TP 2.1 - GENERADORES PSEUDOALEATORIOS

Mateo, Lara 44795 - Retrivi, Vittorio 44727

Mayo 16, 2020

Resumen

Investigación sobre tipos de generadores de números aleatorios, generación de números pseudoaleatorios y cómo probar su aleatoriedad.

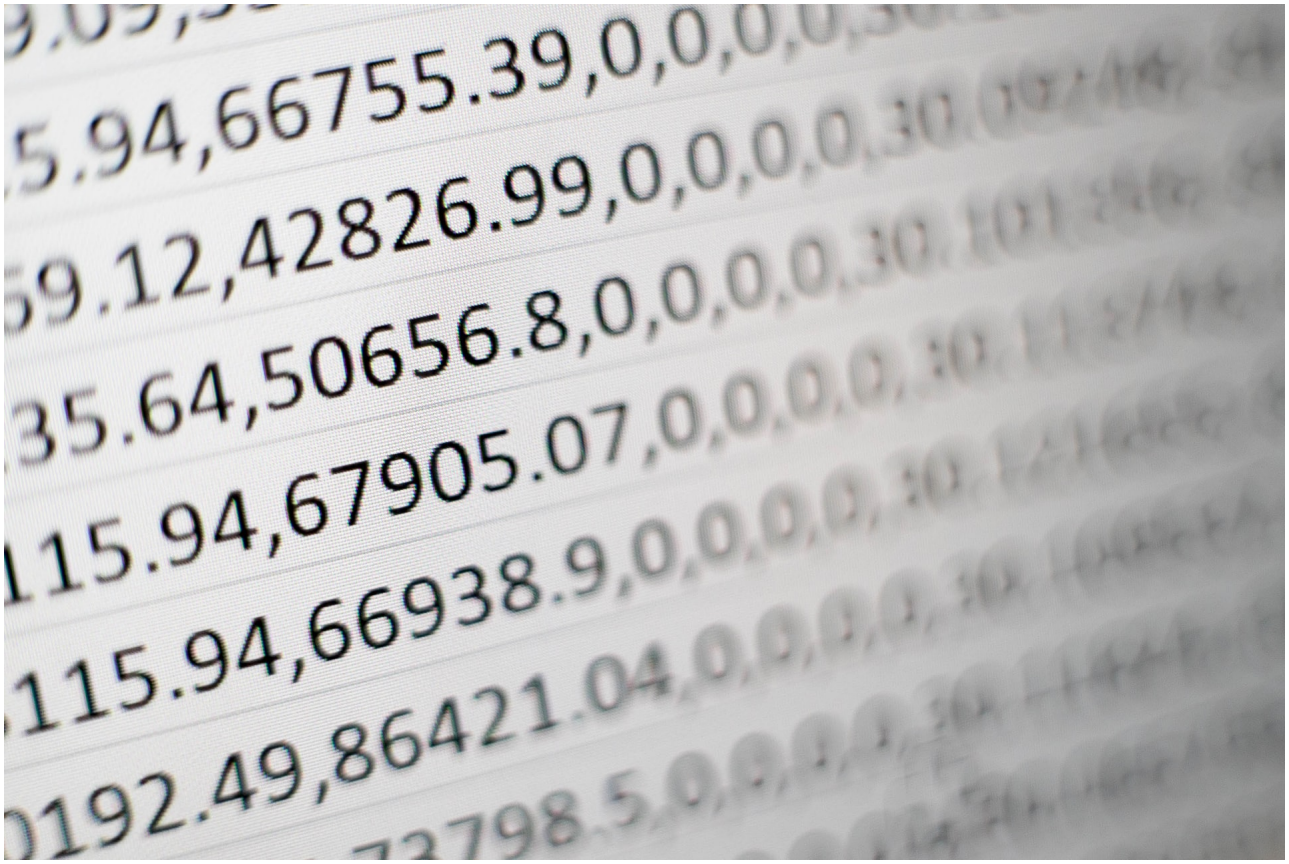
Keywords: Simulación · Python · Números Pseudoaleatorio · GCL · Método de los cuadrados medios · Generador de números mediante posts de Reddit · Test de Paridad · Test de Poker · Test de Kolmogorov-Smirnov · Test de Chi-Cuadrado

Introducción

¿Existe la aleatoriedad?. Una metáfora típica es tirar un dado el cual producirá un resultado al azar. Pero los dados no son más que simples cubos construidos de ciertos materiales. Y las reglas según las cuales caen y rebotan sobre una mesa son puramente deterministas.

