Cómputo Estadístico

Maestría en Análisis Estadístico y Computación

Programa INEGI-CIMAT
Enero-Mayo 2021

Clase 1, viernes 15 de enero, 2021

Programa del curso

Cómputo Estadístico

- **Descripción**: En este curso, se mostrarán los conceptos básicos de cómputo y programación, así como la teoría necesaria para la aplicación de métodos computacionalmente intensivos. Se revisarán metodologías para la imputación de datos. Se hará especial énfasis en la programación y uso de software, así como en aplicaciones de interés.
- **Objetivos generales de la asignatura**: Proporcionar las bases computacionales que sustentan a las principales aplicaciones de los modelos estadísticos, con un enfoque moderno, haciendo uso de algoritmos computacionales intensivos.

Temas y subtemas

I. Generación de variables aleatorias
   a. Números pseudo aleatorios
   b. Métodos congruenciales
   c. Evaluación de métodos
   d. Generación de v.a. continuas: métodos de inversión y aceptación/rechazo
   e. Generación de v.a. discretas
   f. Simulación

II. Métodos computacionalmente intensivos
   a. Algoritmo EM y aplicaciones
   b. Métodos de remuestreo, Bootstrap y aplicaciones
   c. Markov Chain Monte Carlo y aplicaciones
   d. Gibbs Sampling y aplicaciones

Temas y subtemas

III. Métodos de imputación de datos
   a. Métodos basados en regresión y análisis de covarianza
   b. Métodos basados en el algoritmo EM
   c. Imputación Bayesiana
   d. Métodos basados en técnicas de Machine Learning

IV. Redes bayesianas
   a. Axiomas de probabilidad, Probabilidad conjunta, marginal y condicional, independencia
   b. Teorema de Bayes, Teorema de Bayes con normalización, Redes bayesianas
   c. Inferencia en redes Bayesianas, O-ruidoso (Noisy-OR)
   d. Redes Bayesianas temporales: filtro de Kalman, su extendido y filtro de partículas

Aprendizaje y evaluación

- Actividades de aprendizaje
  - Clases
  - Sesiones de ayudantías
  - Laboratorios de cómputo
  - Individuales: tareas, estudio
- Criterios y procedimientos de evaluación y acreditación
  - Examen parcial 1, 40%
  - Examen parcial 2, 40%
  - Tareas, 20%
- Instructores:
  - Dr. Rogelio Ramos Quiroga (`rramosq@cimat.mx`)
  - Dr. José Benito Hernández Chaudary (`jose.chaudary@cimat.mx`)
- Asistentes del curso:
  - Orlando de Jesús Uc Kantum (`orlando.uc@cimat.mx`)
  - Enrique Eduardo Cortés Montes (`enrique.cortes@cimat.mx`)

Bibliografía

1. Gentle, J.E. (2003). Random Number Generation and Monte Carlo Methods (2nd Edition). Springer
2. Rao, C.R. (1993). Handbook of Statistics 9, Computational Statistics. North-Holland
3. Ross, S.M. (2013). Simulation (5th Edition). Academic Press
4. Gelman, A., Carlin, J.B., Stern, H.S., Dunson, D.B., Vehtari, A. & Rubin, D.B. (2014). Bayesian Data Analysis (3rd Edition). CRC Press
5. Russell, S. & Norvig, P. (2010). Artificial Intelligence: A Modern Approach (3rd Edition). Prentice Hall

I. Generación de Variables Aleatorias

I.a. Números pseudo aleatorios

Números pseudo aleatorios

- La generación de números aleatorios tiene muy variadas aplicaciones, desde en cosas mundanas como loterías, hasta en cosas no triviales como la aleatorización en estudios clínicos o la selección de muestras de un marco muestral para realizar una encuesta.
- La característica escencial de los generadores es que son secuencias no predecibles de números, esto es, secuencias "aleatorias".
- Los generadores genuinamente aleatorios, son difíciles de encontrar en la naturaleza, sin embargo se han usado mediciones de fenómenos físicos, por ejemplo, lecturas de decaimiento radioactivo, o también, lecturas de ruido térmico.
- Las secuencias producidas por fenómenos físicos son no reproducibles, lo cual, para la mayoría de las aplicaciones en no deseable.

Algoritmos computacionales

- La inmensa mayoría de aplicaciones que requieren números aleatorios usan algoritmos computacionales. Formalmente, las secuencias no son genuinamente aleatorias, sin embargo, se comportan en forma indistinguible de las aleatorias.
- Las características principales que debe cumplir un generador de números aleatorios uniformes son:
  - Distribución uniforme.
  - Independencia.
  - Repetibilidad y portabilidad.
  - Rapidez computacional.
- Uno de los primeros métodos propuestos fue el método de cuadrados de von Neumann (1951). No es recomendable su uso pero lo presentamos en la siguiente lámina, solo por importancia histórica.

Método de cuadrados de von Neumann

- El método de cuadrados para generar números aleatorios de 4 dígitos consiste en tomar los 4 dígitos centrales del cuadrado del número anterior en la secuencia, por ejemplo, si iniciamos en 9876:

$$9876^2 = 97535376$$
$$5353^2 = 28654609$$
$$6546^2 = 42850116$$
$$8501^2 = 72267001$$
$$2670^2 = 7128900$$
$$1289^2 = 1661521$$
$$6615^2 = 43758225, \text{ etc.}$$

- La secuencia generada es:

$$9876,\ 5353,\ 6546,\ 8501,\ 2670,\ 1289,\ 6615,\ 7582, \cdots$$

von Neumann, implementación en R 1

```
# Método de cuadrados de von Neumann
# con 4 dígitos (no buena idea)
neumann <- function(seed){
  (floor( (seed^2)/10^2 )) %% 10^4 }
RCUAD <- function(n,seed){
  mu  <- rep(0,n)
  for(i in 1:n){
    seed  <- neumann(seed)
    mu[i] <- seed }
  return(mu/10^4)}
seed <- 9876  # cicla al 50-avo
n    <- 80
unif <- RCUAD(n,seed)
```

```
0.5353 0.6546 0.8501 0.2670 0.1289 0.6615 0.7582 0.4867 0.6876 0.2793
0.8008 0.1280 0.6384 0.7554 0.0629 0.3956 0.6499 0.2370 0.6169 0.0565
0.3192 0.1888 0.5645 0.8660 0.9956 0.1219 0.4859 0.6098 0.1856 0.4447
0.7758 0.1865 0.4782 0.8675 0.2556 0.5331 0.4195 0.5980 0.7604 0.8208
0.3712 0.7789 0.6685 0.6892 0.4996 0.9600 0.1600 0.5600 0.3600 0.9600
0.1600 0.5600 0.3600 0.9600 0.1600 0.5600 0.3600 0.9600 0.1600 0.5600
0.3600 0.9600 0.1600 0.5600 0.3600 0.9600 0.1600 0.5600 0.3600 0.9600
0.1600 0.5600 0.3600 0.9600 0.1600 0.5600 0.3600 0.9600 0.1600 0.5600
```

von Neumann, implementación en R 2

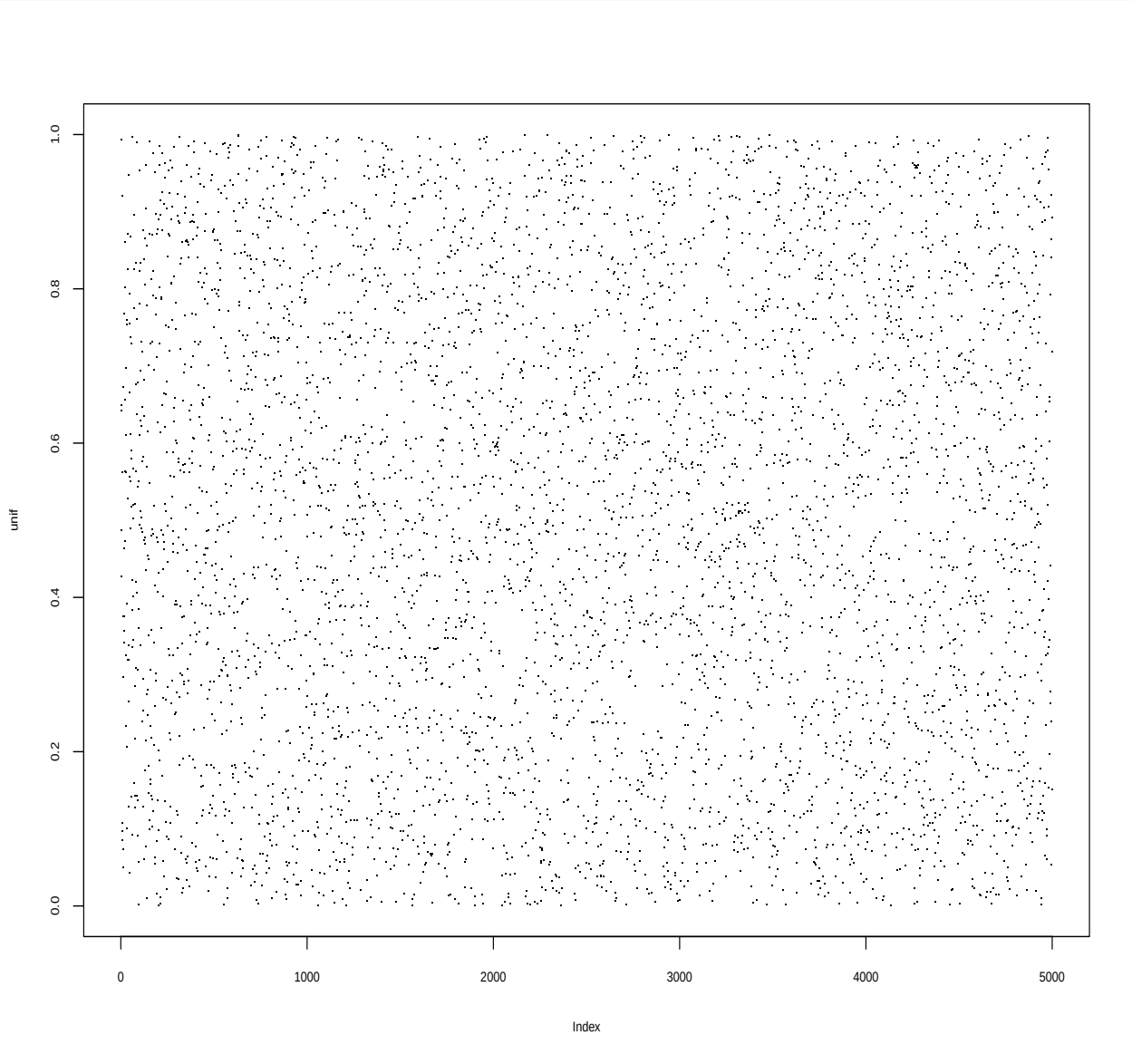
```
# con 10 dígitos
neumann <- function(seed){
  (floor( (seed^2)/10^5 )) %% 10^10 }

RCUAD <- function(n,seed){
  mu  <- rep(0,n)
  for(i in 1:n){
    seed  <- neumann(seed)
    mu[i] <- seed }
  return(mu/10^10)}

n    <- 5000
seed <- 9182736450
unif <- RCUAD(n,seed)

plot(unif,pch=".",cex.axis=.7, cex.lab=.7)
```

von Neumann, se ve bien, aleatorio y uniforme



I.b. Métodos congruenciales

Métodos congruenciales 1

- El método de generadores congruenciales es el más ampliamente usado y conocido. Los generadores están basados en la siguiente fórmula:

$$U_i = (aU_{i-1} + c) \bmod m$$

donde

$$\begin{aligned} U_i &= \text{números enteros pseudo-aleatorios,} \\ U_0 &= \text{valor de inicio o semilla, se escoge al azar,} \\ a, c, m &= \text{constantes que definen al generador.} \end{aligned}$$

- Nota: Recordar que, si h y m son enteros, entonces $h \bmod m$ ("h módulo m") es el residuo de dividir h entre m. En R esta operación es (`h %% m`)

Métodos congruenciales 2

- Note que, para convertir la secuencia generada, $U_0, U_1, \ldots$, en variables uniformes en $(0, 1)$, sólo necesitamos dividirlas entre m, esto es, se usa la secuencia $\{U_i/m\}$.
- Consideremos el funcionamiento del generador con

$$m = 10,\ U_0 = a = c = 7$$

$$\begin{aligned}
U_1 &= (7 \times 7 + 7) \bmod 10 = (56) \bmod 10 = 6 \\
U_2 &= (7 \times 6 + 7) \bmod 10 = (49) \bmod 10 = 9 \\
U_3 &= (7 \times 9 + 7) \bmod 10 = (70) \bmod 10 = 0 \\
U_4 &= (7 \times 0 + 7) \bmod 10 = (7) \bmod 10 = 7 \\
U_5 &= (7 \times 7 + 7) \bmod 10 = (56) \bmod 10 = 6 \\
&\vdots
\end{aligned}$$

- Claramente la secuencia generada 7, 6, 9, 0, 7, 6, 9, 0, 7, ... no es muy aleatoria que digamos!

Comentarios sobre métodos congruenciales 1

- Dado que se utiliza la función "mod m", los posibles valores que produce el algoritmo son los enteros $0, 1, 2, \ldots, m-1$. Lo anterior debido a que, por definición, x mod m es el número que resulta despues de dividir a x entre m.
- Debido que el entero aleatorio U_i depende solamente del entero aleatorio previo U_{i-1}, una vez que se repita el valor previo, la secuencia entera deberá de repetirse. A tal secuencia repetida se le llama ciclo, y su periodo es la longitud del ciclo. La longitud máxima del periodo es m.

Comentarios sobre métodos congruenciales 2

- Si se está interesado en generar variables uniformes $(0, 1)$, la partición más fina del intervalo (0,1) que provee este generador es: $\{0, 1/m, 2/m, \ldots, (m-1)/m\}$. Por supuesto que no es una distribución uniforme $(0, 1)$.
- Los valores de a, c y m determinan la finura con que se hace la partición del intervalo $(0, 1)$, la longitud del ciclo, la uniformidad de la distribución marginal y también afectan la propiedad de independencia de la secuencia que se genera.

