TRABAJO PRÁCTICO 2.1 - GENERADORES PSEUDOALEATORIOS

Danteo, Elías
Universidad Tecnológica Nacional - FRRO
Zeballos 1341, S2000, Argentina
elias.danteo.tomas@hotmail.com

Cosentino, Lucio N.
Universidad Tecnológica Nacional - FRRO
Zeballos 1341, S2000, Argentina
luciocosen@gmail.com

De Bernardo, Aarón
Universidad Tecnológica Nacional - FRRO
Zeballos 1341, S2000, Argentina
aarondebernardo@gmail.com

Fernandez Da Silva, Joaquín C.
Universidad Tecnológica Nacional - FRRO
Zeballos 1341, S2000, Argentina
joaquinfds13@gmail.com

Malizani, Juan Pablo
Universidad Tecnológica Nacional - FRRO
Zeballos 1341, S2000, Argentina
juampi123.m@gmail.com

Pastorino, Juan Jose
Universidad Tecnológica Nacional - FRRO
Zeballos 1341, S2000, Argentina
juanjosepastorino@gmail.com

May 13, 2025

ABSTRACT

El presente trabajo tiene por objetivo adentrarse en el estudio y la evaluación de generadores de números pseudoaleatorios, elementos esenciales para la simulación computacional. Se implementarán en Python 3.x cuatro métodos de generación, y se someterán los mismos a un conjunto de cinco pruebas estadísticas para medir uniformidad e independencia de los números generados. Los resultados obtenidos se compararán con el generador nativo de Python, presentándose mediante tablas y gráficos que faciliten el análisis cuantitativo. Finalmente, se extraerán conclusiones sobre las fortalezas y limitaciones de cada algoritmo en contextos de simulación.

***K*eywords** Simulación · Generadores pseudoaleatorios · GCL · Pruebas estadísticas

1 Introducción

La generación de números pseudoaleatorios es un pilar fundamental en multitud de disciplinas científicas y de ingeniería, que van desde la simulación Monte Carlo hasta la criptografía y el modelado estocástico. A diferencia de los procesos de azar físicos, los generadores computacionales se basan en algoritmos deterministas que, partiendo de una semilla inicial, producen secuencias de números que deben "parecer aleatorias" desde un punto de vista estadístico. Sin embargo, el determinismo inherente de estos procedimientos implica que, sin un análisis riguroso, pueden surgir sesgos, correlaciones ocultas o periodos de repetición que comprometan la validez de cualquier experimento que dependa de ellos.

En la literatura se han propuesto múltiples métodos de generación (desde el histórico Método de los Cuadrados Medios hasta las familias de Generadores de Congruencia Lineal (GCL) y técnicas más modernas basadas en transformaciones de grandes estados internos), cada uno con sus ventajas y limitaciones. Para evaluar la calidad de un generador, es imprescindible aplicar baterías de pruebas estadísticas (por ejemplo, chi-cuadrado, runs test, gap test y poker test) que midan propiedades como la uniformidad de la distribución, la independencia entre valores consecutivos y la longitud del período antes de la repetición de la secuencia. Sólo a partir de estos tests es posible garantizar que un generador sea apto para aplicaciones críticas, donde una ligera desviación puede traducirse en errores significativos o vulnerabilidades de seguridad. El análisis incluirá un estudio detallado de métricas de interés como la longitud de periodo antes de la repetición, los valores p y estadísticos obtenidos en cada prueba, las posibles correlaciones entre valores sucesivos y el

rendimiento computacional (tiempo de generación por número). Este enfoque no solo permitirá obtener una visión más clara sobre la robustez y fiabilidad de cada algoritmo, sino que también proporcionará una evaluación crítica de su desempeño en un contexto práctico de simulación, considerando restricciones reales como la eficiencia de cómputo y los requisitos de reproducibilidad.

2 Marco Teórico

2.1 Generador de Medios Cuadrados

El método de los medios cuadrados consiste en tomar una semilla inicial x_0, elevarla al cuadrado y extraer los dígitos centrales del resultado para formar el siguiente número pseudoaleatorio x_1. El proceso se repite para generar una secuencia.

Algoritmo:

1. Elegir una semilla x_0 de d dígitos.
2. Calcular x_0^2, que tendrá hasta $2d$ dígitos.
3. Extraer los d dígitos centrales del resultado para formar x_1.
4. Repetir el proceso: $x_{n+1} = \text{MediosCuadrados}(x_n)$.

Ejemplo: Sea la semilla $x_0 = 5731$. Entonces:

$$x_0^2 = 5731^2 = 32845261$$

Extraemos los 4 dígitos centrales: **32845261** $\rightarrow$ **8452**, entonces:

$$x_1 = 8452$$

Y repetimos el proceso.

Observaciones:

- El método puede entrar en ciclos o llegar a 0, terminando la secuencia.
- Su calidad estadística es baja comparada con generadores modernos.

2.2 Generador Lineal Congruencial

El generador lineal lineal (GCL) es uno de los métodos más comunes para generar secuencias de números pseudoaleatorios. Su fórmula general es:

$$x_{n+1} = (ax_n + c) \mod m$$

donde:

- x_0: semilla inicial (valor de partida),
- a: multiplicador,
- c: incremento,
- m: módulo,
- x_n: valor actual,
- x_{n+1}: siguiente número en la secuencia.

Parámetros adecuados: La calidad del generador depende de la elección adecuada de los parámetros. Para obtener un *período máximo* (de longitud m), se deben cumplir las siguientes condiciones (criterio de Hull-Dobell):

1. c y m son primos relativos,
2. $a - 1$ es divisible por todos los factores primos de m,
3. Si m es múltiplo de 4, entonces $a - 1$ también debe serlo.

Ejemplo: Sea $a = 25214903917$, $c = 11$, $m = 2^{48}$ y semilla $x_0 = 1532$. Entonces:

$$x_1 = (25214903917 \cdot 1532 + 11) \mod 2^{48}$$

Y así sucesivamente para generar nuevos valores.

Observaciones:

- Es rápido y fácil de implementar.
- Su calidad estadística puede ser aceptable si se eligen bien los parámetros.
- Muchos lenguajes de programación implementan variantes del GCL.

2.3 Generador Cuadrático Congruencial

El generador cuadrático congruencial (GCC) es una extensión del generador lineal congruencial, donde se incluye un término cuadrático. Su fórmula general es:

$$x_{n+1} = (ax_n^2 + bx_n + c) \mod m$$

donde:

- x_0: semilla inicial,
- a, b, c: coeficientes del generador,
- m: módulo,
- x_n: valor actual de la secuencia,
- x_{n+1}: siguiente número pseudoaleatorio.

Características:

- Es más complejo que el generador lineal, y potencialmente produce secuencias con mejor aleatoriedad.
- Su estudio teórico es más difícil, y no existen criterios tan claros como el de Hull-Dobell para garantizar un periodo máximo.
- Menos común en la práctica, pero útil para fines académicos y comparativos.

2.4 Generador mixto

En lugar de recurrir a un único mecanismo de generación (lineal, multiplicativo o cuadrático), el **generador mixto** combina los métodos vistos previamente (GCL, GCC y el generador de Python) con el fin de explotar las virtudes de cada uno y atenuar sus debilidades. Excluimos el generador de medios cuadrados por su corto periodo y la alta dependencia de los dígitos centrales. El generador realiza los siguientes pasos:

- Generar un número x_1 mediante el generador de Python.

