Trabajo Práctico 2.1 – Simulación

Renzo Aimaretti
renzoceronueve@gmail.com

Facundo Sosa Bianciotto
facundososabianciotto@gmail.com

Vittorio Maragliano
maraglianovittorio@gmail.com

Ignacio Amelio Ortiz
nameliortiz@gmail.com

Nicolás Roberto Escobar
escobar.nicolas.isifrro@gmail.com

Juan Manuel De Elia
juanmadeelia@gmail.com

14 de mayo de 2025

Resumen

Este trabajo practico consiste en la simulacion de generadores de numeros pseudoaleatorios, en particular el GCL, el generador de cuadrados medios y el generador de Python. Se implementaron los algoritmos en Python y se compararon los resultados obtenidos realizando 4 tests enfocados en distintos aspectos de calidad del generador. `random`. Se realizaron graficos de histograma y de dispersión para analizar la calidad de los generadores.

1. Introducción

Se desarollaron los 3 simuladores mencionados en python con el propósito de analizar la calidad de los números generados. Se realizaron 4 tests para evaluar la calidad de los números generados. Los tests realizados son:

- Test de Kolmogorov-Smirnov
- Test de Chi-Cuadrado
- Test de Media y Varianza
- Test de Autocorrelación

El numero de corridas y las semillas (seeds) pueden ingresarse como argumentos al script. Si no se ingresan, se utilizan los valores por defecto.

2. Generadores

2.1. GCL

El generador de congruencia lineal (GCL) es un generador de números pseudoaleatorios que utiliza la siguiente fórmula:

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

Donde:

- X_n es el número pseudoaleatorio generado en la iteración n.
- a es el multiplicador.
- c es la constante aditiva.

- m es el módulo.
- X_{n+1} es el siguiente número pseudoaleatorio generado.

Uno de los aspectos críticos de un PRNG es su período, es decir, la longitud máxima de la secuencia antes de que empiece a repetirse. En el caso del generador congruencial lineal (GCL), su período máximo es igual al módulo m, siempre y cuando se cumplan ciertas condiciones sobre los parámetros a, c y la semilla inicial X_0. Para alcanzar el período máximo, la semilla debe recorrer todo el espacio de posibles valores sin colisiones.

Estas condiciones, conocidas como los **criterios de Hull-Dobell**, son las siguientes:

- c y m deben ser coprimos, es decir, $\gcd(c, m) = 1$.
- $a - 1$ debe ser divisible por todos los factores primos de m.
- Si m es divisible por 4, entonces $a - 1$ también debe ser divisible por 4.

Dichas condiciones garantizan que el GCL tenga un período completo igual a m, lo cual maximiza la calidad estadística de la secuencia generada. Si alguna de estas condiciones no se cumple, el generador puede entrar en ciclos cortos, lo que compromete la aleatoriedad y representatividad de los datos simulados.

En este trabajo práctico, se utilizarán los siguientes valores:

- $a = 1664525$
- $c = 1013904223$
- $m = 2^{32}$
- $X_0 = seed$

La semilla X_0 determina completamente la secuencia generada. En este trabajo, se justifica el uso de semillas de hasta 10 cifras decimales debido a la necesidad de cubrir todo el espacio de valores posibles para el módulo m.

Dado que el módulo es $m = 2^{32} \approx 4{,}29 \times 10^9$, la cantidad de cifras decimales necesarias para representar cualquier valor en ese espacio se calcula como:

$$\log_{10}(m) = \log_{10}(2^{32}) = 32 \cdot \log_{10}(2) \approx 32 \cdot 0{,}3010 \approx 9{,}63 \tag{2}$$

Por lo tanto, se requieren hasta **10 dígitos decimales** para representar cualquier semilla válida en este espacio. Esto garantiza que la semilla pueda recorrer completamente el dominio de entrada del generador y aprovechar el máximo período posible.

