Generador de Números Pseudoaleatorios Mediante un Sistema Caótico

Pseudorandom Number Generator Using a Chaotic System

Xxxxx xxxxxxx xxxxxxxxx xxxxxx xxxxxxxx
Xxxxxx xxxxxxx xxxxxxxx xxxxxx xxxxx xxxxx
Xxxx xxxxx xxxxxx
Xxxxxx xxxxxxx xxxxxxxx xxxx
Xxxxx xxxxxxxxx xxxxxx
Xxxxx xxxxxx xxxxxxxxx xxxx Xxxxx xxxxxx
xxxxxxxxxx xxxxxxx xxxxxxxx

Xxxxxxx xxxxxx xxxxxx
XXXXX xxxx xxxxxxxxx xxx
Xxxxxxxxxx xxxxxxx xxxx
Xxxxxxx xxxxxx xxxxx

Resumo — **En este artículo, se presenta una propuesta de un generador de números pseudoaleatorios mediante una modificación de la ecuación de Van de Pol propuesta por Ueda y Akamatsu como un oscilador forzado de resistencia negativa. Este sistema de ecuaciones ha sido simulado mediante la herramienta de cómputo matemático MATLAB. Se obtuvieron sus soluciones estimadas para cada una de sus variables mediante el método de Runge-Kutta de cuarto orden, siendo este un método numérico bastante popular y confiable para la aproximación de la solución de ecuaciones diferenciales. El sistema, cuenta con características caóticas y utiliza un rango de valores en sus constantes, las soluciones obtenidas mediante el método numérico se plantearon como valores que pudiesen presentar características pseudoaleatorias, siendo fundamentado mediante las pruebas estadísticas para valores aleatorios y pseudoaleatorios propuesta por el NIST tras un procesamiento de los datos obtenidos por el método matemático.**

Palabras Clave - Complejidad lineal, excursiones aleatorias variables, generadores pseudoaleatorios, MATLAB, NIST, punto fijo, Runge-Kutta, sistemas caóticos, Van der Pol

Abstract — **In this paper, a proposal of a pseudorandom number generator is presented by means of a modification of the Van de Pol equation proposed by Ueda and Akamatsu as a negative resistance forced oscillator. The proposal has been divided into two parts, the first part consists of the simulation of the system of equations using the mathematical computation tool MATLAB. Its estimated solutions were obtained for each of its variables using the fourth-order Runge-Kutta method, being this a quite popular and reliable numerical method for the approximation of the solution of differential equations. The system, having chaotic characteristics when using a certain range of values in its constants, and the solutions obtained using the numerical method were considered as values that could present pseudo-random characteristics. The second part consists of substantiating, utilizing the statistical tests for random and pseudo-random values proposed by NIST, that these values present pseudo-random characteristics.**

Keywords - Linear complexity, variable random excursions, pseudo-random generators, MATLAB, NIST, fixed point, Runge-Kutta, chaotic systems, Van der Pol.

I. INTRODUCCIÓN

La generación de secuencias de bits verdaderamente aleatorias es comúnmente complicada debido a la cantidad de dificultades técnicas, al tiempo requerido para crear secuencias largas y el costo. Cuando un generador pseudoaleatorio de números (PRNG) es considerado apto para un uso criptográfico, es conocido como un generador de números pseudoaleatorios criptográficamente seguro (CSPRNG), cuando este cumple con ciertos requisitos estrictos. Entre ellos, el generador debe ser capaz de proveer tanto la imprevisibilidad de los valores generados, es decir que estos valores no puedan ser predecibles, y la indistinguibilidad de sus propiedades estadísticas de las propiedades de las secuencias verdaderamente aleatorias [1]. Actualmente, no existe un método universalmente acordado para la evaluación numérica de las secuencias de bits pseudoaleatorias con respecto a las secuencias verdaderamente aleatorias. De igual manera, no hay un solo criterio para comparar la calidad de los PRNGs. Dada la gran variedad de generadores pseudoaleatorios y su importancia para proveer comunicaciones seguras, el problema de como evaluar la calidad un PRNG es muy relevante [2]. Contar con propiedades estadísticas buenas es un requerimiento esencial para las salidas de un PRNG. Los algoritmos comúnmente usados incluyen generadores congruentes lineales, de Fibonacci con retraso y secuencias-m. En criptografía, un PRNG pobre pudiera causar que sus llaves sean débiles o predecibles. Existen diversos conjuntos de pruebas orientados a la medición de aleatoriedad de las secuencias de números generados por un PRNG [3].

A. Generadores de números pseudoaleatorios

Una de las maneras más adecuadas y fiables de generar números aleatorios es emplear algoritmos deterministas que tengan una base matemática sólida. La secuencia generada es

parecida a una secuencia de variables aleatorias independientes e idénticamente distribuidas por otra variable aleatoria, aunque en realidad no es así, por lo que los valores generados son conocidos como números pseudoaleatorios. Estos generadores deben contar con las propiedades mencionadas en la TABLE I, aunque existen algunos tipos de generadores no cumplan con algunas de estas [4].

TABLE I. PROPIEDADES DE UN BUEN PRNG

Propiedades deseables de un generador
La secuencia de valores que proporciona debe parecerse a una secuencia de realizaciones independientes de un variable aleatoria U(0, 1).
Desde la perspectiva de una simulación estadística los resultados deben ser reproducibles y desde la perspectiva criptográfica los resultados deben ser impredecibles.
La secuencia de valores generados debe contar con un ciclo no repetitivo lo más largo posible.
El generador debe ser rápido y ocupar una pequeña porción de recursos internos.
Es deseable que el generador sea portátil.

El generador de cuadrados centrales está basado en el método de cuadrados centrales propuesto por John Von Neumann en 1946. Es considerado como uno de los primeros métodos para la generación automática de números pseudoaleatorio. Su principio esta descrito en la TABLE II, la ventaja de este método es su simplicidad, ya que se genera una secuencia de números con una cantidad de *2k* dígitos. Para obtener el siguiente número, se eleva el valor anterior al cuadrado y se retienen *k* dígitos del valor obtenido como el siguiente valor [5].

TABLE II. PASOS DEL MÉTODO DE CUADRADOS CENTRALES

Proceso por seguir
Inicia con una semilla (un número) de *2k* dígitos.
Se eleva al cuadrado para obtener un número de *2k* dígitos, agregando ceros si es necesario.
Se toman los *k* dígitos centrales como el siguiente valor aleatorio.
Se repiten los pasos anteriores.

