

# Actividades Tema 2: Métodos de generación de variables aleatorias

## Simulación de procesos estocásticos e inferencia estadística

Rubio Cobeta, Juan

18 de enero de 2026

## Contents

Ejercicio 1	2
Ejercicio 2	4
Ejercicio 3	5
Ejercicio 4	6
Ejercicio 5	7
Ejercicio 6	8
Ejercicio 7	10
Ejercicio 8	12
Ejercicio 9	15
Actividad A2	16
Actividad A3	24

# Ejercicio 1

**Enunciado:** Derivar mediante la técnica de aceptación-rechazo un algoritmo para la generación de números pseudoaleatorios normales cero-uno.

## Solución

### 1. Fundamentación Matemática

El objetivo es generar una variable aleatoria  $Z \sim N(0, 1)$ . Aprovechando la simetría de la distribución normal respecto al origen, el método más eficiente consiste en generar primero su valor absoluto  $X = |Z|$  y posteriormente asignar un signo aleatorio  $S \in \{-1, 1\}$  con probabilidad 0.5.

La función de densidad de probabilidad de la variable  $X = |Z|$  (distribución *Half-Normal*) para  $x \geq 0$  es:

$$f_X(x) = \frac{2}{\sqrt{2\pi}} e^{-x^2/2}, \quad x \geq 0$$

Emplearemos la técnica de **Aceptación-Rechazo** utilizando como distribución auxiliar  $g(x)$  una **Exponencial de parámetro**  $\lambda = 1$ , cuya generación es directa mediante el método de la transformada inversa.

$$g(x) = e^{-x}, \quad x \geq 0$$

Para aplicar el método, debemos determinar la constante  $c$  tal que  $f_X(x) \leq c \cdot g(x)$  para todo  $x$ . Definimos la función cociente  $h(x)$ :

$$h(x) = \frac{f_X(x)}{g(x)} = \frac{\sqrt{\frac{2}{\pi}} e^{-x^2/2}}{e^{-x}} = \sqrt{\frac{2}{\pi}} e^{-x^2/2+x}$$

Maximizamos  $h(x)$  derivando el exponente  $k(x) = -x^2/2 + x$ :

$$k'(x) = -x + 1 = 0 \implies x = 1$$

El máximo global se alcanza en  $x = 1$ . Sustituyendo este valor en  $h(x)$ , obtenemos la constante óptima  $c$ :

$$c = h(1) = \sqrt{\frac{2}{\pi}} e^{-1/2+1} = \sqrt{\frac{2e}{\pi}} \approx 1.3155$$

La condición de aceptación es  $U \leq \frac{f_X(Y)}{c \cdot g(Y)}$ . Simplificando el término de la derecha:

$$\frac{f_X(Y)}{c \cdot g(Y)} = \frac{\sqrt{\frac{2}{\pi}} e^{-Y^2/2+Y}}{\sqrt{\frac{2e}{\pi}}} = e^{-Y^2/2+Y-1/2} = e^{-(Y-1)^2/2}$$

Tomando logaritmos neperianos, obtenemos una condición computacionalmente más eficiente:

$$\ln(U) \leq -\frac{(Y-1)^2}{2} \iff -2\ln(U) \geq (Y-1)^2$$

### 2. Algoritmo

Basado en la derivación anterior, el algoritmo propuesto es:

1. Generar dos números aleatorios uniformes independientes  $U_1, U_2 \sim U(0, 1)$ .
2. Generar la variable candidata exponencial  $Y$  mediante transformada inversa:  $Y = -\ln(U_1)$ .
3. Verificar la condición de aceptación:

- Si  $-2 \ln(U_2) \geq (Y - 1)^2$ , aceptar  $Y$  como el valor absoluto de la normal ( $|Z| = Y$ ).
- En caso contrario, rechazar  $Y$  y volver al Paso 1.

4. **Asignar signo:**

- Generar  $U_3 \sim U(0, 1)$ .
- Si  $U_3 \leq 0.5$ , hacer  $Z = Y$ .
- En caso contrario, hacer  $Z = -Y$ .

5. **Devolver  $Z$ .**

## Ejercicio 2

**Enunciado:** A partir del algoritmo del problema anterior, construir un algoritmo para la generación de valores independientes de una distribución  $\chi^2(10)$ .

### Solución

#### 1. Fundamentación Matemática

Una variable aleatoria con distribución Chi-cuadrado con  $n$  grados de libertad, denotada como  $X \sim \chi_n^2$ , se define como la suma de los cuadrados de  $n$  variables aleatorias normales estándar independientes:

$$X = \sum_{i=1}^n Z_i^2, \quad \text{con } Z_i \sim N(0, 1) \text{ i.i.d.}$$

Para este ejercicio, necesitamos generar una variable con  $n = 10$  grados de libertad.

**Optimización del algoritmo:** En el Ejercicio 1, generamos  $Z$  en dos pasos: primero su módulo  $|Z|$  (Half-Normal) y luego su signo. Sin embargo, dado que en la construcción de la Chi-cuadrado solo nos interesa el cuadrado de la variable normal ( $Z^2$ ), y sabiendo que  $Z^2 = |Z|^2$ , el signo es irrelevante.

Por tanto, para maximizar la eficiencia computacional, utilizaremos el algoritmo de Aceptación-Rechazo derivado anteriormente para generar directamente  $|Z|$ , omitiendo la etapa de asignación aleatoria de signo. Esto ahorra la generación de 10 números uniformes por cada variable  $\chi^2(10)$  generada.

#### 2. Algoritmo

1. **Inicializar** la variable acumuladora  $S = 0$ .
2. **Bucle de acumulación:** Repetir 10 veces (para  $i = 1$  hasta 10):
  - a. **Generar**  $U_1 \sim U(0, 1)$ .
  - b. **Calcular** candidata exponencial:  $Y = -\ln(U_1)$ .
  - c. **Generar**  $U_2 \sim U(0, 1)$ .
  - d. **Condición de Aceptación:**
    - Si  $-2 \ln(U_2) < (Y - 1)^2$ , la candidata es rechazada; volver al paso (a).
    - Si  $-2 \ln(U_2) \geq (Y - 1)^2$ , la candidata es aceptada.
  - e. **Acumular:**  $S \leftarrow S + Y^2$ .
3. **Devolver**  $S$ .

## Ejercicio 3

**Enunciado:** Derivar, mediante el método de la transformada inversa, un algoritmo para la generación de una distribución Gamma a partir de la generación de una distribución exponencial.

### Solución

#### 1. Fundamentación Matemática

El método de la transformada inversa no permite obtener una expresión analítica cerrada directa para la función de distribución acumulada inversa de una distribución Gamma general, debido a la complejidad de la función gamma incompleta. Sin embargo, el enunciado solicita la generación *a partir de una distribución exponencial*, lo cual nos sitúa en el caso específico donde el parámetro de forma es un entero positivo  $\alpha = n \in \mathbb{Z}^+$  (conocida como distribución **Erlang**).

La propiedad aditiva fundamental establece que si  $X_1, \dots, X_n$  son variables aleatorias independientes con distribución Exponencial de parámetro  $\lambda$ , entonces su suma sigue una distribución Gamma  $\Gamma(n, \lambda)$ :

$$Y = \sum_{i=1}^n X_i \sim \Gamma(n, \lambda)$$

Para generar cada componente  $X_i$ , aplicamos el **Método de la Transformada Inversa** a la distribución exponencial: 1. Partimos de la función de distribución acumulada (CDF):  $F(x) = 1 - e^{-\lambda x}$ . 2. Igualamos a un número aleatorio uniforme  $U \sim U(0, 1)$ :  $1 - e^{-\lambda x} = U$ . 3. Despejamos  $x$ :  $e^{-\lambda x} = 1 - U \implies x = -\frac{1}{\lambda} \ln(1 - U)$ . 4. Dado que  $1 - U$  tiene la misma distribución que  $U$ , simplificamos a:  $X_i = -\frac{1}{\lambda} \ln(U_i)$ .