Implementación en R 1

- Consideremos el generador congruencial con parámetros

$$a = 2^{16} + 3,\ c = 0,\ m = 2^{31}$$

este generador, conocido como RANDU, fue ampliamante usado en los 70's y 80's pues estaba implementado en el sistema operativo de los sistemas VAX y las máquinas IBM/370. Una implementación en R es como se muestra en las láminas siguientes

Implementación en R 2

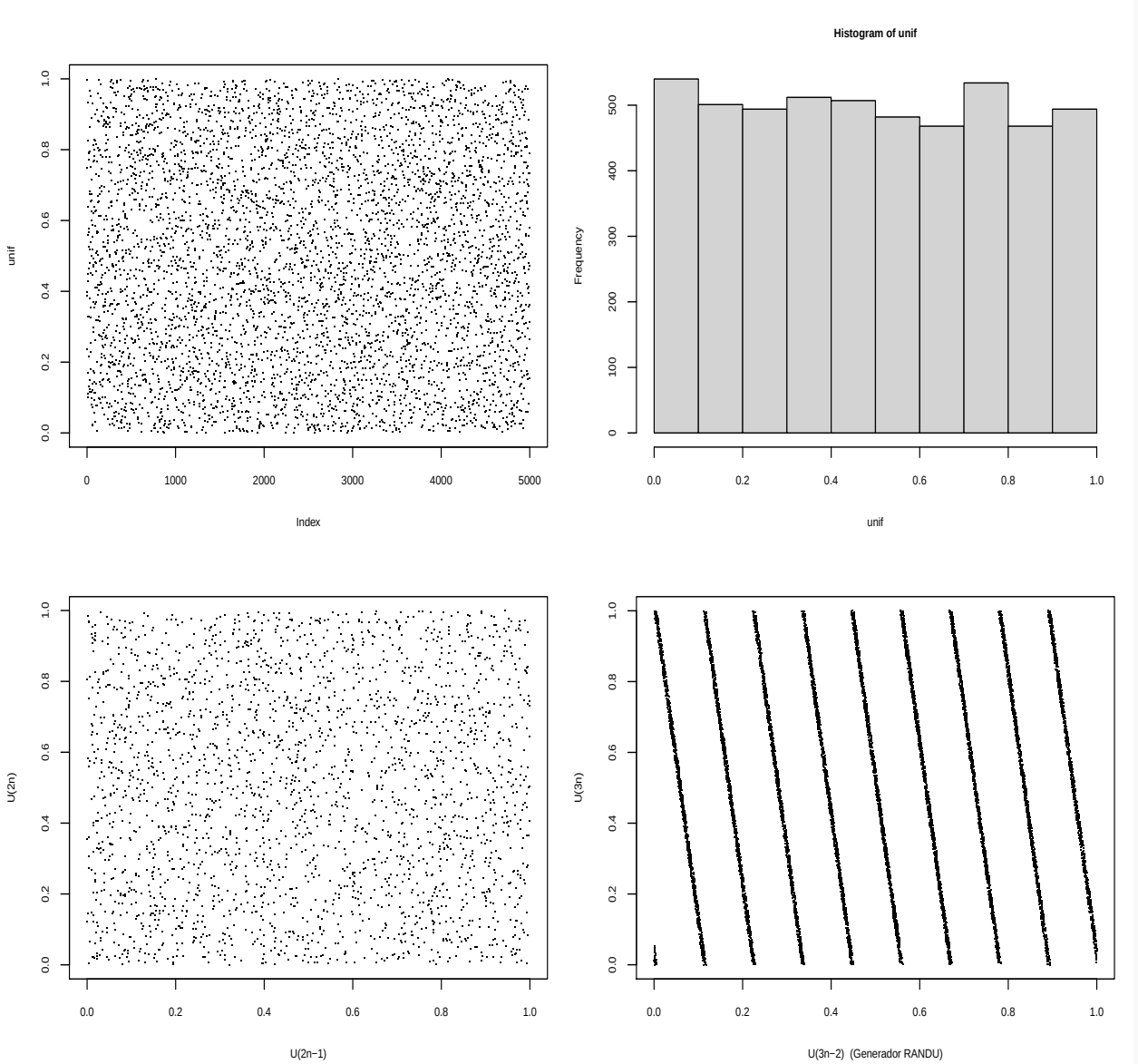
```
randu <- function(seed) (a*seed) %% m

RANDU <- function(n,seed){
  a  <<- 2^16 + 3
  m  <<- 2^31   # solo para ilustrar variables locales -> globales
  mu  <- rep(0,n)
  for(i in 1:n){
    seed  <- randu(seed)
    mu[i] <- seed/m }
  return(list(mues=mu,lastseed=seed))}

# Uso de RANDU, generamos 5000 uniformes (0,1):
n    <- 5000
seed <- 45813
unif <- RANDU(n,seed)$mues
par2 <- matrix(unif,ncol=2,byrow=T)

par(mfrow=c(2,2), mar=c(4, 4, 3, 1) + 0.1)
plot(unif,pch=".",cex.axis=.7, cex.lab=.7)
hist(unif,cex.axis=.7, cex.lab=.7, cex.main=.7)
plot(par2[,1],par2[,2],pch=".",xlab="U(2n-1)", ylab="U(2n)",
     cex.axis=.7, cex.lab=.7)
```

Gráficas de una salida típica de RANDU



Implementación en R 3

- Las primeras tres gráficas en la hoja anterior, muestran el comportamiento de RANDU, y se ven bastante satisfactorias. Sin embargo, la cuarta gráfica muestra un comportamiento extraño no deseable: Si un valor se encuentra entre 0.5 y 0.51, entonces los valores adyacentes están altamente correlacionados. La cuarta gráfica se generó con el siguiente código:

```
seguir <- T
n      <- 60000
seed   <- 45813
sel    <- rep(0,3)
nr     <- 10000

while( seguir ){
  sal  <- RANDU(n,seed)
  rRAN <- matrix(sal$mues,ncol=3,byrow=T)
  sel  <- rbind(sel,rRAN[(rRAN[,2]>.5)&(rRAN[,2]<=.51),])
  if( dim(sel)[1] > nr ) seguir <- F
  seed <- sal$lastseed }

sel <- sel[-1,]

plot(sel[,1],sel[,3],xlab="U(3n-2)  (Generador RANDU)",
     ylab="U(3n)",pch=".", cex.axis=.7, cex.lab=.7)
```

Parámetros recomendados para generadores congruenciales

- Los siguientes parámetros han sido recomendados en la literatura por sus propiedades de uniformidad, independencia y facilidad de implementación. Todos ellos usan módulo $2^{31} - 1$ y tienen ciclos maximales de $2^{31} - 1$ (excepto el primero que tiene un ciclo de $2^{31} - 2$). Para propósitos prácticos $2^{31} - 2 = 2,147,483,646$ debe alcanzar para cualquier simulación.
- Parámetros

$$a = 16,807$$
$$a = 950,706,376$$
$$a = 742,938,285$$
$$a = 1,226,874,159$$
$$a = 62,089,911$$
$$a = 1,343,714,438$$

Implementación

- La implementación en R es muy semejante a la presentada para RANDU.

```
RAND1 <- function(n,seed){
  a  <<- 16807
  m  <<- 2^31 - 1
  mu  <- rep(0,n)
  for(i in 1:n){
    seed  <- randu(seed)
    mu[i] <- seed/m }
  return(list(mues=mu,lastseed=seed))}
# Por ejemplo:
RAND1(100,767)
```

Generador congruencial combinado

- Wichmann y Hill (1982) propusieron una combinación de tres generadores congruenciales

$$X_i = (171 X_{i-1}) \bmod 30,269$$
$$Y_i = (172 Y_{i-1}) \bmod 30,307$$
$$Z_i = (170 Z_{i-1}) \bmod 30,323$$

- Tomando

$$U_i = \left\{ \frac{X_i}{30,269} + \frac{Y_i}{30,307} + \frac{Z_i}{30,323} \right\} \bmod 1$$

- Este generador tiene un ciclo de 6.952×10^{12} (los de las hojas anteriores son del orden de 2×10^9).
- El default en R usa el "Mersenne-Twister", de Matsumoto y Nishimura (1998), el cual tiene un periodo absurdo de $2^{19937} - 1$ (!!).

Implementación

```
randu <- function(seed) (a*seed) %% m

RANWH <- function(n,seed){
  a  <<- c( 171, 172, 170)
  m  <<- c(30269, 30307, 30323)
  mu  <- rep(0,n)
  for(i in 1:n){
    seed  <- randu(seed)
    mu[i] <- (sum(seed/m)) %% 1 }
  return(list(mues=mu,lastseed=seed))}

n    <- 5000
seed <- c(67612,92318,652612)
unif <- RANWH(n,seed)$mues
```

I.c. Evaluación de métodos

Pruebas estadísticas

- Mencionamos que las características principales que debe cumplir un generador de números aleatorios uniformes son:
  - Distribución uniforme
  - Independencia
  - Repetibilidad y portabilidad
  - Rapidez computacional
- En general, los algoritmos lineales congruenciales cumplen bien los dos últimos puntos. Para evaluar las primeras dos propiedades veremos:
  - Bondad de ajuste: Prueba de uniformidad
  - Prueba de aleatoriedad: Prueba de rachas
- En la actualidad, los generadores disponibles en sistemas como R o Python han pasado todas las pruebas y son altamente confiables. Consideraremos estas pruebas por su importancia en la cultura estadística.

Pruebas de bondad de ajuste 1

- Consideremos primero el caso general (distribución no necesariamente uniforme).
- Supongamos datos independientes $x_1, \ldots, x_n$ y queremos investigar que tan congruentes son estos datos con un modelo hipotetizado $f_{\theta_0}(x)$.
- Particionemos el espacio muestral en clases. Calculamos la frecuencia de cada clase y calculamos la frecuencia esperada de cada clase

Clase	A_1	A_2	$\cdots$	A_k	Total
Frec. Obs.	f_1	f_2	$\cdots$	f_k	n
Frec. Esp.	np_1	np_2	$\cdots$	np_k	n

- Las p_j's son calculadas como

$$p_j = P(x \in A_j) = \int_{A_j} f_{\theta_0}(x)dx$$

o estimadas mediante $\tilde{p}_j$ usando el correspondiente máximo verosímil para θ_0.

Pruebas de bondad de ajuste 2

- Consideremos la hipótesis H_0 : datos de $f_{\theta_0}(x)$, la que reformulamos en general como

  $$H_0 : \text{datos con estructura} \quad \text{vs} \quad H_1 : \text{datos sin estructura}$$

  donde "sin estructura" significa que los datos ocurren al azar con $\widehat{p}_j = f_j/n_j$
- El cociente de verosimilitudes es

$$\Lambda = \frac{\tilde{p}_1^{f_1}\ \tilde{p}_2^{f_2}\ \cdots\ \tilde{p}_k^{f_k}}{\widehat{p}_1^{f_1}\ \widehat{p}_2^{f_2}\ \cdots\ \widehat{p}_k^{f_k}} = \prod_{j=1}^{k}\left(\frac{\tilde{p}_j}{\widehat{p}_j}\right)^{f_j} = \prod_{j=1}^{k}\left(\frac{n_j\,\tilde{p}_j}{f_j}\right)^{f_j}$$

- entonces, por propiedades asintóticas del cociente de verosimilitudes:

$$D = -2\log\Lambda = 2\sum_{j=1}^{k}\text{obs}_j\log\left(\frac{\text{obs}_j}{\text{esp}_j}\right) \overset{H_0}{\sim} \chi^2_{k-1-q}$$

  (si θ_0 no requiere ser estimado, entonces $q = 0$).

Pruebas de bondad de ajuste 3

- Una alternativa (equivalente) a la prueba de cociente de verosimilitudes es usar el estadístico de Pearson para bondad de ajuste

