Generadores de números pseudoaleatorios

Papaolo Francisco
papaolofrann@gmail.com
Legajo 50249

Renzi Álvaro
alvarorenzivillani@gmail.com
Legajo 49621

Spertino Mateo
mateospertino@gmail.com
Legajo 51753

Mercado Martin
martinmercado1502@gmail.com
Legajo 48453

Karlen Aguirre Esteban
estebankarlenaguirre@gmail.com
Legajo 48178

Iglina Bruno
brunoiglina@gmail.com
Legajo 49623

Higonet Juan Ignacio
jhigonet89@gmail.com
Legajo 49951

8 de mayo de 2025

Resumen

Este trabajo presenta la implementación y análisis de generadores de números pseudoaleatorios, fundamentales para modelos de simulación. Se programan dos métodos: el Generador Congruencial Lineal (GCL) y el método de los Cuadrados Medios, y se los compara con el generador incluido por defecto en Python. A cada uno se le aplican múltiples pruebas estadísticas para evaluar su calidad, incluyendo uniformidad, independencia, y ajuste a distribuciones teóricas. Los resultados obtenidos se presentan en tablas y gráficos, permitiendo concluir sobre la confiabilidad de cada generador según su comportamiento esperado.

1. Introducción

Los generadores de números pseudoaleatorios (PRNG, por sus siglas en inglés) son herramientas fundamentales en el ámbito de la simulación, ya que permiten modelar fenómenos estocásticos mediante secuencias que, si bien son generadas de forma determinista, imitan el comportamiento del azar. Su uso es esencial en diversas áreas de la informática, como la criptografía, los videojuegos, la inteligencia artificial y, especialmente, la simulación numérica.

A diferencia de los generadores verdaderamente aleatorios, los PRNG reproducen siempre la misma secuencia si se parte del mismo valor inicial (semilla), lo que permite replicar experimentos computacionales y controlar la variabilidad. Estos generadores se basan en algoritmos que, a pesar de su naturaleza determinista, deben cumplir con estrictos requisitos estadísticos para que sus secuencias sean consideradas adecuadas para representar eventos aleatorios.

Este informe tiene como objetivo estudiar e implementar distintos algoritmos generadores de números pseudoaleatorios, con un enfoque particular en el Generador Congruencial Lineal (GCL) y el método de los Cuadrados Medios. Además, se los compara con el generador pseudoaleatorio provisto por defecto en Python. A lo largo del trabajo, se analizan los fundamentos teóricos de cada método, su implementación práctica y la calidad de las secuencias generadas, evaluada mediante diversos tests estadísticos. La finalidad es determinar su adecuación para aplicaciones en simulación, resaltando tanto sus fortalezas como sus limitaciones.

2. Descripción del Trabajo

El trabajo de investigación consiste en construir programas en lenguaje Python 3.x que generen números pseudoaleatorios y que estos se comporten como se espera. Para esto se debe tener en cuenta lo siguientes temas a investigar:

- **Generadores de números aleatorios reales**
- **Generadores de números pseudoaleatorios (Método de los cuadrados, GCLs, otros).**
- **Test para determinar el comportamiento de los generadores.**

Se pide programar al menos dos generadores de números pseudoaleatorios en particular el generador GCL del cual se debe testear con al menos cuatro pruebas para determinar la calidad de generación. También se pide comparar los generadores programados con otros (incluyendo el que posee el lenguaje Python).

3. Metodología

Cuando se trabaja con números generados de forma aleatoria, es fundamental distinguir entre sus dos principales orígenes: los números aleatorios verdaderos (TRNG) y los números pseudoaleatorios (PRNG).

Los TRNG obtienen la aleatoriedad a partir de fenómenos físicos impredecibles y la incorporan al entorno digital. En cambio, los PRNG son generados por algoritmos que imitan el comportamiento aleatorio, pero que en realidad son completamente deterministas. Aunque a simple vista las secuencias producidas por un PRNG parecen no seguir ningún patrón, en el fondo están definidas por fórmulas matemáticas o listas preestablecidas.

