Índice general

Tema 5

Métodos de Monte Carlo y Simulación

5.1. Introducción

5.1.1. ¿A qué simulación nos referimos?

La modelización es un elemento presente en la investigación y especialmente en el ámbito de las ciencias experimentales. Ante una realidad observada posiblemente muy compleja, un modelo para la misma considerará las variables más relevantes para explicar un fenómeno de interés y describirá las relaciones principales entre las mismas. Un modelo puede ser determinista o estocástico (con componentes aleatorias). Estos últimos constituyen una componente esencial en la Inferencia Estadística como ya hemos visto anteriormente. En ellos la incertidumbre es debida por una parte a que no se dispone de toda la información de las variables en el modelo:

> *Nothing in Nature is random. A thing appears random only through the incompleteness of our knowledge.* [Baruch Spinoza]

Y por otra a que el propio modelo puede no recoger todos los elementos necesarios para describirlo.

La Inferencia Estadística proporciona herramientas para estimar los parámetros de un modelo estocástico (o estadístico) y contrastar su validez a partir de los datos observados. Para ello idealmente hace uso de cálculos analíticos exactos (e.g. distribución en el muestreo de los estimadores). Sin embargo esto no siempre es posible y hay que recurrir a soluciones aproximadas. Estas pueden consistir en resultados asintóticos, basados en suposiciones a veces cuestionables, o bien recurrir a la simulación.

En este tema nos vamos a centrar fundamentalmente en aplicaciones de la *simulación estocástica* y sus aplicaciones en algunos problemas estadísticos. En muchos casos utilizamos el término *Monte Carlo* para referirnos a este tipo de simulación. Si bien es cierto que ambos utilizan algoritmos computacionales que se caracterizan por algún tipo de entrada aleatoria, los métodos de Monte Carlo son más generales y en principio no siempre están ligados a modelos estocásticos. Un ejemplo de estos es la integración de Monte Carlo que ilustraremos también en este tema.

5.1.2. Números aleatorios y pseudo-aleatorios

Una sucesión de números aleatorios puros se caracteriza porque no existe ninguna regla que permita conocer sus valores. Esta "impredicibilidad" es crucial en los juegos de azar y en áreas como la criptografía. En las aplicaciones estadísticas habitualmente estamos interesados en secuencias independientes e idénticamente distribuidas (iid) con distribución conocida, algo no siempre posible con estos números aleatorios puros, y además es fundamental la reproducibilidad. Una alternativa son los denominados números "pseudo-aleatorios" que constituyen valores generados mediante software y algoritmos recursivos, un ejemplo son los generadores congruenciales.

Habitualmente los algoritmos de simulación se basan en la posibilidad de generar números pseudo-aleatorios que reproduzcan las propiedades de variables aleatorias generadas independientemente desde una distribución de probabilidad uniforme, $\mathcal{U}(0,1)$.

5.2. Generación de números pseudo-aleatorios y distribuciones de probabilidad en R

Existen diversas herramientas para la generación de números (pseudo-)aleatorios[1] en el sistema base de R.

- La familia de funciones `rdistribución(n,...)` genera secuencias de valores aleatorios de la distribución de probabilidad especificada. Por ejemplo:

```
> rnorm(n=10,mean=10,sd=2) # Normal con media 10 y desviación típica 2

 [1]  8.465395  9.868629  8.711672 10.134511  6.353454 11.078604
 [7] 12.617382  6.512480 11.652253 11.412400

> runif(10) # Uniforme en (0,1)

 [1] 0.29038613 0.82419458 0.88216466 0.53419389 0.37678608
 [6] 0.01492945 0.62204454 0.28409062 0.60306062 0.92535892

> rpois(n=10,lambda=1) # Poisson con media 1

 [1] 0 1 2 0 4 0 0 1 3 0
```

  Algunas de las distribuciones más habituales son:

  - `runif`: uniforme continua

[1] En este tema sólo consideraremos este tipo de números a los que por simplicidad nos referiremos eliminado el prefijo "pseudo".

- `rnorm`: normal
- `rexp`: exponencial
- `rgamma`: gamma
- `rf`: F-Snedecor
- `rt`: t-Student
- `rchisq`: χ^2 (chi-cuadrado)
- `rweibull`: Weibull
- `rpois`: Poisson
- `rbinom`: Binomial
- `rgeom`: geométrica

- La función `sample()` genera muestras con y sin reemplazo. En general corresponden a distribuciones de probabilidad discretas cuyas probabilidades se pueden especificar en el argumento `prob`. Ejemplos:

```
> sample(1:5,5) # permutación

[1] 5 3 2 1 4

> sample(1:5,10,replace=TRUE) # muestra con reemplazo

 [1] 4 2 3 2 5 3 4 3 4 1

> sample(c(0,1),5,replace=TRUE,prob=c(0.4,0.5)) # Bernoulli

[1] 0 1 0 0 0

> sample(cars$speed,3)  # tres valores aleatorios de cars$speed

[1] 11 20  9
```

- La función `set.seed()` permite establecer la semilla[2] y el generador, lo que nos permite reproducir los resultados. Ejemplo:

[2] El objeto `.Random.seed`, localizado en el espacio de trabajo, almacena la semilla actual generada con el reloj del ordenador.

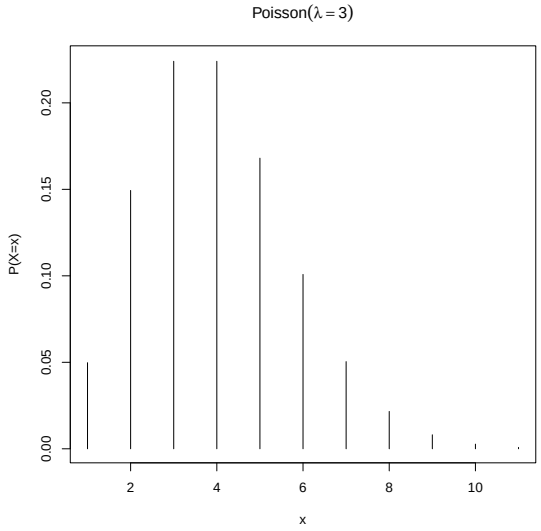
```
> mx<-numeric()
> for (i in 1:10)
+ {
+   set.seed(i)
+   x<-rnorm(100, mean=2)
+   mx[i]<-mean(x)
+ }
> mean(mx)

[1] 2.00625
```

Otra función más general es `RNGkind()` que permite seleccionar el tipo de generador.

La familia de funciones `rdistribución` que hemos descrito antes permite la generación de valores aleatorios de distribuciones de probabilidad. Para cada distribución disponemos además de familias de funciones similares cambiando la `r` inicial (que hace mención a *random*) por:

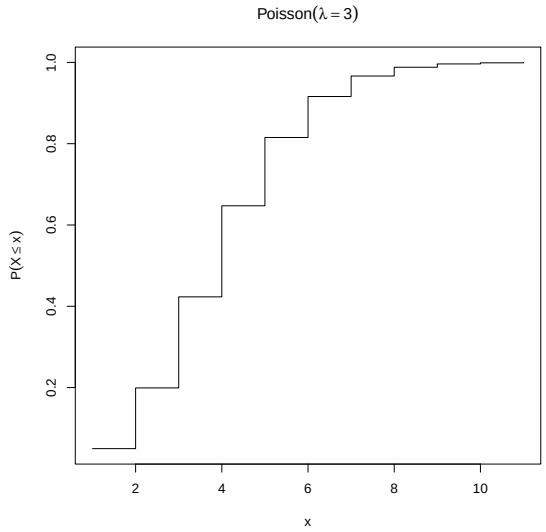
d para evaluar la función de densidad (*density*) o función masa de probabilidad (si es discreta) de la distribución correspondiente.

```
> dpois(x=0:10,lambda=3)

 [1] 0.0497870684 0.1493612051 0.2240418077 0.2240418077
 [5] 0.1680313557 0.1008188134 0.0504094067 0.0216040315
 [9] 0.0081015118 0.0027005039 0.0008101512

> plot(dpois(x=0:10,lambda=3),type='h',xlab='x',
+      ylab='P(X=x)',main=expression(Poisson (lambda==3)))
```



p (*percentile*), evalúa la función de distribución, $F(x) = P(X \leq x)$.

```
> plot(ppois(q=0:10,lambda=3),type='s', xlab='x',ylab=expression(P(X <= x)),
+      main=expression(Poisson (lambda==3)))
```



q (*quantile*), devuelve la inversa de la función de distribución, esto es, el valor x tal que $P(X \leq x) = p$.

```
> qnorm(0.025,lower.tail = FALSE) # valor crítico (Normal) alpha=0.05

[1] 1.959964
```

Un paquete bastante útil para trabajar con distribuciones de probabilidad es *distr*. Podemos consultar un listado completo que incluye funciones en paquetes adicionales en `http://cran.r-project.org/web/views/Distributions.html`.

5.3. Aproximación de una media teórica por simulación

La simulación nos permite aproximar valores teóricos de distribuciones de probabilidad. Una aplicación es la aproximación de la media de un estadístico muestral, a partir de una secuencia generada de valores aleatorios del estadístico.

5.3.1. Ley débil de los grandes números

Es conocido que dados $X_1, \ldots, X_n$ variables aleatorias iid con esperanza finita $\mathbb{E}[X_i] = \mu$, la *ley débil de los grandes números* establece que

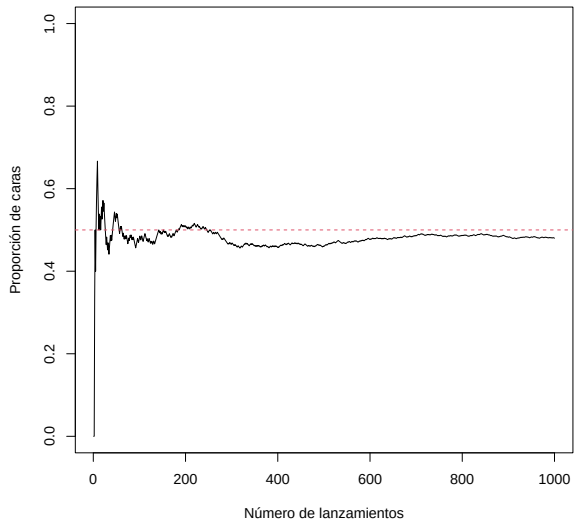
$$\bar{X}_n = \frac{1}{n}\sum_{i=1}^{n} X_i \to \mu,$$

en probabilidad, cuando $n \to \infty$. En particular si las variables son binarias, con distribución de Bernoulli de parámetro p ($P(X_i = 1) = p$), entonces $\bar{X} \to p$ (en probabilidad, cuando $n \to \infty$).

