Investigación Detallada: Generación de Pseudo-números Aleatorios y Caminatas Aleatorias

Angie Gómez, Leonardo Tovar

10 de diciembre de 2025

1. Introducción

El uso de números aleatorios es crucial en la simulación computacional de sistemas físicos, desde el estudio de gases hasta la dinámica cuántica. Dado que las computadoras son máquinas deterministas, es imperativo entender el concepto de *pseudoaleatoriedad* y las herramientas que garantizan la fidelidad de las simulaciones. Este trabajo aborda los fundamentos de los Generadores de Números Pseudo-Aleatorios (PRNG) y su aplicación en la modelización de fenómenos de transporte, como las caminatas aleatorias y la difusión.

2. Concepto y Requisitos de los Pseudo-números Aleatorios

2.1. Concepto de Pseudoaleatoriedad

Un **Generador de Números Pseudo-Aleatorios (PRNG)** es un algoritmo determinista que produce una secuencia de números que, aunque generados por una fórmula fija (partiendo de una *semilla* inicial), son indistinguibles de una secuencia verdaderamente aleatoria para la mayoría de las pruebas estadísticas. La clave reside en que esta secuencia es reproducible, lo cual es vital para la depuración y comparación de resultados en la investigación científica.

2.2. Requisitos Fundamentales de Calidad

Para que un PRNG sea apto para simulaciones físicas, debe satisfacer rigurosos criterios de calidad. La ausencia de cualquiera de ellos puede introducir *artefactos* en los resultados:

1. **Uniformidad:** La distribución de los números generados debe ser perfectamente uniforme en el intervalo deseado (usualmente $[0, 1]$).

2. **Periodo Máximo:** El número de valores que el generador produce antes de que la secuencia comience a repetirse debe ser astronómicamente grande (idealmente mucho mayor que el número total de muestras utilizadas en la simulación).

3. **Independencia y No Correlación:** No debe existir ninguna relación o correlación lineal entre números sucesivos, ni en pares (X_i, X_{i+1}), ni en ternas, ni en dimensiones superiores. Esta es la prueba más difícil para los generadores simples.

4. **Eficiencia:** Debe generar los números de manera rápida, ya que el costo computacional de las simulaciones de Monte Carlo a menudo está dominado por la generación de aleatoriedad.

3. Ejemplo de Generador Simple y Visualización de Correlaciones

3.1. Generador Lineal Congruencial (LCG)

El **Generador Lineal Congruencial (LCG)** es el PRNG más elemental y se define por la relación de recurrencia:

$$X_{i+1} = (aX_i + c) \pmod{m}$$

Donde X_i es el i-ésimo número, a es el multiplicador, c es el incremento, y m es el módulo. El periodo máximo se alcanza si los parámetros son elegidos cuidadosamente (ej. el periodo es m si $c \neq 0$).

3.2. Implementación Simple en C++

Aquí se muestra un ejemplo simple de la lógica LCG, que se puede implementar fácilmente para generar números enteros pseudoaleatorios.

```
long long lcg(long long& seed, long long a, long long c, long long m) {
    // Genera el siguiente número de la secuencia
    seed = (a * seed + c) % m;
    return seed;
}

// Ejemplo: a=1103515245, c=12345, m=2^31 (valores comunes)
```

3.3. Visualización de Correlaciones: El Problema de los Hiperplanos

A pesar de su simplicidad, el LCG exhibe una falla crítica en alta dimensión conocida como el **problema de las capas** o la **estructura de hiperplanos**. Si se toman k números sucesivos del LCG y se grafican como puntos en el espacio k-dimensional $(X_i, X_{i+1}, \ldots, X_{i+k-1})$, estos puntos no llenarán uniformemente el hipercubo, sino que caerán en un número limitado de hiperplanos paralelos.

Para $k = 2$, esto se visualiza trazando pares (X_i, X_{i+1}). En un generador pobre, los puntos se alinearán claramente en líneas diagonales. Un generador de calidad debe mostrar una dispersión uniforme que oculte cualquier patrón. Esta correlación es letal para simulaciones de Monte Carlo que dependen de variables independientes en alta dimensión.

4. Uso y Precauciones de los RNG en Simulaciones Físicas

4.1. Aplicaciones Clave

Los PRNGs son indispensables para:

1. **Integración Numérica (Monte Carlo):** Estimar integrales complejas en alta dimensión, donde los métodos tradicionales fallan.

2. **Estudios de Equilibrio:** Simular sistemas termodinámicos (ej. Modelo de Ising) a través de algoritmos como Metropolis.

3. **Procesos Estocásticos:** Modelar fenómenos donde interviene la aleatoriedad intrínseca (ej. Decaimiento radioactivo, colisiones de partículas).

