TP 2.1 - GENERADORES PSEUDOALEATORIOS

Mateo Simón Arach
Leg. 51394
Universidad Tecnológica Nacional - FRRO
Zeballos 1341, S2000, Argentina
matearach@gmail.com

Francisca Gramaglia
Leg. 51424
Universidad Tecnológica Nacional - FRRO
Zeballos 1341, S2000, Argentina
franciscagramaglia714@gmail.com

Milton Rubén Borsato Gimenez
Leg. 51391
Universidad Tecnológica Nacional - FRRO
Zeballos 1341, S2000, Argentina
borsatomilton@gmail.com

José Ignacio Dayer
Leg. 51508
Universidad Tecnológica Nacional - FRRO
Zeballos 1341, S2000, Argentina
nachodayer@gmail.com

Valentin David Marchese
Leg. 51745
Universidad Tecnológica Nacional - FRRO
Zeballos 1341, S2000, Argentina
marchese52002@gmail.com

Joaquin Garcia Forestello
Leg. 51462
Universidad Tecnológica Nacional - FRRO
Zeballos 1341, S2000, Argentina
joaquingarciaforestello@gmail.com

May 14, 2025

ABSTRACT

El siguiente documento tiene por objetivo detallar el trabajo de clase que debe realizarse para introducirnos en uno de los elementos fundamentales para gran parte de las simulaciones, estos son los generadores de números pseudoaleatorios.

***K*eywords** Simulación · generadores pseudoaleatorios

1 Introducción

Un número pseudoaleatorio es un número generado en un proceso que parece producir números al azar, pero no lo hace realmente. Las secuencias de números pseudoaleatorios no muestran ningún patrón o regularidad aparente desde un punto de vista estadístico, a pesar de haber sido generadas por un algoritmo completamente determinista, ante las mismas condiciones iniciales producen siempre el mismo resultado.

2 Conceptos de Ínteres

Numero aleatorio: Un número aleatorio verdadero es un valor generado por un proceso físico impredecible, como el ruido térmico, el decaimiento radiactivo, o fluctuaciones atmosféricas. No sigue ningún algoritmo matemático ni patrón lógico, y no se puede predecir ni reproducir.
Ventajas:

1. Impredecibles: No se pueden anticipar ni replicar, lo cual es ideal para seguridad y criptografía
2. Aleatoriedad total: No están limitados por fórmulas ni patrones computacionales.
3. No dependen de una semilla inicial: No hay riesgo de generar patrones ocultos como en los PRNGs.

Numero pseudoaleatorio: Un número pseudoaleatorio es un valor generado por un algoritmo matemático que imita el comportamiento de un número aleatorio, pero en realidad es determinístico. Esto significa que, si se conoce la semilla inicial, toda la secuencia de números puede ser reproducida.
Ventajas:

1. Alta velocidad: Los algoritmos generan millones de valores rápidamente y eficientemente.
2. Faciles de implementar: No requieren hardware especial, solo software.
3. Utiles en simulaciones y estadisticas: Son ampliamente usados en experimentos Monte Carlo, juegos, inteligencia artificial, etc.

3 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). A modo de introducción y ejemplos de código se deja un link en la sección de **Recursos online obligatorios**.

4 Descripción del Experimento

Se ejecuta un programa en Python que:

1. Solicita una semilla ingresada por el usuario (minimo 10 caracteres) o utiliza la hora del sistema.
2. Inicializa los 4 generadores (Random Generator GLC, Middle Square, Lehmer Generator y Fibonacci Generator) con la semilla proporcionada.
3. Realiza 10.000 iteraciones de cada generador guardando cada valor obtenido (valores entre 0 y 1) en una lista por cada generador definida previamente. También realizamos lo mismo en el generador propio de Python.
4. Sobre cada generador, realiza cuatro gráficos que nos permiten determinar:

   (a) Histograma que compara la distribución de los datos obtenida con la distribución uniforme esperada.

   (b) Como se distribuyen los valores respecto a la mediana. Si el valor esta por encima de la mediana, se indica 1, en caso contrario, se indica 0. Esto nos permite ver las rachas de cada generador, y concluir cual grado de aleatoridad tiene.

   (c) Comparar la desviación estándar obtenida por cada generador en relación a la obtenida con la hipótesis de que siguiera una distribución uniforme.

   (d) Tendencias, oscilaciones y agrupamientos a través de puntos negros correspondientes a cada valor obtenido.

