable since it has better statistical properties and can produce large quantities of random numbers faster and with less resources than RNG's [1]. In this project, we review a PRNG known as MIXMAX and apply it to estimate the value of Pi.

There are plenty of problems that cannot be solved by using analytical or numerical algorithms because of computing limitations. One of the methods that is useful in some of these cases is the Monte Carlo method, which consists in a wide collection of computational algorithms that make use of randomness to solve, in particular, deterministic problems, i.e., problems with no randomness involved, by using repeated sampling to make numerical estimations of unknown parameters or variables [6]. Because of the convenience of the random-samples, Monte Carlo method could get close to the solution by using just a few or a very large number of samples, thus usually many samples are used so that this method converges to the solution with the accuracy rate that is needed [11]. Monte Carlo methods are also used to obtain numerical approximations of problems with no random parameters and also problems with randomness in their formulation [6].

For example, one classic Monte Carlo application consists in bombing two regions, one inside the other, with random points to estimate the relation between their areas by counting how many points lay inside the inner region. We can find an example of this kind of process in [12]. This is also an interesting way of calculating proper defined integrals by estimating the area under the curve of the function that we want to integrate. In this report, we present two examples of this idea that can be seen as two different ways of estimating the value of Pi numerically. We also compare the characteristics of the generic MATLAB pseudo-random number generator with the ones of MIXMAX by comparing the results obtained in the simulations of this applications involving each of those PRNGs.

Traditionally, pseudo-random numbers were generated with mathematical linear equations. After that, some researches proposed to build secure PRNGs with block ciphering algorithms. In the latter days, PRNGs were constructed using seeds from random sources, which were better. To make PRNGs more efficient and to improve their perform in software, many chaotic systems based PRNGs were proposed to generate random numbers [13]. The PRNGs that we apply belong to this group and are based on the mixing theory; specifically, the $K-$mixed or $K-$systems methods based on Kolmogorov's theory [14]. Our choice is due to the fact that, as indicated in [15], these are among the best PRNGs because of the independence of a value's state with regard to its previous ones. One of the generators which is based on $K-$systems is MIXMAX, a recursively defined method that uses square unimodular matrix multiplication; it is characterized by its speed of random number generation, which is favorable when we compare it with similar ones such as RANLUX [16]. In addition, it is suitable for this work because of its synergy when experimenting with the Monte Carlo method as we can observe in [17].

2. PRNG based on theory of mixing

As PRNG concept was introduced, how can the pseudo-random numbers be characterized in a way they can be distinguished? It is pertinent to evaluate what should we expect of pseudo-random numbers. At a first sight, when we select a number from a set it must be possible to we get any of the numbers in the set, that is, a PRNG should cover all the possible outcomes, i.e., it should fill up what is called the geometrical space. Also, as pseudo-random numbers need to resemble random, a pattern should not be shown between calculations. For this reason, the algorithm used for the PRNG must has some level of complexity and

should not repeat the number sequences too soon. This minimal length is stated as period in what follows [17].

The next question is how to evaluate the quality of a PRNG? One could think that the complexity of the algorithm is what determines how good it is because as much complex they are the more unpredictable it would expected to be, but it is not enough in order to conclude whether it is a good PRNG or not. Many criteria can be used to compare PRNG depending on what is needed, such as:

- Speed: How fast it is to setup it and to deliver a number.
- Complexity: How difficult it is guessing the next number based on the previous ones.
- Randomness: How similar are the numbers generated with random numbers.

There are many tests that can be applied to check if the generator fails as is shown in [18], but as it is mentioned in [17], if a generator pass some of these tests does not imply it is a high-quality one, tests are more adequate to find the generator's weaknesses and how to handle them. Furthermore, using a PRNG that has been studied and whose weaknesses are known is better than using an unknown generator because of the uncertain quality of the outcomes.

A particular property that some PRNG have is that the probability of getting a particular number is independent to the previous ones, which is the best case of "complexity" as we mentioned above. This property is based on mixing theory. In what follows, we define classical dynamical systems, and some PRNGs defined based on a particular group of them. Furthermore, we introduce some of their properties such as entropy and period in order to conclude why can we consider them as good generators.

2.1. Classical dynamical systems

In mathematics, a classical dynamical system is one conformed by a geometrical space and a function that describes the movement of a point depending on time. Some examples include the Lorenz System, the dyadic transformation, the logistic map and the elastic pendulum. These systems can exhibit such an erratic and chaotic behavior that the position of a point at a given moment can seem to be random, in the sense that it has no relation with the final position of other points that started near its starting position. Therefore, the question of which systems are *random enough* arises.

As asserted by James in [15], there are two categories of classical dynamical systems: those that preserve energy, which are called Hamiltonian systems, and those that do not, which are called dissipative systems. Dissipative systems are not good candidates for random number generation because they do not fill up all its geometrical space, thus they do not generate all possible numbers.

Therefore, consider only the Hamiltonian systems, in which the time evolution of the system is characterized by the equations

$$\frac{\partial p}{\partial t} = \frac{\partial \mathcal{H}(q,p,t)}{\partial q}, \qquad \frac{\partial q}{\partial t} = \frac{\partial \mathcal{H}(q,p,t)}{\partial p}, \tag{1}$$

where $\mathcal{H}(q,p,t)$ is known as the Hamiltonian and corresponds to the total energy of the system, q is the vector of generalized coordinates and p is their conjugate momenta [19]. If equations (1) are separable and integrable, the system is exactly soluble and predictable, thus useless for pseudo-random number generation. In this order of ideas, the set of systems suitable for random number generation has been reduced to those that preserve energy for which the Hamiltonian equations are not exactly soluble [15].

Among this kind of classical dynamical systems, the ones that we are focusing on can be represented in the following way: Let $x(i)$ be the $N-$vector representing the state of some points along the continuous trajectory of the abstract dynamical system in $N-$dimensional phase space at time i. Then,

$$x(i+1) = A\ x(i) \qquad \mod 1, \tag{2}$$

where A is a $N \times N$ constant integer matrix such that $\det(A) = 1$. This matrix characterizes the behavior of the system (and can also be thought of as the numerical solution of the equations of motion [17], which are

strictly related to the solutions of the hamiltonian equations (1)). The conditions given to A ensure that it has an integer inverse matrix and therefore the solution of the equations of motion exists. The geometrical space is a unit $N-$hypercube whose topology coincides with a $N+1-$dimensional torus because of the mod operation. Furthermore, the fact that $\det(A) = 1$ implies that the volume of the hypercube is preserved when transformed by the matrix A.

Applying the recursion given by equation (2), we obtain a sequence $\langle x(0), x(1), ..., x(i)\rangle$, which is determined by the value of $x(0)$. As mentioned earlier, some dynamic systems possess an erratic chaotic behavior and any change in the value of $x(0)$ can provoke big changes in the sequence generated. This type of dynamical system, where the time parameter is a discrete variable, is know as *discrete system*.