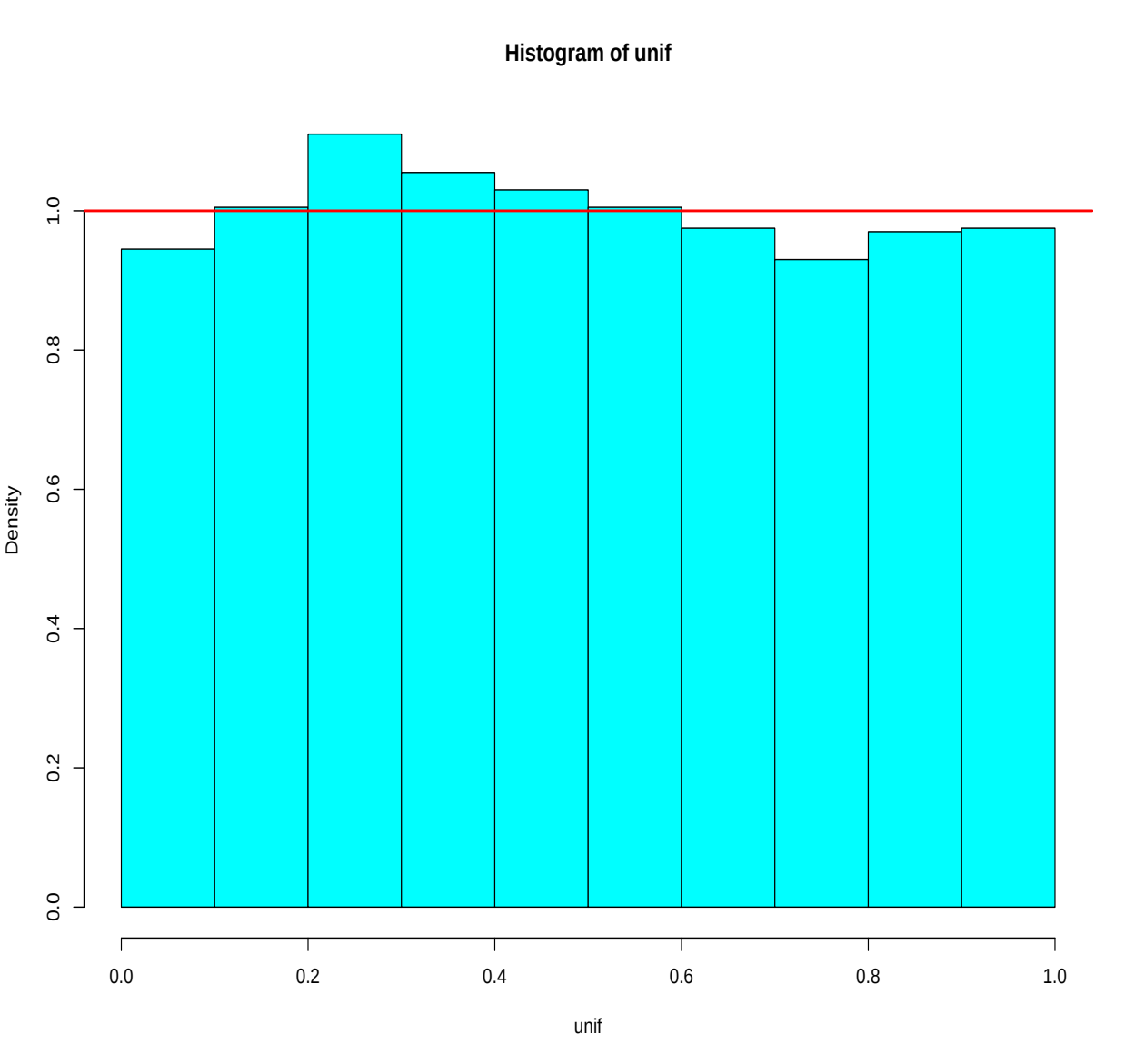
$$D = \sum_{j=1}^{k} \frac{(\text{obs}_j - \text{esp}_j)^2}{\text{esp}_j}$$

  y usamos la misma distribución límite χ^2.
- Su uso para detectar desviaciones de uniformidad, requiere que dividamos $(0, 1)$ en subregiones y contrastemos las frecuencias observadas contra las esperadas. Esto esta sujeto a ciertos niveles de subjetividad, probablemente esto es inevitable y las pruebas no deben tomarse concluyentes en forma individual, si no como parte de un paquete de herramientas de validación, tanto formales como visuales.

Pruebas de bondad de ajuste 4: Ejemplo RANDU

```
# Prueba de uniformidad Ji-Cuadrada
randu <- function(seed) (a*seed) %% m
RANDU <- function(n,seed){
  a  <<- 2^16 + 3
  m  <<- 2^31          # (mismo código de RANDU visto antes)
  mu  <- rep(0,n)
  for(i in 1:n){
    seed  <- randu(seed)
    mu[i] <- seed/m }
  return(list(mues=mu,lastseed=seed))}
# Generamos 2000 uniformes (0,1):
n    <- 2000
seed <- 458
seed <- 45819
unif <- RANDU(n,seed)$mues
Oj   <- hist(unif,breaks=seq(0,1,by=.1),plot=F)$counts
# Ji-cuadrada de Pearson
Ej   <- n/10
P    <- sum( ((Oj-Ej)^2)/Ej )          # [1] 5.23
pv   <- 1-pchisq(P,9)                  # [1] 0.8138153, unif. ok
hist(unif,breaks=seq(0,1,by=.1),col="cyan",freq=F)
abline(h=1,col="red",lwd=2)
```

Histograma de 2000 números generados por RANDU



Clase 2, sábado 16 de enero, 2021

Kolmogorov-Smirnov 1

- La prueba Kolmogorov-Smirnov es usada en Estadística para verificar supuestos distribucionales.
- La idea es simple: La función de distribución empírica es un estimador de la función de distribución verdadera; entonces, si la empírica difiere mucho de la teórica entonces rechazamos que la teórica es la distribución verdadera.
- Los términos que subrayamos implican dos problemas:
  - ¿Cómo medimos discrepancias entre dos funciones?
  - Una vez que cuantificamos la discrepancia, ¿Cómo sabemos si es grande o no?
- Para contestar la primera pregunta tenemos el estadístico D_n de Kolmogorov-Smirnov

$$D_n = \sup_x \mid F_n(x) - F_0(x) \mid$$

donde $F_n(x)$ es la Función de Distribución Empírica y $F_0(x)$ es la Función de Distribución Teórica (o Hipotetizada).

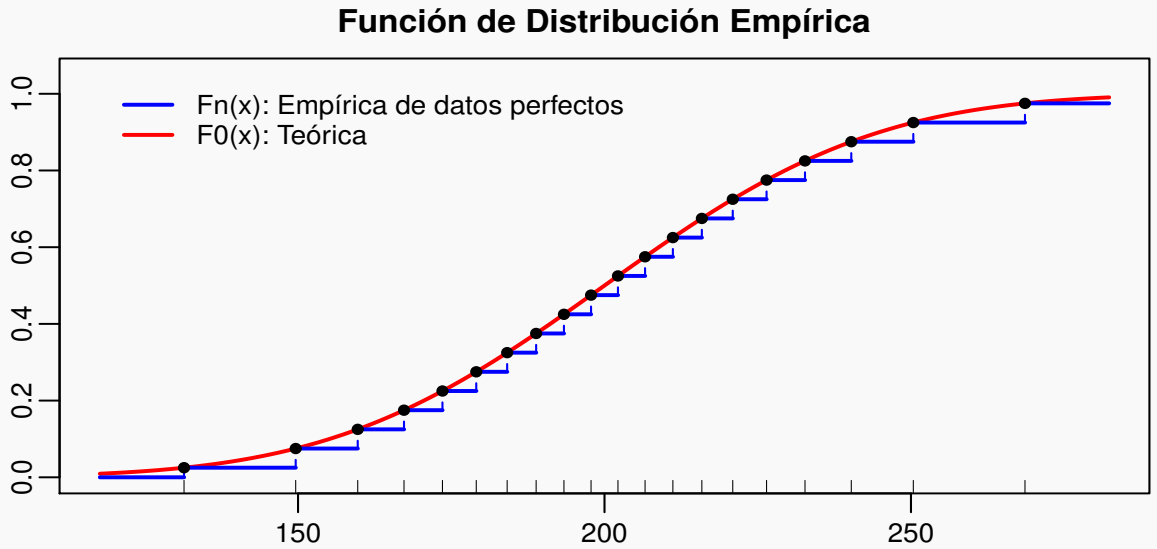
Kolmogorov-Smirnov 2

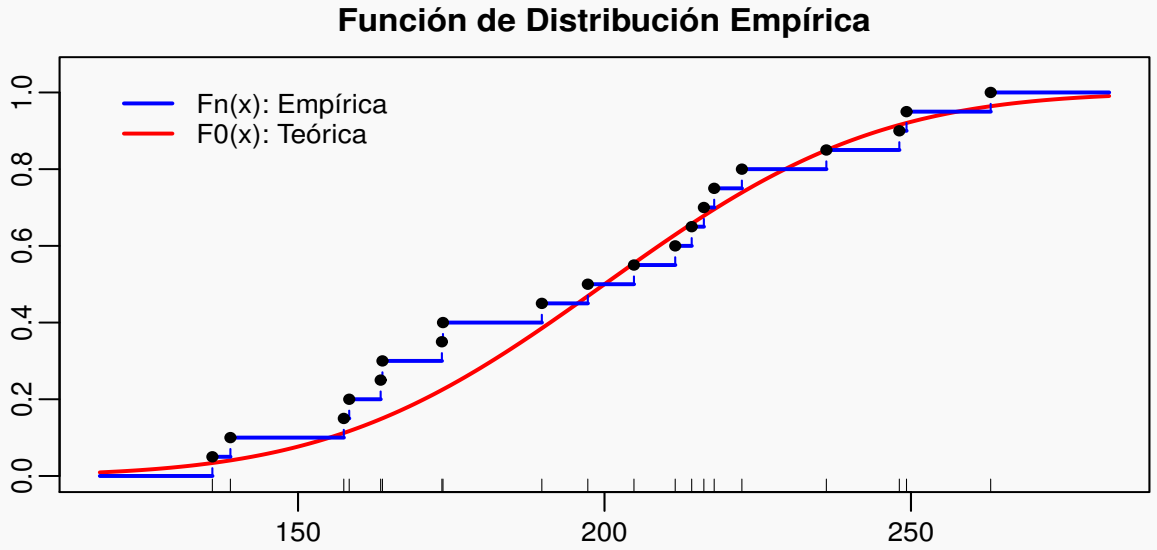
- La función de distribución empírica se define como

$$F_n(x) = \frac{1}{n}\sum_{i=1}^{n} I(x_i \leq x)$$

- En relación a la segunda pregunta, puede mostrarse el resultado (remarcable) que, para el caso continuo, la distribución de D_n no depende de la distribución verdadera $F(x)$; esto es, hay una sola distribución nula de D_n no importando si estamos en el caso Uniforme, Normal, Gama, Weibull, etc.
- La siguiente gráfica ejemplifica la construcción de la Distribución Empírica:

Función de distribución empírica





Código para las gráficas 1

```
par(mfrow=c(2,1),mar=c(2,2,2,2))
n    <- 20; med  <- 200; des  <- 35
delt <- .10
M    <- 200
edf  <- ((1:n)-.5)/n
dat  <- qnorm(edf,mean=med,sd=des)
ran  <- max(dat) - min(dat)
lmin <- min(dat) - delt*ran
lmax <- max(dat) + delt*ran
xx   <- seq(lmin,lmax,length=M)
yy   <- pnorm(xx,mean=med,sd=des)
plot(dat,edf, xlim=c(lmin,lmax), ylim=c(0,1.05), mgp=c(1.5,.5,0),
  xlab="", ylab="", cex.axis=.8, cex.lab=.8,type="n",
  main="Funcion de Distribucion Empirica",cex.main=1)
rug(dat)
lines(xx,yy,col="red",lwd=2)
segments(lmin,0,dat[1],0,col="blue",lwd=2)
for(i in 1:(n-1)){
  segments(dat[i],edf[i],dat[i+1],edf[i],col="blue",lwd=2)}
segments(dat[n],edf[n],lmax,edf[n],col="blue",lwd=2)
segments(dat[1],0,dat[1],edf[1],col="blue",lty=2)
for(i in 2:n){segments(dat[i],edf[i-1],dat[i],edf[i],col="blue",lty=2)}
```

Código para las gráficas 2

```
points(dat,edf,pch=20)
legend(lmin,1.05,legend=c("Fn(x): Empirica de datos perfectos",
  "F0(x): Teorica"), col=c("blue","red"),lwd=2, cex=.8, bty="n")

edf  <- (1:n)/n
set.seed(5963471)
dat  <- sort(rnorm(n,mean=med,sd=des))
xx   <- seq(lmin,lmax,length=M)
yy   <- pnorm(xx,mean=med,sd=des)
plot(dat,edf, xlim=c(lmin,lmax), ylim=c(0,1.05), mgp=c(1.5,.5,0),
  xlab="", ylab="", cex.axis=.8, cex.lab=.8,type="n",
  main="Funcion de Distribucion Empirica",cex.main=1)
rug(dat)
lines(xx,yy,col="red",lwd=2)
segments(lmin,0,dat[1],0,col="blue",lwd=2)
for(i in 1:(n-1)){
  segments(dat[i],edf[i],dat[i+1],edf[i],col="blue",lwd=2) }
segments(dat[n],1,lmax,1,col="blue",lwd=2)
segments(dat[1],0,dat[1],edf[1],col="blue",lty=2)
for(i in 2:n){segments(dat[i],edf[i-1],dat[i],edf[i],col="blue",lty=2)}
points(dat,edf,pch=20)
legend(lmin,1.05,legend=c("Fn(x): Empirica","F0(x): Teorica"),
 col=c("blue","red"),lwd=2, cex=.8, bty="n")
```

Paréntesis: Distribución de D_n, 1

- Consideremos

$$D_n = \sup_x \mid F_n(x) - F_0(x) \mid$$

- La distribución de D_n, no depende de F_0. ¿Por qué?.

$$\begin{aligned}
P(D_n \geq d) &= P\left(\sup_x \mid F_n(x) - F_0(x) \mid \geq d\right) \\
&= P\left(\sup_x \left| \frac{1}{n}\sum_{i=1}^{n} I(x_i \leq x) - F_0(x) \right| \geq d\right) \\
&\overset{(1)}{=} P\left(\sup_x \left| \frac{1}{n}\sum_{i=1}^{n} I\left(F_0(x_i) \leq F_0(x)\right) - F_0(x) \right| \geq d\right) \\
&\overset{(2)}{=} P\left(\sup_x \left| \frac{1}{n}\sum_{i=1}^{n} I\left(U_i \leq F_0(x)\right) - F_0(x) \right| \geq d\right) \\
&\overset{(3)}{=} P\left(\sup_{0\leq y\leq 1} \left| \frac{1}{n}\sum_{i=1}^{n} I\left(U_i \leq y\right) - y \right| \geq d\right)
\end{aligned}$$

Paréntesis: Distribución de D_n, 2

- Nota (1): F_0 es monótona creciente, entonces $x_i \leq x$ si y sólo si $F_0(x_i) \leq F_0(x)$.
- Nota (2): Si $x_i \sim F_0$, entonces $F_0(x_i)$ es Unifome en $(0, 1)$

$$P(F_0(x_i) \leq w) = P(x_i \leq F_0^{-1}(w)) = F_0(F_0^{-1}(w)) = w$$

- Nota (3): Si $y = F_0(x)$, entonces, conforme x varía en todos los reales, y varía entre 0 y 1.
- La última igualdad en la lámina anterior muestra que, efectivamente, la distribución de D_n no depende de F_0.
- Esta observación puede explotarse pues podemos aproximar p-valores usando simulación. Si la distancia observada de Kolmogorov-Smirnov nos da d, entonces simulando muchas veces uniformes $U_1, \ldots, U_n$ podemos calcular la frecuencia con que la cantidad

$$(*) = \sup_{0 \leq y \leq 1} \left| \frac{1}{n} \sum_{i=1}^{n} I\left(U_i \leq y\right) - y \right|$$

sobrepasa a d, y con esto tenemos una aproximación al p-valor.

