Modelación Estadística: Tarea Final

Ivan Vega Gutiérrez

2 de diciembre 2021

Ejercicios 3.3.3.

1. Generar $u_1, \cdots, u_N$ (para N = 50, 100, 1,000, 10,000, 100,000)
   (a) Método congruencial mixto
   (b) Usar R

   A continuación se tiene el algoritmo del generador congruencial.

```
GCM = function(N,a,c,m,x0){
  # x0 : valor inicial
  # a : multiplicador
  # c : incremento
  # m : modulo (de preferencia un número grande)
  # formula  X_i+1 = a X_i(mod m)
  X = vector()
  U = vector()
  X[1] = x0
  for (i in 1:(N-1))
  {
    X[i+1] = (a*X[i] + c)%%m
  }
  U = X/m
  return (U)
}
```

Por otro lado, en R la función que permite generar número de la distribución uniforme es runif(n, a, b), donde n es la cantidad de números aleatorios que sea desa generar, a el límite inferior de la distribución y b el límite superior.

```
N = 50
U1 = runif(N, 0, 1)
```

2. Prueba las hipótesis: $U = \{u_1, \cdots, u_N\}$

$$H_0 : U \sim U(0, 1)$$

   vs H1 : la distribución de U es diferente de la distribución uniforme. Usar los siguientes estadísticos
   (a) Anderson-Darling
   (b) Prueba de rachas

(c) Prueba de Kolmogorov-Smirnov

(d) Prueba χ^2

(a) Anderson Darling(Aleatoriedad)

```
AndersonDarling =  function(X){
  X = sort(X)
  n = length(X)
  suma = 0
  for (k in 1:n)
  {
    suma = suma + ((2*k-1)/n)*(log(X[k]) + log(1- X[n+1-k]))
  }
  A = -n - suma
  return(A)
}
```

(b) Prueba de rachas (Independencia)

```
PruebaRachas = function(X)
{
  n = length(X)
  y = vector()
  for (i in 1:(n-1))
  {
    if (X[i+1] > X[i])
    {
      y[i] = 1
    }
    else
    {
      y[i] = 0
    }
  }
  N= 0
  for (k in 2:length(y))
  {
    if (y[k]!=y[k-1])
    {
      N = N +1
    }
  }
  mu = (2*n-1)/3
  sigma = (16*n-29)/90
  z = (N - mu)/sqrt(sigma)
  alpha = 0.05
  zp = qnorm(alpha)
  if (z > abs(zp))
  {
    return("Rechazar H0")
  }
  else
  {
```

```
    return("No rechazar H0")
  }
}
```

(c) Prueba de Kolmogorov-Smirnov (Bondad y ajuste)

```
KolmogorovSmirnov = function(X)
{
  X = sort(X)
  n = length(X)
  k = 1:n
  D1 = (k/n) - X
  D2 = X - ((k-1)/n)
  maxD = max(D1)
  minD = max(D2)
  D = max(maxD, minD)
  z = 1.36/sqrt(n)
  if (D < z)
  {
    print("No rechazar H0")
  }
  else
  {
    print("Se rechaza H0")
  }
}
```

(d) Prueba χ^2

```
JiCuadrada = function(X)
{
  n = length(X)
  X = sort(X)
  ei = 1/n
  f = X/n
  J = sum((f-ei)^2 / ei)
  if (J > qchisq(0.95, df=n-1))
  {
    print("Rechazar H0")
  }
  else
  {
    print("No rechazar H0")
  }
}
```

Ahora probemos que las distribuciones que creamos efectivamente tienen una distribución uniforme, para ello supongamos que el tamaño de nuestra muestra es N=50, luego

```
N = 50
U1 = GCM(N,3,3,1000,3)
U2 = runif(N, 0, 1)
```

```r
#Para el generador congruencial mixto, se tiene que
AndersonDarling(U1)
```

```
## [1] 1.195185
```

```r
PruebaRachas(U1)
```

```
## [1] "No rechazar H0"
```

```r
KolmogorovSmirnov(U1)
```

```
## [1] "No rechazar H0"
```

```r
JiCuadrada(U1)
```

```
## [1] "No rechazar H0"
```

```r
#Para el generador de R, se tiene que
AndersonDarling(U2)
```

```
## [1] 0.5514043
```

```r
PruebaRachas(U2)
```

```
## [1] "No rechazar H0"
```

```r
KolmogorovSmirnov(U2)
```

```
## [1] "No rechazar H0"
```

```r
JiCuadrada(U2)
```

```
## [1] "No rechazar H0"
```