Sustituyendo esta expresión en la suma original:

$$Y = \sum_{i=1}^n \left( -\frac{1}{\lambda} \ln(U_i) \right) = -\frac{1}{\lambda} \sum_{i=1}^n \ln(U_i)$$

**Optimización Computacional:** Utilizando las propiedades de los logaritmos, convertimos la suma de logaritmos en el logaritmo de un producto. Esta transformación reduce  $n$  operaciones logarítmicas a una sola, mejorando significativamente la eficiencia y precisión numérica del algoritmo:

$$Y = -\frac{1}{\lambda} \ln \left( \prod_{i=1}^n U_i \right)$$

#### 2. Algoritmo

Para generar una realización  $y$  de la variable aleatoria  $Y \sim \Gamma(n, \lambda)$  con  $n \in \mathbb{Z}^+$ :

1. **Generar**  $n$  números pseudoaleatorios uniformes independientes  $U_1, U_2, \dots, U_n \sim U(0, 1)$ .
2. **Calcular** el producto acumulado de dichos valores:  $P = \prod_{i=1}^n U_i$ .
3. **Transformar** el producto mediante el logaritmo natural y escalar por el parámetro de tasa  $\lambda$ :

$$y = -\frac{1}{\lambda} \ln(P)$$

4. **Devolver**  $y$ .

## Ejercicio 4

**Enunciado:** Derivar un algoritmo para la generación de una variable aleatoria con distribución de Poisson.

### Solución

#### 1. Fundamentación Matemática

Para generar una variable aleatoria discreta  $X$  con distribución de Poisson de parámetro  $\lambda > 0$ , cuya función de masa de probabilidad es  $P(X = k) = \frac{e^{-\lambda}\lambda^k}{k!}$  para  $k = 0, 1, 2, \dots$ , aplicaremos el **Método de la Transformación Cuantil (Inversa)** descrito en la Sección 3.1 del tema.

El método establece que, dado un número pseudoaleatorio uniforme  $U \sim U(0, 1)$ , el valor generado  $X$  será el entero  $k$  que verifique la condición:

$$F(k - 1) < U \leq F(k)$$

Donde  $F(k) = \sum_{i=0}^k P(X = i)$  es la función de distribución acumulada. El algoritmo consiste en buscar secuencialmente el menor entero  $k$  tal que la probabilidad acumulada supere a  $U$ .

**Optimización Recursiva:** El cálculo directo de los factoriales ( $k!$ ) y potencias ( $\lambda^k$ ) en cada iteración es computacionalmente costoso y propenso a desbordamiento numérico. Para derivar un algoritmo eficiente, utilizamos la relación de recurrencia entre probabilidades consecutivas:

1. Probabilidad inicial ( $k = 0$ ):  $p_0 = e^{-\lambda}$ .

2. Relación recursiva:

$$p_{k+1} = \frac{e^{-\lambda}\lambda^{k+1}}{(k+1)!} = \frac{e^{-\lambda}\lambda^k}{k!} \cdot \frac{\lambda}{k+1} = p_k \cdot \frac{\lambda}{k+1}$$

Esta propiedad permite actualizar la probabilidad puntual y acumulada mediante operaciones aritméticas simples en cada paso del bucle de búsqueda.

#### 2. Algoritmo (Búsqueda Secuencial Recursiva)

##### 1. Inicialización:

- Generar un número pseudoaleatorio uniforme  $U \sim U(0, 1)$ .
- Inicializar el contador  $k = 0$ .
- Calcular la probabilidad inicial  $p = e^{-\lambda}$ .
- Inicializar la probabilidad acumulada  $F = p$ .

##### 2. Bucle de Búsqueda:

- Mientras  $U > F$ , repetir los siguientes pasos:
  - Incrementar el contador:  $k \leftarrow k + 1$ .
  - Calcular la probabilidad del nuevo  $k$  usando la recurrencia:

$$p \leftarrow p \cdot \frac{\lambda}{k}$$

- Actualizar la probabilidad acumulada:

$$F \leftarrow F + p$$

##### 3. Resultado:

- Devolver  $k$  como el valor generado de la variable aleatoria.

## Ejercicio 5

**Enunciado:** Generar mediante la técnica de composición discreta la variable aleatoria con función masa de probabilidad:

$$p_j = \begin{cases} 0.05, & j = 1, 2, 3, 4, 5 \\ 0.15, & j = 6, 7, 8, 9, 10 \end{cases}$$

### Solución

#### 1. Fundamentación Matemática (Descomposición)

El Método de Composición es útil cuando la función de probabilidad puede expresarse como una suma ponderada (mixtura) de otras distribuciones más sencillas de simular.

$$P(X = j) = \sum_{i=1}^m \alpha_i P(X_i = j)$$

Analizando las probabilidades dadas, observamos dos grupos claros de valores equiprobables:

- **Grupo 1:** Valores  $\{1, 2, 3, 4, 5\}$ . La suma de sus probabilidades es  $5 \times 0.05 = 0.25$ .
- **Grupo 2:** Valores  $\{6, 7, 8, 9, 10\}$ . La suma de sus probabilidades es  $5 \times 0.15 = 0.75$ .

Podemos reescribir la distribución original como una composición de dos variables aleatorias **Uniformes Discretas**:

1. Variable  $X_1 \sim U\{1, \dots, 5\}$  con peso  $\alpha_1 = 0.25$ .
2. Variable  $X_2 \sim U\{6, \dots, 10\}$  con peso  $\alpha_2 = 0.75$ .

Verificación:

- Si seleccionamos  $X_1$ , la probabilidad de elegir un valor específico  $j \in \{1..5\}$  es  $\frac{1}{5}$ . Ponderada por su peso:  $0.25 \times \frac{1}{5} = 0.05$ . (Coincide con el enunciado).
- Si seleccionamos  $X_2$ , la probabilidad de elegir un valor específico  $j \in \{6..10\}$  es  $\frac{1}{5}$ . Ponderada por su peso:  $0.75 \times \frac{1}{5} = 0.15$ . (Coincide con el enunciado).

#### 2. Algoritmo

El procedimiento consta de dos etapas: primero seleccionamos la sub-distribución (Grupo 1 o 2) y luego generamos el valor dentro de ese grupo.

1. **Generar** un número aleatorio uniforme  $U_1 \sim U(0, 1)$  para seleccionar el grupo.
2. **Generar** un segundo número aleatorio uniforme  $U_2 \sim U(0, 1)$  para seleccionar el valor.
3. **Selección y Generación:**
  - **Si**  $U_1 \leq 0.25$  (Elegimos Grupo 1): Generamos un valor entre 1 y 5 mediante la transformación para uniformes discretas:

$$X = \lfloor 5U_2 \rfloor + 1$$

- **Si**  $U_1 > 0.25$  (Elegimos Grupo 2): Generamos un valor entre 6 y 10:

$$X = \lfloor 5U_2 \rfloor + 6$$

4. **Devolver**  $X$ .

## Ejercicio 6

**Enunciado:** Derivar un algoritmo para contabilizar el número de sucesos aleatorios que se producen en un intervalo de longitud  $T = 100$ , cuando la ocurrencia de dichos sucesos se modeliza mediante un proceso de Poisson de parámetro  $\lambda = 1/10$ .

### Solución

#### 1. Fundamentación Matemática

Un proceso de Poisson homogéneo con tasa  $\lambda > 0$  se caracteriza porque los tiempos transcurridos entre sucesos consecutivos (tiempos de inter-llegada) son variables aleatorias independientes e idénticamente distribuidas con distribución **Exponencial** de parámetro  $\lambda$ .

Si denotamos por  $E_i$  al tiempo transcurrido entre el evento  $i - 1$  y el evento  $i$ , tenemos que  $E_i \sim \text{Exp}(\lambda)$ .