2.2. Cuadrados Medios

El generador de cuadrados medios es un generador de números pseudoaleatorios que utiliza la siguiente fórmula:

$$X_{n+1} = \left(\frac{X_n^2}{10^d} \right) \quad \text{mód } 1 \tag{3}$$

Donde:

- X_n es el número pseudoaleatorio generado en la iteración n.
- d es el número de dígitos de la parte entera de X_n^2.
- X_{n+1} es el siguiente número pseudoaleatorio generado.
- $X_0 = seed$

Para este generador, la semilla X_0 debe ser un número de 4 dígitos. La elección de una semilla de 4 dígitos es importante porque el generador de cuadrados medios utiliza los dígitos centrales del cuadrado del número generado en la iteración anterior. Si la semilla es demasiado pequeña, el generador puede no producir una secuencia suficientemente aleatoria. El generador de cuadrados medios es un generador de números pseudoaleatorios que utiliza la propiedad de que el cuadrado de un número tiene una distribución uniforme. Sin embargo, su calidad puede verse afectada por la elección de los parámetros y la forma en que se extraen los dígitos.

2.3. Python

El generador de Python utiliza el algoritmo Mersenne Twister, que es un generador de números pseudoaleatorios de propósito general. Este algoritmo tiene un período muy largo y una buena calidad de aleatoriedad. El generador de Python se basa en la siguiente fórmula:

$$X_{n+1} = (X_n \cdot A + B) \mod M \tag{4}$$

Donde:

- X_n es el número pseudoaleatorio generado en la iteración n.
- A es una matriz de transformación.
- B es un vector de desplazamiento.
- M es el módulo.
- X_{n+1} es el siguiente número pseudoaleatorio generado.

El generador de Python es uno de los generadores de números pseudoaleatorios más utilizados en la actualidad debido a su calidad y facilidad de uso.

3. Tests

3.1. Test de Kolmogorov-Smirnov

El test de Kolmogorov-Smirnov es una prueba estadística que se utiliza para determinar si una muestra de datos proviene de una distribución específica. En este caso, se utilizará para evaluar si los números generados por los generadores son uniformemente distribuidos en el intervalo [0, 1]. El test compara la función de distribución empírica de la muestra con la función de distribución acumulativa de la distribución uniforme.

3.2. Test de Chi-Cuadrado

El test de Chi-Cuadrado es una prueba estadística que se utiliza para determinar si hay una diferencia significativa entre la distribución observada y la distribución esperada. En este caso, se utilizará para evaluar si los números generados por los generadores son uniformemente distribuidos en el intervalo [0, 1]. El test compara la frecuencia observada de los números generados con la frecuencia esperada de una distribución uniforme.

3.3. Test de Media y Varianza

El test de media y varianza se utiliza para evaluar si la media y la varianza de los números generados por los generadores son iguales a los valores esperados para una distribución uniforme. En este caso, se espera que la media sea 0.5 y la varianza sea 1/12.

3.4. Test de Autocorrelación

El test de autocorrelación se utiliza para evaluar si los números generados por los generadores son independientes entre sí. En este caso, se calculará la autocorrelación de los números generados y se comparará con el valor esperado para una distribución uniforme. Si la autocorrelación es significativamente diferente de cero, se puede concluir que los números generados no son independientes. Sin embargo, para una cantidad alta de corridas podemos ver como el generador de cuadrados medios devuelve un resultado fuera de lo esperado dentro de este test, esto puede ser un indicio de fallas para secuencias largas.

4. Resultados

Se obtuvieron los resultados de los tests para cada generador con 1 millon de corridas. En la siguiente tabla se muestran los resultados de los tests realizados:

Generador	Kolmogorov-Smirnov	Chi-Cuadrado	Media y Varianza	Autocorrelación
GCL	0.0005	7.08	0.5000 y 0.0833	0.0007
Cuadrados Medios	0.2100	1499900.00	0.5100 y 0.0500	0.0000
Python	0.0006	10.86	0.5002 y 0.0833	0.0011

Cuadro 1: Resultados de los tests realizados

4.1. Visualización de Patrones en los Números Generados

Además de los tests estadísticos, se generaron imágenes en formato bitmap donde cada píxel representa un número pseudoaleatorio convertido en un valor de escala de grises. Estas visualizaciones permiten detectar patrones estructurales o repetitivos que pueden comprometer la aleatoriedad.

