Generadores de Números Pseudoaleatorios (PRNG) – LCG, Middle-Square, Mersenne Twister, Blum Blum Shub, RANDU

Carlos Solares

Los **generadores de números pseudoaleatorios (PRNG)** producen secuencias de números que *parecen* aleatorios a partir de procedimientos deterministas. A continuación, exploramos cinco PRNG clásicos, presentando su pseudocódigo, fundamento matemático y comparándolos en velocidad, período, seguridad y aplicaciones modernas.

1. Generador Congruencial Lineal (LCG)

Un **LCG (Linear Congruential Generator)** genera la secuencia $\{X_n\}$ mediante la **relación de recurrencia lineal modular** [1, 2]:

$$X_{n+1} = (a\,X_n + c) \text{ mód } m,$$

donde m es el *módulo*, a el *multiplicador*, c el *incremento* y X_0 la *semilla*. Un pseudocódigo sencillo del LCG es:

```
# Parametros: m (modulo), a (multiplicador), c (incremento), X0 (semilla)
X <- X0
Mientras se requieran mas numeros pseudoaleatorios:
    X <- (a * X + c) mod m
    devolver X # (usar X/m para obtener valor en [0,1) si se desea)
```

Fundamento matemático. Para que un LCG tenga **período completo** (maximice la longitud de la secuencia antes de repetirse), los parámetros deben satisfacer el *teorema de Hull–Dobell* [1, 2]: (1) c y m coprimos, (2) $a-1$ divisible por todos los factores primos de m, y (3) $a-1$ divisible por 4 si m lo es. Bajo estas condiciones, el período puede alcanzar m. Sin embargo, un período largo por sí solo no garantiza alta calidad; la elección de a y m es crítica para la uniformidad de la secuencia. Un ejemplo famoso de mala elección es **RANDU** (sección 5).

Ventajas y usos. Muy **eficientes** computacionalmente (una multiplicación y suma modular por número). Útiles donde se requiere gran volumen de números reproducibles (simulaciones sencillas, videojuegos).

Desventajas y seguridad. **No son criptográficamente seguros**: dada suficiente salida, es relativamente fácil predecir siguientes valores (linealidad módulo m).

2. Método de los Cuadrados Medios (Middle-Square)

El **método del cuadrado medio** fue propuesto por John von Neumann (1949) como uno de los primeros PRNG para simulaciones Monte Carlo [3]. Idea: tomar un número, elevarlo al cuadrado y extraer dígitos centrales como nuevo número.

```
# semilla de n digitos decimales (n par)
X <- semilla_n_digitos
Mientras se necesiten mas numeros:
    Y <- X^2
    # representar Y con 2n digitos rellenando con ceros a la izquierda
    Y_str <- formato_de_longitud_2n(Y)
    X <- digitos_centrales(Y_str, n)
    devolver X
```

Fundamento matemático. En la práctica, este método tiene **períodos muy cortos** y tiende a caer en ciclos pequeños o en 0 rápidamente. Un límite superior clásico para el número de estados distintos es $\leq 8^n$ (con n dígitos decimales), pero típicamente es menor [2]. A pesar de su *alta velocidad*, sus **deficiencias estadísticas** son severas.

Aplicaciones modernas. No se usa en producción por su mala calidad estadística; se mantiene como referencia histórica o pedagógica. Existen variantes modernas que lo combinan con secuencias de Weyl para mitigar fallos (p. ej. [4]), pero no son estándar industrial.

3. Mersenne Twister (MT19937)

El **Mersenne Twister** (1997) de Matsumoto y Nishimura [5] es ampliamente usado por su excelente balance rendimiento/calidad. Tiene un **período** $2^{19937} - 1$ (primo de Mersenne) y **equidistribución** hasta 623 dimensiones.

Fundamento y operación (esquema). Mantiene un *estado* de 624 enteros de 32 bits. Cada iteración:

1. Combina bits altos/bajos de dos entradas adyacentes del estado para formar un valor intermedio.
2. Aplica una transformación lineal tipo LFSR (*twist*) con desplazamientos y XOR; reinyecta al estado.
3. Aplica un *tempering* (XOR y desplazamientos) al valor de salida para mejorar distribución de bits.

El pseudocódigo completo es extenso; suele emplearse la implementación estándar [5].

Características. Muy rápido, genera millones de números/seg. **Período enorme** $2^{19937} - 1$. **No criptográfico**: si se recupera el estado (624 salidas), la secuencia se vuelve predecible.

Aplicaciones. Simulaciones Monte Carlo, modelado estadístico, motores de juegos, generación procedural; incluido en múltiples *runtimes* (p. ej. `std::mt19937` de C++).

