GENERADORES PSEUDOALEATORIOS - SIMULACIÓN 2023

Gardeñes Fernando
Cátedra de Simulación - ISI
UTN - FRRo
Zeballos 1341, S2000 Rosario, Santa Fe
fergardenes7@gmail.com

Biscaldi Ivan
Cátedra de Simulación - ISI
UTN - FRRo
Zeballos 1341, S2000 Rosario, Santa Fe
ibiscaldi@frro.utn.edu.ar

Saludas Fausto
Cátedra de Simulación - ISI
UTN - FRRo
Zeballos 1341, S2000 Rosario, Santa Fe
fausaludas14@gmail.com

Schlieper Tadeo
Cátedra de Simulación - ISI
UTN - FRRo
Zeballos 1341, S2000 Rosario, Santa Fe
tadeo.sch@hotmail.com

Nicolás Fierro
Cátedra de Simulación - ISI
UTN - FRRo
Zeballos 1341, S2000 Rosario, Santa Fe
nicofierro1@gmail.com

29 de mayo de 2023

ABSTRACT

Este trabajo tiene como objetivo explorar la aleatoriedad de los números 'aleatorios' generados por computadoras. Se presentaran diferentes técnicas para generar secuencias de números aparentemente aleatorios y se analizaran sus características y propiedades.

1. Introducción

Los números aleatorios son fundamentales en las simulaciones ya que representan la incertidumbre en el modelado del sistema. Sin embargo, la aleatoriedad de los números generados por computadora es limitada y depende del algoritmo utilizado.

Un número pseudoaleatorio es un número generado en un proceso que parece producir números al azar, pero no hace realmente.

Los procesos de generación de estos números son llevados a cabo por algoritmos completamente deterministas, pues estos ante mismas condiciones iniciales producen siempre el mismo resultado. Se denomina a cada uno de estos procesos/algoritmos, *Generador de números pseudoaleatorios* o RNG según sus siglas en ingles(Random number generator).

Una secuencia de números pseudoaleatorios no debería mostrar ningún patrón o regularidad aparente desde un punto de vista estadístico. Para comprobar esto sera necesario realizar pruebas que evalúen la existencia o no de tales patrones.

2. Generadores de números aleatorios reales[1]

Los generadores de números aleatorios reales (RNG) son utilizados en una amplia gama de aplicaciones, desde simulaciones científicas e ingenieriles hasta juegos de azar. A diferencia de los generadores de números pseudoaleatorios, los generadores de números aleatorios reales utilizan fuentes de ruido físico, como el ruido térmico, el ruido eléctrico o

el ruido atmosférico, para generar números verdaderamente aleatorios. Estos números aleatorios reales son esenciales en aplicaciones donde se requiere una verdadera aleatoriedad, como en la criptografía o la generación de claves.

Para generar números aleatorios reales, se utilizan diversos dispositivos que convierten el ruido físico en una señal eléctrica que luego es digitalizada y procesada por algoritmos de generación de números aleatorios. Estos algoritmos se encargan de eliminar cualquier patrón o correlación que pueda existir en la señal para garantizar que los números generados sean verdaderamente aleatorios.

A pesar de su importancia, los generadores de números aleatorios reales también presentan desafíos importantes, como la posibilidad de errores en la fuente de ruido físico o en el proceso de digitalización, que pueden introducir patrones en la señal y afectar la aleatoriedad de los números generados. Por esta razón, es importante realizar pruebas y análisis estadísticos rigurosos para evaluar la calidad y fiabilidad de los generadores de números aleatorios reales.

3. Generadores de números pseudoaleatorios[2]

Dependiendo del modelo que uno busque simular, esto puede requerir cientos, miles o millones de números. Y para cada comportamiento particular de una simulación sera conveniente o no que estos números sigan determinada distribución. Independientemente de la distribución que se requiera, una característica en común que se busca en todos los conjuntos de números a generar es que estos sean 'aleatorios'. Por ello el primer paso siempre es obtener un conjunto de números al que denominaremos 'pseudoaleatorio' dada la dificultad de que estos sean realmente aleatorios.

Una característica de estos conjunto de números es que todos están dentro del intervalo $(0, 1)$. En adelante se los simbolizara a los números pseudoaleatorios como el conjunto r_i:

$$r_i = \{r_1, r_2, r_3, ...r_n\} \quad \textit{Números pseudoaleatorios} \tag{3.1}$$

donde n es el tamaño de la secuencia y se lo conoce como ciclo de vida del generador que creo la secuencia.

Para cada variable aleatoria es necesario crear un gran conjunto r_i. En el pasado era normal buscar que $n > 2^{31}$, actualmente un procesador puede generar más de 2^{200} números.

¿Por qué es necesaria una secuencia de números pseudoaleatorios tan grande?
Si bien los elementos de r_i no están directamente involucrados en una simulación, para poder generar los valores de una variable aleatoria se utilizan los números pseudoaleatorios r_i. Cada variable aleatoria en una simulación puede requerir miles de valores. La extensión temporal de una simulación aumentará exponencialmente los valores que se requieren. Además nunca alcanza con una sola simulación para llegar a una conclusión o comprender el comportamiento de un sistema. Si el ciclo de vida del generador fuera pequeño las simulaciones terminarían volviéndose un proceso cíclico y no aleatorio, y esto llevaría conclusiones equivocadas.

3.1. Condiciones necesarias del conjunto generado

Para generar números pseudoaleatorios es necesario un algoritmo de generación que genere a r_i a partir de determinados parámetros. Los parámetros mas importantes a los que debe responder un generador, y que nunca deben faltar son:

1. N: Ciclo de vida o tamaño de r_i
2. s: Semilla(seed) a partir de la cual se genera la secuencia r_i

Diferentes algoritmos pueden requerir otros parámetros adicionales.

Estos algoritmos son deterministas(ante los mismos parámetros se generan los mismos conjuntos), secuenciales, y realizan las siguientes funciones:

1. **Inicialización**: Recibe un número (la semilla) y pone al generador en su estado inicial
2. **Transición**:Se transforma el estado del generador
3. **Salida**: A partir del estado del generador se genera un numero fijo de bits(unos o ceros)

Una sucesión de bits pseudoaleatorios se obtiene definiendo la semilla y llamando repetidamente la función de transición y la función de salidas. Esto implica, entre otras cosas, que una sucesión de números pseudoaleatorios está completamente determinada por la semilla.