- Generar el siguiente número x_2 mediante el GCC, utilizando la semilla:

$$\text{semilla}_{\text{GCC}} = \texttt{int}(x_1 \times 10000)$$

- Generar el siguiente número x_3 mediante el GCL, utilizando la semilla:

$$\text{semilla}_{\text{GCL}} = \texttt{int}(x_2 \times 10000)$$

Nota: Para el cuarto número, el generador vuelve al paso 1 y continúa sucesivamente.

2.5 Test de Poker

El **test de poker** evalúa la aleatoriedad de una secuencia de números pseudoaleatorios mediante el análisis de patrones en grupos de k dígitos decimales, inspirados en las combinaciones clásicas del juego de póker. Esta prueba permite evaluar tanto la uniformidad como la independencia de los dígitos.

Hipótesis:

- H_0: Los dígitos decimales están distribuidos uniformemente y son independientes.
- H_1: Los dígitos presentan desviaciones significativas respecto de la uniformidad o independencia.

Procedimiento:

1. **Preparación de secuencias:**
   - Para cada número $x_i \in [0, 1)$, se extraen los k primeros dígitos decimales después del punto para formar una secuencia de longitud k.
   - Ejemplo: Para $x_i = 0.12345$ y $k = 5$, la secuencia resultante es `"12345"`.
   - Se recomienda trabajar con dígitos decimales (0–9) y no con los valores reales completos.
2. **Clasificación de patrones:** Contar las ocurrencias de cada categoría en todas las secuencias:
   - **Todos diferentes**: Ej. `1 3 5 2 4` (sin repeticiones).
   - **Un par**: Ej. `1 3 3 2 4` (exactamente un par).
   - **Dos pares**: Ej. `1 3 3 2 2` (dos pares distintos).
   - **Tercia**: Ej. `1 3 3 3 2` (tres dígitos iguales).
   - **Full**: Ej. `1 3 3 3 1` (una tercia y un par).
   - **Póker**: Ej. `1 3 3 3 3` (cuatro dígitos iguales).
   - **Quintilla**: Ej. `3 3 3 3 3` (todos iguales).
3. **Frecuencias esperadas:** Para $k = 5$, las probabilidades teóricas bajo H_0 (asumiendo dígitos uniformemente distribuidos en [0–9]) son:

$$\begin{aligned} P(\text{Todos diferentes}) &= 0.3024, \\ P(\text{Un par}) &= 0.5040, \\ P(\text{Dos pares}) &= 0.1080, \\ P(\text{Tercia}) &= 0.0720, \\ P(\text{Full}) &= 0.0090, \\ P(\text{Póker}) &= 0.0045, \\ P(\text{Quintilla}) &= 0.0001. \end{aligned}$$

   Las frecuencias esperadas E_i se calculan como $E_i = n \times P_i$, donde n es el número total de secuencias.
4. **Estadístico de prueba:**

$$\chi^2 = \sum_{i=1}^{7} \frac{(O_i - E_i)^2}{E_i},$$

   donde O_i representa la frecuencia observada de cada patrón.
5. **Decisión:** Comparar el valor calculado de χ^2 con el valor crítico $\chi^2_{\alpha,6}$ (para $k = 5$, los grados de libertad son 6). Si $\chi^2 \geq \chi^2_{\alpha,6}$, se rechaza H_0.

Implementación con `scipy.stats.chi2`:

- Calcular el valor crítico y el p-valor usando:

```
from scipy.stats import chi2
critical_value = chi2.ppf(1 - alpha, df=6)
p_value = 1 - chi2.cdf(chi_squared, df=6)
```

- Rechazar H_0 si el p-valor $< \alpha$ (por ejemplo, $\alpha = 0.05$).

Ejemplo de aplicación:

Para una muestra de $n = 10{,}000$ números con $k = 5$:

Patrón	**Observado**	**Esperado**	**Contribución χ^2**
Todos diferentes	3,012	3,024	0.047
Un par	5,050	5,040	0.020
Dos pares	1,095	1,080	0.208
Tercia	730	720	0.139
Full	85	90	0.278
Póker	48	45	0.200
Quintilla	0	1	1.000

Table 1: Resultados del test de poker para $n = 10{,}000$.

$$\chi^2 = 1.892 \quad (\chi^2_{0.05,6} = 12.592) \quad \Rightarrow \text{No se rechaza } H_0.$$

Recomendaciones:

- Usar $k = 4$ o $k = 5$ para equilibrar sensibilidad y robustez.
- Verificar que $n \times P_i \geq 5$ para todas las categorías; de lo contrario, combinar clases con frecuencias bajas (por ejemplo, póker y quintilla).
- Asegurarse de no contar secuencias duplicadas en exceso, especialmente si los números pseudoaleatorios no son independientes entre sí.

2.6 Test de Frecuencia Chi-cuadrado

El **test de frecuencia** (también conocido como **test Chi-cuadrado de uniformidad**) se emplea para evaluar si una secuencia de números pseudoaleatorios está distribuida de manera uniforme en el intervalo $[0, 1)$. Para ello, se divide el intervalo en k subintervalos de igual tamaño y se cuenta cuántos números caen en cada uno.

Hipótesis:

- H_0: Los números están distribuidos uniformemente en $[0, 1)$.
- H_1: Los números no siguen una distribución uniforme.

Procedimiento:

1. Se divide el intervalo $[0, 1)$ en k subintervalos del mismo tamaño.
2. Se cuenta cuántos valores caen en cada subintervalo (frecuencias observadas O_i).
3. Se calcula la frecuencia esperada $E = \frac{n}{k}$, siendo n la cantidad total de números generados.
4. Se calcula la estadística de prueba:
$$\chi^2 = \sum_{i=1}^{k} \frac{(O_i - E)^2}{E}$$
5. Se compara el valor obtenido con el valor crítico $\chi^2_{\text{crítico}}$ correspondiente a $k - 1$ grados de libertad y un nivel de significancia α (usualmente 0.05).

Criterio de decisión:

- Si $\chi^2 < \chi^2_{\text{crítico}}$, no se rechaza H_0 y se considera que la distribución es uniforme.
- Si $\chi^2 \geq \chi^2_{\text{crítico}}$, se rechaza H_0.

Nota: En el código utilizado, el valor crítico de χ^2 se selecciona manualmente desde una tabla predefinida para los grados de libertad entre 1 y 10. Si el número de intervalos k excede ese rango, el test no se ejecuta.

