Universidad del Istmo
Curso: Análisis de datos
Catedrático: Juan Andrés García Porres

Investigación e implementaciones

Tarea 2

Juan Pablo Estrada Lucero
Ingeniería en sistemas y ciencias de la computación
0000012782
Fecha de entrega: 12 / 09 / 2025

<u>Primera parte</u>

- **Generadores de congruencia lineal (LCG)**
  Algoritmo que permite obtener secuencia de números pseudoaleatorios calculados con una función lineal definida a trozos discontinua.

  - Fundamento matemático:
    La secuencia de números enteros X_1, X_2, X_3, … está definida por la relación de recurrencia:

    $$X_i = (aX_{i-1} + c) \bmod m$$

    con $0 \leq X_i \leq m - 1$ donde m (el módulo), a (el multiplicador), c (el incremento) y X_0 (la semilla o valor inicial) son números enteros no negativos.

    Además a los números m , a , c y X_0 se les impone las condiciones $m > 0$, $0 < a < m$, $0 \leq c < m$ y $0 \leq X_0 < m$.

    Su período es la longitud máxima de la secuencia antes de repetirse en m.

    Tendrá período completo m para cualquier valor de la semilla X_0 en { 0, 1, … , m − 1 } si y sólo si:
    1. c & m son coprimos, mcd(c,m) = 1
    2. a - 1 es múltiplo de todos los factores primos de m, es decir, a = 1 mod z para todo z que sea factor primo de m.
    3. Si m es múltiplo de 4, entonces a - 1 también lo es.

  - Pseudocódigo:

```
Función LCG(semilla, a, c, m, cantidad)
   // semilla: número inicial
   // a: multiplicador
   // c: incremento
   // m: módulo
   // cantidad: cuántos números aleatorios generar

   X = semilla  // Valor inicial

   Repetir cantidad veces:
      X = (a * X + c) mod m  // Fórmula principal
      Print(X)  // Imprimir el número pseudoaleatorio generado
   Fin Repetir
Fin Función
```

- **Método de cuadros medios o middle-square method**
  Método para generar números pseudoaleatorios.

  - Fundamento matemático:
    Cada número sucesivo se genera tomando, los “n” dígitos centrales del cuadrado del número anterior de n dígitos.

- Pseudocódigo:

```
Función CuadrosMedios(semilla, cantidad, n_digitos)
   // semilla: número inicial
  // cantidad: cantidad de números a generar
  // n_digitos: cantidad de dígitos que tendrá cada número generado

   X = semilla  // Valor inicial

   Repetir cantidad veces:
      cuadrado = X * X  // Elevar al cuadrado
       texto = convertir a cadena con ceros a la izquierda hasta tener el
doble de dígitos

      // Extraer los dígitos del centro
      inicio = posición desde donde extraer los dígitos del centro
      centro = extraer n_digitos desde la posición inicio

      X = convertir centro a número
      Print(X) // Imprimir número pseudoaleatorio generado
   Fin Repetir
Fin Función
```

- **Mersenne Twister**
  Generador de números pseudoaleatorios desarrollado en 1997

  - Fundamento matemático:
    Este genera números en el rango de $[0, 2^w\text{-}1]$

    El algoritmo Mersenne Twister se basa en una recurrencia lineal matricial sobre un campo binario finito F^2.

    Consiste en definir una serie x_i, a través de una relación de recurrencia simple, y luego generar números con la forma $x^T{}_i$, donde T es una matriz F^2 invertible llamada matriz de templado.

    El algoritmo general se caracteriza por las siguientes magnitudes:

    w: tamaño de palabra (en bits)
    n: grado de recurrencia
    m: palabra intermedia, un desplazamiento utilizado en la relación de recurrencia que define la serie x , $1 \leq m < n$
    r: punto de separación de una palabra, o el número de bits de la máscara de bits inferior, $0 \leq r \leq w - 1$
    a: coeficientes de la matriz de torsión en forma normal racional
    b, c: máscaras de bits de templado TGFSR(R)
    s, t: desplazamientos de bits de templado TGFSR(R)

u, d, l: desplazamientos/máscaras de bits de templado Mersenne Twister adicionales

La serie x se define como una serie de cantidades de w bits con la relación de recurrencia:

$$x_{k+n} := x_{k+m} \oplus \left(({x_k}^u \mid {x_{k+1}}^l) A \right) \qquad k = 0, 1, 2, \ldots$$