El algoritmo de generación debe, ante diferentes parámetros, generar un conjunto r_i capaz de superar pruebas de independencia y uniformidad. Por ello es importante tener en cuenta que el conjunto r_i generado debe cumplir de mínimo con las siguientes condiciones:

1. El ciclo de vida debe ser lo suficientemente grande

$$N > 2^{31} \tag{3.1.1}$$

2. Los elementos de r_i deben ser continuos

$$r_i \in \Re \forall i \in [0, N] \tag{3.1.2}$$

3. La media μ_r de r_i no debe estar lejos de $0{,}5$

$$\mu_r \approx \frac{1}{2} \tag{3.1.3}$$

4. La varianza v_r de r_i no debe ser grande ni pequeña

$$v_r \approx \frac{1}{12} \tag{3.1.4}$$

5. La media de cualquier subconjunto de r_i de elementos contiguos no puede estar ni muy por encima ni muy por debajo de μ_r

$$\nexists w = \{a_t, a_{t+1}, a_{t+2}, ...a_{t+u}\} \subset r_i, t \in [0, N), u < N - t/\mu_w \lll \mu_r \vee \mu_w \ggg \mu_r \tag{3.1.5}$$

6. No pueden presentarse patrones de crecimiento o decrecimiento en secuencias de elementos de r_i

$$\nexists w = \{a_t, a_{t+1}, a_{t+2}, ...a_{t+u}\} \subset r_i / a_t \leq a_{t+1} \leq a_{t+2} \leq ...a_{t+u} \quad \vee \quad a_t \geq a_{t+1} \geq a_{t+2} \geq ...a_{t+u} \tag{3.1.6}$$

A continuación se presentaran algunos generadores de números pseudoaleatorios (PRNG).

3.2. Generadores congruenciales lineales- GCL

Los generadores congruenciales lineales(GCL[3]) se introdujeron en 1949 por D.H. Lehemer y son algoritmos que permiten obtener una secuencia de números pseudoaleatorios calculados a partir de una función lineal definida a trozos discontinua, cuya elección de los parámetros determina su calidad para generar tal secuencia y que esta sea 'aleatoria'.

Un generador se define mediante la relación de recurrencia:

$$X_{n+1} = (a \cdot X_n + c) mod(m) \quad \textit{Relacion de recurrencia de un GCL} \tag{3.2.1}$$

Dónde los parámetros son los siguientes y deben cumplir que:

- Módulo: $m \in N \quad | \quad m > 0$
- Multiplicador: $a \in N \quad | \quad 0 < a < m$
- Incremento: $c \in N \quad | \quad 0 \leq c < m$
- Semilla o valor inicial: $X_0 \in N \quad | \quad 0 \leq c < m$

Dado un m, este es la máxima longitud de ciclo posible, es decir, la mayor cantidad posible de elementos que se suceden antes de que estos empiecen a repetirse. Es conveniente que sea lo suficientemente grande, de lo contrario los números obtenidos serian cíclicos y conducirían a errores o sesgos durante una simulación.

Las elecciones de parámetros resultan extremadamente sensible respecto a la calidad de los números producidos por un GCL. Por ejemplo un GCL donde $a = 1$ y $c = 1$ resultara en un simple contador de modulo m, lo cual puede tener un modulo grande pero de ninguna forma sera aleatorio. Ademas una mala elección de X_0 para determinados valores de a y c puede causar que el ciclo sea mucho menos que m. Dado el efecto de los parámetros elegidos sobre la secuencia es posible subdividir a los GLCs en familias de parametros.

3.2.1. Familias de GLCs con c=0 y m primo

A esta familia pertenecen los GLCs propuestos por Lehemer. Si el multiplicador elegido es un numero primo, la longitud del ciclo sera $m-1$ y la semilla deberá estar entre 0 y $m-1$.

Para esta familia de GLCs resulta comun que m sea un numero primo de Mersenne:

$$m = 2^w - 1 \qquad \textit{donde w es un numero primo}$$

La desventaja de esta familia de GLC es que el calculo de la reducción modular resulta mas complicado para la computadora, ya que requiere un producto de doble ancho (ver multiplicación modular de Montgomery[4]) y pasos de reducción explicita. Pues cuando $m = 2^w - d$ la reducción modular puede computarse como $(aXmod2^w) + d\lfloor \frac{aX}{2^w} \rfloor$ y a esto se le hace una resta condicional de m cuando el resultado es muy grande, limitando el numero de sustracciones a $\frac{a\,d}{m}$(y esta es la razón por la cual resulta conveniente usar un numero primo de Mersenne).

Otro problema que puede surgir es que no sea posbile realizar productos de doble ancho en cuyo caso se utiliza el método de Schrage[5], el cual establece que:

$$(a\,x)mod(m) = g(x) + m \cdot h(x) \tag{3.2.2}$$

Donde

$$g(x) = a(x\,mod\,q) - r \cdot (x\,div\,q) \tag{3.2.3}$$

y

$$h(x) = (x\,div\,q) - (ax\,div\,m) \tag{3.2.4}$$

siendo

$$q = m\,div\,a \quad y \quad r = m\,mod\,a \tag{3.2.5}$$

3.2.2. Familias de GLCs con c=0 y m potencia de 2

Esto permite mayor eficiencia del GLC ya que los cálculos del modulo pueden simplificarse truncando bits de la representación binaria.

En esta familia de GLCs el problema es que la maxima longitud de ciclo es $m/4$ y esto sucede cuando el multiplicador a es congruente de modulo 8, con 3 o 5 ($a \equiv 3 \bmod 8 \parallel a \equiv 5 \bmod 8$), pero cuando $a \equiv 5 \bmod 8$ el bit mas bajo no cambia nunca y el siguiente alterna (como consecuencia X es siempre impar) y cuando $a \equiv 3 \bmod 8$ el bit mas bajo alterna y el siguiente es constante.

3.2.3. Familia de GLCs con $c \neq 0$

Con una correcta elección de c es posible lograr que el ciclo del GLC sea de longitud m. Para ello es necesario que se cumpla el teorema de Hull-Dobell:

- m y c son primos relativos.
- a-1 es divisible por todos los factores primos de m.
- a-1 es divisible por 4 si m es divisible por 4

Vale la pena notar que un periodo grande no determina que el generador congruencial es bueno, debemos verificar que los números que generan se comportan como si fueran aleatorios. Los GCLs continúan siendo utilizados en muchas aplicaciones porque con una elección cuidadosa de los parámetros pueden pasar muchas pruebas de aleatoriedad, son rápidos y requieren poca memoria. Por ejemplo:

Fuente	m	(multiplicador) a	(incremento) c
Borland C/C++	2^{32}	22695477	1
Java	$2^{48}-1$	25214903917	11

3.2.4. Números pseudoaleatorios a partir de un GLC

Es importante tener en cuenta que los GLC mencionados generan secuencias de números enteros, y esto contradice que los números pseudoaleatorios deben estar en el rango $[0, 1]$. Para obtener el conjunto r_i mencionado en 3.1 se realiza la siguiente conversión:

$$r_i = \frac{X_i}{m-1} \qquad \textit{Numeros pseudoaleatorios generados por GLC} \tag{3.2.6}$$

donde X_i son los elementos de la secuencia generada por el GLC y m es el multiplicador del GLC.