El método de congruencia lineal es uno de los algoritmos más empleados para la generación de números aleatorios en las simulaciones computaciones. Este método está definido por

$$x_n = a\, x_{n-1} + c \pmod{M} \qquad (1)$$

Donde a, c y M son enteros y x_n es una secuencia de enteros con un rango desde 0 hasta M. Durante los años 60's, los valores usados popularmente eran c=0 y $M=2^w$, siendo *w* un número menor o igual a 4. En este caso mencionado, el valor de *a* era igual a 3 o 5, lo cual es una condicion necesaria y suficiente para el período máximo. Una de las decisiones comunes con este método es escoger M como un número primo y el valor de a como una raíz primitiva módulo de M . Uno de los generadores más populares es conocido como GGL, una subrutina crea por IBM [6]. Las secuencias generadas por los generadores de congruencia lineal (Lineal Congruency Generator, LCG) son predecibles, algunos autores tomaron este problema y propusieron soluciones para este problema de predictibilidad en varios generadores de este tipo comúnmente usados. Aunque algunos otros autores advirtieron sobre el peligro de usar LCGs para aplicaciones de criptografía, a menos que la secuencia generada sea aislada mediante otro generador [7]. Entre los análisis realizados por algunos autores, se probó que este tipo de generador son predecibles de manera eficaz incluso cuando los coeficientes y los módulos son desconocidos para el predictor. Las secuencias generadas pueden ser predichas cuando el predictor conoce las funciones empleadas [8].

Los generadores de Fibonacci con retraso (Lagged Fibonacci Generator, LFG) están especificados por la ecuación.

$$x_n = x_{n-p} \otimes c \pmod{M} \qquad (2)$$

Donde el operador $\otimes$ denota que la operación puede ser +, -, ×, o una operación OR exclusiva. *p* es llamado el retraso del generador. Un generador de Fibonacci aditivo con retraso esta especificado por

$$x_k = x_{k-p} \pm x_{k-p+q} \pmod{M} \qquad (3)$$

Estos generadores cuentan con un rendimiento superior al de un LCG y un LFG que emplean la operación OR exclusiva, ya que cuentan con un período mayor dado por la ecuación 4, mientras que el periodo del LFG con una OR exclusiva esta dado por la ecuación 5.

$$(2^p - 1)2^{l-1} \qquad (4)$$

$$(2^p - 1) \qquad (5)$$

El período de este tipo de generador puede ser largo si el valor de p es escogido apropiadamente con el mismo tamaño de palabra. Sin embargo, este no se encuentra limitado por el tamaño de palabra de la máquina con la que el generador es implementado [9].

Las secuencias-m son empleadas para diversos usos, desde criptografía, detección de errores y pruebas para circuitos VLSI. Los generadores de este tipo pueden generar secuencias periódicas que parecen ser aleatorias. Estas secuencias son generadas mediante un algoritmo el cual usa entradas como la longitud de la secuencia-m, las conexiones de retroalimentación y una semilla inicial para el generador. A pesar de no ser estadísticamente aleatorias, cumplen con pruebas de aleatoriedad como autocorrelación, correlación-cruz, serial, de frecuencia. Su secuencia es imposible de predecir, a menos que se conozca el algoritmo y su semilla. Los generadores de este tipo han empleado registros de desplazamiento de realimentación lineal (Lineal Feedback Shift Register, LFSR) debido a las ventajas mostradas en la TABLE III [10].

TABLE III. VENTAJAS DEL USO DE LSFRS EN GENERADORES DE SECUENCIA-M

Razones
Son adecuados para realizar la implementación en hardware.
Pueden producir secuencias con un período grande.
Pueden producir secuencias con buenas propiedades estadísticas.
Pueden ser analizadas mediante técnicas algebraicas gracias a su teoría matemática para distintas estructuras.

El valor inicial del LFSR es la semilla del generador, sin embargo, debido a que este tipo de registro cuenta con un número finito de posibles estados, eventualmente entrara en un ciclo repetitivo. Este ciclo puede llegar a ser lo suficientemente grande si el LSFR cuenta con una función de retroalimentación escogida correctamente con la que se logré generar la secuencia de bits que pueda parecer ser aleatoria con ciclo repetitivo grande [11].

II. SISTEMAS CAÓTICOS

Como muchos términos en la ciencia, no hay una definición estándar del caos. Sin embargo, el caos puede contar con las siguientes características mostradas en la TABLE IV. Un ejemplo de un sistema caótico en ciencias computaciones son los generadores pseudoaleatorios. En la naturaleza también podemos encontrar sistemas caóticos, como la actividad normal del cerebro que se considera puede ser caótica o la especulación de que mucha periodicidad en los latidos del corazón pudiera indicar alguna enfermedad [12].

TABLE IV. CARACTERÍSTICAS DEL CAOS

Principales características
Si es lineal, no puede ser caótico.
Es determinista ya que cuenta con reglas subyacentes deterministas que cada estado futuro debe de seguir.
Es sensible a las condiciones iniciales, lo cual puede modificar su estado final.
Irregularidad sostenida del sistema, cuenta con un número grande o infinito de patrones periódicos.
Su predicción a largo plazo es imposible debido a la sensibilidad a las condiciones iniciales.

En 1963, Lorenz realizo una publicación donde mostro lo que actualmente es llamado caos, puede ocurrir en sistemas de ecuaciones diferenciales autónomas con tres variables y dos no-linealidades cuadráticas. Rössler encontró sistemas similares en 1976 y 1979. El primer sistema cuenta con una sola no-linealidad cuadrática, el cual es similar al sistema de Lorenz al contener siete términos al ser escrito como tres ecuaciones diferenciales de primer orden. El segundo sistema es un sistema toroidal con seis términos y una no-linealidad cuadrática [13]. En los años 20, Van der Pol y Van der Marker desarrollaron una ecuación de segundo orden de oscilación, con la cual se modelaba el fenómeno de generado por un marcapasos impulsando a un corazón. Dicha ecuación ha recibido varias modificaciones con la finalidad de modelar fenómenos físicos, biológicos, entre otros [14]. Una de sus propiedades particulares, llamada ciclo límite, permite que la ecuación sea empleada para el modelado de fenómenos y sistemas que oscilan de manera periódica y estable [15]. La ecuación 6 muestra la ecuación original propuesta en 1926.