Vamos a utilizar este resultado para aproximar por simulación la probabilidad de cara al lanzar una moneda. El resultado nos dice que la probabilidad se puede obtener como límite de frecuencias relativas, cuando el tamaño de la muestra tiende a infinito. Para ello usamos la función `sample()` que nos permite simular el resultado del lanzamiento de una moneda, y suponemos que no está trucada:

```
> # Probamos primero algunos lanzamientos (uno, diez):
> sample(c(cara=1,cruz=0),1,replace=TRUE,prob=c(0.5,0.5))

cruz
   0

> x<-sample(c(cara=1,cruz=0),10,replace=TRUE,prob=c(0.5,0.5));x

cara cara cara cruz cara cara cara cruz cara cara
   1    1    1    0    1    1    1    0    1    1

> mean(x) # frecuencia relativa de cara en 10 lanzamientos

[1] 0.8

> x<-sample(c(cara=1,cruz=0),50,replace=TRUE,prob=c(0.5,0.5))
> mean(x) # frecuencia relativa de cara en 50 lanzamientos
```

```
[1] 0.58

> # Representamos gráficamente la solución
> set.seed(1) # para poder reproducir los resultados de abajo
> nsim<-1000
> x<-sample(c(cara=1,cruz=0),nsim,replace=TRUE,prob=c(0.5,0.5))
> n<-1:nsim
> plot(n, cumsum(x)/n,type='l',ylab='Proporción de caras',
+      xlab='Número de lanzamientos',ylim=c(0,1))
> abline(h=0.5,lty=2,col=2)
```



La simulación de cada lanzamiento que hemos hecho usando la función `sample()` se podría hacer también con `rbinom(nsim,size=1,prob=0.5)`. Otra programación alternativa podría ser `x<-runif(nsim)<0.5`.

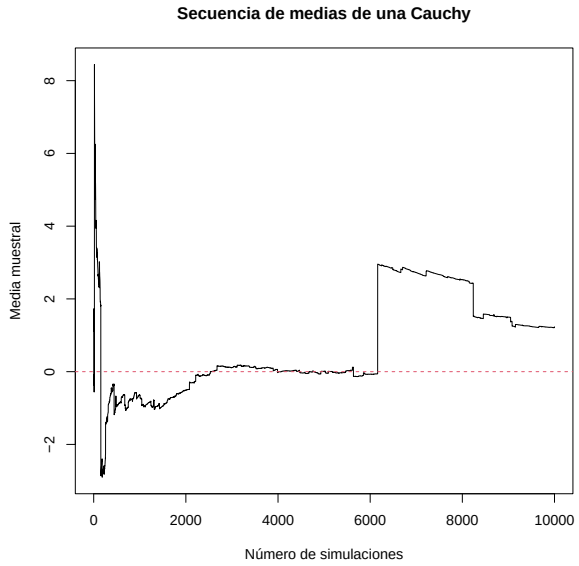
5.3.2. Detección de problemas de convergencia

Una suposición crucial para obtener la convergencia en la ley débil de los grandes números es que las variables X_i tengan esperanza finita. Veamos qué ocurre cuando la esperanza de las variables es infinita. Un conocido ejemplo es la distribución de Cauchy,

$$f(x) = \frac{1}{\pi(1+x^2)}, \quad x \in \mathbb{R}.$$

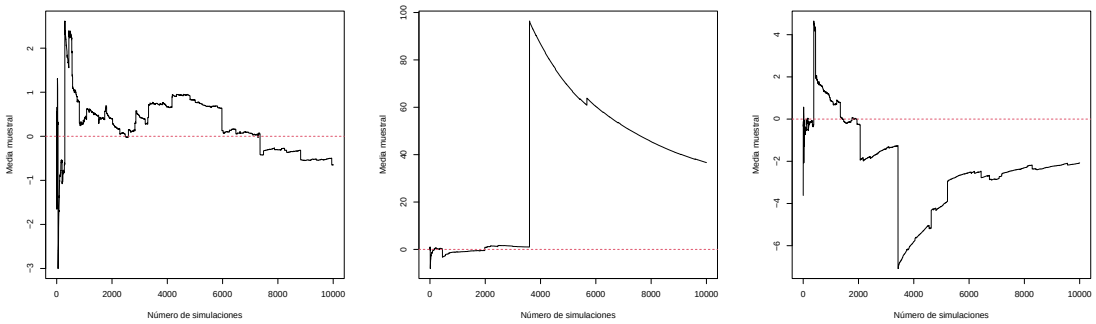
El siguiente código mostrará lo que ocurre con la secuencia de medias $\bar{X}_n$ cuando $n \to \infty$. Utilizamos la función `rcauchy()` que nos permite generar valores aleatorios de dicha distribución.

```
> set.seed(1)
> nsim<-10000
> x<-rcauchy(nsim)
> n<-1:nsim
> plot(n, cumsum(x)/n,type='l',ylab='Media muestral',
+      xlab='Número de simulaciones',
+      main='Secuencia de medias de una Cauchy')
> abline(h=0,lty=2,col=2)
```



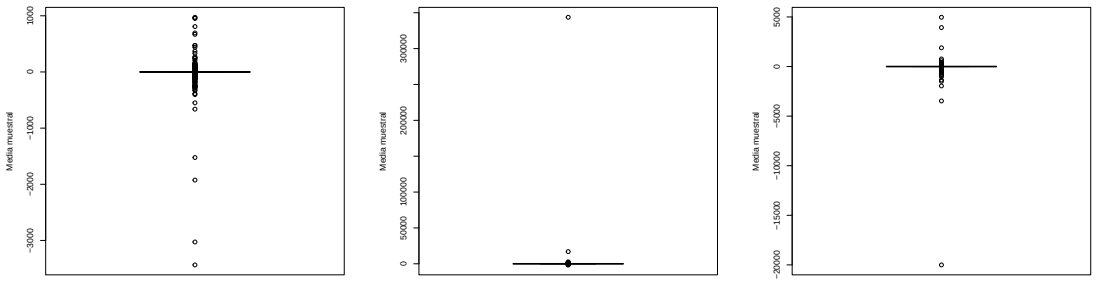
Podemos observar del gráfico anterior que se producen saltos a medida que crece el número de simulaciones y después de 10000 simulaciones aún no se ha estabilizado. Podemos repetir la ejecución cambiando la semilla para cerciorarnos del problema.

```
> for (i in 2:4)
+ {
+   set.seed(i)
+   x<-rcauchy(nsim)
+   plot(n, cumsum(x)/n,type='l',ylab='Media muestral',
+      xlab='Número de simulaciones')
+   abline(h=0,lty=2,col=2)
+ }
```



Para detectar problemas de convergencia en aplicaciones de simulación es recomendable representar la evolución de la aproximación sobre el número de simulaciones, como hemos hecho antes. Además puede ayudar observar algunos análisis descriptivos de las simulaciones. En este caso, un diagrama de cajas nos permite observar los valores (anómalos) que producen los saltos:

```
> for (i in 2:4)
+ {
+   set.seed(i)
+   x<-rcauchy(nsim)
+   boxplot(x,ylab='Media muestral')
+ }
```



5.3.3. Precisión de la aproximación

La ley débil de los grandes números nos permite por tanto aproximar medias y proporciones exactas por medias y proporciones muestrales. La dispersión de estos estimadores determina la precisión de la aproximación.

Si las variables $X_1, \ldots, X_n$ tienen varianza $\sigma^2 < \infty$, entonces la media muestral $\bar{X}_n$ tiene varianza $\mathbb{V}(\bar{X}_n) = \sigma^2/n$. Un estimador insesgado de dicha varianza es $\widehat{\mathbb{V}(\bar{X}_n)} = S_n^2/n$, donde S_n^2 es la cuasivarianza muestral. En el caso de proporciones (variables dicotómicas), la proporción muestral $\widehat{p}_n$, tiene varianza estimada $\widehat{\mathbb{V}(\widehat{p}_n)} = \frac{\widehat{p}_n(1-\widehat{p}_n)}{n-1}$.

Las varianzas anteriores nos dan en la práctica medidas de la precisión de la aproximación de la media (o proporción). Además el Teorema Central del Límite (TCL) nos permite construir intervalos de confianza.

> **Teorema Central del Límite.** Dada una secuencia $X_1, \ldots, X_n$ variables aleatorias iid con esperanza $\mathbb{E}[X_i] = \mu$ y varianza $\mathbb{V}(X_i) = \sigma^2 < \infty$, entonces:
>
> $$Z_n = \frac{\bar{X}_n - \mu}{\sigma/\sqrt{n}} \to \mathcal{N}(0,1),$$
>
> en distribución, cuando $n \to \infty$.

A partir del TCL podemos obtener un intervalo de confianza (asintótico) al nivel $1-\alpha$ para μ:

$$\left(\bar{X}_n - z_{\alpha/2}\frac{S_n}{\sqrt{n}}, \bar{X}_n + z_{\alpha/2}\frac{S_n}{\sqrt{n}}\right),$$

donde $z_{\alpha/2}$ denota el valor crítico de la Normal. El valor $z_{\alpha/2}\dfrac{S_n}{\sqrt{n}}$ se denomina error máximo admisible al nivel de confianza $1-\alpha$.