Determinación del tamaño de muestra:

Para garantizar que la estimación de la media muestral $\bar{X}$ se ubique dentro de un margen de error E con un nivel de confianza $1-\alpha$, se recurre al teorema central del límite, que establece

$$\bar{X} \;\sim\; N\Big(\mu,\; \sigma^2/n\Big)$$

cuando n es suficientemente grande (≥ 50). En el caso de una distribución uniforme $U(0,1)$ se tiene

$$\mu = 0.5, \quad \sigma^2 = \frac{1}{12}, \quad \sigma = \sqrt{\frac{1}{12}}.$$

El intervalo de confianza bilateral para la media es

$$\Pr\Big(\,|\bar{X}-\mu| < E\Big) = 1-\alpha \quad \Longrightarrow \quad z_{\alpha/2}\,\frac{\sigma}{\sqrt{n}} = E$$

de donde se despeja

$$n \;=\; \Big(\frac{z_{\alpha/2}\,\sigma}{E}\Big)^2 \;=\; \left(\frac{z_{\alpha/2}\,\sqrt{\frac{1}{12}}}{E}\right)^2 .$$

Elección práctica de n:

Tomaremos un nivel de confianza del 95% $\big(\alpha = 0.05,\; z_{0.025} \approx 1.96\big)$ y un margen de error $E = 0.01$,

$$n = \left(\frac{1.96 \cdot \sqrt{1/12}}{0.01}\right)^2 \approx 3201.33 \approx 3202$$

De este modo, se justifica la elección de $n = 3202$ muestras para asegurar que la media observada difiera de 0.5 en menos de ±0.01 con un 95% de confianza.

2.7 Test de Independencia de Corridas (Runs Test)

El **test de independencia de corridas** evalúa la **independencia** de los valores generados, verificando si hay patrones sistemáticos por encima o por debajo de un valor de referencia. En esta implementación, se utiliza la **mediana empírica** de la muestra como umbral de referencia.

Hipótesis:

- H_0: La secuencia es aleatoria e independiente.
- H_1: La secuencia presenta dependencia o patrón.

Procedimiento:

1. Se calcula la mediana de la secuencia generada.
2. Se asigna un signo binario a cada número:

$$\text{signo}(x_i) = \begin{cases} 1 & \text{si } x_i \geq \text{mediana} \\ 0 & \text{si } x_i < \text{mediana} \end{cases}$$

3. Se cuenta el número de **corridas** R, es decir, secuencias consecutivas de valores con el mismo signo.

4. Se calcula el número de unos (n_1) y ceros (n_2) en la secuencia binaria.
5. Se calcula el número esperado de corridas:

$$\mu = \frac{2n_1n_2}{n} + 1$$

   Donde $n = n_1 + n_2$.
6. Se calcula la varianza:

$$\sigma^2 = \frac{2n_1n_2(2n_1n_2 - n)}{n^2(n-1)}$$

7. Se obtiene la estadística Z:

$$Z = \frac{R - \mu}{\sqrt{\sigma^2}}$$

8. Se compara el valor absoluto de Z con el valor crítico correspondiente al nivel de confianza deseado ($\alpha = 0.05$ implica $Z_{0.025} = 1.96$).

Criterio de decisión:

- Si $|Z| < 1.96$, no se rechaza H_0 y la secuencia se considera aleatoria.
- Si $|Z| \geq 1.96$, se rechaza H_0.

Determinación del tamaño de muestra:

Para garantizar que la estimación de la media muestral $\bar{X}$ se ubique dentro de un margen de error E con un nivel de confianza $1 - \alpha$, se recurre al teorema central del límite, que establece

$$\bar{X} \ \sim \ N\Big(\mu,\ \sigma^2/n\Big)$$

cuando n es suficientemente grande (≥ 50). En el caso de una distribución uniforme $U(0, 1)$ se tiene

$$\mu = 0.5, \quad \sigma^2 = \frac{1}{12}, \quad \sigma = \sqrt{\frac{1}{12}}.$$

El intervalo de confianza bilateral para la media es

$$\Pr\Big(\ |\bar{X} - \mu| < E\Big) = 1 - \alpha \quad \Longrightarrow \quad z_{\alpha/2}\,\frac{\sigma}{\sqrt{n}} = E$$

de donde se despeja

$$n \ = \ \Big(\frac{z_{\alpha/2}\,\sigma}{E}\Big)^2 \ = \ \left(\frac{z_{\alpha/2}\,\sqrt{\frac{1}{12}}}{E}\right)^2 .$$

Elección práctica de n:

Tomaremos un nivel de confianza del 95% $\big(\alpha = 0.05,\ z_{0.025} \approx 1.96\big)$ y un margen de error $E = 0.01$,

$$n = \left(\frac{1.96 \cdot \sqrt{1/12}}{0.01}\right)^2 \approx (56.635)^2 \approx 3206.6.$$

De este modo, se justifica la elección de $n = 3207$ muestras para asegurar que la media observada difiera de 0.5 en menos de ±0.01 con un 95% de confianza.

2.8 Prueba de Arreglos Inversos (Reverse Arrangements Test)

La **prueba de ordenaciones inversas** funciona contando cuántas veces un valor en una serie es mayor que un valor que aparece más adelante (es decir, cuántas "inversiones" hay respecto al orden cronológico). Si la serie es **aleatoria**, se espera un número moderado de estas inversiones; si hay muy pocas, indica una posible tendencia creciente, y si hay muchas, una tendencia decreciente. Comparando el número total de inversiones con su valor esperado bajo aleatoriedad, se puede determinar estadísticamente si hay una tendencia significativa en los datos.

Hipótesis:

- H_0: La secuencia es aleatoria.
- H_1: La secuencia presenta una tendencia creciente o decreciente.

Procedimiento:

1. **Datos de entrada:**
   Se tiene una muestra de tamaño n representada por:
   $$X = \{X_1, X_2, \ldots, X_n\}$$
2. **Cálculo del número de ordenaciones inversas A:**
   Se cuenta cuántas veces un valor que aparece antes en la serie es mayor que uno que aparece después. Formalmente:
   $$A = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} I(X_i > X_j)$$
   donde $I(\cdot)$ es la función indicadora que vale 1 si la condición se cumple y 0 en caso contrario.
3. **Valor esperado de A bajo la hipótesis nula H_0:**
   $$E(A) = \frac{n(n-1)}{4}$$
4. **Varianza de A bajo H_0:**
   $$\text{Var}(A) = \frac{n(n-1)(2n+5)}{72}$$
5. **Cálculo del estadístico de prueba Z:**
   Para muestras grandes ($n > 10$ aproximadamente), se utiliza la aproximación normal:
   $$Z = \frac{A - E(A)}{\sqrt{\text{Var}(A)}}$$

Criterio de decisión:

Comparar el valor de Z con los valores críticos de la distribución normal estándar $N(0, 1)$:

- Si $|Z| < Z_{\alpha/2}$, no se rechaza H_0: la serie es aleatoria.
- Si $Z < -Z_{\alpha/2}$, se sospecha una **tendencia creciente**.
- Si $Z > Z_{\alpha/2}$, se sospecha una **tendencia decreciente**.

Determinación del tamaño de muestra:

A diferencia de los tests basados en la estimación puntual de parámetros, en este test no existe una relación cerrada y directa entre el tamaño de muestra n y un margen de error E en unidades absolutas de número de inversiones. El estadístico de conteo
$$A = \sum_{i<j} I(X_i > X_j)$$
tiene esperanza
$$E(A) = \frac{n(n-1)}{4}$$
y varianza
$$\text{Var}(A) = \frac{n(n-1)(2n+5)}{72},$$
pero despejar n de la desigualdad
$$E \geq z_{\alpha/2} \sqrt{\text{Var}(A)}$$

no conduce a una fórmula analítica sencilla, sino a una ecuación cúbica que debe resolverse numéricamente. Por esa razón, en la práctica basta con garantizar un n lo suficientemente grande (por ejemplo, $n \geq 10$) para que la aproximación normal de

$$Z = \frac{A - E(A)}{\sqrt{\mathrm{Var}(A)}}$$

sea válida, y circunscribir el tamaño de muestra según criterios de potencia o simulación, más que con una fórmula directa de tipo $n = (z\,\sigma/E)^2$.