3.3. Algoritmo de cuadrados medios

En la década de 1940 John Von Neumman y Nicholas Metropolis propusieron el algoritmo de media de cuadrados[2] para generar secuencias de números pseudoaleatorios.

El algoritmo consiste en que a partir de un numero(semilla) el cual debe tener D dígitos ($D > 3$), se eleva al cuadrado, y de este se seleccionan los D dígitos del centro (*medio*). Originalmente estos dígitos se utilizaban como resultado y a su vez para generar el próximo numero de la secuencia. Pero dado que el numero obtenido es entero y en general no puede pertenecer a r_i, para convertir este numero en pseudoaleatorio se agrega '0.' como prefijo a los dígitos obtenidos.

El algoritmo paso a paso:

1. Se inicia con una semilla x_0 de D dígitos.

$$x_0 \rightarrow \textit{Semilla}$$

2. La semilla se eleva al cuadrado, produciendo un número de $D \cdot 2$ dígitos (si el resultado tiene menos dígitos se añaden ceros al inicio)

$$X_0 = (x_0)^2 \tag{3.3.1}$$

3. Los D dígitos del centro serán el resultado y se utilizaran para generar el próximo numero de la secuencia

$$Y_i \leftarrow D \textit{ dígitos del centro de } (X_i) \tag{3.3.2}$$

$$X_{i+1} = (Y_i)^2 \tag{3.3.3}$$

4. Se toma r_i tal que los dígitos de Y_i son las cifras decimales

$$r_i = 0.Y_i \tag{3.3.4}$$

5. Se repiten los pasos 3 y 4 hasta obtener la cantidad de números deseados.

El inconveniente de este método es que tiene una fuerte tendencia a degenerar a cero rápidamente y que además los números generados pueden repetirse cíclicamente después de una secuencia corta. Por ejemplo si $D = 4$ y $x_0 = 1000$, $Y_1 = 0000$ y la secuencia degenera en 0.

3.4. Algoritmo de productos medios

Con una mecánica parecida a la del algoritmo de cuadrados medios, se parte de 2 números (semillas)con D dígitos($D > 3$), pero en este caso se los multiplicara entre si para generar un numero de $2 \cdot D$ dígitos del cual se tomaran los D dígitos del centro. Luego para continuar la secuencia se toma el segundo numero del producto y el nuevo numero (ambos de D dígitos) y se los multiplica entre si, tomando los D dígitos del centro y se repite este proceso las veces que sean deseadas.

Paso a paso:

1. Se seleccionan 2 números con D dígitos

$$X_0, X_1 \rightarrow \textit{Semillas}$$

2. Se multiplican a los numeros entre si para obtener uno de $2D$ digitos (rellenar con ceros al inicio en caso de que falten) y se toma este como resultado

$$Y_0 = X_0 \cdot X_1 \tag{3.4.1}$$

$$Y_1 \leftarrow D \textit{ digitos del centro de } Y_0 \qquad \textit{(Primer numero generado)} \tag{3.4.2}$$

3. Los D dígitos del centro se toman como resultado, y se utilizan para generar el próximo

$$X_{i+1} = Y_i \qquad \forall i \in [1, n) \tag{3.4.3}$$

$$Y_{i+1} \leftarrow \textit{D digitos del centro de } (X_i \cdot X_{i+1}) \qquad \forall i \in [1, n) \tag{3.4.4}$$

4. Se toma r_i ($i \geq 1$) tal que los dígitos de Y_i son las cifras decimales

$$r_i = 0.Y_i \qquad \forall i \in [1, n] \tag{3.4.5}$$

5. Se repiten 3 y 4 hasta obtener los $n - 1$ numeros deseados

3.5. Algoritmo de multiplicador constante

Usando similar mecanismo que en el algoritmo anterior, pero en este caso usando semillas a a y X_0 (números con D dígitos donde $D > 3$), donde a sera el multiplicador constante a utilizar en todos los pasos.

Paso a paso:

1. Se seleccionan 2 números con D dígitos

$$a \rightarrow \textit{Multiplicador constante}$$

$$X_0 \rightarrow \textit{Semilla}$$

2. Se multiplican a los números entre si para obtener uno de $2D$ dígitos (rellenar con ceros al inicio en caso de que falten) y se toma este como resultado

$$Y_0 \leftarrow \textit{D digitos del centro de } (a \cdot X_0) \tag{3.5.1}$$

3. Los D dígitos del centro se toman como resultado, y se utilizan para generar el próximo

$$X_i = Y_{i-1} \qquad \forall i \in [1, n) \tag{3.5.2}$$

$$Y_i \leftarrow \textit{D digitos del centro de } (a \cdot X_i) \qquad \forall i \in [1, n) \tag{3.5.3}$$

4. Se toma r_i ($i \geq 1$) tal que los dígitos de Y_i son las cifras decimales

$$r_i = 0.Y_i \qquad \forall i \in [0, n] \tag{3.5.4}$$

5. Se repiten 3 y 4 hasta obtener los n numeros deseados

4. Test de determinación de comportamiento

La calidad de un conjunto de números pseudoaleatorios queda determinada en gran medida por su *uniformidad* e *independencia*. Existen diferentes test de comportamiento que permiten evaluar si una secuencia de números pseudoaleatorios cuentan con estas características.

Estos tests pueden categorizarse según el criterio que evalúan: independencia o uniformidad.

4.1. Tests de Uniformidad

Para evaluar la uniformidad del conjunto generado, es necesario comparar su distribución respecto a la distribución uniforme.

Al testear la uniformidad, las hipótesis son:

$$H_0 : R_i \sim U[0, 1]$$

$$H_1 : R_i \nsim U[0, 1]$$

La *Hipótesis nula*[8] [1] H_0 se lee tal que los números están distribuidos uniformemente en el intervalo [0,1]. Si la Hipótesis Nula no puede ser rechazada, significa que no se ha detectado evidencia de no-uniformidad en la muestra. Esto no significa que no se necesiten más tests de uniformidad.