Lo que sucede en realidad es que los dados y su entorno conforman un sistema caótico en el cual no podemos determinar con total precisión su estado inicial calculando así que va a suceder; e incluso conociéndolo, el más pequeño error al hacer la medición sería amplificado enormemente cuando empiezan a rebotar en la mesa.

La cuestión filosófica de fondo permanece: tal vez sólo exista una forma en que puede caer un dado cuando se lanza de una forma determinada, pero si ni vos, ni el dado, ni el universo lo «saben», parecerían razones suficientes para decir que el resultado depende realmente del puro azar.



Análisis

Métodos para la generación de números aleatorios

Cuando hablamos de números generados aleatorios debemos tener en cuenta sus orígenes y sus usos.
Existen dos tipos: **Aleatorios (TRNG) y Pseudoaleatorios (PRNG)**.

Los TRNG extraen la aleatoriedad de los fenómenos físicos y la introducen en una computadora. Por su parte, 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 pseudo-aleatorios no muestran ningún patrón o regularidad aparente desde un punto de vista estadístico a pesar de haber sido generadas por un algoritmo completamente determinista.

La diferencia básica entre PRNG y TRNG es fácil de entender si compara números aleatorios generados por computadora con tiradas de un dado. Debido a que los PRNG generan números aleatorios mediante el uso de fórmulas matemáticas o listas precalculadas, el uso de uno corresponde a alguien que tira un dado muchas veces y anota los resultados. Cada vez que pides una tirada de dados, obtienes el siguiente en la lista. Efectivamente, los números parecen aleatorios, pero están realmente predeterminados. Los TRNG funcionan haciendo que una computadora realmente tire el dado o, más comúnmente, usen algún otro fenómeno físico que sea más fácil de conectar a una computadora que un dado.

Los PRNG son eficientes, lo que significa que pueden producir muchos números en poco tiempo, y deterministas, es decir, que una secuencia de números dada puede reproducirse en una fecha posterior si se conoce el punto de partida de la secuencia.
La eficiencia es una buena característica si su aplicación necesita muchos números, y el determinismo es útil si necesita volver a reproducir la misma secuencia de números en una etapa posterior. Los PRNG también suelen ser periódicos, esto significa que la secuencia eventualmente se repetirá. Si bien la periodicidad casi nunca es una característica deseable, los PRNG modernos tienen un período que es tan largo que puede ignorarse para la mayoría de los propósitos prácticos.

Estas características hacen que los PRNG sean adecuados para aplicaciones donde se requieren muchos números y donde es útil que la misma secuencia se pueda reproducir fácilmente. Ejemplos populares de tales aplicaciones son las aplicaciones de simulación y modelado. Los PRNG no son adecuados para aplicaciones en las que es importante que los números sean realmente impredecibles, como el cifrado de datos y los juegos de azar.

A continuación, describimos los algoritmos de generación utilizados en el trabajo:

Generadores de congruencia lineal (GCL)

Un generador de congruencia lineal (GCL) es un algoritmo que permite obtener una secuencia de números pseudoaleatorios calculados a partir de una función lineal definida a trozos discontinua. Es uno de los métodos más antiguos y conocidos para la generación de números pseudoaleatorios.

El generador se define a partir de la siguiente relación de recurrencia:

$X_{n+1} = (aX_n + c) \text{ mód } m$

donde X es la secuencia de valores pseudoaleatorios, y

m, $0 < m$
es el módulo,

a, $0 < a < m$
es el multiplicador,

c, $0 \leq c < m$
es el incremento y

X_0, $0 \leq X_0 < m$
es la semilla.

