# EST-24107: Simulación

**Profesor**: Alfredo Garbuno Iñigo — Otoño, 2023 — Números no uniformes.
**Objetivo**: En esta sección del curso veremos métodos para generar números aleatorios de distribuciones un poco mas generales que una distribución uniforme. En particular veremos algunos ejemplos de distribuciones continuas, discretas y mezclas. Además veremos una prueba de uniformidad adicional a la prueba Kolmogorov-Smirnov.
**Lectura recomendada**: Capítulo 2 de [1]. El capítulo 3 de [2] presenta los métodos para generar números aleatorios gaussianos.

## 1. Generación de variables no uniformes

R, por ejemplo, tiene distintos generadores de variables aleatorias. Un catálogo nos muestra que podemos generar mucho mas variables que solamente una variable uniforme.

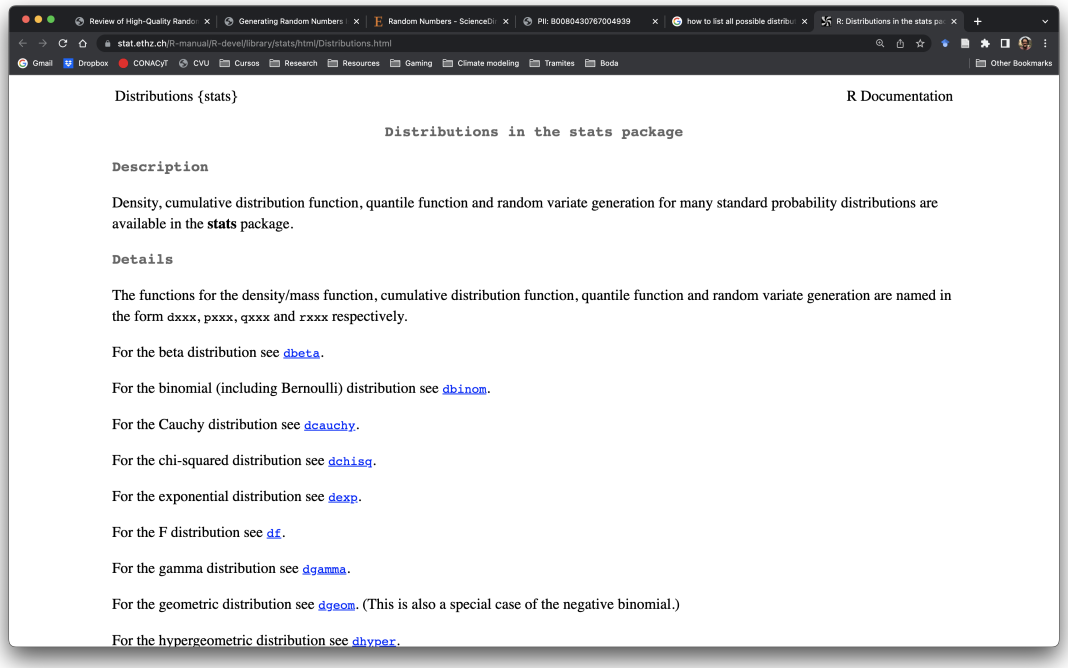



FIGURA 1. *Catálogo de distribuciones en R.*

En esta sección exploraremos los mecanismos tradicionales para generar números pseudo-aleatorios de distribuciones un poco mas generales que una distribución uniforme.

### 1.1. Método de la transformada inversa

De su curso de cálculo de probabilidades se acordarán que si tenemos una variable aleatoria $X$ y una función de acumulación $\mathbb{P}$. Si utilizamos $U = \mathbb{P}(X)$, con $U$ una variable aleatoria con distribución $\mathsf{U}(0, 1)$. Entonces, $X \sim \mathbb{P}$.

**Pregunta.** Considera $X \sim \mathsf{Exp}(1)$, de tal forma que $\mathbb{P}(x) = 1 - e^{-x}$. Si sólo sabemos generar números uniformes, ¿cómo generarías números de $X$?

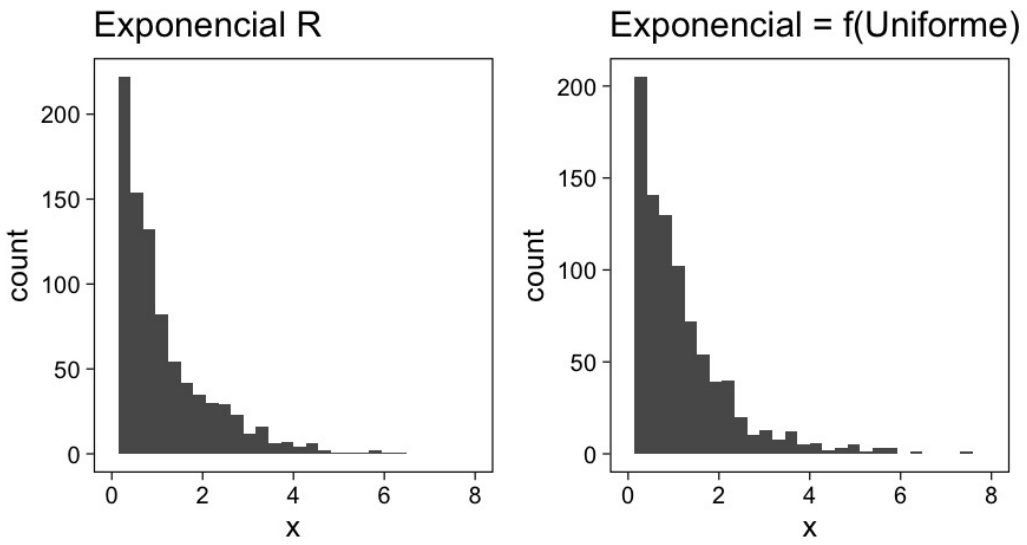



FIGURA 2. *Histogramas de simulaciones utilizando el método directo (izquierda) contra el de la transformada inversa (derecha).*

**Pregunta.** Considera las siguientes distribuciones:

1. Logística con $\mu, \beta$ y función de acumulación

$$\mathbb{P}(x) = \frac{1}{1 + e^{-(x-\mu)/\beta}}\,. \tag{1}$$

2. Cauchy con parámetros $\mu, \beta$ y función de acumulación

$$\mathbb{P}(x) = \frac{1}{2} + \frac{1}{\pi}\mathsf{arctan}((x-\mu)/\beta)\,. \tag{2}$$

Genera números aleatorios con valores $\mu = 1$, $\beta = 2$ y comenta el rol de cada parámetro en cada distribución.