Paréntesis: Distribución de D_n, 3

- El cálculo mencionado en la hoja anterior puede simplificarse usando la siguiente equivalencia

$$(*) = \max \left\{ \frac{j}{n} - U_{(j)}, U_{(j)} - \frac{j-1}{n}, \;\; j = 1, \ldots, n \right\}$$

- Por ejemplo, supongamos un valor observado de $D_n = 0.10369$, ¿cuál es el p-valor corresondiente?

```
# Simulación de Dn para obtener p-valores
set.seed(656)
n  <- 20
d  <- 0.10369
M  <- 10000
j  <- 1:n
Dp <- rep(0,M)
for(i in 1:M){
  u <- sort(runif(n))
  Dp[i] <- ( max( cbind(j/n-u,u-(j-1)/n) ) >= d ) }
pvalor <- mean(Dp)
pvalor #  0.9668
```

Kolmogorov-Smirnov 3

- Ahora bien,¿Cómo se efectúa, por ejemplo, una prueba de Normalidad?. Para ello, calculamos el valor del estadístico D_n de Kolmogorov-Smirnov y vemos si es improbablemente grande.
- Veamos un ejemplo: Los siguientes datos son 20 registros de pesos (en gramos) de pollitos

156	162	168	182	186	190	190	196	202	210
214	220	226	230	230	236	236	242	246	270

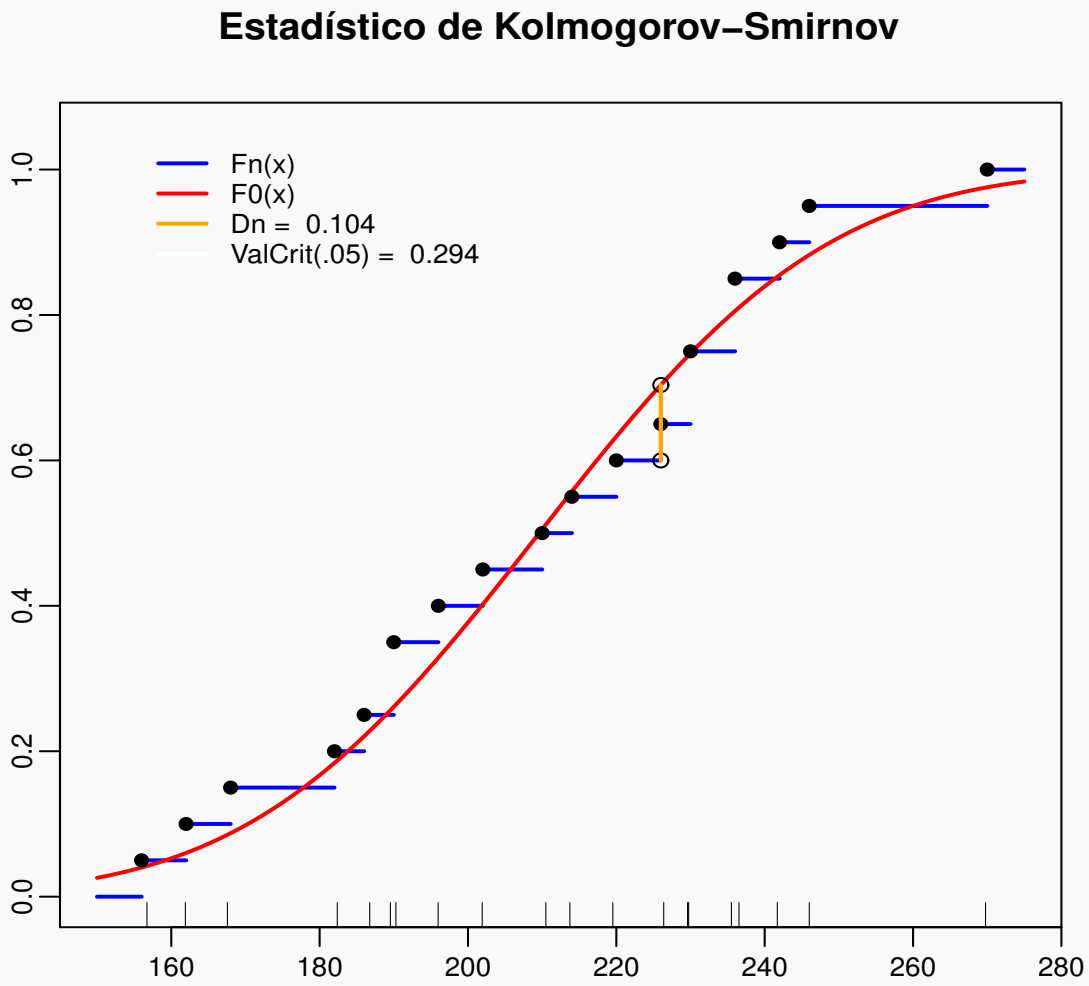
- Deseamos probar la hipótesis

$$H_0 : \text{Los datos provienen de: } N(\mu, \sigma^2)$$

donde $\mu = 209.6$ y $\sigma = 30.65$ (es crucial saber de dónde vienen estos valores, pero, por lo pronto, supongamos que estos son los valores en los que estamos interesados).

- Calculamos primero el valor de D_n y lo comparamos contra el valor de un cuantil de la distribución nula de D_n, como se muestra en la siguiente gráfica:

Bondad de ajuste a normalidad



Kolmogorov-Smirnov 4

- Como el valor de $D_n = 0.104$ no sobrepasa al valor crítico $D_{n,.05} = 0.294$, entonces no rechazamos la aseveración de que los datos provienen de una distribución normal.
- (En Estadística así es como nos toca vivir: No estamos diciendo que los datos son Normales, simplemente decimos que es razonablemente aceptable considerarlos normales pues no hay evidencia de lo contrario... :)
- El código R usado para obtener la gráfica anterior se muestra enseguida.

Código R 1

```
# Prueba de normalidad. Pesos (en gramos) de 20 pollitos
vc05 <- 0.2940701          # New Cambridge Statistical Tables
dat  <- sort(c(156,162,168,182,186,190,190,196,202,210,
               214,220,226,230,230,236,236,242,246,270))
datu <- unique(dat)
n    <- length(dat)
nu   <- length(datu)
aa   <- table(dat)
edf  <- cumsum(aa)/n
edf0 <- c(0,edf[-nu])
xx   <- seq(150,275,length=200)
yy   <- pnorm(xx,mean=200,sd=35)
ye   <- pnorm(datu,mean=200,sd=35)
yy   <- pnorm(xx,mean=mean(dat),sd=sd(dat))
# usando estimaciones, los valores criticos no válidos
ye   <- pnorm(datu,mean=mean(dat),sd=sd(dat))
Dn   <- max(max(abs(ye-edf)),max(abs(ye-edf0)))  #  0.1037224
ii   <- which(Dn == pmax(abs(ye-edf),abs(ye-edf0)))[1]
if( abs(ye[ii]-edf[ii])==Dn ){
 yD <- edf[ii] }else{
 yD <- edf0[ii] }
```

Código R 2

```
plot(datu,edf, xlim=c(150,275), ylim=c(0,1.05), mgp=c(1.5,.5,0),
  xlab="", ylab="", cex.axis=.8, cex.lab=.8,type="n",
  main="Estadistico de Kolmogorov-Smirnov")
rug(jitter(dat))
segments(150,0,datu[1],0,col="blue",lwd=2)
for(i in 1:(nu-1)){
 segments(datu[i],edf[i],datu[i+1],edf[i],col="blue",lwd=2)}
segments(datu[nu],1,275,1,col="blue",lwd=2)
lines(xx,yy,col="red",lwd=2)
points(datu,edf,pch=16)
points(c(datu[ii],datu[ii]),c(ye[ii],yD))
segments(datu[ii],ye[ii],datu[ii],yD,lwd=2,col="orange")

legend(155,1.05,legend=c("Fn(x)","F0(x)",paste("Dn = ",round(Dn,3)),
 paste("ValCrit(.05) = ",round(vc05,3))),
 col=c("blue","red","orange","white"),lwd=2, cex=.8, bty="n")
```

Kolmogorov-Smirnov 5

- Si bien es cierto que la distribución de D_n no depende de la distribución verdadera $F(x)$, en la práctica uno debe especificar completamente todos los parámetros que definen la distribución nula.
- Usualmente lo que se hace es estimar los parámetros en base a los datos; por ejemplo, con los datos anteriores calculamos $\bar{x} = 209.6$ y $s = 30.65$ y luego estos valores los tomamos como μ y σ respectivamente.
- Sin embargo, es sabido que estimar los parámetros para definir la distribución nula altera sus propiedades (esto es, la distribución tabulada no es la correcta).
- Al uso de estimación de parámetros para especificar la nula y usar los cuantiles correctos de D_n se le llama "Prueba de Lilliefors". Es fácil encontrar en internet tablas para esta prueba, aunque también es fácil simular en R la distribución del estadístico de Lilliefors, como se muestra enseguida.

Prueba de Lilliefors

```
# Lilliefors
M  <- 50000
Dn <- rep(0,M)
n  <- 20
y  <- (1:n)/n
y0 <- y - 1/n

for(i in 1:M){
 dat   <- sort(rnorm(n))
 yt    <- pnorm(dat,mean=mean(dat),sd=sd(dat))
 Dn[i] <- max(max(abs(yt-y)),max(abs(yt-y0))) }

quantile(Dn,p=c(.8,.85,.9,.95,.99))

      80%       85%       90%       95%       99%
0.1586585 0.1661773 0.1764335 0.1918497 0.2229242
```

- El valor crítico correcto es $D_{n,.05} = 0.19185$, así que la conclusión de que los datos de pesos de pollitos son razonablemente normales no cambia.

Prueba K-S en R

```
library(dgof)
datj <- ifelse(dat==190,c(189,191),dat)   # prueba no permite empates
datj <- ifelse(datj==230,c(229,231),datj)
datj <- ifelse(datj==236,c(235,237),datj)

ks.test(datj,"pnorm",mean(datj),sd(datj))

#         One-sample Kolmogorov-Smirnov test
# data:  datj
# D = 0.10369, p-value = 0.9672
# alternative hypothesis: two-sided
```

Pruebas de rachas 1

- Considere la siguiente secuencia de símbolos

$$A \quad A \quad B \quad B \quad B \quad A \quad B \quad B \quad B \quad B \quad A \quad A \quad B \quad A$$

diremos que, en esta secuencia, hay 7 "rachas":

$$\underline{A \quad A} \quad \underline{B \quad B \quad B} \quad \underline{A} \quad \underline{B \quad B \quad B \quad B} \quad \underline{A \quad A} \quad \underline{B} \quad \underline{A}$$

- El número de rachas en una secuencia nos da un indicador del grado de aleatoriedad en el que los símbolos A's y B's se encuentran ordenados.
- Muchas rachas o muy pocas rachas nos dirían que los símbolos **no** aparecen al azar; por ejemplo, 13 y 2 rachas en, respectivamente:

$$\underline{B} \quad \underline{A} \quad \underline{B} \quad \underline{A} \quad \underline{B} \quad \underline{A} \quad \underline{B} \quad \underline{A} \quad \underline{B} \quad \underline{A} \quad \underline{B} \quad \underline{A} \quad \underline{B \quad B}$$

y

$$\underline{A \quad A \quad A \quad A \quad A \quad A} \quad \underline{B \quad B \quad B \quad B \quad B \quad B \quad B \quad B}$$

Pruebas de rachas 2

- Considere las hipótesis

$$H_0 : \text{Los símbolos están en orden aleatorio} \quad \text{vs}$$
$$H_1 : \text{Los símbolos } \textbf{no} \text{ están en orden aleatorio}$$

  y considere el estadístico de prueba $r = \#$ de rachas.
- ¿Cómo podemos encontrar una región de rechazo para H_0 correspondiente a un nivel de significancia $\alpha = .05$?
- Algunas posibles regiones de rechazo son, por ejemplo, $R = \{2\}, \quad R = \{13\}, \quad R = \{2, 13\}$, $R = \{2, 3, 12, 13\}$ etc.; la cuestión es, ¿Cuál tiene el tamaño más cercano a .05?, en otras palabras, ¿Cuál región es tal que $P(r \in R) \approx .05$? (la probabilidad se calcula bajo el supuesto de que H_0 es verdadera).
- Para este problema es claro que necesitamos saber la distribución de r cuando los símbolos están en orden aleatorio.