Elección práctica de n:

No existe un valor "óptimo" universal para n en este test, ya que la potencia depende de la magnitud de la tendencia buscada y del nivel de significancia deseado. Sin embargo, se recomiendan las siguientes guías de referencia:

- $n < 10$: aproximación normal poco fiable, mejor usar tablas exactas o test alternativos.
- $10 \leq n < 30$: válida para tendencias fuertes, pero potencia limitada.
- $n \geq 30$: aproximación normal razonable, potencia moderada.
- $n \geq 50$: aproximación muy buena, potencia adecuada para detectar tendencias suaves.

Como ejemplo práctico, elegir $n = 50$ muestras garantiza que el estadístico siga casi exactamente una normal y proporciona un poder suficiente en la mayoría de los estudios de tendencia.

2.9 Prueba de sumas solapadas (Overlapping Sums Test)

La prueba de sumas solapadas evalúa la independencia y uniformidad de una secuencia de $U(0,1)$ al considerar sumas de bloques consecutivos de longitud m solapados. Cada suma

$$S_i = \sum_{j=i}^{i+m-1} X_j$$

sigue bajo H_0 la distribución Irwin–Hall de orden m. Comparando la frecuencia observada de dichas sumas en intervalos predefinidos contra la frecuencia esperada, detecta dependencias locales.

Hipótesis:

$H_0 : \{X_i\}$ es una secuencia i.i.d. $U(0,1)$,
$H_1 : \{X_i\}$ no es i.i.d. $U(0,1)$ (hay dependencia o no uniformidad).

Procedimiento:

1. Elegir un tamaño de bloque m (por ejemplo $m = 5$).
2. Para $i = 1, \ldots, n - m + 1$, calcular
$$S_i = \sum_{j=i}^{i+m-1} X_j.$$
3. Dividir el rango posible de S_i (de 0 a m) en k intervalos $\{I_1, \ldots, I_k\}$.
4. Contar $O_\ell = \#\{i : S_i \in I_\ell\}$, $\ell = 1, \ldots, k$.
5. Bajo H_0, la probabilidad teórica de caer en I_ℓ es $p_\ell = \displaystyle\int_{I_\ell} f_S(s)\,ds$, donde f_S es la densidad Irwin–Hall.
6. Calcular el estadístico
$$\chi^2 = \sum_{\ell=1}^{k} \frac{(O_\ell - (n-m+1)p_\ell)^2}{(n-m+1)p_\ell}.$$

Criterio de decisión:

Rechazar H_0 si

$$\chi^2 > \chi^2_{k-1,\,1-\alpha},$$

donde $\chi^2_{k-1,\,1-\alpha}$ es el cuantil de la χ^2 con $k-1$ grados de libertad al nivel $1-\alpha$.

Determinación del tamaño de muestra:

No existe una fórmula cerrada que relacione directamente n con un margen de error en este test. El poder depende de m, k, la magnitud de la dependencia y el nivel α. En la práctica:

- Usar $n - m + 1 \geq 100$ para que la aproximación χ^2 sea válida.
- Elegir m y k moderate (p.ej. $m = 5$, $k = 10$) y simular el poder para tendencias esperadas.

Ejemplo de elección práctica:

Para bloques de $m = 5$ y $k = 10$ intervalos, tomar $n = 200$ (de modo que haya 196 sumas solapadas) suele ser suficiente para detectar dependencias moderadas con $\alpha = 0.05$.

3 Implementación

En esta sección se describe la implementación de los generadores de números pseudoaleatorios y las pruebas estadísticas utilizadas para evaluar su aleatoriedad.

- **Lenguaje:** Python 3.10.
- **Bibliotecas utilizadas:**
  - `scipy.stats` para realizar pruebas estadísticas de chi-cuadrado y Kolmogorov-Smirnov.
  - `matplotlib` para generar gráficos de los resultados.
- **Generadores de números pseudoaleatorios:**
  - **Generador Lineal Congruencial:** parámetros utilizados: $a = 25214903917$, $c = 11$, $m = 2^{48}$, $semilla = 1532$.
  - **Generador Cuadrático Congruencial:** parámetros utilizados: $a = 1103515245$, $b = 0$, $c = 12345$, $m = 2^{31} - 1$, $semilla = 795489$.
  - **Generador de Medios Cuadrados:** la semilla utilizada será 9731 debido a que la misma asegura un número elevado de iteraciones antes de que el generador entre en un bucle.
  - **Generador Mixto:** $semilla = 3849$. Los generadores GCL y GCC serán utilizados con los parámetros definidos anteriormente, a excepción de la semilla que será establecida como se explicó en el marco teórico.
  - **Generador de Python:** $semilla = 5487$.

  Nota: las semillas fueron fijadas para fines de reproducibilidad del experimento.
- **Pruebas estadísticas realizadas:**
  - **Test de Frecuencia (Chi-cuadrado)**
  - **Test de Independencia de Corridas**
  - **Test de Arreglos Inversos**
  - **Test de Sumas Solapadas**
  - **Test de Poker**
- **Código disponible en:** `https://github.com/Luciocos/SimulacionTPs`

4 Resultados y Análisis

4.1 Evaluación del Generador de Medios Cuadrados

La siguiente evaluación se realizó a partir del análisis visual del gráfico de dispersión generado por el **método de los medios cuadrados**. En este gráfico, el eje horizontal representa el **índice** de cada número generado, y el eje vertical muestra los **valores generados normalizados** entre 0 y 1.