To illustrate the effect of a matrix with the properties described above over the $N-$hypercube, consider the matrix

$$A = \begin{pmatrix} 13 & 8 \\ 21 & 13 \end{pmatrix}. \tag{3}$$

Note that $\det(A) = 1$ and its eigenvalues are $\lambda_1 = 13 + 2\sqrt{42}$, and $\lambda_2 = 13 - 2\sqrt{42}$, which implies that $|\lambda_k| \neq 1$ for $k \in \{1, 2\}$. In Figure 1 we can see how A transforms the $2-$hypercube.

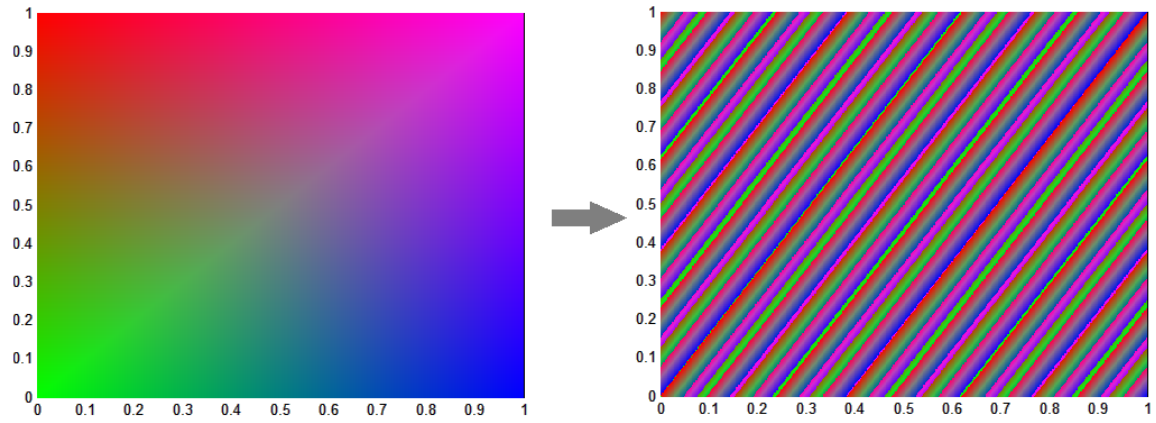

Figure 1: Effect of a matrix A that defines a 2×2 discrete hamiltonian dynamical system over the $2-$hypercube when applying the algorithm $x(i+1) = A\ x(i)$ once.

An interesting interpretation of Figure 1 is that the matrix A *mixes* the points of the hypercube, bringing together points that were far away and vice versa. Moreover, the final state of a point gives little to none information about its original one, which is a very important property to ensure the unpredictability of a PRNG. In the following subsection we formalize this property and study its implications.

2.2. The mixing property

Among the dynamical systems that represented by equation 2, we seek for those that possess independence. That is to say that the location of a point at a given time does not provide information of its locations at previous moments. This property can be formalized as follows.

Let $x(i)$ and $x(j)$ be the state of the dynamical system at two times i and j according to equation (2). Let also v_1 and v_2 be any two subspaces of the geometrical space, with measures (volumes relative to the total allowed volume), respectively $\mu(v_1)$ and $\mu(v_2)$. Then the dynamical system is said to be a $1-$system (with $1-$mixing) if

$$P(x(i) \in v_1) = \mu(v_1), \tag{4}$$

where P is a probability function, and a $2-$system (with $2-$mixing) if

$$P\Big(x(i) \in v_1 \text{ and } x(j) \in v_2\Big) = \mu(v_1)\mu(v_2), \tag{5}$$

for all i and j sufficiently far apart. Generally, a $K-$system has the same property for an arbitrarily large number of points and subvolumes. This is known in the literature as the *mixing property*. In [20], we can see that $K-$systems form a hierarchy: A $n-$system is a $m-$system for all $m < n$.

It has been proved that if a dynamical system is represented as in equation (2), it is a $K-$system if any eigenvalue λ of A is such that $|\lambda| \neq 1$ [17]. Note that this implies that the matrix defined in equation (3) represents a $K-$system.

The mixing property condenses the statistical independence we are looking for: each point is (asymptotically) independent of all the other points in the sense that the probability of finding it in any volume is independent of all the other states it has been in [15]. Therefore, the study of $K-$systems and their properties is (and has been for the last decades [17]) very promising for the development of PRNGs.

2.3. K-systems properties

The $K-$systems have four important properties that have to be taken into account when we are going to develop a PRNG. This properties are the entropy, the divergence of nearby trajectories, the asymptotic independence, and the period.

- **The entropy:** The entropy $h(\mathbf{P})$ of a probability distribution $\mathbf{P}$ measures the amount of randomness in the distribution $\mathbf{P}$. When we are talking about a dynamical system, it is natural to use the entropy in order to quantify the randomness of this system [21]. In that case, we are interested in finding the entropy of the matrix that describes the system, rather than a probability distribution. Now, we show a specific way to calculate the entropy of a $K-$system.

  As we saw previously, the eigenvalues of the matrix A cannot lie on the unit circle in the complex plane. In fact, in order to obtain sufficient mixing as early as possible, we would need to have the eigenvalues as far as possible from the unit circle [17]. For that reason, if we want to measure the distance between the eigenvalues and the unit circle, we are going to use the Kolmogorov entropy, that is

  $$h(A) := \sum_{k:|\lambda_k|>1} \ln|\lambda_k|, \tag{6}$$

  where the sum is taken over the eigenvalues with absolute values greater than 1 and A is a $N \times N$ constant integer matrix. This entropy measures the disorder in the system, and it must be positive for an asymptotically chaotic system [17].

- **The divergence of nearby trajectories:** We can observe how the mixing process works in a $K-$system by noticing the behavior of two trajectories which start at nearby points in a state space. Let us take a constant integer $N \times N$ matrix A such that $\det(A) = 1$, and let $a(0)$ and $b(0)$ be two points in a state space, separated by a very small distance $D(0) = D(a(0), b(0))$, where D is defined by $D(a, b) = \max_k d_k$ and d_k is given by

  $$d_k := \min\{|a_k - b_k|, 1 - |a_k - b_k|\} \tag{7}$$

  and k runs over the N components of the vector indicated. We can produce new points $a(1)$ and $b(1)$ using the equation (2) and with those points calculate a new distance $D(1) = D(a(1), b(1))$; and if we continue this process to produce two sequences of points $a(i)$ and $b(i)$, we can produce a set of distances $D(i) = D(a(i), b(i))$ in such a way the distance D reaches a plateau at the value expected for truly random points [17] (this value change for the different PRNGs). Therefore, we have the next result: if we have a matrix A that represents a $K-$system, the distances $D(i)$ will diverge exponentially with respect to i, and if we plot the distance on a logarithmic scale, the points $D(i)$ vs. i should lie on a straight line. The rate of those divergences is defined as

  $$v := \ln|\lambda|_{\max}, \tag{8}$$

  where $|\lambda|_{\max}$ is the modulus of the largest eigenvalue of the matrix A and which, for a $K-$system, should be the slope of the straight line described above [17].

