

Modelación y Simulación 2025

Lab 07

30.octubre.2025

En las clases anteriores hemos aprendido cómo simular y generar una muestra de una variable aleatoria X , a partir de una variable uniforme. En este lab, investigaremos como construir un generador de una variable uniforme $Y \sim \text{Unif}(0, 1)$.

1. (Generador Pseudoaleatorio Uniforme).

Implementar un generador de números pseudo-aleatorios de tipo LCG (*linear congruential generator*), para generar

- a) una muestra uniforme finita con valores x_1, x_2, \dots, x_n .
- b) una muestra de una distribución uniforme $\text{Unif}(0, 1)$.

Defina usted los parámetros: el módulo m , las constantes $0 \leq a, c < m$, y el tamaño N de la muestra generada, y repita sus experimentos para 2 conjuntos diferentes de parámetros.

En ambos casos, muestre estadísticos, histogramas y elabore una prueba de hipótesis para contrastar la muestra generada contra la muestra requerida, para determinar el buen funcionamiento del generador pseudo aleatorio. Use como tamaño de muestra un valor adecuado de N .

2. (Mersenne Twister).

Investigar e implementar en Python un generador aleatorio de tipo *Mersenne Twister* para generar una distribución uniforme $\text{Unif}(0, 1)$.

Igual que en el ejercicio anterior, muestre estadísticos, histogramas y elabore una prueba de hipótesis para contrastar la muestra generada contra la muestra teórica, para determinar el buen funcionamiento del generador pseudo aleatorio. Use como tamaño de muestra un valor adecuado de N .

3. (Tests NIST SP 800-22).

Para determinar si un generador pseudo-aleatorio cumple estándares de calidad, se recomienda aplicarle una batería de pruebas estadísticas que evalúe uniformidad, independencia y ausencia de patrones deterministas en la secuencia generada.

Por ejemplo, la publicación *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications* describe una serie de 15 pruebas estadísticas que evalúan distintas propiedades de una secuencia binaria: frecuencia, autocorrelación, bloques, etc.

https://csrc.nist.gov/projects/random-bit-generation/documentation-and-software?utm_source=chatgpt.com

Investigar cómo implementar en Python la batería de pruebas NIST SP 800-22, y aplicarlas a una lista de 1,000,000 de bits aleatorios generados con el CLG y el Mersenne Twister de los Ejercicios 1 y 2. En Python se puede llamar a las librerías `sts-pylib`, `nistrng`, `sp80022suite` u otras que ya incluyen estos tests.

Elabore una tabla en donde se vea el desempeño de los generadores en cada uno de los test anteriores, junto con el p -value obtenido y concluya cuál de los dos métodos se desempeña mejor.

4. **(Comparar muestras de una Geométrica).**

Para una distribución geométrica $\text{Geom}(p)$, generar una muestra aleatoria de tamaño N usando la librería `scipy.stats`, a la cual llamaremos la muestra teórica. Luego, generar una muestra del mismo tamaño, pero usando el algoritmo de la transformada integral. A esta le llamaremos la muestra empírica.

Comparar ambas muestras usando las prueba de Chi Cuadrado y la prueba de Kolmogorov-Smirnov. ¿Qué concluye? ¿Se pueden considerar como muestras de una misma distribución? Explique sus conclusiones a partir de las pruebas de hipótesis. Use un nivel de confianza de $\alpha = 0.05$.

5. **(Comparar muestras de una Normal)** Para una distribución normal $\mathcal{N}(\mu, \sigma^2)$, generar una muestra aleatoria de tamaño N usando la librería `scipy.stats`, a la cual llamaremos la muestra teórica. Luego, generar una segunda muestra del mismo tamaño (llamada la muestra empírica), pero usando el algoritmo de la transformada integral.

Comparar ambas muestras usando las prueba de Chi Cuadrado y la prueba de Kolmogorov-Smirnov. ¿Qué concluye? ¿Se pueden considerar como muestras de una misma distribución? Explique sus conclusiones a partir de las pruebas de hipótesis. Use un nivel de confianza de $\alpha = 0.05$.