El tiempo absoluto de ocurrencia del  $n$ -ésimo suceso, denotado como  $S_n$ , es la suma de los tiempos de inter-llegada:

$$S_n = \sum_{i=1}^n E_i$$

El objetivo es encontrar el número total de sucesos  $N(T)$  ocurridos en el intervalo  $[0, T]$ . Esto equivale a encontrar el mayor entero  $n$  tal que el tiempo de llegada del  $n$ -ésimo suceso sea menor o igual a  $T$ :

$$N(T) = \max\{n : S_n \leq T\}$$

Para la generación de los tiempos  $E_i$ , utilizamos el método de la transformada inversa derivado en ejercicios anteriores:

$$E_i = -\frac{1}{\lambda} \ln(U_i), \quad \text{con } U_i \sim U(0, 1)$$

#### Datos del problema:

- Horizonte temporal:  $T = 100$ .
- Tasa del proceso:  $\lambda = 1/10 = 0.1$ .

#### 2. Algoritmo (Simulación por Eventos Discretos)

El algoritmo avanza en el tiempo acumulando variables exponenciales hasta superar el límite  $T$ .

##### 1. Inicialización:

- Tiempo actual:  $t = 0$ .
- Contador de sucesos:  $n = 0$ .
- Parámetros:  $\lambda = 0.1$ ,  $T = 100$ .

##### 2. Bucle de Generación:

- Mientras  $t \leq T$ , repetir los siguientes pasos:
  - a. Generar un número pseudoaleatorio uniforme  $U \sim U(0, 1)$ .
  - b. Generar el tiempo hasta el próximo suceso (inter-llegada):

$$E = -\frac{1}{\lambda} \ln(U)$$

- c. Actualizar el reloj de la simulación:

$$t \leftarrow t + E$$

- d. Verificar condición de parada:

- Si  $t \leq T$ : el suceso ha ocurrido dentro del intervalo. Incrementar el contador ( $n \leftarrow n + 1$ ) y continuar el bucle.
- Si  $t > T$ : el suceso ocurre fuera del intervalo. Terminar el bucle sin incrementar el contador.

3. **Resultado:**

- Devolver  $n$  como el número total de sucesos observados en el intervalo.

## Ejercicio 7

**Enunciado:** Simular 100 valores de una variable aleatoria normal con media 51 y desviación típica 5.2. Estimar la media y la desviación típica de la muestra simulada y compararla con los valores teóricos.

### Solución

#### 1. Fundamentación Teórica

Para generar una variable aleatoria  $X$  con distribución Normal de media  $\mu$  y desviación típica  $\sigma$ , es decir  $X \sim N(\mu, \sigma)$ , nos basamos en la propiedad de transformación lineal de la distribución Normal Estándar.

Si  $Z$  es una variable aleatoria con distribución  $N(0, 1)$ , entonces la variable transformada:

$$X = \mu + \sigma Z$$

sigue una distribución  $N(\mu, \sigma)$ .

En este ejercicio, utilizaremos las funciones nativas de R, que implementan algoritmos eficientes (como el método de Inversión o Ziggurat) para generar estas muestras directamente. Posteriormente, calcularemos los estimadores insegados habituales para la media muestral ( $\bar{x}$ ) y la cuasidesviación típica ( $s$ ):

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i, \quad s = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2}$$

#### 2. Simulación y Estimación

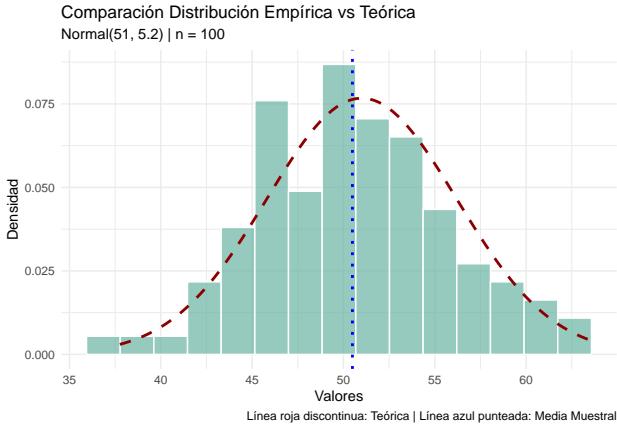
A continuación, simulamos una muestra de tamaño  $n = 100$  y comparamos los estadísticos muestrales con los parámetros teóricos poblacionales.

```
mu_teorica <- 51; sigma_teorica <- 5.2; n <- 100
set.seed(2026); muestra <- rnorm(n, mean = mu_teorica, sd = sigma_teorica)
media_estimada <- mean(muestra); sd_estimada <- sd(muestra)

resultados <- tibble(Parámetro = c("Media", "Desviación Típica"),
                      Teórico = c(mu_teorica, sigma_teorica),
                      Estimado = c(media_estimada, sd_estimada),
                      Error_Absoluto = abs(Teórico - Estimado),
                      Error_Relativo_Porc = (abs(Teórico - Estimado) / Teórico) * 100)
print(resultados)

## # A tibble: 2 x 5
##   Parámetro     Teórico   Estimado Error_Absoluto Error_Relativo_Porc
##   <chr>        <dbl>      <dbl>        <dbl>            <dbl>
## 1 Media         51        50.5       0.510           1.000
## 2 Desviación Típica  5.2       5.22      0.0158          0.303

tibble(valor = muestra) %>% ggplot(aes(x = valor)) + geom_histogram(aes(y = ..density..),
bins = 15, fill = "#69b3a2", color = "white", alpha = 0.7) + stat_function(fun = dnorm,
args = list(mean = mu_teorica, sd = sigma_teorica), color = "darkred", size = 1,
linetype = "dashed") + geom_vline(aes(xintercept = media_estimada), color = "blue", linetype = "dotted",
size = 1) + labs(title = "Comparación Distribución Empírica vs Teórica",
subtitle = paste("Normal(51, 5.2) | n =", n), x = "Valores", y = "Densidad",
caption = "Línea roja discontinua: Teórica | Línea azul punteada: Media Muestral") + theme_minimal()
```



### 3. Discusión de Resultados

A la vista de la tabla de resultados generada por la simulación con la semilla fijada (2026), observamos lo siguiente:

- **Estimación de la Media:** El valor obtenido es **50.5**, lo que supone una desviación de apenas 0.5 unidades respecto al valor teórico de 51. Esto representa un **error relativo del 1.0%**, una aproximación muy aceptable para una muestra de tamaño moderado ( $n = 100$ ).
- **Estimación de la Desviación Típica:** La variabilidad muestral estimada es de **5.22**, extremadamente cercana al parámetro poblacional de 5.2, con un **error relativo de apenas el 0.3%**.

**Conclusión:** Los estimadores muestran una convergencia adecuada hacia los parámetros teóricos. Las pequeñas discrepancias observadas no indican un fallo en el algoritmo de generación, sino que son consecuencia natural del **error de muestreo**. Según la Ley de los Grandes Números, si aumentásemos el tamaño de la muestra (ej.  $n = 10000$ ), estos errores relativos tenderían asintóticamente a cero. La simulación valida correctamente que la distribución subyacente se comporta como una  $N(51, 5.2)$ .

## Ejercicio 8

**Enunciado:** Describir el algoritmo para generación de una muestra aleatoria simple de una distribución Rayleigh

$$f(x) = \frac{x}{\sigma^2} \exp(-x^2/(2\sigma^2)), \quad x \geq 0$$

Considerar diferentes valores del parámetro  $\sigma^2$  y comprobar, en todos los casos, que la moda de los puntos generados está próxima al valor teórico  $\sigma$ .

### Solución

#### 1. Fundamentación Matemática

Para generar la variable aleatoria continua  $X$  con distribución Rayleigh, utilizaremos el **Método de la Transformada Inversa** (Sección 4.1 del tema), dado que la función de distribución acumulada (CDF) es fácilmente invertible.