H_1 la hipótesis alternativa [2] indica que los números no están distribuidos uniformemente. Cuando los datos evidenciaran que esta hipótesis se cumple, entonces debe rechazarse la hipótesis nula.

4.1.1. Chi-cuadrado

El test de chi-cuadrado fue publicado por Karl Pearson en 1900. La notación original incluía el uso de x^2, de donde derivó el nombre. Este test devuelve una probabilidad aproximada de que tan posible es la ocurrencia de una determinada salida.

Suponiendo que tenemos n observaciones independientes (en este caso elementos de una secuencia generada por RNG), donde cada uno cae en cierta categoría k. Sea Y_s el número de observaciones en la s-ésima categoría y p_s la probabilidad de que una observación caiga en la categoría s. Entonces podemos esperar que

$$Y_s \approx np_s$$

para grandes valores de n. Para calcular que tan 'alejados' estamos de los valores esperados, se define el estadístico V_i (donde i se refiere a 'inadecuado') como sigue:

$$V_i = (Y_1 - np_1)^2 + (Y_2 - np_2)^2 + ... + (Y_k - np_k)^2 \tag{4.1.1}$$

se elevan los términos al cuadrado para obtener valores positivos. Cabe destacar que V_i le da el mismo 'peso' a cada categoría, si cada p_s fuera distinto, entonces V_i no sería correcto ya que haría más énfasis en algunas discrepancias y podría no detectar otras. Entonces definimos al estadístico V (llamado Chi-cuadrado de $Y_1, ..., Y_k$) como:

$$V = \frac{(Y_1 - np_1)^2}{np_1} + \frac{(Y_2 - np_2)^2}{np_2} + ... + \frac{(Y_k - np_k)^2}{np_k} = \sum_{i=1}^{k} \frac{(Y_i - np_i)^2}{np_i} \tag{4.1.2}$$

Donde Y_i también puede identificarse como la frecuencia observada en el intervalo, y np_i como la frecuencia esperada en el intervalo.

Para definir si el valor V es razonable o no, debemos hacer referencia a la tabla X^2. Se busca en la línea de $v = k - 1$ y, junto con un intervalo de confianza predefinido al cuál llamaremos q, se compara V con el valor en la fila v y columna q, entonces

- si $V <$ al valor de tabla, se afirma la hipótesis nula (distribución es uniforme) con q % de confianza.
- si $V >$ al valor de tabla, se rechaza la hipótesis nula (distribución no uniforme) con q % de confianza.

4.1.2. Kolmogorov-Smirnov

El test de Kolmogorov-Smirnov (KS) fue presentado originalmente por A. N. Kolmolov en 1933, y revisado por N.V. Smirnov en 1939. Resulta útil en las áreas en la que el test Chi-cuadrado no, y ambos tests pueden ser utilizados en conjunto.

Primero presentaremos la *Función de distribución acumulada* (cdf por sus siglas en inglés), dada una variable aleatoria X, la cdf $F_X(x)$ se define como:

$$F_X(x) = \textit{ probabilidad de que } (X \leq x) \tag{4.1.3}$$

Cabe destacar que $F_X(x)$ tiene rango $[0, 1]$ y sera creciente (o constante en algunos intervalos) mientras x crece desde $-\infty$ hacia $+\infty$.

[1] Hipótesis nula: supone la veracidad de lo afirmado a menos que los datos de alguna muestra sirvan de evidencia de que sucede lo contrario

[2] Hipótesis alternativa: la hipótesis alterativa(H_1) se contrasta con la hipótesis nula, es decir, indica lo contrario. Cuando la evidencia indicara que se cumple la hipótesis alternativa, esto implica que hay datos suficientes para rechazar la hipótesis nula.

Si X toma los valores de una secuencia generada por un RNG, para el test KS se requiere que $F_X(x)$ sea continua. Los valores generados pueden tomar cualquier valor dentro de cierto intervalo, llegando a una cdf continua. Este $F_X(x)$ es la distribución teórica esperada del RNG.

Suponiendo que tomamos n observaciones independientes de X, creado los valores $X_1, X_2, ..., X_n$, definimos la *distribución empírica de la función* $F_n(x)$ como:

$$F_n(x) = \frac{\textit{numero de } X_1, X_2, ..., X_n \textit{ que son } \leq x}{n} \tag{4.1.4}$$

El test KS compara $F_X(x)$ con $F_n(x)$ midiendo la diferencia entre las dos funciones de distribución. Cuando n es lo suficientemente grande, se puede esperar que las dos funciones sean similares si la secuencia examinada es realmente aleatoria. Por lo tanto, se mide la diferencia entre las funciones creando los siguientes estadísticos:

$$K_n^+ = (F_n(x) - F_X(x))\sqrt{n} \qquad -\infty < x < +\infty \tag{4.1.5}$$

$$K_n^- = (F_X(x) - F_n(x))\sqrt{n} \qquad -\infty < x < +\infty \tag{4.1.6}$$

Es decir, K_n^+ es la desviación mayor cuando F_n es mayor que F_X, mientras que F_n^- es la mayor desviación cuando F_n es menor que F_X. El $\sqrt{n}$ está presente porque para una x fija, la desviación estándar de $F_n(x)$ es proporcional a $1\sqrt{n}$, por lo que multiplicar por $\sqrt{n}$ permite que tanto K_n^+ como K_n^- sean independientes de n.

Luego, estos estadísticos son utilizados para comparar los valores en la tabla de distribución de K_n^+ y K_n^-.

Similar al test Chi-cuadrado, el valor de n debe ser elegido de forma tal que sea lo suficientemente grande como para detectar si las funciones $F_n(x)$ y $F_X(x)$ son diferentes, pero no demasiado grandes.

Se pueden definir los siguientes pasos a seguir:

1. Ordenar los elementos de la secuencia de menor a mayor, donde X_i sea la i-ésima observación más pequeña, tal que:
$$X_1, X_2, ..., X_i, ..., X_n$$
2. Calcular K_n^+ y K_n^-
3. Localizar los valores en la tabla para un valor específico de confianza y el valor n.
4. Si el valor de la muestra K_n es mayor al valor de la tabla, la hipótesis nula es rechazada.
   Si el valor de la muestra K_n es menor al valor de la tabla, no se detecta diferencia entre la distribución de la muestra y la distribución uniforme, la hipótesis nula es aceptada.