### 1.2. Generalización

En el caso anterior asumimos que la función inversa de la función de acumulación existe. Sin embargo, no siempre es el caso. Necesitamos definir lo siguiente.

**Definición 1.1** (Función inversa generalizada)**.** Sea $X$ una variable aleatoria con función de acumulación $\mathbb{P}$. La función inversa generalizada está definida como

$$\mathbb{P}^{-1}(u) = \text{ínf}\{x | F(x) \geq u\}\,. \tag{3}$$

**Teorema 1.2** (De la Función Inversa)**.** Sea $\mathbb{P}$ una función de distribución. Sea $\mathbb{P}^{-1}(\cdot)$ la función inversa generalizada de $\mathbb{P}$ y $U \sim \mathsf{U}(0,1)$. Entonces, la variable aleatoria $X = \mathbb{P}^{-1}(U)$ tiene distribución $\mathbb{P}$.

## 2. Distribuciones continuas

Hay situaciones donde generar números aleatorios puede ser extremadamente sencillo pues existe una relación entre la distribución que queremos generar y ciertos componentes fáciles de simular.

### 2.1. Variables $\chi^2$

Recordarán de su curso de cálculo de probabilidades que si $X_i \sim \mathsf{Exp}(1)$ entonces

$$Y = 2\sum_{j=1}^{\nu} X_j \sim \chi^2_{2\nu}\,, \tag{4}$$

con $\nu \in \mathbb{N}$.

### 2.2. Variables gamma

También recordarán que si $X_i \sim \mathsf{Exp}(1)$ entonces

$$Y = \beta\sum_{j=1}^{\alpha} X_j \sim \mathsf{Gamma}(\alpha,\beta)\,, \tag{5}$$

con $\alpha \in \mathbb{N}$.

### 2.3. Variables beta

También recordarán que si $X_i \sim \mathsf{Exp}(1)$ entonces

$$Y = \frac{\sum_{j=1}^{\alpha} X_j}{\sum_{j=1}^{\alpha+\beta} X_j} \sim \mathsf{Beta}(\alpha,\beta)\,, \tag{6}$$

con $\alpha, \beta \in \mathbb{N}$.

**Pregunta.** Prueba las igualdades anteriores.

#### 2.3.1. Código:

Por ejemplo, podemos utilizar el siguiente código para generar variables de una $\chi^2_6$ .

```
set.seed(108)
U ← runif(3 * 10^4)       # Genera uniformes
U ← matrix(U, nrow = 3)   # Transforma a matriz
X ← -log(U)               # Transforma a exponenciales
X ← 2 * apply(X, 2, sum) # Suma los tres renglones
summary(X)
```

```
 Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.151   3.505   5.420   6.064   7.926  27.904
```

A partir de la versión 4.1.1 `R` cuenta con un operador especial (`|>`) llamado `pipe` el cual permite *anidar* ciertas funciones y evitar la asignación repetitiva de variables.

```
set.seed(108)
runif(3 * 10^4) ▷          # Genera uniformes
  matrix(nrow = 3) ▷       # Transforma a matriz
  log() ▷                  # Calcula logaritmos
  apply(2, function(x){-2 * sum(x)} ) ▷
  summary()
```

```
 Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.151   3.505   5.420   6.064   7.926  27.904
```

### 2.3.2. Usando `distributions`

La idea de `distributions` es definir instancias† de variables aleatorias. Esto es, definir una variable aleatoria de una familia paramétrica utilizando un valor definido de los parámetros que la definen. Por ejemplo, una $\mathsf{N}(\mu, \sigma) = \mathsf{N}(0, 1)$ o una $\mathsf{U}(a, b) = \mathsf{U}(0, 1)$.

```
library(distributions)
## Definimos nuestra variable aleatoria
x ← UniformRandomVariable$new()

## Muestreamos 10,000 por cada renglon.
set.seed(108)
x$sample(10**4, 3) ▷
  log() ▷
  apply(2, function(x){-2 * sum(x)} ) ▷
  summary()
```

```
 Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.151   3.505   5.420   6.064   7.926  27.904
```

## 2.4. Variables Gaussianas correlacionadas

Supongamos que queremos generar un par de variables $X \in \mathbb{R}^2$ de tal forma que

$$X \sim \mathsf{N}(0, \Sigma) \ , \tag{7}$$

donde $\Sigma_{ii} = 1$ para $i \in \{1, 2\}$ y $\Sigma_{ij} = \rho$ con $i \neq j$. Supongamos que sólo sabemos generar números aleatorios $\mathsf{N}(0, 1)$.

¿Cómo generaríamos los vectores aleatorios que necesitamos?

¿Qué saben de propiedades matriciales de vectores aleatorios?

```
set.seed(108)
Sigma ← diag(2); Sigma[1,2] ← .75; Sigma[2,1] ← .75;
L ← chol(Sigma)

Z ← rnorm(2 * 10^4)        # Generamos vectores estandar
Z ← matrix(Z, nrow = 2)    # Reacomodamos en matriz
X ← t(L) %*% Z             # Transformacion lineal
cov(t(X))
```

```
        [,1]   [,2]
[1,] 1.0173 0.7772
[2,] 0.7772 1.0312
```

El operador `%*%` ejemplifica uno de las limitantes por diseño de `R`. Pues no está hecho para realizar operaciones vectoriales de manera nativa. Por ejemplo, en `Matlab` las operaciones son nativas y en `python` a través de `numpy` las operaciones matriciales también (y parte de los métodos).