Ejemplo

Como ejemplo consideramos la aproximación por simulación de la media de una distribución Normal estándar. Consideramos para ello 1000 simulaciones:

```
> set.seed(1)
> nsim<-1000
> x<-rnorm(nsim)
```

La aproximación de la media teórica, en este caso $\mu = 0$, será:

```
> mean(x)

[1] -0.01164814
```

Como medida de la precisión de la aproximación podemos considerar el error estándar de la media $S_n/\sqrt{n}$: [3]

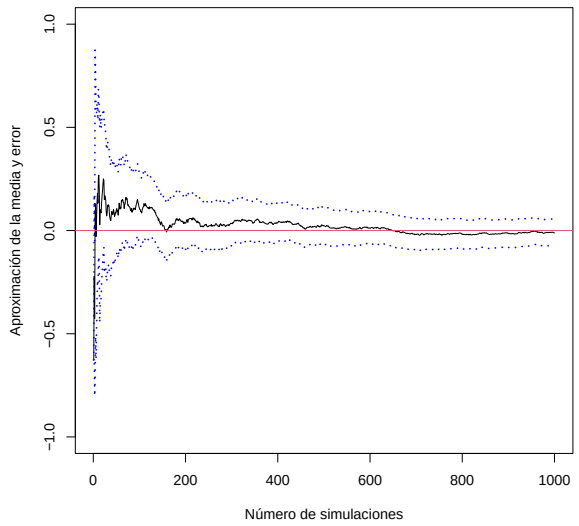
```
> # error estándar
> sd(x)/sqrt(nsim)

[1] 0.03272691
```

[3] También podríamos usar la varianza teórica sustituyendo S_n^2 por σ^2, ya que en este caso es conocida.

```
> # error máximo admisible al nivel de confianza 0.95
> qnorm(0.975)*sd(x)/sqrt(nsim)

[1] 0.06414357
```

Una forma conveniente de visualizar la convergencia es representar la secuencia de medias como hacíamos antes, añadiendo ahora los correspondientes intervalos de confianza que muestran el error de la aproximación:

```
> n<-1:nsim
> x<-x[n]
> # medias muestrales para n=1,2,...,nsim
> estim<-cumsum(x)/n
> # correspondientes errores de estimación
> # por simplicidad con varianza muestral en lugar de cuasivarianza
> estim.err<-sqrt(cumsum((x-estim)^2))/n
> # gráfico de convergencia
> plot(n,estim, type='l',xlab='Número de simulaciones',
+      ylab='Aproximación de la media y error',ylim=c(-1,1))
> # media teórica:
> abline(h=0,col=2)
> # intevalos de confianza (1-alpha=0.95)
> z<-qnorm(0.975)
> lines(estim-z*estim.err,lty=3,lwd=2,col='blue')
> lines(estim+z*estim.err,lty=3,lwd=2,col='blue')
```



Ejercicio. Comprueba y visualiza la convergencia en el caso variables con distribución uniforme, $\mathcal{U}(-3, 3)$.

5.3.4. Determinación del número de simulaciones

En los ejemplos anteriores hemos fijado el número de simulaciones como un valor del orden de 1000 o incluso 10000. Este número por supuesto es muy importante y su magnitud dependerá del problema de aproximación concreto y del objetivo que se pretende alcanzar.

Por ejemplo puede interesar obtener una aproximación con un nivel de precisión prefijado. Así se puede formar el problema de determinar n tal que el error máximo admisible para la aproximación de la media (al nivel de confianza $1 - \alpha$) sea inferior a un valor prefijado $\epsilon > 0$.

En el caso en que la varianza de las X_i, σ^2, sea conocida podemos formular el problema como:

$$\textit{Determinar } n \textit{ tal que } z_{\alpha/2}\frac{\sigma}{\sqrt{n}} < \epsilon.$$

Y la solución sería $n = \left(z_{\alpha/2}\frac{\sigma}{\epsilon}\right)^2$.

Ejercicio. Calcular el número de simulaciones necesario para aproximar la media μ de la siguientes distribuciones con un error máximo admisible de $0.1|\mu|$:

- Normal con media $\mu = 10$ y desviación típica $\sigma = 5$.
- Chi-cuadrado con 10 grados de libertad ($\mu = 10$, $\sigma^2 = 2 * \mu$).
- Poisson con parámetro $\lambda = 10$ ($\mu = 10$, $\sigma^2 = \mu$).

En el caso en que la varianza σ^2 sea desconocida, el problema se formularía usando el error máximo admisible estimado, esto es,

$$\textit{Determinar } n \textit{ tal que } z_{\alpha/2}\frac{S_n}{\sqrt{n}} < \epsilon.$$

Y el siguiente algoritmo nos permitirá en este caso determinar n:

1. Para $l = 0$ fijar un tamaño inicial n_0 (e.g. 30).
2. Generar $X_1, \ldots, X_{n_0}$, y calcular el error máximo admisible $e_0 = z_{\alpha/2}\frac{S_{n_0}}{\sqrt{n_0}}$.
3. Mientras que $e_l > \epsilon$:
   - $l = l + 1$,

- $n_l = \left(z_{\alpha/2} \frac{S_{n_0}}{\epsilon} \right)^2$
- Generar $X_1, \ldots, X_{n_l}$, y calcular el error e_l.

4. Devolver n_l.

5.4. Integración de Monte Carlo

La simulación proporciona herramientas para aproximar integrales multidimensionales del tipo:

$$I = \int \cdots \int h(x_1, \ldots, x_k)\, dx_1 \cdots dx_k,$$

con la ventaja, con respecto a los métodos de integración numérica, de que la velocidad de convergencia no depende del número de dimensiones k. En esta sección vamos a ver que aproximar una integral es un problema similar a aproximar una media.

5.4.1. Integración de Monte Carlo clásica

Integrales con límites finitos

Consideremos para empezar un caso sencillo, supongamos que nuestro objetivo es una integral del tipo:

$$I = \int_0^1 h(x)\, dx.$$

Teniendo en cuenta que la función de densidad de la distribución uniforme en el intervalo (0,1) es $f(x) = 1$, la integral anterior se puede escribir como:

$$I = \int_0^1 h(x)\, dx = \mathbb{E}[h(X)],$$

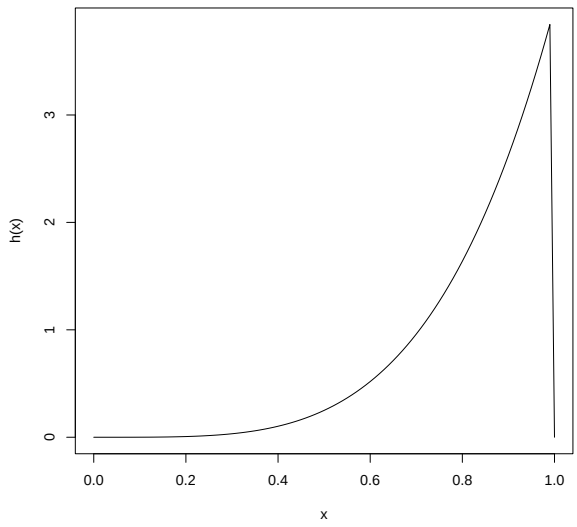
donde $X \sim \mathcal{U}(0,1)$. Ahora dado que la integral se ha escrito como una media teórica, podemos aproximarla simulando n variables aleatorias iid, $X_1, \ldots, X_n \sim \mathcal{U}(0,1)$, como sigue:

$$I = \int_0^1 h(x)\, dx = \mathbb{E}[h(X)] \approx \frac{1}{n} \sum_{i=1}^{n} h(X_i).$$

Ejemplo: Vamos a utilizar el procedimiento anterior para aproximar la integral $\int_0^1 4x^4\, dx$, y compararemos con el valor exacto que en este caso es $4/5$.

```
> # definimos la función a integrar
> h<-function(x) (4*x^4) * (x>0 & x<1)
> # visualizamos la función en el dominio de integración
```

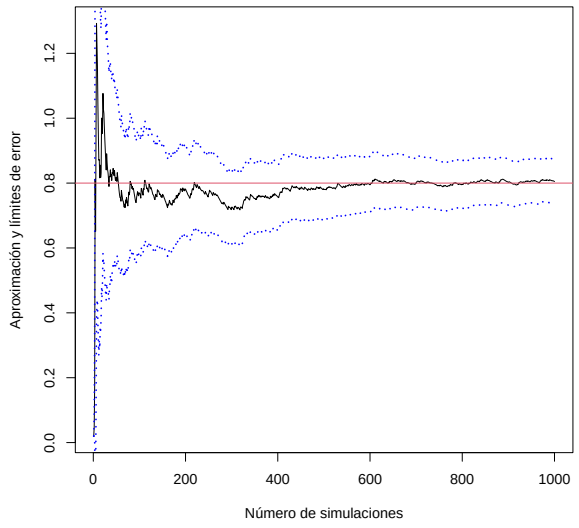
```
> curve(h,0,1)
> # fijamos el número de simulaciones (n=nsim)
> nsim<-100
> # calculamos la aproximación
> set.seed(1)
> x<-runif(nsim) # x1,...,xn
> hx<-sapply(x,h) # h(x1),...h(xn)
> mean(hx) # la aproximación final

[1] 0.7680473

> # valor exacto
> 4/5

[1] 0.8
```



Podemos completar el resultado anterior creando un gráfico que nos permita visualizar la convergencia, además de calcular el error de estimación. Para ello empleamos las mismas herramientas que utilizamos en la sección anterior:

```
> nsim<-1000
> set.seed(1)
> x<-runif(nsim)
> hx<-sapply(x,h)
> # aproximaciones para $n=1,...,nsim$
> estim<-cumsum(hx)/(1:nsim)
> # errores de estimación correspondientes
```

```
> estim.err<-sqrt(cumsum((hx-estim)^2))/(1:nsim)
> plot(1:nsim,estim,type='l',ylab='Aproximación y límites de error',
+      xlab='Número de simulaciones')
> z<-qnorm(0.025,lower.tail = FALSE)
> lines(estim - z*estim.err,col='blue',lwd=2,lty=3)
> lines(estim + z*estim.err,col='blue',lwd=2,lty=3)
> abline(h=4/5,col=2)
>
> # Aproximación final y su error
> estim[nsim]

[1] 0.8048108

> estim.err[nsim]

[1] 0.03432479
```



El procedimiento anterior se puede generalizar al caso de límites de integración genéricos (a, b) como sigue:

$$I = \int_a^b h(x)\, dx = (b-a) \int_a^b h(x) \frac{1}{b-a}\, dx = (b-a)\mathbb{E}[h(X)],$$

con $X \sim \mathcal{U}(a, b)$. Con lo que la aproximación de Monte Carlo sería:

$$I \approx (b-a)\frac{1}{n}\sum_{i=1}^{n} h(X_i),$$

con $X_1, \ldots, X_n \sim \mathcal{U}(a, b)$.