Pruebas de rachas 3

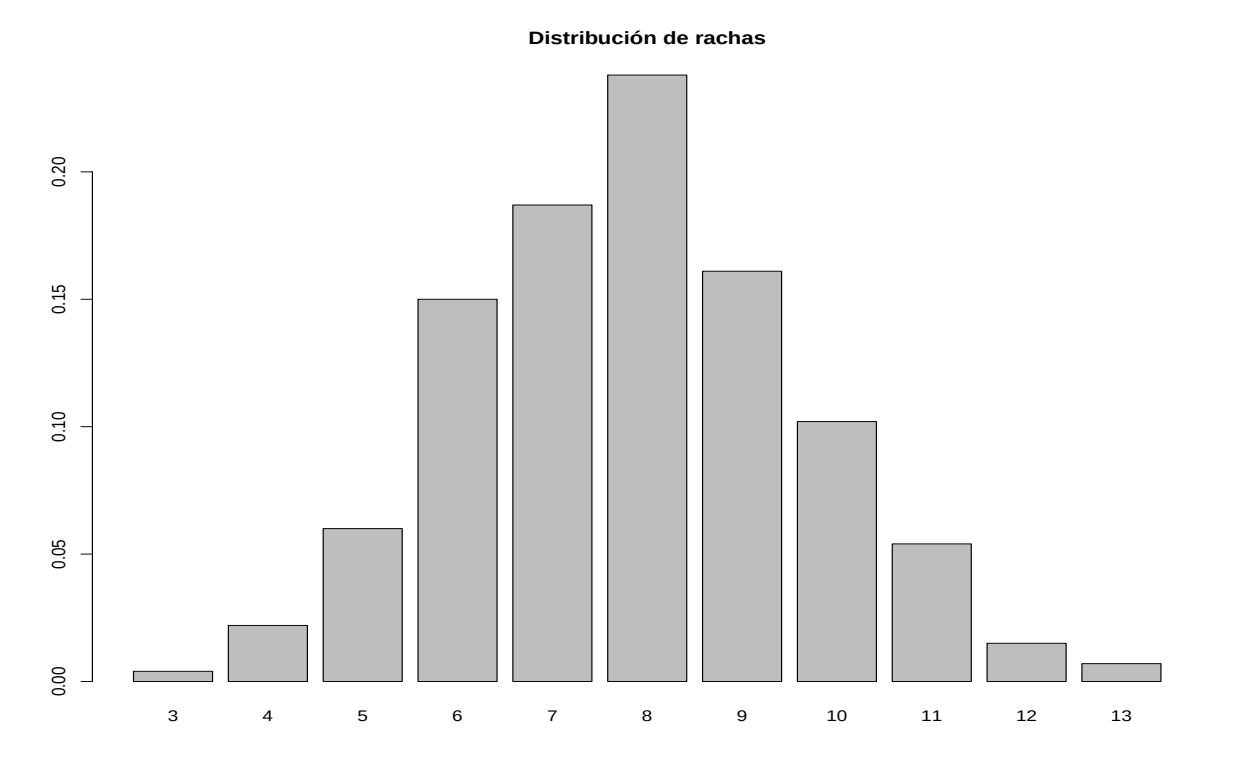
- Con el fin de conocer la distribución de r, usamos simulación. Por ejemplo, podemos hacer:

```
M  <- 1000
r  <- rep(0,M)          # M debe ser grande
n  <- 14
for( i in 1:M){
  orden <- sample( c(0,0,0,0,0,0,1,1,1,1,1,1,1,1) )
  r[i]  <- cuenta( orden ) }
```

- donde cuenta es una función que cuenta las rachas (esta es la parte laboriosa). Al final `r` es un vector de longitud `M` y con ello nos puede dar una buena idea de su distribución; la región de rechazo se obtendría de las colas de esa distribución.

```
cuenta <- function(a){
  run <- 1
  for(i in 2:n){
    if( a[i] == a[i-1] ){next}
    run <- run + 1}
  return(run)}
barplot( table(r) / M , main="Distribución de rachas")
```

Distribución de rachas




```
obs <- c(0,0,1,1,1,0,1,1,1,1,0,0,1,0)
cuenta(obs) # 7
# 7 rachas es un resultado probable => no rechazamos orden aleatorio
```

Prueba de rachas en paquete de R

```
obs <- c(0,0,1,1,1,0,1,1,1,1,0,0,1,0)
library(snpar)
runs.test(obs, exact=TRUE, alternative=c("two.sided","less","greater"))
#         Exact runs test
# data:  obs
# Runs = 7, p-value = 0.8252
# alternative hypothesis: two.sided
```

- Para determinar si la salida de un generador produce resultados con características de "aletoriedad", podemos definr una racha como el número de dígitos consecutivos ordenados en forma creciente, luego otra racha formada por los que están en forma decreciente, y así sucesivamente.
- Por ejemplo, en la secuencia

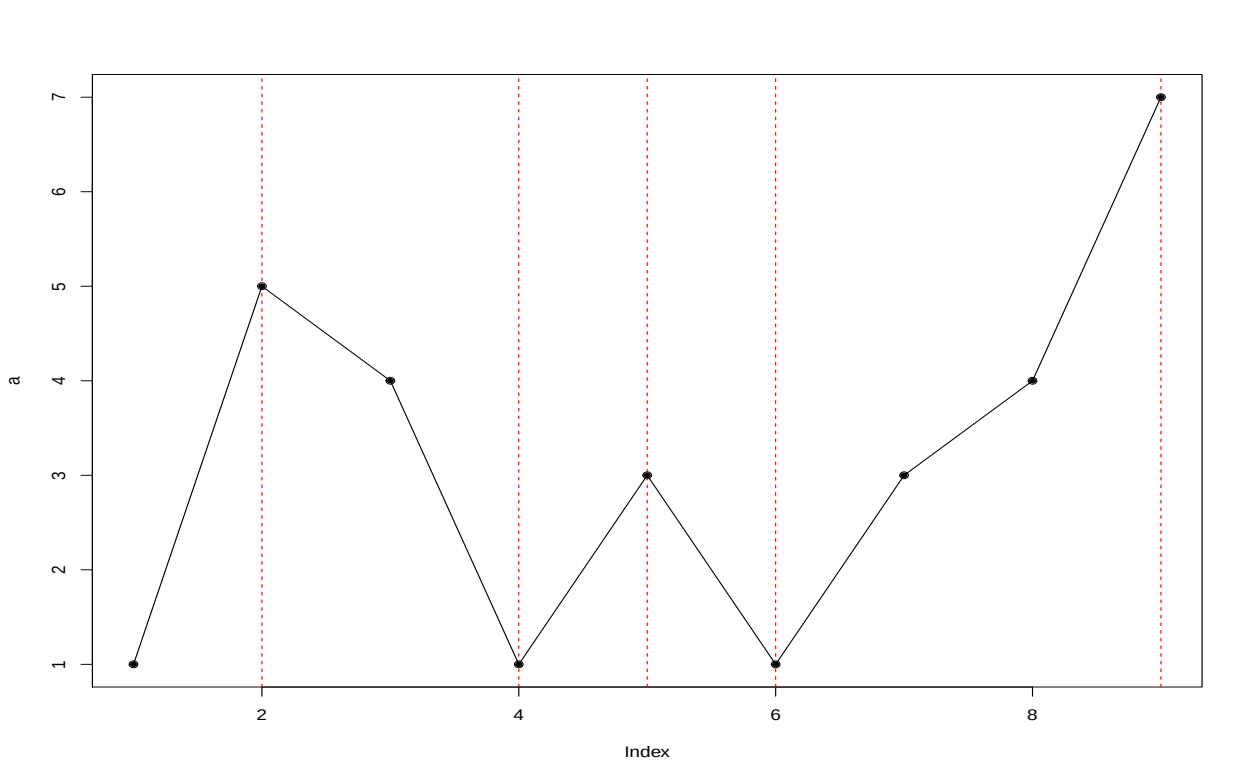
$$1, 5, 4, 1, 3, 1, 3, 4, 7$$

identificamos 5 rachas

$$15, 41, 3, 1, 347$$

Rachas y aleatoriedad

```
a <- c(1,5,4,1,3,1,3,4,7)
plot(a,type="l")
points(a,pch=20)
```



- Declaramos falta de aleatoriedad en los dos extremos: cuando hay muchas rachas o muy pocas.

Autocorrelación

- Si tenemos la secuencia $x_1, x_2, x_3, \ldots$, la autocorrelación de primer orden se estima calculando la correlación de Pearson de esa serie con la serie obtenida corriendo una posición la serie original

$$\begin{array}{ccccc} x_1 & x_2 & x_3 & \cdots & x_{n-1} \\ x_2 & x_3 & x_4 & \cdots & x_n \end{array}$$

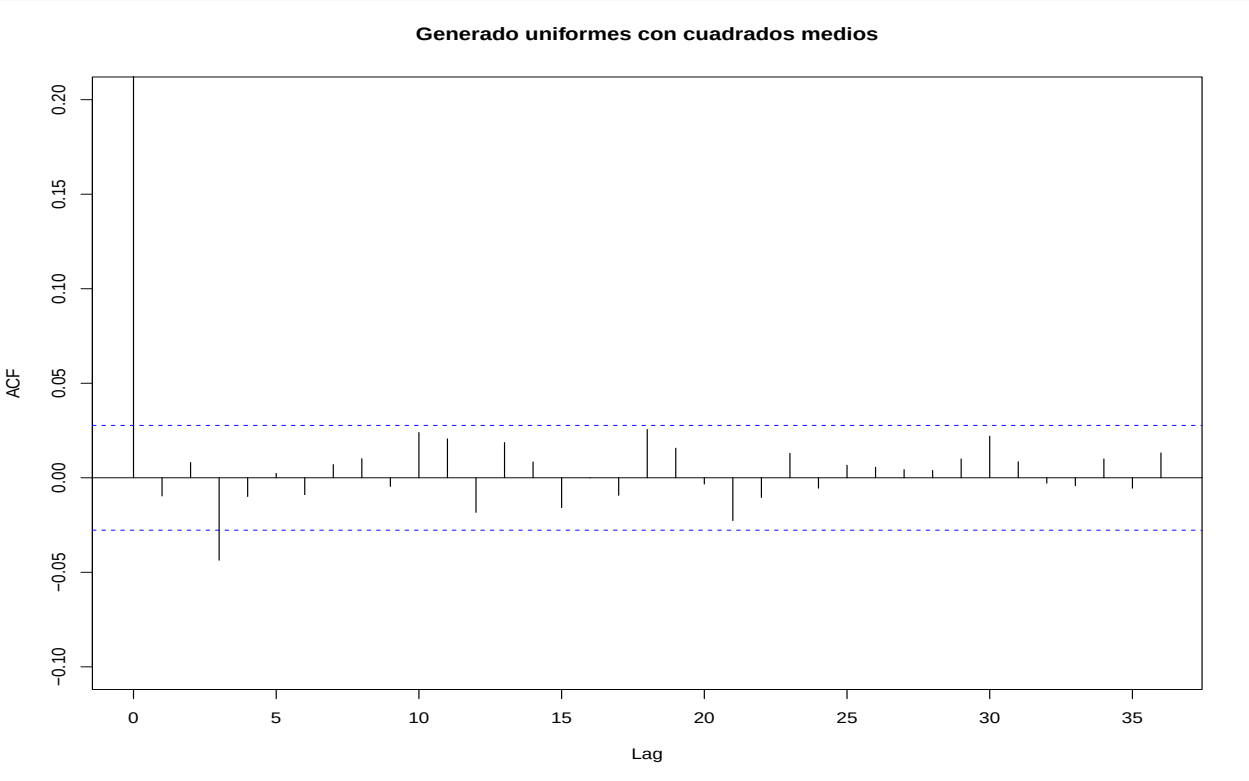
- Por supuesto, es desable que esta autocorrelación sea 0 o muy baja. Similarmente, la autocorrelación de segundo orden se estima corriendo la serie original dos posiciones

$$\begin{array}{ccccc} x_1 & x_2 & x_3 & \cdots & x_{n-2} \\ x_3 & x_4 & x_5 & \cdots & x_n \end{array}$$

- y así sucesivamente, se puede evaluar la autocorrelación de diferentes órdenes. La siguiente gráfica muestra las autocorrelaciones para una serie generada usando cuadrados medios de von Neumann de 10 dígitos.

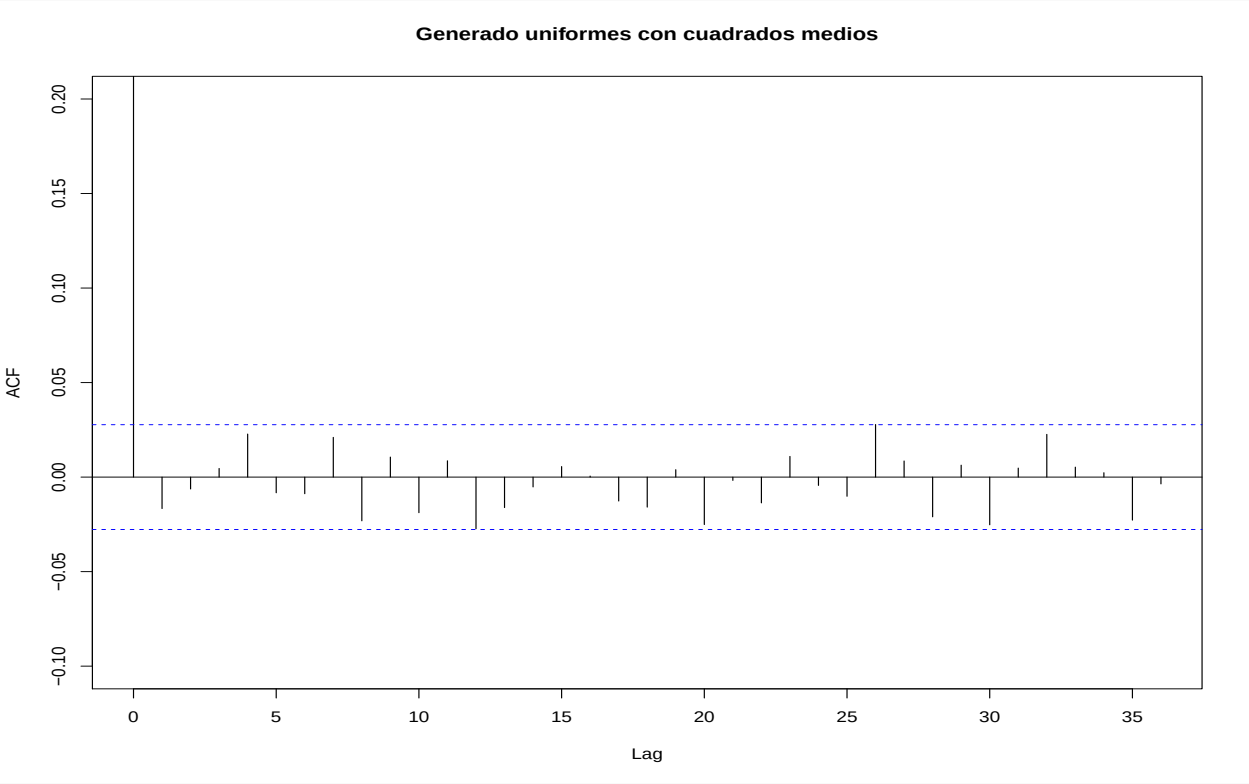
Autocorrelaciones

```
seed <- 6582014372
unif <- RCUAD(n,seed) # versión de 10 dígitos
acf(unif,main="Generado uniformes con cuadrados medios",ylim=c(-.1,.2))
```



Autocorrelaciones, usando `runif`

```
n    <- 5000
set.seed(54)
unif <- runif(n)
acf(unif, main="Generado uniformes con runif", ylim=c(-.1,.2))
```



I.d. Generación de v.a. continuas: métodos de inversión y aceptación /rechazo

Generación de variables aleatorias no-uniformes

- Un resultado importante para la generación de variables aleatorias no-uniformes es el Teorema de la Probabilidad Inversa.
- **Teorema**: Si X es una variable aleatoria continua con función de distribución $F(x)$ y si U es una variable aleatoria uniforme en $(0, 1)$, entonces, si $W = F^{-1}(U)$, se cumple que $W \stackrel{d}{=} X$.
- La prueba de este resultado es como sigue

$$\begin{aligned} F_W(w) &= P(W \leq w) = P(F^{-1}(U) \leq w) \\ &= P(U \leq F(w)) = F(w) \end{aligned}$$

  esto es, W y X tienen la misma distribución.
- Este resultado es usado para generar variables que tengan la distribución F, mediante la generación de Uniformes, U, y luego aplicando la transformación inversa $F^{-1}(U)$.

