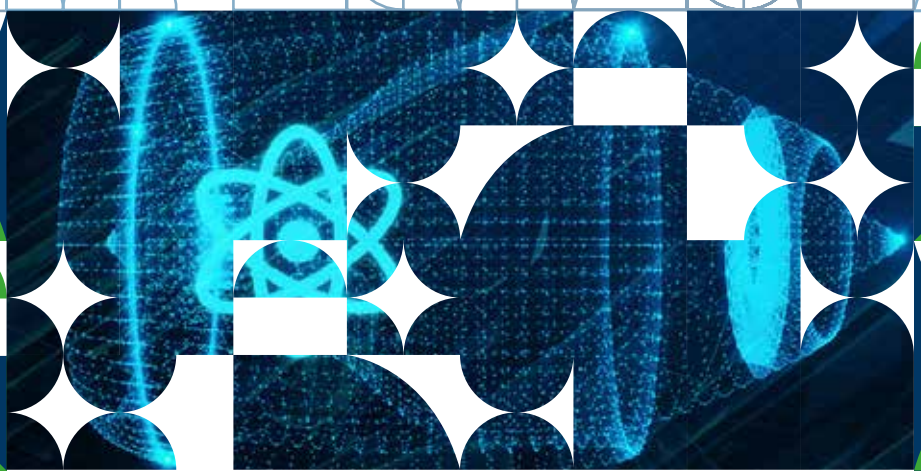

ESTUDIO DEL COMPORTAMIENTO DINÁMICO DE LOS CIRCUITOS MEMRISTIVOS Y SU SINCRONIZACIÓN A TRAVÉS DE LA SIMULACIÓN NUMÉRICA USANDO PYTHON

STUDY OF THE DYNAMIC BEHAVIOR OF MEMRISTIVE CIRCUITS AND THEIR SYNCHRONIZATION THROUGH NUMERICAL SIMULATION USING PYTHON

ESTUDIO DEL COMPORTAMIENTO DINÁMICO DE LOS CIRCUITOS MEMRISTIVOS Y SU SINCRONIZACIÓN A TRAVÉS DE LA SIMULACIÓN NUMÉRICA USANDO PYTHON

STUDY OF THE DYNAMIC BEHAVIOR OF MEMRISTIVE CIRCUITS AND THEIR SYNCHRONIZATION THROUGH NUMERICAL SIMULATION USING PYTHON

Ilbay - Paca, Juan[1];
Rentería - Bustamante, Leonardo[2]

[1]Universidad Nacional de Chimborazo, Ecuador, joilbay.felc@unach.edu.ec
[2]Universidad Nacional de Chimborazo, Ecuador, leonardo.renteria@unah.edu.ec

RESUMEN

Este estudio investiga el comportamiento dinámico y las propiedades de sincronización de los circuitos memristivos mediante simulaciones numéricas utilizando Python. Los circuitos memristivos, conocidos por sus características no lineales y dependientes de la memoria, son de gran interés para aplicaciones en computación neuromórfica y comunicaciones seguras. La investigación empleó modelos matemáticos de memristores y utilizó las bibliotecas computacionales de Python para simular la dinámica de los circuitos y analizar los mecanismos de sincronización en sistemas acoplados. La metodología incluyó la construcción de modelos basados en ecuaciones diferenciales, su resolución numérica mediante el método de Runge-Kutta y la visualización de los resultados. Se observaron y analizaron comportamientos clave, como oscilaciones periódicas y estados caóticos, y se estudiaron las propiedades de sincronización simulando circuitos acoplados. Se exploraron tres modelos representativos: Chua-Stanford, Memductor y un modelo experimental basado en Chua. Los resultados indican que los memristores inducen un comportamiento de histéresis que amplifica la complejidad dinámica y facilita la sincronización de sistemas inicialmente no sincronizados a través de un factor de acoplamiento. Con el tiempo, los circuitos evolucionan hacia un comportamiento coherente, demostrando cómo el caos puede ser controlado y sincronizado. Finalmente, este estudio demuestra la utilidad de las simulaciones basadas en Python para avanzar en la comprensión del comportamiento de los circuitos memristivos y sus posibles aplicaciones en sistemas electrónicos y computacionales.

Palabras clave: Circuitos de Chua, Memristores, Python, Sincronización, Histéresis, Memductor.

ABSTRACT

This study investigates the dynamic behavior and synchronization properties of memristive circuits through numerical simulations using Python. Memristive circuits, known for their non-linear and memory-dependent characteristics, are of significant interest for applications in neuromorphic computing and secure communications. The research employed mathematical models of memristors and used Python's computational libraries to simulate circuit dynamics and analyze synchronization mechanisms in coupled systems. The methodology involved constructing differential equation-based models, solving them numerically using the Runge-Kutta method, and visualizing the results. Key behaviors such as periodic oscillations, and chaotic states were observed and analyzed, and synchronization properties were studied by simulating coupled circuits. Three representative models are explored: Chua-Stanford, Memductor, and an experimental Chua-based model. The results indicate that memristors induce hysteresis behavior, which amplifies dynamic complexity and facilitates synchronization of initially unsynchronized systems through a coupling factor. Over time, the circuits evolve towards coherent behavior, demonstrating how chaos can be controlled and synchronized. Finally, this study demonstrates the utility of Python-based simulations in advancing the understanding of memristive circuit behavior and their potential applications in electronic and computational systems.

Keywords: *Chua circuits, Memristors, Synchronization, Hysteresis, Memductor, Python.*

Recibido: Agosto 2024 Aceptado: Diciembre 2024
Received: August 2024 Accepted: December 2024

1. INTRODUCCIÓN

Los circuitos memristivos han surgido como un área crucial de estudio dentro de la electrónica moderna debido a sus propiedades únicas y aplicaciones potenciales en campos como la computación neuromórfica, el almacenamiento de memoria y la computación analógica. El memristor, conceptualizado originalmente por Leon Chua en 1971, es un elemento de circuito pasivo no lineal que vincula la carga eléctrica y el flujo magnético [1], Fig.1.

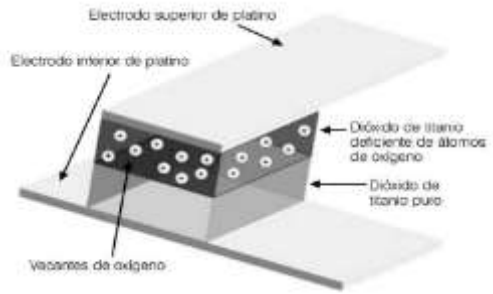

Fig. 1: Estructura del memristor ***[2].***

A diferencia de los elementos de un circuito tradicionales (resistencias, condensadores e inductores), los memristores exhiben un comportamiento dependiente de la memoria, lo que les permite "recordar" sus estados anteriores cuando se corta la energía. Esta característica los hace particularmente prometedores para crear circuitos que pueden imitar redes neuronales biológicas y sistemas complejos [3].
A este comportamiento se lo denomina histéresis Fig.2, presenta una forma de lazo cerrado y muestra la relación entre el voltaje aplicado (eje x) y la corriente resultante (eje y). A medida que el voltaje cambia de dirección, la curva no sigue la misma trayectoria evidenciando que la resistencia cambia en función de la carga acumulada, y la corriente no retorna a su posición original al invertir el voltaje [1].