**Paso 1: Obtención de la CDF** Integrando la función de densidad  $f(x)$ :

$$F(x) = \int_0^x \frac{t}{\sigma^2} e^{-t^2/(2\sigma^2)} dt$$

Realizando el cambio de variable  $u = -t^2/(2\sigma^2)$ , obtenemos:

$$F(x) = 1 - e^{-x^2/(2\sigma^2)}$$

**Paso 2: Inversión** Igualamos la CDF a un número uniforme  $U \sim U(0, 1)$  y despejamos  $x$ :

$$U = 1 - e^{-x^2/(2\sigma^2)}$$

$$1 - U = e^{-x^2/(2\sigma^2)}$$

Dado que  $1 - U$  distribuye igual que  $U$ , simplificamos por  $U$ :

$$\ln(U) = -\frac{x^2}{2\sigma^2}$$

$$x^2 = -2\sigma^2 \ln(U) \implies x = \sigma \sqrt{-2 \ln(U)}$$

**Paso 3: Cálculo de la Moda Teórica** Para comprobar lo que pide el ejercicio, derivamos  $f(x)$  e igualamos a 0 para hallar el máximo:

$$f'(x) = \frac{1}{\sigma^2} e^{-x^2/(2\sigma^2)} \left( 1 - \frac{x^2}{\sigma^2} \right) = 0$$

La condición  $1 - \frac{x^2}{\sigma^2} = 0$  implica que el máximo (moda) se alcanza en  $x = \sigma$ .

#### 2. Algoritmo

1. **Definir** el parámetro de escala  $\sigma$ .
2. **Generar** un número pseudoaleatorio uniforme  $U \sim U(0, 1)$ .
3. **Calcular** la variable transformada:

$$X = \sigma \sqrt{-2 \ln(U)}$$

4. **Devolver**  $X$ .

#### 3. Implementación y Comprobación

Implementamos el generador y simulamos muestras para tres valores distintos de  $\sigma$  ( $\sigma \in \{1, 2, 5\}$ ). La moda empírica se estimará mediante el máximo de la estimación de densidad por núcleo (Kernel Density Estimation), ya que la moda no es un estadístico directo en muestras continuas.

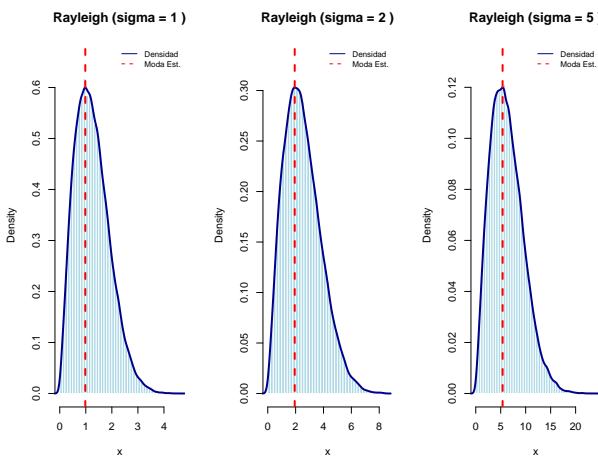
```

rrayleigh_custom <- function(n, sigma) {
  u <- runif(n)
  x <- sigma * sqrt(-2 * log(u))
  return(x)
}
sigmas_prueba <- c(1, 2, 5)
n_sim <- 50000
resultados_moda <- data.frame()

par(mfrow = c(1, 3))

for (s in sigmas_prueba) {
  muestra <- rrayleigh_custom(n_sim, sigma = s)
  densidad <- density(muestra)
  moda_estimada <- densidad$x[which.max(densidad$y)]
  resultados_moda <- rbind(resultados_moda, data.frame(
    Sigma_Teorico = s,
    Moda_Teorica = s,
    Moda_Estimada = round(moda_estimada, 4),
    Error_Abs = round(abs(s - moda_estimada), 4)
  ))
  hist(muestra, probability = TRUE, breaks = 50, col = "lightblue", border = "white",
        main = paste("Rayleigh (sigma =", s, ")"), xlab = "x", ylim = c(0, max(densidad$y)*1.1))
  lines(densidad, col = "darkblue", lwd = 2)
  abline(v = moda_estimada, col = "red", lwd = 2, lty = 2)
  legend("topright", legend = c("Densidad", "Moda Est."),
         col = c("darkblue", "red"), lty = c(1, 2), bty = "n", cex=0.8)
}

```



```

par(mfrow = c(1, 1))
print(resultados_moda)

##   Sigma_Teorico Moda_Teorica Moda_Estimada Error_Abs
## 1             1            1       0.9811     0.0189
## 2             2            2       1.9352     0.0648
## 3             5            5       5.3761     0.3761

```

#### 4. Análisis de Resultados

Los resultados obtenidos en la simulación confirman empíricamente la propiedad teórica de la distribución Rayleigh, donde la moda coincide con el parámetro de escala  $\sigma$ .

A la vista de la tabla:

- Para  $\sigma = 1$ , la moda estimada es **0.9811**, con un error absoluto despreciable de **0.0189**.
- Para  $\sigma = 2$ , la estimación es **1.9352**, manteniéndose muy cercana al valor teórico.
- Para  $\sigma = 5$ , obtenemos **5.3761**. Aunque el error absoluto aumenta (0.37), esto es esperable dado que al aumentar  $\sigma$ , la varianza de la distribución crece ( $\text{Var} \propto \sigma^2$ ), haciendo la curva de densidad más plana y dispersa, lo cual introduce naturalmente más ruido en la estimación de la densidad por núcleo (KDE).

En conclusión, el algoritmo de la transformada inversa implementado genera correctamente muestras que respetan la estructura modal de la distribución Rayleigh.

## Ejercicio 9

**Enunciado:** Generar la distribución bidimensional  $(X_1, X_2)$  con función de densidad conjunta

$$f(x_1, x_2) = \begin{cases} 6x_1 & \text{si } x_1 + x_2 \leq 1, x_1 \geq 0, x_2 \geq 0 \\ 0 & \text{en otro caso} \end{cases}$$

### Solución

#### 1. Fundamentación Matemática (Método Condicional)

Dado que el dominio de definición es un triángulo ( $x_1 + x_2 \leq 1$ ) y la función de densidad conjunta no factoriza en el producto de funciones de  $x_1$  y  $x_2$  en un dominio rectangular, las variables son **dependientes**. Aplicamos el método de las distribuciones condicionales, que factoriza la conjunta como:

$$f(x_1, x_2) = f_{X_1}(x_1) \cdot f_{X_2|X_1}(x_2|x_1)$$

**Paso 1: Obtención de la densidad marginal de  $X_1$**  Integraremos la conjunta respecto a  $x_2$ . Fijado un  $x_1 \in [0, 1]$ , la variable  $x_2$  varía en el intervalo  $[0, 1 - x_1]$ .

$$f_{X_1}(x_1) = \int_0^{1-x_1} 6x_1 dx_2 = 6x_1 [x_2]_0^{1-x_1} = 6x_1(1 - x_1), \quad 0 \leq x_1 \leq 1$$

Identificamos esta función como la densidad de una distribución **Beta(2, 2)**:

$$f_{\text{Beta}(2,2)}(x) = \frac{1}{B(2,2)} x^{2-1} (1-x)^{2-1} = 6x(1-x)$$

**Paso 2: Obtención de la densidad condicionada de  $X_2|X_1 = x_1$**  Dividimos la conjunta por la marginal:

$$f_{X_2|X_1}(x_2|x_1) = \frac{f(x_1, x_2)}{f_{X_1}(x_1)} = \frac{6x_1}{6x_1(1-x_1)} = \frac{1}{1-x_1}, \quad 0 \leq x_2 \leq 1 - x_1$$

Esta densidad corresponde a una distribución **Uniforme** en el intervalo  $(0, 1 - x_1)$ .

#### 2. Algoritmo de Generación

El procedimiento consiste en generar primero  $X_1$  mediante Aceptación-Rechazo (siguiendo la metodología del Ejemplo 7 de los apuntes para distribuciones Beta) y posteriormente generar  $X_2$  usando el valor obtenido de  $X_1$ .