Integrales con límites infinitos (caso general)

El procedimiento que hemos descrito antes es válido siempre que los límites de integración sean finitos (a, b). Veamos cómo se extiende al caso general de una integral del tipo:

$$I = \int h(x)\ dx.$$

La idea es descomponer la función a integrar, $h(x)$, como producto de dos funciones, $h(x) = c(x)f(x)$, donde $f(x)$ sea una función de densidad. En ese caso la integral anterior se podría ver como una esperanza:

$$I = \int h(x)\ dx = \int c(x)f(x)\ dx = \mathbb{E}[c(X)],$$

donde X es una variable aleatoria con función de densidad $f(x)$. A partir de aquí la aproximación se obtendría como:

$$I = \mathbb{E}[c(X)] \approx \frac{1}{n}\sum_{i=1}^{n} c(X_i),$$

para variables aleatorias $X_1, \ldots, X_n$ iid con densidad f.

Ejemplo: Vamos a utilizar el procedimiento a anterior para aproximar la integral

$$\int_2^\infty \frac{1}{\sqrt{2\pi}} e^{-x^2/2}\ dx$$

Dado que el integrando corresponde en este caso a una función de densidad, en concreto a la de la distribución Normal estándar, podemos considerar la siguiente descomposición:

$$\begin{aligned} I &= \int_2^\infty \frac{1}{\sqrt{2\pi}} e^{-x^2/2}\ dx = \int c(x)f(x)\ dx \\ c(x) &= I(x > 2), \quad \text{donde } I \text{ es la función indicadora} \\ f(x) &= \frac{1}{\sqrt{2\pi}} e^{-x^2/2} \end{aligned}$$

Por tanto la aproximación la obtenemos generando $X_1, \ldots, X_n$ con densidad $f(x)$, esto es, desde la distribución $\mathcal{N}(0, 1)$, como sigue:

```
> nsim<-1000
> set.seed(1)
> x<-rnorm(nsim) # x1,...,xn
> cc<-function(x) (x>2)
> cx<-cc(x) ## c(x1),...,c(xn)
> # aproximación:
> mean(cx)
```

```
[1] 0.027

> # error
> qnorm(0.025,lower.tail=FALSE)*sd(cx)/sqrt(nsim)

[1] 0.01005087
```

La aproximación anterior la podemos comparar con el valor verdadero que se puede calcular como:

```
> pnorm(2,lower.tail = FALSE)

[1] 0.02275013
```

5.4.2. Muestreo por importancia

Aunque existen métodos estándar para generar valores aleatorios de muchas distribuciones[4], esta tarea puede ser bastante ardua dependiendo de la forma de la función de densidad $f(x)$. Técnicas como el denominado muestreo por importancia (*importance sampling*) permiten simplificar esta tarea obteniendo la aproximación de la integral

$$I = \int c(x)f(x)\ dx$$

como sigue:

1. Generar variables iid $Z_1, \ldots, Z_n$ desde una distribución con función de densidad $g(z)$ cuyo soporte incluya el soporte de $f(x)$.

2. Reescribir la integral como:

$$\begin{aligned} I &= \int c(x)f(x)\ dx = \int c(z)\frac{f(z)}{g(z)}g(z)\ dz \\ &= \int c(z)w(z)g(z)\ dz = \mathbb{E}[c(Z)w(Z)], \end{aligned}$$

   donde $w(z) = f(z)/g(z)$.

3. Aproximar la integral como:

$$I \approx \frac{1}{n}\sum_{i=1}^{n} c(Z_i)w(Z_i).$$

[4] En los ejemplos hemos considerado la familia de funciones de R `rdistribución` y la siguiente sección veremos métodos generales para distribuciones tanto discretas como continuas.

En la elección de la función de densidad auxiliar $g(z)$ se tienen en cuenta diversas consideraciones. Por ejemplo si la densidad $g(z)$ tiene colas más pesadas que la densidad $f(x)$, entonces es más probable que de lugar a simulaciones con varianza finita. Por otro lado es deseable que los $w(Z_i)$ sean homogéneos para evitar problemas de inestabilidad relacionados con datos influyentes[5].

Ejemplo: Vamos a utilizar la técnica de muestreo por importancia para calcular una aproximación de la integral:

$$\int_{4.5}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-x^2/2}\, dx,$$

considerando como función de densidad auxiliar una exponencial de parámetro $\lambda = 1$ truncada en 4.5, esto es,

$$g(z) = \lambda e^{-\lambda(z-4.5)}, \quad z > 4.5.$$

Con dicha densidad auxiliar la integral se aproximaría como:

$$\int_{4.5}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-x^2/2}\, dx = \int w(z)g(z)\, dz \approx \frac{1}{n}\sum_{i=1}^{n} w(Z_i),$$

a partir de variables iid $Z_1, \ldots, Z_n$, con densidad $g(z)$, siendo $w(z) = f(z)/g(z)$, con $f(z) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$, la densidad Normal estándar.

```
> nsim<-1000
> set.seed(1)
> z<-rexp(nsim)+4.5
> w<-dnorm(z)/dexp(z-4.5)
> # aproximación:
> mean(w)

[1] 3.144811e-06

> # error
> qnorm(0.025,lower.tail=FALSE)*sd(w)/sqrt(nsim)

[1] 2.687412e-07
```

Ejercicio. Comparar la aproximación anterior con la que se obtendría usando integración de Monte Carlo clásica.

[5]Una referencia para ampliar este tema puede ser el libro de Robert, C.P. y Casella, G. (2010). *Introducing Monte Carlo Methods with R*. Springer.

5.5. Simulación de variables aleatorias

En Estadística la simulación de variables aleatorias y en general de modelos estadísticos es parte esencial en muchos procedimientos. Por ejemplo para investigar las propiedades de estimadores, aproximando su distribución. En un contexto más general, simular una variable aleatoria con una función de densidad dada es un paso fundamental en la integración de Monte Carlo.

En esta sección nos vamos a centrar en el problema de simular valores de una variable aleatoria unidimensional. Si bien hemos descrito antes funciones en R que implementa la generación de distribuciones notables en la Estadística (la familia de funciones a la que nos hemos referido como `rdistribución`), vamos a estudiar los procedimientos básicos que permiten la simulación de una variable X genérica con función de densidad $f(x)$ (o función masa de probabilidad si es discreta). Por simplicidad describiremos el caso de variables continuas no obstante todos los métodos se generalizan para variables discretas.[6]

Describimos a continuación métodos básicos que toman como punto de partida que disponemos de un método de generación de números pseudo-aleatorios uniformes en $(0, 1)$. Recordamos que en R disponemos para ello de la función `runif()`.

5.5.1. Método de inversión

Se trata de un método sencillo y que en principio se aplica a cualquier distribución continua siempre que dispongamos de la función cuantil, esto es, la inversa de la función de distribución. La motivación (y justificación) del método es la siguiente:

> Si X es una variable aleatoria continua con función de distribución $F(x)$, continua y estrictamente monótona (invertible), entonces se tiene que
>
> $$U = F(X) \sim \mathcal{U}(0, 1).$$
>
> El recíproco también es cierto y por tanto, si $U \sim \mathcal{U}(0, 1)$ entonces $F^{-1}(U) = X$ tiene función de distribución $F(x)$.

El resultado anterior justifica el siguiente algoritmo para simular variables (iid) $X_1, \ldots, X_n$:

> **Algoritmo** (método de inversión):
>
> 1. *Generar* $U_1, \ldots, U_n$ *desde una distribución* $\mathcal{U}(0, 1)$.
> 2. *Devolver* $X_i = F^{-1}(U_i)$, $i = 1, \ldots, n$.

[6] Ver por ejemplo el libro de Robert, C.P. y Casella, G. (2010). *Introducing Monte Carlo Methods with R*. Springer. También el manual de Fernández-Casal, R. y Cao, R. (2022). *Simulación Estadística*. Disponible en `https://rubenfcasal.github.io/simbook/`

Ejemplo 1: Usando el algoritmo anterior generamos valores aleatorios desde una distribución Exponencial de parámetro (*rate*) $\lambda > 0$. Esta distribución tiene función de densidad y función de distribución dadas por:

$$\begin{aligned} f(x) &= \lambda e^{-\lambda x}, \quad x \geq 0 \\ F(x) &= 1 - e^{-\lambda x}, \quad x \geq 0. \end{aligned}$$

Con esto calculamos la inversa de $F(x)$:

$$u = F(x) = 1 - e^{-\lambda x} \quad \Leftrightarrow \quad x = F^{-1}(u) = -\frac{\log(1-u)}{\lambda}$$

Con lo que el algoritmo anterior sería:

1. Generar $U_1, \ldots, U_n$ desde una distribución $\mathcal{U}(0,1)$.

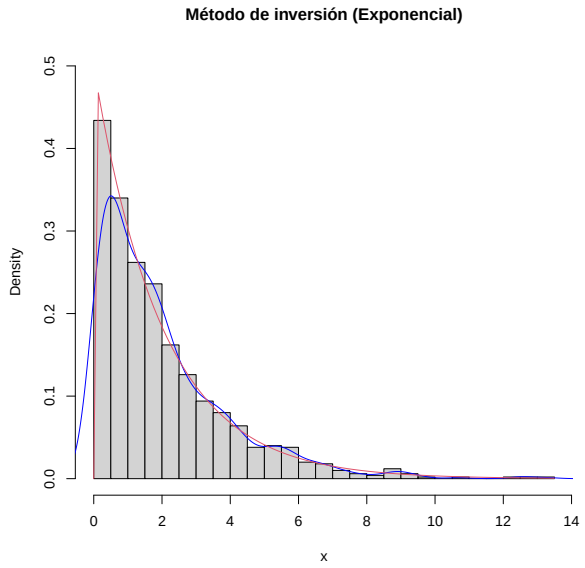
2. Devolver $X_i = -\frac{\log(1-U_i)}{\lambda}$, $i = 1, \ldots, n$.