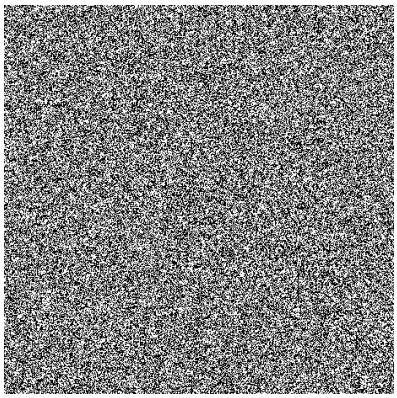

Figura 1: Visualización de números generados con GCL

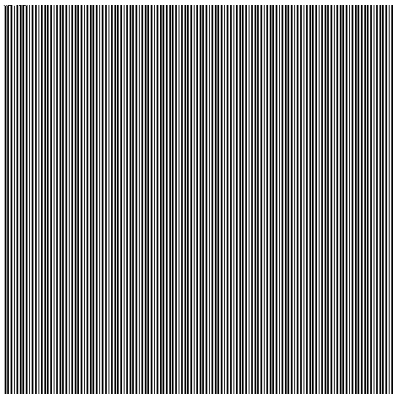

Figura 2: Visualización de números generados con Cuadrados Medios

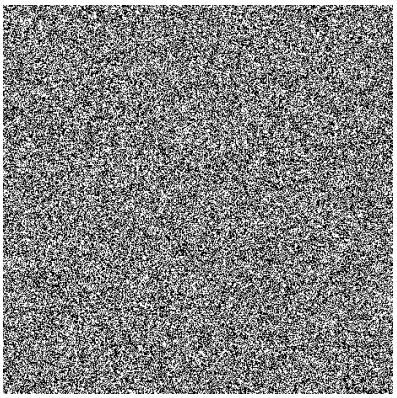

Figura 3: Visualización de números generados con Python

Estas visualizaciones confirman los resultados de los tests estadísticos. La imagen del generador de Python (Fig. 3) muestra un patrón de ruido aleatorio uniformemente distribuido, sin estructuras visibles. El generador GCL (Fig. 1) presenta algunas estructuras sutiles que sugieren cierta predictibilidad, pero mantiene una apariencia mayormente aleatoria. En contraste, el generador de Cuadrados Medios (Fig. 2) exhibe patrones evidentes y repetitivos, lo que revela su pobre calidad como fuente de números pseudoaleatorios. Ademas, esta imagen nos confirma el error por parte del test de autocorrelacion el cual devuelve un valor igual 0 contrastando con el mapa de bits donde se observan patrones claros.

4.2. Relación entre período y número de corridas para el generador GCL

Con el propósito de analizar la influencia del **período** en la calidad de los números generados por un **Generador Congruencial Lineal (GCL)**, se realizaron pruebas controladas variando el número de corridas en relación al período máximo teórico del generador. Particularmente, se implementó un GCL con parámetros que determinan un módulo reducido: $m = 2^{16} = 65{,}536$, lo que implica que el período del generador no puede exceder dicho valor.

Para este experimento, se generaron **1.000.000 de valores pseudoaleatorios**, superando ampliamente el período del generador, con el objetivo de estudiar el efecto de la repetición cíclica sobre las propiedades estadísticas de la secuencia.

Los resultados mostraron que:

En el mapa de bits generado a partir de los valores binarizados, se observaron **patrones regulares visuales**, lo que indica la **presencia de ciclos** en la secuencia numérica. Este fenómeno es esperado al superar el período del generador, ya que la sucesión entra en una fase periódica donde los valores comienzan a repetirse en el mismo orden.

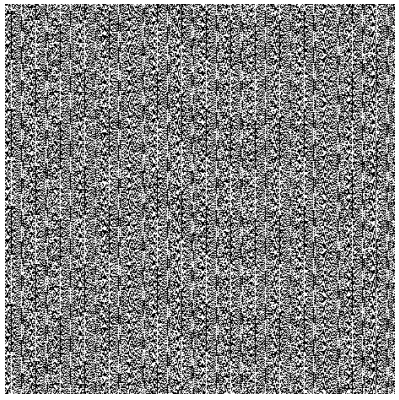

Figura 4: Visualización de números generados con GCL

Sin embargo, los **tests estadísticos clásicos** utilizados (media-varianza, chi-cuadrado, Kolmogórov-Smirnov y autocorrelación) **no detectaron anomalías significativas**. Los resultados obtenidos fueron:

Generador	Kolmogorov-Smirnov	Chi-Cuadrado	Media y Varianza	Autocorrelación
GCL	0.0001	0.09	0.4999 y 0.0833	0.0771

Cuadro 2: Resultados de los tests realizados para diferentes GCL

Esto pone en evidencia una **limitación de los tests estadísticos univariados**, los cuales pueden no ser sensibles a la **estructura cíclica o correlación a largo plazo** cuando la distribución marginal de los valores se mantiene aproximadamente uniforme. Es decir, aunque la distribución de frecuencias y otros indicadores se ajusten a lo esperado para una secuencia aleatoria, la **dependencia temporal introducida por la repetición del ciclo** puede pasar desapercibida para estos tests.

