

Generación de números pseudo-aleatorios (LCG) & generación de VA

Juliana Quintana-Rojas

Departamento de Ingeniería Industrial

Universidad de Los Andes

Método de Congruencia Lineal (MCL) **Linear Congruent Generator (LCG)**

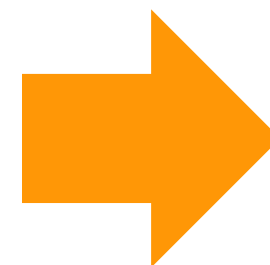
LCG para la generación de números pseudoaleatorios en computación es el método más extendido, introducido por **Lehmer en 195.**

Fundamento matemático

El método genera una secuencia de enteros $X_1, X_2, \dots$ basándose en una fórmula recursiva:

- **m (Módulo)**: Determina el rango de los valores generados ($0 \leq X_i \leq m-1$).
- **a (Multiplicador)**: Un entero constante menor que el módulo.
- **c (Incremento)**: Un entero constante.
 $c > 0$, se llama método mixto
 $c = 0$, se llama método multiplicativo
- **Z_0 (Semilla o Seed)**: El valor inicial que dispara la secuencia.

Número pseudo-aleatorio
 $U_i \rightarrow U_i \in [0,1]$



Método de Congruencia Lineal (MCL)

$a = 20, X_0 = 47, c = 35, \text{ y } m = 159$

Los valores X_i y U_i correspondientes serían:

Propiedades de un buen generador

- Periodo Largo:** Debido a su naturaleza determinista, la secuencia eventualmente se repite. *El periodo es la longitud de la secuencia antes de que los números vuelvan a aparecer en el mismo orden.*
- Densidad Máxima:** Los valores de U_i deben ser lo suficientemente densos en para no dejar grandes huecos.
- Velocidad y Portabilidad:** Debe ser rápido de calcular y producir los mismos resultados en diferentes computadoras.

Fuente	m	(multiplicador) a	(incremento) c	Bits de salida en $rand()$ o $Random(L)$
<i>Numerical Recipes</i>	2^{32}	1664525	1013904223	
Borland C/C++	2^{32}	22695477	1	bits 30..16 en $rand()$, 30..0 en $lrand()$
glibc (usado por GCC) ⁷	$2^{31} - 1$	1103515245	12345	bits 30..0
ANSI C: Watcom, Digital Mars, CodeWarrior, IBM VisualAge C/C++ ⁸ C99, C11: Suggestion in the ISO/IEC 9899 ⁹	2^{31}	1103515245	12345	bits 30..16
Borland Delphi, Virtual Pascal	2^{32}	134775813	1	bits 63..32 de (semilla * L)
Turbo Pascal	2^{32}	134775813 (0x8088405 ₁₆)	1	
Microsoft Visual/Quick C/C++	2^{32}	214013 (343FD ₁₆)	2531011 (269EC3 ₁₆)	bits 30..16
Microsoft Visual Basic (6 and earlier) ¹⁰	2^{24}	1140671485 (43FD43FD ₁₆)	12820163 (C39EC3 ₁₆)	
RtlUniform from Native API ¹¹	$2^{31} - 1$	2147483629 (7FFFFFFD ₁₆)	2147483587 (7FFFFFFC3 ₁₆)	
Apple CarbonLib, C++11's <code>minstd_rand0</code> ¹²	$2^{31} - 1$	16807	0	ver MINSTD
C++11's <code>minstd_rand</code> ¹²	$2^{31} - 1$	48271	0	ver MINSTD
MMIX by Donald Knuth	2^{64}	6364136223846793005	1442695040888963407	
Newlib, Musl	2^{64}	6364136223846793005	1	bits 63...32
VMS's <code>MTH\$RANDOM</code> , ¹³ old versions of glibc	2^{32}	69069 (10DCD ₁₆)	1	
.Random en Java, POSIX <code>[ln]rand48</code> , glibc <code>[ln]rand48[_r]</code>	$2^{48} - 1$	25214903917 (5DEECE66D ₁₆)	11	bits 47...16
<code>random0</code> ^{14 15 16 17 18} Si X_n es par, entonces X_{n+1} será impar, y viceversa— el bit más bajo oscila a cada paso.	$134456 = 2^{3 \cdot 7^5}$	8121	28411	$\frac{X_n}{134456}$
POSIX ¹⁹ <code>[jm]rand48</code> , glibc <code>[mj]rand48[_r]</code>	2^{48}	25214903917 (5DEECE66D ₁₆)	11	bits 47...15
POSIX <code>[de]rand48</code> , glibc <code>[de]rand48[_r]</code>	2^{48}	25214903917 (5DEECE66D ₁₆)	11	bits 47...0
cc65 ²⁰	2^{23}	65793 (10101 ₁₆)	4282663 (415927 ₁₆)	bits 22...8
cc65	2^{32}	16843009 (1010101 ₁₆)	826366247 (31415927 ₁₆)	bits 31...16
Anteriormente de uso común: RANDU ⁶	2^{31}	65539	0	

