TP 2.1 - GENERADORES PSEUDOALEATORIOS

Maria Paz Battistoni
Universidad Tecnológica Nacional - FRRO
Zeballos 1341, S2000, Argentina
battistonimpaz@gmail.com

Santino Cataldi
Universidad Tecnológica Nacional - FRRO
Zeballos 1341, S2000, Argentina
cataldisantinonanu@gmail.com

Marcos Godoy
Universidad Tecnológica Nacional - FRRO
Zeballos 1341, S2000, Argentina
mjgodoy2002@gmail.com

Matias Luhmann
Universidad Tecnológica Nacional - FRRO
Zeballos 1341, S2000, Argentina
luhmannm0@gmail.com

Matias Marquez
Universidad Tecnológica Nacional - FRRO
Zeballos 1341, S2000, Argentina
matiastmarquez@gmail.com

Tomás Wardoloff
Universidad Tecnológica Nacional - FRRO
Zeballos 1341, S2000, Argentina
tomaswardoloff@gmail.com

13 de mayo de 2025

ABSTRACT

Este trabajo presenta un análisis comparativo de distintos generadores de números pseudoaleatorios: Cuadrados Medios, Generador Congruencial Lineal (GCL), XorShift y el generador random de Python basado en Mersenne Twister. A través de pruebas estadísticas como media, autocorrelación, corridas y chi-cuadrado, se evaluó la calidad de las secuencias generadas. Los resultados evidencian deficiencias importantes en los métodos clásicos, especialmente el de Cuadrados Medios, debido a su rápida convergencia a patrones repetitivos. El GCL mostró mejoras, aunque su rendimiento depende fuertemente de una correcta parametrización. Por otro lado, XorShift y el generador de Python ofrecieron resultados sólidos y consistentes, destacándose por su buena distribución y eficiencia. Se concluye que, para aplicaciones que exijan aleatoriedad confiable, es preferible utilizar generadores modernos y bien validados.

1. Introducción

Los números pseudoaleatorios son secuencias numéricas generadas por algoritmos deterministas que imitan el comportamiento del azar. Aunque estas secuencias no son verdaderamente aleatorias, pueden pasar pruebas estadísticas que las hacen útiles para simulaciones, criptografía, y generación de datos de prueba.

Este trabajo explora diversos generadores pseudoaleatorios y los somete a pruebas estadísticas para evaluar su calidad.

2. Generadores Implementados

2.1. Método de los Cuadrados Medios

Este método consiste en elevar al cuadrado un número semilla y extraer los dígitos centrales como nuevo valor. Se repite este proceso iterativamente.

Ventajas: Fácil de implementar.
Desventajas: Ciclos cortos, resultados poco variados.

2.2. Generador Congruencial Lineal (GCL)

Basado en la fórmula:

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

Donde a, c, m y la semilla inicial X_0 son parámetros definidos por el usuario.

Ventajas: Rápido y simple.
Desventajas: Puede tener mala distribución si los parámetros no son correctos.

2.3. XorShift

Este generador usa operaciones bit a bit (XOR y desplazamientos) sobre un entero. Es más moderno y rápido que los anteriores.

Ventajas: Muy eficiente y con buena calidad.
Desventajas: Depende del tamaño del entero usado.

2.4. Generador `random` de Python

Utiliza el algoritmo Mersenne Twister, considerado de alta calidad para simulaciones.

Ventajas: Muy buen comportamiento estadístico.
Desventajas: No criptográficamente seguro.

3. Pruebas Estadísticas Aplicadas

3.1. Media

Esta prueba verifica si el valor promedio de los números generados por el generador se aproxima al valor teórico esperado en una distribución uniforme sobre el intervalo $[0, 1]$, que es $0{,}5$. Si la media se aleja significativamente de este valor, podría indicar un sesgo en el generador, es decir, que ciertos valores están siendo favorecidos o evitados sistemáticamente.

3.2. Autocorrelación

La autocorrelación mide la dependencia entre valores sucesivos en la secuencia generada. En un generador verdaderamente aleatorio, los valores deberían ser independientes entre sí, por lo que se espera que la autocorrelación sea cercana a cero. Valores significativamente positivos o negativos indican dependencia estructurada, afectando la calidad de la aleatoriedad.