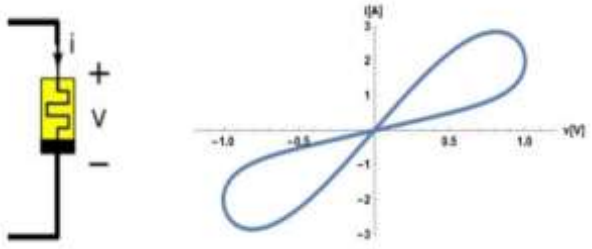

Fig. 2: Histéresis clásica del memristor

El comportamiento dinámico de los circuitos memristivos involucra interacciones intrincadas influenciadas por sus propiedades no lineales, lo que resulta en dinámicas ricas y complejas [4]. Estas dinámicas pueden exhibir comportamientos como oscilaciones, bifurcaciones y respuestas caóticas, lo que hace que su estudio sea crucial para avanzar en el conocimiento de los fenómenos de sincronización y el diseño de sistemas electrónicos robustos. Para crear un circuito memristivo caótico, basta con reemplazar el diodo de Chua en un circuito de Chua por un memristor, Fig. 3.

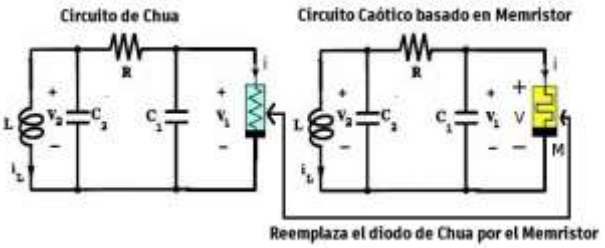

Fig. 3: Circuito de Chua con memristor ***[5].***

Por otro lado, la sincronización, es un fenómeno en el que dos o más sistemas alinean su dinámica a través del acoplamiento o la interacción. Esta desempeña un papel esencial en campos que van desde las comunicaciones seguras hasta la sincronización neuronal en la actividad cerebral [6]. La sincronización de circuitos caóticos memristivos o no, se da lugar cuando se interconecta dos o más circuitos idénticos a través de una resistencia de acoplamiento Fig.4.

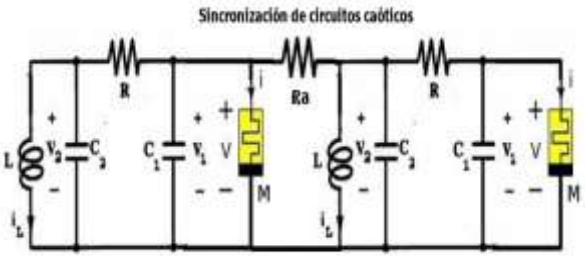

Fig. 4: Circuitos caóticos sincronizados

Dado el creciente interés en las aplicaciones de circuitos memristivos para comprender mejor su comportamiento, el objetivo de este artículo es examinar la dinámica de los circuitos memristivos, identificar los mecanismos de sincronización y demostrar cómo las simulaciones numéricas pueden ofrecer información sobre su funcionalidad. En este

estudio se utilizó el lenguaje de programación Python que con sus bibliotecas versátiles como NumPy, SciPy y Matplotlib, proporciona una plataforma poderosa para modelar, analizar y visualizar sistemas complejos [7].

2. TRABAJOS RELACIONADOS

La exploración de circuitos memristivos ha ganado considerable atención en los últimos años debido a sus potenciales aplicaciones en áreas como la ingeniería neuromórfica, la comunicación basada en el caos y el modelado de sistemas complejos. Varios investigadores han profundizado en las propiedades dinámicas y los comportamientos de sincronización de estos circuitos, contribuyendo al creciente cuerpo de conocimientos.

Ali et al. [8] realizaron un estudio exhaustivo sobre el modelado basado en memristores para aplicaciones neuromórficas. Esta investigación enfatizó la capacidad del memristor para emular la plasticidad sináptica, una característica esencial para desarrollar redes neuronales artificiales energéticamente eficientes. Los autores utilizaron técnicas de simulación para demostrar cómo se podían integrar elementos memristivos en sistemas neuromórficos, destacando su potencial para reducir el consumo de energía y mejorar la adaptabilidad en las implementaciones de hardware. Este trabajo sentó las bases de cómo se podían aprovechar los memristores para crear sistemas computacionales inspirados en el cerebro.

Batista et al. [9] se centró en la sincronización de circuitos memristivos caóticos, empleando simulaciones numéricas para investigar cómo los mecanismos de acoplamiento podrían alinear el comportamiento de los componentes individuales. El estudio reveló que estrategias de acoplamiento específicas, como la retroalimentación lineal y no lineal, fueron efectivas para lograr la sincronización entre sistemas caóticos. Esta investigación proporcionó información crítica para diseñar circuitos con estabilidad y rendimiento mejorados, particularmente relevante para aplicaciones donde se necesita una actividad coordinada entre circuitos, como en el procesamiento distribuido y la transmisión segura de datos.

Bao et al. [10] ampliaron la comprensión del comportamiento dinámico de los circuitos memristivos mediante el análisis de sus propiedades caóticas. Los investigadores utilizaron simulaciones numéricas para explorar la influencia de las condiciones iniciales variables y los cambios de parámetros en el comportamiento del circuito. El estudio demostró que los circuitos memristivos podrían exhibir patrones caóticos complejos, que podrían aprovecharse para aplicaciones prácticas como la generación de números aleatorios, algoritmos de cifrado y sistemas de comunicación seguros. Esta investigación subrayó la importancia de comprender y controlar el comportamiento caótico para aprovechar las capacidades únicas de los circuitos memristivos.

Muthuswamy y Chua [5] contribuyeron con una exploración fundamental de la dinámica no lineal presente en circuitos basados en memristores, particularmente sistemas oscilatorios. Su investigación ilustró cómo estos circuitos podrían exhibir fenómenos como bifurcaciones y caos, que son características de los sistemas dinámicos complejos. Al simular estos comportamientos, el estudio proporcionó información crucial sobre las condiciones bajo las cuales los circuitos memristivos podrían pasar de estados oscilatorios estables a regímenes caóticos. Esta comprensión es esencial para diseñar circuitos que puedan evitar o aprovechar el comportamiento caótico, según la aplicación prevista.

Estudios adicionales han explorado las aplicaciones y el control de la dinámica de los memristores en contextos más amplios. Por ejemplo, los investigadores han demostrado cómo se pueden incorporar los memristores al hardware para tareas que requieren aprendizaje adaptativo y procesamiento en tiempo real. Se ha demostrado que la naturaleza dependiente de la memoria del memristor facilita el comportamiento dependiente del estado, lo que lo hace adecuado para aplicaciones en filtros adaptativos y almacenamiento de memoria no volátil.

En conjunto, estos estudios subrayan la naturaleza multifacética de la investigación de circuitos memristivos, desde sus propiedades fundamentales hasta sus aplicaciones prácticas. Los hallazgos destacan que la sincronización y el análisis del comportamiento dinámico son áreas clave de enfoque que continúan inspirando más investigaciones.
Estos trabajos proporcionan una base sólida para el presente estudio que busca ampliar la comprensión del comportamiento y la sincronización de circuitos memristivos a través de simulaciones numéricas

avanzadas utilizando Python. Este enfoque permite una exploración detallada de la dinámica de circuitos, ofreciendo conocimientos prácticos para el diseño y desarrollo de sistemas electrónicos y computacionales innovadores.

3. MATERIALES Y MÉTODOS

Este trabajo tuvo como objetivo explorar el comportamiento dinámico y las propiedades de sincronización de circuitos memristivos a través de simulaciones numéricas utilizando Python. La metodología abarca procedimientos detallados, materiales y un diseño experimental para garantizar la reproducibilidad de los resultados.

3.1 Procedimiento

El estudio se llevó a cabo en una serie de pasos estructurados, como se describe a continuación:

Modelado de circuitos: Los modelos de circuitos memristivos se construyeron en función de representaciones matemáticas previamente validadas de memristores, específicamente utilizando las ecuaciones diferenciales no lineales que describen su relación voltaje-corriente, las cuales se introducen en el circuito de Leon Chua para estudiar cómo afectan las propiedades caóticas.