En el algoritmo anterior podemos observar que $1 - U_i \sim \mathcal{U}(0,1)$, con lo que en el paso 2 sería equivalente a devolver directamente $X_i = -\frac{\log(U_i)}{\lambda}$. Utilizamos esta simplificación para implementarlo en R, generando $n = 1000$ valores aleatorios de una Exponencial con parámetro $\lambda = 0.5$:

```
> n<-1000
> set.seed(1)
> u<-runif(n)
> lambda<-0.5
> x<- -log(u)/lambda
> x[1:10]

 [1] 2.6522156 1.9770568 1.1142510 0.1925642 3.2021268 0.2143027
 [7] 0.1138281 0.8286148 0.9268855 5.5681482
```

Vamos a comprobar que los valores generados corresponden en efecto a valores de la distribución deseada. Lo hacemos primero gráficamente, representando la distribución de los X_i's generados mediante un histograma y la densidad suavizada, y comparando con la densidad de la Exponencial.

```
> hist(x,freq=FALSE,breaks='FD',main='Método de inversión (Exponencial)',
+      ylim=c(0,0.5))
> lines(density(x),col='blue')
> curve(dexp(x,rate=lambda),add=TRUE,col=2)
```



Finalmente podemos confirmar con un test de bondad de ajuste, por ejemplo el de Kolmogorov-Smirnov que está implementado en la función `ks.test()`:

```
> ks.test(x,pexp,rate=lambda)


One-sample Kolmogorov-Smirnov test

data:  x
D = 0.024366, p-value = 0.5928
alternative hypothesis: two-sided
```

Observamos que el p-valor nos indica que no se observan desviaciones de la hipótesis nula planteada, en este caso que el vector `x` corresponda a una muestra generada desde una Exponencial de parámetro `lambda=0.5`.

Ejemplo 2: Generamos ahora valores aleatorios desde una distribución doble Exponencial de parámetro $\lambda > 0$. Esta distribución, también denominada distribución de Cauchy, tiene función de densidad y función de distribución dadas por:

$$\begin{aligned} f(x) &= \frac{\lambda}{2} e^{-\lambda|x|}, \quad x \in \mathbb{R} \\ F(x) &= \begin{cases} \frac{1}{2} e^{\lambda x} & \text{si } x < 0 \\ 1 - \frac{1}{2} e^{-\lambda x} & \text{si } x \geq 0 \end{cases} \end{aligned}$$

Implementamos dos funciones, una que calcula la función de densidad, `ddexp()` y otra que realiza la simulación de n valores aleatorios, `rddexp()`, usando el método de inversión:

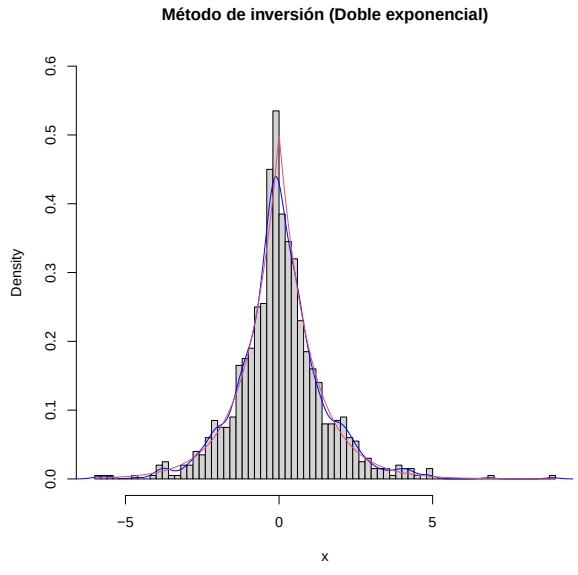
```
> ddexp <- function(x, lambda)  lambda*exp(-lambda*abs(x))/2
>
> rddexp<-function(n,lambda)
+ {
+   u <- runif(n)
+   x<-ifelse (u<0.5, log(2*u)/lambda, -log(2*(1-u))/lambda)
+   return(x)
+ }
```

Ahora generamos la secuencia de $n = 1000$ valores aleatorios de la doble exponencial de parámetro $\lambda = 1$:

```
> n<-1000
> set.seed(1)
> lambda<-1
> x<-rddexp(n,lambda)
> x[1:10]

 [1] -0.6329606 -0.2953812  0.1574807  1.6950807 -0.9079162
 [6]  1.5934630  2.2013881  0.3880117  0.2987135 -2.0909269
```

Finalmente representamos la densidad de los valores generados y comparamos con la densidad teórica:

```
> hist(x,freq=FALSE,breaks='FD',ylim=c(0,0.6),
+      main='Método de inversión (Doble exponencial)')
> lines(density(x),col='blue')
> curve(ddexp(x,lambda),add=TRUE,col=2)
```



Y calculamos el test de Kolmogorov-Smirnov usando la función `ks.test()`. Para ello necesitamos primero crear una función vectorial que calcule la función de distribución de la doble exponencial, y después pasaremos esta función como argumento a `ks.test()`.

```
> # la función de distribución de la doble exponencial
> pddexp<-function(x,lambda)
+ {
+   x<-ifelse (x<0, exp(lambda*x)/2, 1-exp(-lambda*x)/2)
+   return(x)
+ }
> # pasamos la función como argumento a ks.test()
> ks.test(x,pddexp,lambda=lambda)


One-sample Kolmogorov-Smirnov test

data:  x
D = 0.024366, p-value = 0.5928
alternative hypothesis: two-sided
```

El gráfico anterior y el p-valor obtenido nos confirman la adecuación de los valores generados.

Ejercicio: La distribución Weibull de parámetros $\lambda, \alpha > 0$ corresponde a una distribución de probabilidad continua con funciones de densidad y distribución dadas por:

$$\begin{aligned} f(x) &= \alpha\lambda^{\alpha}x^{\alpha-1}e^{-(\lambda x)^{\alpha}}, \quad x \geq 0 \\ F(x) &= 1 - e^{-(\lambda x)^{\alpha}}, \quad x \geq 0. \end{aligned}$$

Con lo que la inversa de $F(x)$ se obtendría como:

$$u = F(x) = 1 - e^{-(\lambda x)^{\alpha}} \quad \Leftrightarrow \quad x = F^{-1}(u) = \frac{(-\log(1-u))^{1/\alpha}}{\lambda}$$

Utilizando el método de inversión se pide generar $n = 1000$ valores de un distribución Weibull de parámetros[7] $\lambda = 0.5$ y $\alpha = 2$. Evaluar usando gráficos y el contraste de Kolmogorov-Smirknow que en efecto los valores generados provienen de dicha distribución.

Ventajas e incovenientes del método de inversión

- Aplicable a cualquier distribución continua para la que se pueda obtener una expresión explícita para la inversa de la función de distribución.

[7] En R la densidad Weibull está implementada en la función `dweibull()` usando parámetros denominados `shape` y `scale` que corresponden a α y $1/\lambda$, respectivamente.

- Hay situaciones en que obtener dicha expresión no es posible o es muy costosa (mucho tiempo de computación).

Una alternativa y posible solución al caso anterior consiste en utilizar métodos numéricos para resolver $F(x) - u = 0$, o usar una aproximación de $F^{-1}(u)$ (método de inversión aproximada). Otra alternativa es usar el método que describimos a continuación.

5.5.2. Método de aceptación-rechazo

Hay muchas distribuciones para las que el método de inversión no resulta apropiado. En estos casos una alternativa es recurrir a métodos indirectos donde en general:

- Generamos un valor candidato para la variable aleatoria.
- Aceptamos el valor generado sólo si verifica una condición particular.

Uno de estos métodos es el denominado método de aceptación-rechazo. Sólo requiere disponer de una expresión de la función de densidad f de la variable aleatoria X que se desea simular. El método utiliza una función de densidad auxiliar g de la que es más sencillo simular valores y que debe cumplir las dos condiciones siguientes:

- f y g tienen soportes compatibles (e.g. $g(x) > 0$ cuando $f(x) > 0$).
- Existe una constante M tal que $\frac{f(x)}{g(x)} \leq M$ para todo x en el soporte de f.

Algoritmo (método de aceptación-rechazo):

1. *Generar U desde una distribución $\mathcal{U}(0,1)$.*
2. *Generar un valor candidato Y desde la densidad auxiliar $g(y)$.*
3. *Comprobación:*
   - *Si $U < \frac{1}{M}\frac{f(Y)}{g(Y)}$ entonces $X = Y$ se acepta como valor simulado de f.*
   - *En caso contrario rechazar el candidato Y y volver al paso 1.*

Ejemplo 1 (Beta desde Uniforme): Usando el algoritmo anterior generamos valores desde una distribución Beta usando la distribución Uniforme como modelo auxiliar:

- Densidad objetivo: $f(x) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} x^{a-1}(1-x)^{b-1},\ 0 \leq x \leq 1$; con $a = 2.7$ y $b = 6.3$.
- Densidad auxiliar: $g(x) = 1,\ 0 \leq x \leq 1$.