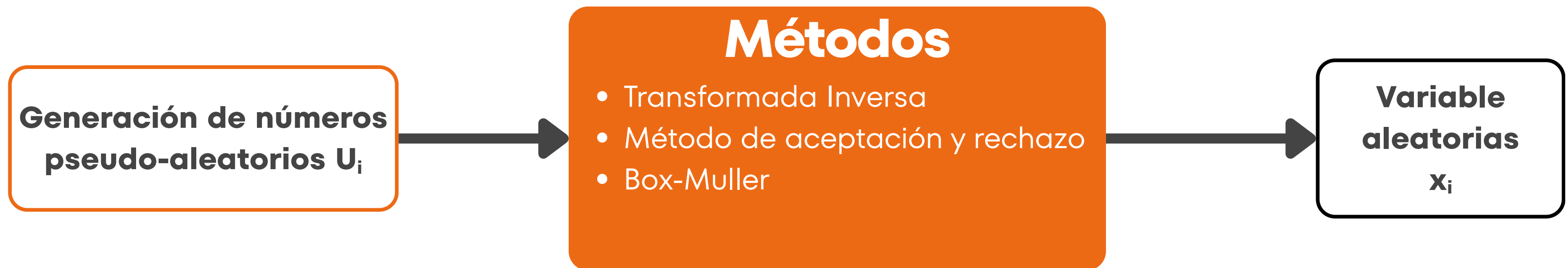
Generación de números pseudo-aleatorios en Python U(0,1)

Construir y comparar 4 generadores con MCL para $N = 100.000$

Generador	Tipo	a	c	m
Multiplicativo bueno	multiplicativo	16807	0	$2^{31} - 1$
Multiplicativo malo	multiplicativo	1001	0	2^{14}
Mixto bueno	mixto	1664525	1013904223	2^{32}
Mixto malo	mixto	1001	1	2048

Generación de variables aleatorias

Los números pseudo-aleatorios son el ingrediente básico en la simulación. Los lenguajes de simulación generan números pseudo-aleatorios que son usados para generar **variables aleatorias**.