Método de los cuadrados medios

Este método fue inventado por John von Neumann y descrito por primera vez en una conferencia en 1949.

El método se realiza de la siguiente forma:

Se inicia con una semilla de X dígitos par:

$semilla = 9731$

Esta semilla se eleva al cuadrado, produciendo un número de 2X dígitos o más (si el resultado tiene menos de 2X dígitos se añaden ceros al inicio).

$value = 94692361$

Los X números centrales serán el siguiente número en la secuencia, y se devuelven como resultado.

$seed = 6923$

La problemática de este generador es que converge rápidamente en ciclos cortos, por ejemplo, si aparece un cero se propagará por siempre.

Generador propio de Reddit

Inspirándonos en los generadores vistos decidimos probar hacer nuestro propio generador.
Primero pensamos en hacer una modificación al del GCL relacionándolo con raíces de un binomio, pero era más complicado de lo que pensábamos y optamos por pensar en uno más divertido.
Nuestro primer pensamiento fue hacer un generador basándose en imágenes que se están publicando en el momento, pero el conflicto era obtener imágenes sin buscar nada en específico.
Entonces se nos ocurrió basarlo en tweets de Twitter, la dificultad se encontraba esta vez en el código de marcado html de la página, el cual no era tan simple de analizar, optamos por buscar páginas que tuvieran un formato más fácil de des-estructurar, observamos que la web de Reddit tenía un marcado más simple y nos pusimos en marcha. A continuación se explica su funcionamiento en detalle.

Al solicitar un nuevo número aleatorio, el generador:

1. Capta el título del post más reciente de `https://www.reddit.com/new/`
2. El título es convertido a binario
3. Los bits son pasados por 4 compuertas X-OR
4. Se decodifica la tira de bits utilizando BCD para convertirlo a base decimal
5. Se toman los primeros 4 decimales como resultado final

Librería Random de Python

Se utilizó la librería Random de Python como un algoritmo lo bastante fiable como para ser usado a modo de comparación entre los demás desarrollados. En particular, esta librería utiliza el algoritmo Mersenne Twister como núcleo de la implementación.

Pruebas empleadas

Para verificar la aleatoriedad de los generadores hay que ponerlos a prueba
Las pruebas que vamos a realizar sobre nuestras generaciones de números aleatorios son:

- $X^2(Chi - Cuadrado)$
- Paridad
- Poker
- Kolmogorov-Smirnov

Test de Chi-Cuadrado

Una prueba de chi-cuadrado, es una prueba de hipótesis estadística que es válida para realizar cuando la estadística de prueba es chi-cuadrado distribuida bajo la hipótesis nula, específicamente la prueba de chi-cuadrado de Pearson y sus variantes. La prueba de chi cuadrado de Pearson se usa para determinar si hay una diferencia estadísticamente significativa entre las frecuencias esperadas y las frecuencias observadas en una o más categorías de una tabla de contingencia.
En las aplicaciones estándar de esta prueba, las observaciones se clasifican en clases mutuamente excluyentes.

Si la hipótesis nula es verdadera, el estadístico de prueba calculado a partir de las observaciones sigue una distribución de frecuencia X^2.

El propósito de la prueba es evaluar la probabilidad de que las frecuencias observadas supongan que la hipótesis nula es verdadera.
Las estadísticas de prueba que siguen una distribución de Chi-Cuadrado ocurren cuando las observaciones son independientes y normalmente distribuidas, lo que a menudo se justifica bajo el teorema del límite central. También hay pruebas de Chi-Cuadrado para probar la hipótesis nula de independencia de un par de variables aleatorias basadas en observaciones de los pares.

Test de paridad

El test de paridad se explica de forma simple verificando que la proporción de números pares e impares generados sea la misma.

Test de Poker

La prueba de póker se basa en la frecuencia en que ciertos dígitos se repiten en cada número de una serie.