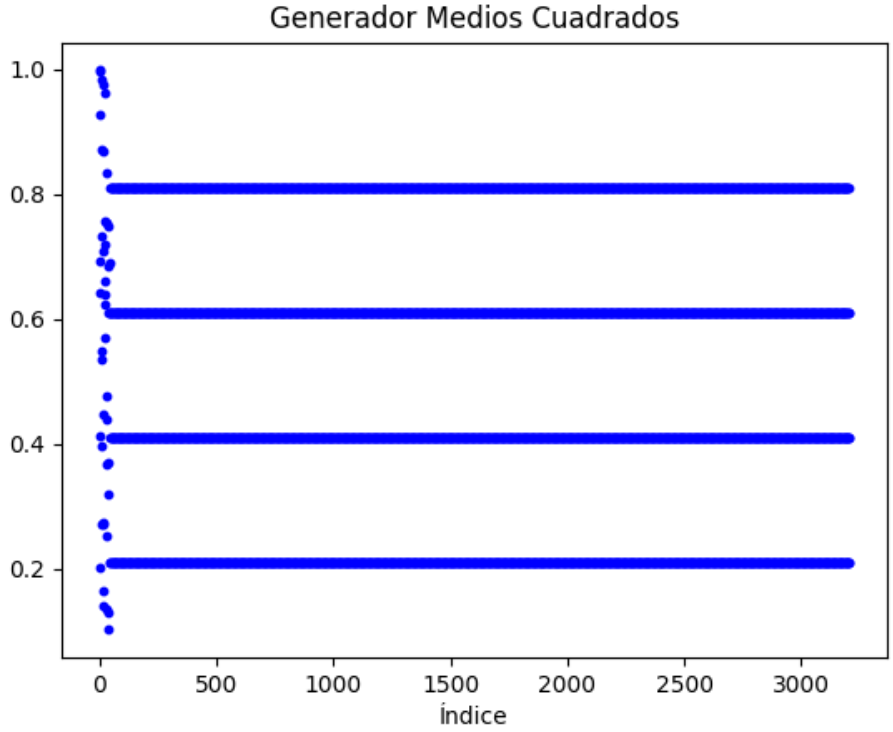

Figure 1: Generador Lineal Congruencial

- **Patrón repetitivo y ciclo corto:** los puntos se agrupan en líneas horizontales bien definidas, lo cual indica que el generador entra en ciclos de repetición rápidamente. Esto sugiere un **periodo muy corto**.
- **Falta de aleatoriedad:** un generador pseudoaleatorio ideal debería generar valores con apariencia caótica y sin patrones visibles. La estructura observada muestra una **ausencia de variabilidad aleatoria suficiente**.
- **Alta dependencia de la semilla:** el comportamiento del generador varía fuertemente con el valor inicial. En muchos casos, puede degenerar rápidamente en ciclos o en valores nulos constantes.

En conclusión, el generador de medios cuadrados presenta deficiencias notables y no es adecuado para aplicaciones modernas de simulación, ya que:

- Tiene ciclos extremadamente cortos.
- Genera secuencias con patrones evidentes.
- No cumple con los requisitos estadísticos de uniformidad y aleatoriedad.

Por estas razones, en la actualidad se recomienda el uso de generadores más robustos como el *Mersenne Twister*, que es el utilizado por defecto en Python mediante el módulo `random`.

4.2 Evaluación del Generador Lineal Congruencial

Se analizó la secuencia generada por el **método congruencial lineal**, observando la dispersión de 3207 valores normalizados entre 0 y 1. Este generador es uno de los más antiguos y ampliamente utilizados por su simplicidad.

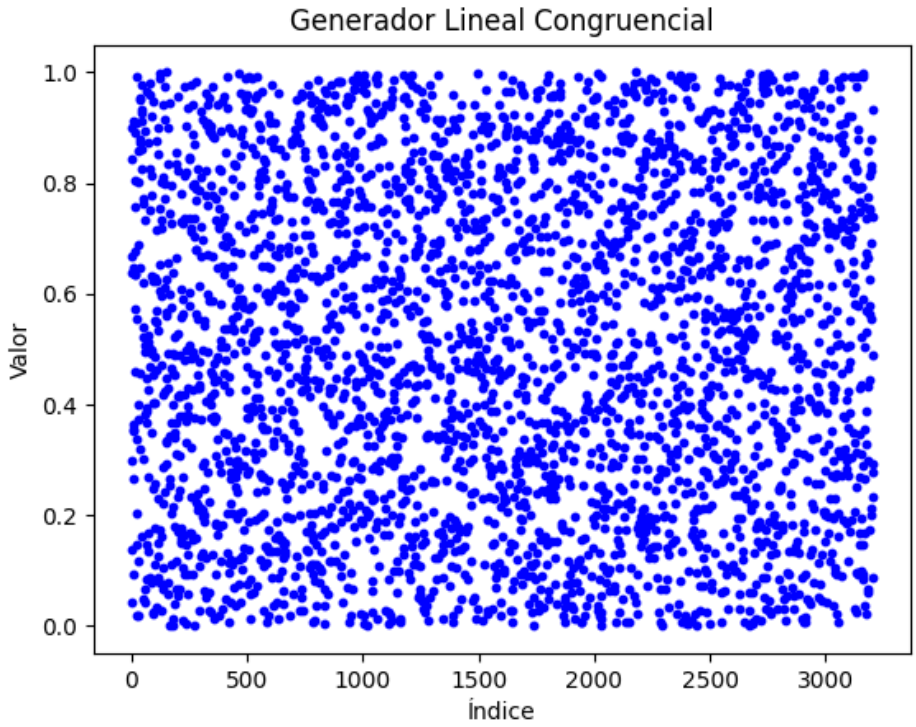

Figure 2: Generador Lineal Congruencial

- **Presencia de estructuras lineales:** se observan agrupaciones en líneas rectas oblicuas, lo que indica una dependencia entre valores sucesivos.

- **Distribución parcialmente uniforme:** aunque hay una cierta dispersión, la falta de un patrón totalmente caótico reduce la calidad de la aleatoriedad.

- **Periodos relativamente largos:** dependiendo de los parámetros elegidos (modulo, multiplicador y constante), puede generar secuencias largas antes de repetirse, pero no siempre garantiza independencia estadística.

En conclusión, aunque útil en contextos didácticos, el generador congruencial lineal presenta deficiencias frente a estándares modernos de aleatoriedad. Puede ser útil para simulaciones simples, pero no para aplicaciones críticas como criptografía o simulación estadística precisa.

4.3 Evaluación del Generador Cuadrático Congruencial

Este generador extiende el modelo lineal incorporando un término cuadrático. Se analizaron 3207 valores normalizados y graficados.

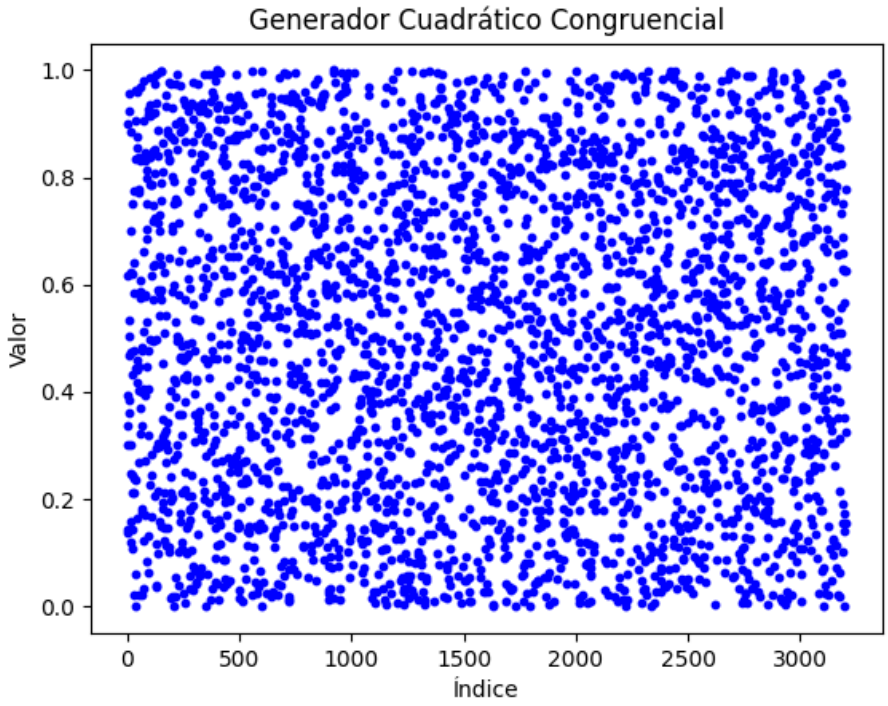

Figure 3: Generador Cuadrático Congruencial

- **Comportamiento caótico inicial:** en las primeras etapas se observa una mayor dispersión de los valores generados.

