TP 2.1 GENERADORES PSEUDOALEATORIOS

Pecoraro Lucio
Universidad Tecnológica Nacional - FRRO
Zeballos 1341, S2000, Argentina
Legajo 50239
luciopecoraro2002@gmail.com

Berto Leandro
Universidad Tecnológica Nacional - FRRO
Zeballos 1341, S2000, Argentina
Legajo 45368
leandroberto2010@gmail.com

Capiglioni Rodrigo
Universidad Tecnológica Nacional - FRRO
Zeballos 1341, S2000, Argentina
Legajo 47298
RodrigoCapiglioni@gmail.com

Broda Tomás
Universidad Tecnológica Nacional - FRRO
Zeballos 1341, S2000, Argentina
Legajo 47299
tomasbroda13@gmail.com

May 14, 2025

ABSTRACT

El siguiente documento tiene como objetivo el estudio y análisis de los resultados de distintas pruebas realizadas sobre la generación de numeros pseudoaleatorios, con el fin de determinar si efectivamente se pueden tomar como aleatorios los numeros generados

1 Introduccion

La generación de números pseudoaleatorios juega un rol central en la simulación de fenómenos estocásticos. Si bien estos números son producidos por algoritmos deterministas, su utilidad práctica depende de su capacidad para simular el comportamiento de una fuente de aleatoriedad genuina. Por tal motivo, resulta imprescindible analizar si las secuencias generadas cumplen con ciertos criterios estadísticos que nos permitan considerarlas efectivamente aleatorias.

Entre las propiedades deseables se encuentran la uniformidad, la independencia, la ausencia de correlación y la imprevisibilidad. A fin de evaluar dichas propiedades, se aplican diversos tests estadísticos como la prueba de frecuencia (o uniformidad), prueba de series, prueba de autocorrelación y prueba de poker, entre otras. Estos tests permiten identificar patrones ocultos o comportamientos no deseados que podrían comprometer la validez de una simulación.

Este análisis busca no solo validar el comportamiento de generadores implementados manualmente —como el Generador Congruencial Lineal (GCL)—, sino también comparar su desempeño con generadores incorporados en bibliotecas modernas como `random` de Python. De esta forma, se establece un marco cuantitativo para juzgar hasta qué punto una secuencia pseudoaleatoria puede ser considerada "suficientemente aleatoria" para fines prácticos.

2 Conceptos teoricos

Para evaluar la calidad de los generadores pseudoaleatorios implementados, se aplicarán distintas pruebas estadísticas comúnmente aceptadas en el ámbito de la simulación. Estas pruebas permiten determinar si una secuencia de números generada posee características compatibles con un comportamiento verdaderamente aleatorio.

2.1 Prueba de Chi-cuadrado

La prueba de Chi-cuadrado es una de las más utilizadas para verificar la uniformidad en la distribución de los números generados. Se divide el intervalo [0,1) en k subintervalos de igual tamaño y se compara la frecuencia observada en cada subintervalo con la frecuencia esperada. La estadística de prueba es:

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

donde O_i representa la frecuencia observada y E_i la frecuencia esperada en el i-ésimo subintervalo. Si el valor calculado de χ^2 se encuentra dentro del rango crítico para un nivel de significancia dado, se acepta que la distribución es uniforme.

2.2 Prueba de Poker

Esta prueba se basa en analizar bloques de d dígitos extraídos de los números generados, categorizándolos según la repetición de dígitos (por ejemplo, todos distintos, un par, una tercia, etc.). Se calcula la frecuencia observada de cada categoría y se compara con la frecuencia teórica esperada, utilizando la estadística de chi-cuadrado. Esta prueba permite detectar patrones no aleatorios en la estructura interna de los números generados.