4. Blum Blum Shub (BBS)

Blum Blum Shub (1986) es un PRNG *cripto-seguro* con seguridad demostrable basada en la dificultad de factorizar [6]. Define:

$$x_{n+1} = x_n^2 \bmod M, \quad M = pq,$$

donde p, q son primos de Blum ($p \equiv q \equiv 3 (\bmod 4)$). La salida suele ser el LSB de x_{n+1} (o varios bits).

```
# Elegir p, q primos grandes: p == q == 3 (mod 4)
M <- p * q
Elegir semilla x, 1 < x < M, gcd(x,M)=1
Mientras se necesiten mas bits/numeros:
    x <- (x * x) mod M
    bit <- LSB(x) # o extraer algunos bits bajos
    devolver bit
```

Seguridad. Bajo supuestos estándar, predecir el siguiente bit es tan difícil como **factorizar** M (relacionado con residuocidad cuadrática); por ello BBS es un *CSPRNG*.

Período y rendimiento. El período efectivo depende de p, q y de la semilla, pudiendo ser enorme (longitudes de miles de bits para M). **Es lento**: cada paso requiere aritmética modular de gran tamaño y típicamente rinde pocos bits por iteración. En práctica suele preferirse AES-CTR, ChaCha20, etc., por eficiencia.

5. RANDU

RANDU es un LCG histórico muy utilizado en IBM System/360 (1960s–70s) [9], definido por:

$$V_{j+1} = 65539 \cdot V_j \text{ mód } 2^{31},$$

con semilla impar. Fue célebre por sus **graves defectos estadísticos**: los triples (V_j, V_{j+1}, V_{j+2}) caen en unos pocos planos en 3D ("random numbers fall mainly in the planes") [7]. Además, su período efectivo es $\approx 2^{29}$, menor al máximo teórico, por mala elección de parámetros.

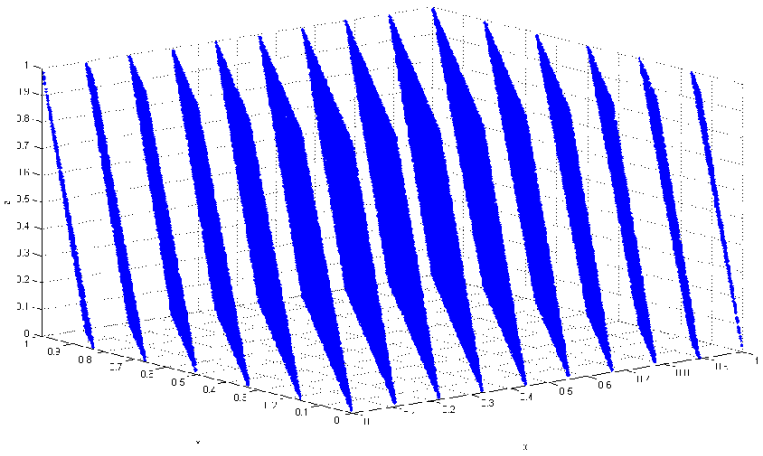

Figura 1: Tripletas consecutivas (V_j, V_{j+1}, V_{j+2}) de RANDU muestran concentración en planos paralelos (correlación fuerte [7]).

Lección. RANDU ilustra que la eficiencia sin calidad estadística invalida simulaciones. Motivó pruebas como el *spectral test* y mejores PRNG (p. ej. Park–Miller [8]).

Comparativa: Velocidad, Período, Seguridad y Usos

Algoritmo	Velocidad	Período	Seguridad	Casos de uso típicos
LCG	Muy alta	Hasta m (si Hull–Dobell)	No segura	Simulaciones sencillas, videojuegos; hoy se prefieren alternativas más robustas [1].
Middle-Square	Muy alta (cómputo)	Muy corto (ciclos/0)	No segura	Histórica/pedagógica; variantes tipo Weyl como curiosidad moderna [4].
Mersenne Twister	Alta (millones/s)	$2^{19937}-1$	No segura	Simulación/MC, estadística, juegos; PRNG por defecto en varios entornos [5].
Blum Blum Shub	Baja (modular grande)	Enorme (dep. de M)	CSPRNG (sí)	Criptografía teórica/alta seguridad; en práctica se usan AES-CTR/ChaCha20 por rendimiento [6].
RANDU	Muy alta (histórica)	$\approx 2^{29}$	Nula (mala estadística)	Ejemplo histórico de *qué no usar*; reemplazado por generadores mejores [7, 9].

Notas. Un período largo no garantiza calidad; tests espectrales y baterías como TestU01/Diehard son esenciales para validar PRNG. Para *seguridad*, usar CSPRNG modernos (AES-CTR, ChaCha20, Hash-DRBG, etc.).