# 3. Distribuciones discretas

Ahora, veremos algunas técnicas generales para distribuciones discretas. O mejor dicho, para generar números aleatorios con soporte en los enteros.

Supongamos que nuestro objetivo es poder generar de una $X \sim \mathbb{P}_\theta$ donde $X \in \mathbb{Z}$. La estrategia es `guardar todas las probabilidades del soporte`. Es decir, calcular

$$p_0 = \mathbb{P}_\theta(X \leq 0)\,, \quad p_1 = \mathbb{P}_\theta(X \leq 1)\,, \ldots\,, \tag{8}$$

generar $U \sim \mathsf{U}(0,1)$ y establecer

$$X = k \text{ si } \; p_{k-1} < U < p_k\,. \tag{9}$$

## 3.1. Binomial

Supongamos que nos interesa $X \sim \mathsf{Bin}(10, 0.3)$, el vector de probabilidades lo podemos calcular con la función `pbinom(k, 10, .3)`.

```
k ← 1:10
pbinom(k, 10, .3)
```

```
[1] 0.1493 0.3828 0.6496 0.8497 0.9527 0.9894 0.9984 0.9999 1.0000 1.0000
```

```
rbinomial ← function(nsamples, size, theta){
  probs ← pbinom(1:10, size, theta)
  x ← c()
  for (jj in 1:nsamples){
    u ← runif(1)
    x[jj] ← which(probs > u)[1]
  }
  return(x)
}
```

## 3.2. Poisson

Ahora supongamos que nos interesa simular de una Poisson con parámetro $\lambda = 7$.
¿Cuál es el soporte de una $\mathsf{Bin}(10, 0.3)$? ¿Cuál es el soporte de una $\mathsf{Poisson}(7)$?
Tenemos que guardar las probabilidades

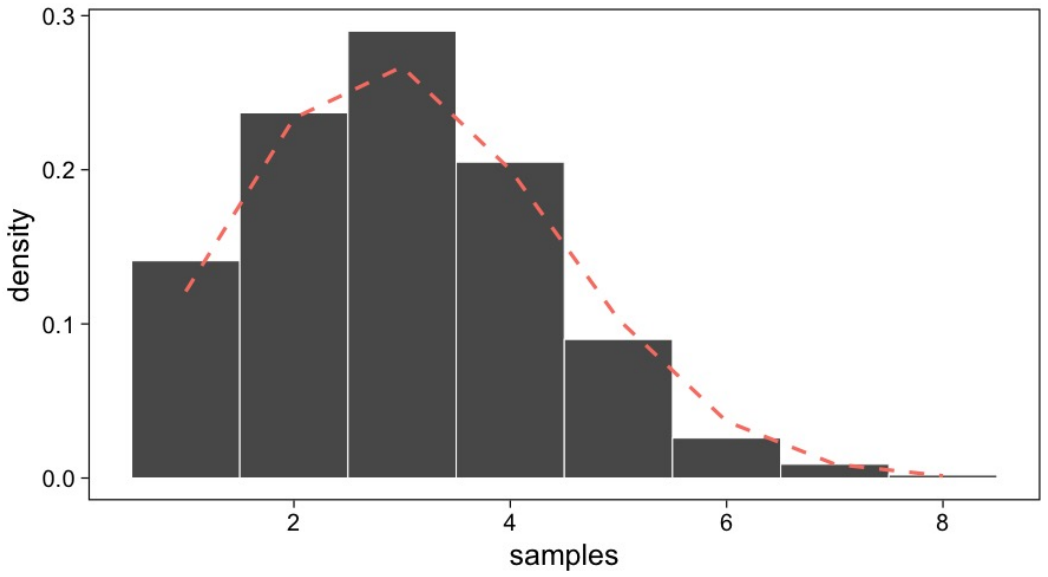



FIGURA 3. *Muestras de una binomial utilizando el método descrito anteriormente.*

```
k ← 1:24
ppois(k, 7)
```

```
 [1] 0.007295 0.029636 0.081765 0.172992 0.300708 0.449711 0.598714 0.729091
 [9] 0.830496 0.901479 0.946650 0.973000 0.987189 0.994283 0.997593 0.999042
[17] 0.999638 0.999870 0.999956 0.999986 0.999995 0.999999 1.000000 1.000000
```

```
rpoisson ← function(nsamples, lambda){
  probs ← ppois(1:30, lambda)
  x ← c()
  for (jj in 1:nsamples){
    u ← runif(1)
    x[jj] ← which(probs > u)[1]
  }
  return(x)
}
```

El problema de generar números aleatorios de la manera anterior es la necesidad de *guardar* el vector de probabilidades. Por ejemplo, una Poisson(100). El intervalo $\lambda \pm 3\sqrt{\lambda}$ es $(70, 130)$.

**Proposición 3.1** (Regla empírica o regla de las 3 sigmas)**.** Si $X$ es una variable aleatoria con media y varianza finitas. Entonces, la probabilidad de que una realización de $X$ se encuentre a mas de 3 desviaciones estándar de la media es a lo mas 5 %.