Fuente: https://en.wikipedia.org/wiki/Mersenne_Twister

Para inicializar el estado interno del algoritmo se requiere:

$$\mathbf{x}_i = f \cdot (\mathbf{x}_{i-1} \oplus (\mathbf{x}_{i-1} \gg (w-2))) + i$$

Fuente: Presentación - Clase 9 - Análisis de datos

- donde f es igual a 1812433253

Dada una semilla n, inicial, se puede generar:

$$\mathbf{x}_n := \mathbf{x}_m \oplus (\mathbf{x}_0^u | \mathbf{x}_1^l) A$$

Fuente: Presentación - Clase 9 - Análisis de datos

si k = 0, las únicas constantes son n, m, r y A. En este caso, A es una matriz dispersa en donde la última fila se denota por $a = (a_{w-1}, \ldots, a_0)$. Si $x = (x_{w-1}, \ldots, x_0)$, entonces:

$$\mathbf{x}A = (x_0 a_{w-1}, x_{w-1} + x_0 a_{w-2}, \ldots, x_1 + x_0 a_0)$$

Fuente: Presentación - Clase 9 - Análisis de datos

En el caso de MT19937, se tiene que w = 32, n = 624, m = 397, r = 31 y a = 0x9908B0DF

Luego de ello, se hace lo que se denomina el Tempering, a través de una transformación invertible T tal que:

$$\mathbf{x} \mapsto \mathbf{z} = \mathbf{x}T$$

Fuente: Presentación - Clase 9 - Análisis de datos

donde T consiste de las siguientes operaciones:

$$\begin{aligned}
\mathbf{y} &:= \mathbf{x} \oplus (\mathbf{x} \gg u) \\
\mathbf{y} &:= \mathbf{y} \oplus ((\mathbf{y} \ll s) \otimes \mathbf{b}) \\
\mathbf{y} &:= \mathbf{y} \oplus ((\mathbf{y} \ll t) \otimes \mathbf{c}) \\
\mathbf{z} &:= \mathbf{y} \oplus (\mathbf{y} \gg l)
\end{aligned}$$

Fuente: Presentación - Clase 9 - Análisis de datos

Resumiendo los pasos a realizar en el algoritmo:

1. Inicializamos un array de tamaño 624 vectores binarios compuesto de 32 bits, donde la primera entrada es la semilla inicial y las otras se dan por la transformación de $x_{i.}$

2. Generamos el pseudoaleatorio, aplicando el apartado de tempering a cada entrada inicial.

3. Luego de terminar las entradas, generamos nuevas a través del apartado del twisting.

4. Luego para normalizar el valor de los outputs, dividimos entre 2^{32}.

- Pseudocódigo:

```
Función MersenneTwister_Inicializar(semilla)
   // semilla: número inicial para generar el estado
   Definir n = 624
   Definir w = 32
   Definir f = 1812433253

   estado[0] = semilla

   Repetir i desde 1 hasta n-1:
      estado[i] = (f * (estado[i-1] XOR (estado[i-1] >> (w-2))) + i) mod 2^w
   Fin Para

   indice = 0
   Retornar (estado, indice)
   Fin Función


Función MersenneTwister_Generar(estado, indice)
   Definir constantes:
      n = 624, m = 397, w = 32, r = 31
      a = 0x9908b0df
      u = 11, s = 7, t = 15, l = 18
      b = 0x9d2c5680, c = 0xefc60000
      UMASK = (2^w - 1) << r   // bits altos
      LMASK = (2^w - 1) >> (w - r) // bits bajos

   k = indice
   j = (k - (n - 1)) mod n

   // Combinar bits altos y bajos de dos posiciones
   x = (estado[k] AND UMASK) OR (estado[j] AND LMASK)

   // Mezclar con desplazamiento
   xA = x >> 1
```

```
    Si (x es impar) Entonces
        xA = xA XOR a
    Fin Si

    // Combinar con un estado anterior
    j = (k - (n - m)) mod n
    x = estado[j] XOR xA

    estado[k] = x
    k = (k + 1) mod n
    indice = k

    // Tempering
    y = x
    y = y XOR (y >> u)
    y = y XOR ((y << s) AND b)
    y = y XOR ((y << t) AND c)
    z = y XOR (y >> l)

    Retornar (z, estado, indice)
Fin Función


Función MersenneTwister_Aleatorio(semilla, cantidad)
    // Genera "cantidad" números pseudoaleatorios
    (estado, indice) = MersenneTwister_Inicializar(semilla)

    Repetir cantidad veces:
        (numero, estado, índice) = MersenneTwister_Generar(estado, indice)
        Print(numero)
    Fin Repetir
Fin Función
```