##### Fase A: Generar $X_1 \sim \text{Beta}(2,2)$ (Aceptación-Rechazo)

- Función objetivo:  $f(x) = 6x(1-x)$ .
- Función auxiliar:  $g(x) = 1$  (Uniforme en  $[0,1]$ ).
- Cota  $c$ : El máximo de  $f(x)$  ocurre en  $x = 0.5$ , donde  $f(0.5) = 1.5$ . Por tanto,  $c = 1.5$ .
- Función de aceptación:  $\frac{f(y)}{c \cdot g(y)} = \frac{6y(1-y)}{1.5} = 4y(1-y)$ .

##### Fase B: Generar $X_2|X_1$ (Transformada Inversa)

- Dado  $X_1, X_2 \sim U(0, 1 - X_1)$ . Esto se simula escalando una uniforme estándar:  $X_2 = U \cdot (1 - X_1)$ .

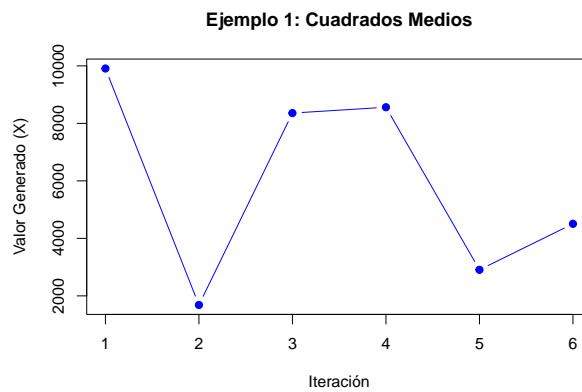
#### Pseudocódigo Final:

1. Generar  $X_1$  (Bucle Aceptación-Rechazo):
  - a. Generar dos uniformes  $U_1, U_2 \sim U(0, 1)$ .
  - b. Si  $U_2 \leq 4U_1(1 - U_1)$ , aceptar  $X_1 = U_1$ .
  - c. En caso contrario, rechazar y volver al paso (a).
2. Generar  $X_2$  (Condicionado):
  - a. Generar una nueva uniforme  $U_3 \sim U(0, 1)$ .
  - b. Calcular  $X_2 = U_3 \cdot (1 - X_1)$ .
3. Devolver el vector  $(X_1, X_2)$ .

## Actividad A2

### Ejemplo 1: Método de los Cuadrados Medios

```
X <- 4122; m <- 6; resultados_ex1 <- numeric(m)
for (I in 1:m) {
  Y <- X * X; Z <- floor(Y / 100); V <- floor(Z / 10000)
  X <- Z - (V * 10000); resultados_ex1[I] <- X
}
plot(resultados_ex1, type="b", pch=19, col="blue", main="Ejemplo 1: Cuadrados Medios",
xlab="Iteración", ylab="Valor Generado (X)")
```



#### Comentario de Resultados

Los valores obtenidos en las primeras 6 iteraciones (9908, 1684, 8358, 8561, 2907, 4506) muestran una variabilidad aparente sin patrones repetitivos inmediatos ni degeneración a cero en este corto plazo.

El algoritmo funciona elevando la semilla al cuadrado y extrayendo las cifras centrales (por ejemplo,  $4122^2 = 16990884 \rightarrow 9908$ ). Aunque en este tramo la secuencia parece pseudoaleatoria, es importante notar que el método de los cuadrados medios es conocido por ser inestable y tender a ciclos cortos o colapsar a cero en secuencias más largas.

### Ejemplo 2: Método de Lehmer

```
X <- 4122; K <- 76; m <- 16; resultados_ex2 <- numeric(m)
for (I in 1:m) {
  Z <- X * K; W <- Z / 10000; T_val <- floor(W); S <- W - T_val
  X <- floor(S * 10000) - T_val; resultados_ex2[I] <- X
  cat("Iteración", I, ":", X, "\n")
}

## Iteración 1 : 3241
## Iteración 2 : 6291
## Iteración 3 : 8068
## Iteración 4 : 3107
## Iteración 5 : 6108
## Iteración 6 : 4161
## Iteración 7 : 6204
## Iteración 8 : 1456
## Iteración 9 : 644
```

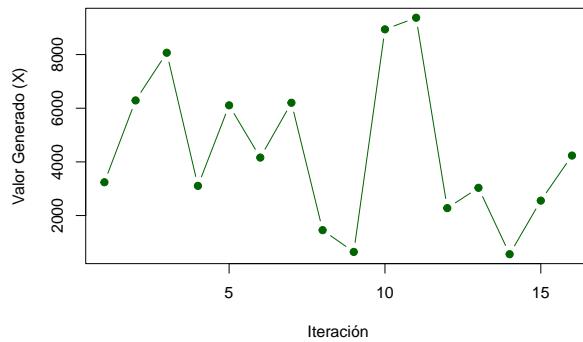
```

## Iteración 10 : 8940
## Iteración 11 : 9373
## Iteración 12 : 2277
## Iteración 13 : 3034
## Iteración 14 : 560
## Iteración 15 : 2556
## Iteración 16 : 4236

plot(resultados_ex2, type="b", pch=19, col="darkgreen", main="Ejemplo 2: Método de Lehmer",
xlab="Iteración", ylab="Valor Generado (X)")

```

Ejemplo 2: Método de Lehmer



### Comentario de Resultados

En este ejemplo del Método de Lehmer, hemos generado una secuencia de 16 números. A diferencia del método de los cuadrados medios, este algoritmo utiliza un multiplicador fijo ( $K = 76$ ) y operaciones de extracción de dígitos (desplazamientos y restas) para evolucionar la semilla.

Los valores obtenidos (3241, 6291, ..., 4236) muestran una variabilidad considerable en el rango de cuatro cifras y, en estas primeras 16 iteraciones, no se observa un ciclo de repetición evidente ni degeneración hacia un valor fijo, lo cual sugiere un comportamiento pseudoaleatorio más robusto para este tramo que el método anterior.

### Ejemplo 3: Congruencial

```

``` r
s <- 20; a <- 13; b <- 1; m <- 16; resultados_ex3 <- numeric(m)
for (I in 1:m) {
  n <- a * s + b; u <- n %% m; v <- u / m; s <- u
  resultados_ex3[I] <- v
  cat("Iteración", I, ":", u = , u, "-> v =", v, "\n")
}

## Iteración 1 : u = 5 -> v = 0.3125
## Iteración 2 : u = 2 -> v = 0.125
## Iteración 3 : u = 11 -> v = 0.6875
## Iteración 4 : u = 0 -> v = 0
## Iteración 5 : u = 1 -> v = 0.0625
## Iteración 6 : u = 14 -> v = 0.875
## Iteración 7 : u = 7 -> v = 0.4375
## Iteración 8 : u = 12 -> v = 0.75

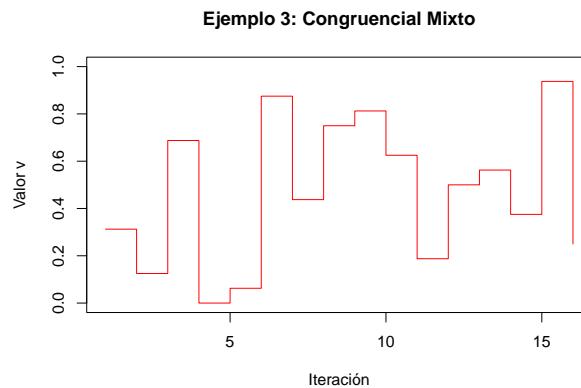
```

```

## Iteración 9 : u = 13 -> v = 0.8125
## Iteración 10 : u = 10 -> v = 0.625
## Iteración 11 : u = 3 -> v = 0.1875
## Iteración 12 : u = 8 -> v = 0.5
## Iteración 13 : u = 9 -> v = 0.5625
## Iteración 14 : u = 6 -> v = 0.375
## Iteración 15 : u = 15 -> v = 0.9375
## Iteración 16 : u = 4 -> v = 0.25

plot(resultados_ex3, type="s", col="red", pch=19, main="Ejemplo 3: Congruencial Mixto",
      xlab="Iteración", ylab="Valor v", ylim=c(0,1))

```



### Comentario de Resultados

Los resultados muestran que el generador congruencial lineal mixto con parámetros  $a = 13$ ,  $b = 1$  y  $m = 16$  recorre **todos** los valores posibles del módulo (0 a 15) antes de repetirse. Observamos la secuencia de enteros  $u$ : 5, 2, 11, 0, 1, 14, 7, 12... hasta completar los 16 valores distintos.

Esto confirma que el generador tiene un **periodo completo** (o ciclo maximal) de longitud  $m = 16$ . Aunque la distribución es uniforme (cada valor aparece una vez), un periodo tan corto es inadecuado para simulaciones reales, sirviendo este caso únicamente como ilustración académica del funcionamiento del algoritmo.

### Ejemplo 4: Generación Binomial