$$d^2x / dt - e\,(1 - x^2)\,dx / dt + x = 0 \qquad (6)$$

Aplicando una función de forzamiento, la ecuación 6 se convierte en la ecuación 7

$$d^2x / dt - e\,(1 - x^2)\,dx / dt + x = B \cos t \qquad (7)$$

Cambiando $y = dx/dt$, la ecuación 7 puede ser reescrita en términos de variables de estado x y y como se muestra en la ecuación 8 y 9 [15].

$$dx / dt = y \qquad (8)$$

$$dy / dt = \mu\,(1 - x^2)\,y - x^3 + B \cos t \qquad (9)$$

Ueda y Akamatsu describieron fenómenos de transición caótica en un oscilador forzado de resistencia negativa, este sistema fue descrito mediante las ecuaciones 10 y 11 [16].

$$dx / dt = y \qquad (10)$$

$$dy / dt = \mu\,(1 - x^2)\,y - x^3 + B \cos \nu t \qquad (11)$$

Dentro de los resultados encontrados por Ueda y Akamatsu, al aplicar excitaciones periódicas para B entre 1.0 a 17 y para ν entre 0.60 a 4.0, el sistema presento características caóticas.

A. Usos y aplicaciones

Los usos y aplicaciones de los sistemas caóticos pueden ser catalogados mediante la TABLE V [12].

TABLE V. USOS Y APLICACIONES DE LOS SISTEMAS CAÓTICOS

Categoría	***Usos y aplicaciones.***
Control	Control del comportamiento irregular de dispositivos y sistemas
Síntesis	Control potencial de la epilepsia, mejoras al tramado de sistemas.
Sincronización	Comunicaciones seguras, radio de canal de banda caótica, encriptación.
Procesamiento de información	Codificación, decodificación y almacenamiento de información en el caos, rendimiento optimizado de redes neuronales, reconocimiento de patrones.
Predicción a corto plazo	Enfermedades contagiosas, clima, economía.

III. PRUEBAS DEL NIST

El Instituto Nacional de Estándares y Tecnología (National Institute of Standards and Technology, NIST por sus siglas en inglés), desarrollo una serie de pruebas para generadores de números aleatorios y pseudoaleatorios para aplicaciones criptográficas. Dicha serie de pruebas, llamadas sp800-22 Test suite, consta de quince pruebas estadísticas mediante las cuales

se ponen a prueba dos hipótesis que determinan si la secuencia de bits es aleatoria o no. Cada prueba se encarga de calcular un valor conocido como *P-value*, el cual representa la probabilidad de que el generador de números sea aleatorio. Este valor debe ser igual a 1 para considerarse como una secuencia perfectamente aleatoria [17].

A. *Prueba de complejidad lineal*

La prueba de complejidad lineal determina si la secuencia pudiera considerarse aleatoria para generadores RNG o pseudoaleatorias en el caso de generadores PRNG. La prueba emplea la ecuación 12 para obtener el valor de la variable *P-value* [18].

$$\text{P-value} = \text{igamc}\,(K / 2, X^2(obs) / 2) \qquad (12)$$

B. *Prueba de excursiones aleatorias variantes*

Esta prueba consiste en obtener una conclusión *P-value* independiente para cada estado. El enfoque es cantidad de veces que ocurre cierto estado en cada caminata aleatoria, con el fin de detectar desviaciones de número esperado de visitas a cada estado. En esta prueba, el valor de *P-value* está dado por la ecuación 13 [18].

$$\text{P-value} = \text{erfc}\,(|\xi\,(x) - J| / \sqrt{2J\,(4|x\ \ |-2)}) \qquad (13)$$

IV. DESARROLLO E IMPLEMENTACIÓN DE LAS ECUACIONES

El sistema de ecuaciones propuesto se basa en las ecuaciones 10 y 11 empleado por Ueda y Akamatsu para su oscilador forzado de resistencia negativa. El sistema ha sido implementado con los valores mostrados en la TABLE VI. Tanto el valor de la constante B y la constante ν se encuentran dentro del rango donde el sistema puede presentar características caóticas.

TABLE VI. VALORES EMPLEADOS EN EL SISTEMA DE ECUACIONES

Valor	***Usos y aplicaciones***
Constante μ	0.05
Constante B	1
Constante ν	1
Valor de y_0	1
Valor de x_0	1

La implementación del sistema ha sido realizada en la herramienta de cómputo numérico MATLAB, empleando el método de Runge-Kutta de cuarto orden para obtener las soluciones del sistema de ecuaciones. El método numérico, también conocido como RK4, es uno de los más empleados de esta familia y se encuentra dado por:

$$k_1 = hf\,(t_n, y_i) \qquad (14)$$
$$k_2 = hf\,(t_n + h / 2,\ y_i + k_1 / 2) \qquad (15)$$
$$k_3 = hf\,(t_n + h / 2,\ y_i + k_2 / 2) \qquad (16)$$
$$k_4 = hf\,(t_n + h,\ y_i + k_3) \qquad (17)$$
$$y_{n+1} = y_n + (k_1 + 2k_2 + 3k_4 + k_4) / 6 \qquad (18)$$

El método RK es efectivo y comúnmente usado para resolver problemas de ecuaciones diferenciales con un valor inicial. Este método puede ser empleado para desarrollar un método numérico altamente preciso mediante las propias funciones sin requerir de derivadas con un orden alto de estas mismas [20]. Este método intenta mejorar el problema del método de Euler, siempre y cuando se escoja un incremento de paso lo suficientemente pequeño para que este alcance una precisión razonable para la resolución del problema. Lo negativo de este método, es que se requiere calcular los incrementos k_i que implica el algoritmo [21]. Aunque Romeo lo menciona como un problema para las hojas de cálculo de Excel, este problema se puede presentar al momento de implementar el algoritmo mediante software como C, Python o MATLAB, o incluso en el caso más débil, en hardware. Siendo este último más delicado al hablar de dispositivos como FPGAs o microcontroladores, donde la cantidad de recursos puede ser más limitada que una computadora. El algoritmo por seguir para obtener la aproximación de la solución del sistema de ecuaciones diferenciales mediante el método RK4 esta descrito en la Fig.1 [22]. Ya que se trata de un método iterativo, es decir que las ecuaciones 14 a 18 son repetidas *n* veces según la cantidad de pasos dados por el incremento.