Test de kolmogorov-Smirnov

En estadística, la prueba de Kolmogórov-Smirnov (también prueba K-S) es una prueba no paramétrica que determina la bondad de ajuste de dos distribuciones de probabilidad entre sí.
En el caso de que queramos verificar la normalidad de una distribución, la prueba de Lilliefors conlleva algunas mejoras con respecto a la de Kolmogórov-Smirnov; y, en general, el test de Shapiro–Wilk o la prueba de Anderson-Darling son alternativas más potentes.
Conviene tener en cuenta que la prueba Kolmogórov-Smirnov es más sensible a los valores cercanos a la mediana que a los extremos de la distribución. La prueba de Anderson-Darling proporciona igual sensibilidad con valores extremos.

La función de distribución empírica Fn para n observaciones ordenadas independientes e idénticamente distribuidas se define como:

$$F_n(x) = \frac{1}{n}\sum_{i=1}^{n} \begin{cases} 1 & \text{si } y_i \leq x, \\ 0 & \text{alternativa.} \end{cases}$$

Para dos colas el estadístico viene dado por

$$D_n^+ = \text{máx}(F_n(x) - F(x))$$

$$D_n^- = \text{máx}(F(x) - F_n(x))$$

Valores y gráficas obtenidas

Parámetros	Generador	Tipo de generador	Prueba χ^2	Prueba de Kolmogórov-Smirnov	Prueba de póker	Prueba de paridad
Semilla: 3183856186 Módulo: 2^{32} = 4294967296 multiplicador: 65539 Incremento: 0 Longitud de la serie: 200000 números	***LCG (RANDU)***	*Pseudoaleatorio*	**Aprobada**	**Aprobada**	**Rechazada**	**Aprobada**
Semilla: 973160574 Módulo: 2^{32} = 4294967296 multiplicador: 65539 Incremento: 0 Longitud de la serie: 200000 números	***LCG (RANDU)***	*Pseudoaleatorio*	**Aprobada**	**Aprobada**	**Aprobada**	**Aprobada**
Semilla: 3183856186 Módulo: 2^{32} = 4294967296 multiplicador: 134775813 Incremento: 1 Longitud de la serie: 200000 números	***LCG (Turbo Pascal)***	*Pseudoaleatorio*	**Rechazada**	**Aprobada**	**Aprobada**	**Aprobada**
Semilla: 973160574 Módulo: 2^{32} = 4294967296 multiplicador: 134775813 Incremento: 1 Longitud de la serie: 200000 números	***LCG (Turbo Pascal)***	*Pseudoaleatorio*	**Aprobada**	**Aprobada**	**Aprobada**	**Aprobada**
Semilla: 7185296402312374 Módulo: 2^{64} = 18446744073709551616 Multiplicador: 6364136223846793005 Incremento: 1442695040888963407 Longitud de la serie: 200.000 números	***LCG (MMIX By Donald Knuth)***	*Pseudoaleatorio*	**Rechazada**	**Aprobada**	**Aprobada**	**Aprobada**
Semilla: 875302 Longitud de la serie: 200000 números	***Middle Square Generator (semilla pequeña)***	*Pseudoaleatorio*	**Rechazada**	**Rechazada**	**Rechazada**	**Rechazada**
Semilla: 876034075289102346 Longitud de la serie: 200000 números	***Middle Square Generator (semilla extensa)***	*Pseudoaleatorio*	**Rechazada**	**Rechazada**	**Aprobada**	**Aprobada**
Longitud de la serie: 200000 números	***Python Random***	*Pseudoaleatorio*	**Aprobada**	**Aprobada**	**Aprobada**	**Aprobada**
Longitud de la serie: 17747 números	***'From Reddit' Generator***	*Aleatorio*	**Aprobada**	**Aprobada**	**Aprobada**	**Aprobada**

Figura 1: Resultados de cada uno de los test aplicados a los generadores desarrollados