Una forma simple de entender esta diferencia es comparando un dado físico con una lista de resultados preanotados: mientras el TRNG se asemeja a tirar un dado real, el PRNG equivale a recorrer una lista de tiradas ya conocidas. Por lo tanto, aunque los números parezcan aleatorios, en el caso del PRNG están completamente determinados por su punto de partida (la semilla).

Los PRNG destacan por su eficiencia, ya que permiten generar grandes volúmenes de números en poco tiempo, y por su carácter reproducible, lo que significa que una misma secuencia puede volver a obtenerse si se conoce la semilla original. Sin embargo, tienen una naturaleza periódica, es decir, eventualmente las secuencias se repiten. Afortunadamente, los generadores modernos tienen periodos tan extensos que esta repetición no representa un problema en la mayoría de las aplicaciones.

Estas propiedades hacen que los PRNG sean ideales para entornos donde se requieren muchas muestras aleatorias y es útil poder repetir los experimentos, como ocurre en la simulación y el modelado. Sin embargo, no son apropiados para ámbitos donde la imprevisibilidad absoluta es crucial, como en la criptografía o los juegos de azar.

A continuación se detallará cada generador.

3.1. Generador Congruencial Lineal (GLC)

3.1.1. Descripción del método

El Generador Congruencial Lineal (GCL) es uno de los métodos más conocidos y sencillos para generar números pseudoaleatorios. Se basa en una fórmula matemática recursiva que, a partir de un valor inicial denominado *semilla*, produce una secuencia de valores que aparentan ser aleatorios.

La fórmula general utilizada por este método es:

$$X_{n+1} = (aX_n + c) \mod m$$

donde X representa la sucesión de números generados, y los parámetros son:

- m: el **módulo**, un número entero positivo ($m > 0$),
- a: el **multiplicador** ($0 < a < m$),
- c: el **incremento** ($0 \leq c < m$),
- X_0: la **semilla inicial** ($0 \leq X_0 < m$).

Esta fórmula asegura que, partiendo de una misma semilla y parámetros, se obtenga siempre la misma secuencia. El comportamiento del generador depende en gran medida de la elección adecuada de estos valores.

3.1.2. Parámetros del Método GLC

- **Semilla (X_0):** Representa el valor inicial desde el cual se comienza a generar la secuencia. La elección de una buena semilla es esencial, ya que puede influir significativamente en la calidad de la secuencia obtenida.
- **Multiplicador (a):** Es el coeficiente que define cómo se propagan los valores en la secuencia. Su selección debe ser cuidadosa, ya que afecta directamente la uniformidad y distribución de los números generados.
- **Incremento (c):** Introduce una variación adicional en la fórmula del generador. En ciertos casos, se puede utilizar un valor nulo ($c = 0$), situación en la cual el método se reduce a su versión multiplicativa.
- **Módulo (m):** Determina el límite superior del rango de valores posibles en la secuencia. Generalmente se elige un valor grande —como una potencia de 2— para ampliar la cobertura del espacio muestral y extender el periodo del generador.
- **Cantidad (n):** Determina la cantidad de números a generar.

3.1.3. Propiedades Deseadas

Características ideales que debe tener un buen GLC:

- **Largo período:** idealmente cercano a m, sin ciclos cortos.
- **Uniformidad:** los valores generados deben distribuirse uniformemente.
- **Independencia:** no deben haber correlaciones entre los números.

- **Reproducibilidad:** dado que es determinístico, debe permitir reproducir la misma secuencia.
- **Eficiencia computacional:** debe ser rápido y simple de calcular.
- **Sensibilidad a la semilla:** distintos valores de X_0 deben producir secuencias no correlacionadas.

3.2. Método de los cuadrados medios

3.2.1. Descripción del método de los cuadrados medios

El método de los cuadrados medios se fundamenta en una operación matemática simple: elevar un número al cuadrado. A partir de una semilla inicial, el procedimiento se desarrolla de la siguiente manera:

- **Definición de la semilla:** Se selecciona un número inicial X_0.
- **Cálculo del cuadrado:** Se calcula el cuadrado del número actual X_n, obteniendo X_n^2.
- **Extracción central:** Se toman una cantidad determinada de dígitos centrales del resultado para formar el nuevo número X_{n+1}.
- **Iteración:** Se repite el proceso utilizando X_{n+1} como nueva entrada para la siguiente iteración.