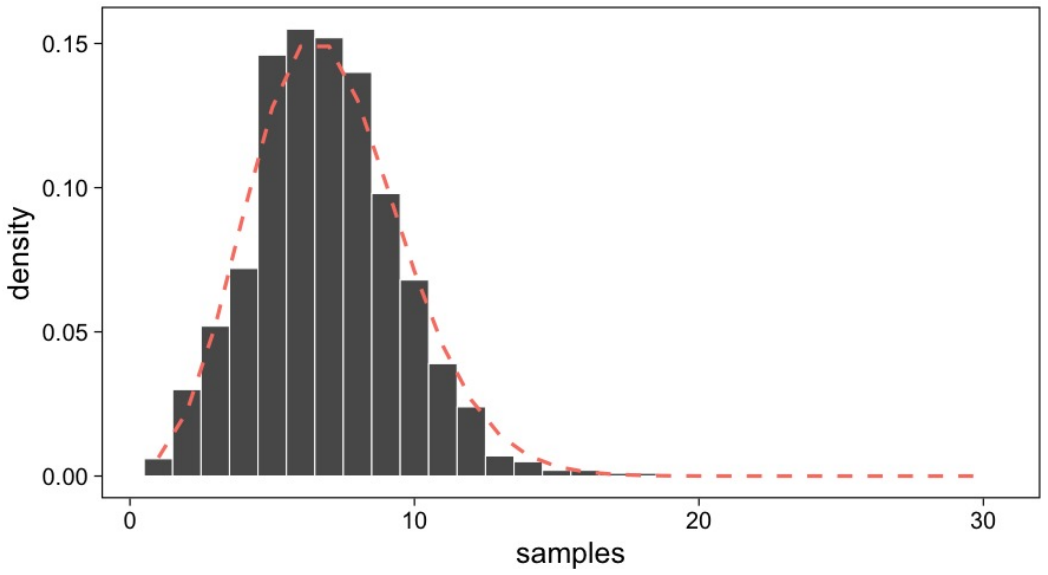



FIGURA 4. *Muestras de una Poisson utilizando el método descrito anteriormente.*

## 4. Mezclas

Otra familia de distribuciones que es muy interesante de simular son las mezclas. Es decir, cuando podemos escribir

$$\pi(x) = \int_{\mathcal{Y}} \pi(x|y)\,\pi(y)\,\mathrm{d}y\,, \quad \text{ó} \quad \mathbb{P}(x) = \sum_{i\in\mathcal{Y}} \mathbb{P}(x|Y=i)\,\mathbb{P}(Y=i)\,, \tag{10}$$

siempre y cuando sea sencillo generar números aleatorios de la condicional y la marginal adicional.

Por ejemplo, para generar números aleatorios de una $t$ Student con $\nu$ grados de libertad. Podemos usar la representación

$$X|y \sim \mathsf{N}(0, \nu/y)\,, \quad Y \sim \chi^2_\nu\,. \tag{11}$$

### 4.1. Binomial negativa

La variable aleatoria $X \sim \mathsf{BinNeg}(n,\theta)$ tiene una representación

$$X|y \sim \mathsf{Poisson}(y)\,, \quad Y \sim \mathsf{Gamma}(n,\beta)\,, \tag{12}$$

donde $\beta = (1-\theta)/\theta$.

```
nsamples ← 10^4
n ← 6; theta ← .3
y ← rgamma(nsamples, n, rate = theta/(1-theta))
x ← rpois(nsamples, y)
```

**Pregunta.** Prueba que una binomial negativa puede ser expresada como una mezcla Poisson-Gamma.

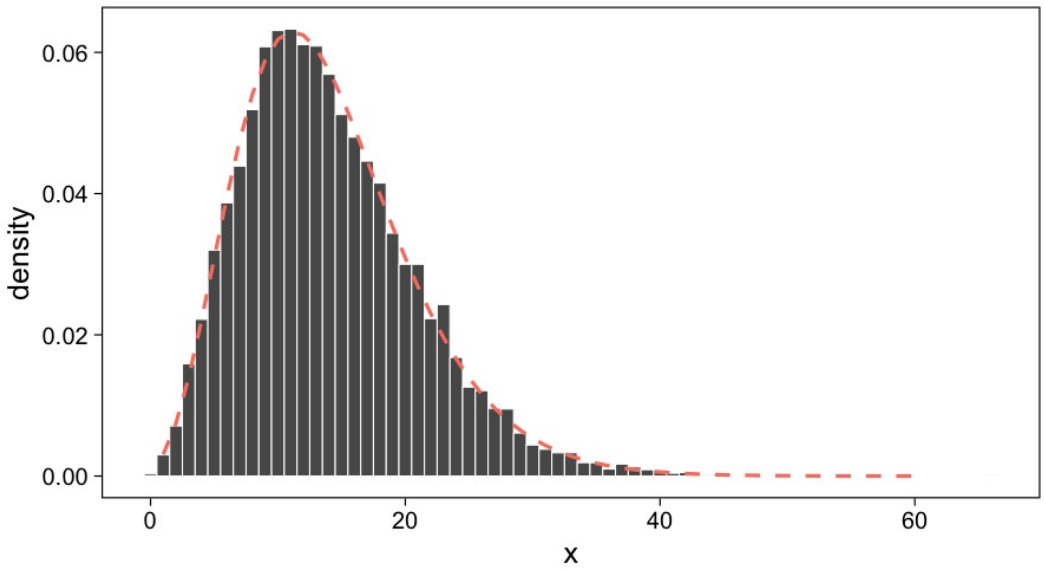



FIGURA 5. *Muestras de una binomial negativa utilizando el método de mezcla descrito anteriormente.*

### 4.2. Beta Binomial

Una distribución bastante conocida es la distribución Beta-Binomial la cual, como su nombre indica, está conformada por una mezcla de una $\mathsf{Bin}(n, \theta)$ y una $\mathsf{Beta}(\alpha, \beta)$. Es decir, $x \sim \mathsf{BetaBin}(n, \alpha, \beta)$ si su función de masa de probabilidad está dada por

$$\mathbb{P}(x) = \binom{n}{x} \frac{B(x+\alpha, n-x+\beta)}{B(\alpha, \beta)}\,, \tag{13}$$

donde $B(\alpha, \beta)$ se conoce como la función Beta $B(\alpha, \beta) = (\Gamma(\alpha)\Gamma(\beta))/\Gamma(\alpha+\beta)$.

```
nsamples ← 10^4
n ← 20; a ← 5; b ← 2
theta ← rbeta(nsamples, a, b)
x ← rbinom(nsamples, n, theta)
```