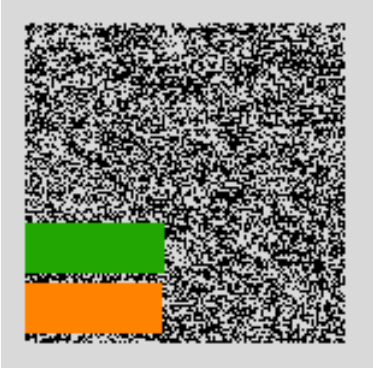


Figura 2: Representación gráfica de nuestro generador

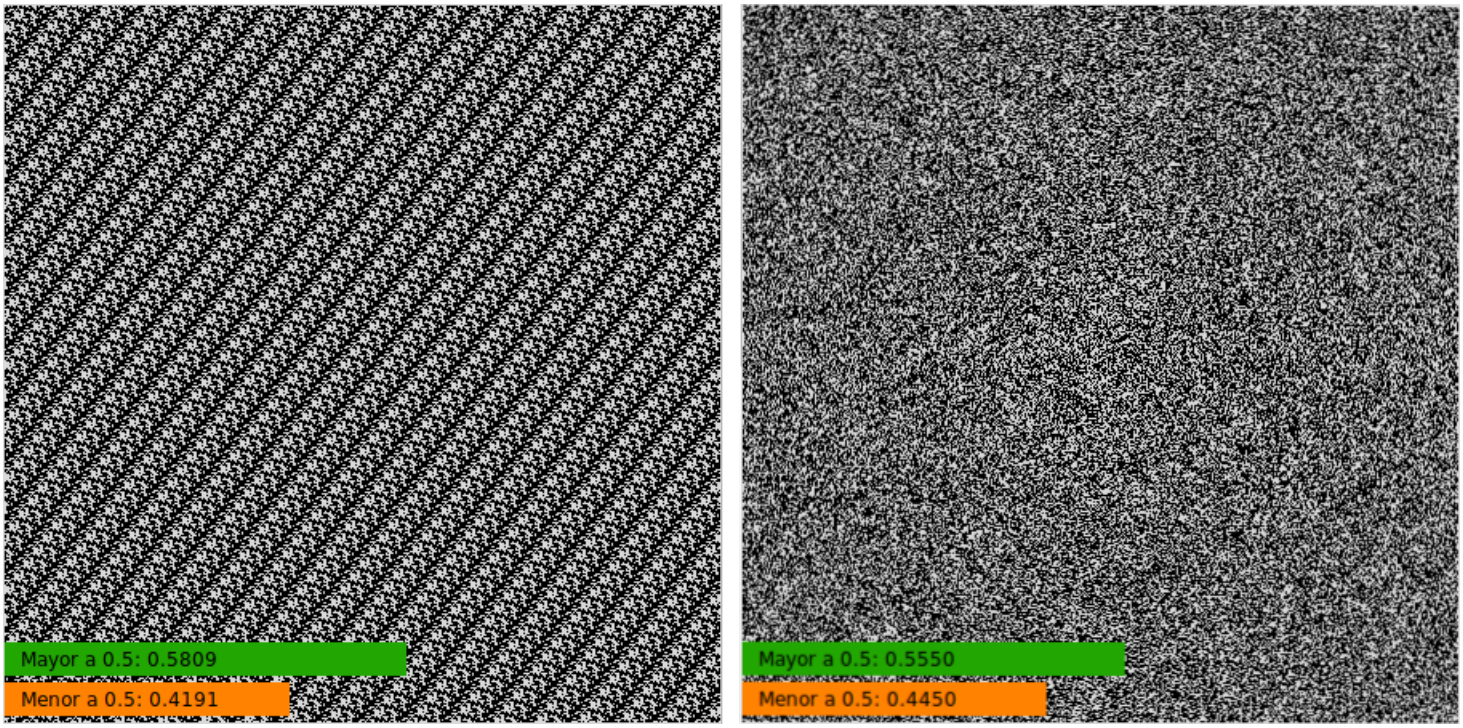


Figura 3: Método de los cuadrados medios con semilla pequeña (izquierda) y semilla extensa (derecha)