- **Circuito de Chua**

El circuito de Chua es un sistema no lineal que genera un comportamiento caótico, y su dinámica está gobernada por un conjunto de ecuaciones diferenciales adimensionales, que describen el voltaje del primer condensador x, del segundo condensador y y la corriente z del inductor, como se muestra a continuación:

$$\begin{aligned} \frac{dx}{dt} &= \propto \left(y - x - f(x)\right) \\ \frac{dy}{dt} &= x - y + z \\ \frac{dz}{dt} &= -\beta y \end{aligned} \qquad (1)$$

Además de la función no lineal $f(x)$

$$f(x) = b.x + 0.5.(a-b).(|x+1| - |x-1|) \qquad (2)$$

Donde $\alpha, \beta, a\ y\ b$ son constantes que dependen de los valores específicos del circuito.

Las ecuaciones descritas capturan cómo los voltajes y corrientes en el circuito evolucionan con el tiempo, y la presencia de no linealidades hace que el sistema muestre un comportamiento caótico para ciertos valores de los parámetros [5].

Configuración de simulación numérica: Se eligió Python como el software principal debido a sus amplias bibliotecas para el cálculo numérico y la visualización, así como por su naturaleza libre y abierta. Se emplearon bibliotecas como NumPy, SciPy y Matplotlib para realizar cálculos, resolver ecuaciones diferenciales y representar gráficamente los resultados.
Configuración de parámetros iniciales: Las condiciones iniciales y los valores de los parámetros (por ejemplo, rango de resistencia, estado inicial de memristividad y voltaje de entrada) se definieron cuidadosamente en función de la literatura para reflejar escenarios típicos del mundo real.
Ejecución de la simulación: Las ecuaciones diferenciales que describen los circuitos memristivos se resolvieron utilizando métodos de integración numérica, como el método Runge-Kutta, implementado a través de la función odeint en SciPy. Esto permitió la simulación del comportamiento del circuito durante un período de tiempo determinado.
Análisis de sincronización: Se simularon circuitos acoplados para evaluar las propiedades de sincronización. Se analizó el bloqueo de fase y frecuencia entre circuitos mediante la representación gráfica de la evolución temporal de los estados memristivos.
Registro y análisis de datos: Se registraron y analizaron los datos de salida para identificar comportamientos como oscilaciones periódicas y estados caóticos. Se generaron gráficos y métricas numéricas para visualización e interpretación.

3.2 Materiales

El estudio utilizó los siguientes materiales:
Software: Python (versión 3.9+): para codificar y ejecutar simulaciones.
Bibliotecas: NumPy (para operaciones numéricas), SciPy (para solucionadores de ecuaciones diferenciales), Matplotlib (para visualización de datos) y Seaborn (para mejorar los gráficos).
Hardware: Una estación de trabajo estándar equipada con un procesador Intel i7, 16 GB de RAM y un SSD de 1 TB para el almacenamiento y procesamiento de datos.

3.3 Diseño experimental

El diseño experimental incluyó los siguientes componentes:

Modelo de simulación: El estudio involucró modelos de circuito único y de circuitos acoplados. Se llevaron a cabo simulaciones de circuito único para comprender el comportamiento independiente, mientras que se utilizaron modelos acoplados para investigar las propiedades de sincronización.

Selección de parámetros: Los principales parámetros en el experimento incluyeron la forma de onda del voltaje de entrada (entrada sinusoidal o de paso periódico), las condiciones iniciales de memristancia y la fuerza de acoplamiento entre circuitos en modelos sincronizados.

Métricas de salida: Las métricas principales incluyeron histéresis del memristor, la evolución de la memristancia, voltajes y corrientes a lo largo del tiempo y la sincronización, según el tipo de análisis. El diseño experimental se estructuró para replicar las condiciones del mundo real lo más fielmente posible, manteniendo al mismo tiempo un entorno controlado para observar respuestas específicas de los circuitos memristantes.

Siguiendo el procedimiento descrito, utilizando los materiales especificados y adhiriendo al diseño experimental, este estudio tuvo como objetivo analizar y documentar exhaustivamente el comportamiento dinámico y los mecanismos de sincronización de los circuitos memristivos utilizando Python.

4. RESULTADOS Y DISCUSIÓN

En esta sección se muestran los modelos que se derivan de la aplicación de las leyes de Kirchhoff al circuito original de Chua, reemplazando el diodo no lineal por un memristor. Este cambio introduce ecuaciones diferenciales que describen cómo evolucionan los voltajes y corrientes, incorporando la dinámica no lineal del memristor. Las ecuaciones incluyen términos de acoplamiento dinámico para estudiar la sincronización entre sistemas inicialmente desincronizados, manteniendo las propiedades caóticas del diseño original. A continuación, se presentan los resultados obtenidos a partir de tres modelos diferentes: Stanford, Memristico-Chua y Memductor.

Modelo de Stanford

Representa la dinámica de memristores basada en la evolución del gap físico en un filamento, con una dependencia no lineal del voltaje y parámetros térmicos, su histéresis, Fig.5, muestra una transición suave en la relación corriente-voltaje.

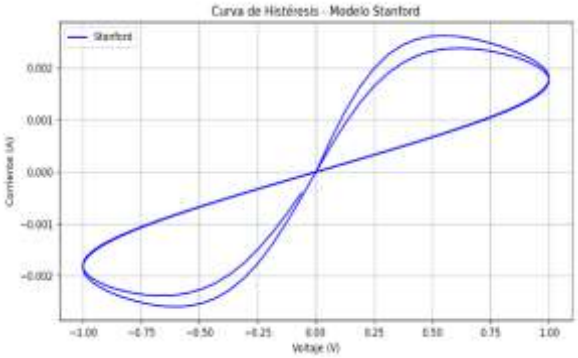

Fig. 5: Histéresis del memristor - modelo de Stanford. R_on= 100, R_off= 1600, ∝= 0.03 y β= 0.9 v(t)= Asin (2π.f.t), A= 1, f= 1Hz.

El sistema de ecuaciones que describe el modelo de Stanford es el siguiente [11]:

$$\begin{aligned} \frac{dx}{dt} &= \propto \left(y - x - f(x)\right) \\ \frac{dy}{dt} &= x - y + z \\ \frac{dz}{dt} &= -\beta y \\ \frac{dg}{dt} &= -e\left(-\frac{q.E_a.g}{k.T_0}\right).e\left(-\frac{q.a_0.\gamma_0}{l.k.T_0}.x\right) \end{aligned} \qquad (3)$$

$$f(x) = b.x + 0.5.(a-b).(|x+1| - |x-1|) \qquad (4)$$

El atractor caótico del modelo presenta una estructura compleja Fig.6 y refleja la interacción entre el voltaje aplicado y el gap físico del memristor. A medida que varía el campo eléctrico, el atractor muestra una trayectoria no lineal que oscila de manera irregular, con un comportamiento impredecible y una dependencia marcada de las condiciones iniciales: $[x, y, z, g] = [0.1, 0, 0, 0.5e-9]$.

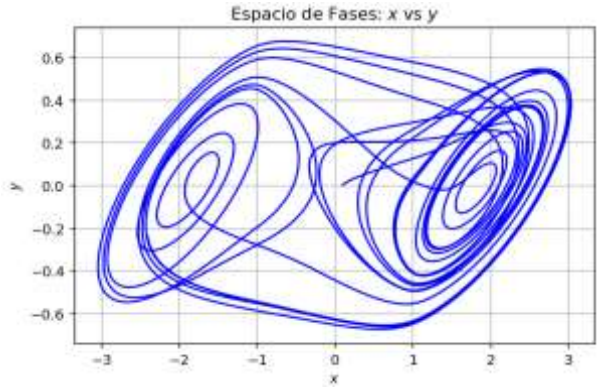

Fig. 6: Atractor caótico del modelo Chua-Stanford. $\propto= 10$, $\beta = 15$, $q = 1.6e^{-19}$, $T_0 = 300$, $E_\propto = 0.8$, $a_0 = 1e^{-10}$, $l = 10e^{-9}$, $\gamma_0 = 2.5$, $k = 1.38e^{-23}$, $g_1 = 0.5e^{-9}$, $g_2 = 6e^{-9}$, $a = -1.27$, $b = -0.68$.