3.2.2. Propiedades y Limitaciones

Propiedades:

- **Fácil de implementar:** solo se necesita elevar al cuadrado y extraer dígitos.
- **Determinístico:** como todos los métodos pseudoaleatorios.

Limitaciones:

- **Poca estabilidad:** rápidamente puede caer en ciclos muy cortos (incluso cero).
- **Pequeño rango de salida:** los valores pueden perder dígitos significativos rápidamente.
- **Sensibilidad extrema:** un mal valor inicial puede llevar a secuencias inútiles.
- **No uniforme:** no garantiza una distribución uniforme en el rango $[0, 1)$.
- **Obsoleto:** raramente usado hoy en día por sus malas propiedades estadísticas.

4. Parámetros de lanzamiento

- **Generador GLC**
  python randomGLC.py -s 12345 -a 1664525 -c 1013904223 -m 2**32 -n 8000
  Donde -s es la semilla X_0

- **Generador Mid-Square**
  python randomMidSquare.py -s 9731 -n 100

- **Generador Nativo Python**
  python randomPython.py -s 1234 -n 8000

5. Resultados Obtenidos

5.1. Generador Mid-Square

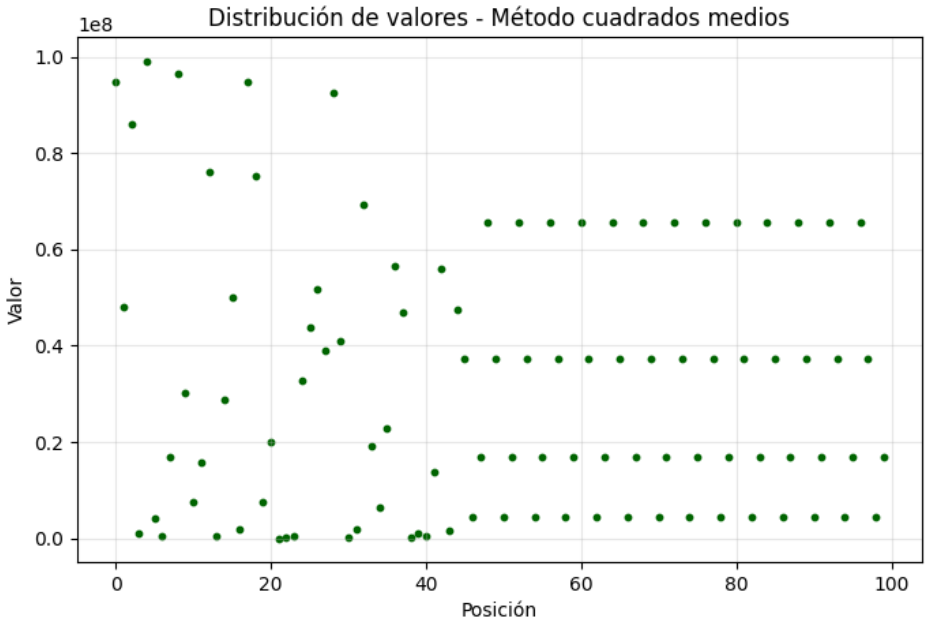

Figura 1: Distribución del Generador MidSquare

El gráfico que se muestra revela el comportamiento del método de cuadrados medios, y coincide con lo que esperamos ver en este tipo de generador. Lo que se observa es una característica (o defecto) clásico del método de cuadrados medios de von Neumann:

- Fase inicial variable: Los primeros 45 números muestran cierta variabilidad y parecen distribuirse de manera aleatoria.
- Ciclo posterior: Después de aproximadamente 45 iteraciones, el generador entra en un ciclo claramente visible, donde los valores se repiten en un patrón de 4 puntos distintos (4 niveles horizontales).