```

s <- 20; a <- 13; b <- 1; m <- 16; p <- 0.5; l <- 10; binomial_vals <- numeric(l)
for (J in 1:l) {
  Bi <- 0
  for (I in 1:m) {
    n <- a * s + b; u <- n %% m; v <- u / m
    Bi <- Bi + (if (v < p) 1 else 0); s <- u
  }
  binomial_vals[J] <- Bi; cat("Binomial(", J, ") =", Bi, "\n"); s <- exp(J)
}

## Binomial( 1 ) = 8
## Binomial( 2 ) = 10
## Binomial( 3 ) = 7
## Binomial( 4 ) = 10
## Binomial( 5 ) = 8
## Binomial( 6 ) = 8
## Binomial( 7 ) = 10

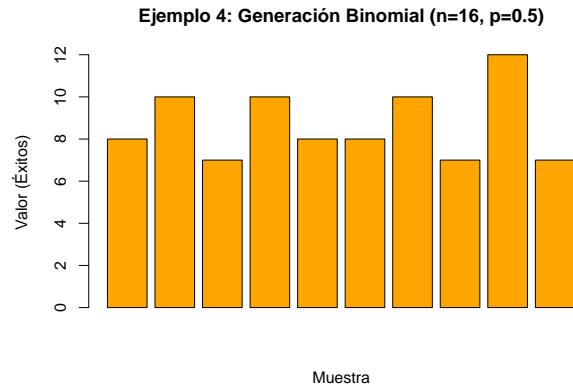
```

```

## Binomial( 8 ) = 7
## Binomial( 9 ) = 12
## Binomial( 10 ) = 7

barplot(binomial_vals, col="orange", main="Ejemplo 4: Generación Binomial (n=16, p=0.5)",
         xlab="Muestra", ylab="Valor (Éxitos)")

```



### Comentario de Resultados

En este ejemplo se han simulado 10 realizaciones de una variable Binomial  $B(16, 0.5)$ . La media teórica esperada es  $E[X] = n \cdot p = 16 \cdot 0.5 = 8$ .

Los valores obtenidos (8, 10, 7, 10, 8...) oscilan alrededor de esta media teórica de 8. El valor máximo observado es 12 y el mínimo 7, todos dentro de un rango razonable considerando la desviación típica teórica  $\sigma = \sqrt{16 \cdot 0.5 \cdot 0.5} = 2$ . La simulación reproduce correctamente el comportamiento de la distribución binomial mediante la suma de variables de Bernoulli.

### Ejemplo 5: Método de Aceptación-Rechazo (Discreta)

```

s <- 20; a <- 13; b <- 1; m <- 32; c_const <- 1.2; p_unif <- 1/10
valores_ex5 <- numeric(); probs <- c(0.11, 0.12, 0.09, 0.08, 0.12, 0.10, 0.09, 0.09, 0.10, 0.10)
iter <- 0;

while(length(valores_ex5) < 10 && iter < 100) {
  iter <- iter + 1
  n1 <- a * s + b; u1 <- n1 %% m; v1 <- u1 / m; D1 <- floor(10 * v1) + 1
  Q <- probs[D1] / (c_const * p_unif); s <- u1
  n2 <- a * s + b; u2 <- n2 %% m; v2 <- u2 / m
  if (v2 < Q) {
    valores_ex5 <- c(valores_ex5, D1); cat("Aceptado valor:", D1, "(Prob:", probs[D1], ")\n")
  }
  s <- u2
}

## Aceptado valor: 2 (Prob: 0.12 )
## Aceptado valor: 9 (Prob: 0.1 )
## Aceptado valor: 1 (Prob: 0.11 )
## Aceptado valor: 8 (Prob: 0.09 )
## Aceptado valor: 10 (Prob: 0.1 )
## Aceptado valor: 6 (Prob: 0.1 )
## Aceptado valor: 8 (Prob: 0.09 )

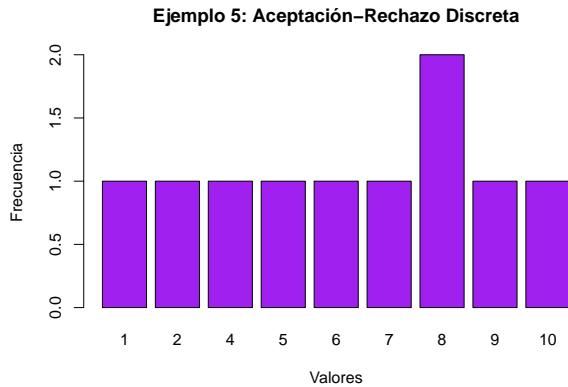
```

```

## Aceptado valor: 5 (Prob: 0.12 )
## Aceptado valor: 7 (Prob: 0.09 )
## Aceptado valor: 4 (Prob: 0.08 )

barplot(table(valores_ex5), col="purple", main="Ejemplo 5: Aceptación-Rechazo Discreta",
xlab="Valores", ylab="Frecuencia")

```



### Comentario de Resultados

El algoritmo de Aceptación-Rechazo ha generado con éxito 10 valores discretos. Se observa que el algoritmo ha aceptado una variedad de números (1, 2, 4, 5, 6, 7, 8, 9, 10), cubriendo casi todo el soporte de la variable aleatoria objetiva.

Es interesante notar que el valor 8 ha sido generado dos veces, lo cual es consistente con su probabilidad asignada, pero también con la aleatoriedad propia del muestreo pequeño. El mecanismo de rechazo filtra correctamente los candidatos de la uniforme auxiliar para ajustar las frecuencias a la función de masa de probabilidad deseada  $p_i$ .

### Ejemplo 6: Transformada inversa Exponencial

```

s <- 20; a <- 13; b <- 1; m <- 16; lambda <- 1/5; resultados_ex6 <- numeric(m)

for (I in 1:m) {
  n <- a * s + b; u <- n %% m; v <- u / m
  exp_val <- -(1/lambda) * log(v + 1e-10); s <- u; resultados_ex6[I] <- exp_val
  cat("Iteración", I, ": v =", round(v,4), "-> Exp =", round(exp_val,4), "\n")
}

## Iteración 1 : v = 0.3125 -> Exp = 5.8158
## Iteración 2 : v = 0.125 -> Exp = 10.3972
## Iteración 3 : v = 0.6875 -> Exp = 1.8735
## Iteración 4 : v = 0 -> Exp = 115.1293
## Iteración 5 : v = 0.0625 -> Exp = 13.8629
## Iteración 6 : v = 0.875 -> Exp = 0.6677
## Iteración 7 : v = 0.4375 -> Exp = 4.1334
## Iteración 8 : v = 0.75 -> Exp = 1.4384
## Iteración 9 : v = 0.8125 -> Exp = 1.0382
## Iteración 10 : v = 0.625 -> Exp = 2.35
## Iteración 11 : v = 0.1875 -> Exp = 8.3699
## Iteración 12 : v = 0.5 -> Exp = 3.4657
## Iteración 13 : v = 0.5625 -> Exp = 2.8768