- **The asymptotic independence:** We know that, if a and b are two points in a state plane, they are asymptotically independent in a $K-$system, but what happens with the relation between $a(i)$ and $b(i)$, the sequence of points defined above, when their distance $D(i)$ is small? For the development of a PRNG, we need that the sequence of points, $a(i)$ and $b(i)$, to be asymptotically independent. Following the analysis done in the divergence of nearby trajectories, we can observe that when D reaches a plateau at the value expected for truly random points, i indicates the number of iterations required in a $K-$system to be "sufficiently asymptotic"; that is, given two points $a(i)$ and $b(i)$ in the sequence, this two points are going to be asymptotically independent. We may call this criterion the "2-mixing criterion", since it apparently assures that 2×2 correlations due to nearby trajectories will be insignificant [17].

- **The Period:** One property of the discrete systems is that they have finite period. Now, following the definition of the $K-$systems, we can observe that $K-$systems are discrete systems; and for that reason, the trajectories that we use to develop a PRNG have finite period. This previous property means that the trajectory "eats up" each state space as it proceeds, since this space can never return to its previously state until its period finishes. The fact that the available state space becomes progressively smaller indicates a defect in the development of the PRNG, but we can make this weak point undetectable if the period of the trajectory is long enough [17].

In summary, PRNGs based on mixing theory have a great and known behavior because, as mentioned above, some important properties are known, namely: entropy, which determines how random is the distribution; the exponential divergence of nearby trajectories, and also that it is characterized for statistical independence, i.e., that each point is independent of all other points, which is a fact directly related to the trajectories. This properties let us evaluate if these generators are high-quality pseudo-random number generators, because as it was said before, it's important for PRNGs to not show a pattern, not only to look random but to be random enough, and these generators meet that with the mentioned properties [15].

With this general description of PRNGs based on mixing theory, we can proceed to examine some of the most known and successful of them.

2.4. RANLUX PRNG

Until the 90's, random number generators were based on the following theory: given a finite set X, a function $f : X \to X$ and a seed value x, a sequence of values was obtained: $x, f(x), f(f(x))$ and so on. One of the most common kind of RNG was the lagged Fibonacci generators in which X is the set of $1 \times r$ vectors $x = (x_1, x_2, \ldots, x_r)$, each x_i is in some finite set S with a binary operation $\oplus$ and f is defined by $f(x_1, x_2, \ldots, x_r) = (x_2, x_3, \ldots, x_r, x_1 \oplus x_{r+1-s})$ [22]. Taking into account these generators, in 1991, Marsaglia and Zaman in [22] proposed a new class of random number generator: RCARRY, which was based on add with carry (AWC) and subtract with borrow (SWB). Briefly, the generators were of four types:

1. $x_n = x_{n-s} - x_{n-r} - c \mod b,\ m = b^r - b^s + 1.$
2. $x_n = x_{n-r} - x_{n-s} - c \mod b,\ m = b^r - b^s - 1.$
3. $x_n = x_{n-r} + x_{n-s} + c \mod b,\ m = b^r + b^s - 1.$
4. $x_n = b - 1 - x_{n-r} - x_{n-s} - c \mod b,\ m = b^r + b^s + 1.$

Here, r and s are chosen values by the user, b is the base, m is the period and c is the value of the "carry bit" [22]. One characteristic of these new generators is the long period which makes that any defects arising from the finiteness of the state space should remain undetectable. For example for one specific SWB, the period is approximately 5×10^{171} [17].

Some years later, Lüscher realized that if SWB is called 24 times in succession starting with a $24-$vector x_0, it generates another $24-$vector x_1 whose relation with x_0 is given by equation (2), and he could determine the matrix A which would reproduce almost the same sequence as SWB. The only difference would be due to the "carry bit", but it does not alter significantly the mixing properties of the generator. Therefore if

this bit was neglected, the PRNG had a structure that could be represented by equation (2) and related to a $K-$system [17].

On the other hand, RCARRY failed several tests of randomness. Because of this Lüscher in [23] proposed other method, namely RANLUX, which consists in applying RCARRY with decimation; this process is developed in order to reach the value expected for truly random points ($D \approx 12/25$ where D is defined just before equation (7)). More precisely, if p is a value from 24 to 389 chosen by the user, this generator delivers 24 random numbers and discards (decimates) $p-24$ numbers before delivering 24 more. RANLUX offers different degrees of decimation, known as "luxury levels". The lowest luxury levels (small values for p) provide very little or no decimation and are very fast; higher levels are slower but more reliable: Lüscher shows that the quality increases continuously as p varies from 24 up to 389, at which point all 24 bits of mantissa are thoroughly chaotic [17, 24].

The original implementation was made in Fortran and it is described in [24]; currently it is implemented in `C` and `C++`, and incorporated in different libraries. Besides, RANLUX is suited for testing packages that test PRNG's for randomness. If RANLUX passes the test at low luxury levels, then the test is not very sensitive; if it fails at the highest luxury level, then it is likely that the test itself is defective [17].

2.5. MIXMAX PRNG

MIXMAX is a family of PRNG based on the theories of Anosov C-systems defined in [16] and Kolmogorov $K-$systems that we mention in previous subsections. It was developed by George Savvidy in 1991 [25]. In detail, the particular system chosen performs linear automorphisms of the unit hypercube in $\mathbb{R}^n$ determinated by equation (2). The choice of A for MIXMAX is presented in [26] and defined for all $N \geq 3$ as

$$A := \begin{pmatrix} 1 & 1 & 1 & 1 & \cdots & 1 & 1 \\ 1 & 2 & 1 & 1 & \cdots & 1 & 1 \\ 1 & 3+s & 2 & 1 & \cdots & 1 & 1 \\ 1 & 4 & 3 & 2 & \cdots & 1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \\ 1 & N & N-1 & N-2 & \cdots & 3 & 2 \end{pmatrix}. \tag{9}$$

In this case the initial value $x(0) \neq 0$ is the seed and we can see that $A \in \mathbb{Z}^{N\times N}$, Let see that this matrix A satisfies two properties: (i) $\det(A) = 1$ and (ii) their eigenvalues λ_k do not lie on the unit circle , i.e. $|\lambda_k| \neq 1$ for all $k \in \{1, \cdots, N\}$:

(i) To proof this property we are going to proceed by induction over the size of the matrix A which we will denote as A_N, starting with $N = 3$ we have

$$\det(A_3) = \begin{vmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 3+s & 2 \end{vmatrix} = \begin{vmatrix} 2 & 1 \\ 3+s & 2 \end{vmatrix} - \begin{vmatrix} 1 & 1 \\ 1 & 2 \end{vmatrix} + \begin{vmatrix} 1 & 2 \\ 1 & 3+s \end{vmatrix} = (1-s) - (1) + (1+s) = 1.$$

Now suppose that the property is true for $N \geq 3$, lets see that is also true for $N+1$. Note that we can express

$$A_{N+1} = \begin{pmatrix} A_N & \mathbf{1} \\ b^T & 2 \end{pmatrix},$$

where $b^T = (1 \quad N \quad N-1 \quad \cdots \quad 3 \quad 2)$. Since the determinant of A_N is different from zero, then A_N^{-1} exists and we can calculate the determinant of the partitioned matrix as follows:

$$\det(A_{N+1}) = \begin{vmatrix} A_N & \mathbf{1} \\ b^T & 2 \end{vmatrix} = \det(A_N)(2 - b^T A_N^{-1}\mathbf{1}) = (2 - b^T A_N^{-1}\mathbf{1}).$$