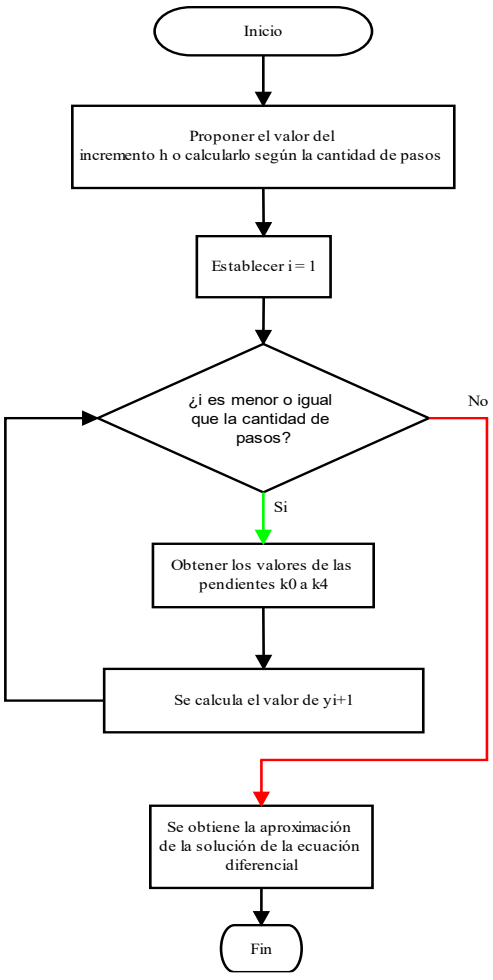


Figure 1. Algoritmo del método RK4

A. *Implementación del método de Runge-Kutta de cuarto orden*

El método RK puede ser extendido a sistemas de ecuación, por lo que la fórmula recursiva de cada paso está dada por las ecuaciones 19 a 28 [23].

$$k_1 = hf(t_n, x_i, y_i) \quad (19)$$

$$m_1 = hg(t_n, x_i, y_i) \quad (20)$$

$$k_2 = hf(t_n + h/2, x_i + k_1/2, y_i + m_1/2) \quad (21)$$

$$m_2 = hg(t_n + h/2, x_i + k_1/2, y_i + m_1/2) \quad (22)$$

$$k_3 = hf(t_n + h/2, x_i + k_2/2, y_i + m_2/2) \quad (23)$$

$$m_3 = hg(t_n + h/2, x_i + k_2/2, y_i + m_2/2) \quad (24)$$

$$k_4 = hf(t_n + h, x_i + k_3, y_i + k_3) \quad (25)$$

$$m_4 = hg(t_n + h, x_i + k_3, y_i + k_3) \quad (26)$$

$$x_{n+1} = x_n + (k_1 + 2k_2 + 3k_4 + k_4)/6 \quad (27)$$

$$y_{n+1} = y_n + (k_1 + 2k_2 + 3k_4 + k_4)/6 \quad (28)$$

Se debe tener cuidado al determinar las pendientes. La manera correcta de implementar el método numérico es desarrollar todas las pendientes para todas las variables en el valor inicial. Estas pendientes, k_i y m_i, se utilizarán para obtener las aproximaciones de las variables dependientes en el punto medio del intervalo hasta llegar al punto final de este. Finalmente, estas pendientes son combinadas en funciones de incremento que sean empleadas para realizar la aproximación final [24]. Para la implementación del método numérico, se empleó un ciclo for, el cual determina su índice final mediante la dimensión del arreglo empleado para almacenar los valores aproximados que serán calculados para las variables del sistema de ecuaciones. La dimensión de este arreglo es determinada por el intervalo de tiempo establecido y el incremento de paso propuesto. El sistema fue probado con intervalos de 0 a 512, 0 a 1024,0 a 2048,0 a 4096 y 8192 segundos, con un incremento de 0.000488281. Estos valores fueron escogidos pensando en llevar el sistema a una implementación en hardware, por lo que se usaron valores dados por las potencias del sistema binario empleando la notación de punto fijo. El ciclo se encargó de obtener los valores estimados de la solución del sistema de ecuaciones caótico y almacenarlos en arreglos. Mediante los intervalos seleccionados, se buscó obtener una cantidad significativa de datos para ser ingresados a la prueba de generación de bits aleatorios del NIST.

B. *Preparación y manejo de datos*

MATLAB cuenta con la función **fi**, la cual permite la creación de objetos de punto fijo, el cual cuenta con una longitud predeterminada de 16 bits. La función permite seleccionar si esta devuelve un objeto con el valor, representación de signo, longitud de palabra y longitud de la fracción [25]. La representación en punto fijo de los valores generados por el sistema de ecuaciones pudiese ser útil para su posterior manejo y almacenamiento en un sistema embebido en términos de espacio. Para realizar la conversión del objeto de punto fijo a binario se ha empleado la función **bin** de MATLAB. Esta función está diseñada para la representación binaria de un objeto de punto fijo sin signo, por lo que, si el objeto de punto binario cuenta con signo negativo, la conversión binaria mostrará el valor absoluto de la cantidad [26]. Al momento de generar el objeto de punto, se está tomando en cuenta todo el arreglo generado con los valores aproximados de la solución del sistema de ecuaciones, para ser convertido a sistema binario. Finalmente, los datos convertidos han sido exportados en archivos de texto, con el objetivo de ser analizados mediante las pruebas estadísticas del NIST.

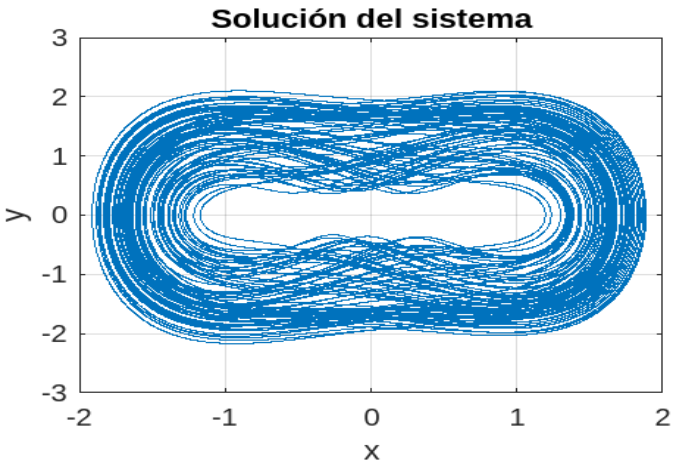
V. Análisis de Resultados

El sistema muestra un comportamiento similar en todas las pruebas, debido a que se mantuvieron los valores iniciales para cada una de las pruebas realizadas. El objetivo fue analizar como era su respuesta al mantener el sistema funcionando por un tiempo prolongado. El método RK4, al tener un incremento de tiempo muy pequeño, permitió obtener una gran cantidad de datos como se muestra en la TABLE VII.