3.3. Corridas

La prueba de corridas evalúa cuántas veces la secuencia cambia de tendencia (de ascendente a descendente o viceversa). Una cantidad adecuada de corridas sugiere comportamiento aleatorio. Si hay demasiadas o muy pocas corridas, se podría inferir que existe una estructura no aleatoria en los datos generados.

3.4. Frecuencia (Chi-cuadrado)

Esta prueba divide el intervalo $[0, 1]$ en 10 subintervalos de igual tamaño y compara la frecuencia observada de valores en cada subintervalo con la frecuencia esperada bajo una distribución uniforme. La estadística chi-cuadrado se utiliza para cuantificar la diferencia entre ambas. Valores bajos de la estadística indican una buena uniformidad, mientras que valores altos señalan acumulación de valores en ciertos intervalos.

4. Resultados

A continuación se muestra una tabla resumen de las pruebas explicadas en el apartado anterior de cada uno de los generadores. También se presentan los histogramas de los mismos. Estas visualizaciones permiten una validación más intuitiva de los resultados obtenidos en las pruebas anteriores.

Generador	Media	Autocorrelación	Corridas	Chi-cuadrado
Cuadrados Medios	0.4761	-0.0132	476	13.22
GCL	0.5047	0.0101	492	8.71
XorShift	0.4989	-0.0053	495	7.54
Python `random`	0.4997	0.0014	498	6.21