Para corroborar el comportamiento caótico de la sincronización del modelo de Stanford, se plantean diferentes condiciones iniciales para dos circuitos idénticos conectados entre sí, Fig.7, asegurando que comiencen desde estados distintos y, mediante un acoplamiento adecuado, logren sincronizarse con el tiempo.

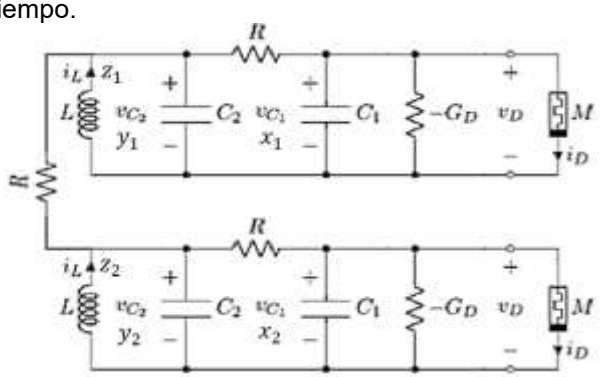

Fig. 7: Sincronización de dos circuitos – modelo de Stanford

Las condiciones iniciales son:

$$[x_1, y_1, z_1, g_1] = [0.1, -0.1, 0.1, -0.1]$$

$$[x_2, y_2, z_2, g_2] = [-0.5, 0.2, -0.2, 0.1]$$

Y las ecuaciones para la sincronización de sistemas caóticos son:

Sistema 1

$$\begin{aligned} \frac{dx_1}{dt} &= \propto (y_1 - x_1 - f(x_1)) \\ \frac{dy_1}{dt} &= x_1 - y_1 + z_1 \\ \frac{dz_1}{dt} &= -\beta y_1 \\ \frac{dg_1}{dt} &= -e\left(-\frac{q.E_a.dg_1}{k.T_0}\right).e\left(-\frac{q.a_0.\gamma_0}{l.k.T_0}.x_1\right) \end{aligned} \quad (5)$$

Sistema 2

$$\begin{aligned} \frac{dx_2}{dt} &= \propto (y_2 - x_2 - f(x_2)) \\ \frac{dy_2}{dt} &= x_2 - y_2 + z_2 \\ \frac{dz_2}{dt} &= -\beta y_2 + K(z_1 - z_2) \\ \frac{dg_2}{dt} &= -e\left(-\frac{q.E_a.dg_2}{k.T_0}\right).e\left(-\frac{q.a_0.\gamma_0}{l.k.T_0}.x_2\right) \end{aligned} \quad (6)$$

Para la sincronización, se emplea un factor de acoplamiento basado en la resistencia y ajustando la evolución del gap físico, se observa cómo ambos sistemas, aunque inicialmente diferentes, logran sincronizarse y seguir trayectorias comunes Fig.8.

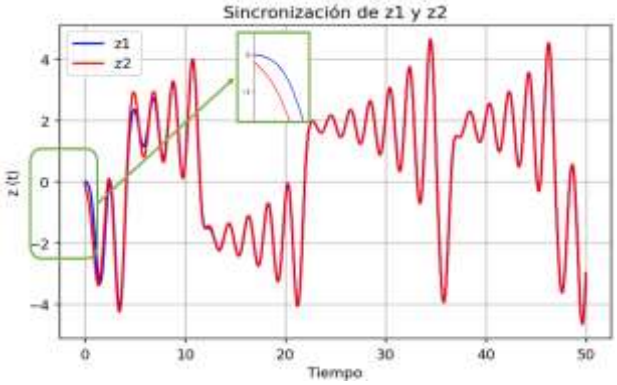

Fig. 8: Señales sincronizadas partiendo de condiciones iniciales distintas, modelo de Stanford con factor de acoplamiento K=1.1.

Modelo memristivo de Chua

Describe sistemas electrónicos no lineales con memoria y caos, destacando una característica función que introduce múltiples estados de equilibrio

y trayectorias complejas. Es el modelo conceptual base para los memristores, su histéresis Fig.9, refleja una dinámica caótica, con múltiples puntos de equilibrio y una dependencia fuerte de las características no lineales del sistema, el cálculo de la histéresis utiliza los mismos parámetros descritos en el modelo de Stanford.

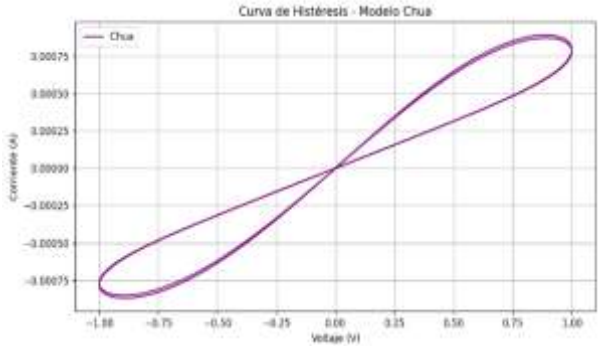

Fig. 9: Histéresis del memristor - modelo de Chua

Sistema de ecuaciones del modelo memristivo de Chua [12]:

$$\frac{d\emptyset}{dt} = \frac{v_1}{\zeta}$$

$$\frac{dv_1}{dt} = \frac{1}{C_1}\left(\frac{v_2 - v_1}{R} - W(\emptyset).v_1\right) \quad (7)$$

$$\frac{d_{v2}}{dt} = \frac{1}{C_2}\left(\frac{v_1 - v_2}{R} - i_L\right)$$

$$\frac{di_L}{dt} = \frac{v_2}{L}$$

$$W_{(\emptyset)} = \alpha + 3\beta\emptyset^2 \quad (8)$$

El atractor caótico Fig.10, muestra una trayectoria compleja y oscilatoria, que cambia de forma impredecible debido a su alta sensibilidad a las condiciones iniciales $[\phi, v_1, v_2, I_L] = [0, 0.1, 0.1, 0]$. Representa el comportamiento caótico del sistema, caracterizado por patrones irregulares y no repetitivos.

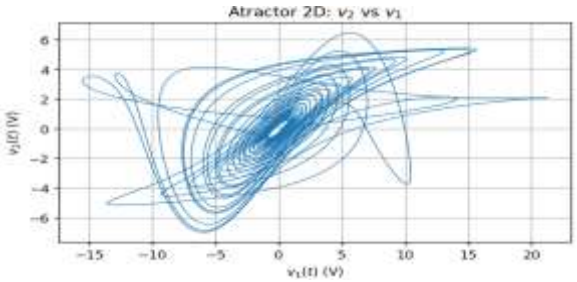

Fig. 10: Atractor caótico del modelo de Chua. $C_1 = C_2 = 6.98$, $L = 18e^{-3}$, $R = 2000$, $\zeta = 3.854e^{-4}$, $\propto = -0.667e^{-3}$, $\beta = 0.029e^{-3}$.

Sincronización de dos sistemas caóticos.

Se considera la sincronización como una herramienta para validar el estado caótico del sistema Fig.11; partiendo de condiciones iniciales diferentes, se emplea una interacción entre los sistemas controlada por un término de acoplamiento en las ecuaciones diferenciales. Este proceso asegura que, a pesar de la complejidad y la naturaleza caótica de los atractores, ambos sistemas pueden converger hacia trayectorias sincronizadas con el tiempo, evidenciando la estabilidad relativa en un contexto de caos.