6. Métodos de Monte Carlo para Aproximación de Integrales

Los métodos de Monte Carlo (MC) aproximan integrales por promediado de muestras aleatorias. Sea $f:[a,b]\to\mathbb{R}$ integrable y $X\sim \mathrm{Unif}(a,b)$. Entonces

$$\mathbb{E}[f(X)] \;=\; \frac{1}{b-a}\int_a^b f(x)\,dx \quad\Longrightarrow\quad \int_a^b f(x)\,dx \;=\; (b-a)\,\mathbb{E}[f(X)]$$

Usando n muestras i.i.d. $X_1,\dots,X_n\sim\mathrm{Unif}(a,b)$, el **estimador MC** de la integral es

$$\widehat{I}_n \;=\; \frac{b-a}{n}\sum_{i=1}^{n} f(X_i) \tag{1}$$

Varianza y error estándar Sea $Y=(b-a)f(X)$. Entonces $\widehat{I}_n=\frac{1}{n}\sum_{i=1}^n Y_i$ con Y_i i.i.d., de modo que

$$\mathbb{E}[\widehat{I}_n]=\int_a^b f(x)\,dx, \qquad \mathrm{Var}(\widehat{I}_n)=\frac{\mathrm{Var}(Y)}{n}=\frac{(b-a)^2\,\mathrm{Var}(f(X))}{n}$$

En práctica, estimamos el *error estándar* con la varianza muestral de Y_i: si $s_Y^2=\frac{1}{n-1}\sum_{i=1}^n(Y_i-\overline{Y})^2$, entonces

$$\mathrm{EE}(\widehat{I}_n) \;\approx\; \frac{s_Y}{\sqrt{n}}$$

Intervalo de confianza (CLT) Por el Teorema Central del Límite, para n grande

$$\widehat{I}_n \;\approx\; \mathcal{N}\Big(\int_a^b f(x)\,dx,\; \frac{(b-a)^2\,\mathrm{Var}(f(X))}{n}\Big),$$

y un IC bilateral del $100(1-\alpha)\,\%$ es

$$\widehat{I}_n \;\pm\; z_{\alpha/2}\,\frac{s_Y}{\sqrt{n}},$$

donde $z_{\alpha/2}$ es el cuantil normal estándar (p. ej., $z_{0{,}025}\approx 1{,}96$ para $95\,\%$)

Pseudocódigo MC (integración con $X \sim \textbf{Unif}(a,b)$)

```
Entrada: funcion f(x), intervalo [a,b], size de muestra n
Salida: estimacion I_hat, error_est (aprox), IC_95

for i = 1..n:
    Xi <- Uniforme(a,b)
    Yi <- (b - a) * f(Xi)
I_hat <- promedio(Yi)
sY2 <- varianza_muestral(Yi)
error_est <- sqrt(sY2 / n)
IC_95 <- [ I_hat - 1.96*error_est , I_hat + 1.96*error_est ]
devolver (I_hat, error_est, IC_95)
```

Ejemplo 1: $\int_0^1 \sin(\pi x)\,dx$

Aquí $a = 0$, $b = 1$ y $f(x) = \sin(\pi x)$. El estimador MC es

$$\widehat{I}_n = \frac{1-0}{n}\sum_{i=1}^{n}\sin(\pi X_i) = \frac{1}{n}\sum_{i=1}^{n}\sin(\pi X_i), \quad X_i \sim \text{Unif}(0,1)$$

El valor analítico de la integral es

$$\int_0^1 \sin(\pi x)\,dx \;=\; \Big[-\frac{\cos(\pi x)}{\pi}\Big]_0^1 \;=\; \frac{2}{\pi} \;\approx\; 0{,}63662$$

Varianza (cálculo cerrado). Como $X \sim \text{Unif}(0,1)$ y $f(X) = \sin(\pi X)$:

$$\mathbb{E}\big[\sin^2(\pi X)\big] = \int_0^1 \sin^2(\pi x)\,dx = \frac{1}{\pi}\int_0^{\pi} \sin^2 t\,dt = \frac{1}{2}$$

Además, $\mathbb{E}[\sin(\pi X)] = \frac{2}{\pi}$, así que

$$\text{Var}(f(X)) = \tfrac{1}{2} - \left(\tfrac{2}{\pi}\right)^2 \approx 0{,}0947$$

Como $b - a = 1$, se tiene $\text{Var}(\widehat{I}_n) = \text{Var}(f(X))/n \approx 0{,}0947/n$ y

$$\text{EE}(\widehat{I}_n) \approx \frac{0{,}30775}{\sqrt{n}}$$

Ejemplo 2: $\int_0^2 \frac{1}{\sqrt{2\pi}} e^{-x^2/2}\,dx$

Sea $\varphi(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$ la densidad normal estándar. Queremos

$$I = \int_0^2 \varphi(x)\,dx$$

Con $X \sim \text{Unif}(0,2)$ ($a = 0$, $b = 2$) y $f(x) = \varphi(x)$, el estimador MC es

$$\widehat{I}_n = \frac{2-0}{n}\sum_{i=1}^{n}\varphi(X_i) = \frac{2}{n}\sum_{i=1}^{n}\varphi(X_i), \quad X_i \sim \text{Unif}(0,2)$$