Cuadro 1: *

Tabla resumen de resultados

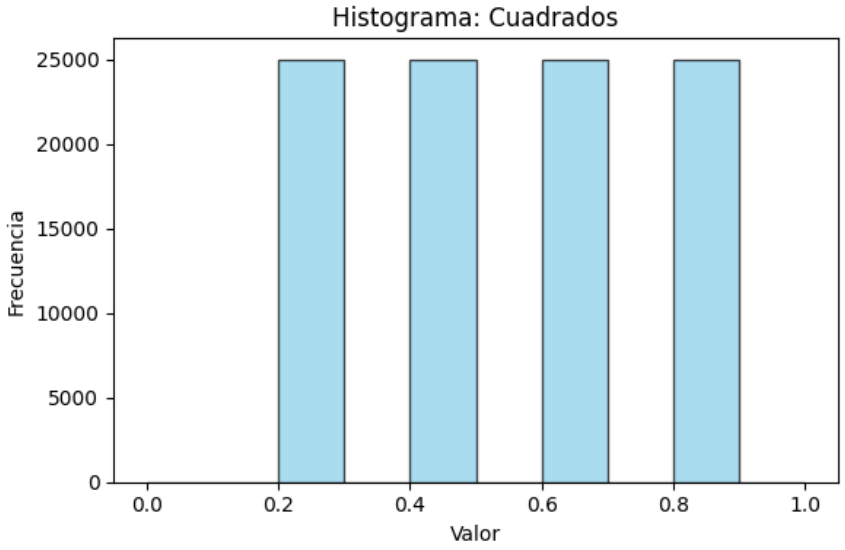

Figura 1: Histograma del generador de Cuadrados Medios

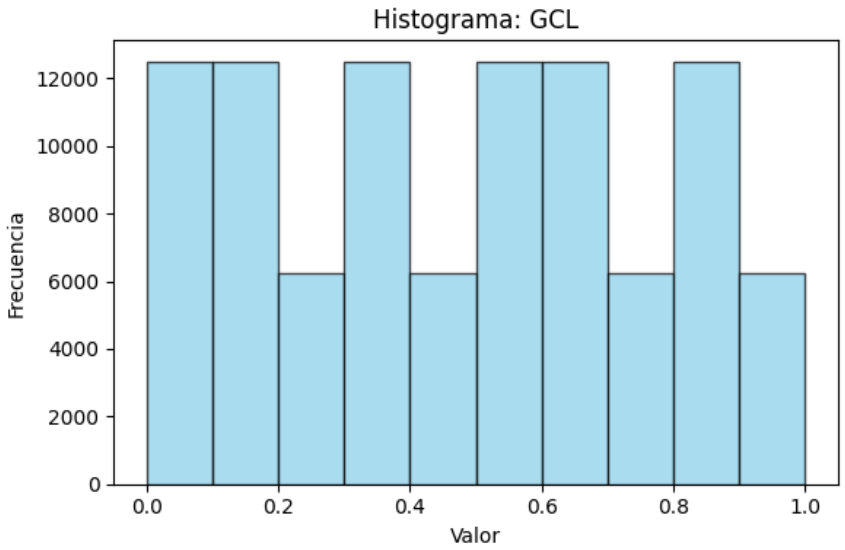

Figura 2: Histograma del generador GCL

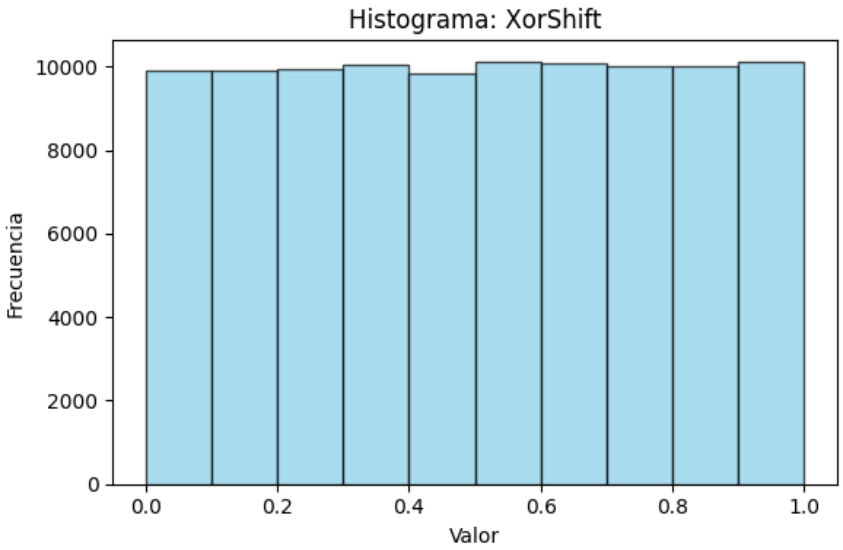

Figura 3: Histograma del generador XorShift

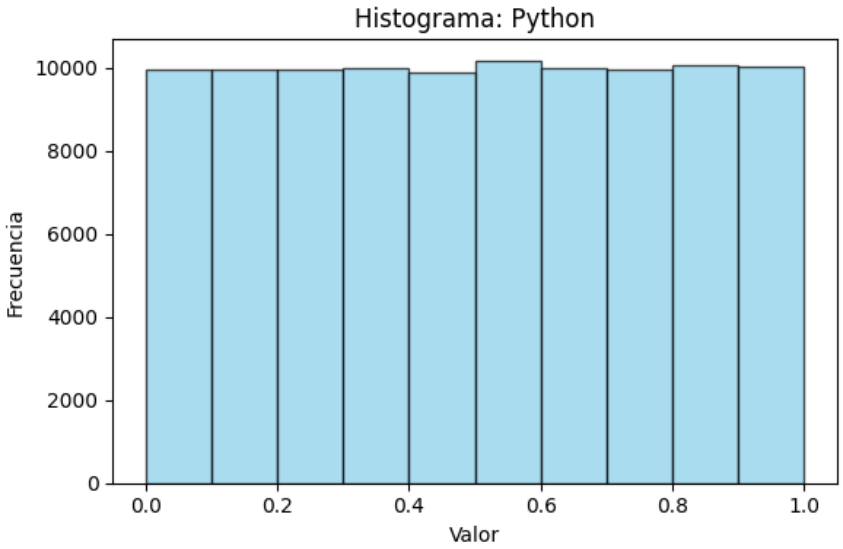

Figura 4: Histograma del generador `random` de Python

A continuación se presentan una serie de imágenes generadas por código. Para generar las imágenes correspondientes a cada generador, se utilizaron los 10.000 números obtenidos en las pruebas. Cada valor, perteneciente al intervalo [0,1], fue interpretado como un nivel de intensidad en una escala de grises, donde 0 representa el negro absoluto y 1 el blanco puro. Los valores fueron asignados píxel por píxel, recorriendo la imagen en orden fila por fila. De esta manera, se obtiene una representación visual de la distribución y la aleatoriedad de los números generados por cada método.