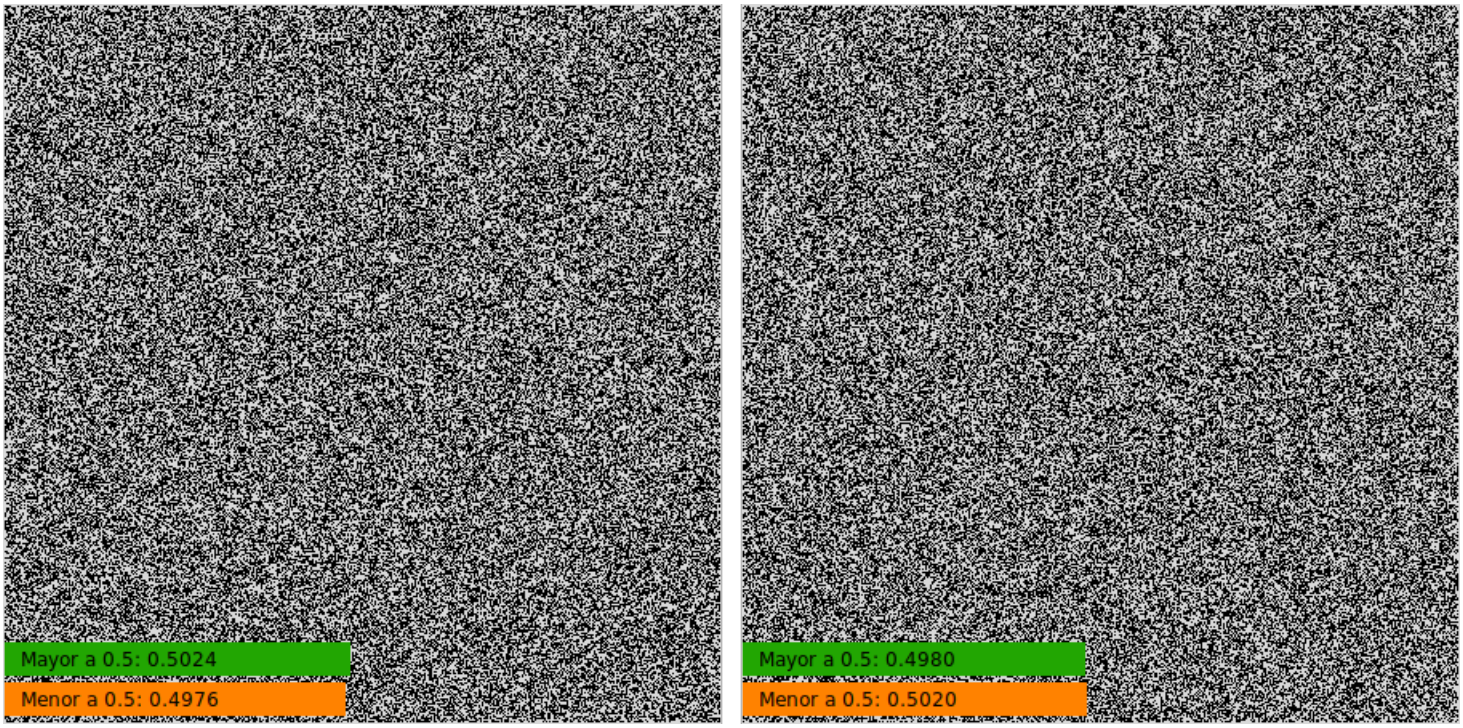


Figura 4: LCG (RANDU) [Seed = 973160574] (izquierda) y Biblioteca Random de Python (derecha)

Discusión

Con respecto al Código empleado

Se decidió separar a cada elemento del código en diferentes módulos. Esto nos ayudó a la hora de la división de las tareas, posibilitando que cada integrante pueda enfocarse en un aspecto concreto del código sin interferir en el trabajo del otro, reduciendo así la cantidad de errores y aumentando la eficiencia al programar.

Así, el código se divide en:

- Generadores: módulo encargado de almacenar el código de cada uno de los generadores desarrollados.
- Generador Propio: dado que el generador inventado propiamente por nosotros requería de una programación más extensa y el uso de otras funciones decidimos separarlo en un módulo independiente.
- Tests: se encarga de almacenar todos los tests como así también las utilidades para graficar los mismos.
- Menú: interfaz que permite interactuar con los distintos generadores.
- Utils: módulo de utilidades varias para el buen funcionamiento del código.