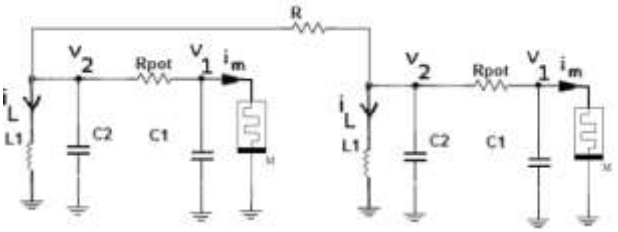

Fig. 11: Sincronización de dos circuitos – modelo de Chua.

Ecuaciones para la sincronización de sistemas caóticos

Condiciones iniciales:

$[\phi_1, v_{1,1}, v_{2,1}, i_{L1}] = [0, 0.1, 0.1, 0]$

$[\phi_2, v_{1,2}, v_{2,2}, i_{L2}] = [0, -0.1, -0.1, 0]$

Sistema 1

$$\frac{d\emptyset_1}{dt} = -\frac{v_{1,1}}{\zeta}$$
$$\frac{dv_{1,1}}{dt} = \frac{1}{C_1}\left(\frac{v_{2,1} - v_{1,1}}{R} - W(\emptyset_1).v_{1,1}\right) \quad (9)$$
$$\frac{d_{v2,1}}{dt} = \frac{1}{C_2}\left(\frac{v_{1,1} - v_{2,1}}{R} - i_{L1}\right)$$
$$\frac{di_{L1}}{dt} = \frac{v_{2,1}}{L}$$

Sistema 2

$$\frac{d\emptyset_2}{dt} = -\frac{v_{1,2}}{\zeta}$$
$$\frac{dv_{1,2}}{dt} = \frac{1}{C_1}\left(\frac{v_{2,2} - v_{1,2}}{R} - W(\emptyset_2).v_{1,2}\right) \quad (10)$$
$$\frac{d_{v2,2}}{dt} = \frac{1}{C_2}\left(\frac{v_{1,2} - v_{2,2}}{R} - i_{L2}\right) + k(v_{2,1} - v_{2,2})$$
$$\frac{di_{L2}}{dt} = \frac{v_{2,2}}{L}$$

La sincronización en el modelo de Chua Fig.12, se logra mediante un factor de acoplamiento que fuerza a dos sistemas a seguir trayectorias similares, incluso con condiciones iniciales diferentes. Esto permite que ambos sistemas se alineen con el tiempo, superando su comportamiento caótico.

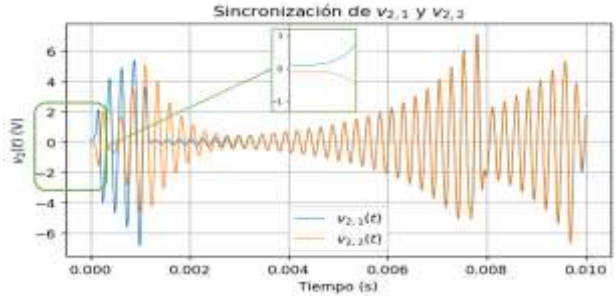

Fig. 12: Señales sincronizadas partiendo de condiciones iniciales distintas-modelo de Chua. K=0.55e⁴.

Modelo Memductor

Es un modelo general de memristor que enfatiza la memoria intrínseca basada en la integración de carga o flujo, con una respuesta lineal local y un comportamiento dinámico ajustable según la señal aplicada, su histéresis muestra un bucle más estrecho y una fuerte dependencia del historial del sistema, con una memoria que modula la relación voltaje-corriente de manera distintiva Fig.13. Igualmente, para el cálculo de la histéresis se utilizan los parámetros descritos en el modelo de Stanford.

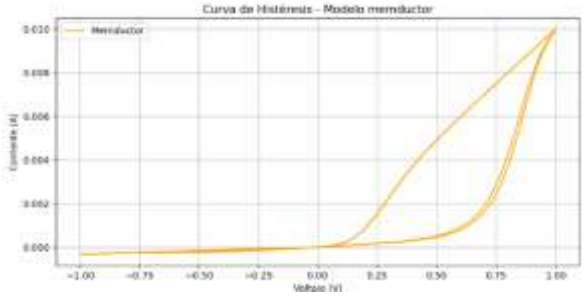

Fig. 13: Histéresis del memristor - modelo de memductor

El sistema de ecuaciones del modelo Memductor es el siguiente [13]:

$$\dot{x} = 4x + 16y + 0.1u - 2xu^2$$
$$\dot{y} = x - y + z \quad (11)$$
$$\dot{z} = -15y - 0.5z$$
$$\dot{u} = -x$$

El atractor caótico del Memductor es más impredecible comparado con los otros dos modelos Fig.14, pero sigue mostrando un comportamiento no lineal y dependiente de la memoria del sistema. Su forma refleja la evolución controlada por la carga o flujo, con trayectorias complejas que se adaptan a las condiciones impuestas por el acoplamiento.

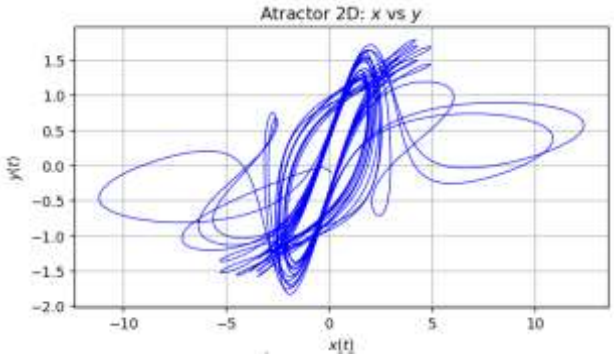

Fig. 14: Atractor caótico del modelo de memristor. $a = 4$, $b = 16$, $c = 0.1$, $d = -2$, $e = -15$, $f = -0.5$ y $g = 3$.

Sincronización de dos sistemas caoticos

Para el Memductor, la sincronización se plantea iniciando ambos sistemas desde estados iniciales diferentes y permitiéndoles interactuar mediante un acoplamiento proporcional a la memoria intrínseca

del sistema. Este enfoque demuestra cómo, incluso en sistemas que dependen fuertemente de su historial, las trayectorias pueden alinearse en función del tiempo Fig.15.

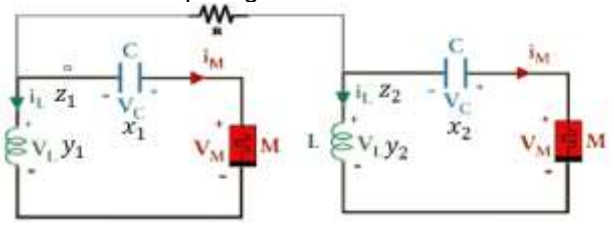

Fig. 15: Sincronización de dos circuitos – modelo de memristor.

Ecuaciones x_1 y_1 para la sincronización de sistemas caóticos.

Condiciones iniciales:

$$[x_1, y_1, z_1, u_1] = [0.1, 0, 0, 0.5e-9]$$

$$[x_2, y_2, z_2, u_2] = [0.2, 0.1, -0.1, 6e-9]$$

Sistema 1

$$\begin{aligned} \dot{x_1} &= 4x_1 + 16y_1 + 0.1u_1 - 2x_1u_1^2 \\ \dot{y}_1 &= x_1 - y_1 + z_1 \\ \dot{z}_1 &- 15y_1 - 0.5z_1 \\ \dot{u}_1 &= -x_1 \end{aligned} \qquad (12)$$

Sistema 2

$$\begin{aligned} \dot{x_2} &= 4x_2 + 16y_2 + 0.1u_2 - 2x_1u_1^2 \\ \dot{y}_2 &= x_2 - y_2 + z_2 \\ \dot{z}_2 &= -15y_2 - 0.5z_2 + 3(z_1 - z_2) \\ \dot{u}_2 &= -x_2 \end{aligned} \qquad (13)$$

Para sincronizar el modelo se aplica un factor de acoplamiento basado en la memoria intrínseca del sistema, ambos atractores logran sincronizarse a lo largo del tiempo Fig.16, independientemente de sus condiciones iniciales, mostrando que el sistema puede estabilizarse con el acoplamiento adecuado.