To calculate the inverse A_N^{-1}, we must see some particular cases of values of N:

$$A_3^{-1} = \begin{pmatrix} 1-s & s+1 & -1 \\ -1 & 1 & 0 \\ s+1 & -s-2 & 1 \end{pmatrix}, \qquad A_4^{-1} = \begin{pmatrix} s+1 & -s & 1 & -1 \\ -1 & 1 & 0 & 0 \\ s+1 & -s-2 & 1 & 0 \\ -2s & 2s+1 & -2 & 1 \end{pmatrix},$$

$$A_5^{-1} = \begin{pmatrix} 1 & 0 & 0 & 1 & -1 \\ -1 & 1 & 0 & 0 & 0 \\ s+1 & -s-2 & 1 & 0 & 0 \\ -2s & 2s+1 & -2 & 1 & 0 \\ s & -s & 1 & -2 & 1 \end{pmatrix}, \qquad A_6^{-1} = \left(\begin{array}{cc|cccc} \mathbf{1} & 0 & 0 & 0 & 1 & -1 \\ \mathbf{-2+1} & \mathbf{1} & 0 & 0 & 0 & 0 \\ \mathbf{1}+s & \mathbf{-2}-s & \mathbf{1} & 0 & 0 & 0 \\ -2s & \mathbf{1}+2s & \mathbf{-2} & \mathbf{1} & 0 & 0 \\ s & -s & \mathbf{1} & \mathbf{-2} & \mathbf{1} & 0 \\ \hline 0 & 0 & 0 & \mathbf{1} & \mathbf{-2} & \mathbf{1} \end{array}\right).$$

For $N \geq 5$, we have

$$A_N^{-1} = e_{21} + se_{31} - se_{32} - 2se_{41} + 2se_{42} + se_{51} - se_{52} + e_{1(n-1)} - e_{1N} + D_N,$$

where $\{e_{11}, e_{12}, \cdots, e_{1N}, e_{21}, \cdots, e_{NN}\}$ is the canonical basis of $\mathbb{R}^{N\times N}$ and

$$D_N = \begin{pmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ -2 & 1 & 0 & 0 & \cdots & 0 \\ 1 & -2 & 1 & 0 & \cdots & 0 \\ 0 & 1 & -2 & 1 & \cdots & 0 \\ 0 & 0 & 1 & -2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 \end{pmatrix}_{N\times N}$$

Therefore, we obtain

$$b^T A_3^{-1} = (s \ -s \ 1).$$

And for $N \geq 4$,

$$\begin{aligned} b^T A_N^{-1} &= [-(N-1) - (N-1)s + (N-1)(1+s)]e_1^T \\ &\quad + [N + (N-1)(-2-s) - (N-3)s + (N-2)(1+2s)]e_2^T + e_N^T \\ &= e_N^T. \end{aligned}$$

Then we have $b^T A^{-1}\mathbf{1} = 1$, hence $\det(A_{N+1}) = 1$. From this and by the induction principle, the result is valid for all $N \geq 3$.

(*ii*) We can find a proof of this fact in [16]. We exemplify a particular case of our interest, this is when $N = 256$ and $s = 0$.

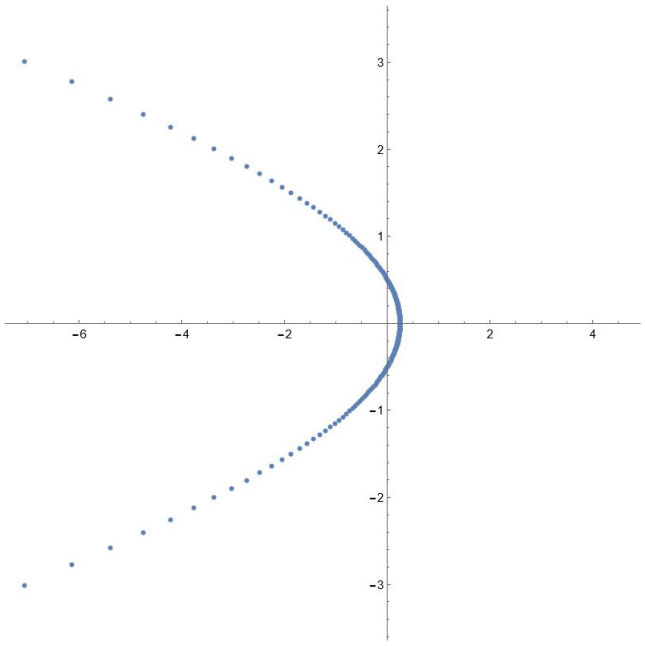

Figure 2: Eigenvalues of the MIXMAX matrix $A(256, 0)$ with the smallest distance from the origin of the complex plane. Some of them lay inside the unit circle and some lay outside of it. The eigenvalues with larger distance from the origin are not depicted in this picture.

With the previous properties and from what we have discussed previously, we can assert the matrix A represents a dynamic system such that the nearby trajectories diverge exponentially.

Clearly, the MIXMAX matrix A is defined recursively. In addition, the only entry of A chosen by the user is $A_{32} = 3 + s$ where $s \in \mathbb{Z}$ and usually $s = 0$ or $s = -1$. If for some s there is an eigenvalue that lies on the unit circle, the only way around is to choose another s.

To generate N random numbers, the straightforward evaluation of the matrix-vector product in (9) requires $O(N^2)$ operations, making it hopelessly slow compared to other common PRNGs. Therefore, a faster algorithm was needed. In [16] we can see that the amount of operations required was reduced to $O(N \ln N)$ with a lot of effort, which is much faster but still not fast enough. It is also mentioned that later Savvidy discovered an algorithm that reduces the amount of operations to $O(N)$, making it competitive with the fastest generators in terms of speed and, more importantly, making the time to generate one number essentially independent of N. Nevertheless, in Section 4 we use the classic algorithm because of its simplicity and easy implementation. MIXMAX random number generator is currently made available by the author Savvidyin a portable implementation in the `C` language at `www.mixmax.hepforge.org/`.

The quality of MIXMAX as a PRNG is explored in [17], where they explain in detail that with the possible exception of $N = 10$ (which some may consider too small) all the generators between $N = 10, 15, 44, 88, 256, 1000$ satisfy our criteria for highest quality PRNGs with no known defects, provided of course that appropriate decimation is applied. MIXMAX also provides the possibility of seeding generators at different remote points to avoid overlapping with sequences used in other calculations. In the next section, we will explore and discuss these statistical characteristics that make MIXMAX such an effective PRNG.

3. Statistical characteristics of MIXMAX PRNG

Stochastic processes are extensively studied in the branch of statistics [27]. Among the different processes, we find a special type of discrete stochastic process in which the probability of an event occurring depends only on the immediately preceding event; this type of processes are known as Markov processes; and its

feature of forgetfulness, as the property of Markov. Contrasting the Markov property with the Mixing Property defined in section 2.2, we can see that the former turns out to be a generalization of the latter.