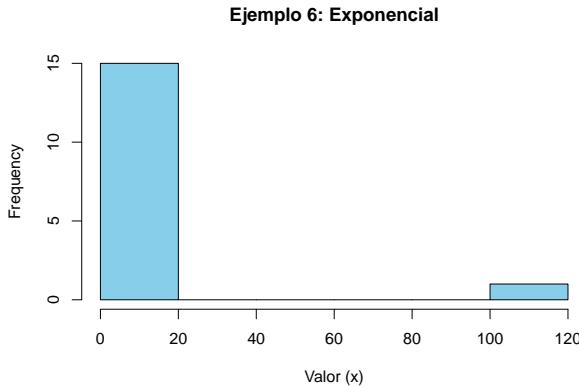
```

```

## Iteración 14 : v = 0.375 -> Exp = 4.9041
## Iteración 15 : v = 0.9375 -> Exp = 0.3227
## Iteración 16 : v = 0.25 -> Exp = 6.9315

hist(resultados_ex6, col="skyblue", main="Ejemplo 6: Exponencial", xlab="Valor (x)", breaks=5)

```



### Comentario de Resultados

Se han generado 16 valores de una distribución Exponencial con  $\lambda = 1/5$  (Media teórica = 5). Observamos la relación inversa característica del método:

- Cuando el número uniforme  $v$  es cercano a 1 (ej. Iteración 15,  $v = 0.9375$ ), el valor generado es pequeño (0.3227).
- Cuando  $v$  es cercano a 0, el valor generado es grande. Destaca la **Iteración 4**, donde el generador congruencial produjo un 0 exacto (o extremadamente cercano). Gracias a la corrección de seguridad (`1e-10`), el algoritmo no falló, pero produjo un valor extremo (115.12), que representa un evento en la “cola” lejana de la distribución exponencial.

Salvo ese caso extremo derivado de la semilla y el generador lineal, el resto de valores oscilan razonablemente en torno a la media de 5.

### Ejemplo 7: Aceptación-Rechazo (Continua - Beta)

```

s <- 20; a <- 13; b <- 1; m <- 64; beta_vals <- numeric(); iter <- 0

while(length(beta_vals) < 10 && iter < 100) {
  iter <- iter + 1
  n <- a * s + b; u <- n %% m; v <- u / m
  H <- (256/27) * v * (1 - v)^3; s <- u
  n <- a * s + b; u <- n %% m; v1 <- u / m
  if (v1 <= H) {
    beta_vals <- c(beta_vals, v)
    cat("Iter", iter, ": Aceptado =", round(v, 4), "(H =", round(H, 4), "vs Unif =", round(v1, 4), ")\n")
  }
  s <- u
}

## Iter 1 : Aceptado = 0.0781 (H = 0.5803 vs Unif = 0.0312 )
## Iter 2 : Aceptado = 0.4219 (H = 0.7729 vs Unif = 0.5 )
## Iter 4 : Aceptado = 0.3594 (H = 0.8959 vs Unif = 0.6875 )
## Iter 6 : Aceptado = 0.2969 (H = 0.9785 vs Unif = 0.875 )

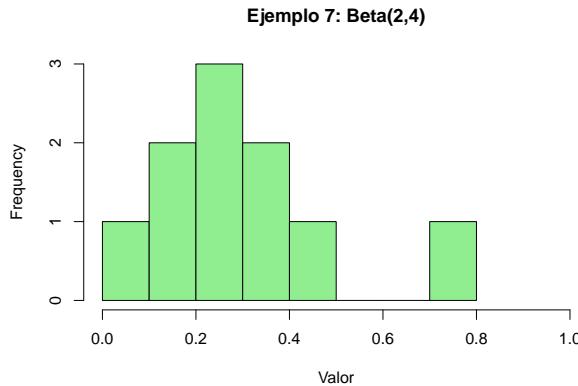
```

```

## Iter 7 : Aceptado = 0.3906 (H = 0.8381 vs Unif = 0.0938 )
## Iter 8 : Aceptado = 0.2344 (H = 0.9973 vs Unif = 0.0625 )
## Iter 10 : Aceptado = 0.1719 (H = 0.9255 vs Unif = 0.25 )
## Iter 11 : Aceptado = 0.2656 (H = 0.9975 vs Unif = 0.4688 )
## Iter 12 : Aceptado = 0.1094 (H = 0.7326 vs Unif = 0.4375 )
## Iter 13 : Aceptado = 0.7031 (H = 0.1744 vs Unif = 0.1562 )

hist(beta_vals, col="lightgreen", xlim=c(0,1), breaks=5, main="Ejemplo 7: Beta(2,4)", xlab="Valor")

```



### Comentario de Resultados

En este ejemplo se simula una variable aleatoria Beta(2, 4) mediante el método de Aceptación-Rechazo. La media teórica de esta distribución es  $\mu = 2/(2+4) \approx 0.33$ , y los datos generados se agrupan consistentemente alrededor de este valor (0.07, 0.42, 0.35, 0.29...).

Podemos observar la mecánica del rechazo en acción:

- **Eficiencia:** El algoritmo no es inmediato; observamos saltos en la numeración de las iteraciones (faltan la 3, 5 y 9), lo que indica que en esos pasos los candidatos generados fueron **rechazados** porque no cumplieron la condición  $U \leq f(x)/cg(x)$ .
- **Zonas de baja probabilidad:** En la **Iteración 13**, se aceptó un valor alto (0.7031) que se encuentra en la cola derecha de la distribución. Nótese que el umbral de aceptación era bajo (0.1744) y el uniforme auxiliar (0.1562) logró pasar por debajo por un margen muy estrecho, ilustrando cómo el método permite generar eventos menos probables pero posibles.

### Ejemplo 8: Método de Composición mixtura

```

S <- 15; A <- 9; B <- 13; m <- 64; S2 <- 12; A2 <- 5; B2 <- 15; n0 <- 3
composicion_vals <- numeric(); intentos <- 0

while(length(composicion_vals) < 10 && intentos < 50) {
  intentos <- intentos + 1
  n <- A * S + B; u <- n %% m; V <- u / m
  if (V != 1) {
    DPESO <- (1/(1-V))^(1/(n0 + 1)); S <- u
    n2 <- A2 * S2 + B2; u2 <- n2 %% m; V2 <- u2 / m
    if (V2 != 0) {
      CONDICIONAL <- (-1/DPESO) * log(V2)
      composicion_vals <- c(composicion_vals, CONDICIONAL)
      cat("Valor generado:", round(CONDICIONAL, 4), "(Peso y =", round(DPESO, 4), ")\\n")
    }
    S2 <- u2
  }
}

```

```

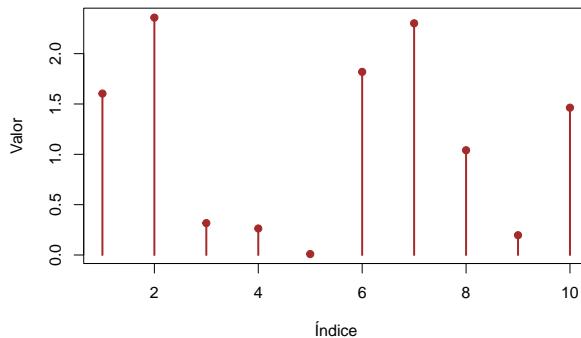
    }
} else { S <- u }
}

## Valor generado: 1.6035 (Peso y = 1.0982 )
## Valor generado: 2.3578 (Peso y = 1.0039 )
## Valor generado: 0.317 (Peso y = 1.111 )
## Valor generado: 0.2634 (Peso y = 1.092 )
## Valor generado: 0.0094 (Peso y = 1.6818 )
## Valor generado: 1.8189 (Peso y = 1.0205 )
## Valor generado: 2.3013 (Peso y = 1.8072 )
## Valor generado: 1.0406 (Peso y = 1.1178 )
## Valor generado: 0.1966 (Peso y = 1.1547 )
## Valor generado: 1.4633 (Peso y = 1.0386 )