Ejemplo: Generación de Exponenciales

- Apliquemos el resultado anterior a la distribución exponencial:

$$\text{Si } X \sim \text{Exp}(\lambda), \text{ entonces, } f(x) = \lambda e^{-\lambda x} \text{ y } F(x) = 1 - e^{-\lambda x}$$

- Para usar el Teorema, invertimos la función de distribución

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

- Entonces, para generar exponenciales con parámetro λ, generamos primero uniformes, U_i, y luego usamos la transformación correspondiente,

$$X_i = -\frac{1}{\lambda} \log(1 - U_i), \quad i = 1, 2, \cdots$$

- Ahora, la distribución de $1–U_i$ es la misma que la distribución de U_i, entonces el algoritmo para generar exponenciales es:

$$X_i = -\frac{1}{\lambda} \log(U_i), \quad i = 1, 2, \cdots$$

Ejemplo: Generación de variables Weibull

- El caso de la distribución Weibull es igualmente sencillo

$$\text{Si } X \sim \text{Weibull}(\theta, \beta), \text{ entonces, } f(x) = \frac{\beta}{\theta}\exp\left\{-\left(\frac{x}{\theta}\right)^{\beta}\right\}$$

y

$$F(x) = 1 - \exp\left\{-\left(\frac{x}{\theta}\right)^{\beta}\right\}$$

- Invertimos la función de distribución

$$1 - \exp\left\{-\left(\frac{x}{\theta}\right)^{\beta}\right\} = u \Leftrightarrow x = \theta\ [-\log(1-u)]^{1/\beta}$$

- Nuevamente, podemos usar U en vez de $1-U$ y el algoritmo para generar Weibulls es:

$$X_i = \theta\ [-\log(U_i)]^{1/\beta}, \quad i = 1, 2, \cdots$$

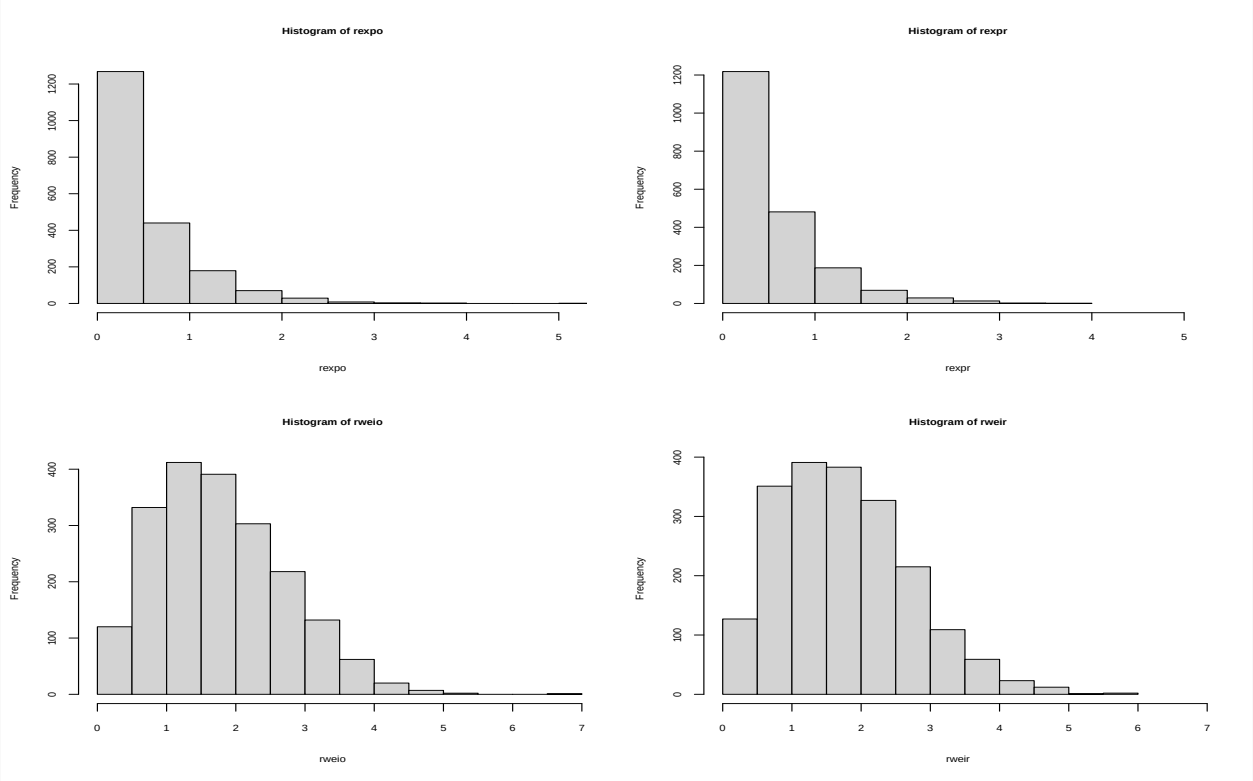
Código en R

```
# Generar Exponenciales y Weibulls
set.seed(7479)
n     <- 2000
lam   <- 2
rexpo <- -(1/lam)*log(runif(n))
rexpr <- rexp(n,rate=lam)
teta  <- 2
beta  <- 2
rweio <- teta*(-log(runif(n)))^(1/beta)
rweir <- rweibull(n, shape=beta, scale=teta )

par(mfrow=c(2,2))
hist(rexpo,cex.axis=.7,cex.lab=.7,cex.main=.7,xlim=c(0,5.1),breaks=12)
hist(rexpr,cex.axis=.7,cex.lab=.7,cex.main=.7,xlim=c(0,5.1),breaks=12)
hist(rweio,cex.axis=.7,cex.lab=.7,cex.main=.7,xlim=c(0,6.8),breaks=12)
hist(rweir,cex.axis=.7,cex.lab=.7,cex.main=.7,xlim=c(0,6.8),breaks=12)
```

Comparación

- Las gráficas de la izquierda son variables generadas a partir de uniformes, y las de la derecha usaron los generadores internos de R.



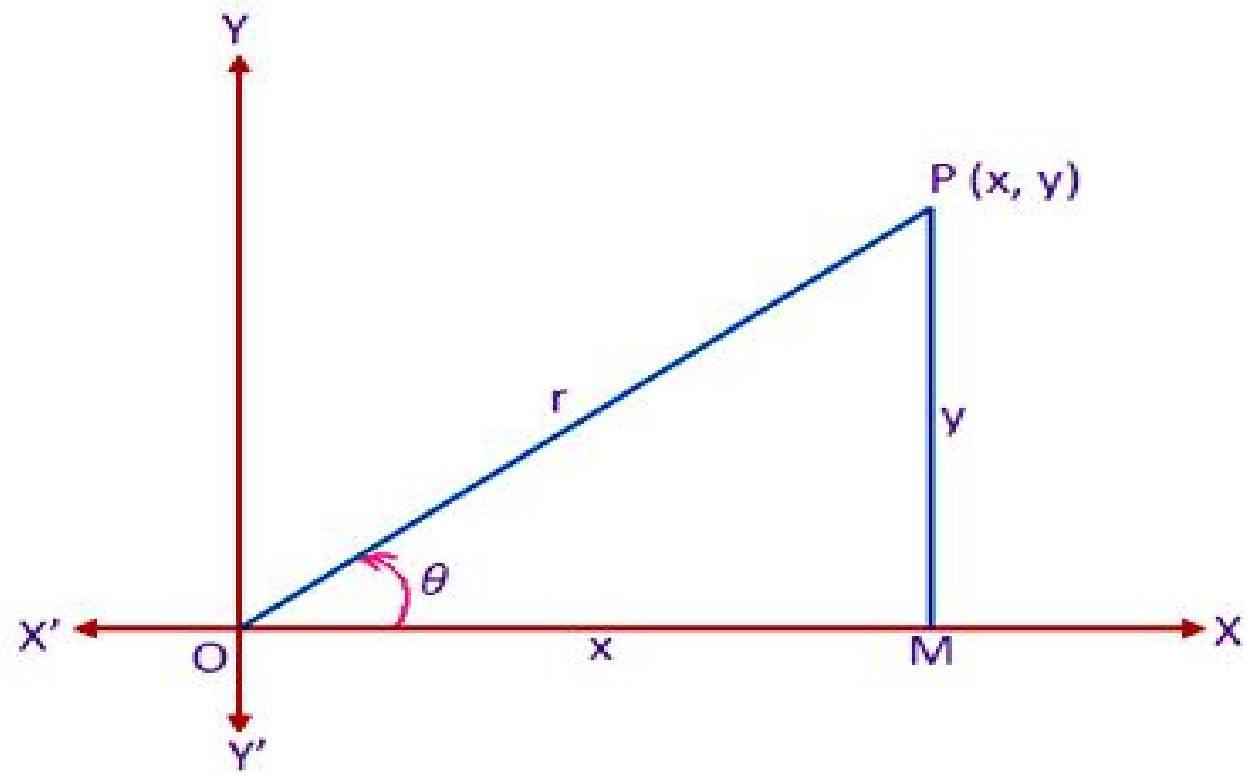
Generación de Normales. Método de Box y Muller

- El uso del teorema de la probabilidad inversa es (en teoría) siempre posible de implementar pues la función de distribución siempre es monótona creciente (al menos para las variables continuas usuales) y por lo tanto siempre es posible invertirla.
- Sin embargo, para el caso de la distribución normal no es posible tener una forma cerrada para la inversa de su función de distribución, de modo que usamos otro procedimiento para la generación de normales.

Recordar: Coordenadas polares

- La relación entre coordenadas cartesianas y polares es

$$x = r\ \mathrm{Cos}(\theta)$$
$$y = r\ \mathrm{Sen}(\theta)$$



Método de Box y Muller

- La clave del método es observar que, si

$$X \sim \text{Normal}(0,1) \quad \text{y} \quad Y \sim \text{Normal}(0,1)$$

con X y Y independientes, entonces

$$r^2 \sim \text{Exp}(\lambda = 1/2)$$
$$\theta \sim \text{Unif}(0, 2\pi)$$

- $\cdots$ y como ya sabemos generar Uniformes y Exponenciales, aprovechamos estas relaciones.
- Algoritmo: Generar secuencia (U_{i1}, U_{i2}), $i = 1, 2, \cdots$
- Entonces

$$X_i = \sqrt{-2\log(U_{i1})}\,\text{Cos}(2\pi U_{i2})$$
$$Y_i = \sqrt{-2\log(U_{i1})}\,\text{Sen}(2\pi U_{i2})$$