5 Resultados

A continuación se presentan los gráficos generados por los distintos tests. Cada uno muestra el comportamiento de los datos a partir de la semilla proporcionada. Primero plasmaremos los resultados utilizando como semilla la hora del sistema:

Generador	Tipo de Generador	ChiSquaredUniformityTest	MediumTest	PatternTest	SumTest
GCL	Pseudoaleatorio	OK ($p = 0{,}113$)	4995 / 5001	Sin patrón visible	OK ($Z = 0{,}13$)
Middle Square	Pseudoaleatorio	OK ($p = 0{,}666$)	4948 / 5001	Sin patrón visible	OK ($Z = 1{,}69$)
Fibonacci	Pseudoaleatorio	ERROR ($p = 0{,}000$)	4994 / 5001	Patrón visible	ERROR ($Z = 3{,}99$)
Lehmer	Pseudoaleatorio	OK ($p = 0{,}805$)	5017 / 5001	Sin patrón visible	OK ($Z = 1{,}20$)
Python (random)	Pseudoaleatorio	OK ($p = 0{,}432$)	5014 / 5001	Sin patrón visible	OK ($Z = 0{,}60$)

Table 1: Resumen de resultados de las pruebas estadísticas con semilla automática (hora del sistema)

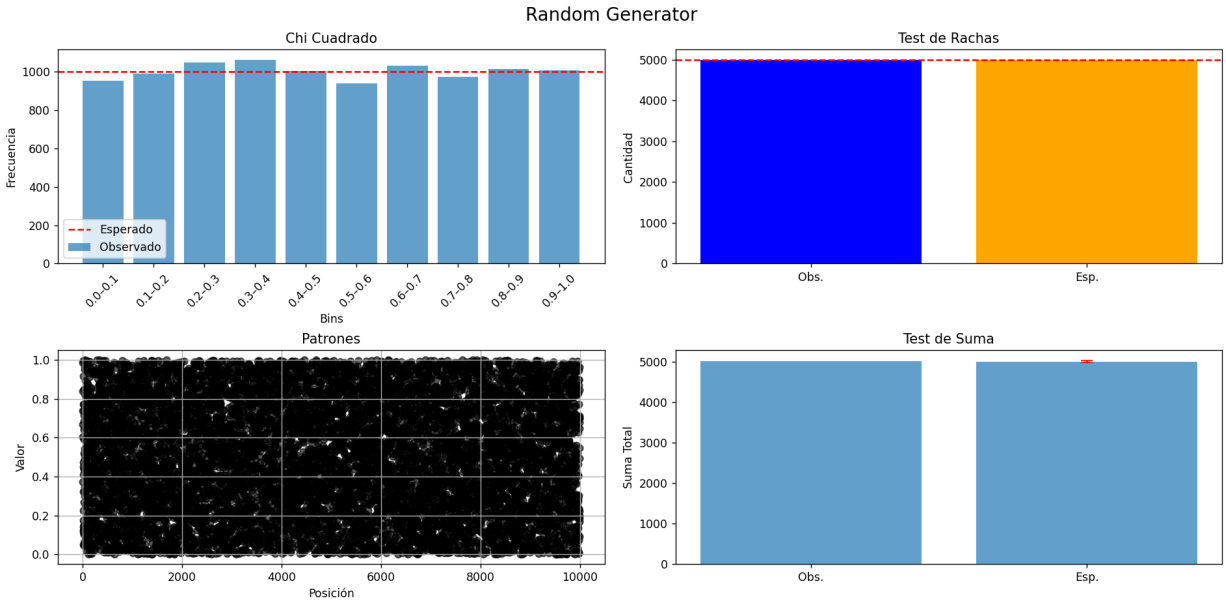

Figure 1: Random Generator con semilla automática (hora del sistema)