Las probabilidades para cada una de las manos del póker diferentes se muestran enseguida:

1. Todos diferentes = 0.3024
2. Un par = 0.504
3. Dos pares = 0.108
4. Tercia = 0.072
5. Póker = 0.009
6. Full = 0.0045
7. Generala = 0.0001

2.3 Prueba de Corridas

La prueba de corridas examina la secuencia de números en términos de ascensos y descensos, o bien en función de si los valores están por encima o debajo de la media. Una "corrida" es una sucesión de números que cumplen una misma condición, en este caso, mayores o menores que la media. Se cuenta la cantidad total de corridas y se compara con el valor esperado. Dado un conjunto de n números pseudoaleatorios, se clasifican según si están por encima o por debajo de la media. Se define:

- n_1: cantidad de valores mayores que la media.
- n_2: cantidad de valores menores a la media.
- R: número total de corridas observadas.

Valor esperado de corridas:

$$\mu_R = \frac{2n_1n_2}{n_1 + n_2} + 1$$

Varianza de la cantidad de corridas:

$$\sigma_R^2 = \frac{2n_1n_2(2n_1n_2 - n_1 - n_2)}{(n_1 + n_2)^2(n_1 + n_2 - 1)}$$

Estadístico de prueba:

$$Z = \frac{R - \mu_R}{\sigma_R}$$

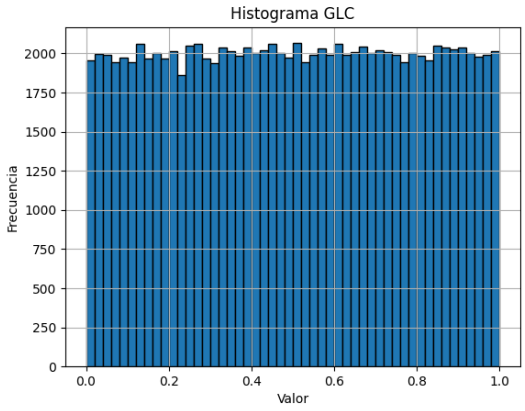
2.4 Prueba de Series

La prueba de series crecientes y decrecientes se utiliza para detectar patrones de dependencia en secuencias de números pseudoaleatorios, observando el comportamiento relativo entre pares consecutivos de valores. Una secuencia verdaderamente aleatoria no debería mostrar una cantidad excesiva ni insuficiente de secuencias en orden creciente o decreciente.

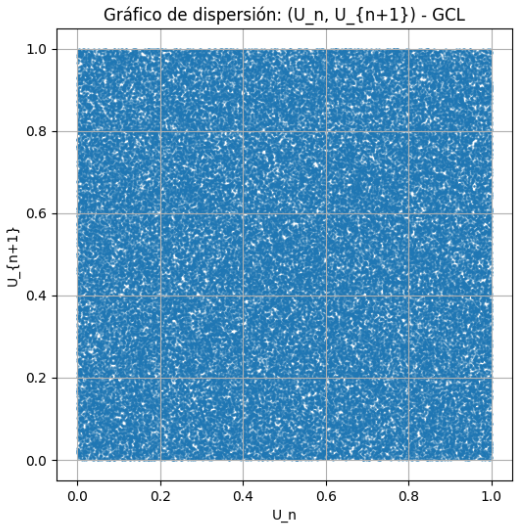
3 Resultados y análisis

Se presentan a continuación los resultados obtenidos para cada generador de números pseudoaleatorios, aplicando las mismas pruebas estadísticas y gráficas a cada uno

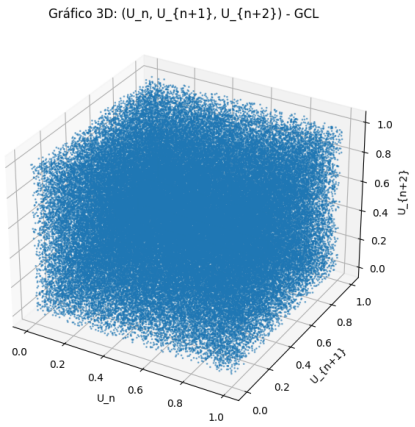
3.1 Generador Congruencial Lineal (GCL)



(a) Histograma



(b) Dispersión 2D



(c) Dispersión 3D

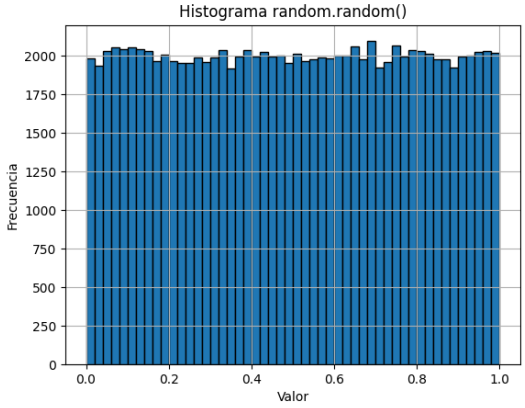
Figure 1: Resultados visuales del generador GCL.