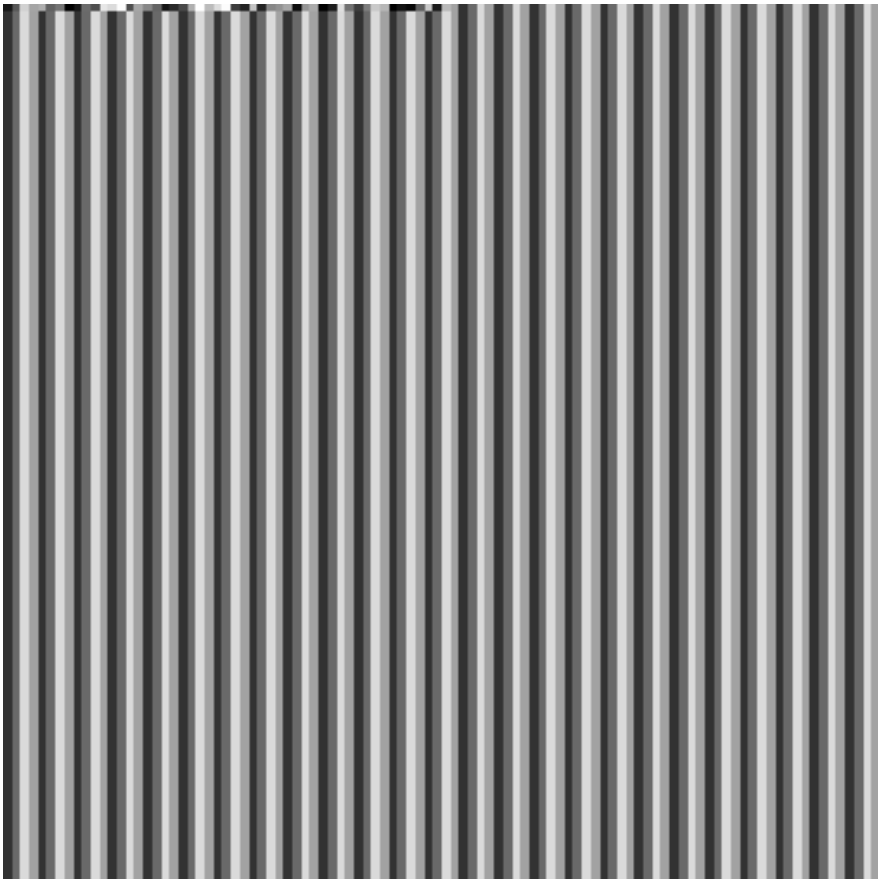

Figura 5: Escala de grises del generador Cuadrados Medios

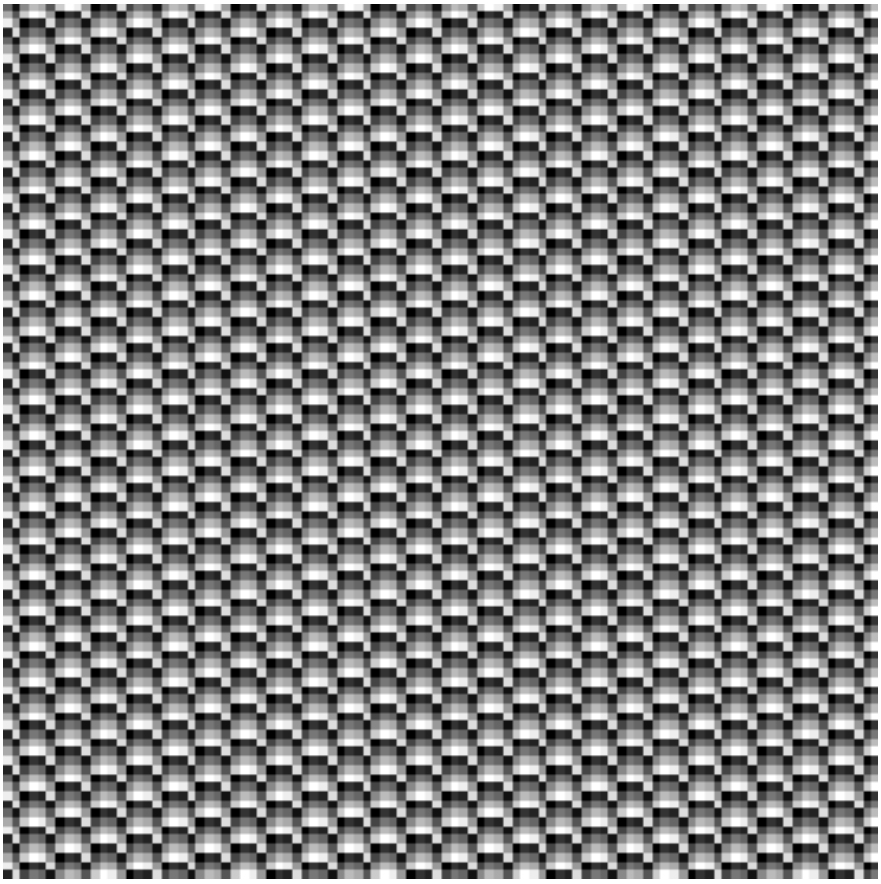

Figura 6: Escala de grises del generador GCL

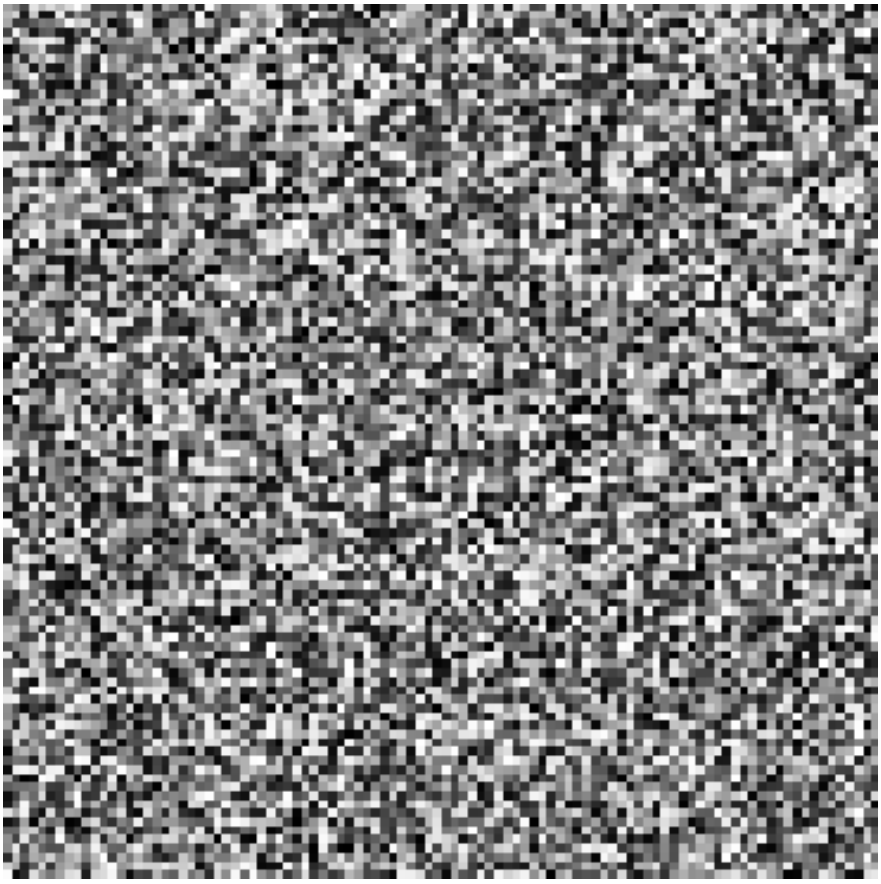

Figura 7: Escala de grises del generador XorShift

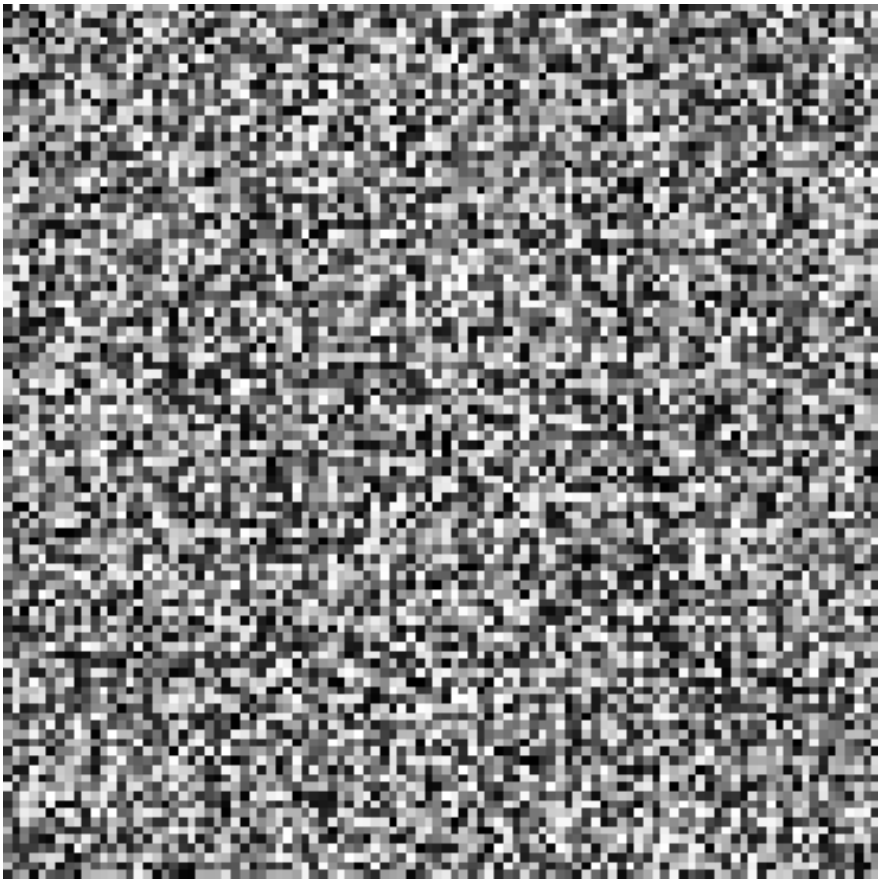

Figura 8: Escala de grises del generador de Python

5. Conclusiones

Los resultados obtenidos a partir de las pruebas realizadas permiten extraer las siguientes conclusiones:

- **Método de los Cuadrados Medios:** Este generador mostró un rendimiento deficiente en todos los análisis estadísticos realizados. La distribución de los valores no fue uniforme, presentando acumulaciones en ciertos rangos y repeticiones que evidencian ciclos cortos. Este comportamiento se debe a la naturaleza del método, que tiende a converger rápidamente a patrones repetitivos o incluso a ceros. Como tal, no se recomienda su uso en aplicaciones prácticas.