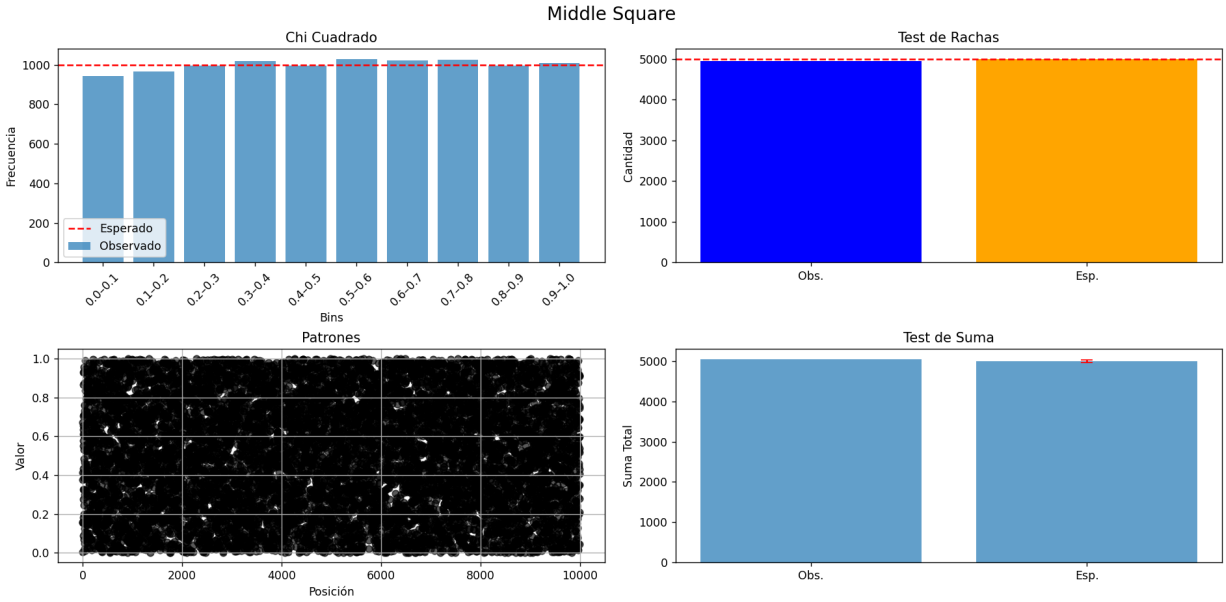

Figure 2: Middle Square con semilla automática (hora del sistema)

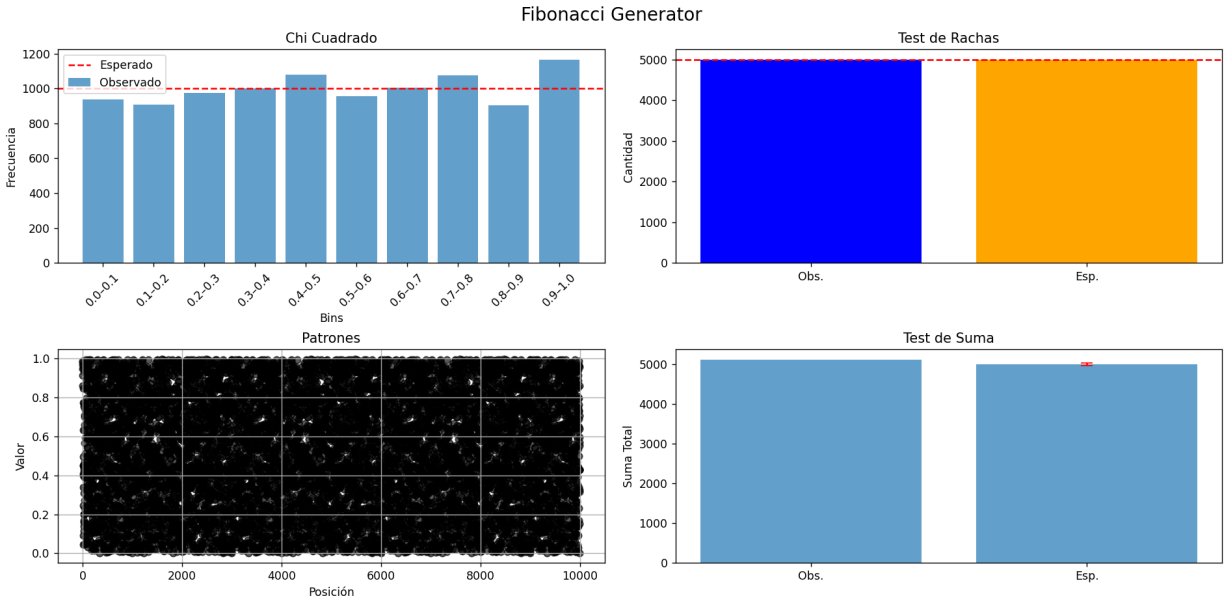

Figure 3: Fibonacci Generator con semilla automática (hora del sistema)