La densidad auxiliar verifica las condiciones anteriores donde

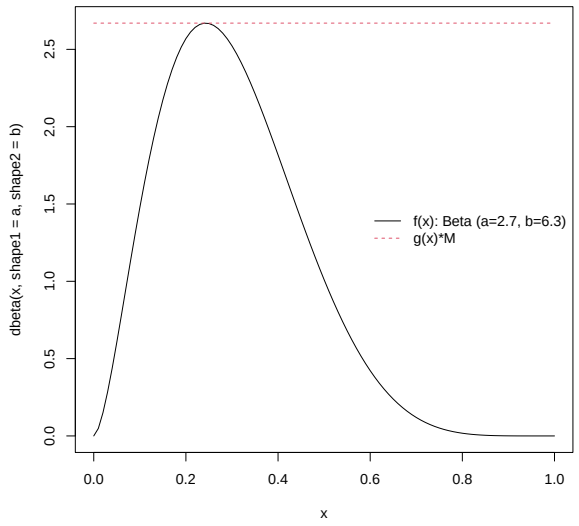
$$M = \sup_x \frac{f(x)}{g(x)} = \sup_x f(x)$$

se puede calcular numéricamente:

```
> a<-2.7; b<-6.3
> res<-optimize(f=function(x) dbeta(x,shape1=a,shape2=b),
+          maximum=TRUE,interval=c(0,1))
> res

$maximum
[1] 0.2428608

$objective
[1] 2.669744

> M<-res$objective
```

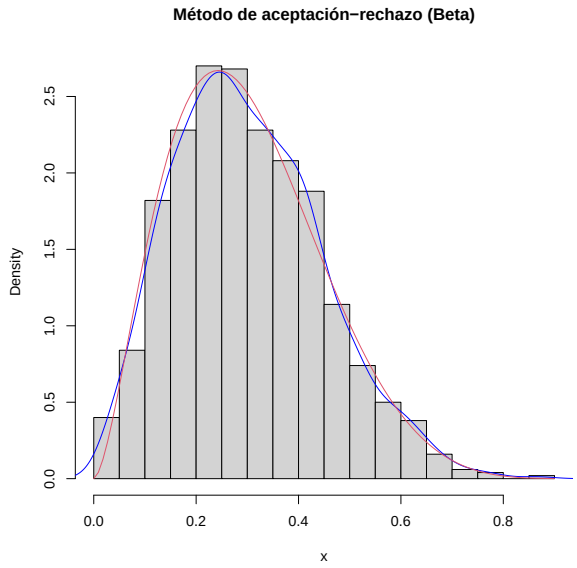
Podemos representar la densidad objetivo y la densidad auxiliar reescalada que se utiliza en la comprobación en el paso 3 del algoritmo:

```
> a<-2.7; b<-6.3
> curve(dbeta(x,shape1=a,shape2=b), 0,1)
> curve(M*dunif(x),0,1,add=TRUE,col=2,lty=2)
> legend('right', legend=c('f(x): Beta (a=2.7, b=6.3)',
+                          'g(x)*M'),
+        col=c(1,2),lty=c(1,2),bty='n')
```



Implementamos ahora el algoritmo de aceptación-rechazo para obtener $n = 1000$ valores de la Beta. Vamos a incluir un contador para comprobar cuántas generaciones son necesarias:

```
> n<-1000
> x<-double(n)
> f<-function(x) dbeta(x,shape1=a,shape2=b)
> g<-function(x) 1
>
> # Generación de los n valores a través del algoritmo
> set.seed(1)
> contador<-0
> for (i in 1:n)
+ {
+   u<-runif(1)
+   y<-runif(1)
+   contador<-contador+1
+   while (u>f(y)/(M*g(y)))
+   {
+     u<-runif(1)
+     y<-runif(1)
+     contador<-contador+1
+   }
+   x[i]<-y
+ }
> # Número de simulaciones total que han sido necesarias
> contador

[1] 2696

> # Veremos después que el número esperado
> n*M

[1] 2669.744

> # Comprobamos que la secuencia generada procede de la distribución deseada
> hist(x,freq=FALSE,breaks='FD',main='Método de aceptación-rechazo (Beta)')
> lines(density(x),col='blue')
> curve(dbeta(x,shape1=a,shape2=b),add=TRUE,col=2)
> ks.test(x,pbeta,shape1=a,shape2=b)


One-sample Kolmogorov-Smirnov test

data:  x
D = 0.018385, p-value = 0.8879
alternative hypothesis: two-sided
```



Observación: El uso de la Uniforme como densidad auxiliar es posible para cualquier densidad objetivo $f(x)$ que esté acotada en un intervalo cerrado, como era el caso de la Beta. No obstante el porcentaje de aceptación (y por tanto la eficiencia del algoritmo) no es muy alto.

Eficiencia del algoritmo y elección de g

Se puede probar que la probabilidad de aceptación en el paso 3 es $1/M$. Así en el ejemplo anterior donde usamos la Uniforme como densidad auxiliar el porcentaje de aceptación era de 37.46 %, no muy elevado. Además el número de generaciones de la densidad auxiliar necesarias para obtener un valor aceptable, N, es una variable aleatoria con distribución geométrica de parámetro $p = 1/M$. Por tanto el número medio de generaciones necesarias hasta obtener un valor aceptable es $E[N] = 1/p = M$. Esto nos indica que la eficiencia del algoritmo depende de M. Cuanto más cerca esté M de 1 mayor será la eficiencia[8].

A partir de las consideraciones anteriores se podría plantear el problema de encontrar una densidad auxiliar g óptima tal que el valor M asociado esté lo más próximo a 1 posible. Buscando dentro de una familia paramétrica de densidades auxiliares $\{g_\theta; \theta \in \Theta\}$, se formularía:

$$M_{opt} = \min_{\theta \in \Theta} \max_{x} \frac{f(x)}{g_\theta(x)}. \tag{5.1}$$

Vamos a implementar esta opción a continuación con otro ejemplo.

Ejemplo 2: Usando de nuevo el algoritmo de aceptación-rechazo generamos ahora valores

[8] Teniendo en cuenta que $M = 1$, implicaría que $f = g$, en cuyo caso el algoritmo no tendría sentido.

de una Normal estándar usando una distribución doble exponencial como modelo auxiliar:

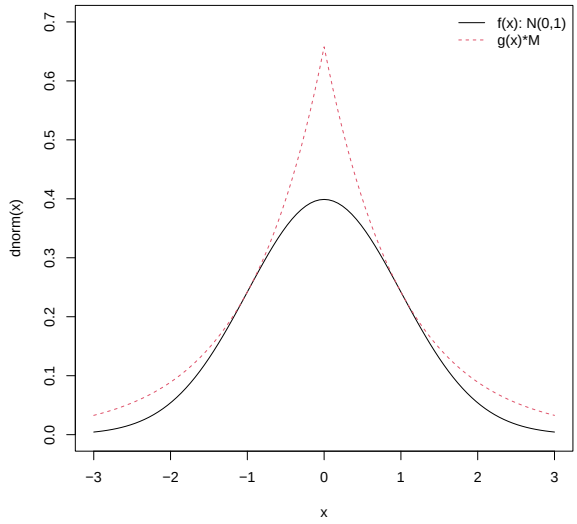
- Densidad objetivo: $f(x) = \frac{1}{\sqrt{2\pi}} \exp(-x^2/2),\ x \in \mathbb{R}$.
- Densidad auxiliar: $g(x) = \frac{\lambda}{2} \exp(-\lambda|x|),\ x \in \mathbb{R}$.

Dado que la densidad auxiliar depende del parámetro λ, comenzamos resolviendo numéricamente el problema de encontrar el valor óptimo para el algoritmo según la ecuación anterior (5.1):

```
> # Densidad de la doble exponencial:
> ddexp<-function(x,lambda) lambda*exp(-lambda*abs(x))/2
>
> # Construimos una función que dado lambda nos da el M óptimo
> M.lambda<-function(lambda)
+ {
+   optimize(f=function(x) dnorm(x)/ddexp(x,lambda), maximum=TRUE,
+            interval=c(0,2))$objective
+ }
> # Minimizamos la función anterior en lambda para obtener el óptimo
> # y el correspondiente M
> res<-optimize(M.lambda,interval=c(0.5,2))
> res

$minimum
[1] 0.9999987

$objective
[1] 1.315489

> lambda.opt<-res$minimum
> M<-res$objective
```

La doble exponencial con $\lambda \approx 1$ es por tanto la óptima para el algoritmo dentro de la familia de la doble exponencial. El porcentaje de aceptación en este caso es de $100/M$ =76.02 %, y el número esperado de simulaciones necesarias para obtener un valor aceptable es $M = 1.32$. Vamos a representar la densidad objetivo y esta densidad auxiliar reescalada:

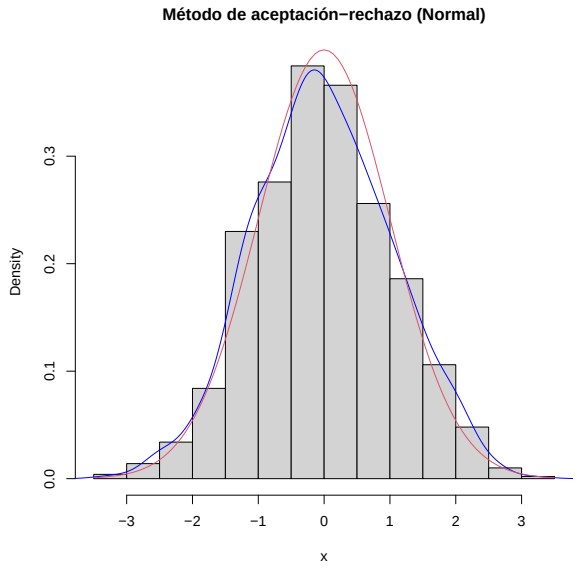
```
> curve(dnorm(x), -3,3,ylim=c(0,0.7))
> curve(M*ddexp(x,lambda.opt),add=TRUE,col=2,lty=2)
> legend('topright', legend=c('f(x): N(0,1)','g(x)*M'),
+        col=c(1,2),lty=c(1,2),bty='n')
```