Transformada Inversa

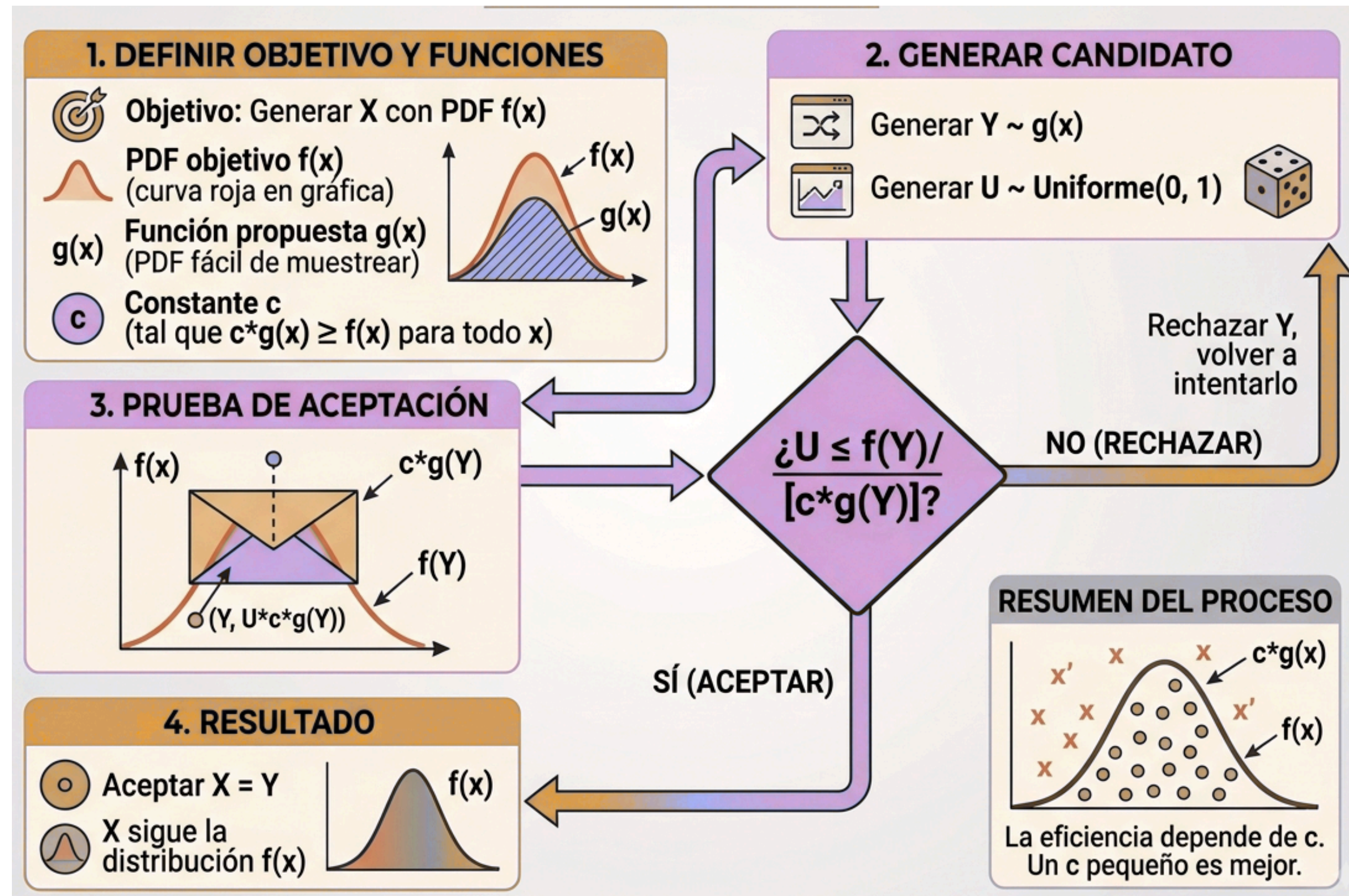
1. Calcular la Función Acumulada de Probabilidad:

2. Generar un número pseudoaleatorio $\mathbf{u_i}$ y evaluarlo en la Función Acumulada de Probabilidad Inversa $\mathbf{F_x^{-1}(u_i)}$

Dist. Triangular

Dist. Exponencial

Aceptación y rechazo



Aceptación y rechazo

Implementación de Transformada inversa y Aceptación y Rechazo en Python

Derivar e implementar el método de la **transformada inversa** para distribuciones continuas de uso común (Uniforme, Exponencial y Triangular) y el método de **aceptación y rechazo** para una densidad polinómica y una distribución Beta, validando cada implementación mediante histogramas contra la densidad teórica.