El generador congruencial lineal (GCL) presenta un comportamiento visualmente adecuado en términos de uniformidad y dispersión. El histograma muestra una distribución pareja sin vacíos ni concentraciones, mientras que los gráficos de dispersión 2D y 3D no evidencian alineamientos o patrones estructurados.

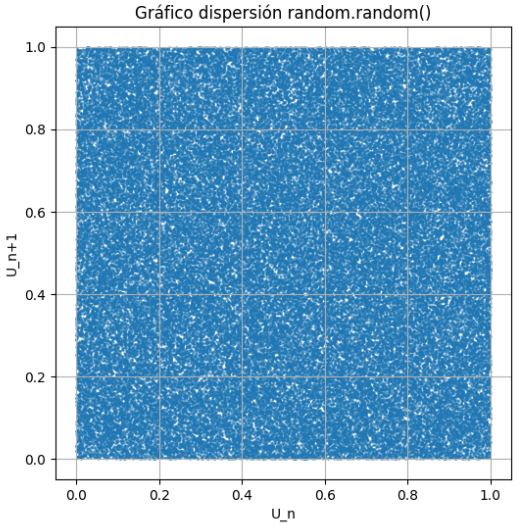
Desde el punto de vista estadístico, el test de frecuencia arrojó resultados aceptables para muestras grandes ($p = 0.44$), aunque algo sospechosos para $n = 10.000$ ($p = 0.032$). El test de corridas dio $Z = -0.93$ y $p = 0.34$, indicando un número de transiciones esperado.

En conjunto, el GCL puede considerarse aceptablemente aleatorio para simulaciones de tamaño mediano a grande, pero su sensibilidad a tamaños pequeños sugiere tener precaución en aplicaciones con muestras limitadas.

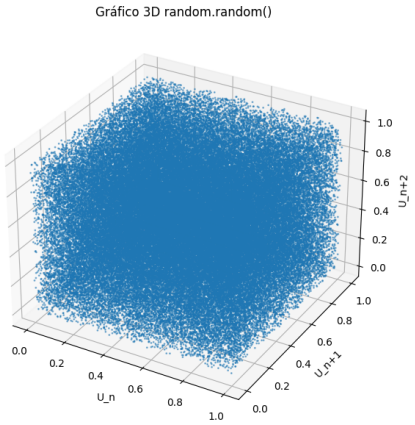
3.2 Generador `random()` de Python



(a) Histograma



(b) Dispersión 2D



(c) Dispersión 3D

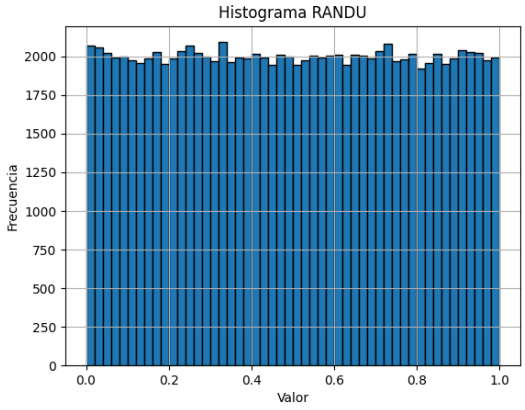
Figure 2: Resultados visuales del generador `random()`.

El generador `random()` de Python mostró un rendimiento sólido y consistente en todos los aspectos evaluados. El histograma confirma una distribución uniforme de los valores generados, mientras que los gráficos de dispersión 2D y 3D reflejan una excelente cobertura del espacio, sin indicios de patrones repetitivos o dependencia entre valores.