4.2. Tests de Independencia

Para evaluar la independencia del conjunto generado, se busca la existencia de alguna relación entre diferentes muestras del conjunto. En caso de no encontrarse alguna relación entonces el conjunto es independiente.

Al testear independencia, las hipótesis son:

$$H_0 : R_i \sim independiente$$
$$H_1 : R_i \nsim independiente$$

Esta *Hipótesis nula* H_0 se lee tal que los números son independientes. Si no puede rechazarse la hipótesis significa que no se ha detectado evidencia de dependencia. Esto no implica que no se deban realizar más tests.

Para cada test, se debe establecer un nivel de significancia α, esto es la probabilidad de rechazar la hipótesis nula cuando esta es verdadera:

$$\alpha = P(\textit{rechazo de } H_0 | H_0 \textit{ es verdadero}) \tag{4.2.1}$$

Por lo general se utiliza $\alpha = 0{,}01$ o $0{,}05$. Si varios test se llevan a cabo sobre la misma muestra de la secuencia de números generados, la probabilidad de rechazar la hipótesis nula aumenta. Por ejemplo si $\alpha = 0{,}05$ y se realizan 5 tests diferentes sobre la muestra, la probabilidad de rechazar la hipótesis nula en al menos uno de los test puede llegar a ser $0{,}25$.

Así mismo, si un solo test se lleva a cabo sobre varias muestras, la probabilidad también aumenta. Por ejemplo, si 100 muestras son probadas por el mismo test, con $\alpha = 0{,}05$, se puede esperar que al menos 5 de los test sean rechazados.

4.2.1. Test de Autocorrelación

Los test de autocorrelación[10] se centran en la dependencia entre los números de una secuencia.

Dada una secuencia $Y_1, Y_2, Y_3, ...Y_N$ que se corresponde con los instantes $X_1, X_2, X_3, ...X_N$, y cuya media es $\overline{Y}$ se define como función de autocorrelación con retraso k a:

$$r_k = \frac{\sum_{i=1}^{N-k} \left(Y_i - \overline{Y}\right)\left(Y_{i+k} - \overline{Y}\right)}{\sum_{i=1}^{N} \left(Y_i - \overline{Y}\right)^2} \tag{4.2.2}$$

A diferencia de la correlación, que es la medida en que los valores de 2 variables están relacionados entre si, la autocorrelación es la medida en que 2 elementos de una misma variable están relacionados en los momentos X_i e X_{i+k}.

Para detectar la no aleatoriedad de un conjunto, es común que se evalúe únicamente el primer retraso ($k = 1$).

r_k es la medida en que los k-ésimos elementos están autocorrelacionados. Cuando sus valores son cercanos a 1 o -1, entonces la autocorrelación es grande y los k-ésimos números no son independientes. *Observación: cuando* $k = 0$ $r_k = 1$*, esto sucede porque se esta comparando a los elementos con si mismos*

4.2.2. Test de rachas (arriba y abajo de la media)[11]

El test de rachas más preciso para comprobar aleatoriedad es aquel que cuenta el número de valores que hay por encima y por debajo de la media (en general de cualquier cuantil). Lo que se busca es la existencia de patrones estadísticamente inaceptables de incremento o decrecimiento entre números adyacentes.

Primero se determina una nueva secuencia S, que se obtiene de reemplazar con 1 a los elementos de r_i por encima de la media, y con 0 a los que están por debajo.
Luego se determinaran los valores:

- n_0: ocurrencia de ceros en S
- n_1:ocurrencia de unos en S
- c_0:cantidad de subsecuencias de unos consecutivos o ceros consecutivos en S
- $N = n_0 + n_1$

Luego se calcula el valor de μ_{c_0}, que es la esperanza matemática de c_0 que se puede estimar según n_0 y n_1, y también $\sigma^2_{c_0}$ que es la varianza o desviación cuadrada.

$$\mu_{c_0} = \frac{2n_0n_1}{N} + \frac{1}{2}$$

$$\sigma^2_{c_0} = \frac{2n_0n_1(2n_0n_1 - N)}{N^2(N-1)}$$

A partir de c_0,la esperanza matemática y la desviación se obtiene el estadístico:

$$Z_0 = \frac{c_0 - \mu_{c_0}}{\sigma_{c_0}}$$

Si n_0 y n_2 son lo suficientemente grande la variable c_0 tiende a una distribución normal.

Dado un nivel de confianza $1 - \alpha$ donde α es el nivel de significancia, si Z_0 cumple que $Z_{\frac{-\alpha}{2}} < Z_0 < Z_{\frac{\alpha}{2}}$ entonces la secuencia r_i no se puede rechazar porque sus elementos son independientes con un nivel de confianza $1 - \alpha$. En caso contrario se dice que los elementos de r_i no son independientes.

5. Evaluación de PRNGs

A continuacion se mostraran los histogramas de 2 muestras c/u de los diferentes PRNGs respecto a 20 intervalos de igual tamaño en $[0, 1]$. Posteriormente se presentaran los resultados de dichas muestras de los PNRGs frente a los test de uniformidad e independencia.

Las muestras están referidas como muestras 'a' y 'b'. Las muestras 'a' y 'b' proceden de diferentes parámetros, y además tienen una importante diferencia respecto a su tamaño.

5.1. Histogramas de PNRGs

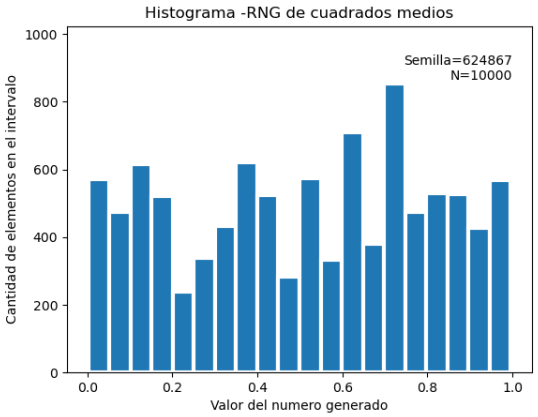

Figura 1: PRNG de cuadrados medios(a)