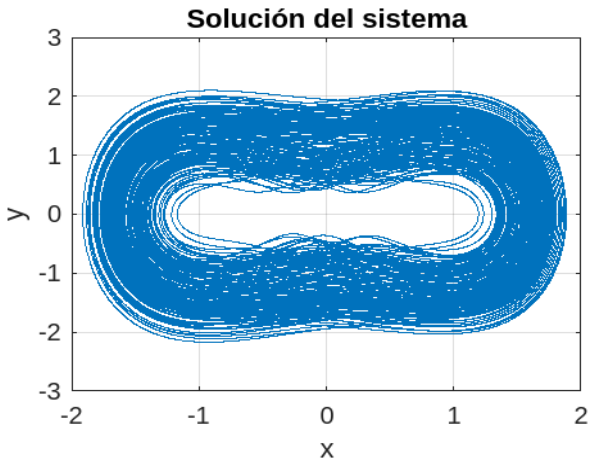
TABLE VII. CANTIDAD DE DATOS OBTENIDA DURANTE CADA SIMULACIÓN

Intervalo de tiempo (s)	*Cantidad de datos obtenida*
0-512	1048576
0-1024	2097153
0-2048	4194306
0-4096	8388612
0-8192	16777224

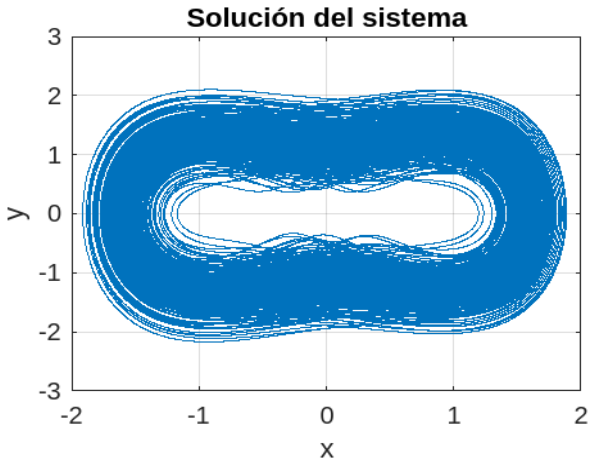
La razón por la que los valores iniciales no fueron cambiados en cada una de las pruebas debido a que se buscaba ampliar la longitud de los datos obtenidos. Graficando los valores aproximados de la solución del sistema, las soluciones del sistema generan varias trayectorias que parecieran tener la misma posición en el plano como se aprecia en la Fig. 2 empleando distintos intervalos y las mismas condiciones iniciales.



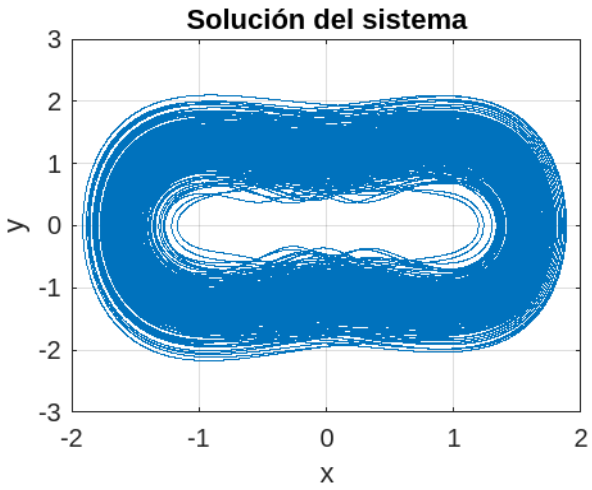
a) Sistema caótico con un intervalo de 0 a 512.



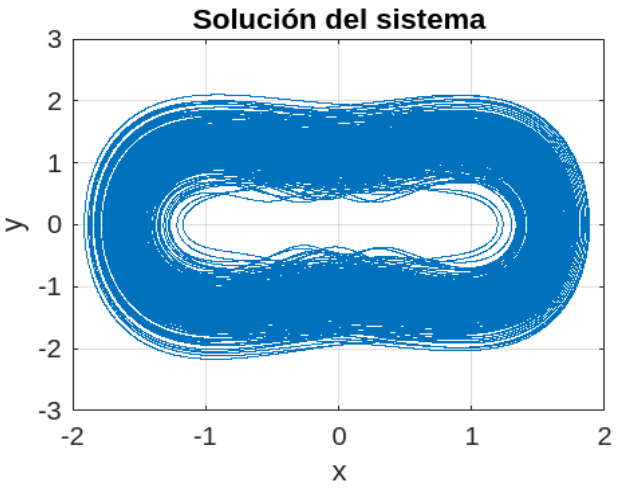
b) Sistema caótico con un intervalo de 0 a 1024.



c) Sistema caótico con un intervalo de 0 a 2048.



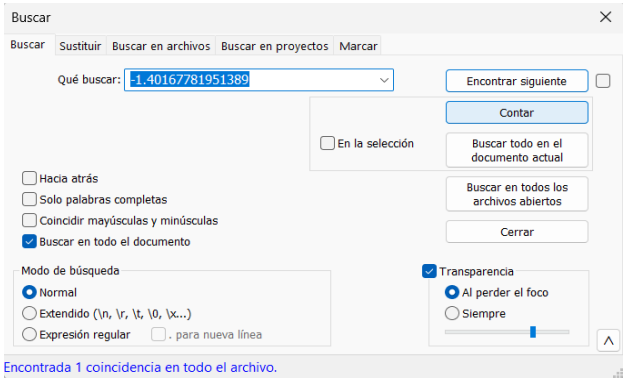
d) Sistema caótico con un intervalo de 0 a 4096.