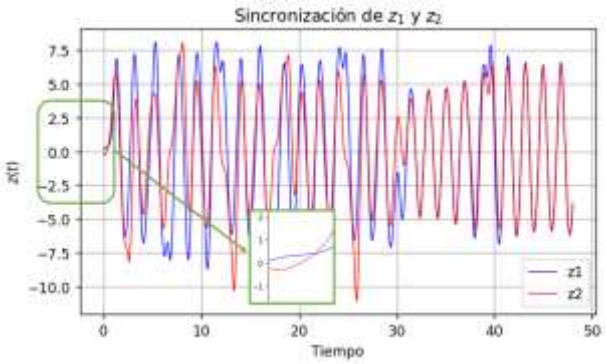

Fig. 16: Señales sincronizadas partiendo de condiciones iniciales distintas-modelo de Memristor. K=3.

DISCUSIÓN

Las simulaciones de los modelos estudiados evidencian la presencia de atractores caóticos en las proyecciones bidimensionales, destacando la complejidad de sus dinámicas. En todos los casos, se observó la sincronización de las trayectorias en de las variables tras un periodo transitorio gracias al término de acoplamiento dinámico, asegurando coherencia en el estado sincronizado. Las gráficas bidimensionales resaltan la estructura compleja de los atractores y validan la efectividad del diseño, al comparar sistemas acoplados y no acoplados.

Modelo Chua-Stanford:
La sincronización de los atractores caóticos logrado mediante un factor de acoplamiento, confirma la viabilidad de estabilizar sistemas caóticos con memristores. Este hallazgo es consistente con estudios previos, como los de Di Marco et al., y expande el conocimiento sobre la aplicación de sistemas memristivos en fenómenos no lineales.

Modelo Experimental Basado en Chua:
La sincronización observada en atractores complejos es consistente con los trabajos de Muthuswamy y Chua, demostrando la viabilidad experimental de estos sistemas en entornos reales. La sincronización eficaz entre los atractores destaca la capacidad de los memristores para estabilizar dinámicas caóticas.

Modelo Memductor:
El memductor demuestra su capacidad para generar atractores bidimensionales y curvas de histéresis, con la sincronización de los atractores mostrando la viabilidad de controlar el caos mediante acoplamientos basados en memoria. Estos

resultados respaldan hallazgos previos y amplían su aplicación en sistemas controlados.

5. CONCLUSIÓN

Los modelos estudiados resaltan la capacidad de Python como herramienta clave para el análisis y desarrollo de sistemas dinámicos complejos, destacando su utilidad en la implementación de simulaciones numéricas, la resolución de ecuaciones diferenciales y la visualización de atractores caóticos mediante bibliotecas como NumPy, Matplotlib y SciPy. Estas herramientas permitieron abordar las dinámicas no lineales introducidas por los memristores y estudiar la sincronización de variables específicas en sistemas caóticos, abriendo potenciales aplicaciones en criptografía, comunicaciones seguras y análisis de redes neuronales. Python no solo facilitó el manejo eficiente de estos modelos, sino que también evidenció su potencial como plataforma esencial en la investigación científica de sistemas no lineales. Como trabajos futuros, se propone desarrollar bibliotecas personalizadas que automaticen el análisis de modelos dinámicos y explorar la integración de Python con tecnologías como inteligencia artificial y computación simbólica para optimizar la resolución de problemas matemáticos complejos, posicionando a Python como una herramienta interdisciplinaria en matemáticas avanzadas y modelado computacional.

6. REFERENCIAS BIBLIOGRÁFICAS

[1] L. Chua, “Memristor-The missing circuit element,” *IEEE Transactions on Circuit Theory*, vol. 18, no. 5, pp. 507–519, 1971, doi: 10.1109/TCT.1971.1083337.

[2] C. O. Marambio, K. C. Valenzuela, and A. R. Estay, “Memristor. Una perspectiva general,” Interciencia, vol. 39, no. 7, pp. 458–467, Jul. 2014.

[3] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, “The missing memristor found,” *Nature*, vol. 453, no. 7191, pp. 80–83, May 2008, doi: 10.1038/nature06932.

[4] A. Adamatzky and L. Chua, Eds., *Memristor Networks*. Cham: Springer International Publishing, 2014. doi: 10.1007/978-3-319-02630-5.

[5] B. MUTHUSWAMY and L. O. CHUA, “SIMPLEST CHAOTIC CIRCUIT,” *International Journal of Bifurcation and Chaos*, vol. 20, no. 05, pp. 1567–1580, May 2010, doi: 10.1142/S0218127410027076.

[6] Y. V. Pershin and M. Di Ventra, “Memory effects in complex materials and nanoscale systems,” *Adv Phys*, vol. 60, no. 2, pp. 145–227, Apr. 2011, doi: 10.1080/00018732.2010.544961.

[7] E. Miranda and J. Suñé, “Memristors for Neuromorphic Circuits and Artificial Intelligence Applications,” *Materials*, vol. 13, no. 4, p. 938, Feb. 2020, doi: 10.3390/ma13040938.

[8] M. Mayacela, L. Rentería, L. Contreras, and S. Medina, “Comparative Analysis of Reconfigurable Platforms for Memristor Emulation,” *Materials*, vol. 15, no. 13, p. 4487, Jun. 2022, doi: 10.3390/ma15134487.

[9] L. Rentería, M. Mayacela, K. Torres, W. Ramírez, R. Donoso, and R. Acosta, “FPGA-Based Numerical Simulation of the Chaotic Synchronization of Chua Circuits,” *Computation*, vol. 12, no. 9, p. 174, Aug. 2024, doi: 10.3390/computation12090174.

[10] E. Bilotta, F. Chiaravalloti, and P. Pantano, “Spontaneous Synchronization in Two Mutually Coupled Memristor-Based Chua’s Circuits: Numerical Investigations,” *Math Probl Eng*, vol. 2014, pp. 1–15, 2014, doi: 10.1155/2014/594962.

[11] M. Di Marco, M. Forti, G. Innocenti, and A. Tesi, “Harmonic Balance Design of Oscillatory Circuits Based on Stanford Memristor Model,” *IEEE Access*, vol. 11, pp. 127431–127445, 2023, doi: 10.1109/ACCESS.2023.3331107.

[12] L. Xiong, X. Zhang, S. Teng, L. Qi, and P. Zhang, “Detecting Weak Signals by Using Memristor-Involved Chua’s Circuit and Verification in Experimental Platform,” *International Journal of Bifurcation and Chaos*, vol. 30, no. 13, p. 2050193, Oct. 2020, doi: 10.1142/S021812742050193X.

[13] B. MUTHUSWAMY, “IMPLEMENTING MEMRISTOR BASED CHAOTIC CIRCUITS,” *International Journal of Bifurcation and Chaos*, vol. 20, no. 05, pp. 1335–1350, May 2010, doi: 10.1142/S0218127410026514.