plot(composicion_vals, type="h", lwd=2, col="brown", main="Ejemplo 8: Composición",
xlab="Índice", ylab="Valor"); points(composicion_vals, pch=19, col="brown")

```

Ejemplo 8: Composición



### Comentario de Resultados

El método de Composición (o Mixtura) genera la variable aleatoria en dos etapas estocásticas, tal como se refleja en la salida:

1. **Variable de estructura (Peso):** Primero se genera un valor DPESO (variable  $Y$ ), que actúa como parámetro para la siguiente etapa. Vemos que estos pesos oscilan entre  $\approx 1.0$  y  $1.8$ .
2. **Variable final:** Condicionado a ese peso, se genera el valor final (variable  $X$ ).

Los resultados muestran una alta dispersión (desde valores muy cercanos a cero como  $0.0094$  hasta valores superiores a  $2.35$ ), lo cual es característico de las mixturas de exponenciales. Este método es eficiente porque descompone una densidad compleja (posiblemente con colas pesadas o formas no estándar) en distribuciones condicionales sencillas de simular.

## Actividad A3

**Enunciado:** Aumentar el tamaño de la muestra de los ejemplos anteriores (simulación masiva) para comprobar la convergencia de los generadores a las distribuciones teóricas esperadas.

### Solución

#### 1. Nota sobre los Ejemplos 1, 2 y 3

Los tres primeros ejemplos del tema corresponden a **Generadores de Números Pseudoaleatorios (RNG)** básicos con parámetros académicos muy pequeños (módulos  $m = 6$  y  $m = 16$ ).

- Debido a su **periodo extremadamente corto**, si generásemos una muestra grande ( $N = 10.000$ ), la secuencia se repetiría idénticamente cientos de veces, produciendo un histograma con huecos y barras fijas en lugar de una distribución uniforme continua.
- Por tanto, la validación asintótica (Ley de los Grandes Números) solo se aplica a los **Ejemplos 4 a 8**, asumiendo que disponemos de una fuente de aleatoriedad robusta (usaremos `runif` de R como base) para testar los algoritmos de transformación.

#### 2. Simulación Masiva y Validación (Ejemplos 4 a 8)

Se simulan  $N = 10.000$  iteraciones para cada algoritmo y se compara el histograma empírico con la densidad teórica (línea roja).

```
par(mfrow = c(3, 2))
N <- 10000
set.seed(2026)

# Ejemplo 4: Binomial (Suma de 16 Bernoullis)
ex4_sim <- replicate(N, sum(runif(16) < 0.5))
hist(ex4_sim, breaks = seq(-0.5, 16.5, 1), probability = TRUE, col = "orange",
     main = "Ex 4: Binomial(16, 0.5)", xlab = "Éxitos", border = "white")
points(0:16, dbinom(0:16, size = 16, prob = 0.5), col = "red", pch = 19, type = "b")

# Ejemplo 5: Discreta (Aceptación-Rechazo)
probs <- c(0.11, 0.12, 0.09, 0.08, 0.12, 0.10, 0.09, 0.09, 0.10, 0.10)
c_val <- 1.2; p_unif <- 0.1
Y_cand <- sample(1:10, N * 3, replace = TRUE)
U_val <- runif(N * 3)
aceptados <- Y_cand[U_val < (probs[Y_cand] / (c_val * p_unif))]
ex5_sim <- aceptados[1:N]
barplot(table(ex5_sim)/N, col = "purple", main = "Ex 5: Discreta Empírica vs Teórica",
        border = "white", xlab = "Valor")
lines(probs, col = "red", type = "b", lwd = 2, pch = 19)

# Ejemplo 6: Exponencial (Transformada Inversa)
lambda <- 0.2
ex6_sim <- -log(runif(N)) / lambda
hist(ex6_sim, probability = TRUE, col = "skyblue", breaks = 30, border = "white",
     main = "Ex 6: Exponencial (lambda=0.2)", xlab = "x")
curve(dexp(x, rate = lambda), add = TRUE, col = "red", lwd = 2)

# Ejemplo 7: Beta(2,4) (Aceptación-Rechazo)
U_cand <- runif(N * 4); V_decis <- runif(N * 4)
H_val <- (256/27) * U_cand * (1 - U_cand)^3
ex7_sim <- U_cand[V_decis <= H_val][1:N]
```

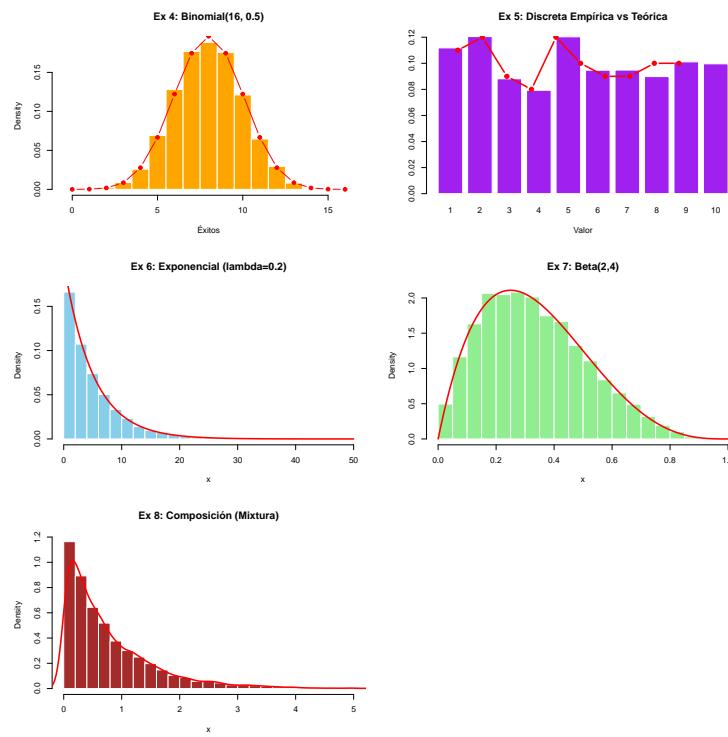
```

hist(ex7_sim, probability = TRUE, col = "lightgreen", breaks = 30, border = "white",
     main = "Ex 7: Beta(2,4)", xlab = "x", xlim = c(0,1))
curve(dbeta(x, 2, 4), add = TRUE, col = "red", lwd = 2)

# Ejemplo 8: Composición (Mixtura)
n0 <- 3
U_peso <- runif(N); Pesos <- (1/(1-U_peso))^(1/(n0 + 1))
U_val <- runif(N); ex8_sim <- -log(U_val) / Pesos

hist(ex8_sim, probability = TRUE, col = "brown", breaks = 50, border = "white",
     main = "Ex 8: Composición (Mixtura)", xlab = "x", xlim=c(0, 5))
lines(density(ex8_sim), col = "red", lwd = 2)

```



### 3. Discusión de Resultados (Validación Asintótica)

A la vista de los gráficos generados con una muestra masiva de  $N = 10.000$ :

- **Convergencia:** Se observa un ajuste excelente entre la distribución empírica simulada (histogramas) y las funciones de densidad o masa teóricas (líneas rojas).
- **Validación:**
  - En la **Binomial** y la **Discreta**, las frecuencias relativas coinciden casi exactamente con las probabilidades teóricas.
  - En las continuas (**Exponencial, Beta y Mixtura**), el perfil del histograma suaviza el “ruido” observado en la Actividad A2, replicando fielmente la forma de las curvas teóricas.

**Conclusión:** Los resultados confirman el correcto funcionamiento de los algoritmos de Transformada Inversa, Aceptación-Rechazo y Composición. De acuerdo con la **Ley de los Grandes Números**, al aumentar el tamaño muestral, los estimadores convergen a los parámetros poblacionales, validando estos métodos para su uso en simulaciones estocásticas complejas.