Aunque el **Generador Congruencial Lineal (GCL)** puede ser un **buen generador** de números pseudoaleatorios en situaciones donde el período no se ve comprometido, este experimento resalta la importancia de elegir un **número adecuado de corridas** que no exceda el **período del generador**. Si el número de corridas es mayor que el período, se introduce una **repetición cíclica** que afecta la aleatoriedad de la secuencia y, por ende, puede comprometer la calidad de los resultados generados.

En conclusión, este experimento demuestra que, al trabajar con generadores pseudoaleatorios, es **fundamental asegurar que el número de valores generados no exceda el período del generador**, o en su defecto, utilizar generadores con períodos suficientemente grandes como para evitar que los ciclos introduzcan correlaciones o patrones detectables, especialmente en contextos donde se requiere alta calidad de aleatoriedad.

5. Conclusión general

Los generadores de números pseudoaleatorios analizados en este trabajo presentan diferencias notables en calidad y desempeño según los tests aplicados:

- **Generador Python (Mersenne Twister):** Este generador mostró los mejores resultados en todos los aspectos evaluados. Los valores obtenidos para la media (0.5002) y la varianza (0.0833) están

cercanos a los valores ideales para una distribución uniforme. En cuanto a los tests estadísticos, los p-valores para la prueba de Kolmogorov-Smirnov (0.0006) y la prueba chi-cuadrado (10.86) indican que la secuencia generada se ajusta de manera muy adecuada a una distribución uniforme. La autocorrelación es prácticamente nula (0.0011), lo que refuerza la independencia de los números generados. Esto confirma que el Mersenne Twister sigue siendo una opción robusta y confiable para aplicaciones científicas y simulaciones.

- **Generador Congruencial Lineal (GCL):** Aunque el GCL mostró un comportamiento adecuado en cuanto a los estadísticos de media y varianza (0.5000 y 0.0833, respectivamente), su p-valor en la prueba de Kolmogorov-Smirnov (0.0005) y en la prueba chi-cuadrado (7.08) sugiere una leve desviación respecto a la distribución uniforme perfecta. A pesar de que estos resultados podrían considerarse satisfactorios, el test de autocorrelación reveló que la dependencia entre números consecutivos es mínima (0.0007). Sin embargo, se observó que, al realizar un gran número de corridas (en este caso, 1 millón), los ciclos dentro de la secuencia generada se hicieron evidentes, especialmente en la visualización de mapas de bits. Los tests de Kolmogorov-Smirnov y chi-cuadrado no lograron detectar estos ciclos, ya que estos tests se enfocan en la distribución uniforme de la secuencia en general, sin considerar la posible repetición de ciclos en una secuencia larga. Esto subraya la importancia de elegir un periodo adecuado en el GCL para evitar la repetición de secuencias y garantizar una mayor aleatoriedad en aplicaciones a largo plazo.
- **Generador de Cuadrados Medios:** Los resultados del generador de Cuadrados Medios fueron notoriamente deficientes. Los p-valores obtenidos en las pruebas de Kolmogorov-Smirnov (0.2100) y chi-cuadrado (1499900.00) son muy bajos, lo que indica un rechazo evidente de la hipótesis de uniformidad. La media y la varianza (0.5100 y 0.0500) se desvían considerablemente de los valores ideales esperados. Además, la autocorrelación de 0.0000 refleja una falta de dependencia inmediata entre valores consecutivos, lo cual podría ser engañoso ya que no garantiza una distribución uniforme ni la independencia en secuencias a largo plazo. Este generador tiene serias limitaciones y es inapropiado para aplicaciones modernas que requieran alta calidad en la generación de números pseudoaleatorios.

En resumen, el GCL, a pesar de tener un desempeño decente, muestra limitaciones en cuanto a la detección de ciclos cuando se utilizan periodos inadecuados, lo que no es capturado por los tests aplicados. En cambio, el generador Python (Mersenne Twister) sigue siendo la opción más fiable para aplicaciones que requieren una calidad estadística alta, mientras que el generador de Cuadrados Medios es claramente obsoleto y no es adecuado para aplicaciones actuales debido a sus serias deficiencias estadísticas.