ANEXO:

Anexo 1: Código histéresis del memristor – Modelo Stanford

```
import numpy as np
import matplotlib.pyplot as plt

# Parámetros
R_on = 100  # Resistencia en estado "on"
R_off = 1600  # Resistencia en estado "off"
alpha = 0.03  # Velocidad de cambio de la resistencia
beta = 0.9  # Trayectoria de histéresis

# Tiempo y voltaje senoidal
tiempo = np.arange(0, 2, 0.01)
voltaje_senoidal = np.sin(2 * np.pi * 1 * tiempo)

# Inicialización de corriente y carga
corriente = np.zeros_like(voltaje_senoidal)
q = 0

# Función de resistencia
def calcular_resistencia_stanford(q, R_on, R_off, beta):
    return R_on + (R_off - R_on) * (1 - np.exp(-beta * np.abs(q)))

# Simulación
for i, V in enumerate(voltaje_senoidal):
    q += alpha * V * (1 - beta * np.abs(q))  # carga
    R = calcular_resistencia_stanford(q, R_on, R_off, beta)
    corriente[i] = V / R

# Gráfica
plt.figure(figsize=(10, 5))
plt.plot(voltaje_senoidal, corriente, label="Stanford", color="blue")
plt.xlabel("Voltaje (V)")
plt.ylabel("Corriente (A)")
plt.title("Curva de Histéresis - Modelo Stanford")
plt.grid(True)
plt.legend()
plt.show()
```

Anexo 2: Código sincronización de dos circuitos caóticos – Modelo Stanford

```
import numpy as np
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Parámetros del sistema
alpha = 10
beta = 15
a = -1.27
b = -0.68
q = 1.6e-19
k = 1.38e-23
T0 = 300
Ea_g = 0.8
a0 = 1e-10
l = 10e-9
gamma0 = 2.5
g_min = 0.5e-9
g_max = 6.0e-9
V0 = 1.0
# Constante de acoplamiento
coupling_strength = 1.1

# Función no lineal
def f(x):
    return b * x + 0.5 * (a - b) * (np.abs(x + 1) - np.abs(x - 1))

# Sincronización
def synchronized_system(t, state):
    x1, y1, z1, g1, x2, y2, z2, g2 = state

    # gaps
    g1 = np.clip(g1, g_min, g_max)
    g2 = np.clip(g2, g_min, g_max)

    # Sistema 1
    dx1_dt = alpha * (y1 - x1 - f(x1))
    dy1_dt = x1 - y1 + z1
    dz1_dt = -beta * y1
    dg1_dt = -np.exp(-q * Ea_g * g1 / (k * T0)) * np.exp(q * a0 * gamma0 / (l * k * T0) * x1)

    # Sistema 2
    dx2_dt = alpha * (y2 - x2 - f(x2))
    dy2_dt = x2 - y2 + z2
    dz2_dt = -beta * y2 + coupling_strength * (z1 - z2)
    dg2_dt = -np.exp(-q * Ea_g * g2 / (k * T0)) * np.exp(q * a0 * gamma0 / (l * k * T0) * x2)

    return [dx1_dt, dy1_dt, dz1_dt, dg1_dt, dx2_dt, dy2_dt, dz2_dt, dg2_dt]

# Condiciones iniciales
x0_m, y0_m, z0_m, g0_m = 0.1, 0, 0, g_min
x0_s, y0_s, z0_s, g0_s = 0.2, 0.1, -0.1, g_max
initial_conditions = [x0_m, y0_m, z0_m, g0_m, x0_s, y0_s, z0_s, g0_s]
```

```
# Parámetros
t_span = (0, 50)
t_eval = np.linspace(t_span[0], t_span[1], 5000)

# Resolución
solution = solve_ivp(
    synchronized_system,
    t_span,
    initial_conditions,
    t_eval=t_eval,
    method="RK45"
)
t = solution.t
x1, y1, z1, g1, x2, y2, z2, g2 = solution.y

# Gráficas
plt.figure(figsize=(12, 10))
# Maestro-esclavo (x1 vs x2 y x2 vs x1)
plt.subplot(2, 2, 1)
plt.plot(x1, y1, 'b', label="x1")
plt.plot(x2, y2, 'r', label="x2 ")
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Espacio de fases (x1 vs x2)')
plt.legend()
plt.grid(True)
# Sincronización
plt.subplot(2, 2, 2)
plt.plot(t, z1, 'b', label="z1")
plt.plot(t, z2, 'r', label="z2 ")
plt.xlabel('Tiempo')
plt.ylabel('z (t)')
plt.title('Sincronización de z1 y z2')
plt.legend()
plt.grid(True)

# Gap
plt.subplot(2, 2, 3)
plt.plot(t, g1, 'b', label="g1")
plt.plot(t, g2, 'r', label="g2 ")
plt.xlabel('Tiempo (s)')
plt.ylabel('g')
plt.title('Evolución del gap (g)')
plt.legend()
plt.grid(True)

# Atractores 3D
ax = plt.subplot(2, 2, 4, projection='3d')
ax.plot(x1, y1, z1, 'b', label="x1")
ax.plot(x2, y2, z2, 'r', label="x2")
ax.set_xlabel('x1')
ax.set_ylabel('x2')
ax.set_zlabel('z')
ax.set_title('Atractores 3D')
ax.legend()
ax.grid(True)

plt.tight_layout()
plt.show()
```

Anexo 3: Código histéresis del memristor – Modelo Chua

```
import numpy as np
import matplotlib.pyplot as plt

# Parámetros
R_on = 100  # Resistencia en estado "on"
R_off = 1600  # Resistencia en estado "off"
alpha = 0.03  # Controla la velocidad de cambio de la resistencia
beta = 0.9  # Trayectoria de histéresis

# Tiempo y voltaje senoidal
tiempo = np.arange(0, 2, 0.01)
voltaje_senoidal = np.sin(2 * np.pi * 1 * tiempo)

# Inicialización de corriente y carga
corriente = np.zeros_like(voltaje_senoidal)
q = 0

# Función de resistencia
def calcular_resistencia_chua(q, R_on, R_off):
    return R_on + (R_off - R_on) * (1 - q**2)

# Simulación
corriente_chua = np.zeros_like(voltaje_senoidal)
q = 0

for i, V in enumerate(voltaje_senoidal):
    q += alpha * V * (1 - beta * q**2)
    R = calcular_resistencia_chua(q, R_on, R_off)
    corriente_chua[i] = V / R

# Gráfica
plt.figure(figsize=(10, 5))
plt.plot(voltaje_senoidal, corriente_chua, label="Chua", color="purple")
plt.xlabel("Voltaje (V)")
plt.ylabel("Corriente (A)")
plt.title("Curva de Histéresis - Modelo Chua")
plt.grid(True)
plt.legend()
plt.show()
```