4.2. Precauciones Críticas

- **Evitar Generadores Legacy:** El uso de funciones antiguas como `rand()` (típico LCG de bajo periodo) es inaceptable en trabajos serios.
- **Sembrado (Seeding) Seguro:** La semilla debe elegirse cuidadosamente. Para simulaciones reproducibles, se usa una semilla fija. Para asegurar una verdadera independencia entre múltiples corridas (ej. en cómputo paralelo), es necesario usar semillas generadas por fuentes de entropía robustas o generadores de alta dimensión para evitar solapamientos.
- **Pruebas de Batería:** Antes de ser adoptado, un PRNG debe pasar suites de pruebas estadísticas rigurosas como Diehard o TestU01, que validan su uniformidad y no correlación.

5. Descripción Detallada del Generador MIXMAX

El generador **MIXMAX** es un PRNG matricial diseñado específicamente para superar las deficiencias de los LCGs y otros generadores de menor calidad. Pertenece a la clase de Generadores Congruenciales Lineales Matriciales (MLCG).

- **Mecánica:** El estado del generador no es un solo número, sino un vector de N números. La generación del siguiente vector de estado $\mathbf{X}_{i+1}$ se realiza mediante la multiplicación de una matriz especial A por el vector de estado anterior $\mathbf{X}_i$:

$$\mathbf{X}_{i+1} = (A\mathbf{X}_i + \mathbf{C}) \pmod{M}$$

- **Fortalezas:** El diseño de la matriz A es clave. El generador MIXMAX, particularmente con un tamaño de matriz $N = 240$, tiene un periodo superior a 10^{430}, lo que lo hace prácticamente infinito para cualquier simulación. Sus propiedades de mezcla (MIX) y su periodo máximo (MAX) lo hacen ideal para cálculos de Física Teórica y Monte Carlo en la Red (Lattice Monte Carlo).

6. Caminata Aleatoria y su Relación con la Difusión

6.1. Definición de Caminata Aleatoria (Random Walk)

Una caminata aleatoria es un modelo matemático de trayectoria que consiste en una sucesión de pasos discretos, donde la dirección y/o longitud de cada paso es aleatoria. Es el modelo básico para describir el movimiento de partículas bajo colisiones aleatorias.

6.2. El Vínculo Fundamental con la Difusión

La caminata aleatoria es la manifestación microscópica del fenómeno macroscópico de la **Difusión** (Movimiento Browniano). La dispersión de las partículas se describe mediante la relación entre el desplazamiento cuadrático medio y el tiempo. Para una caminata aleatoria en un medio homogéneo, se cumple la ley de escalamiento:

$$\langle R^2(t) \rangle \propto t$$

Donde $\langle R^2(t) \rangle$ es la distancia cuadrática media recorrida después del tiempo t. Al compararla con la ecuación de difusión:

$$\langle R^2(t) \rangle = 2dDt$$

Donde d es la dimensión del espacio y D es el **coeficiente de difusión**.

- **Difusión Normal:** En una caminata aleatoria estándar, $\langle R^2(t) \rangle$ crece linealmente con el tiempo (t^1).
- **Difusión Anómala:** Si el exponente es diferente de 1 (ej. $\langle R^2(t) \rangle \propto t^\alpha$ con $\alpha \neq 1$), hablamos de difusión anómala, que ocurre en medios complejos o fractales.

La simulación de caminatas aleatorias usando PRNGs es, por tanto, una herramienta esencial para calcular numéricamente el coeficiente de difusión de un sistema.

7. Comparación Detallada de Generadores: `rand()`, `drand48()`, `<random>`

La evolución de los generadores en C/C++ refleja el aumento en la demanda de aleatoriedad de alta calidad en la ciencia:

1. `rand()` **(`stdlib.h`):** Esta es la función tradicional del estándar de C. Típicamente implementada como un LCG simple con un módulo de 2^{15} o 2^{31}. Su periodo y propiedades estadísticas son **pobres**. No es aceptable para simulaciones científicas rigurosas debido al riesgo de correlaciones.
2. `drand48()` **(POSIX):** Una mejora sobre `rand()`. Utiliza un LCG con un módulo más grande, de 48 bits, lo que resulta en un periodo significativamente mayor. Sin embargo, sigue siendo un LCG y puede fallar en pruebas estadísticas de alta dimensión, por lo que su uso es **limitado** a aplicaciones no críticas.
3. `<random>` **(C++11 en adelante):** Esta librería introdujo un marco de trabajo moderno y flexible, separando los *motores* de generación (PRNGs) de las *distribuciones*. Los motores más recomendados aquí son:
   - `std::mt19937` **(Mersenne Twister):** Es el estándar de facto. Tiene un periodo enorme de $2^{19937} - 1$ y pasa la mayoría de las pruebas. Es la opción **altamente recomendada** por defecto.
   - `std::ranlux24`**:** Ofrece una calidad estadística aún mayor, a costa de ser más lento que Mersenne Twister.

En resumen, la única opción viable para la física computacional moderna es la librería `<random>`, usando motores probados como `std::mt19937`.