When simulating arbitrary Markov chains with long correlation lengths, the generator dimension must be greater than the correlation length multiplied by the consumption per update. To give an idea of the size of the generator, in RANLUX it has a state vector of size 24. Unlike generators designed by computer scientists who tend to emphasize even distribution and randomness in terms of bits, MIXMAX has strong theoretical guarantees in terms of floating point output [17].

As it was mentioned in section 2.3, MIXMAX methods provide high quality statistical properties as large entropy and speed, among other remarkable ones. The entropy of this system is nonzero, and it is possible to calculate its value in terms of the eigenvalues of the operator A as it was showed in equation (6). In [28], Savvidy mentions that a system with larger entropy h is more densely populated by the periodic trajectories of the period q.

The matrix in the equation (9) is an operator parametrised by N, the matrix size, and s, the "magic number". As it is showed below in the table taken from [28], the entropy strongly depends on these two parameters.

N	s	Entropy	$\log_{10}$(period)
7307	0	4676.5	134158
20693	0	13243.5	379963
25087	0	16055.7	460649
28883	1	18485.1	530355
40045	-3	25628.8	735321
44851	-3	28704.6	823572

Table 1: Relation between some values of the parameters N and s and the properties of the method, namely the entropy and length of period.

Where the period for the last row is a number of nearly a million digits, which is more than enough for to the sequences that are required in many problems like those mentioned in the introduction, and, according to Savvidy [28], the condition for a system to be empirically acceptable is to have an entropy greater than 50, which is not a problem for MIXMAX as we can see in the table above.

The recursion given in equation (2) can be used to generate uniform random variables in floating point hardware, but as this hardware has finite precision, the full sequence could be assumed to lie in rational numbers. Therefore, the components $x_i(k)$ of the $k-$th vector

$$x(k) = (x_1(k), \ldots, x_N(k)) \tag{10}$$

are of the form $x_i(k) = a_i(k)/p$, where $a_i(k) \in \mathbb{N}$ for $i \in \{1, 2, \ldots, N\}$. This p could also be defined as the larger Mersenne number, which is a number of the form $2^n - 1$, with $n \in \mathbb{N}$, so that is lesser than the maximum integer that is a machine number. In [16], it is mentioned that the period can not exceed the cardinality of the units of the finite vector field, which is equal to $p^N - 1$ if, and only if, the characteristic polynomial $P(x) = (-1)^N \det |A - xI|$ is primitive in the extended Galois finite field (GF$[p^N]$), where p is a Mersenne prime. Thus, the period of the recursion hits $p^N - 1$, its maximal value, independent of the seed, but this is not possible unless $p = 2$. In this paper it is also analyzed the maximum period a sequence can have using primitive characteristic polynomials and that the matrix is idempotent.

The implementation of MIXMAX in [16] uses 64-bit integer arithmetic internally, so that the results obtained there can be used at the next section computing the algorithm. Some of these results are that, for a matrix of dimension N from 10 to 3150, the respective period is longer than 10^{165}, and that in practice $N = 256$ became the default value because of the results of the Big Crush test, which runs about 160 tests over any given PRNG. Futhermore, MIXMAX with $N = 256$ requires decimation but less than RANLUX, which makes it one of the world's fastest high-quality PRNG [17].

3.1. Extended MIXMAX

In order to reach faster convergence to asymptotic mixing, a generating matrix with larger entropy was searched through transformations of the matrix A presented in (9), which do not change the determinant but do move the eigenvalues further away from the unit circle. One of these modifications was introduced using a new integer m to produce a new group of matrices which is called the three-parameter family $A(N, s, m)$, constructed as

$$A(N,s,m) := \begin{pmatrix} 1 & 1 & 1 & 1 & \cdots & 1 & 1 \\ 1 & 2 & 1 & 1 & \cdots & 1 & 1 \\ 1 & m+2+s & 2 & 1 & \cdots & 1 & 1 \\ 1 & 2m+2 & m+2 & 2 & \cdots & 1 & 1 \\ 1 & 3m+2 & 2m+2 & m+2 & \cdots & 1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & (N-2)m+2 & (N-3)m+2 & (N-4)m+2 & \cdots & m+2 & 2 \end{pmatrix}. \tag{11}$$

Savvidy in [28] points out that efficient implementation in software can be achieved for particular choices of m of the form $2^k + 1$. Besides, he selected some values for N, s and m and studied their entropies and periods; some are showed in the Table 2 taken from [28]. In this we can see, for example, the increase on the entropy for small sizes N compared with original MIXMAX. It is the case of $N = 7307$, whose entropy $h = 4676,5$ in the original matrix; and $N = 240$ with entropy $h = 8679,2$.

N	s	m	Entropy	$\log_{10}$(period)
8	0	$2^{53}+1$	220.4	129
17	0	$2^{36}+1$	374.3	294
40	0	$2^{42}+1$	1106.3	716
60	0	$2^{52}+1$	2090.5	1083
96	0	$2^{55}+1$	3583.6	1745
120	1	$2^{51}+1$	4171.4	2185
240	4870132302560991	$2^{51}+1$	8679.2	4389

Table 2: Relation between some values of the parameters N, s and m, and the properties of the method, namely the entropy and length of period.

Taking into account the previous results, it is possible to conclude that for the new family of parameters, $N = 240$ with its respective s and m have the best stochastic properties. Despite this, the spectral properties of the matrix generators depends on m and the best are achieved when m is between 2^{24} and 2^{36}. Making analysis with this consideration for $N = 8$ with $m = 2^{36} + 1$, and $N = 240$ with $m = 2^{32} + 1$, spectral indexes of $1,49 \cdot 10^{-8}$ and $7.6 \cdot 10^{-10}$ were obtained respectively [29]. Although entropy increases applying extended MIXMAX, these generators don't exhibit exponential divergence as they should for a $K-$system [17].

3.2. The spectral test of MIXMAX

The spectral test is another important way to check the quality of the pseudo-random numbers generated by PNRG's. Not only do all good generators pass this test, all generators now known to be bad actually fail it. Thus this is by far the most powerful test known [30]; for that reason, we are going to use this test for verify the quality of MIXMAX.

The spectral test embodies aspects of an empirical test and a theoretical test. It has a theoretical test part because deals with the full period of the pseudo-random number sequence; and it has an empirical test part because requires a computer program to determine the results [30].

As we know, MIXMAX is based on the theory of Kolmogorov $K-$systems and, for that reason, the sequence produced by MIXMAX is given by (2). Now, let's take the following vectors

$$g(i) = (x(i), x(i+1), \cdots, x(i+r-1)) \in [0,1)^{rN}, \tag{12}$$

where r is the number of successive outputs of the generator, $r > 1$, N is the length of the matrix A and $i = 0, 1, \cdots, p^N - 1$, where p is the denominator of each rational coordinates of the vector $x(i)$.