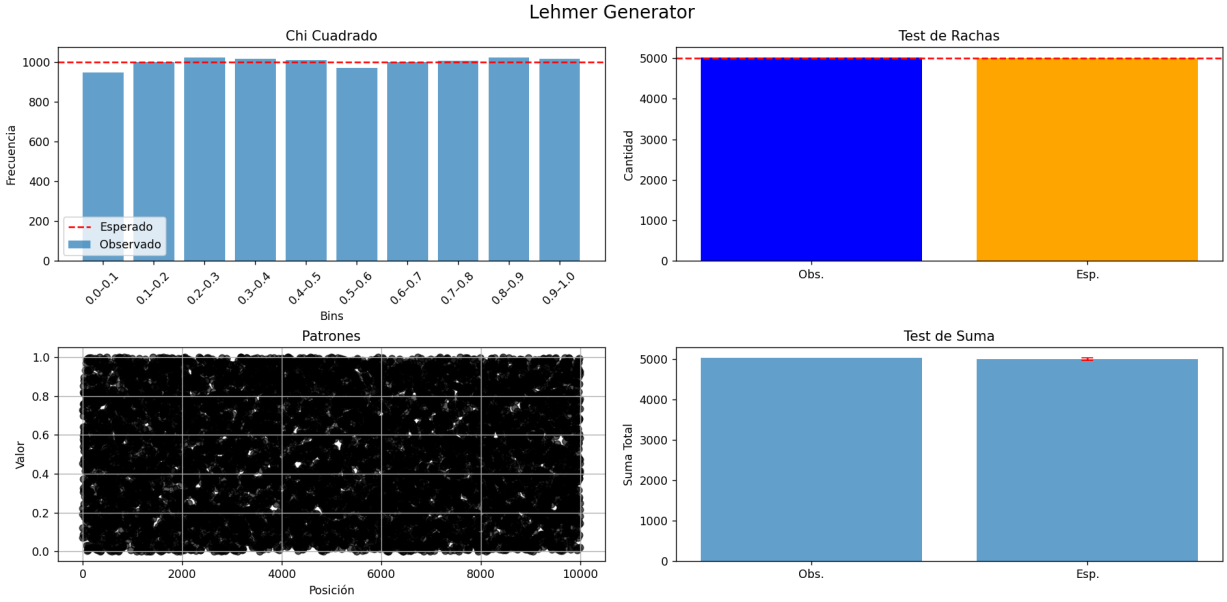

Figure 4: Lehmer Generator con semilla automática (hora del sistema)

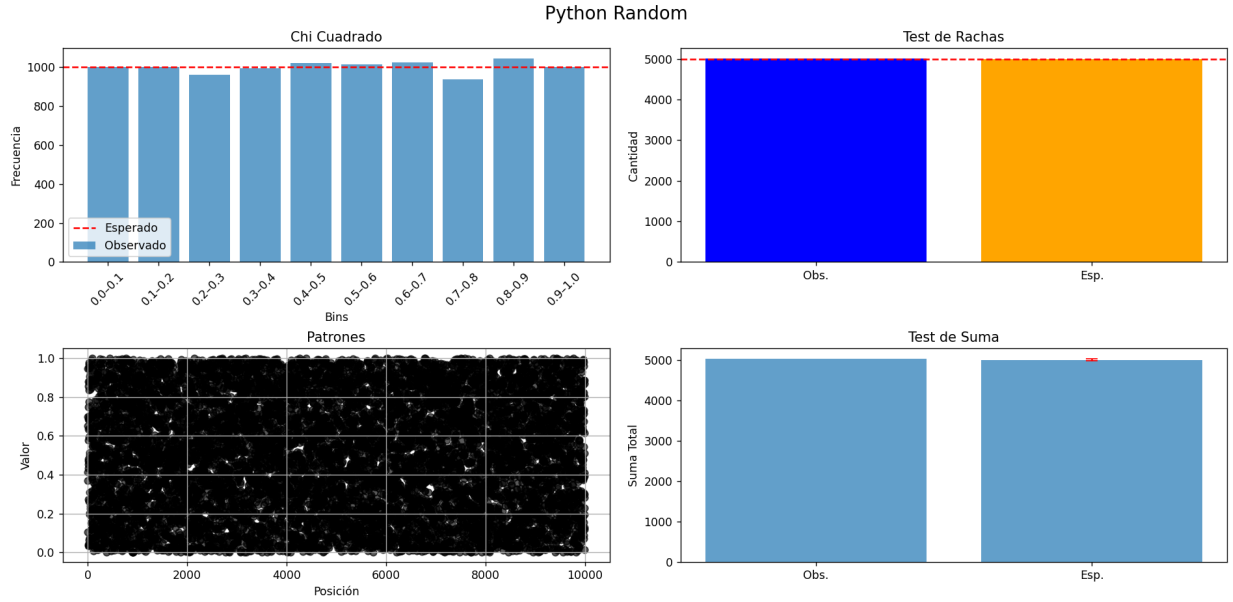

Figure 5: Python Random con semilla automática (hora del sistema)