Anexo 4: Código sincronización de dos circuitos caóticos – Modelo Chua

```
import numpy as np
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Parámetros
alpha = -0.667e-3
beta = 0.029e-3
zeta = 8200 * 47e-9
C1 = 6.8e-9
C2 = 68e-9
L = 18e-3
R = 2000

# Función de memductancia
def W(phi):
    return alpha + 3 * beta * phi**2

# Sistema sincronizado
def synchronized_system(t, state):
    phi1, v1_1, v2_1, iL1, phi2, v1_2, v2_2, iL2 = state

    # Sistema 1
    dphi1_dt = -v1_1 / zeta
    dv1_1_dt = (1 / C1) * ((v2_1 - v1_1) / R - W(phi1) * v1_1)
    dv2_1_dt = (1 / C2) * ((v1_1 - v2_1) / R - iL1)
    diL1_dt = v2_1 / L

    # Sistema 2
    dphi2_dt = -v1_2 / zeta
    dv1_2_dt = (1 / C1) * ((v2_2 - v1_2) / R - W(phi2) * v1_2)
    dv2_2_dt = (1 / C2) * ((v1_2 - v2_2) / R - iL2)
    diL2_dt = v2_2 / L

    # Acoplamiento
    coupling_strength = 0.55e4
    dv2_2_dt += coupling_strength * (v2_1 - v2_2)

    return [dphi1_dt, dv1_1_dt, dv2_1_dt, diL1_dt, dphi2_dt, dv1_2_dt, dv2_2_dt, diL2_dt]

# Condiciones iniciales
initial_conditions = [0, 0.1, 0.1, 0, 0, -0.1, -0.1, 0]

# Tiempo
t_span = (0, 0.01)
t_eval = np.linspace(t_span[0], t_span[1], 5000)

# Sistema sincronizado
solution = solve_ivp(synchronized_system, t_span, initial_conditions, t_eval=t_eval,
method='RK45')
t = solution.t
phi1, v1_1, v2_1, iL1, phi2, v1_2, v2_2, iL2 = solution.y

# Gráficas
plt.figure(figsize=(12, 10))
```

```
# Gráfico 2D: v2_1 vs v1_1 y v2_2 vs v1_2
plt.subplot(2, 2, 1)
plt.plot(v1_1, v2_1, label='Sistema 1', linewidth=0.8)
plt.plot(v1_2, v2_2, label='Sistema 2', linewidth=0.8)
plt.xlabel(r'$v_1(t)$ (V)')
plt.ylabel(r'$v_2(t)$ (V)')
plt.title('Atractor 2D: $v_2$ vs $v_1$')
plt.legend()
plt.grid(True)

# Gráfico 2D: Sincronización de v2_1 y v2_2
plt.subplot(2, 2, 2)
plt.plot(t, v2_1, label='$v_{2,1}(t)$', linewidth=0.8)
plt.plot(t, v2_2, label='$v_{2,2}(t)$', linewidth=0.8)
plt.xlabel('Tiempo (s)')
plt.ylabel(r'$v_2(t)$ (V)')
plt.title('Sincronización de $v_{2,1}$ y $v_{2,2}$')
plt.legend()
plt.grid(True)

# Gráfico 2D: phi1 vs iL1 y phi2 vs iL2
plt.subplot(2, 2, 3)
plt.plot(phi1, iL1, label='Sistema 1', linewidth=0.8)
plt.plot(phi2, iL2, label='Sistema 2', linewidth=0.8)
plt.xlabel(r'$\phi(t)$ (Wb)')
plt.ylabel(r'$i_L(t)$ (A)')
plt.title(r'Atractor 2D: $i_L$ vs $\phi$')
plt.legend()
plt.grid(True)

# Gráfico 3D: v1_1, v2_1, phi1 y v1_2, v2_2, phi2
ax = plt.subplot(2, 2, 4, projection='3d')
ax.plot(v1_1, v2_1, phi1, color='blue', linewidth=0.5, label='Sistema 1')
ax.plot(v1_2, v2_2, phi2, color='red', linewidth=0.5, label='Sistema 2')
ax.set_xlabel(r'$v_1(t)$ (V)')
ax.set_ylabel(r'$v_2(t)$ (V)')
ax.set_zlabel(r'$\phi(t)$ (Wb)')
ax.set_title('Atractores 3D')
ax.legend()
ax.grid(True)

# Título
plt.suptitle('Sincronización de Sistemas Memristivos', fontsize=14)
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```

Anexo 5: Código histéresis del memristor – Modelo Memductor

```
import numpy as np
import matplotlib.pyplot as plt

# Parámetros
R_on = 100  # Resistencia en estado "on"
R_off = 1600  # Resistencia en estado "off"
alpha = 0.03  # Controla la velocidad de cambio de la resistencia
beta = 0.9  # Trayectoria de histéresis

# Tiempo y voltaje senoidal
tiempo = np.arange(0, 2, 0.01)
voltaje_senoidal = np.sin(2 * np.pi * 1 * tiempo)

# Inicialización de corriente y carga
corriente = np.zeros_like(voltaje_senoidal)
q = 0

# Función de resistencia
def calcular_resistencia_memductor(q, V, R_on, R_off):
    return R_on + (R_off - R_on) * (1 - np.tanh(10 * V * q))

# Simulación
corriente_memductor = np.zeros_like(voltaje_senoidal)
q = 0

for i, V in enumerate(voltaje_senoidal):
    q += alpha * V * (1 - beta * q**2)
    R = calcular_resistencia_memductor(q, V, R_on, R_off)
    corriente_memductor[i] = V / R

# Gráfica
plt.figure(figsize=(10, 5))
plt.plot(voltaje_senoidal, corriente_memductor, label="Memductor", color="orange")
plt.xlabel("Voltaje (V)")
plt.ylabel("Corriente (A)")
plt.title("Curva de Histéresis - Modelo memductor")
plt.grid(True)
plt.legend()
plt.show()
```

Anexo 6: Código sincronización de dos circuitos caóticos – Modelo Memductor

```
import numpy as np
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Sistema de ecuaciones diferenciales
def chaotic_system(t, state):
    x, y, z, u = state
    dx_dt = 4 * x + 16 * y + 0.1 * u - 2 * x * u**2
    dy_dt = x - y + z
    dz_dt = -15 * y - 0.5 * z
    du_dt = -x
    return [dx_dt, dy_dt, dz_dt, du_dt]

# Sincronización:
def synchronized_system(t, state):
    x1, y1, z1, u1, x2, y2, z2, u2 = state

    # Sistema 1
    dx1_dt = 4 * x1 + 16 * y1 + 0.1 * u1 - 2 * x1 * u1**2
    dy1_dt = x1 - y1 + z1
    dz1_dt = -15 * y1 - 0.5 * z1
    du1_dt = -x1

    # Sistema 2
    dx2_dt = 4 * x2 + 16 * y2 + 0.1 * u2 - 2 * x2 * u2**2
    dy2_dt = x2 - y2 + z2
    dz2_dt = -15 * y2 - 0.5 * z2 + 3 * (z1 - z2)
    du2_dt = -x2

    return [dx1_dt, dy1_dt, dz1_dt, du1_dt, dx2_dt, dy2_dt, dz2_dt, du2_dt]

# Parámetros
t_span = (0, 48)
t_eval = np.linspace(t_span[0], t_span[1], 5000)

# Condiciones iniciales
initial_conditions = [0.1, -0.1, 0.1, -0.1, -0.5, 0.2, -0.2, 0.1]

# Solucion
solution = solve_ivp(
    synchronized_system,
    t_span,
    initial_conditions,
    t_eval=t_eval,
    method="RK45"
)

# Solucion
t = solution.t
x1, y1, z1, u1, x2, y2, z2, u2 = solution.y

# Graficas
plt.figure(figsize=(12, 10))
```

```
# Gráfico 2D: x1 vs y1 y x2 vs y2
plt.subplot(2, 2, 1)
plt.plot(x1, y1, 'b', linewidth=0.8, label="Sistema 1")
plt.plot(x2, y2, 'r', linewidth=0.8, label="Sistema 2")
plt.xlabel(r'$x(t)$')
plt.ylabel(r'$y(t)$')
plt.title(r'Atractor 2D: $x$ vs $y$')
plt.legend()
plt.grid(True)

# sincronización
plt.subplot(2, 2, 2)
plt.plot(t, z1, 'b', linewidth=0.8, label="z1")
plt.plot(t, z2, 'r', linewidth=0.8, label="z2")
plt.xlabel('Tiempo')
plt.ylabel(r'$z(t)$')
plt.title(r'Sincronización de $z_1$ y $z_2$')
plt.legend()
plt.grid(True)

# Gráfico 2D: y1 vs u1 y y2 vs u2
plt.subplot(2, 2, 3)
plt.plot(y1, u1, 'b', linewidth=0.8, label="Sistema 1")
plt.plot(y2, u2, 'r', linewidth=0.8, label="Sistema 2")
plt.xlabel(r'$y(t)$')
plt.ylabel(r'$u(t)$')
plt.title(r'Atractor 2D: $y$ vs $u$')
plt.legend()
plt.grid(True)

# Gráfico 3D: x1, y1, z1 y x2, y2, z2
ax = plt.subplot(2, 2, 4, projection='3d')
ax.plot(x1, y1, z1, color='blue', linewidth=0.5, label="Sistema 1")
ax.plot(x2, y2, z2, color='red', linewidth=0.5, label="Sistema 2")
ax.set_xlabel(r'$x(t)$')
ax.set_ylabel(r'$y(t)$')
ax.set_zlabel(r'$z(t)$')
ax.set_title('Atractores 3D')
ax.legend()
ax.grid(True)

# Título
plt.suptitle('Sincronización de Sistemas Caóticos', fontsize=14)
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```