In [31, 32], we can observe that, independently of the parameters N and s of the operator $A(N, s)$ and of the operator $A(N, s, m)$ of the MIXMAX extended version, the spectral index is $l_{rN} = \frac{1}{\sqrt{3}}$; which is the inverse of the maximal distance between the set of hyper-planes covering all the points [30]. Knowing this index, L'Ecuyer finds in [32] a relationship between the second, the $(N+1)-$th and the $(N+2)-$th coordinates of (12), that goes as follows:

$$g(i)_2 + g(i)_{N+1} - g(i)_{N+2} = \begin{cases} 0, \\ 1. \end{cases} \tag{13}$$

This relationship also exists for MIXMAX extendend PRNG, and goes as follows

$$g(i)_j - 2g(i)_{j+1} + g(i)_{j+2} - g(i)_{j+N+2} - (m-1)g(i)_{j+N+1} = 0, \qquad \mod 1, \tag{14}$$

when $j = 4, \cdots, N-2 \mod N$ [29]. This last relationship tell us that the spectral index is inversely proportional to m; and for that reason, MIXMAX extended has an advantage over the original MIXMAX PRNG; and if m is in the region between 2^{24} and 2^{36}, we would have a better spectral index [29].

3.3. Monte Carlo method and MIXMAX

We are interested in the relation between Monte Carlo method and MIXMAX PRNG, in order to determine whether it is appropriate to apply the latter in the former, and what considerations we must make while doing so.

For starters, we have seen in this section and throughout the report that the MIXMAX family methods have statistical characteristics of very high quality. Specifically, they have large entropy, the best relation between speed and size of the state, tuneable parameters and availability for implementing the parallelization. A key fact for optimizing these characteristics is that the speed of the generator does not depend of N; conversely, the quality of the other properties increases with N [28]. This implies that, generally, we want to apply MIXMAX with large values of N. Savvidy used Leonov's theorem to proved that, for a MIXMAX application to a Monte Carlo approximation of a $D-$dimensional problem, if we take $N \geq D$, the numerical simulation converges to the real solution of the problem [33, 34].

Now let's look at some combination of parameters that have proved to be optimal for Monte Carlo implementation. If we consider just the two parameters N and s, the combination that gives the best relation of speed, reasonable size and availability for implementing the parallelization by skipping is $N = 256$ and $s = 487013230256099064$ [28]. This election of parameters is also optimal if we consider the extended MIXMAX with N, s and m. In that case, the best election for m is $m = 2^{51} + 1$ [28].

Taking those last conclusions into account, we are now going to present some applications of MIXMAX method using $N = 256$ and $s = 487013230256099064$.

4. Applications and computational results

One of the applications which we are going to develop is a classic approximation of the value of Pi (π) using Monte Carlo method. This process consists in considering a square and a circle inscribed in it, and drawing points randomly inside the square. Thereafter, the areas of the circle and the square are estimated with the number of points inside the circle and the number of total points, respectively.

As we know, the area of a square with side l is $A_s = l^2$ and the area of the circle inscribed in it is $A_c = \pi(l/2)^2$. This gives the following relation between the two areas: $A_c = \pi A_s/4$. Using this, we can

estimate π as four times the quotient between our estimations of the areas of the circle and the square. This process is illustrated in Figure 3.

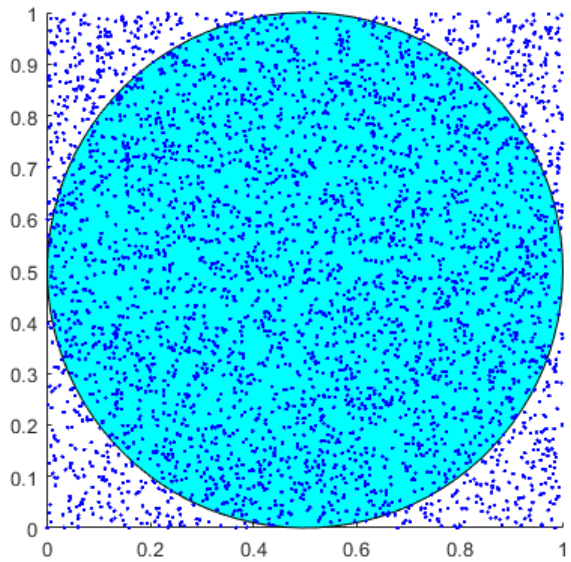

Figure 3: Illustration of the approximation of π with Monte Carlo method using the relation between the areas of a square and a circle inscribed in it. In this case, we use a small amount of random points (3800) and we obtain an approximation of 3.125263157894737.

It is easy to see that the approximation obtained is not so precise. Therefore, we can try to draw larger amounts of random points (which are nothing more than pairs of random numbers) in the square in order to get better approximations of the relation between the two areas.

We run some simulations using MIXMAX PRNG with different values of N and s and the results are summarized in Table 3. For the case $N = 256$, we implement decimation by skipping one of every three random numbers generated. We also do the same experiment using the random numbers generated by `rand`, the default function for generating random numbers in MATLAB. The results of those simulations are summarized in Table 4.

N	s	p	Number of points	Approximation	Standard deviation	Error
20693	0	1	20693	3.142774851399023	0.011649869728158	0.001182197809230
7307	0	3	21921	3.141312896309476	0.011854250636122	0.000279757280317
256	0	130	22230	3.141840755735493	0.012927858580200	0.000248102145700
256	-1	130	22230	3.141333333333333	0.010295535230836	0.000259320256460
256	s_0	130	22230	3.141930724246513	0.010029809760939	0.000338070656720
256	s_0	150	25650	3.141507992202730	0.009966416038700	0.000084661387063
256	s_0	200	34200	3.141614035087719	0.008486833817344	0.000021381497926

Table 3: Results of the simulations using Monte Carlo method and MIXMAX PRNG to approximate π based on the relation between the areas of a square and a circle inscribed in it. The parameter p indicates that, for each simulation, the algorithm that generates N random numbers is run p times, giving a larger list of points. The number of points is given by $p \cdot N'$, where N' is the length of the vector produced by the iteration number 120 of $x(i+1) = A\ x(i)$, where $A = A(N, s)$, after considering decimation by skipping. We use the notation $s_0 = 487013230256099064$. For each set of values of the parameters, the simulation is repeated 50 times and the final value of the approximation is the average of those 50 results.

Number of points	Approximation	Standard deviation	Error
20693	3.143401150147392	0.012106908530085	0.001808496557599
21921	3.139769171114457	0.009403230622164	0.001823482475336
22230	3.141822762033288	0.012798727084675	0.000230108443495
25650	3.141158674463937	0.010562103070116	0.000433979125856
34200	3.145129824561403	0.009360633209391	0.003537170971610

Table 4: Results of the simulations using the `rand` function in Matlab and Monte Carlo method to approximate π based on the relation between the areas of a square and a circle inscribed in it. The number of points for each simulation is chosen to so that it matches the number of points used with MIXMAX. For each values of that parameter, the simulation is repeated 50 times and the final value of the approximation is the average of those 50 results.