Ahora, plasmaremos los resultados utilizando como semilla una proporcionada por el usuario:

Generador	Tipo de Generador	ChiSquaredUniformityTest	MediumTest	PatternTest	SumTest
GCL	Pseudoaleatorio	OK ($p = 0{,}406$)	4993 / 5001	Sin patrón visible	OK ($Z = 0{,}94$)
Middle Square	Pseudoaleatorio	OK ($p = 0{,}957$)	5097 / 5001	Patrón visible	OK ($Z = -0{,}05$)
Fibonacci	Pseudoaleatorio	OK ($p = 0{,}990$)	5020 / 5001	Sin patrón visible	OK ($Z = 0{,}20$)
Lehmer	Pseudoaleatorio	OK ($p = 0{,}172$)	4948 / 5001	Sin patrón visible	OK ($Z = -0{,}24$)
Python (random)	Pseudoaleatorio	OK ($p = 0{,}472$)	4980 / 5001	Sin patrón visible	OK ($Z = -1{,}15$)

Table 2: Resultados de las pruebas estadísticas con semilla `3791201610`

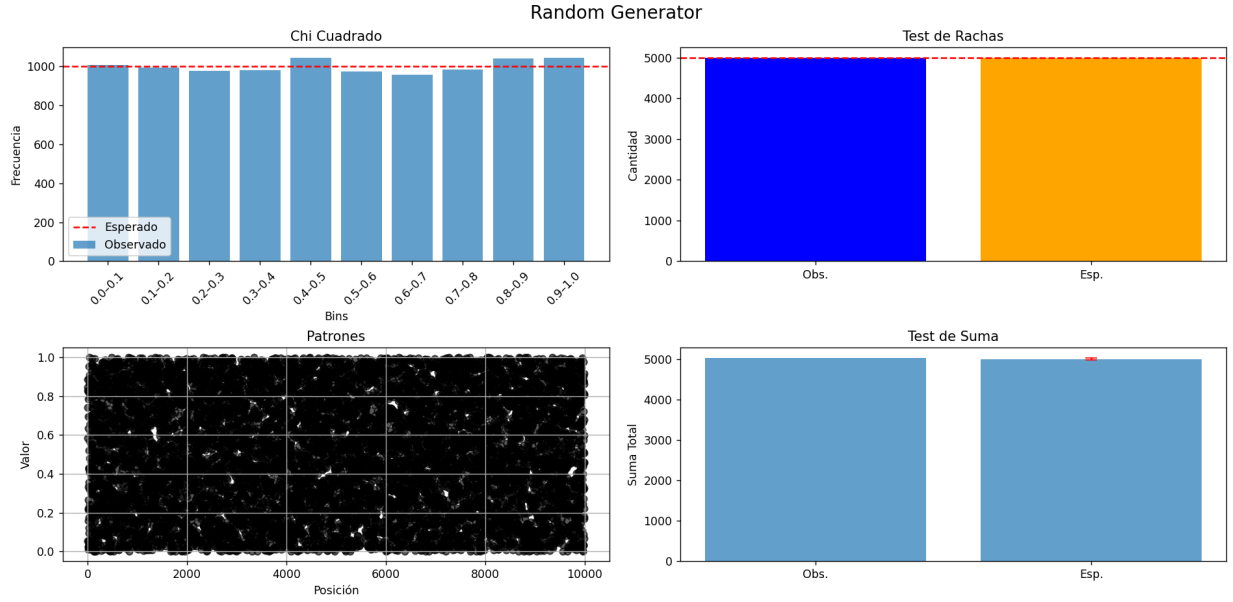

Figure 6: Random Generator con semilla inicial 3791201610

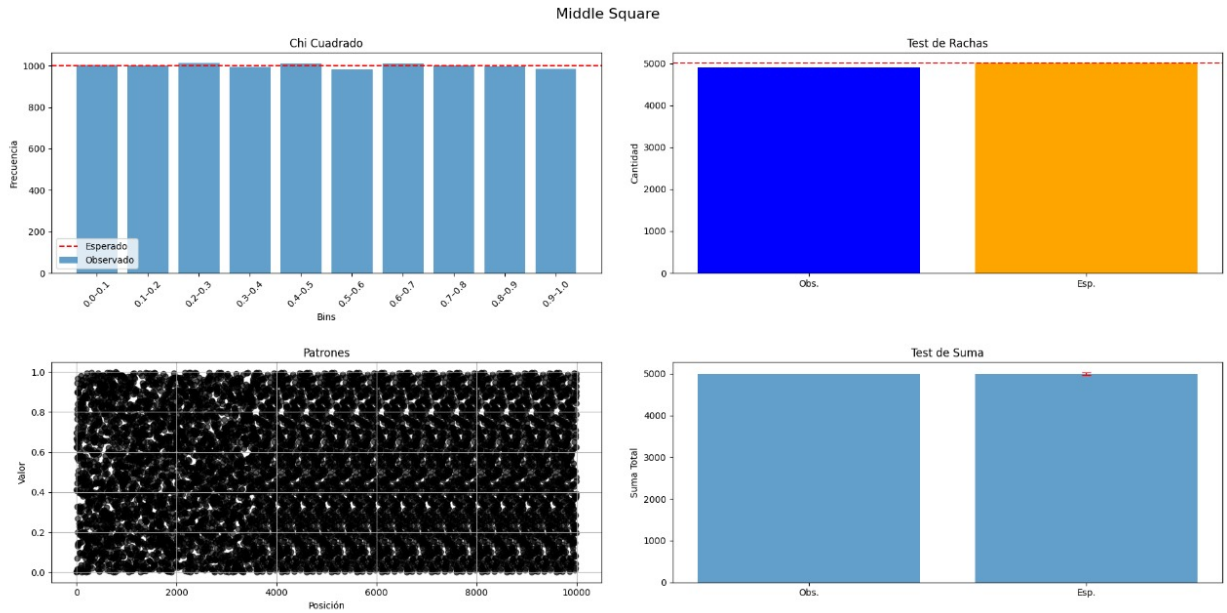

Figure 7: Middle Square con semilla inicial 3791201610

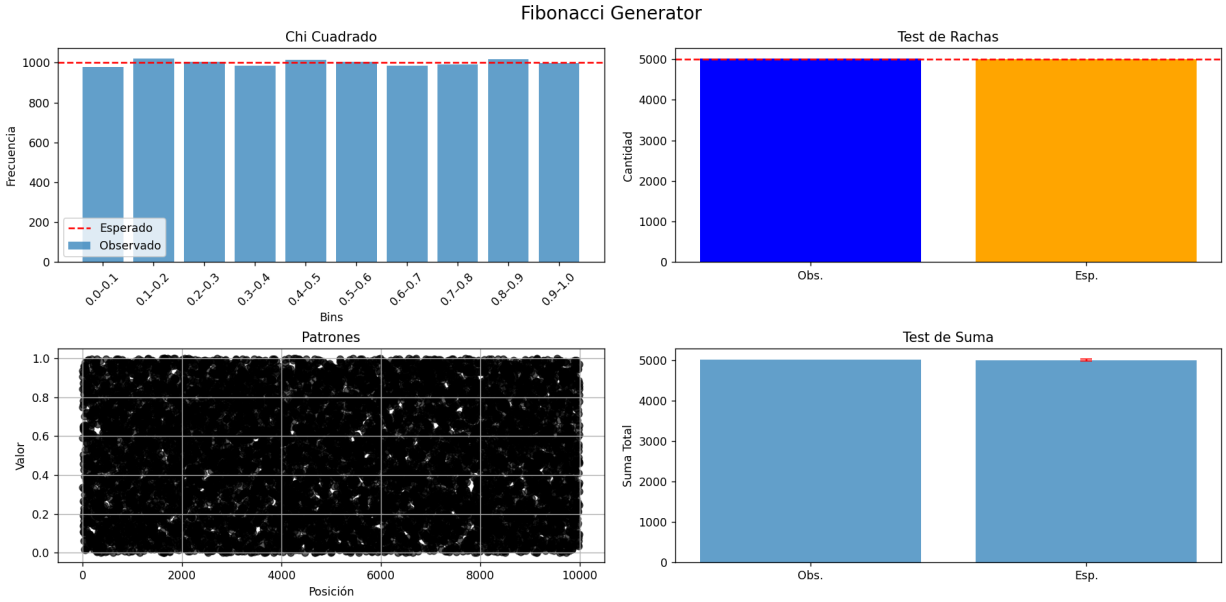

Figure 8: Fibonacci Generator con semilla inicial 3791201610

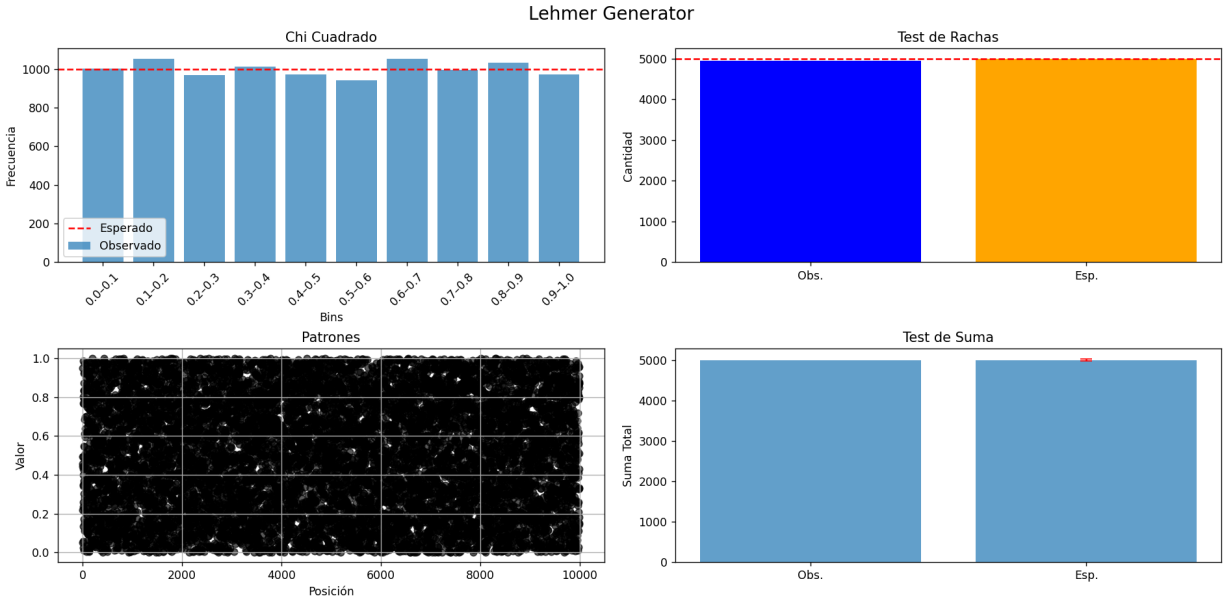