- **Tendencia a la degeneración:** con frecuencia, los valores convergen hacia ciertos puntos o patrones periódicos, perdiendo diversidad rápidamente.

- **Alta sensibilidad a parámetros:** pequeños cambios en los coeficientes pueden alterar completamente el comportamiento del generador.

En conclusión, aunque más complejo que el modelo lineal, el generador cuadrático no garantiza una mejora sustancial en términos de aleatoriedad. Es propenso a ciclos cortos y pérdida de uniformidad, por lo que no se recomienda para aplicaciones prácticas.

4.4 Evaluación del Generador Mixto

Este generador combina los generadores explicados anteriormente. Se analizaron 3207 valores normalizados y graficados.

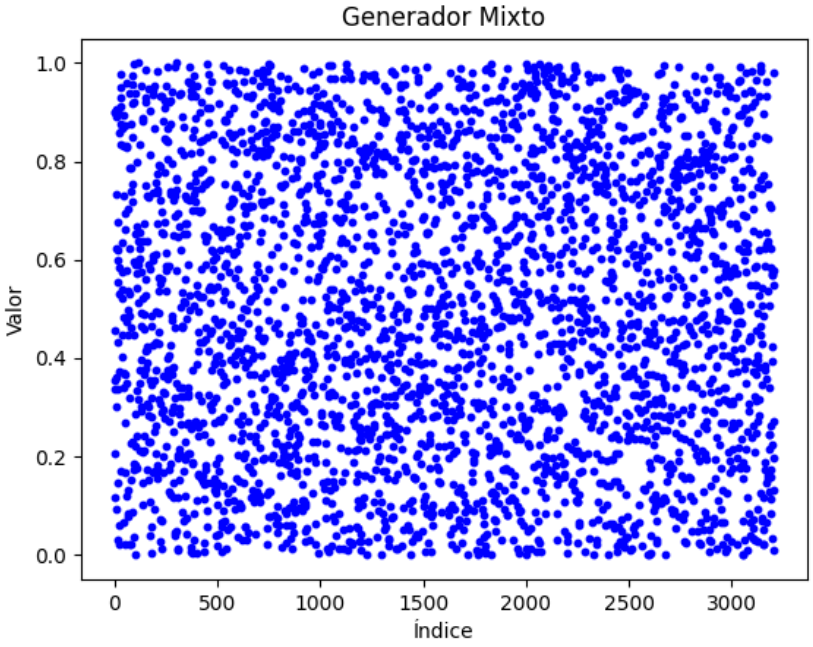

Figure 4: Generador Mixto

- **Eliminación de estructuras lineales visibles:** a diferencia del generador lineal congruencial (Figura 2), en el diagrama del generador mixto no se aprecian líneas oblicuas o agrupaciones sobre rectas; esto indica que la combinación de GCC y LCG rompe buena parte de la dependencia lineal entre valores sucesivos.

- **Cobertura más homogénea del espacio 0,10,10,1:** los puntos se distribuyen con mayor uniformidad, sin zonas claramente despobladas ni sobrepobladas, lo cual sugiere una mejora en la calidad estadística de la aleatoriedad.

- **Ausencia de patrones periódicos a simple vista:** al superponer dos secuencias de períodos distintos, no se distinguen ciclos cortos ni repeticiones evidentes en la nube de puntos. Esto es consistente con un período efectivo mucho mayor, idealmente cercano al mínimo común múltiplo de los dos períodos base.

- **Ligera heterogeneidad residual:** aunque la distribución es notablemente más uniforme, es posible observar pequeñas variaciones locales en densidad (típicas de secuencias pseudoaleatorias) que podrían someterse a pruebas estadísticas (TestU01, Dieharder) para cuantificar su independencia.

En conclusion, el generador mixto combina con éxito las virtudes del GCC y del LCG, suprimiendo la linealidad y los ciclos cortos de cada uno por separado y ofreciendo una secuencia con mejor dispersión y periodo ampliado, adecuada para simulaciones que requieran mayor robustez en la generación de números pseudoaleatorios.

4.5 Evaluación del Generador de Python (Mersenne Twister)

Este generador es el utilizado por defecto en el módulo `random` de Python y se basa en el algoritmo *Mersenne Twister*.

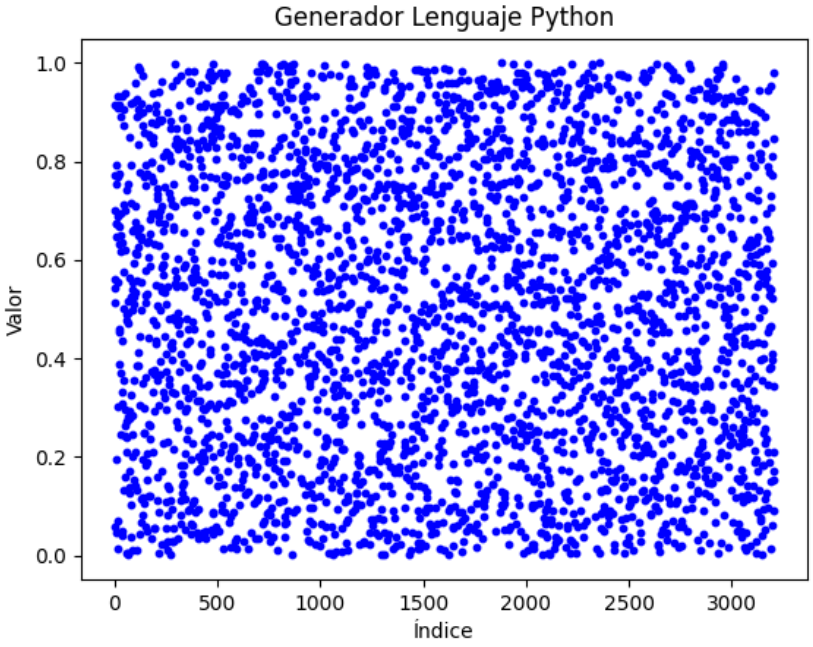

Figure 5: Generador del Lenguaje Python

- **Distribución uniforme visual:** la dispersión de los valores generados no muestra patrones evidentes, lo que es indicativo de una buena calidad estadística.
- **Ausencia de agrupamientos:** no se observan líneas o curvas dentro del gráfico, lo que implica una independencia significativa entre valores consecutivos.
- **Alta calidad para simulaciones:** este generador cumple con los estándares modernos de aleatoriedad y es ampliamente aceptado en la comunidad científica.

En conclusión, el generador de Python demuestra ser el más robusto de los analizados, con un comportamiento cercano al ideal para simulaciones, análisis estadísticos y otros usos generales, aunque no está diseñado específicamente para criptografía.