- **Blum Blum Shub**
  Generador pseudoaleatorio de números.
  - Fundamento matemático:
    Este se compone de la siguiente manera:
    $$x_{n+1} = (x_n)^2 \bmod M$$
    - donde:
      M = p*q (donde p y q son dos números primos muy grandes)

    En cada paso del algoritmo se obtiene un resultado para x_n los dos números primos, p y q, deben ser ambos congruentes a 3 (mod 4)

  - Pseudocódigo:

```
        Función BlumBlumShub(semilla, M, cantidad)
            // semilla: número inicial (debe ser coprimo con M)
```

```
    // M: número compuesto obtenido de multiplicar dos primos grandes
    // cantidad: cuántos números pseudoaleatorios se quieren generar

    X = semilla

    Repetir cantidad veces:
        X = (X * X) mod M //Elevación al cuadrado

        Print(X)  // Imprimir número pseudoaleatorio generado
    Fin Repetir
Fin Función
```

- **RANDU**
    - Fundamento matemático:
      Es un LCG, el cual se define a través de la siguiente recurrencia:
      $$V_{j+1} = 65539 * V_j \bmod 2^{31}$$

      Teniendo en consideración que V_0 es un número impar.

      La normalización del valor se obtiene mediante:
      $$X_j = V_j / 2^{31}$$

    - Pseudocódigo:

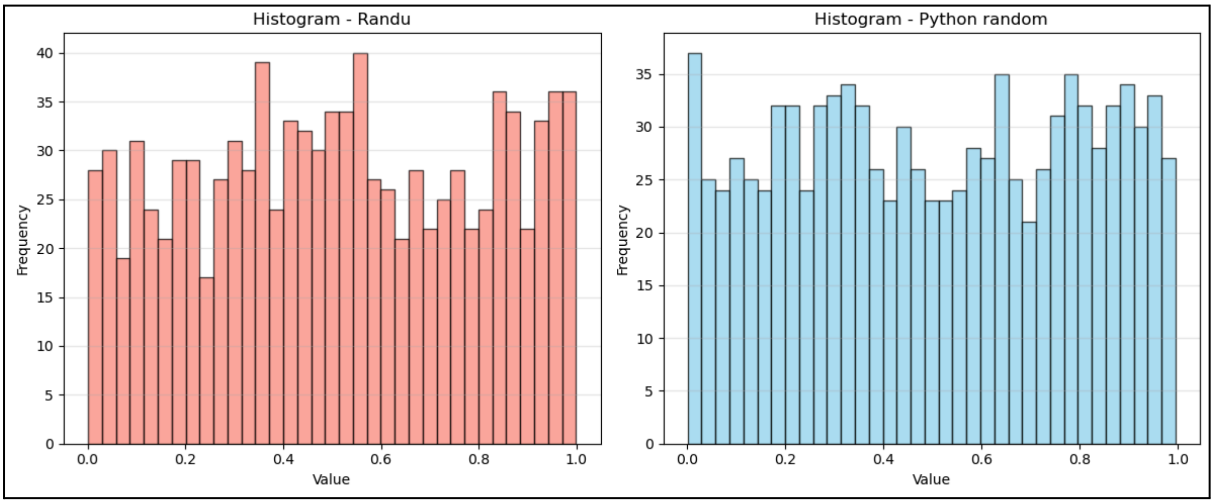
```
Función RANDU(semilla, cantidad)
    // semilla: número inicial
    // cantidad: cuántos números pseudoaleatorios a generar
    a = 65539  // Multiplicador específico del RANDU
    m = 2^31

    X = semilla

    Repetir cantidad veces:
        X = (a * X) mod m  // Fórmula del generador
        Print(X) // Imprimir el número pseudoaleatorio generado
    Fin Repetir
Fin Función
```

Tabla comparativa entre cada algoritmo:

Algoritmo	Velocidad	Periodo	Seguridad / Criptografía	Casos de uso típicos
Generadores de congruencia lineal (LCG)	Muy rápida	Moderado, depende de los parámetros (hasta ~2^{31}–1 para 32 bits)	Baja, predecible, no seguro criptográficamente	Simulaciones simples, juegos, prototipos, generación de datos no críticos

Método de cuadrados medios (Middle-square)	Lenta, especialmente para números grandes	Muy corto, suele degenerar rápidamente	Muy baja, altamente predecible y con ciclos cortos	Educación, experimentos históricos, demostración de conceptos
Mersenne Twister	Muy rápida	Muy largo ($\sim 2^{19937}-1$)	Media, no es seguro para criptografía	Simulaciones científicas, videojuegos, generación de números aleatorios de propósito general
Blum Blum Shub	Lenta, requiere operaciones de factorización	Muy largo (depende de los números primos grandes usados)	Muy alta, es seguro criptográficamente	Aplicaciones criptográficas, generación de claves, tokens de seguridad
RANDU	Rápida	Moderado (ciclos de $\sim 2^{31}/4$)	Muy baja, tiene correlaciones graves	Históricamente usado en mainframes IBM, hoy en día obsoleto

<u>Comparación entre Randu y librería Random de python:</u>



Fuente: Propia

Al visualizar el histograma de ambos algoritmos, se puede visualizar que bajo las mismas condiciones, la librería de random llega a tener una mejor uniformidad entre los valores, por otra parte, el valor de chi^2 para random es mucho mejor que el que se obtiene del algoritmo implementado de randu, por lo que se puede comprender que este tiende a tener una mucho mejor uniformidad respecto a randu, por lo que lo visto en el histograma se sustenta de estos valores, adicionalmente, el valor de Randu suele estar muy cerca de los valores críticos, lo cual da un indicio que la generación de valores no es completamente uniforme y puede presentar sesgos o correlaciones no deseadas, reduciendo su confiabilidad como generador pseudoaleatorio.

Bibliografía:

Andrew Dotson. (2018, June 8). *Monte Carlo Integration In Python For Noobs*. YouTube. https://www.youtube.com/watch?v=WAf0rqwAvgg

Date un Voltio. (2015, July 7). *¿En qué consiste el Método Montecarlo?* YouTube. https://www.youtube.com/watch?v=WJjDr67frtM

de, C. (2005, June 20). *método de resolución de problemas*. Wikipedia.org; Wikimedia Foundation, Inc. https://es.wikipedia.org/wiki/M%C3%A9todo_de_Montecarlo

de, C. (2007, September 16). *Blum Blum Shub*. Wikipedia.org; Wikimedia Foundation, Inc. https://es.wikipedia.org/wiki/Blum_Blum_Shub

de, C. (2012, January 11). *Mersenne twister*. Wikipedia.org; Wikimedia Foundation, Inc. https://es.wikipedia.org/wiki/Mersenne_twister

de, C. (2024, August 20). *generador de números pseudoaleatorios*. Wikipedia.org; Wikimedia Foundation, Inc. https://es.wikipedia.org/wiki/M%C3%A9todo_del_medio_del_cuadrado

GeeksforGeeks. (2022, November 3). *How to Add leading Zeros to a Number in Python*. GeeksforGeeks. https://www.geeksforgeeks.org/python/how-to-add-leading-zeros-to-a-number-in-python/

Generador lineal congruencial. (2023, July 17). Wikipedia. https://es.wikipedia.org/wiki/Generador_lineal_congruencial

Leon Ramirez. (2019, October 1). *Metodo de Montecarlo para solución de Integrales*. YouTube. https://www.youtube.com/watch?v=hg-Ghcedu_Y

NeuralNine. (2023, June 21). *Pseudo-Random Number Generator From Scratch in Python*. YouTube. https://www.youtube.com/watch?v=mXBGXU0zJnw

Statistics Crystallized. (2025, June 4). *The Most Popular Pseudo-Random Number Generator - The Mersenne-Twister*. YouTube. https://www.youtube.com/watch?v=TF4PLUcJO5w

Welcome To Zscaler Directory Authentication. (2025). Wordpress.com. https://simulacion2017.wordpress.com/2017/02/23/2-2-1-algoritmo-de-cuadrados-medios/

Wikipedia Contributors. (2024, August 6). *RANDU*. Wikipedia; Wikimedia Foundation.