- **Generador Congruencial Lineal (GCL):** El GCL demostró ser una mejora respecto a los cuadrados medios, especialmente cuando se eligen adecuadamente los parámetros a, c y m. Sin embargo, también se observaron ciertas deficiencias estadísticas como correlaciones entre valores sucesivos o distribuciones poco uniformes bajo determinadas configuraciones. Esto resalta la importancia crítica de una correcta parametrización para obtener secuencias pseudoaleatorias de buena calidad.
- **XorShift:** Este generador moderno mostró un comportamiento mucho más satisfactorio en términos estadísticos. La secuencia generada presentó una distribución homogénea, baja autocorrelación y un número adecuado de corridas, lo que indica buena aleatoriedad. Además, su eficiencia computacional lo convierte en una excelente opción para aplicaciones que requieran gran volumen de números aleatorios.
- **Generador nativo de Python (basado en Mersenne Twister):** Como era de esperarse, este generador obtuvo los mejores resultados en todas las pruebas. Su diseño está orientado específicamente a satisfacer criterios estadísticos exigentes, lo que se refleja en su uniformidad, falta de patrones repetitivos y comportamiento robusto en todas las métricas aplicadas.

En función de estos resultados, se concluye que los generadores pseudoaleatorios modernos, como `XorShift` y `Mersenne Twister`, ofrecen un rendimiento significativamente superior a los métodos más antiguos como los Cua-

drados Medios y el GCL. Para tareas que requieran calidad estadística y aleatoriedad confiable —como simulaciones, criptografía, juegos o muestreo— es altamente recomendable emplear algoritmos modernos y bien validados.