Figure 9: Lehmer Generator con semilla inicial 3791201610

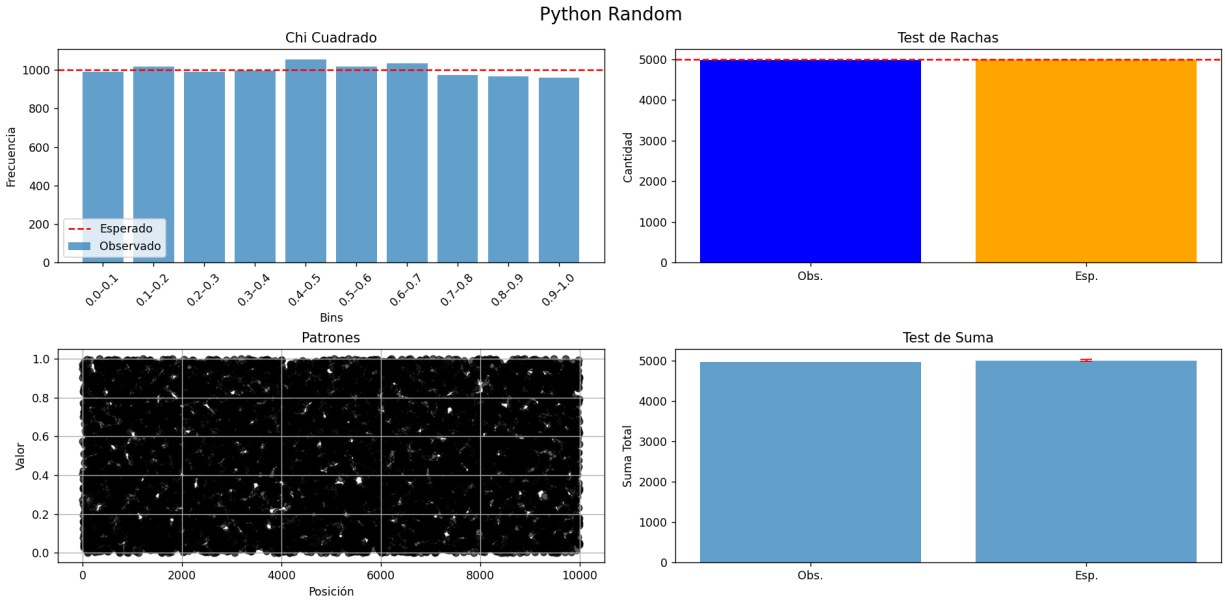

Figure 10: Python Random con semilla inicial 3791201610

6 Conclusión

6.1 Con semilla automática (hora del sistema)

Tomando en cuenta que utilizamos como semilla la hora del sistema, a partir del análisis de los generadores implementados y las pruebas estadísticas realizadas, podemos concluir que:

En general, los resultados obtenidos fueron positivos, con la mayoría de los generadores superando las pruebas estadísticas de uniformidad, rachas y suma total esperada.