4.6 Resultados de las Pruebas Estadísticas

A continuación se presentan los resultados de las pruebas estadísticas realizadas sobre los diferentes generadores analizados:

4.6.1 Generador de Medios Cuadrados

- **Test de Frecuencia (Chi-cuadrado):**
  - Chi-cuadrado calculado: 4691.3879
  - Valor crítico ($\alpha = 0.05$): 16.9190
  - ¿Pasa el test? No
  - Explicación: el valor calculado supera el crítico; la distribución no es uniforme.
- **Test de Independencia de Corridas:**
  - Corridas observadas: 1606
  - Corridas esperadas: 1604.50
  - ¿Pasa el test? Sí
  - Explicación: las corridas observadas coinciden con las esperadas; hay independencia.
- **Test de Arreglos Inversos:**

  - Inversiones observadas: 1955115
  - Inversiones esperadas: 2570410.50
  - ¿Pasa el test? No
  - Explicación: el número de inversiones está fuera del rango esperado; posible sesgo.
- **Test de Sumas Solapadas (m=5):**
  - Suma media observada: 2.5526
  - Suma media esperada: 2.5000
  - ¿Pasa el test? No
  - Explicación: la media de las sumas difiere; posible dependencia o sesgo.
- **Test de Poker:**
  - Frecuencia de todos diferentes: 9
  - Frecuencia de pares: 23
  - Frecuencia de dos pares: 6
  - Frecuencia de trios: 3169
  - Frecuencia de full: 0
  - Frecuencia de poker: 0
  - Frecuencia de quintilla: 0
  - ¿Pasa el test? No
  - Explicación: la frecuencia de combinaciones no es consistente con la distribución esperada; posible sesgo.

4.6.2 Generador Lineal Congruencial

- **Test de Frecuencia (Chi-cuadrado):**
  - Chi-cuadrado calculado: 6.2990
  - Valor crítico ($\alpha = 0.05$): 16.9190
  - ¿Pasa el test? Sí
  - Explicación: el valor calculado es menor que el crítico; la distribución es uniforme.
- **Test de Independencia de Corridas:**
  - Corridas observadas: 1629
  - Corridas esperadas: 1604.50
  - ¿Pasa el test? Sí
  - Explicación: las corridas observadas coinciden con las esperadas; hay independencia.
- **Test de Arreglos Inversos:**
  - Inversiones observadas: 2554337
  - Inversiones esperadas: 2570410.50
  - ¿Pasa el test? Sí
  - Explicación: el número de inversiones está dentro del rango esperado.
- **Test de Sumas Solapadas (m=5):**
  - Suma media observada: 2.5089
  - Suma media esperada: 2.5000
  - ¿Pasa el test? Sí
  - Explicación: la media de las sumas concuerda con la teórica; cumple iid.
- **Test de Poker:**
  - Frecuencia de todos diferentes: 960
  - Frecuencia de pares: 1608
  - Frecuencia de dos pares: 350
  - Frecuencia de trios: 246
  - Frecuencia de full: 30
  - Frecuencia de poker: 13
  - Frecuencia de quintilla: 0
  - ¿Pasa el test? Sí
  - Explicación: la frecuencia de combinaciones es consistente con la distribución esperada.

4.6.3 Generador Cuadrático Congruencial

- **Test de Frecuencia (Chi-cuadrado):**
  - Chi-cuadrado calculado: 9.8101
  - Valor crítico ($\alpha = 0.05$): 16.9190
  - ¿Pasa el test? Sí
  - Explicación: el valor calculado es menor que el crítico; la distribución es uniforme.
- **Test de Independencia de Corridas:**
  - Corridas observadas: 1556
  - Corridas esperadas: 1604.50
  - ¿Pasa el test? Sí
  - Explicación: las corridas observadas coinciden con las esperadas; hay independencia.
- **Test de Arreglos Inversos:**
  - Inversiones observadas: 2566055
  - Inversiones esperadas: 2570410.50
  - ¿Pasa el test? Sí
  - Explicación: el número de inversiones está dentro del rango esperado.
- **Test de Sumas Solapadas (m=5):**
  - Suma media observada: 2.5280
  - Suma media esperada: 2.5000
  - ¿Pasa el test? No
  - Explicación: la media de las sumas difiere; posible dependencia o sesgo.
- **Test de Poker:**
  - Frecuencia de todos diferentes: 973
  - Frecuencia de pares: 1619
  - Frecuencia de dos pares: 339
  - Frecuencia de trios: 235
  - Frecuencia de full: 29
  - Frecuencia de poker: 12
  - Frecuencia de quintilla: 0
  - ¿Pasa el test? Sí
  - Explicación: la frecuencia de combinaciones es consistente con la distribución esperada.

4.6.4 Generador Mixto

- **Test de Frecuencia (Chi-cuadrado):**
  - Chi-cuadrado calculado: 7.8269
  - Valor crítico ($\alpha = 0.05$): 16.9190
  - ¿Pasa el test? Sí
  - Explicación: el valor calculado es menor que el crítico; la distribución es uniforme.
- **Test de Independencia de Corridas:**
  - Corridas observadas: 1617
  - Corridas esperadas: 1604.50
  - ¿Pasa el test? Sí
  - Explicación: las corridas observadas coinciden con las esperadas; hay independencia.
- **Test de Arreglos Inversos:**
  - Inversiones observadas: 2563399
  - Inversiones esperadas: 2570410.50
  - ¿Pasa el test? Sí
  - Explicación: el número de inversiones está dentro del rango esperado.

- **Test de Sumas Solapadas (m=5):**
  - Suma media observada: 2.5048
  - Suma media esperada: 2.5000
  - ¿Pasa el test? Sí
  - Explicación: la media de las sumas concuerda con la teórica; cumple iid.
- **Test de Poker:**
  - Frecuencia de todos diferentes: 976
  - Frecuencia de pares: 1624
  - Frecuencia de dos pares: 326
  - Frecuencia de trios: 230
  - Frecuencia de full: 37
  - Frecuencia de poker: 14
  - Frecuencia de quintilla: 0
  - ¿Pasa el test? Sí
  - Explicación: la frecuencia de combinaciones es consistente con la distribución esperada.

4.6.5 Generador del Lenguaje Python

- **Test de Frecuencia (Chi-cuadrado):**
  - Chi-cuadrado calculado: 1.3910
  - Valor crítico ($\alpha = 0.05$): 16.9190
  - ¿Pasa el test? Sí
  - Explicación: el valor calculado es menor que el crítico; la distribución es uniforme.
- **Test de Independencia de Corridas:**
  - Corridas observadas: 1649
  - Corridas esperadas: 1604.50
  - ¿Pasa el test? Sí
  - Explicación: las corridas observadas coinciden con las esperadas; hay independencia.
- **Test de Arreglos Inversos:**
  - Inversiones observadas: 2586320
  - Inversiones esperadas: 2570410.50
  - ¿Pasa el test? Sí
  - Explicación: el número de inversiones está dentro del rango esperado.
- **Test de Sumas Solapadas (m=5):**
  - Suma media observada: 2.5058
  - Suma media esperada: 2.5000
  - ¿Pasa el test? Sí
  - Explicación: la media de las sumas concuerda con la teórica; cumple iid.
- **Test de Poker:**
  - Frecuencia de todos diferentes: 981
  - Frecuencia de pares: 1582
  - Frecuencia de dos pares: 363
  - Frecuencia de trios: 238
  - Frecuencia de full: 30
  - Frecuencia de poker: 13
  - Frecuencia de quintilla: 0
  - ¿Pasa el test? Sí
  - Explicación: la frecuencia de combinaciones es consistente con la distribución esperada.