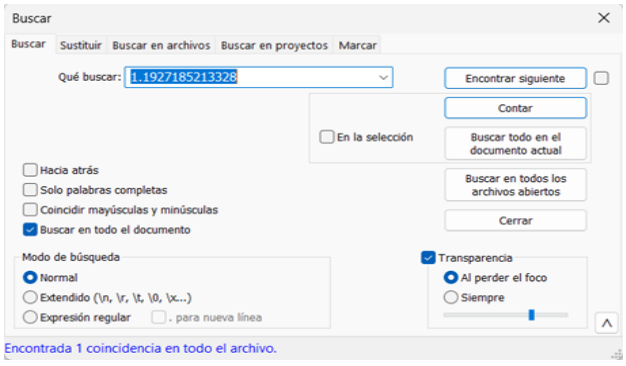
d) Sistema caótico con un intervalo de 0 a 8192.

Figure 2. Solución del sistema en diversos intervalos con las mismas condiciones iniciales, a) intervalo de 0 - 512, b) intervalo de 0 - 1024, c) intervalo de 0 - 2048, d) intervalo de 0 - 4096, c) intervalo de 0 - 8192.

Aunque a simple vista, parece que los valores obtenidos por la solución del sistema pudiesen estar repetidos, estos presentan un ligero distanciamiento entre ellos. Realizando una búsqueda rápida en los valores obtenidos en la solución aproximada del sistema dentro del intervalo de 0 a 8192, no parece existir algún valor repetido como se muestra en la Fig. 3.



a) -1.40167781951389



b) 1.1927185213328

Figure 3. Búsqueda rápida de valores repetidos con los valores a) -1.40167781951389, b) 1.1927185213328.

Para evaluar las secuencias producidas por el generador de números pseudoaleatorios que se presenta en este trabajo, se obtuvieron tres secuencias con diferente longitud que fueron de 33, 50 y 100 millones de bits. Las secuencias se evaluaron por las pruebas estadísticas del NIST, donde, se mostró un valor satisfactorio en la prueba de Complejidad Lineal para los tres casos. Estos resultados demuestran que la secuencia se concluye como compleja y pseudoaleatoria para esta prueba en específico, sin importar la cantidad de bits que sean generados. Pero, provocando variaciones en el valor probabilístico de $P-value$, como se observa en la Fig. 4 que es el resultado para la secuencia de 33 millones de bits, la cual muestra un valor bajo en $P-value$ comparado con los valores que se muestran en la Fig. 5 que corresponde a la secuencia de 50 Mb y la Fig. 6 a la de 100 Mb.

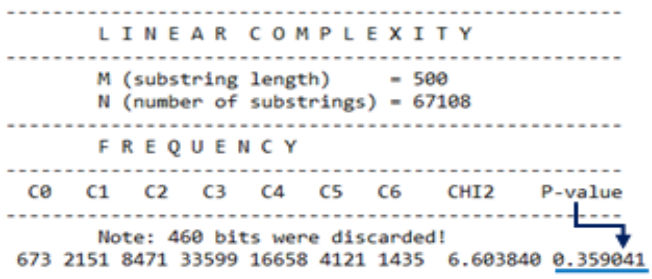


Figure 4. Resultado de la prueba de Complejidad Lineal para la secuencia de 33 millones de bits.

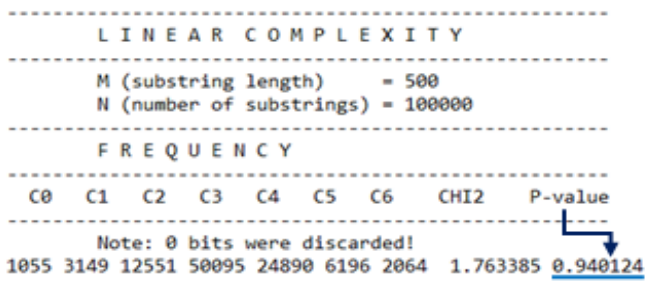


Figure 5. Resultado de la prueba de Complejidad Lineal para la secuencia de 50 millones de bits.

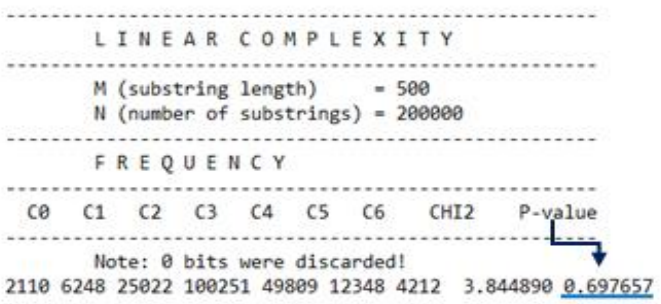


Figure 6. Resultado de la prueba de Complejidad Lineal para la secuencia de 100 millones de bits.

Sin embargo, cuando aumentaron los bits de la secuencia a 50 millones y 100 millones de datos, se aprobaron todos los estados de la prueba Excursiones Aleatorias Variante, que se conforma de un total de 18 estados del -9 al +9. Esto se debe al aumento del número de ciclos para que la prueba se ejecute, es decir, se deben cumplir con tener mínimo 500 ciclos para que la prueba continue realizándose, de lo contrario se detiene y arroja un mensaje que indica insuficiencia de ciclos [27], [17]. En la TABLE VIII se muestran algunos valores en binario que fueron obtenidos del sistema caótico. Diez de estos datos tienen mayor cantidad de ceros que limitan el número de ciclos que se generan en la prueba de Excursiones Aleatorias Variante y diez mantienen mayor cantidad de unos para incrementar el número de ciclos. Con lo anterior, se evita tener una insuficiencia o un exceso de estos que no permita que la prueba se realice.

TABLE VIII. VALORES EN BINARIOS CON MAYOR CANTIDAD DE CEROS Y MAYOR CANTIDAD DE UNOS OBTENIDOS DEL SISTEMA CAÓTICO.