Implementamos ahora el algoritmo con la elección anterior para simular $n = 1000$ valores de la Normal estándar. Utilizamos las funciones `ddexp()` y `rddexp()`, que creamos antes para el ejemplo de simulación de la doble exponencial usando el método de inversión. Volvemos a incluir el contador de simulaciones necesarias así como herramientas para comprobar que la generación es adecuada:

```
> # Función para simular de la doble exponencial (método de inversión)
> rddexp<-function(n,lambda)
+ {
+   u <- runif(n)
+   x<-ifelse (u<0.5, log(2*u)/lambda, -log(2*(1-u))/lambda)
+   return(x)
+ }
> # Método de aceptación-rechazo para simular la normal
> n<-1000
> x<-double(n)
> f<-function(x) dnorm(x)
> g<-function(x) ddexp(x,lambda=lambda.opt)
> # Generación de los n valores a través del algoritmo
> set.seed(1)
> contador<-0
> for (i in 1:n)
+ {
+   u<-runif(1)
+   y<-rddexp(1,lambda.opt)
+   contador<-contador+1
+   while (u>f(y)/(M*g(y)))
```

```
+   {
+     u<-runif(1)
+     y<-rddexp(1,lambda.opt)
+     contador<-contador+1
+   }
+   x[i]<-y
+ }
> # Número de simulaciones total que han sido necesarias
> contador

[1] 1329

> # El número esperado es
> n*M

[1] 1315.489

> # Comprobamos que la secuencia generada procede de la distribución deseada
> hist(x,freq=FALSE,breaks='FD',main='Método de aceptación-rechazo (Normal)')
> lines(density(x),col='blue')
> curve(dnorm(x),add=TRUE,col=2)
>
> # Confirmamos con el test de Kolmogorov-Smirnov
> ks.test(x,pnorm)


One-sample Kolmogorov-Smirnov test

data:  x
D = 0.027191, p-value = 0.4505
alternative hypothesis: two-sided
```



Modificaciones del método de aceptación-rechazo

En la eficiencia y el tiempo de computación del método descrito antes influyen varios factores:

- La proporción de aceptación $1/M$, que debería ser lo mayor posible. Para ello la cota $f(x) < Mg(x)$ debería ser lo más ajustada posible.
- La dificultad de simular desde la densidad auxiliar, que debería ser lo más simple posible.
- El tiempo necesario para la comprobación del Paso 3.

Si los factores anteriores no están bien ajustados entonces el algoritmo puede ser inviable en la práctica. Existen algunas modificaciones y mejoras de este algoritmo que buscan optimizar estos factores, entre ellas están:

- Para evitar (posiblemente costosas) evaluaciones $f(x)$, se puede usar una función *squeeze* (Marsaglia, 1977), $s(x)$, que aproxime la densidad f por debajo, $s(x) < f(x)$, estando lo más cerca posible de f. El algoritmo se podría modificar en ese caso como:

  1. Generar U desde una distribución $\mathcal{U}(0, 1)$.
  2. Generar un valor candidato Y desde la densidad auxiliar $g(y)$.
  3. Comprobación:
     - Si $U < \frac{1}{M}\frac{s(Y)}{g(Y)}$ entonces $X = Y$ se acepta como valor simulado de f.
     - En caso contrario, si $U < \frac{1}{M}\frac{f(Y)}{g(Y)}$ entonces $X = Y$ se acepta como valor simulado de f.

- En caso contrario rechazar el candidato Y y volver al paso 1.

Existen métodos generales para buscar estas funciones.

- Otros algoritmos: muestreo por rechazo adaptativo, método del cociente de uniformes.[9]

5.5.3. Otros métodos

Método de composición

Cuando la densidad objetivo, $f(x)$, se puede expresar como una mixtura discreta de densidades, esto es,

$$f(x) = \sum_{j=1}^{k} w_j f_j(x), \quad \text{con } \sum_j w_j = 1, w_j \geq 0, \text{y } f_j \text{ densidades},$$

entonces se puede definir el siguiente algoritmo:[10]

1. Generar J con distribución $P(J = j) = w_j$.
2. Generar X desde la densidad f_J.

Un ejemplo podría ser la función de densidad doble exponencial que veíamos antes, que se puede escribir como mixtura de dos densidades:

$$\begin{aligned} f(x) &= \frac{1}{2} f_1(x) + \frac{1}{2} f_2(x) \\ f_1(x) &= \lambda e^{-\lambda x}, \quad x \geq 0 \\ f_2(x) &= \lambda e^{\lambda x}, \quad x < 0 \end{aligned}$$

Otro ejemplo de mixtura discreta es la densidad suavizada que calcula la función `density()`. Se trata de un estimador tipo núcleo (*kernel density estimate*). Simular desde este tipo de mixtura es parte esencial de algunos métodos Bootstrap, en concreto los denominados Bootstrap suavizados.

Método de Box-Muller

Se trata de un método utilizado para la generación de normales independientes. Se basa en la siguiente propiedad:

> Dadas $E \sim Exp(1)$ (Exponencial de parámetro $\lambda = 1$) y $U \sim \mathcal{U}(0, 1)$, entonces $X_1 = \sqrt{2E}\cos(2\pi U)$ y $X_2 = \sqrt{2E}\sin(2\pi U)$, son dos normales estándar independientes.

[9] Se pueden ver más detalles y algunas referencia en Fernández-Casal, R. y Cao, R. (2022). *Simulación Estadística*, `https://rubenfcasal.github.io/simbook/`

[10] En la Sección 5.6.2 veremos otro ejemplo de simulación usando este método.

El algoritmo por tanto es muy sencillo de implementar (**Ejercicio**).

La función `rnorm()` que simula valores de la Normal estándar no utiliza este método por defecto[11] pero sí que está incorporado como una posible opción. Para ello habría que evaluar previamente `set.seed` especificando el argumento `normal.kind='Box-Muller')` (o en `RNGkind()`).

5.6. Aplicaciones en Inferencia Estadística

5.6.1. Distribuciones en el muestreo

> *Dada $X_1, \ldots, X_n$ es una muestra aleatoria simple de una población X, un estimador de un parámetro θ ($\theta \in \Theta$) de dicha población es una función de la muestra, $\widehat{\theta}(X_1, \ldots, X_n)$, que no depende de ningún parámetro desconocido y con valores en Θ. Se denomina distribución en el muestreo de $\widehat{\theta}$ a su distribución de probabilidad.*

Para algunos estimadores conocidos es posible obtener analíticamente (de forma exacta) la distribución en el muestreo, por ejemplo sabemos que si la población $X \sim N(\mu, \sigma)$, la distribución en el muestreo de la media muestral es:

$$\bar{X}_n = \frac{1}{n}\sum_{i=1}^{n} X_i \sim \mathcal{N}(\mu, \frac{\sigma}{\sqrt{n}}). \tag{5.2}$$

Como primer ejemplo vamos a aproximar la distribución de $\bar{X}_n$ por simulación y a confirmar el resultado anterior.

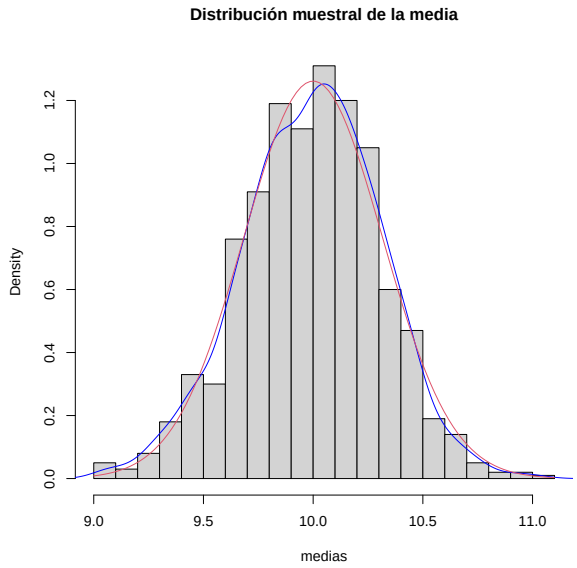
Ejemplo 1 (Distribución de la media muestral)

Asumimos una población $X \sim N(\mu, \sigma)$ con $\mu = 10$ y $\sigma = 1$. Consideramos la media muestral, $\bar{X}_n$ basada en una muestra de tamaño $n = 10$. Para aproximar la distribución muestral de $\bar{X}_n$ vamos a simular `nsim=1000` muestras de la población de tamaño $n = 10$ y a calcular desde cada una de ellas la media muestral. Con esto tendremos `nsim=1000` valores aleatorios de la media muestral que nos permitirán aproximar su distribución y representarla a través de un histograma o una densidad suavizada. El resultado debe confirmar la distribución exacta dada en (5.2), que en este caso sería $\bar{X}_n \sim \mathcal{N}(10, 1/\sqrt{10})$.

```
> mu<-10
> sigma<-1
> n<-10  # tamaño de la muestra
> nsim<-1000  # número de simulaciones
> # simulamos nsim=1000 muestras de tamaño n=10
> # y las almacenamos por filas en una matriz (nsim*n)
```

[11] Si no el de la transformación inversa con una aproximación muy buena de la función de distribución.

```
> set.seed(1)
> muestras<-matrix(rnorm(nsim*n,mean=mu,sd=sigma), ncol=n, nrow=nsim)
>
> # A partir de cada muestra calculamos la media muestral
> medias<-rowMeans(muestras)
> # medias contiene los nsim=1000 valores simulados de la media muestral
> # un histograma de estos valores da una aproximación de la distribución muestral
> hist(medias,breaks=20,freq=FALSE,main='Distribución muestral de la media')
> # superponemos la densidad suavizada
> lines(density(medias),col='blue')
> # ahora la distribución exacta (N(mu, sigma/sqrt(n)))
> curve(dnorm(x,mean=mu,sd=sigma/sqrt(n)),col=2,add=TRUE)
```



Vemos que la aproximación es bastante buena ya que la densidad suavizada está muy cerca de la densidad exacta. Si incrementamos el número de simulaciones nos acercaremos más a la exacta. Prueba por ejemplo con `nsim`=10^5 simulaciones.