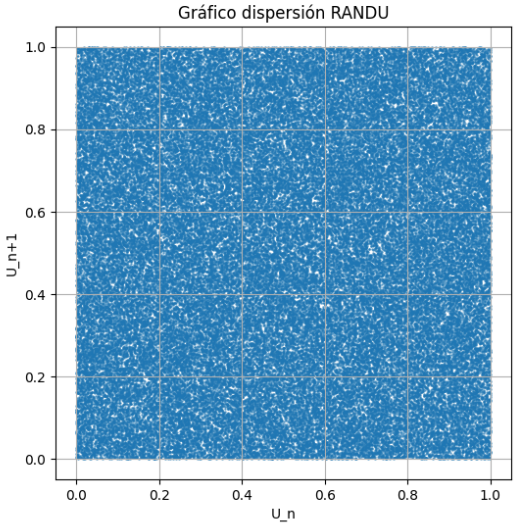
Los resultados estadísticos fueron óptimos: el test de frecuencia arrojó valores $p = 0.49$ y $p = 0.51$ para muestras de $n = 10.000$ y $n = 100.000$ respectivamente, lo cual indica una distribución uniforme estable. En cuanto a la prueba de corridas, se obtuvo $Z = 0.16$ y $p = 0.86$, valores que respaldan la ausencia de sesgos o estructuras en la secuencia.

En resumen, `random()` es un generador altamente confiable para simulaciones, robusto incluso en muestras pequeñas, y sin necesidad de ajustes adicionales. Su comportamiento estadístico y visual respalda su uso como referencia para comparar otros generadores.

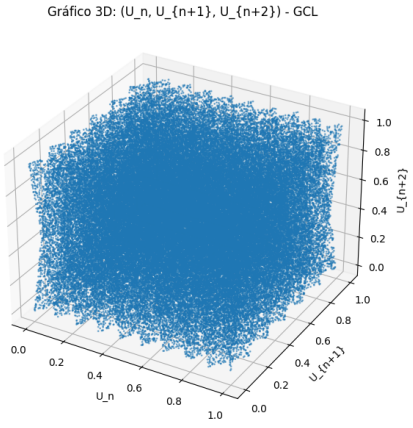
3.3 Generador RANDU



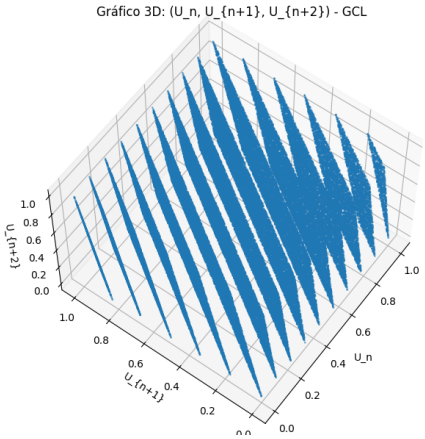
(a) Histograma



(b) Dispersión 2D



(c) Dispersión 3D



(d) Dispersion 3D desde vista superior

Figure 3: Resultados visuales del generador RANDU.

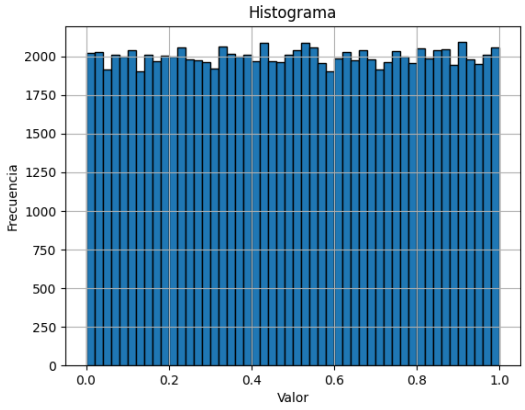
El generador RANDU, históricamente conocido por sus deficiencias estructurales, mostró en este análisis una mezcla de comportamientos aceptables y otros que confirman sus limitaciones.

Visualmente, si bien el histograma y la dispersión 2D no muestran irregularidades notorias, el gráfico tridimensional revela un patrón de alineamientos en planos, evidenciando el clásico problema que lo caracteriza negativamente. Esta estructura en planos es un signo claro de que la secuencia no es verdaderamente uniforme ni aleatoria en el espacio tridimensional.