### 4.3. Mezclas Gaussianas

Otro modelo *famoso* es el de una mezcla de Gaussianas. Donde tenemos $k$ posibles poblaciones cada una con proporción $\pi_k \in (0, 1)$ y $\sum \pi_k = 1$. Y además, cada población tiene comportamiento distinto

$$\pi(x|k) = \mathsf{N}(x|\mu_k, \sigma_k)\,. \tag{14}$$

```
nsamples ← 10^5
y ← sample(1:3, size = nsamples, prob = c(.1, .7, .2), replace = TRUE)
x ← rnorm(nsamples,
          mean = ifelse(y==1, 1, ifelse(y==2, 2, 5)),
          sd = ifelse(y==1, 0.1, ifelse(y==2, 0.5, 1)))
```

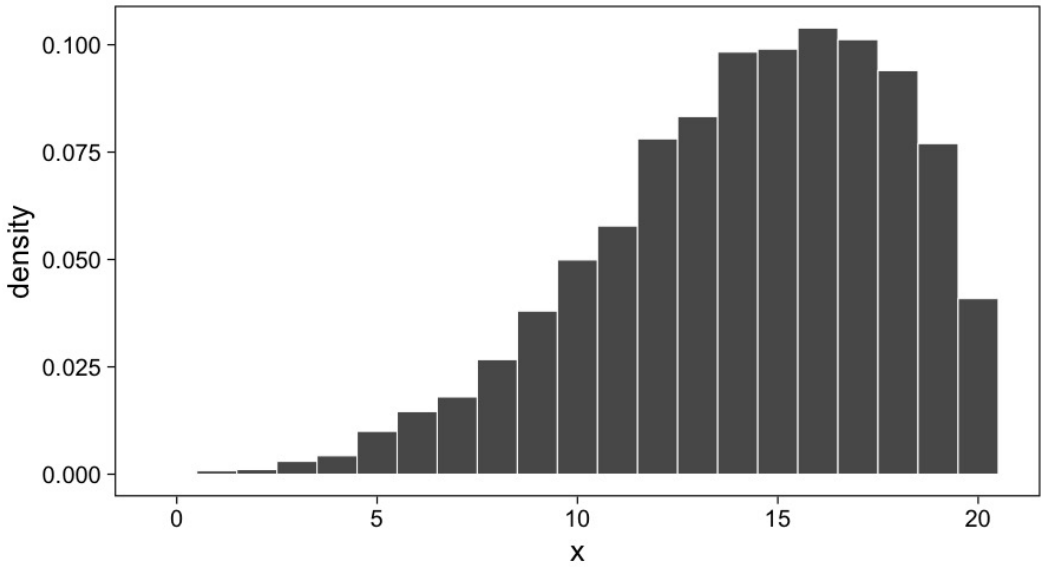



FIGURA 6. *Muestras de una beta binomial utilizando el método de mezclas descrito anteriormente.*

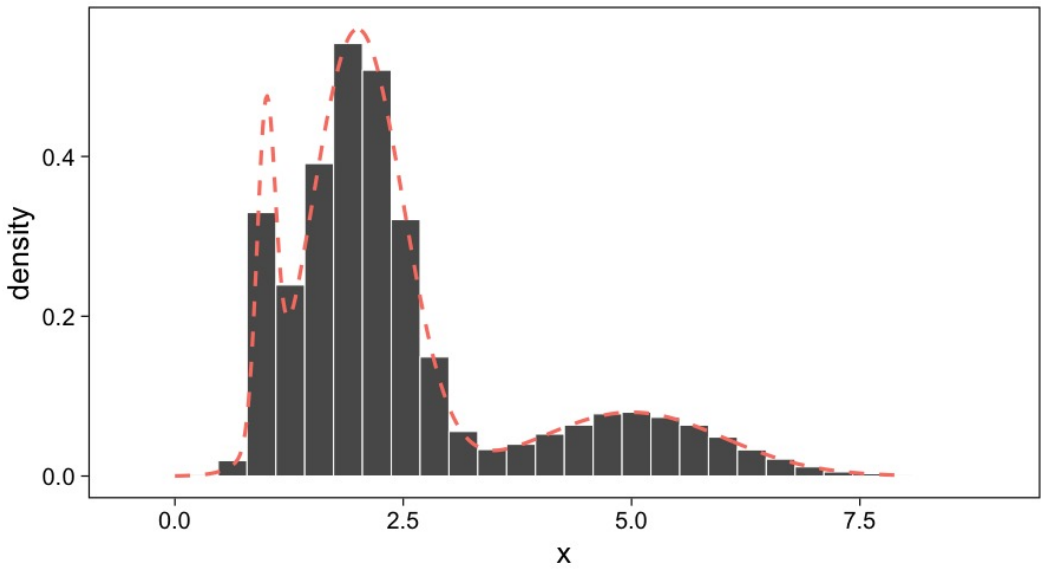



FIGURA 7. *Muestras de una mezcla Gaussiana de tres componentes utilizando el método descrito anteriormente.*

### 4.4. Comentarios

- El procedimiento para generar aleatorios de una mezcla nos permite obtener `distribuciones marginales a partir de la conjunta`.
- La combinación de componentes nos permite ser selectivos en los métodos de generación que podamos utilizar (respetando la estructura de `dependencia condicional`).
- Por ejemplo, en el modelo

$$\mathbb{P}(x) = \sum_{j \in \mathcal{Y}} \mathbb{P}(X|Y=j)\mathbb{P}(Y=j)\,. \tag{15}$$

## 5. Prueba $\chi^2$

Podemos usar otro mecanismo para probar estadísticamente si nuestros números pseudo aleatorios siguen la distribución que deseamos.

Podemos pensar en esta alternativa como la versión `discreta` de la prueba `KS`.