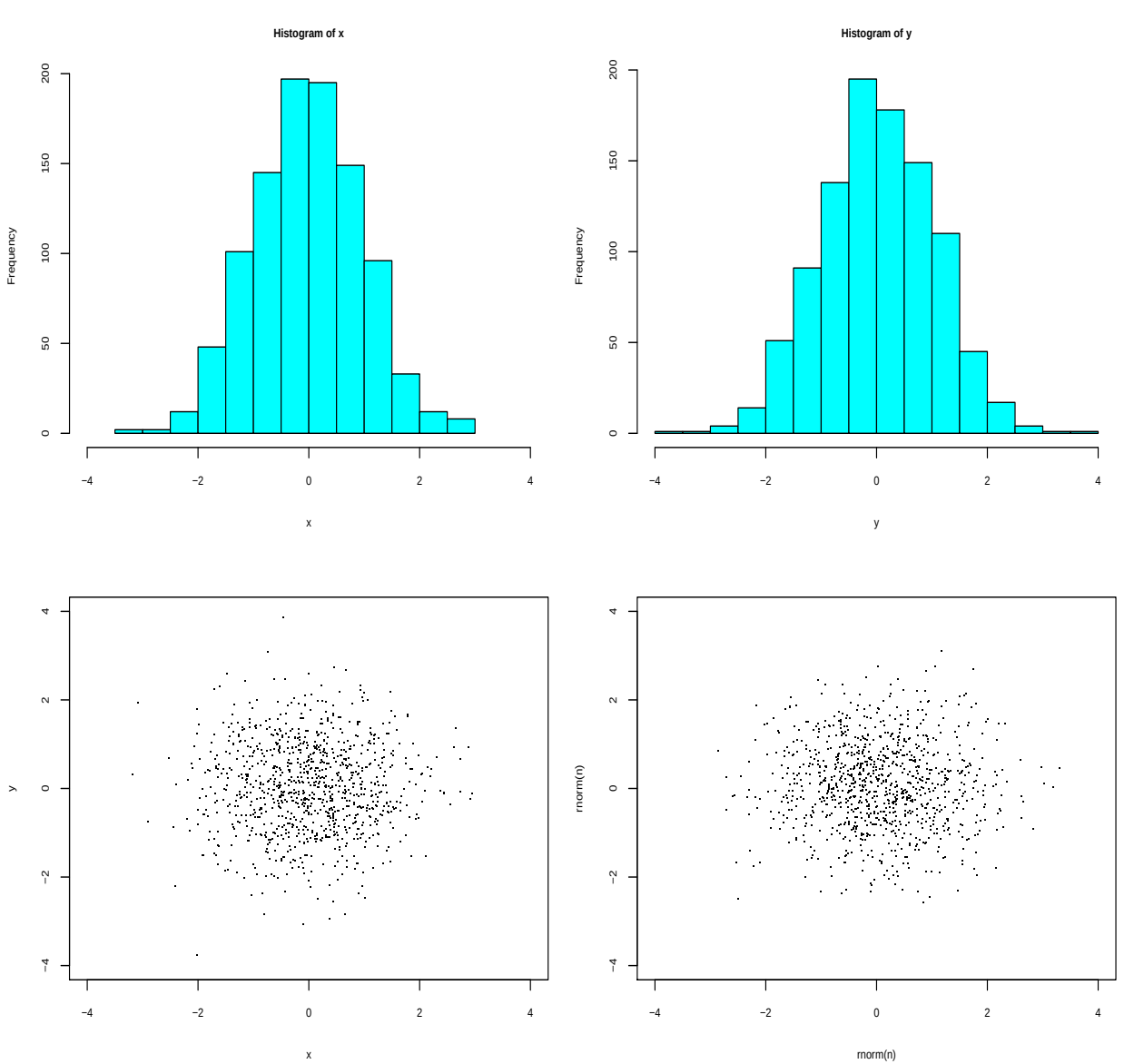
Código en R

```
# Box Muller

n  <- 1000
u1 <- runif(n)
u2 <- runif(n)
x  <- sqrt(-2*log(u1))*cos(2*pi*u2)
y  <- sqrt(-2*log(u1))*sin(2*pi*u2)
r  <- c(-4,4)

par(mfrow=c(2,2), mar=c(4, 4, 3, 1) + 0.1)
hist(x, cex.axis=.7, cex.lab=.7,col="cyan",
     cex.main=.7, xlim=r)
hist(y, cex.axis=.7, cex.lab=.7,col="cyan",
     cex.main=.7, xlim=r)
plot(x,y, pch=".", cex.axis=.7,
     cex.lab=.7, xlim=r, ylim=r)
plot(rnorm(n),rnorm(n), pch=".",
     cex.axis=.7, cex.lab=.7, xlim=r, ylim=r)
```

Visualización



Método aceptación-rechazo

- Supongamos que queremos simular de una densidad $f(x)$, no necesariamente normalizada (esto es importante en inferencia Bayesiana)
- Supongamos además, que se cuenta con una densidad $g(x)$ de la cual es factible simular, y es tal que $f(x) \leq M\, g(x)$ para toda x en el soporte de f. El siguiente algoritmo produce simulaciones de f, mediante simulaciones a partir de g
- Algoritmo Aceptación-Rechazo
  1. Genera $X \sim g(x)$, $U \sim U[0,1]$
  2. Aceptar $Y = X$, si
  $$U \leq \frac{f(X)}{M\, g(X)}$$
  3. Si no se acepta, regresar a 1.

¿Por qué funciona?

La distribución de las Y's simuladas, es la misma que la distribución de la variable objetivo.

$$P(Y \leq y) = P\left(X \leq y \mid U \leq \frac{f(X)}{M\, g(X)}\right)$$
$$= \frac{P\left(X \leq y\ ,\ U \leq \frac{f(X)}{M\, g(X)}\right)}{P\left(U \leq \frac{f(X)}{M\, g(X)}\right)} \equiv \frac{A}{B} = P(X \leq y)$$

donde la última igualdad se sigue de

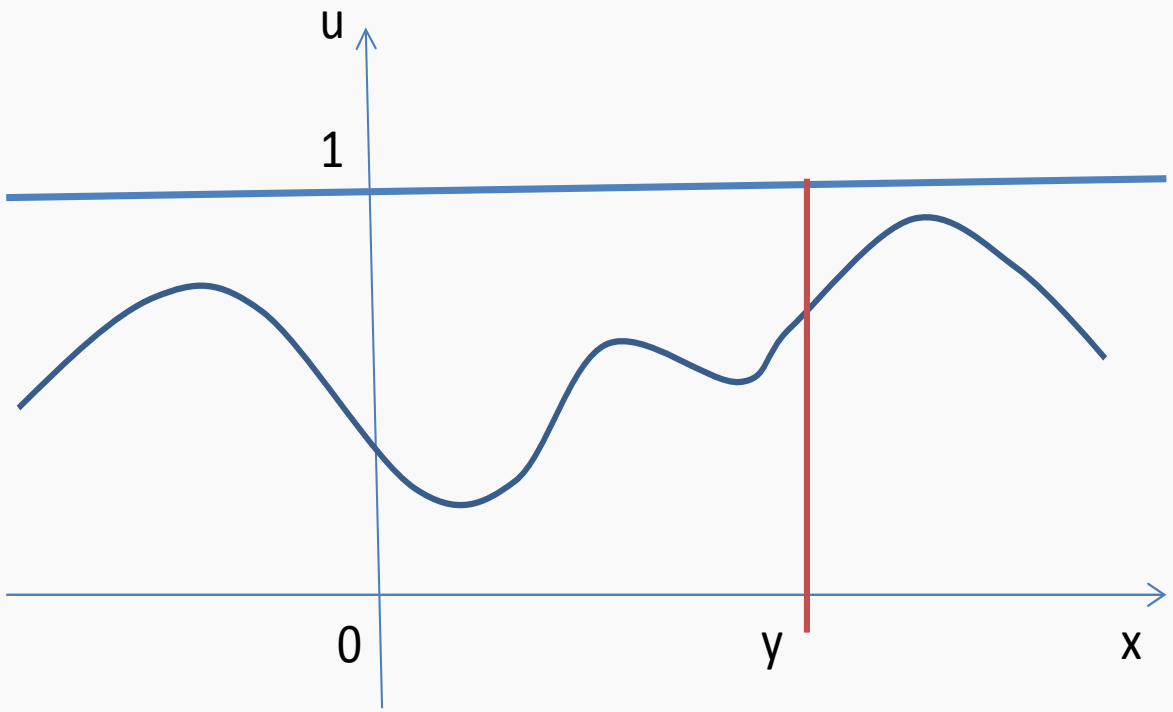
$$A = \int_{-\infty}^{y} \int_{0}^{f(x)/Mg(x)} f(x,u)\ du\ dx = \int_{-\infty}^{y} g(x) \int_{0}^{f(x)/Mg(x)} h(u)\ du\ dx$$
$$= \int_{-\infty}^{y} \frac{1}{M} f(x) dx = \frac{1}{M} \int_{-\infty}^{y} f(x) dx$$

$$B = \int_{-\infty}^{\infty} \int_{0}^{f(x)/Mg(x)} f(x,u)\ du\ dx = \int_{-\infty}^{\infty} g(x) \int_{0}^{f(x)/Mg(x)} h(u)\ du\ dx$$
$$= \int_{-\infty}^{\infty} \frac{1}{M} f(x) dx = \frac{1}{M} \int_{-\infty}^{\infty} f(x) dx = \frac{1}{M} \text{ cte. de normalización de } f(x)$$

Aceptación-rechazo

Aquí visualizamos la región de integración para el numerador A

$$A = P\left(X \leq y\ ,\ U \leq \frac{f(X)}{M\ g(X)}\right) = \int_{-\infty}^{y} \int_{0}^{f(x)/Mg(x)} f(x,u)\ du\ dx$$



Ejemplo: Generar variables beta $B(a = 2, b = 7)$

- La densidad beta con parámetros a y b es

$$f(x) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}\, x^{a-1}(1-x)^{b-1}$$

con

$$E(X) = \frac{a}{a+b}, \qquad \text{Var}(X) = \frac{ab}{(a+b)^2(a+b+1)}$$

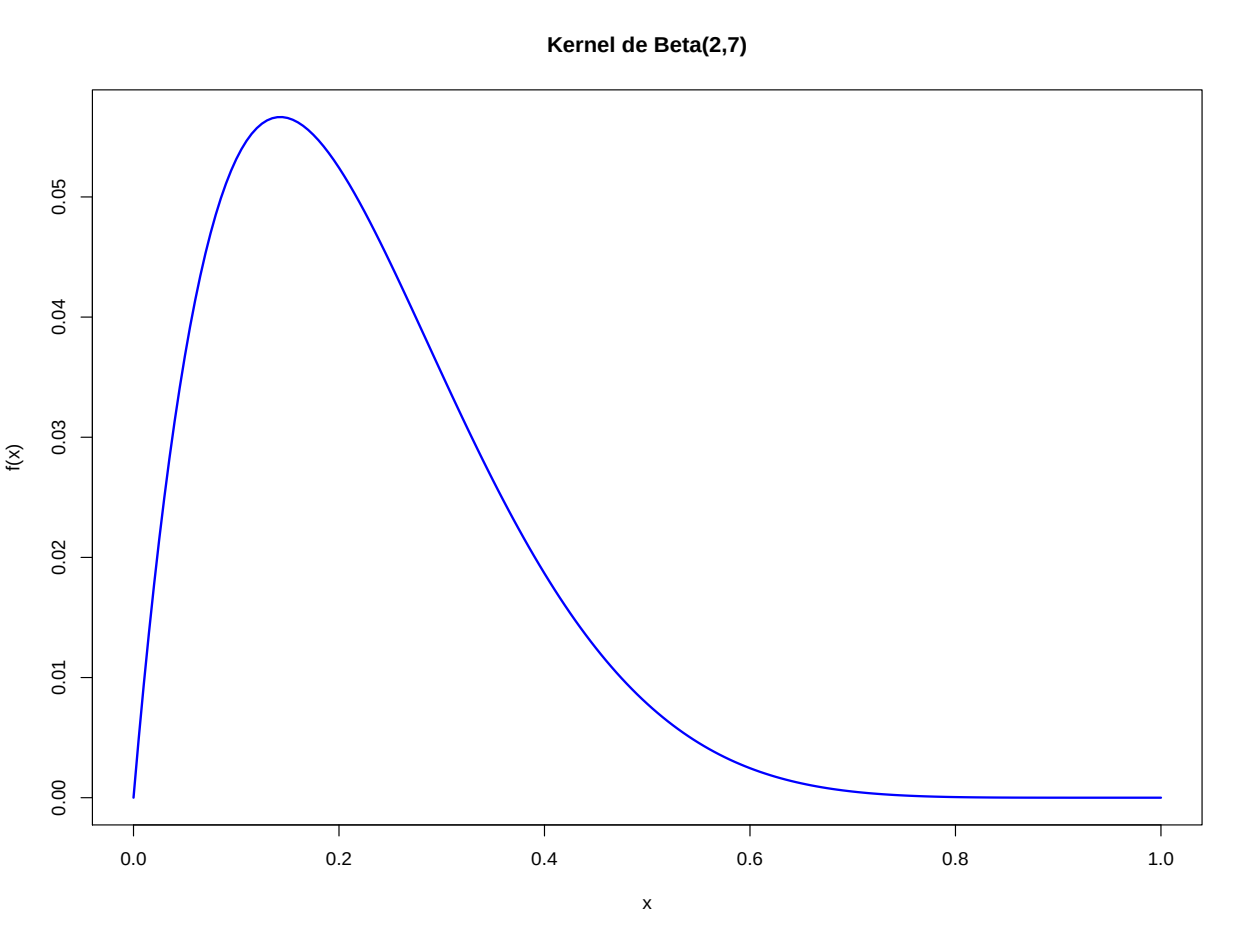
- Supongamos que tenemos solo el núcleo de la distribución

$$f(x) = x^{a-1}x^{b-1} = x(1-x)^6$$

- Consideremos $g(x)$ la distribución Uniforme(0,1). Es fácil ver que $M = 0.05665278$ es tal que