El generador **GCL** mostró resultados aceptables, con un valor p = 0,113 y un Z-score de 0,13.

El generador **Middle Square**, también se comportó correctamente en esta ejecución, sin mostrar patrones visibles ni sesgo aparente (p = 0,666, Z = 1,69).

El generador **Fibonacci**, sin embargo, presentó resultados inesperadamente deficientes. Falló el test de uniformidad con un valor p prácticamente cero, y su suma total estuvo considerablemente sesgada (Z-score $\approx 3{,}99$), lo que sugiere una dependencia excesiva de la semilla inicial aleatoria.

Tanto **Lehmer** como el generador **Python (random)** funcionaron correctamente, manteniendo valores dentro de los rangos esperados en todas las pruebas aplicadas.

Este experimento evidencia que, incluso con semillas aleatorias, algunos generadores como Fibonacci pueden ser sensibles a las condiciones iniciales.

6.2 Con semilla inicial elegida por el usuario

Ahora, teniendo en cuenta que se usó como semilla una elegida por el usuario, podemos concluir que:

El generador **Middle Square**, obtuvo un valor $p = 0{,}957$ en el test de uniformidad, un Z-score cercano a 0 y mostró patrones visibles.

Los generadores **Fibonacci** y **Lehmer** mantuvieron su buen rendimiento, con distribuciones uniformes (p > 0,17) y valores medios dentro del rango esperado.

El generador **GCL**, si bien tuvo un valor p más ajustado (0,406), superó todas las pruebas sin desviaciones relevantes.

Finalmente, el generador **Python (random)** mostró un rendimiento muy bueno, como es habitual, sin presentar evidencia de sesgos ni estructuras.

En conclusión, todos los generadores evaluados en esta instancia se comportaron de forma adecuada, todos los generadores pasaron satisfactoriamente las tres pruebas.

7 Fórmulas empleadas

Los metodos utilizados fueron los siguientes:

1. **Generador Lineal Congruencial:** El Generador Lineal Congruencial (GLC) es uno de los métodos más antiguos y simples para generar números pseudoaleatorios.

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

   Donde:
   X_n es el numero pseudoaleatorio actual
   a es el multiplicador
   c es la constante aditiva del generador
   m es el modulo

2. **Middle Square Method:** Es un generador de números pseudoaleatorios propuesto por John von Neumann en 1949. Es un método simple, aunque poco usado debido a limitaciones, como ciclos cortos y tendencia a caer en cero o repetir valores.
   Este método parte de una semilla, un numero inicial de n dígitos. Luego se eleva al cuadrado, obteniendo un numero de 2n dígitos. Por ultimo, se toman los 4 numeros centrales del numero obtenido, y se repite el proceso.

3. **Fibonacci Generator:** Utiliza la secuencia de Fibonacci para generar números que aparentan ser aleatorios, es un tipo de generador recursivo.

   $$X_n = (X_{n-j} \operatorname{op} X_{n-k}) \mod m$$

   Donde:
   X_n es el numero pseudoaleatorio actual
   $X_{n-j}yX_{n-k}$ son los dos numeros anteriores a la secuencia actual
   jyk son las posiciones anteriores a la secuencia
   op es una operacion como suma, resta, XOR, etc
   m es el numero para limitar el valor maximo

4. **Lehmer Generator:** Es un método matemático eficiente para la generación de números pseudoaleatorios, es una variante del Generador Lineal Congruencial, pero sin termino aditivo.

$$X_{n+1} = (a \cdot X_n) \mod m$$

Donde:
X_n es el numero pseudoaleatorio actual
a es el multiplicador. Este debe elegirse con cuidado por que puede derivar en ciclos muy cortos o distribuciones poco aleatorias.
m es el modulo (usualmente numero primo grande o potencia de 2).

8 Enlaces de Interés

- Distintos tests para realizar
`https://www.random.org/analysis/`
- Evaluación y comparacion de Random.org y varios generadores comunmente usados
`https://www.random.org/analysis/Analysis2005.pdf`
- Resumen numeros pseudoaleatorios
`https://tereom.github.io/est-computacional-2018/numeros-pseudoaleatorios.html`