Valores cargados a 0	*Valores cargados a 1*
0000000000000000101101000011 01001	1111111111111111101101101001 101
0000000000000000101000000101 11000	1111111111111111011001110010 001
0000000000000000100011001000 00110	1111111111111111000101111010 100
0000000000000000011110001010 10011	1111111111111111010010000010 110
0000000000000000011001001100 11110	1111111111111111001111000101 0111
0000000000000000010100001111 01001	1111111111111111000101001001 0111
0000000000000000001111010001 10010	1111111111111111011101100110 1011
0000000000000000001010010011 11010	1111111111111111011000101000 1011
0000000000000000000101010110 00001	1111111111111111010011101010 1011
0000000000000000000000011000 00111	1111111111111111001110101100 0111

Por lo tanto, para la secuencia con 50 Mb se obtuvieron 520 ciclos, pero al aumentar los bits a 100 Mb, los ciclos se aumentaron al doble con un total de 1206. En ambos casos la prueba se ejecutó satisfactoriamente. Por cada estado que conforman la prueba, se determinó una conclusión independiente de pseudoaleatoriedad. Donde el valor $P-value$ debe ser mayor a 0.01, como se muestran en la TABLE IX. Estos también fueron graficados como se observa en la Fig. 7 para ver las variaciones en los resultados de $P-value$ por cada estado en ambas secuencias. La cuales mostraron valores cercanos a 1 en los estados -7, -6, 4, 5, 6, 7 y 9 para $P-value1$, en cambio, para $P-value2$ los estados -7, -1 y 9 se mantiene cercanos a 1 que es un valor que concluye las secuencias de los generadores como perfectamente pseudoaleatorias. En este caso se concluye la secuencia como pseudoaleatoria.

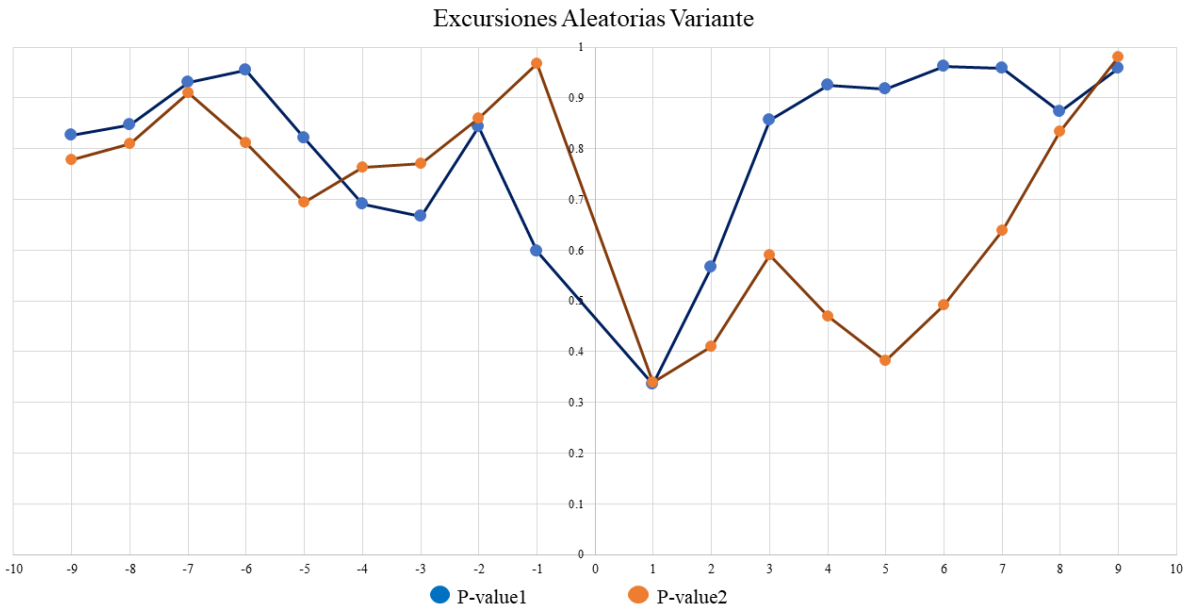


Figure 7. Resultados de P-value de la prueba excursiones aleatorias variante para la secuencia de 50 millones de datos P-value1 y 100 millones P-value2

TABLE IX. RESULTADOS DE P-VALUE DE LA PRUEBA EXCURSIONES ALEATORIAS VARIANTE PROPUESTA POR EL NIST, PARA UNA SECUENCIA DE 50 Y 100 MILLONES DE BITS.

	50 millones	***100 millones***
Estado	***P-value1***	***P-value2***
-9	0.827351	0.778335
-8	0.847622	0.808906
-7	0.931464	0.910074
-6	0.955265	0.810772
-5	0.820115	0.693834
-4	0.690272	0.764069
-3	0.667274	0.770752
-2	0.843881	0.860030
-1	0.598091	0.967517
1	0.336417	0.338570
2	0.566718	0.410563
3	0.856935	0.591093
4	0.925299	0.469422
5	0.917675	0.381276
6	0.962715	0.491708
7	0.958846	0.639266
8	0.872780	0.833439
9	0.958015	0.980301

VI. CONCLUSIONES

Siguiendo la metodología de las pruebas estadísticas del NIST fue posible validar que el sistema caótico basado en la modificación de la ecuación de Van der Pol presenta características pseudoaleatorias en los valores aproximados de su solución obtenidos mediante el método RK4, tras pasar dos de las quince pruebas estadísticas. A pesar de que esto signifique un buen inicio para esta propuesta, los resultados obtenidos por las pruebas muestran que aún hay varios puntos por trabajar con la finalidad de que los valores generados por el sistema puedan considerarse aún más pseudoaleatorios. Esto con la finalidad de ser llevado a hardware y seguir validando que los datos generados cumplan con estas características.

AGRADECIMIENTOS

El presente trabajo ha sido apoyado por el Consejo Nacional de Humanidades, Ciencia y Tecnología CONAHCYT y por la Universidad de Guadalajara.

REFERENCIAS BIBLIOGRÁFICA

[1] N. Ferguson and B. Schneier, Practical Cryptography. John Wiley & Sons, 2003.

[2] V. Grozov, M. Budko, A. Guirik, and M. Budko, “Efficiency comparison of pseudorandom number generators based on strong cryptographic algorithms,” 2018 10th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), Nov. 2018, doi: 10.1109/icumt.2018.8631223.

[3] L. Min, K. Hu, L. Zhang, and Y. Zhang, “Study on pseudorandomness of some pseudorandom number generators with application,” Proceedings of the 2013 Ninth International Conference on Computational Intelligence and Security, Dec. 2013, doi: 10.1109/cis.2013.126.