Con respecto a los métodos utilizados

Procedemos a observar en forma detallada los resultados obtenidos.

Generadores de congruencia lineal (GCL)

Observando la imagen izquierda de la **figura 4** notamos que no existen patrones reconocibles a simple vista en la generación y que el resultado se asemeja en gran forma a la imagen derecha (serie generada a partir de la librería Random de Python).

Si bien este algoritmo provee series de números con generaciones muy uniformes, observando la **tabla 1** notamos que es muy importante la selección de una semilla adecuada para cada conjunto de parámetros (módulo, multiplicador, incremento); de lo contrario, el generador podría llegar a fallar en alguno de los tests.

Método de los cuadrados medios

Analizando la **tabla 1** y comparándola con la **figura 3** podemos notar de forma simple que utilizando dicho generador con una semilla pequeña provocará que el generador converja de forma apresurada hacia un ciclo corto, provocando que el mismo sea rechazado en cada uno de los test. De forma gráfica, esto se expresa en un patrón claramente reconocible en la imagen izquierda de la **figura 3**

En la imagen de la derecha (con números generados a partir de una semilla más extensa) si bien no podemos percibir un patrón a simple vista, las barras de porcentaje nos muestran que el generador produce un porcentaje notablemente mayor de números por encima del 0.5, esto se ve reflejado en la tabla con el generador siendo rechazado en 2 de los 4 tests.

Estos resultados eran claramente esperables ya que es bien conocido que este método, aunque bueno a nivel educativo, no provee resultados fiables en la práctica.

Generador propio

Pese al escepticismo en torno a la efectividad de este generador, el mismo logró superar todas las pruebas.
Como mencionamos anteriormente, los números generados se producen a partir de frases escritas por usuarios de internet, estas frases equivalen a la semilla de nuestro algoritmo.
Si bien originalmente los bits de esta semilla no son realmente aleatorios, ya que se repiten patrones en el lenguaje (mayoritariamente inglés), con las transformaciones que fueron aplicadas para evitar estos patrones probablemente redujimos los mismos.

Conclusiones

La investigación realizada fue útil para aprender y reforzar lo teórico de la aleatoriedad y como se relaciona con la estadística. Una parte importante del aprendizaje es la experimentación y la curiosidad, incentivar esto aumentará el conocimiento y lo hará más aplicable a la realidad.

A continuación realizamos una reflexión sobre nuestro generador:

La discusión que efectuamos al ver los resultados del generador fue sobre si realmente son números aleatorios o son pseudoaleatorios.
Respecto a esto concluimos con la siguiente postura, creemos que en un origen son aleatorios ya que no podemos predecir fácilmente que va a ser la cadena de caracteres que recibimos, aunque debido a la naturaleza humana podríamos quizás aproximar las predicciones con el uso de la psicología y la estadística.

Supongamos que son verdaderamente aleatorios, **¿Podrían dejar de serlo?**

En nuestra opinión sí, ya que si se usara nuestro generador para, por ejemplo, la lotería, es probable que si se diera a conocer el funcionamiento del generador se burlara la aleatoriedad gracias a individuos que escribieran posts intencionalmente para afectar a los números generados.
Además, debido a que el método precisa realizar sucesivas peticiones http para obtener datos, no es para nada eficiente.
Particularmente, nos llevó alrededor de 24 horas obtener los 17747 números utilizados para realizar los tests. Lo que hace que el método quede en algo anecdótico y sea ineficaz llevarlo a la práctica.
De igual manera, si bien lo producido no es un producto final para un mercado, si son escalones propios que mejorarán la calidad de futuros resultados.

Referencias

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

[2] https://es.wikipedia.org/wiki/Generador_lineal_congruencial

[3] https://www.random.org/randomness/

[4] https://en.wikipedia.org/wiki/Randomness_tests

[5] https://es.wikipedia.org/wiki/Prueba_de_Kolmogorov-Smirnov

Índice de figuras