Lo que estamos poniendo a prueba es

$$H_0 : \mathbb{P}(x) = \mathbb{P}_0(x)\ \forall x \quad \text{contra} \quad H_1 : \mathbb{P}(x) \neq \mathbb{P}_0(x) \text{ para alguna } x\,. \tag{16}$$

### 5.1. Procedimiento de la prueba $\chi^2$

1. Hacemos una partición del rango de la distribución supuesta en $k$ subintervalos con límites $\{a_0, a_1, \ldots, a_k\}$, y definimos $N_j$ como el número de observaciones (de nuestro generador de pseudo-aleatorios) en cada subintervalo.
2. Calculamos la proporción esperada de observaciones en el intervalo $(a_{j-1}, a_j]$ como

$$p_j = \int_{a_{j-1}}^{a_j} \mathrm{d}\mathbb{P}(x)\,. \tag{17}$$

3. La estadística de prueba es

$$\chi^2 = \sum_{j=1}^{k} \frac{(N_j - np_j)^2}{np_j}\,. \tag{18}$$

> Nota que estamos comparando dos histogramas. El histograma observado que construimos a partir de nuestros números pseudo-aleatorios contra el histograma que esperaríamos de la distribución. ¿Puedes pensar en algún problema con esta prueba?

La visualización correspondiente sería lo siguiente. Utilizamos nuestro generador para obtener muestras.

```
## Esto es para poner a prueba un pseudo generador
rpseudo.uniform ← function(nsamples, seed = 108727){
  x0 ← seed; a ← 7**5; m ← (2**31)-1;
  x  ← x0;
  for (jj in 2:nsamples){
    x[jj] ← (a * x[jj-1]) %% m
  }
  x/m
}
```

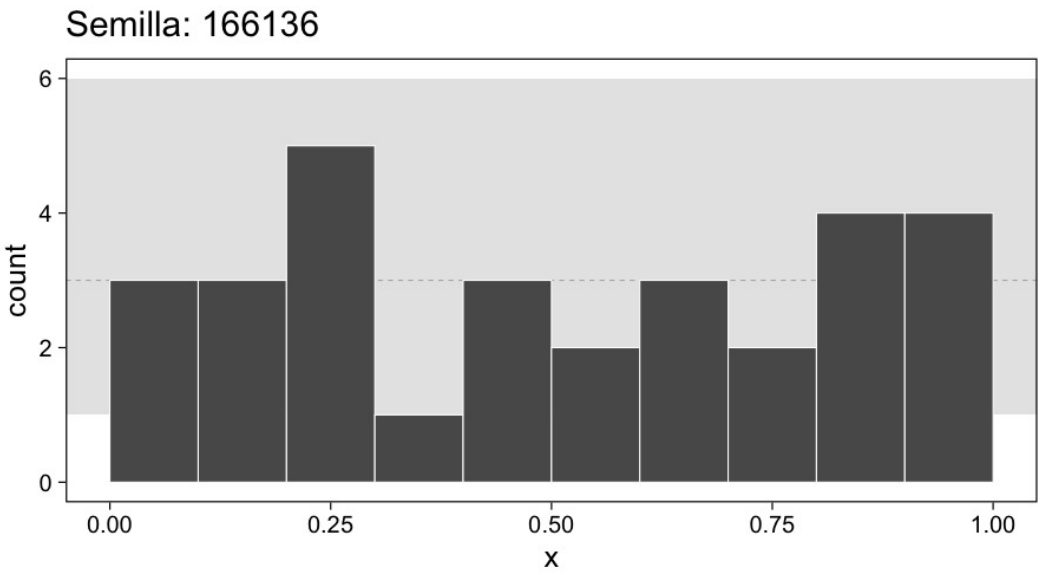



FIGURA 8. *Visualización de la realización de la prueba de $\chi^2$ descrita anteriormente.*

### 5.1.1. Pregunta:

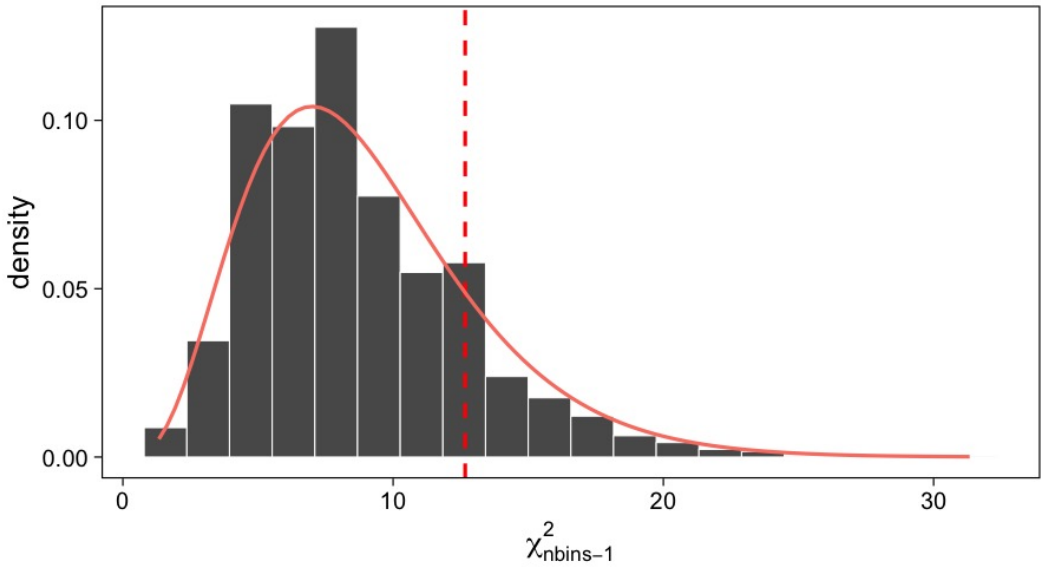
¿Qué esperaríamos de nuestro estadístico $\chi^2$ si nuestro generador de pseudo-aleatorios es incorrecto?