El valor de referencia (vía CDF normal) es

$$I = \Phi(2) - \Phi(0) = \Phi(2) - \tfrac{1}{2} \approx 0{,}97725 - 0{,}5 \approx 0{,}47725$$

En este caso $\mathrm{Var}(f(X))$ no tiene forma cerrada simple bajo $\mathrm{Unif}(0,2)$, pero el error estándar se estima con la varianza muestral de $Y_i = 2\,\varphi(X_i)$:

$$\mathrm{EE}(\widehat{I}_n) \;\approx\; \frac{s_Y}{\sqrt{n}}, \qquad s_Y^2 = \frac{1}{n-1}\sum_{i=1}^{n}(Y_i - \overline{Y})^2$$

Con el IC del 95 %:

$$\widehat{I}_n \pm 1{,}96\,\frac{s_Y}{\sqrt{n}}$$

Integral	**Generador**	**n**	**Estim.**	**EE**	**IC$_{95\,\%}$ (inf)**	**IC$_{95\,\%}$ (sup)**	**Valor real**	**Error abs.**
$I1 = \int_0^1 \sin(\pi x)\,dx$	MT	200000	0.635299	0.000689	0.633949	0.636649	0.636620	0.001321
$I2 = \int_0^2 \varphi(x)\,dx$	MT	200000	0.476408	0.000515	0.475397	0.477418	0.477250	0.000842
$I1 = \int_0^1 \sin(\pi x)\,dx$	RANDU	200000	0.636281	0.000687	0.634934	0.637628	0.636620	0.000338
$I2 = \int_0^2 \varphi(x)\,dx$	RANDU	200000	0.476869	0.000514	0.475861	0.477877	0.477250	0.000381

Cuadro 1: Resultados de integración Monte Carlo para dos integrales de prueba, comparando el generador estándar de Python (MT19937) con RANDU. Se reporta el estimador, error estándar (EE), intervalo de confianza al 95 %, valor real de la integral y error absoluto.

Comparacion de resultados

Referencias

[1] P. L'Ecuyer, *Good Parameters and Implementations for Combined Multiple Recursive Random Number Generators*, Operations Research, 47(1), 159–164, 1999. (Ver también: P. L'Ecuyer, "Tables of Linear Congruential Generators of Different Sizes and Good Lattice Structure," *Math. Comp.*, 68(225):249–260, 1999.)

[2] D. E. Knuth, *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms* (3rd ed.), Addison-Wesley, 1997. (Caps. 3.2–3.3: LCG, pruebas espectrales, middle-square.)

[3] N. Metropolis, "The Beginning of the Monte Carlo Method," *Los Alamos Science*, Special Issue, 1987. (Contexto histórico; von Neumann y middle-square.)

[4] B. Widynski, "Middle Square Weyl Sequence RNG," arXiv:1704.00358, 2017.

[5] M. Matsumoto and T. Nishimura, "Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator," *ACM Transactions on Modeling and Computer Simulation*, 8(1):3–30, 1998.

[6] L. Blum, M. Blum, and M. Shub, "A Simple Unpredictable Pseudo-Random Number Generator," *SIAM Journal on Computing*, 15(2):364–383, 1986.

[7] G. Marsaglia, "Random Numbers Fall Mainly in the Planes," *Proceedings of the National Academy of Sciences (PNAS)*, 61(1):25–28, 1968.

[8] S. K. Park and K. W. Miller, "Random Number Generators: Good Ones Are Hard to Find," *Communications of the ACM*, 31(10):1192–1201, 1988.

[9] IBM, *IBM System/360 Scientific Subroutine Package (SSP)*. (Detalle histórico del uso de RANDU en mainframes IBM System/360; múltiples fuentes secundarias lo documentan.)

[10] R. E. Caflisch, "Monte Carlo and Quasi-Monte Carlo Methods," *Acta Numerica*, 7:1–49, 1998. (Referencia de revisión moderna sobre MC y sus aplicaciones en integración.)

[11] I. M. Sobol, "Uniformly Distributed Sequences with an Additional Uniform Property," *USSR Computational Mathematics and Mathematical Physics*, 16(5):236–242, 1976. (Métodos cuasi-Monte Carlo.)

[12] R. Y. Rubinstein, *Simulation and the Monte Carlo Method*, Wiley, 1981. (Texto clásico de Monte Carlo, aplicaciones en integración y optimización.)

[13] A. B. Owen, *Monte Carlo Theory, Methods and Examples*, Stanford University, 2013. Disponible en línea. (Tratado moderno, incluye estimadores, reducción de varianza y ejemplos de integración.)