The concept of approximating areas using Monte Carlo method can be also used to approximate the area under a curve and, therefore, proper defined integrals. For example, if we consider

$$I = \int_0^1 f(x)dx, \quad f(x) = \frac{1}{1+x^2},$$

we can do a simple calculation to show that $\pi = 4I$. Thus, if we draw some points inside the unit square and count how many of them lay under the curve defined by f, we obtain an approximation of I, which can be multiplied by 4 to get an approximation of π as illustrated in Figure 4

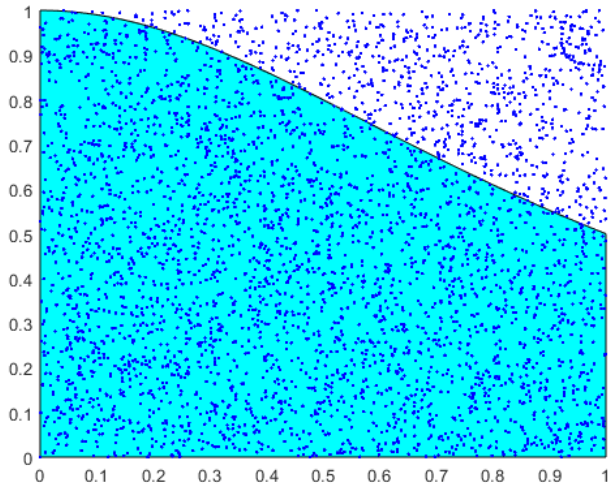

Figure 4: Illustration of the estimation of π with Monte Carlo method to approximate the integral between 0 and 1 of the function $f(x) = 1/(1+x^2)$. In this case, we use a small amount of random points (3800) and we obtain an approximation of 3.150526315789474.

We run some simulations using the same parameters and specifications that we use for our first application. The results are summarized on tables 5 and 6.

N	s	p	Number of points	Approximation	Standard deviation	Error
20693	0	1	20693	3.143621514521819	0.010113032724455	0.002028860932026
7307	0	3	21921	3.142776333196478	0.012294861793064	0.001183679606685
256	0	130	22230	3.141462887989205	0.011380300110278	0.000129765600588
256	-1	130	22230	3.141538461538462	0.010469865447343	0.000054192051331
256	s_0	130	22230	3.141642825011246	0.011564491495779	0.000050171421453
256	s_0	150	25650	3.142231578947369	0.010734311417924	0.000638925357576
256	s_0	200	34200	3.142269005847953	0.008846514282170	0.000676352258160

Table 5: Results of the simulations using Monte Carlo method and MIXMAX PRNG to approximate π based on the approximation of an integral. The parameter p indicates that, for each simulation, the algorithm that generates N random numbers is run p times, giving a larger list of points. The number of points is given by $p \cdot N'$, where N' is the length of the vector produced by the iteration number 120 of $x(i+1) = A\ x(i)$, where $A = A(N, s)$, after considering decimation by skipping. We use the notation $s_0 = 487013230256099064$. For each set of values of the parameters, the simulation is repeated 50 times and the final value of the approximation is the average of those 50 results.

Number of points	Approximation	Standard deviation	Error
20693	3.142044169525925	0.010913323133566	0.000451515936132
21921	3.141426029834406	0.010045682277892	0.000166623755387
22230	3.139580746738642	0.011760010365316	0.002011906851151
25650	3.141919688109161	0.010447329361771	0.000327034519368
34200	3.144638596491229	0.007768996314193	0.003045942901436

Table 6: Results of the simulations using the `rand` function in Matlab and Monte Carlo method to approximate π based on the the approximation of an integral. The number of points for each simulation is chosen to so that it matches the number of points used with MIXMAX. For each values of that parameter, the simulation is repeated 50 times and the final value of the approximation is the average of those 50 results.

We can see that the results of our simulation using MIXMAX PRNG are decent approximations of Pi considering the coarseness of the experiment. In fact, when compared to similar simulations like the one that we do using the `rand` function or the one done in [12], MIXMAX results stand out because of their precision and consistency. It is also worth pointing out that the results that we obtain using $N = 256$ have similar (and even greater) precision that the ones with higher values of N even though the calculations are much faster for the lower value, which confirms the theoretical ideas stated in previous sections.

Regarding the `rand` function that MATLAB uses by default, it initializes Mersenne Twister (MT) generator with seed 0. This PRNG has period $2^{19937} - 1$ and, in MATLAB's default, each value is generated with double precision. Possible values are multiples of 2^{-53} on the interval $(0, 1)$ with uniform distribution [35]. As we saw previously, MIXMAX also generates values with double precision, and considers a seed which necessarily is different of 0 and changes each time that the algorithm is implemented, which may be one of the reasons that explain why we get better approximations with this PRNG. On the other hand, calculations with `rand` are faster than with MIXMAX. One explanation for this is that MT requires $O(N)$ operations to generate N random numbers [35], the same computational complexity as the fastest version of MIXMAX, which is not the one that we use in this report. Our version of MIXMAX algorithm requires $O(N^2)$ operations in order to generate N random numbers. In this order of ideas, new comparisons that involve speed can be made in the future implementing the more modern version of MIXMAX algorithm.

The default PRNG used by MATLAB can be changed with the optional inputs of the `rand` function, but Mersenne Twister is consider to be the best option that this software offers because of its statistical characteristics. For instance, it has the longest period and best speed [36].

5. Conclusions and discussion

In this report we study the PRNGs based on theory of mixing and classical dynamical systems, deepening in MIXMAX PRNG. We can conclude that this PRNG shows good statistical properties like its entropy, divergence of nearby trajectories, mixing and length of period, which is reflected in the quality of the random numbers generated. We analyze this quality both theoretically and experimentally, performing well when compared against the more generic PRNGs such as Mersenne Twister, which is used by MATLAB to generate random numbers by default. This shows that we can use classical dynamical systems to establish PRNGs that have nothing to envy from the more popular ones.

Another advantage of the PRNGs such as MIXMAX (that is, those based on classical dynamical systems) that we notice throughout this report is that they have a simple statement. The quality of the PRNG is strongly linked to the properties of its matrix, which helps us to study them as in tandem but also individually. This fact also implies that new high quality PRNGs can be created in the future if new matrices with the correct properties were to be found.

We can also conclude that the quality of the random numbers generated by classical dynamical systems is not necessarily linked to the size of the matrix of this system, and MIXMAX is an example of this fact as we show in our simulations. Thus, we could search for higher quality PRNGs based on dynamical systems without increasing excessively the dimensions of the matrix.

One aspect of MIXMAX PRNG that we don't study experimentally in this report is its performance in terms of speed and its relation with the quality of the numbers generated, due to the fact that we don't use the fastest version of MIXMAX's algorithm and, therefore, it wouldn't be a fair analysis. Nevertheless, it can be an interesting factor to consider in the comparison against MATLAB's PRNG, specially because we can also use it to test the theoretical ideas about that subject mentioned in this report.

References

[1] L. Bassham, A. Rukhin, J. Soto, J. Nechvatal, M. Smid, S. Leigh, et al., A statistical test suite for random and pseudorandom number generators for cryptographic applications, Tech. rep., Special Publication (NIST SP), National Institute of Standards and Technology, Gaithersburg, MD (2010).

[2] L. Zhou, Z. Wang, Numerical simulation of cavitation around a hydrofoil and evaluation of a rng $\kappa - \varepsilon$ model, Journal of Fluids Engineering 130 (1). doi:https://doi.org/10.1115/1.2816009.