Ejercicio: El Teorema Central del Límite nos dice que la distribución (5.2) se verifica también aunque la población X no sea Normal, si el tamaño muestral $n \to \infty$. Repite la simulación hecha en el ejemplo cambiando la población desde la que se generan las muestras a cada una de las siguientes distribuciones:

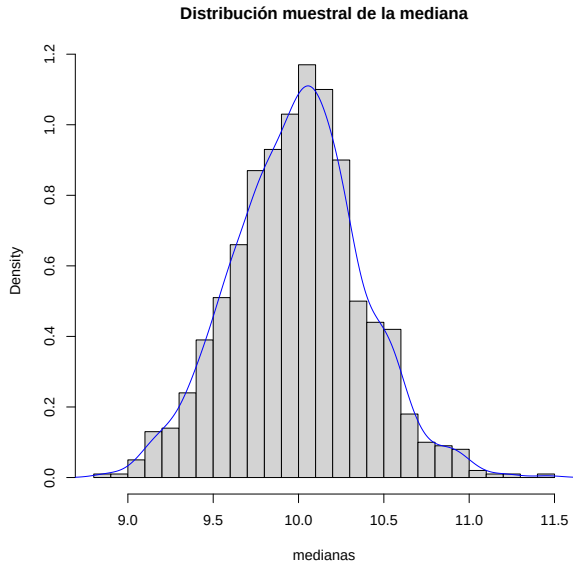
1. $X \sim$ Exponencial de parámetro $\lambda = 0.1$ ($\mathbb{E}[X] = 1/\lambda$, $\mathbb{V}(X) = 1/\lambda^2$).
2. $X \sim \mathcal{U}(10, 30)$ ($\mathbb{E}[X] = 10$, $\mathbb{V}(X) = (30-10)^2/12$).
3. $X \sim Pois(10)$ ($\mathbb{E}[X] = \mathbb{V}(X) = 10$).

Considera primero el caso de muestras de tamaño $n = 10$ como hicimos en el ejemplo, y luego aumenta a $n = 50$ y después a $n = 1000$.

Ejemplo 2 (Distribución de la mediana)

Ahora vamos a aproximar la distribución de la mediana. Suponemos de nuevo que la población es Normal, $X \sim N(\mu, \sigma)$, con $\mu = 10$ y $\sigma = 1$, y consideramos como estimador de la mediana basada en muestras de tamaño $n = 10$. En este caso no tenemos la distribución exacta[12] por lo que sólo nos quedaremos con la aproximación que nos da la simulación.

```
> mu<-10
> sigma<-1
> n<-10  # tamaño de la muestra
> nsim<-1000  # número de simulaciones
> # simulamos nsim=1000 muestras de tamaño n=10
> # y las almacenamos por filas en una matriz (nsim*n)
> set.seed(1)
> muestras<-matrix(rnorm(nsim*n,mean=mu,sd=sigma), ncol=n, nrow=nsim)
>
> # A partir de cada muestra calculamos la medianal
> medianas<-apply(muestras,1,median)
> # medianas contiene los nsim=1000 valores simulados de la media muestral
> # un histograma de estos valores nos da una aproximación de la distribución muestral
> hist(medianas,breaks=20,freq=FALSE,main='Distribución muestral de la mediana')
> # superponemos la densidad suavizada
> lines(density(medianas),col='blue')
```



[12] Si bien se puede probar que es aproximadamente Normal.

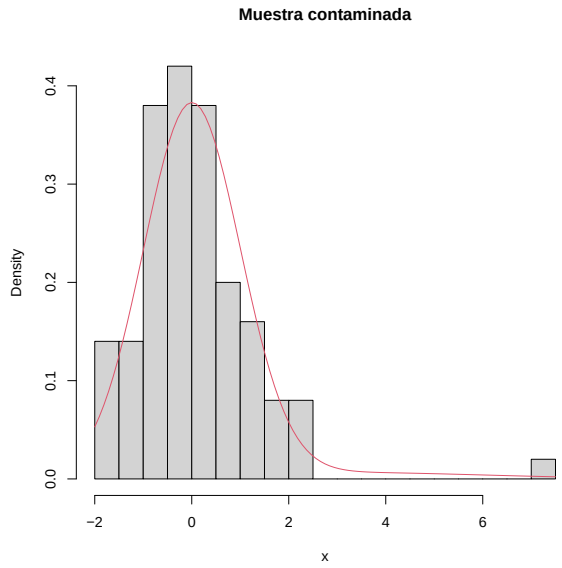
Ejercicio: Simula la distribución muestral del primer cuartil. Considera de nuevo que la población es Normal, $X \sim N(\mu, \sigma)$, con $\mu = 10$ y $\sigma = 1$, y muestras de tamaño $n = 10$.

5.6.2. Comparación de estimadores

Usando simulación podemos también comparar dos estimadores para un mismo parámetro. Vamos a hacerlo por ejemplo comparando los estimadores media y mediana. Sabemos que la mediana es más robusta que la media, ya que esta última es muy sensible a valores anómalos. Vamos a ilustrar esta situación.

Consideramos como población X una Normal estándar de la que se obtienen muestras "contaminadas" con valores anómalos. Esto lo podemos hacer simulando las muestras desde una función de densidad definida como una mixtura de dos densidades, $f(x) = w_1 f_1(x) + w_2 f_2(x)$, donde f_1 corresponde a una $\mathcal{N}(0, 1)$, con probabilidad $w_1 = 0.95$ y, f_2 a una $\mathcal{N}(3, 3)$ con probabilidad $w_2 = 0.05$. Para generar muestras desde esta densidad podemos usar el método de composición que describimos en la sección 5.5.3. Por ejemplo generamos a continuación una muestra de tamaño $n = 100$:

```
> n<-100
> set.seed(1)
> j<-rbinom(n,1,0.05)  # 1's y 0's indicando si es f2 o f1, respectivamente
> x<-rnorm(n, 3*j, 1+2*j)  # esto genera n valores de f1 o f2 dependiendo de j
> # representamos un histograma de la muestra generada
> hist(x,breaks='FD',freq=FALSE,main='Muestra contaminada')
> # superponemos la densidad desde la que se generó
> curve(0.95*dnorm(x,0,1)+0.05*dnorm(x,3,3),add=TRUE,col=2)
```

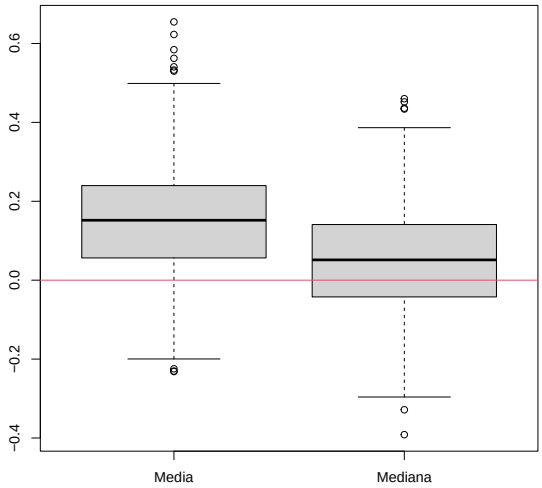


Podemos observar la presencia de datos anómalos en la muestra. Vamos a ver cómo afectan a la media muestral y la mediana como estimadores de $\mu = 0$. Para ello empezamos calculando la media y la mediana de la muestra generada:

```
> mean(x)

[1] 0.0652462

> median(x)

[1] -0.08687757
```

Como estimadores de $\mu = 0$ ambos parecen comportarse de manera similar en esta muestra. No obstante esto no significa que ocurra siempre ya que estos valores dependen de la muestra. Por ello vamos a estudiar las propiedades de los dos estimadores comparando su distribución muestral. Simulamos para ello `nsim=1000` muestras como hicimos en ejemplos anteriores, y desde cada una de ellas calculamos los dos estimadores. Los `nsim=1000` valores generados de cada estimador nos dan una aproximación de su distribución muestral.

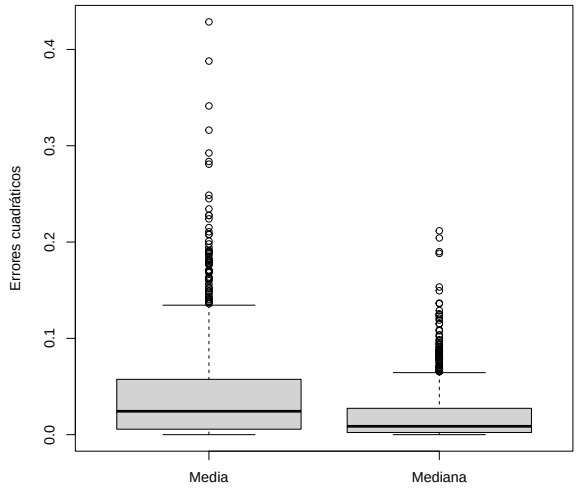
```
> n<-100
> set.seed(1)
> j<-rbinom(n*nsim,1,0.05)
> muestras<-matrix(rnorm(n*nsim, 3*j, 1+2*j), nrow=nsim, ncol=n)
> # cada fila de la matriz 'muestras' es una muestra de tamaño n
>
> # calculamos los estimadores
> medias<-apply(muestras,1,mean)
> medianas<-apply(muestras,1,median)
>
> # comparamos la distribución muestral usando un boxplot
> boxplot(medias,medianas,names=c('Media','Mediana'))
> # una línea horizontal indicando el valor a estimar (mu=0)
> abline(h=0,col=2)
```



El gráfico anterior nos muestra que la media se ve fuertemente afectada por la presencia de valores anómalos, en este caso en el lado derecho de la distribución.

Para complementar la comparación podemos también aproximar el error cuadrático medio de los estimadores, $\mathbb{E}[(\widehat{\theta} - \theta)^2]$ (en nuestro caso $\theta = \mu = 0$):

```
> # errores cuadráticos medios
> ecm.media<-mean((medias-0)^2)
> ecm.media

[1] 0.0423613

> ecm.mediana<-mean((medianas-0)^2)
> ecm.mediana

[1] 0.01990137

> # boxplot de las desviaciones al cuadrado
> boxplot(medias^2,medianas^2,ylab='Errores cuadráticos',
+         names=c('Media','Mediana'))
```



Los resultados anteriores confirman que la mediana es mejor estimador que la media en presencia de datos anómalos.