## 5.2. Aplicación de la prueba

```
set.seed(108727)
## Generamos muestra de nuestro generador (congruencial lineal)
samples ← tibble(x = rpseudo.uniform(nsamples))

## Calculamos frecuencias relativas y teoricas
Fn ← hist(samples$x, breaks = nbins, plot = FALSE)$counts/nsamples
F0 ← 1/nbins

## Calculamos nuestra referencia
X2.obs ← (nsamples*nbins)*sum((Fn - F0)**2)

## La imprimimos en pantalla
paste("íEstadstico: ", round(X2.obs, 3))
```

```
## Replicando experimentos ================================================
experiment ← function(nsamples, nbreaks = 20){
  samples ← data.frame(x = runif(nsamples))
  Fn ← hist(samples$x, breaks = nbreaks, plot = FALSE)$counts/nsamples
  F0 ← 1/nbreaks
  X2 ← (nsamples*nbreaks)*sum((Fn - F0)**2)
  return(X2)
}

X2 ← c()
for (jj in 1:5000){
  X2[jj] ← experiment(nsamples, nbins)
}
```

En la Apartado 5.2 se muestra el histograma de las réplicas del estadístico $\chi^2$ bajo el generador uniforme (lo tomamos como la distribución de la hipótesis nula) y comparamos contra el observado (línea punteada). Adicional, se incorpora la densidad de una $\chi^2_{k-1}$ (leáse ji-cuadrada con $k-1$ grados de libertad) que es la distribución asintótica del estadístico.




Por lo tanto, la probabilidad de haber observador una estadístico $\chi^2$ tan extremo como el que observamos si el generador hubiera sido el que suponemos es:

```
[1] "Estadistico: 12.6667, Probabilidad: 0.1628"
```

Que podemos comparar contra el que obtenemos de una prueba "tradicional":

```
counts.obs ← Fn*nsamples
chisq.test(counts.obs, p = rep(1, nbreaks)/nbreaks, simulate.p.value = TRUE)
```

```
Error: object 'nbreaks' not found
```

- La prueba $\chi^2$ pues usualmente no es buena cuando el número de observaciones es menor a 50.
- La prueba `KS` tiene mejor potencia que la prueba $\chi^2$:

```
ks.test(samples$x, "punif")
```

```

        Exact one-sample Kolmogorov-Smirnov test

data:  samples$x
D = 0.16, p-value = 0.4
alternative hypothesis: two-sided
```

### 5.3. Aplicación de pruebas

En la práctica se utiliza una colección de pruebas pues cada una es sensible a cierto tipo de desviaciones. La bateria de pruebas mas utilizada es la colección de pruebas `DieHARD` que desarrolló George Marsaglia y que se ha ido complementando con los años.

## 6. Distribución normal

Hasta ahora no hemos realmente hablado de cómo generar números de una de las distribuciones mas utilizadas. La distribución normal. Si pensamos en utilizar el método de la transformada inversa tendríamos que resolver la inversa de

$$\text{Prob}\{X \leq x\} = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(s-\mu)^2}{2\sigma^2}} \mathrm{d}s\,. \tag{19}$$

Sólo veremos ideas generales de cómo generamos números aleatorios de una distribución `normal estándar`.

### 6.1. Utilizando el teorema del límite central

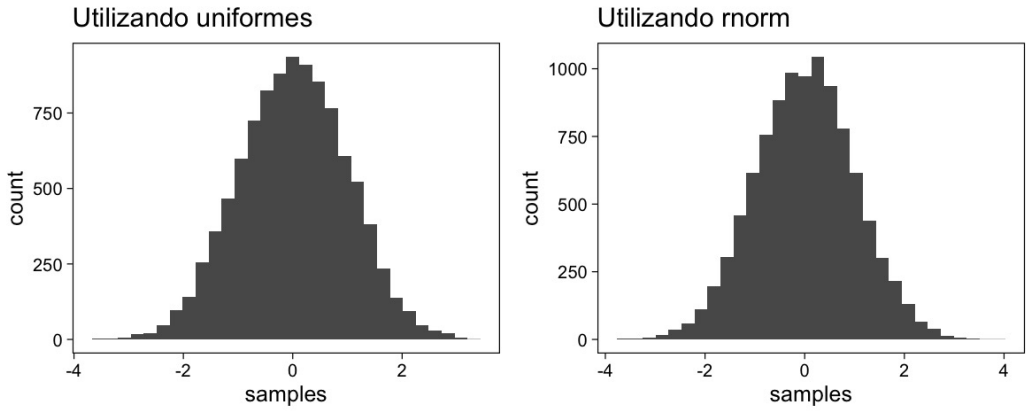
Podemos utilizar una colección $U_1, \ldots, U_k \overset{\text{iid}}{\sim} \mathsf{U}(a, b)$ de tal forma que tengan

$$\mathbb{E}(U_i) = \frac{1}{2}\,, \qquad \mathbb{V}(U_i) = \frac{1}{12}\,. \tag{20}$$

pues por el CLT tenemos que

$$Z = \frac{\sum_{i=1}^{k} U_j - k/2}{\sqrt{k/12}} \dot{\sim} \mathsf{N}(0, 1)\,. \tag{21}$$

```
nsamples ← 10^4; k ← 12
U ← runif(k * nsamples)
U ← matrix(U, nrow = k)
X ← apply(U, 2, function(x){ (sum(x) - k/2)/sqrt(k/12) })
```




Depende de $k$ la calidad de la aproximación. Sin embargo, siempre será un método aproximado.

### 6.2. El método de Box-Müller

Este método se basa en transformación de coordenadas polares a cartesianas. Es decir,

$$(u_1, u_2) \to (z_1, z_2)\,, \tag{22}$$

donde $u_i$ son uniformes independientes y $z_i$ son normales estándar independientes.

Las coordenadas polares del vector $(z_1, z_2)$ son

$$R^2 = z_1^2 + z_2^2\,, \qquad \tan\theta = \frac{z_2}{z_1}\,. \tag{23}$$

La función de densidad conjunta para $(R^2, \theta)$ es igual a

$$\pi(r^2, \theta) = \frac{1}{2}\frac{1}{2\pi} e^{-\frac{r^2}{2}}\,, \quad 0 < r^2 < \infty\,, \quad 0 < \theta < 2\pi\,. \tag{24}$$

Podemos notar que tenemos

$$\pi(r^2, \theta) = \pi(r^2) \cdot \pi(\theta)\,. \tag{25}$$

Es decir, son `independientes`.