Este comportamiento es una debilidad conocida del método de cuadrados medios:

- Eventualmente cae en ciclos.
- A menudo estos ciclos son cortos.
- Una vez que entra en un ciclo, la secuencia pierde completamente su aleatoriedad.

RESULTADOS

1. **Test de Frecuencia (ÉXITO):**
   Chi²: 0.7856, p-valor: 0.9998
   Los dígitos siguen distribución uniforme.
2. **Test Chi Cuadrado (FALLO):**
   Chi²: 85.0000, p-valor: 0.0000
   Los valores no siguen distribución uniforme.

3. **Test Póker (FALLO):**
   Chi²: 7141.5013, p-valor: 0.0000
   Distribución de patrones no aleatoria.

4. **Test Rachas (FALLO):**
   Rachas: 56, Z: -2.4733
   Secuencia no independiente.
   Muy pocas rachas detectadas.

Explicación de los Resultados del Generador Mid-Square

1. **Test de Frecuencia (Chi²: 0.7856, p-valor: 0.9998)**
   Este test solo verifica si cada dígito individual (0-9) aparece aproximadamente con la misma frecuencia.
   El Mid-Square pasa este test porque los dígitos individuales están bastante bien distribuidos.
   Es el test menos riguroso de todos, ya que solo evalúa la distribución de dígitos independientes.

2. **Test Chi Cuadrado (Chi²: 85.0000, p-valor: 0.0000)**
   Este test examina la distribución general de los valores generados en intervalos.
   Falló porque, aunque los dígitos individuales estén bien distribuidos, los números completos no se distribuyen uniformemente en todo el rango posible.
   El valor p de 0.0000 indica que hay una probabilidad extremadamente baja de que esta distribución sea aleatoria.

3. **Test Póker (Chi²: 7141.5013, p-valor: 0.0000)**
   Analiza patrones dentro de los dígitos de cada número (como si fueran cartas de póker).
   El valor Chi² extremadamente alto (7141.5013) muestra que hay patrones muy fuertes y repetitivos.
   Esto ocurre porque el método Mid-Square tiende a generar ciertos patrones de dígitos con mayor frecuencia.

4. **Test de Rachas (Rachas: 56, Z: -2.4733)**
   Evalúa si hay independencia entre números consecutivos.
   El valor Z negativo (-2.4733) indica que hay menos rachas de lo esperado.
   Esto revela una fuerte correlación entre números consecutivos, lo cual es un problema típico del método Mid-Square.

¿Por qué ocurre esto?
El problema principal del Mid-Square es que entra en ciclos rápidamente:

$$\ldots \to 8100 \to 6100 \to 2100 \to 4100 \to 8100 \to 6100 \to 2100 \to 4100 \to \ldots$$

Observa que al final de la secuencia, después de generar unos pocos valores, el generador ha entrado en un ciclo de solo 4 valores (8100, 6100, 2100, 4100) que se repiten constantemente. Aunque cada dígito individual pueda estar bien distribuido, esto causa:

- Una distribución no uniforme en el espacio de valores posibles.
- Patrones muy predecibles (fallando el test de póker).
- Valores consecutivos altamente correlacionados (fallando el test de rachas).

Este comportamiento cíclico es una debilidad conocida del método Mid-Square, y es por eso que no se usa en aplicaciones que requieren alta calidad de aleatoriedad, a diferencia del generador de Python (presentado más adelante) que pasa todas las pruebas.