6. Codigo

```
import argparse
import numpy as np
import random
from scipy.stats import chisquare, kstest
import matplotlib.pyplot as plt
import os

# GENERADORES
def gcl(seed, a, c, m, n):
    x = seed
    numeros = []
    for _ in range(n):
        x = (a * x + c) % m
        numeros.append(x / m)
    return numeros

def cuadrados_medios(seed, n):
    numeros = []
    x = seed
    for _ in range(n):
        x = int(str(x**2).zfill(8)[2:6])
        numeros.append(x / 10000)
    return numeros

def python_random(n):
    return [random.random() for _ in range(n)]

# TESTS ESTADÍSTICOS
```

```
def media_varianza_test(nums):
    media = np.mean(nums)
    varianza = np.var(nums)
    return media, varianza

def chi_cuadrado_test(nums, bins=10):
    freq, _ = np.histogram(nums, bins=bins, range=(0, 1))
    esperada = len(nums) / bins
    chi2, p = chisquare(freq, f_exp=[esperada]*bins)
    return chi2, p

def kolmogorov_smirnov_test(nums):
    d, p = kstest(nums, 'uniform')
    return d, p

def autocorrelacion_test(nums, lag=1):
    n = len(nums)
    x = np.array(nums)
    x_mean = np.mean(x)
    num = np.sum((x[:n-lag] - x_mean) * (x[lag:] - x_mean))
    den = np.sum((x - x_mean)**2)
    r = num / den
    return r

def guardar_visualizacion_bitmap(nums, nombre, size=(512, 512), carpeta="visualizaciones_prueba"):
    if not os.path.exists(carpeta):
        os.makedirs(carpeta)
    total_elementos = len(nums)
    if total_elementos < size[0] * size[1]:
        lado = int(np.sqrt(total_elementos))
        size = (lado, lado)
        print(f"Advertencia: Ajustando bitmap a {size} debido a que hay solo {total_elementos} números")
    data = np.array(nums[:size[0]*size[1]])
    binarizado = np.where(data > 0.5, 1, 0)
    imagen = binarizado.reshape(size)
    plt.figure(figsize=(6, 6), dpi=100)
    plt.imshow(imagen, cmap='gray', interpolation='nearest')
    plt.axis('off')
    plt.savefig(f"{carpeta}/{nombre}_bitmap.png", bbox_inches='tight', pad_inches=0)
    plt.close()

# MAIN
if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("--gcl-seed", type=int, default=1234567890, help="Semilla para GCL(10 digitos)")
    parser.add_argument("--cuad-seed", type=int, default=1234, help="Semilla para Cuadrados Medios(4 digitos)")
    parser.add_argument("-n", "--n", type=int, default=10000, help="Número de corridas")
    args = parser.parse_args()

    GCLseed = args.gcl_seed
    cuad_seed = args.cuad_seed
    n = args.n

    print(f"Using GCL seed: {GCLseed}")
    print(f"Using Cuadrados_Medios seed: {cuad_seed}")

    a = 1664525
    c = 1013904223
    m = 2**32

    if(len(str(GCLseed))!=10 or len(str(cuad_seed))!=4):
        print("Error: La semilla de GCL debe tener 10 digitos y la de cuadrados medios 4 digitos")
        exit(1)

    gcl_nums = gcl(GCLseed, a, c, m, n)
    cuad_nums = cuadrados_medios(cuad_seed, n)
    py_nums = python_random(n)

    generadores = {
        "GCL": gcl_nums,
        "Cuadrados_Medios": cuad_nums,
        "Python": py_nums
    }

    for nombre, nums in generadores.items():
        nums = np.array(nums, dtype=np.float32)

        print(f"\n--- Resultados para {nombre} ---")
        media, varianza = media_varianza_test(nums)
        chi2, p_chi = chi_cuadrado_test(nums)
        d, p_ks = kolmogorov_smirnov_test(nums)
```

```
r = autocorrelacion_test(nums)

print(f"Media: {media:.4f}, Varianza: {varianza:.4f}")
print(f"Chi²: {chi2:.2f}, p-valor: {p_chi:.4f}")
print(f"KS D: {d:.4f}, p-valor: {p_ks:.4f}")
print(f"Autocorrelación (lag 1): {r:.4f}")

guardar_visualizacion_bitmap(nums, f"{nombre}")
```