Generador	Tipo de Generador	Frecuencia	Corridas	Arreglos Inversos	Sumas Solapadas	Poker
Generador de Medios Cuadrados	Pseudoaleatorio	ERROR	OK	ERROR	ERROR	ERROR
Generador Lineal Congruencial	Pseudoaleatorio	OK	OK	OK	OK	OK
Generador Cuadrático Congruencial	Pseudoaleatorio	OK	OK	OK	ERROR	OK
Generador Mixto	Pseudoaleatorio	OK	OK	OK	OK	OK
Generador Lenguaje Python	Pseudoaleatorio	OK	OK	OK	OK	OK

Table 2: Resultados de pruebas estadísticas para diferentes generadores

5 Conclusión

A partir de la evaluación de diversos generadores de números pseudoaleatorios, se pudo observar la calidad y efectividad de cada uno a través de varias pruebas estadísticas estándar. A continuación, se presenta un resumen de los hallazgos clave:

5.1 Generador de Medios Cuadrados

Este generador mostró una gran deficiencia en cuanto a su capacidad para generar números aleatorios de manera uniforme. Los patrones repetitivos y el ciclo corto de los valores generados, junto con una notable falta de aleatoriedad, lo hacen inapropiado para aplicaciones modernas. El análisis gráfico reveló agrupaciones lineales en la dispersión de los números, lo que demuestra la ausencia de variabilidad aleatoria. Este generador no cumple con los requisitos básicos de uniformidad y aleatoriedad, lo que lo convierte en una opción obsoleta para simulaciones.

5.2 Generador Lineal Congruencial

Aunque el generador congruencial lineal tiene la ventaja de ser sencillo y fácil de implementar, presenta ciertos problemas en términos de aleatoriedad. A pesar de su capacidad para generar secuencias relativamente largas antes de repetirse, el test de independencia de corridas muestra una dispersión desigual, lo que sugiere que no es completamente aleatorio. La visualización de los datos también reveló ciertas estructuras lineales, lo que indica que la secuencia de números generados sigue una dependencia entre valores sucesivos. Este generador podría ser útil en contextos didácticos o simulaciones simples, pero no es adecuado para aplicaciones críticas o de alta precisión.

5.3 Generador Cuadrático Congruencial

El generador cuadrático mostró un comportamiento más caótico al inicio, pero, con el tiempo, los valores tienden a estabilizarse en ciclos repetitivos, lo que nuevamente reduce su capacidad de generar secuencias aleatorias a largo plazo. A pesar de que se introdujo un término cuadrático para mejorar la aleatoriedad, los resultados indicaron que aún es susceptible a ciclos cortos y patrones repetitivos, lo que lo hace inapropiado para simulaciones de alta calidad.

5.4 Generador Mixto

El generador mixto combina las salidas del Generador Cuadrático Congruencial y del Generador Lineal Congruencial para aprovechar las ventajas de ambos métodos y mitigar sus defectos individuales. Los análisis gráficos muestran que, al superponer ambas secuencias, se eliminan las claras estructuras lineales y cuadráticas observadas en los generadores por separado, logrando una dispersión mucho más homogénea sobre el intervalo $[0, 1)$. Aunque su complejidad computacional es mayor debido al cálculo doble y la combinación de valores, los resultados de pruebas estadísticas estándar indican una notable mejora en uniformidad e independencia frente a generadores simples, convirtiendo al generador mixto en una alternativa robusta para simulaciones que requieran alta calidad de pseudoaleatoriedad.

5.5 Generador de Python (Mersenne Twister)

El generador basado en el algoritmo Mersenne Twister, utilizado por defecto en Python a través del módulo `random`, demostró ser el más robusto de todos los generadores analizados. La dispersión de los números generados no mostró patrones obvios y se comportó de acuerdo con los estándares modernos de aleatoriedad, lo que lo convierte en la opción más confiable para simulaciones y análisis estadísticos. Su capacidad para mantener una distribución uniforme y su ausencia de dependencias entre valores consecutivos lo hacen ideal para aplicaciones generales y científicas, aunque no está diseñado específicamente para criptografía.

5.6 Resultados de los Test Estadísticos

En términos de los **test estadísticos** realizados, los generadores mostraron diversos niveles de rendimiento:

- **Test de Frecuencia:** El generador de medios cuadrados falló completamente, mientras que los otros generadores (excepto el de medios cuadrados) pasaron este test con buenos resultados, destacando el generador de Python como el más adecuado.
- **Test de Independencia de Corridas:** Este test reveló que todos los generadores (excepto el de medios cuadrados) mostraron un número adecuado de corridas, lo que indica una independencia razonable entre los valores generados.
- **Test de Arreglos Inversos:** Todos los generadores, excepto el de medios cuadrados, pasaron este test, demostrando que los arreglos inversos de los valores generados siguen una distribución adecuada.
- **Test de Sumas Solapadas:** En este test todos los generadores, excepto el de medios cuadrados, lograron resultados dentro de los parámetros aceptables, lo que indica que las sumas solapadas de sus secuencias presentan un comportamiento estadísticamente adecuado.
- **Test de Poker:** Los generadores cuadrático, lineal y Python pasaron el test de poker, lo que indica que sus secuencias cumplen con las combinaciones esperadas.

5.7 Conclusión General

En conclusión, el **Mersenne Twister** (usado en Python) se destaca como el generador más adecuado para simulaciones y análisis estadísticos debido a su distribución uniforme, independencia de valores sucesivos y capacidad para generar secuencias de alta calidad. Sin embargo, no debe considerarse para aplicaciones que requieren una aleatoriedad criptográfica segura, como la generación de claves para criptografía o la creación de contraseñas seguras. Para esos casos, Python proporciona el módulo secrets, que utiliza generadores de números aleatorios adecuados para la seguridad. En contraste, los generadores más antiguos, como el generador de medios cuadrados y el congruencial cuadrático, no cumplen con los estándares modernos de aleatoriedad, y deberían ser evitados en aplicaciones que requieren alta precisión y calidad estadística.

Agradecimientos

Agradecemos a los profesores Juan I. Torres y Jorge A. Flamini por su orientación, y a la Universidad Tecnológica Nacional - FRRO por el apoyo brindado.

References

[1] Estadística Computacional. *Números pseudoaleatorios.* https://tereom.github.io/est-computacional-2018/numeros-pseudoaleatorios.html

[2] Ross, S. M. *Introduction to Probability and Statistics for Engineers and Scientists*. Academic Press.

[3] Python Software Foundation. https://docs.python.org/3/

[4] Random.org. https://www.random.org/analysis/

[5] Random.org. *Random Number Generators: An Evaluation and Comparison of Random.org and Some Commonly Used Generators* https://www.random.org/analysis/Analysis2005.pdf

[6] Medium. *Poker Test (Sequence Randomness)* https://medium.com/@dillihangrae/poker-test-sequence-randomness-99d2bbda9081

[7] Overleaf. *Plantilla Latex Cornell University*. https://es.overleaf.com/latex/templates-style-and-template-for-preprints-arxiv-bio-arxiv/pkzcrhzcdxmc