5.2. Generador GLC

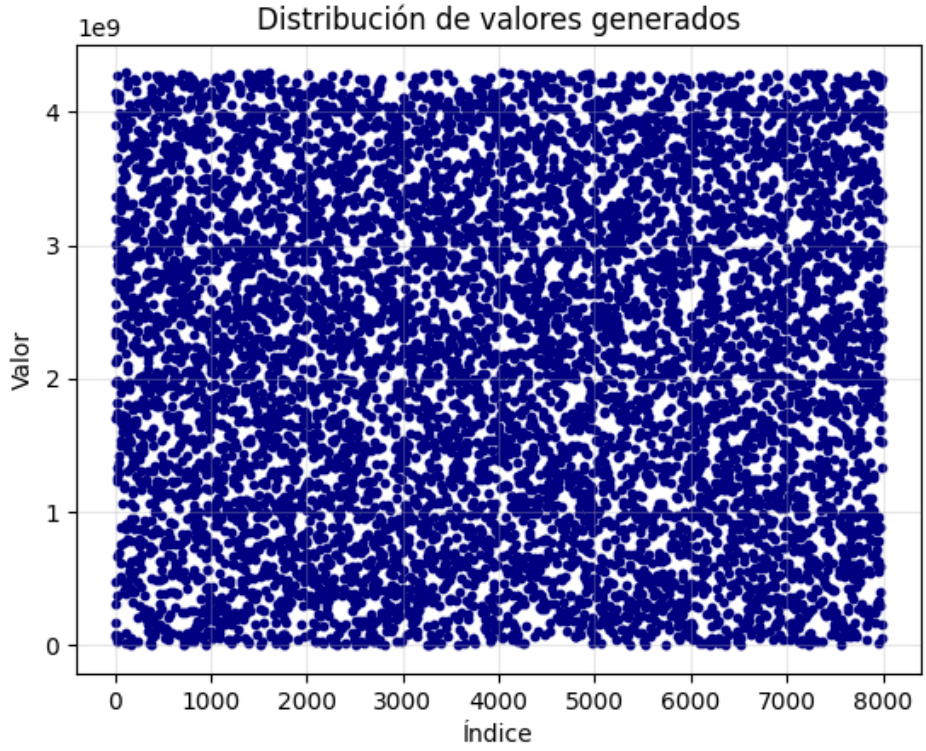

Figura 2: Distribución del Generador Lineal Congruencial (GLC)

Este gráfico muestra un comportamiento completamente diferente al anterior y corresponde al resultado de un Generador Lineal Congruencial (GLC) ejecutado con buenos parámetros. Las diferencias son notables:

- **Distribución uniforme:** Los valores se distribuyen uniformemente a lo largo de todo el rango (aproximadamente de 0 a 4.2×10^9), sin áreas vacías ni agrupaciones visibles.
- **Ausencia de ciclos cortos:** A diferencia del método de cuadrados medios, no se observan patrones cíclicos evidentes. La distribución parece genuinamente aleatoria.
- **Mayor número de valores:** Se generaron aproximadamente 8,000 valores (contra los 100 del ejemplo anterior), y aun así no se alcanza a detectar ciclicidad.
- **Cobertura completa:** Los valores llenan todo el espacio vertical disponible, indicando que el generador es capaz de producir números en todo el rango posible.

Este es un excelente ejemplo de por qué los Generadores Lineales Congruenciales con parámetros bien seleccionados son mucho más eficaces que el método de cuadrados medios para aplicaciones prácticas. La calidad estadística es notablemente superior.

Los GLC bien parametrizados tienen períodos muy largos (en este caso, potencialmente hasta $2^{32} \approx 4.3\times10^9$ valores antes de repetirse), lo que los hace adecuados para simulaciones y aplicaciones que requieren grandes cantidades de números pseudoaleatorios.

RESULTADOS

1. **Test de Frecuencia (ÉXITO):**
   Chi²: 0.0123, p-valor: 1.0000
   Los dígitos siguen distribución uniforme.

2. **Test Chi Cuadrado (ÉXITO)**:
   Chi²: 10.8825, p-valor: 0.2838
   Los valores siguen distribución uniforme.

3. **Test Póker (ÉXITO):**
   Chi²: 2.6162, p-valor: 0.8552
   Distribución de patrones aleatoria.

4. **Test Rachas (ÉXITO):**
   Rachas: 5358, Z: 0.6630
   Secuencia independiente.

Explicación de los Resultados del Generador Lineal Congruencial (GLC)

1. **Test de Frecuencia (Chi² = 0.0123, p-valor = 1.0000)**
   Chi² muy bajo (0.0123): Indica una distribución extremadamente uniforme de los dígitos.
   p-valor = 1.0000: Valor prácticamente perfecto. Cuando p >0.05, aceptamos la hipótesis de que los dígitos están uniformemente distribuidos.
   Los dígitos aparecen con frecuencias casi idénticas, exactamente como esperaríamos en números verdaderamente aleatorios.