[4] E. A. Luengo, “A brief and understandable guide to pseudo-random number generators and specific models for security,” Statistics Surveys, vol. 16, no. none, Jan. 2022, doi: 10.1214/22-ss136.

[5] H. Ali-Pacha, N. Hadj-Said, A. Ali-Pacha, M. A. Mohamed, and M. Mamat, “Cryptographic adaptation of the middle square generator,” International Journal of Electrical and Computer Engineering, vol. 9, no. 6, p. 5615, Dec. 2019, doi: 10.11591/ijece.v9i6.pp5615-5627.

[6] S. Tezuka, “Linear Congruential Generators,” in Springer eBooks, 1995, pp. 57–82. doi: 10.1007/978-1-4615-2317-8_3.

[7] C.-C. Li and B. Sun, “Using linear congruential generators for cryptographic purposes.,” Computers and Their Applications, pp. 13–19, Jan. 2005, [Online]. Available: http://www.itk.ilstu.edu/faculty/chungli/mypapers/CATA2005.pdf

[8] H. Krawczyk, “How to predict congruential generators,” Journal of Algorithms, vol. 13, no. 4, pp. 527–545, Dec. 1992, doi: 10.1016/0196-6774(92)90054-g.

[9] S. Aluru, “Parallel additive lagged Fibonacci random number generators,” ICS ’96: Proceedings of the 10th International Conference on Supercomputing, Jan. 1996, doi: 10.1145/237578.237591.

[10] A. Ahmad, A. Al-Mashari, and A. M. J. Al-Lawati, “On locking conditions in m-sequence generators for the use in digital watermarking,” 2009 Proceeding of International Conference on Methods and Models in Computer Science (ICM2CS), Dec. 2009, doi: 10.1109/icm2cs.2009.5397982.

[11] Kumar AP, Rajput P, Shukla B. FPGA Implementation of 8, 16 and 32 Bit LFSR with Maximum Length Feedback Polynomial using VHDL. 2012 International Conference on Communication Systems and Network Technologies, Rajkot, Gujarat, India, 2012.

[12] W. Ditto and T. Munakata, “Principles and applications of chaotic systems,” Communications of the ACM, vol. 38, no. 11, pp. 96–102, Nov. 1995, doi: 10.1145/219717.219797.

[13] J. C. Sprott, “Simple chaotic systems and circuits,” American Journal of Physics, vol. 68, no. 8, pp. 758–763, Aug. 2000, doi: 10.1119/1.19538.

[14] W. D. M. Mendoza, A. R. A. Orellana, and L. B. R. Leal, “UNA REVISIÓN DE LA ECUACIÓN DE VAN DER POL y SUS MODIFICACIONES,” Revista Bases De La Ciencia, vol. 7, no. ESPECIAL, pp. 213–226, Dec. 2022, doi: 10.33936/revbasdelaciencia.v7iespecial.4743.

[15] G. A. P. Lobato, “Oscilador de Van Der Pol,” Unam1, Jan. 2018, [Online]. Available: https://www.academia.edu/35768562/Oscilador_de_Van_Der_Pol

[16] Y. Ueda and N. Akamatsu, “Chaotically transitional phenomena in the forced negative-resistance oscillator,” IEEE Transactions on Circuits and Systems, vol. 28, no. 3, pp. 217–224, Mar. 1981, doi: 10.1109/tcs.1981.1084975.

[17] Xxxxx xxxxxxxxxx xxxxxxxxxxxxxxx xxxxxxxxxxxxxx xxxxxxxxxxxxxx xxxxxxxxxxxxx xxxxxxxxxxxxxx xxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxx xxxxx xxxxxxxxxxxxxx xxxxxxxxxxxxx xxxxxxxxxxxxx xxxxxxxxxxxxxx xxxxxxxxxxxxxx xxxxxxxxxxxxxx xxxxxxxxxxxxxx xxxxxxxxxxxxxx xxxxxxxxxxxxxx xxxxxxxxxxxxxx xxxxxxxxxxxxxx xxxxxxxxxxxxxx

[18] Xxxx xxxxxxxxxx xxxxxxxxxxxxx xxxxxxxxxxx xxxxxxxxxxx xxxxxxxx xxxxxxxxxxxx xxxxxxxxxxxxxx xxxxxxxxxxxxx xxxxxxxx xxxxxxxxxxxxxxx xxxxxxxxxxxxxx xxxxxxxxxx xxxxxxxxxx xxxxxxx xxxxxxxxxxxxx xxxxxxxxxxxxx xxxxxxxxxxxxxx xxxxxxxx xxxxxxxxxx

[19] G. R. Lindfield and J. E. T. Penny, “Solution of differential equations,” in Elsevier eBooks, 2012, pp. 233–281. doi: 10.1016/b978-0-12-386942-5.00005-9.

[20] L. Zheng and X. Zhang, “Numerical methods,” in Elsevier eBooks, 2017, pp. 361–455. doi: 10.1016/b978-0-12-811753-8.00008-6.

[21] G. Romeo, “Mathematics for dynamic economic models,” in Elsevier eBooks, 2020, pp. 139–215. doi: 10.1016/b978-0-12-817648-1.00004-9.

[22] A. N. Hurtado and F. C. D. Sánchez, “Ecuaciones diferenciales ordinarias: Métodos de Runge-Kutta,” in Métodos numéricos aplicados a la ingeniería, 4th ed., 2012.

[23] M. L. Abell and J. P. Braselton, “First-Order equations,” in Elsevier eBooks, 2014, pp. 27–87. doi: 10.1016/b978-0-12-417219-7.00002-8.

[24] Chapra, “Problemas con valores iniciales: Método clásico de Runge-Kutta de cuarto orden,” in Métodos numéricos aplicados con MATLAB para ingenieros y científicos, 5th ed., McGraw Hill, 2023.

[25] “Crear un objeto numérico de punto fijo - MATLAB - MathWorks América Latina.” https://la.mathworks.com/help/fixedpoint/ref/embedded.fi.html

[26] “Representación binaria sin signo del entero almacenado del objeto fi - MATLAB bin - MathWorks América Latina.” https://la.mathworks.com/help/fixedpoint/ref/embedded.fi.bin.html

[27] “NIST Technical Series Publications.” [Online]. Available: https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-22r1a.pdf.