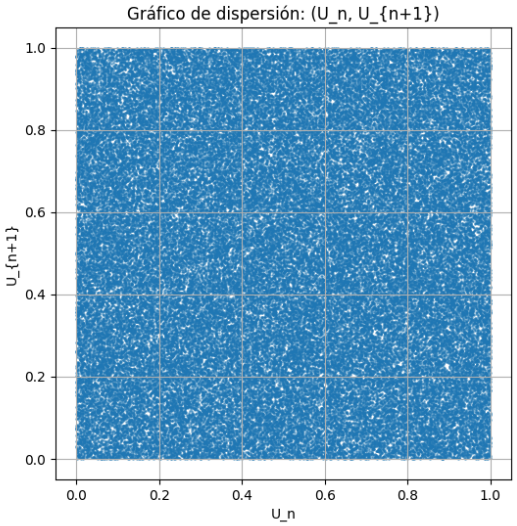
Desde el punto de vista estadístico: - El coeficiente de correlación de Pearson fue bajo ($\rho = 0.0059$) y con valor $p = 0.063$, cercano pero aún dentro de la aceptación. - En el test de Chi-cuadrado, se obtuvo $\chi^2 = 8.9856$ con $p = 0.4386$, resultado dentro del rango aceptable. - La prueba de corridas dio $Z = -1.46$, $p = 0.1441$, también indicando comportamiento aceptable respecto a cambios por encima y por debajo de la media. - En la prueba de póker, el valor de $\chi^2 = 10.224$ (gl = 6) con $p = 0.1155$ también se encuentra dentro del umbral aceptable.

A pesar de que RANDU logra superar los test estadísticos tradicionales de uniformidad, corridas y póker en muestras grandes, el patrón visual de alineamientos en 3D confirma su debilidad estructural. Esto lo vuelve inapropiado para simulaciones sensibles al espacio muestral multidimensional, aunque puede pasar desapercibido en evaluaciones univariadas.

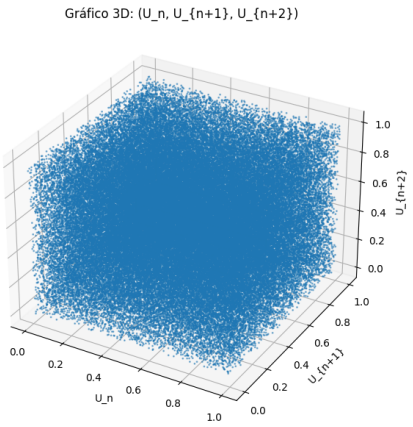
3.4 Generador XORSHIFT32



(a) Histograma



(b) Dispersión 2D



(c) Dispersión 3D

Figure 4: Resultados visuales del generador XORSHIFT32.

El generador XORSHIFT32 presentó un comportamiento muy sólido tanto visual como estadísticamente. El histograma evidenció una distribución uniforme y sin acumulaciones o vacíos notorios. El gráfico de dispersión 2D muestra independencia entre valores consecutivos, y el gráfico 3D —habitualmente crítico para detectores de defectos estructurales— no evidenció alineamientos, lo cual es un buen indicador para este tipo de generador bitwise.

En cuanto a los resultados estadísticos:

- En el test de Chi-cuadrado, se obtuvo un valor $\chi^2 = 3.50$ con $p = 0.941$, indicando una excelente uniformidad. - El test de corridas dio $Z = 1.41$, $p = 0.158$, y la prueba de series $Z = 0.32$, ambos bien dentro del rango esperado bajo

hipótesis de aleatoriedad. - La prueba de póker arrojó $\chi^2 = 4.83$, gl = 6, $p = 0.5655$, también aceptable sin evidencias de patrones estructurales en la composición de dígitos. - El coeficiente de correlación de Pearson fue cercano a cero ($\rho = -0.0025$, $p = 0.425$), indicando independencia lineal entre valores.