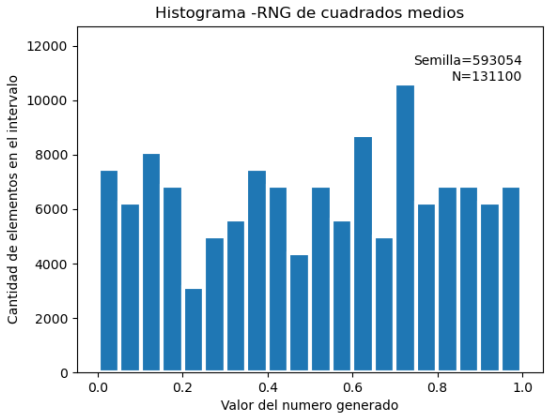

Figura 2: PRNG de cuadrados medios(b)

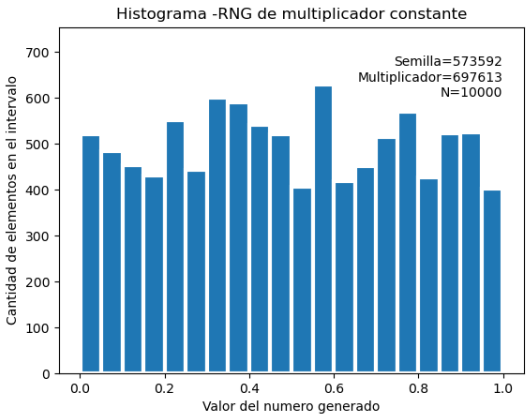

Figura 3: PRNG de multiplicador constante(a)

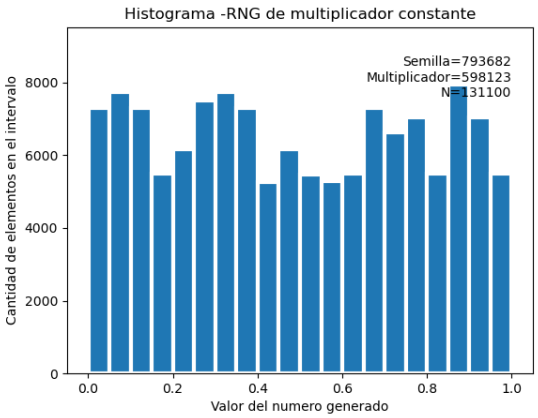

Figura 4: PRNG de multiplicador constante(b)

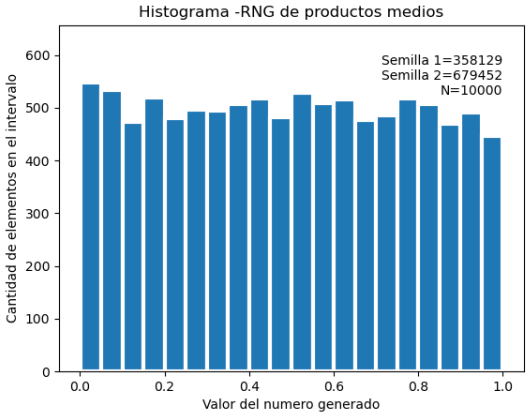

Figura 5: PRNG de productos medios(a)

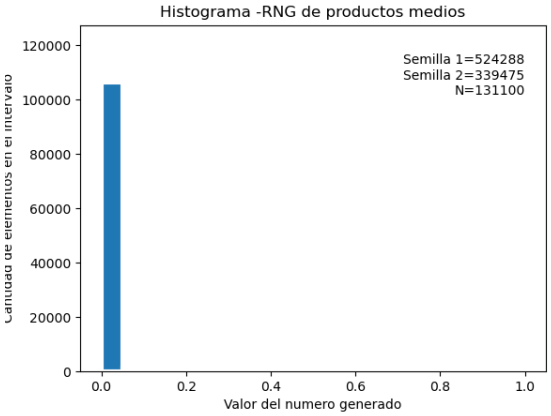

Figura 6: PRNG de productos medios(b)

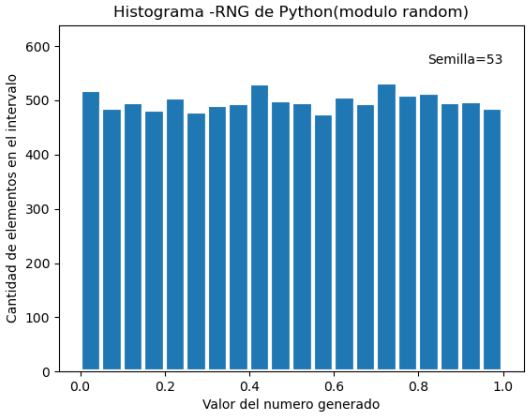

Figura 7: PRNG de Python(modulo random)(a)

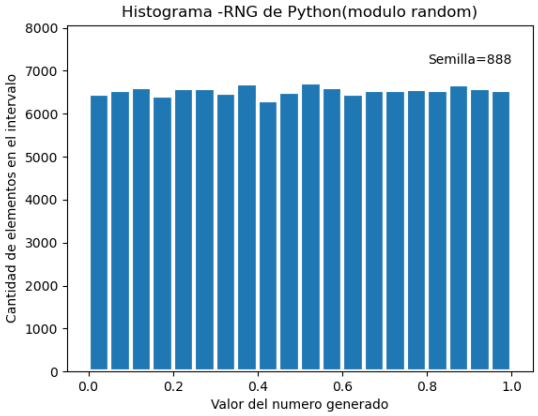

Figura 8: PRNG de Python(modulo random)(b)

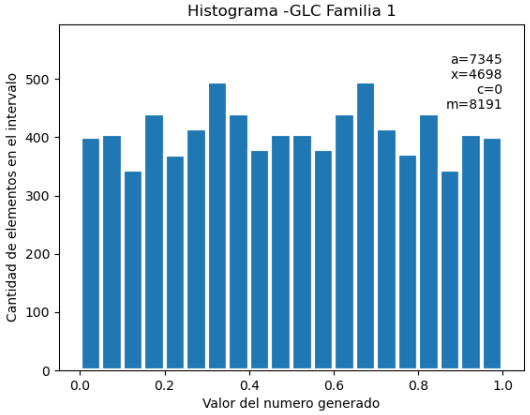

Figura 9: Histograma de GLC1 (a)

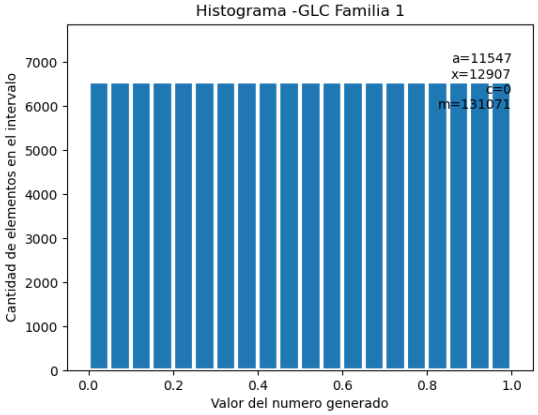

Figura 10: Histograma de GLC1 (b)