[3] S. Raychaudhuri, Introduction to Monte Carlo simulation, WSC '08, Winter Simulation Conference, 2008, pp. 91–100. doi:https://dl.acm.org/doi/10.5555/1516744.1516768.

[4] J. Bischoff, R. Gold, J. Horton, Music for an interactive network of microcomputers, Computer Music Journal 2 (3) (1978) 24–29. doi:https://doi.org/10.2307/3679453.

[5] S. K. Das, C. Nita-Rotaru, M. Kantarcioglu (Eds.), Decision and Game Theory for Security: 4th International Conference, GameSec 2013, Fort Worth, TX, USA, November 11-12, 2013, Proceedings, 1st Edition, no. 8252 in Security and Cryptology, Springer International Publishing : Imprint: Springer, Cham, 2013.

[6] J. E. Gentle, Random number generation and Monte Carlo methods, oCLC: 1073466407 (2005).

[7] S. Pironio, A. Acín, S. Massar, A. Boyer de la Giroday, D. N. Matsukevich, P. Maunz, et al., Random numbers certified by Bell's theorem, Nature 464 (7291) (2010) 1021–1045. doi:10.1038/nature09008.

[8] J. Garnier, Wave propagation in one-dimensional random media, Vol. 12, Panoramas et Synthèses, 2001.

[9] E. Uner, Generating random numbers, Embedded system.

[10] N. Haller, C. Metz, P. J. Nesser, M. Straw, A one-time password system, RFC 2289 (1998) 1–25.

[11] I. M. Sobol, A primer for the Monte Carlo method, Boca Raton, 1994.

[12] G. Addati, F. Celano, J. Churruarín, Modelos y simulación para aproximar el valor de pi, no. 591 in Documentos de Trabajo, Universidad del CEMA, 2016, pp. 1–27.

[13] K. Sathya, J. Premalatha, V. Rajasekar, Investigation of strength and security of pseudo random number generators, in: IOP Conference Series: Materials Science and Engineering, Vol. 1055, IOP Publishing, 2021, p. 012076.

[14] P. Walters, An introduction to ergodic theory, no. 79 in Graduate texts in mathematics, Springer-Verlag, New York, 1982.

[15] F. James, Finally, a theory of random number generation, 2002, pp. 114–121. doi:10.1142/9789812777140_0009.

[16] G. Konstantin, The mixmax random number generator, Computer Physics Communications 196 (2015) 161–165.

[17] F. James, L. Moneta, Review of high-quality random number generators, Comput Softw Big Sci 4 (2). doi:https://doi.org/10.1007/s41781-019-0034-3.

[18] J. Wiśniewska, Comparing quality of pseudo- and truerandom numbers obtained from different sources, Symulacja w Badaniach i Rozwoju 9 (1) (2018) 73–82.
URL http://yadda.icm.edu.pl/yadda/element/bwmeta1.element.baztech-364d5315-a7ab-4474-8c49-269dbcca1d60/c/Wisniewska_Comparing_quality_SwBiR_V9_1-2_2018.pdf

[19] L. D. Landau, E. M. Lifshitz, Mechanics, Vol. 1 of Course of Theoretical Physics., Sykes, J. B. (John Bradbury), Bell, J. S. (3rd ed.). Oxford, 1976.
[20] J. Berkovitz, R. Frigg, F. Kronz, The ergodic hierarchy, random- ness and hamiltonian chaos, Stud Hist Philos Mod Phys 37 (2006) 661–691.
[21] K. A., Probability Theory, Springer-Verlag London, 2014.
[22] G. Marsaglia, A. Zaman, A new class of random number generators, The Annals of Applied Probability 1. doi:10.1214/aoap/1177005878.
[23] M. Lüscher, A portable high-quality random number generator for lattice field theory simulations, Computer Physics Communications 79 (1) (1994) 100–110. doi:https://doi.org/10.1016/0010-4655(94)90232-1.
URL https://www.sciencedirect.com/science/article/pii/0010465594902321
[24] F. James, Ranlux: A fortran implementation of the high-quality pseudorandom number generator of lüscher, Computer Physics Communications 79 (1) (1994) 111–114. doi:https://doi.org/10.1016/0010-4655(94)90233-X.
URL https://www.sciencedirect.com/science/article/pii/001046559490233X
[25] G. Savvidy, N. Ter-Arutyunyan-Savvidy, On the monte carlo simulation of physical systems, Journal of Computational Physics 97 (2) (1991) 566–572. doi:https://doi.org/10.1016/0021-9991(91)90015-D.
[26] N. Akopov, G. Savvidy, N. Ter-Arutyunyan-Savvidy, Matrix generator of pseudorandom numbers, Journal of Computational Physics 97 (2) (1991) 573–579. doi:https://doi.org/10.1016/0021-9991(91)90016-E.
URL https://www.sciencedirect.com/science/article/pii/002199919190016E
[27] S. M. Ross, J. J. Kelly, R. J. Sullivan, W. J. Perry, D. Mercer, R. M. Davis, T. D. Washburn, E. V. Sager, J. B. Boyce, V. L. Bristow, Stochastic processes, Vol. 2, Wiley New York, 1996.
[28] K. Savvidy, G. Savvidy, Spectrum and entropy of c-systems mixmax random number generator, Chaos, Solitons & Fractals 91 (2016) 33–38. doi:https://doi.org/10.1016/j.chaos.2016.05.003.
URL https://www.sciencedirect.com/science/article/pii/S0960077916301576
[29] N. Martirosyan, K. Savvidy, G. Savvidy, Spectral test of the mixmax random number generators, Chaos, Solitons & Fractals 118 (2019) 242–248. doi:https://doi.org/10.1016/j.chaos.2018.11.024.
[30] D. E. Knuth, The Art of Computer Programming, Volume 2: Seminumerical Algorithms, 3rd Edition, Addison-Wesley, Boston, 1997.
[31] S. Konitopoulos, K. G. Savvidy, A priori tests for the mixmax random number generator (2018). arXiv:1804.01563.
[32] P. L'Ecuyer, P. Wambergue, E. Bourceret, Spectral Analysis of the MIXMAX Random Number Generators, INFORMS Journal on Computing (2018) 1–18.
URL https://hal.inria.fr/hal-01634350
[33] V. D. Leonov, On the central limit theorem for ergodic endomorphisms of the compact commutative groups, Dokl. Acad. Nauk SSSR, 124 (5) (1969) 980–983.
[34] G. Savvidy, Anosov c-systems and random number generators, Theor Math Phys 188 (2016) 1155–1171. doi:https://doi.org/10.1134/S004057791608002X.
[35] M. Matsumoto, T. Nishimura, Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator, ACM Trans. Model. Comput. Simul. 8 (1) (1998) 3–30. doi:10.1145/272991.272995.
URL https://doi.org/10.1145/272991.272995
[36] H.-b. SHEN, X.-m. LI, J. YU, X.-z. PEN, X.-l. YAN, The noise model and fast simulation of a random number generator [j], Acta Simulata Systematica Sinica 4.