Se mostró un comportamiento robusto en todas las pruebas aplicadas. Aunque es un generador muy simple y extremadamente rápido, su desempeño fue comparable al de generadores más complejos como `random()`. No se observaron debilidades estadísticas ni visuales significativas en las pruebas aplicadas.

3.5 Comparación de modelos

A continuación se presenta una tabla comparativa entre los generadores analizados:

Generador	Prueba	Estadístico	p	Conclusión
GCL	Corridas	$Z = -0.93$	0.34	Aceptado
	Frecuencia (n=100k)	$\chi^2 = 9.48$	0.44	Aceptado
	Gráficos	–	–	Aceptado visualmente
random()	Corridas	$Z = 0.16$	0.86	Aceptado
	Frecuencia (n=100k)	$\chi^2 = 8.21$	0.51	Aceptado
	Gráficos	–	–	Aceptado visualmente
RANDU	Corridas	$Z = -1.46$	0.144	Aceptado
	Frecuencia (n=100k)	$\chi^2 = 8.99$	0.439	Aceptado
	Póker	$\chi^2 = 10.22$ (gl = 6)	0.116	Aceptado
	Gráficos	–	–	**No aceptado**
XORSHIFT32	Corridas	$Z = 1.41$	0.158	Aceptado
	Frecuencia (n=100k)	$\chi^2 = 3.50$	0.941	Aceptado
	Póker	$\chi^2 = 4.83$ (gl = 6)	0.566	Aceptado
	Gráficos	–	–	Aceptado visualmente

Table 1: Resumen comparativo de resultados estadísticos y visuales por generador.

4 Conclusion

De acuerdo a los resultados resumidos en la Tabla 1, todos los generadores analizados —excepto RANDU— superaron satisfactoriamente las pruebas estadísticas aplicadas, incluyendo las pruebas de corridas, frecuencia, y póker, con valores de p mayores al umbral típico de significancia, lo que indica que no se encontraron evidencias suficientes para rechazar la hipótesis de aleatoriedad.

El generador random() de Python mostró el desempeño más sólido en todas las pruebas, seguido por XORShift32, que también presentó buenos resultados tanto en pruebas estadísticas como en la evaluación visual. El generador GCL, si bien arrojó resultados aceptables, mostró una leve dependencia del ajuste de parámetros, siendo necesario un análisis más detallado para garantizar su robustez.

En contraste, el generador RANDU no fue aceptado visualmente, lo que refuerza su conocida debilidad para producir secuencias aleatorias de calidad, a pesar de que estadísticamente algunas pruebas fueron superadas. Este caso subraya la importancia de complementar las pruebas cuantitativas con análisis gráficos.

En síntesis, el trabajo evidencia la relevancia de combinar métodos estadísticos y visuales para evaluar generadores pseudoaleatorios. La correcta elección del algoritmo, junto con un análisis riguroso, es esencial para asegurar la confiabilidad de los números generados en aplicaciones que requieren alta calidad en la aleatoriedad.

5 Codigo del programa

El código del programa realizado se encuentra publicado en:

`https://github.com/pecorarolucio/tps-simulacion-2025/tree/main/TP2`

6 Bibliografía

References

[1] García, M. *Generador XORShift*. Recuperado de https://www.maxgcoding.com/xorshiftpnrg

[2] InfluentialPoints. *Runs tests: principles, properties and assumptions*. Recuperado de https://influentialpoints.com/Training/runs_tests-principles-properties-assumptions.htm

[3] RANDOM.ORG. (2005). *Statistical Analysis of the RANDOM.ORG True Random Number Generator*. Recuperado de https://www.random.org/analysis/Analysis2005.pdf

[4] RANDOM.ORG. *Randomness Analysis*. Recuperado de https://www.random.org/analysis/

[5] Tereom. (2018). *Números pseudoaleatorios*. Estimación Computacional. Recuperado de https://tereom.github.io/est-computacional-2018/numeros-pseudoaleatorios.html

[6] Python Software Foundation. *random — Generate pseudo-random numbers*. Documentación oficial de Python. Recuperado de https://docs.python.org/3/library/random.html

[7] idocpub. *Generadores y pruebas estadísticas*. Recuperado de https://idoc.pub/documents/idocpub-6klz2po2qvlg