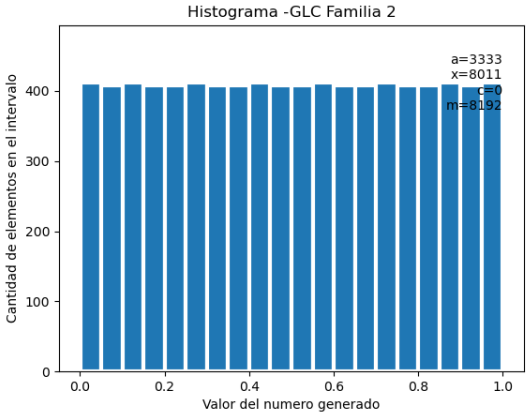

Figura 11: Histograma de GLC2 (a)

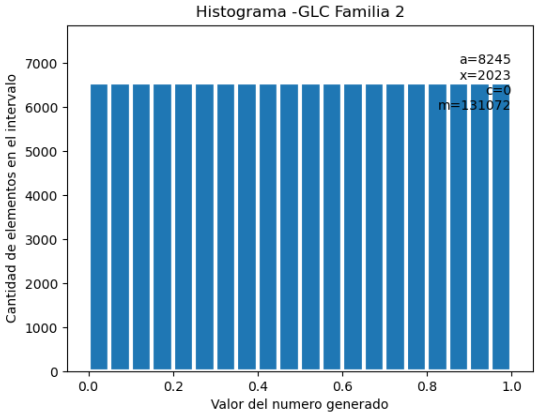

Figura 12: Histograma de GLC2 (b)

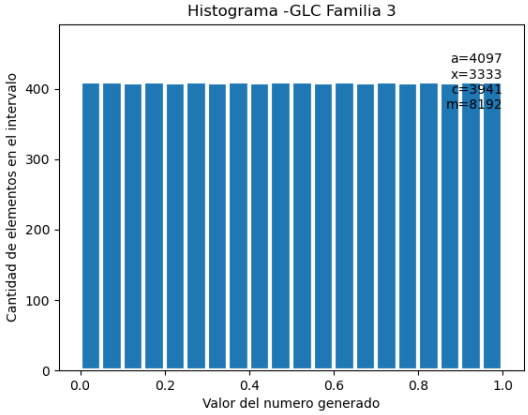

Figura 13: Histograma de GLC3 (a)

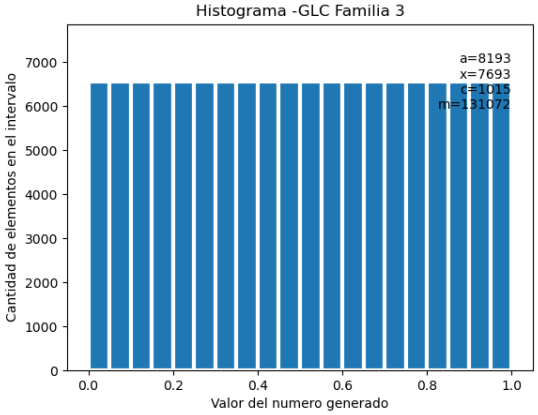

Figura 14: Histograma de GLC3 (b)

5.2. Comparativa de resultados de los tests

En la siguiente tabla se muestran los resultados de evaluar algunas muestras en los test de uniformidad e independencia anteriormente mencionados.

Aclaración: En la seccion parametros ‘s’ refiere seed o semilla, y en los casos de los 3 tipos de GLC la semilla esta señalada como ‘x’

Observaciones: todos los test se realizaron con los siguientes parámetros:

1. Test X^2:
   - Intervalo de confianza= 0.975
   - Grados de libertad= 9 (casos a)
   - Grados de libertad= 19 (casos b)
2. Test KS:
   - Nivel de significancia= 0.025
3. Test de rachas:
   - Nivel de significancia= 0.025
4. Test de autocorrelacion:
   - Retraso=1

PRNG	Parametros	Test X^2	Test KS	Test rachas	Autocorrelación
Multiplicador constante (a)	s=573592 mult=697613 N=10000	No pasa	No pasa	Pasa	0.025184685058
Multiplicador constante (b)	s=793682 mult=598123 N=131100	No pasa	No pasa	No pasa	-0.05538607561
Cuadrados medios (a)	s=624867 N=10000	No pasa	No pasa	No pasa	0.1859465587748
Cuadrados medios (b)	s=593054 N=131100	No pasa	No pasa	No pasa	0.1852997614247
Productos medios (a)	s1=358129 s2=679452 N=10000	Pasa	Pasa	Pasa	0.0161748758501
Productos medios (b)	s1=524288 s2=339475 N=131100	No pasa	No pasa	No pasa	0.7080382880432
random (Python) (a)	s=53	Pasa	Pasa	Pasa	0.000538304188
random (Python) (b)	s=888	Pasa	Pasa	No pasa	-0.003527638049
GLC Familia 1 (a)	a=7345 x=4698 c=0 m=8191	No pasa	Pasa	No pasa	0.022228952347342745
GLC Familia 1 (b)	a=11547 x=12907 c=0 m=131071	Pasa	Pasa	Pasa	$9.283335778856089 \text{x} 10^{-5}$
GLC Familia 2 (a)	a=3333 x=8011 c=0 m=8192	Pasa	Pasa	Pasa	0.003480284601514249
GLC Familia 2 (b)	a=8245 x=2023 c=0 m=131072	Pasa	Pasa	Pasa	$9.35123503001114 \text{x} 10-5$
GLC Familia 3 (a)	a=4097 x=3333 c=3941 m=8192	Pasa	Pasa	Pasa	0.19573623322411138
GLC Familia 3 (b)	a=8193 x=7693 c=1015 m=131072	Pasa	Pasa	Pasa	0.001557668961030695

Cuadro 1: Resultados de tests

Observaciones: El PRNG de productos medios(b) ha degenerado en una constante

Dado que se ha mencionado que para el caso de un GLC en que c=0 y m es primo, existe un algoritmo alternativo, se proporcionara una tabla comparativa de rendimiento entre el algoritmo común y el alternativo(así como de los demás métodos de generación). Dicho método no fue introducido en la tabla anterior debido a que sus resultados son exactamente los mismos que los de 'GLC Familia 1' (producen exactamente el mismo conjunto r_i).