$$f(x) \leq M\, g(x), \quad 0 \leq x \leq 1$$

Núcleo de distribución objetivo

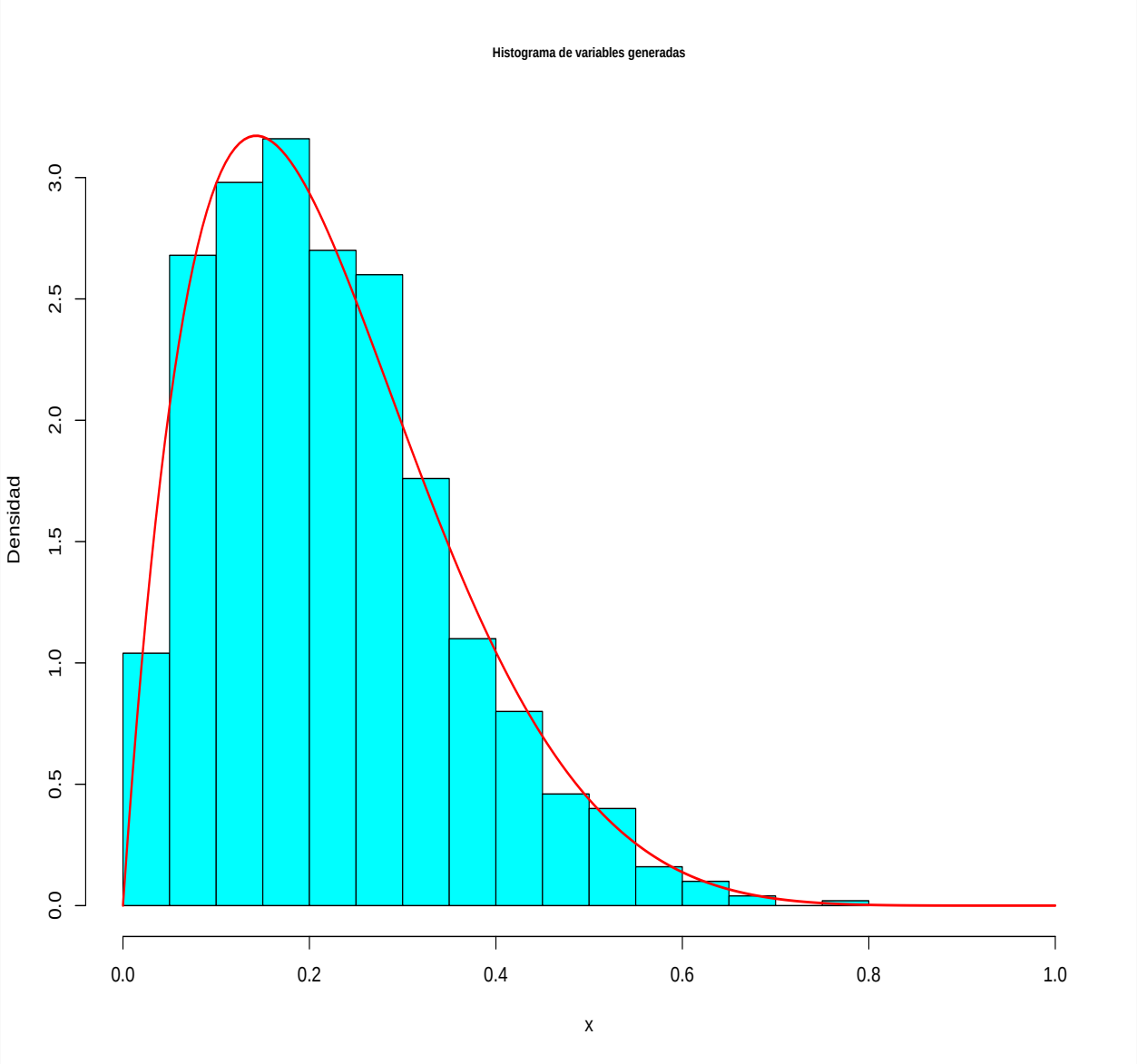



```
# Gráfica del kernel, f(x), de la distribución objetivo
f  <- function(x){ x*(1-x)^6 }
x  <- seq(0,1,length=201)
plot(x,f(x), type="l", lwd=2, col="blue", main="Kernel de Beta(2,7)")
```

Simulación vía aceptación-rechazo 1

```
# Ejemplo: Generar variables beta(2,7)
# Distribución propuesta: Uniforme
g  <- function(x){1}
M  <- (1/7)*(1-1/7)^6  # 0.05665278 valor de M, es el máximo de f
# la constante M es tal que f(x) <= M g(x) donde g(x) = 1, 0<x<1
set.seed(588989);   n      <- 1000
seguir <- TRUE; rbet   <- rep(0,n)
kk     <- 0         # contador del número total de iteraciones
i      <- 0         # variables aceptadas
while( seguir ){
 kk <- kk+1
 u  <- runif(1)
 z  <- runif(1)
 if( u <= f(z)/(M*g(z)) ){
  i       <- i+1
  rbet[i] <- z
  if( i==n ){ seguir<-FALSE }}}
hist(rbet,prob=T,xlim=c(0,1), col="cyan", ylim=c(0,3.18),cex.main=.7,
     main="Histograma de variables generadas",ylab="Densidad",xlab="x")
db <- dbeta(x, shape1=2, shape2=7);  lines(x,db, col="red", lwd=2)
# El número esperado, K, de iteraciones por cada aceptación es
# 3.17, asi que kk debe andar alrededor de 3170 (3.17 x 1000),
# para esta semilla, kk es 3042.
```

Simulación vía aceptación-rechazo 2



Número esperado de variables generadas por cada aceptación

- La probabilidad de aceptar un dato generado es

$$P(A) = P\left(U_1 \leq \frac{1}{M} U_2 (1 - U_2)^6\right)$$

- Puede verse que esta probabilidad de aceptación es

$$\begin{aligned} P(A) &= \int_0^1 P\left(U_1 \leq \frac{1}{M} u_2 (1 - u_2)^6\right) du_2 \\ &= \frac{1}{M} \int_0^1 u_2 (1 - u_2)^6 du_2 = \frac{1}{M} \frac{\Gamma(2)\Gamma(7)}{\Gamma(9)} = 0.3152033 \end{aligned}$$

- El número esperado de datos generados hasta obtener un aceptado es (recordar distribución geométrica):

$$\frac{1}{P(A)} = \frac{1}{0.3152033} = 3.172556$$

- Así, si queremos $n = 1000$ betas generadas, entonces, en promedio requeriremos generar 3,173 variable, para obtener 1000.

Tarea 1 (1)

1. Consideremos las últimas tendencias de cierta acción de la bolsa. Suponga que dicha acción esta en nuestro portafolio de inversión:

   $$\uparrow \uparrow \uparrow \downarrow \uparrow \uparrow \downarrow \downarrow \downarrow \downarrow \uparrow \downarrow \downarrow \downarrow$$

   Suponga que las caídas y bajadas son más o menos del mismo nivel. El comportamiento de la acción, ¿es consistente con una variabilidad aleatoria?

Tarea 1 (2)

2. **Integración Montecarlo**. La circunferencia inscrita en el cuadrado, que se muestra a la izquierda en la siguiente hoja, fue hecha con los siguientes comandos:

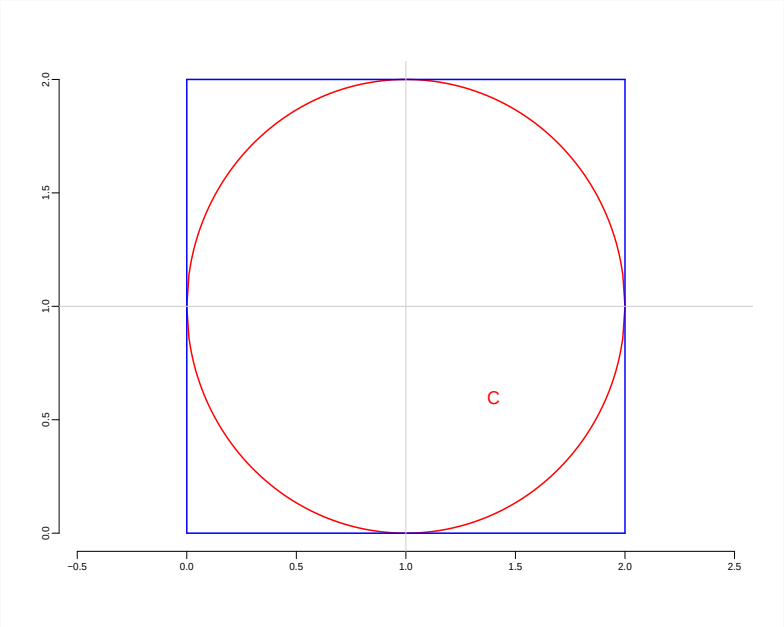
```
h  <- 1; r  <- 1
xx <- seq(0,2*h,length=200)
yu <- h + sqrt(r^2-(xx-h)^2)
yl <- h - sqrt(r^2-(xx-h)^2)
plot(h,h,xlab="",ylab="",type="n", mgp=c(1.5,.5,0), asp=1,
 xlim=c(0,2*h), ylim=c(0,2*h), cex.axis=.8, bty="n" )
lines(xx,yu,lwd=2,col="red")
lines(xx,yl,lwd=2,col="red")
segments(0,0,2*h,0,lwd=2,col="blue")
segments(2*h,0,2*h,2*h,lwd=2,col="blue")
segments(2*h,2*h,0,2*h,lwd=2,col="blue")
segments(0,2*h,0,0,lwd=2,col="blue")
abline(h=h,v=h,col=gray(.8))
text(1.4,.6,"C",cex=1.5,col="red")
```

Si genero un punto al azar dentro del cuadrado, la probabilidad de que ese punto también esté dentro de la circunferencia, es igual al cociente de las dos áreas

$$p = \frac{A_{\text{circ}}}{A_{\text{cuad}}} = \frac{A_{\text{circ}}}{4}$$

de modo que $A_{\text{circ}} = 4\ p$.

Tarea 1 (3)



2.a. Usando simulación, estime el área del círculo. Para ello, genere x_1, Uniforme entre 0 y 2; luego genere y_1 también Uniforme entre 0 y 2. Si el punto (x_1, y_1) cae dentro de la circunferencia, registramos un 1 (un "exito") si no, entonces registramos un 0 ("fracaso"). Hacemos esto muchas veces y tendremos una secuencia de 1's y 0's, la cual es una muestra aleatoria de una distribución Bernoulli(p). Entonces podemos estimar p mediante $\widehat{p}$, la proporción observada de éxitos y, con ello, podemos estimar el área del círculo.

Tarea 1 (4)

2.b. Usando técnicas similares a las del inciso anterior, estime el área de la región, A, acotada por las curvas en rojo de la gráfica de la hoja anterior a la derecha. Las curvas superior e inferior son, respectivamente

$$s(x) = \text{Sen}(x) + 2 \qquad \text{y} \qquad I(x) = \frac{1}{\sqrt{x + 0.5}} - \frac{1}{2}$$

2.c. En los dos incisos anteriores es fácil saber, sin necesidad de simulación, los valores correctos de las respectivas áreas; sin embargo, en otras situaciones no es posible conocer los valores verdaderos, así que es conveniente tener una idea del error que se comete con estos métodos de simulación. Si $\widehat{p}$ es la proporción observada de éxitos y es un estimador de la proporción verdadera, p, entonces un intervalo de confianza para p, está dado por

$$\widehat{p} \pm z_{\alpha/2} \sqrt{\frac{\widehat{p}\,(1 - \widehat{p})}{n}}$$

de aquí, el error máximo para estimar p, con una confianza del $100(1 - \alpha)\%$, está dado por $z_{\alpha/2} \sqrt{\widehat{p}\,(1 - \widehat{p})/n}$. Para los incisos anteriores calcule los errores máximos en las estimaciones de las áreas y compárelos con los errores reales.

Tarea 1 (5)

2.d. Consideremos de nuevo el problema del área del círculo. Básicamente, lo que estamos haciendo es calcular la integral doble

$$I = \int_0^2 \int_0^2 g(x,y)\, dx\, dy = \int_0^2 \int_0^2 I_{\{(x,y)\,\in\, \mathrm{C}\}}\, dx\, dy$$

donde $g(x,y) = I_{\{(x,y)\,\in\, \mathrm{C}\}}$ es la función indicadora de la región C, la cual vale 1 si $(x,y) \in$ C y vale 0 de otra forma. Cuando dijimos "genere x_1, Uniforme entre 0 y 2; luego genere y_1 también Uniforme entre 0 y 2", lo que estábamos haciendo era simular de la Uniforme en el rectángulo $(0,2) \times (0,2)$; esto es, generamos una realización (x,y) de la densidad uniforme bivariada

$$U(x,y) = \begin{cases} \frac{1}{4} & \text{si } 0 \le x \le 2,\ 0 \le y \le 2 \\ 0 & \text{de otra forma} \end{cases}$$

Note que

$$I = \int_0^2 \int_0^2 \frac{I_{\{(x,y)\,\in\, \mathrm{C}\}}}{U(x,y)}\, U(x,y)\, dx\, dy = E\left[\frac{I_{\{(X,Y)\,\in\, \mathrm{C}\}}}{U(X,Y)}\right] \equiv E\left[h(X,Y)\right]$$

esto es, la integral, I, no es otra cosa más que el valor esperado de $h(X,Y)$.

Tarea 1 (6)

2.d. (Cont.) Ahora, si tenemos una muestra aleatoria $(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)$, entonces, por la Ley de los Grandes Números, tenemos que

$$\frac{1}{n}\sum_{i=1}^{n} h(x_i, y_i) \ \rightarrow E[h(X, Y)] = I$$

y ya está; ésta observación nos dice que si queremos aproximar la integral I, entonces podemos generar muchas (x_i, y_i)'s y promediar todas las $h(x_i, y_i)$'s y esto sería $\widehat{I}$. Entre paréntesis, para el problema en el cuadrado, tenemos que $U(x, y) = 1/4$ y entonces $h(x, y) = 4I_{\{(x,y) \in C\}}$. Tomemos estas ideas y consideremos un problema más simple. Consideremos la integral

$$I = \int_a^b g(x)dx$$

Sea $f(x)$ una densidad (de la cual podamos generar muestras aleatorias); entonces

$$I = \int_a^b g(x)dx = \int_a^b \frac{g(x)}{f(x)}\, f(x)dx \equiv \int_a^b h(x)\, f(x)dx \ = \ E[\, h(X)\,] \ \leftarrow \ \frac{1}{n}\sum_{i=1}^{n} h(x_i)$$

donde $x_1, \cdots, x_n$ son una muestra aleatoria tomada de la densidad $f(x)$.

Tarea 1 (7)

2.d. (Cont.) Considere la integral

$$I = \int_0^{\pi/2} \text{Sen}(x)\, dx$$

- Aproxime el valor de I usando simulación (usando como f a una Uniforme).
- Nuevamente aproxime a I pero ahora usando muestras tomadas de $f(x) = 8x/\pi^2$, para $0 \leq x \leq \pi/2$. ¿Cuál elección de f fue mejor?.

En este segundo punto, la densidad de donde se muestrea, $f(x)$, es parecida a la función que se desea integrar, $g(x)$; a esto se le llama "muestreo por importancia". En general, para integrales unidimensionales no se usa integración Montecarlo, son preferidos los métodos de Cuadratura; sin embargo, para integrales multidimensionales, los enfoques basados en simulación son muy valiosos pues las fórmulas para usar Cuadratura se vuelven muy complejas e ineficientes, mientras que integración Montecarlo es, en general, sencilla de aplicar.

Entregar: Viernes 29 de enero.