Asi que tenemos que

$$r^2 \sim \mathsf{Exp}(2)\,, \tag{26}$$

$$\theta \sim \mathsf{U}(0, 2\pi)\,. \tag{27}$$

Lo cual podemos finalmente escribir como

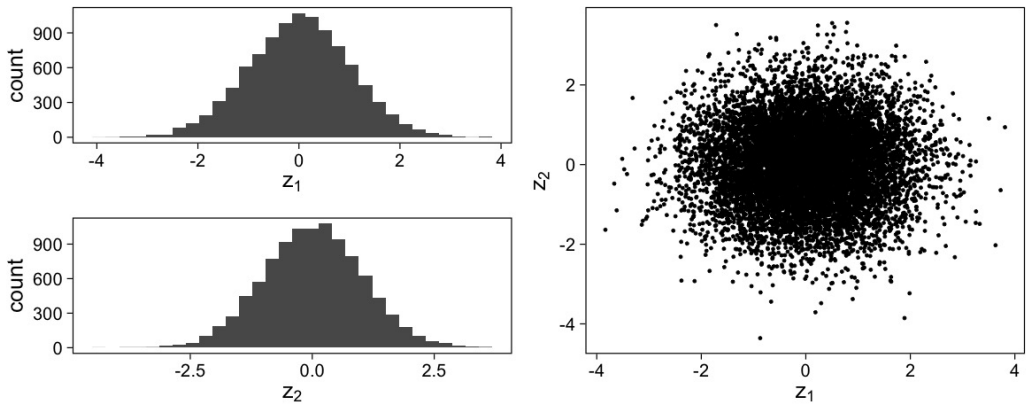
$$z_1 = \sqrt{-2\log u_1}\cos(2\pi u_2)\,, \tag{28}$$

$$z_2 = \sqrt{-2\log u_1}\sin(2\pi u_2)\,. \tag{29}$$

$$\tag{30}$$

El problema de esto es que necesitamos calcular funciones complejas $(\sin(\cdot), \cos(\cdot), \sqrt{\cdot})$.

```
rnormal.bm ← function(n){
  r ← sqrt(-2 * log(runif(n)))
  theta ← runif(n, 0, 2 * pi)
  z ← matrix(0, nrow = 2, ncol = n)
  z[1,] ← r * cos(2 * pi * theta)
  z[2,] ← r * sin(2 * pi * theta)
  return(z)
}
```



### 6.3. Método de Marsaglia

Usaremos el método de Box-Müller como punto de partida (coordenadas polares y cartesianas). Lo que nos interesa es poder generar números aleatorios dentro del círculo. Asi que generamos

$$V_1 = 2U_1 - 1\,, \quad V_2 = 2U_2 - 2\,, \tag{31}$$

a partir de $U_i \sim \mathsf{U}(0,1)$.

Si generamos un punto `dentro` del círculo unitario nos lo quedamos como una simulación válida. Si no, entonces repetimos la generación de los número aleatorios uniformes.

Si el punto se encuentra dentro del círculo entonces extraemos sus coordenadas por medio de

$$S = V_1^2 + V_2^2\,, \tag{32}$$

$$z_1 = \sqrt{\frac{-2\log S}{S}}V_1\,, \qquad z_2 = \sqrt{\frac{-2\log S}{S}}V_2\,. \tag{33}$$

El método anterior tiene probabilidad $\pi/4$ de generar un punto dentro del círculo. Esto es, para generar 2 números gaussianos independientes usaremos en promedio $4/\pi \approx 1.273$ iteraciones de lanzar dardos. Esto es, generar en promedio 2.546 números aleatorios y un cálculo de logaritmos, una raíz cuadrada, una división y 4.546 multiplicaciones.

## 7. Conclusiones y comentarios

- El método de la transformada inversa es útil. Sin embargo, no siempre se puede calcular de forma cerrada (distribución normal).
- El modelo de mezclas es muy común en aplicaciones financieras, estudios de mercado y problemas de clasificación (segmentación).

```
sessionInfo()
```

```
R version 4.3.1 (2023-06-16)
Platform: x86_64-apple-darwin20 (64-bit)
Running under: macOS Ventura 13.5.1

Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/4.3-x86_64/Resources/lib/
    libRblas.0.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/4.3-x86_64/Resources/lib/
    libRlapack.dylib;  LAPACK version 3.11.0

locale:
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

time zone: America/Mexico_City
tzcode source: internal

attached base packages:
[1] stats     graphics  grDevices datasets  utils     methods   base

other attached packages:
 [1] distributions_0.1.2 scales_1.2.1        patchwork_1.1.2
```

```
 [4] lubridate_1.9.2     forcats_1.0.0       stringr_1.5.0
 [7] dplyr_1.1.2         purrr_1.0.1         readr_2.1.4
[10] tidyr_1.3.0         tibble_3.2.1        ggplot2_3.4.2
[13] tidyverse_2.0.0

loaded via a namespace (and not attached):
 [1] gtable_0.3.3      compiler_4.3.1   renv_1.0.0       crayon_1.5.2
 [5] tidyselect_1.2.0 R6_2.5.1          labeling_0.4.2   generics_0.1.3
 [9] munsell_0.5.0     pillar_1.9.0     tzdb_0.4.0       rlang_1.1.1
[13] utf8_1.2.3        stringi_1.7.12   timechange_0.2.0 cli_3.6.1
[17] withr_2.5.0       magrittr_2.0.3   grid_4.3.1       hms_1.1.3
[21] lifecycle_1.0.3   vctrs_0.6.3      glue_1.6.2       farver_2.1.1
[25] fansi_1.0.4       colorspace_2.1-0 tools_4.3.1      pkgconfig_2.0.3
```

## Referencias

[1] C. Robert and G. Casella. *Monte Carlo Statistical Methods*. Springer Science & Business Media, mar 2013. 1

[2] S. M. Ross. *Simulation*. 2013. 1