PRNG	Parametros	Cant. num. generados(n)	Tiempo de generacion(t)	t/n (aprox.)
Multiplicador constante (a)	s=573592 mult=697613 N=10000	10000	0.015507619998970767	$1.5507619998970767x10^{-6}$
Cuadrados medios (a)	s=624867 N=10000	10000	0.017844310001237318	$1.7844310001237318x10^{-6}$
Productos medios (a)	s1=358129 s2=679452 N=10000	10000	0.01660365499992622	$1.6603654999926222x10^{-6}$
random (Python) (a)	s=53	10000	0.0014794899998378241	$1.479489999837824x10^{-7}$
GLC Familia 1 (a)	a=7345 x=4698 c=0 m=8191	8191	0.009395071000199096	$.1469992675130137x10^{-6}$
GLC Familia 1(Schrage) (a)	a=7345 x=4698 m=8191	8191	0.013180728999941493	$1.6091721401466846x10^{-6}$
GLC Familia 2 (a)	a=3333 x=8011 c=0 m=8192	8192	0.009520415999986653	$1.1621601562483708x10^{-6}$
GLC Familia 3 (a)	a=4097 x=3333 c=3941 m=8192	8192	0.0095073909999428	$1.160570190422705x10^{-6}$
Multiplicador constante (b)	s=793682 mult=598123 N=131100	131100	0.20116594199953397	$1.5344465446188708x10^{-6}$
Cuadrados medios (b)	s=593054 N=131100	131100	0.22611965899977804	$1.7247876353911369x10^{-6}$
Productos medios(b)	s1=524288 s2=339475 N=131100	131100	0.22269577699989895	$1.6986710678863384x10^{-6}$
random (Python) (b)	s=888	131100	0.017788483000003907	$1.3568636918385893x10^{-7}$
GLC Familia 1 (b)	a=11547 x=12907 c=0 m=131071	131071	0.14620087900038925	$1.1154326967856296x10^{-6}$
GLC Familia 1(Schrage) (b)	a=11547 x=12907 m=131071	131071	0.202036157999828	$1.5414253191005485x10^{-6}$
GLC Familia 2 (b)	a=8245 x=2023 c=0 m=131072	131072	0.14199572100005753	$1.0833413772587397x10^{-6}$
GLC Familia 3 (b)	a=8193 x=7693 c=1015 m=131072	131072	0.14172780500030058	$1.0812973403953596x10^{-6}$

Cuadro 2: Comparativa de rendimiento

5.3. Observaciones de los resultados

A partir de los resultados obtenidos al evaluar los distintos PRNGs del presente informe (y teniendo en cuenta los resultados de otras muestras, ya que 2 muestras no son suficientes ni representativas). Se observa que los GLC arrojan mejores resultados frente a los test realizados, que los generadores no congruenciales(productos medios, cuadrados medios, algoritmo de multiplicador constante). Aunque los GLC enfrentan el problema de repetir ciclos en momentos determinados por sus parámetros, no tienen el problema de degenerar en cero como los algoritmos no congruenciales.

Respecto a la diferencia entre los GLC, es notable la mejoría de las familias 2 y 3 por sobre los de familia 1, especialmente cuando m es potencia de 2 en las familias 2 y 3. El problema de la familia 2 es que aunque aparentemente es la mas aleatoria, es cíclica, de hecho evaluando autocorrelación con un salto de m/4 puede encontrarse una fuerte autocorrelación.

En cuanto a los GLC de la familia 1, con las tecnologías actuales puede resultar mas óptimo utilizar el algoritmo clásico que el de Schrage. Tal vez sucede lo contrario cuando los valores de los parámetros son muy grandes.

Asi mismo el PRNG de python genera numeros con mucha menor correlacion que todos los algoritmos aunque no garantiza mayor uniformidad.

6. Conclusión

Como conclusión final nos inclinamos por decir que los algoritmos de GLC generan números con mayor 'aleatoriedad' eligiendo parámetros pertenecientes a las familias 2 y 3 frente a la familia 1 y algoritmos no congruenciales (por lo menos los mencionados en este informe). Mientras los GLC de familia 2 tienen mayor aleatoriedad, los de familia 3 permiten generar mas números (sin generar ciclos) a partir de un conjunto de parámetros.

Pero si hay disponibles generadores mas refinados, como el que proporciona Python mediante su modulo random se recomienda utilizar estos. Aunque estos puedan ser menos uniformes, son mucho mas 'aleatorios'.

7. Referencias

Referencias

[1] Números pseudoaleatorios - Github- María Teresa Ortiz

https://tereom.github.io/est-computacional-2018/numeros-pseudoaleatorios.html

[2] SIMULACION Y ANALISIS DE SISTEMAS CON PROMODEL- Eduardo García Dunna, Heriberto

https://www.google.com.ar/books/edition/SIMULACION_Y_ANALISIS_DE_SISTEMAS_CON_PR/VuEfwtFr1QMC?hl=es-419&gbpv=1&pg=PR1&printsec=frontcover

[3] Generadores congruenciales lineales- Wikipedia

https://en.wikipedia.org/wiki/Linear_congruential_generator

[4] Multiplicacion modular de Montgomery

https://en.wikipedia.org/wiki/Montgomery_modular_multiplication

[5] Metodo de Schrage

https://www.cse.wustl.edu/~jain/iucee/ftp/k_26rng.pdf#page=19

[6] Statistical Analysis - Random.org

https://www.random.org/analysis/

[7] Testing Random Number Generators - Dan Biebighauser

[8] Hipótesis Nula - Wikipedia

https://es.wikipedia.org/wiki/Hip%C3%B3tesis_nula

[9] Kolmogorov-Smirnov Goodness of Fit Test - NIST

https://www.itl.nist.gov/div898/handbook/eda/section3/eda35g.htm

[10] Autocorrelation - NIST

https://www.itl.nist.gov/div898/handbook/eda/section3/eda35c.htm

[11] Prueba de Independencia Corridas de Arriba y Abajo de la Media- Universidad Nacional Pedro Ruiz Gallo

https://www.studocu.com/pe/document/universidad-nacional-pedro-ruiz-gallo/investigacion-de-operaciones-ii/prueba-de-independencia-corridas-arriba-y-abajo-de-la-media/5373960

8. Apéndice

8.1. Repositorio GitHub

El código en Python descrito en este proyecto puede ser consultado en el siguiente URL: https://github.com/Faus14/Simulacion_2023/tree/main/TP-2.1-Generadores.