2. **Test Chi Cuadrado (Chi² = 10.8825, p-valor = 0.2838)**
   Chi² moderado (10.8825): Valor razonable que indica que los números están bien distribuidos en diferentes intervalos.
   p-valor = 0.2838: Al ser mayor que 0.05, aceptamos la hipótesis de distribución uniforme.
   Los valores generados se distribuyen uniformemente en todo el rango posible.

3. **Test Póker (Chi² = 2.6162, p-valor = 0.8552)**
   Chi² bajo (2.6162): Sugiere que los patrones de dígitos están muy cerca de lo esperado teóricamente.
   p-valor = 0.8552: Valor muy alto, lo que indica fuerte evidencia de que los patrones de dígitos siguen lo esperado en una secuencia aleatoria.
   Los números generados muestran exactamente la distribución esperada de patrones de dígitos.

4. **Test Rachas (Rachas = 5358, Z = 0.6630)**
   Rachas = 5358: Número de cambios de tendencia en la secuencia.
   Z = 0.6630: El estadístico Z mide cuánto se desvía el número de rachas observado del esperado. Valores entre -1.96 y +1.96 son aceptables.
   El valor Z está bien dentro del rango aceptable, lo que indica que los números son independientes entre sí.

El Generador Lineal Congruencial con los parámetros utilizados produce una secuencia de alta calidad estadística que pasa todas las pruebas de aleatoriedad. Esto confirma que es un excelente generador para aplicaciones prácticas, a diferencia del método de cuadrados medios que fallaba en la mayoría de las pruebas.

5.3. Generador Python

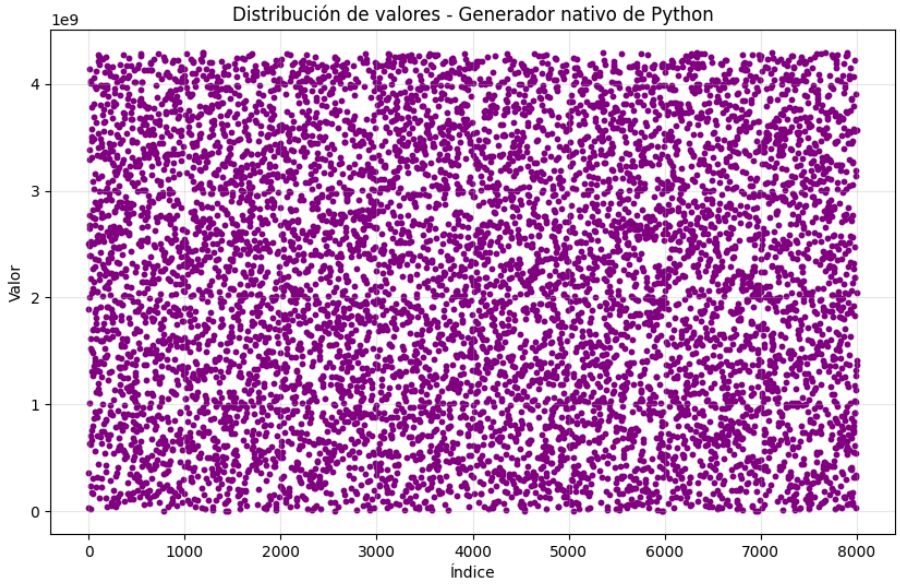

Figura 3: Distribución del Generador de Python

El gráfico muestra la distribución de los valores generados por el generador nativo de números pseudoaleatorios de Python. Lo que se visualiza:

- **Eje X (Índice):** Representa la posición de cada número en la secuencia generada (del 0 al 8000).
- **Eje Y (Valor):** Muestra el valor de cada número generado (de 0 a 4.3×10^9, aproximadamente 2^{32}-1).
- **Puntos violetas:** Cada punto representa un número aleatorio individual.

Características clave:

- Distribución uniforme: Los puntos están distribuidos de manera muy uniforme en todo el espacio vertical, lo que indica que el generador produce valores en todo el rango posible con igual probabilidad.
- Ausencia de patrones: No se observan patrones visibles, líneas, grupos o estructuras reconocibles, lo que sugiere una buena aleatoriedad.
- Cobertura completa: Los valores cubren todo el rango desde cerca de 0 hasta el máximo posible, sin "huecos"significativos.
- Independencia: No hay correlación visible entre valores consecutivos (los puntos adyacentes no muestran relación entre sí).

Esta visualización confirma por qué el generador nativo de Python pasó todas las pruebas estadísticas con éxito. Al usar el algoritmo Mersenne Twister, Python produce secuencias con excelentes propiedades estadísticas, muy superiores a los métodos más sencillos como Mid-Square que vimos antes, que mostraba ciclos repetitivos y patrones claros. Esta distribución ruidosa y uniforme es exactamente lo que queremos ver en un generador de números pseudoaleatorios de alta calidad.

RESULTADOS

1. **Test de Frecuencia (ÉXITO):**
   Chi²: 0.0139, p-valor: 1.0000
   Los dígitos siguen distribución uniforme

2. **Test Chi Cuadrado (ÉXITO)**:
   Chi²: 1.5775, p-valor: 0.9965
   Los valores siguen distribución uniforme

3. **Test Póker (ÉXITO):**
   Chi²: 2.4441, p-valor: 0.8747
   Distribución de patrones aleatoria

4. **Test Rachas (ÉXITO)**:
   Rachas: 5301, Z: -0.8486
   Secuencia independiente

Explicación de los Resultados del Generador Nativo Python

1. **Test de Frecuencia (Chi² = 0.0139, p-valor = 1.0000)**
   Este valor de Chi² extremadamente bajo indica que la distribución de dígitos individuales es casi perfectamente uniforme.
   El p-valor de 1.0000 significa que hay una probabilidad del 100 % de que esta distribución ocurra por azar en un generador realmente aleatorio.
   En otras palabras, la distribución de dígitos es estadísticamente indistinguible de una distribución ideal.

2. **Test Chi Cuadrado (Chi² = 1.5775, p-valor = 0.9965)**
   A diferencia del Mid-Square (que tenía un valor de 85.0000), el generador de Python tiene un Chi² muy bajo.
   El p-valor cercano a 1 indica que la distribución de los valores completos en todo el rango es extremadamente uniforme.
   Significa que los números generados cubren todo el espacio de valores posibles sin concentrarse en regiones específicas.

3. **Test Póker (Chi² = 2.4441, p-valor = 0.8747)**
   Nuevamente se obtiene un valor menor de Chi² en comparación con el Mid-Square, que tenía un Chi² de 7141.5013.
   Este valor bajo indica que los patrones de dígitos dentro de cada número siguen una distribución aleatoria.
   El p-valor de 0.8747 significa que hay una probabilidad del 87.47 % de que estos patrones ocurran por azar en un buen generador.
   No hay combinaciones de dígitos que aparezcan con frecuencias anormales.

4. **Test Rachas (Rachas = 5301, Z = -0.8486)**
   Z cercano a cero indica que el número de rachas está cerca del valor esperado teóricamente.
   Aunque el valor es ligeramente negativo (hay un poco menos de rachas de lo esperado), está dentro del rango aceptable.
   Esto confirma que no hay correlación significativa entre números consecutivos.

El generador nativo de Python (basado en Mersenne Twister) pasa todas las pruebas estadísticas con resultados excelentes, demostrando ser:

- Uniformemente distribuido tanto a nivel de dígitos individuales como valores completos.

- Libre de patrones predecibles entre dígitos.
- Capaz de producir valores consecutivos estadísticamente independientes.
- De calidad muy superior a métodos más sencillos como Mid-Square.

Estos resultados reflejan por qué el algoritmo Mersenne Twister de Python es ampliamente utilizado en aplicaciones científicas y de simulación donde la calidad de la aleatoriedad es crítica.

GENERADOR	SEMILLA	TEST FRECUENCIA	TEST CHI²	TEST PÓKER	TEST RACHAS
Mid-Square	9731	Éxito	Fallo	Fallo	Fallo
GLC	12345	Éxito	Éxito	Éxito	Éxito
Python	1234	Éxito	Éxito	Éxito	Éxito

Figura 4: Resumen de los resultados obtenidos

6. Conclusión

A lo largo de este trabajo se analizaron y compararon distintos generadores de números pseudoaleatorios con el objetivo de evaluar su idoneidad para aplicaciones en simulación. Se implementaron y estudiaron dos métodos clásicos: el Generador Congruencial Lineal (GLC) y el método de los Cuadrados Medios, además de contrastarlos con el generador nativo de Python basado en el algoritmo Mersenne Twister.

Los resultados obtenidos evidencian diferencias marcadas en la calidad estadística de los números generados por cada método. El generador Mid-Square presentó comportamientos claramente indeseables, como ciclos cortos y patrones repetitivos, lo cual quedó reflejado en su bajo rendimiento en la mayoría de los tests estadísticos aplicados, especialmente en el test de Chi Cuadrado, el test de Póker y el de Rachas. Estas deficiencias lo convierten en una opción poco confiable para tareas que exigen secuencias con apariencia verdaderamente aleatoria.

En contraste, el Generador Congruencial Lineal mostró un desempeño notablemente superior, pasando exitosamente todas las pruebas estadísticas con una distribución uniforme, independencia entre valores y ausencia de patrones visibles. Esto pone de relieve la importancia de una correcta selección de parámetros en este tipo de algoritmos, lo cual puede marcar la diferencia entre un generador útil y uno defectuoso, incluso dentro de una misma clase de métodos.

Finalmente, el generador por defecto de Python también superó todas las pruebas con éxito y mostró un excelente comportamiento visual y estadístico. Sin embargo, al tratarse de un generador pseudoaleatorio, mantiene la naturaleza determinista que caracteriza a todos los PRNG. Si bien sus resultados fueron satisfactorios, no se lo puede considerar como inherentemente superior sin tener en cuenta los posibles riesgos de predictibilidad en ciertos contextos.

En conclusión, este trabajo resalta no solo las diferencias entre algoritmos generadores, sino también la relevancia crítica de evaluar estadísticamente la calidad de los números generados antes de aplicarlos a modelos de simulación. Un generador que pase las pruebas adecuadas es esencial para garantizar resultados confiables y representativos del comportamiento aleatorio que se busca simular. Por estas razones, el generador nativo de Python basado en Mersenne Twister se destaca como el mejor generador evaluado en este estudio.

7. Referencias

- **Generadores Congruenciales Lineales.**
  Wikipedia. *Linear congruential generator.*
  `https://en.wikipedia.org/wiki/Linear_congruential_generator`
- **Método de los Cuadrados Medios (Mid-Square).**
  GeeksforGeeks. *Mid Square Method – Random Number Generation.*
  `https://www.geeksforgeeks.org/mid-square-method-random-number-generation/`
- **Test de Chi Cuadrado y otras pruebas estadísticas.**
  NIST (National Institute of Standards and Technology). *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications.*
  `https://csrc.nist.gov/publications/detail/sp/800-22/rev-1a/final`
- **Test de Póker, Corridas (Rachas) y Frecuencia.**
  JStatSoft. *A Review of Pseudorandom Number Generators.*
  `https://www.jstatsoft.org/article/view/v083i08`
- **Generador de Python (Mersenne Twister).**
  Python Documentation. *random — Generate pseudo-random numbers.*
  `https://docs.python.org/3/library/random.html`
- **Conceptos generales de generación y evaluación de números aleatorios.**
  Brilliant.org. *Pseudorandom Number Generators.*
  `https://brilliant.org/wiki/pseudorandom-number-generators/`
- **Visualización y ciclos en generadores.**
  Rosetta Code. *Random number generator (visual test).*
  `https://rosettacode.org/wiki/Random_number_generator_(visual_test)`