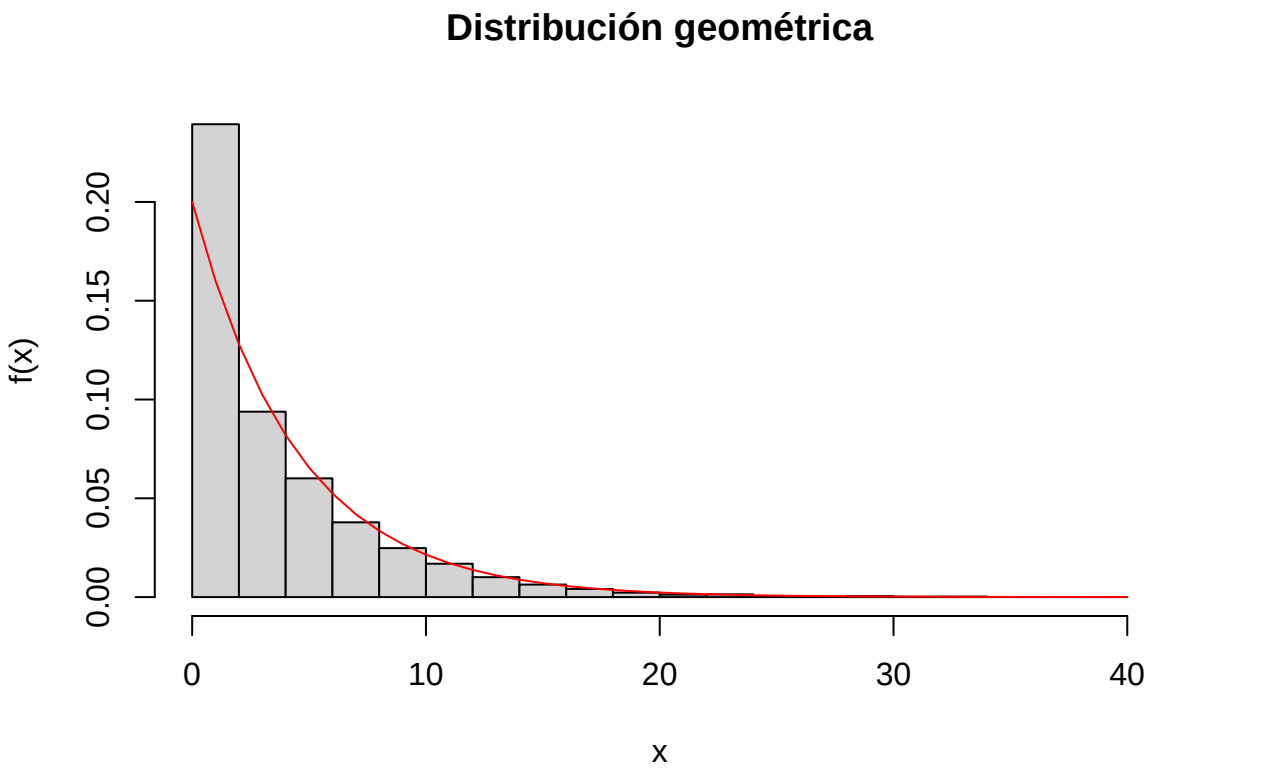
De lo anterior se puede concluir que las muestras uniformes que se generan son buenas.

Ejercicios 3.4.2

1. Escoja un valor p en la distribución geométrica y simule 10,000 salidas.

Para simular una distribución geométrica se tiene el siguiente algoritmo

```r
geometrica = function(n,p)
{
  u = runif(n)
  X = vector()
  for (i in 1:n){
    k=1
    while(TRUE){
      pk = sum(p*(1-p)^(seq(1,k)-1))
      if(pk>u[i]){
        X[i] = k-1
        break
      }
      k = k+1
    }
  }
  return(X)
}

X = geometrica(10000, 0.2)
# Comparamos
hist(X,freq=FALSE, main="Distribución geométrica",ylab="f(x)",xlab="x")
x = seq(0,max(X),1)
y=dgeom(x,0.2)
lines(x,y,col="red")
```



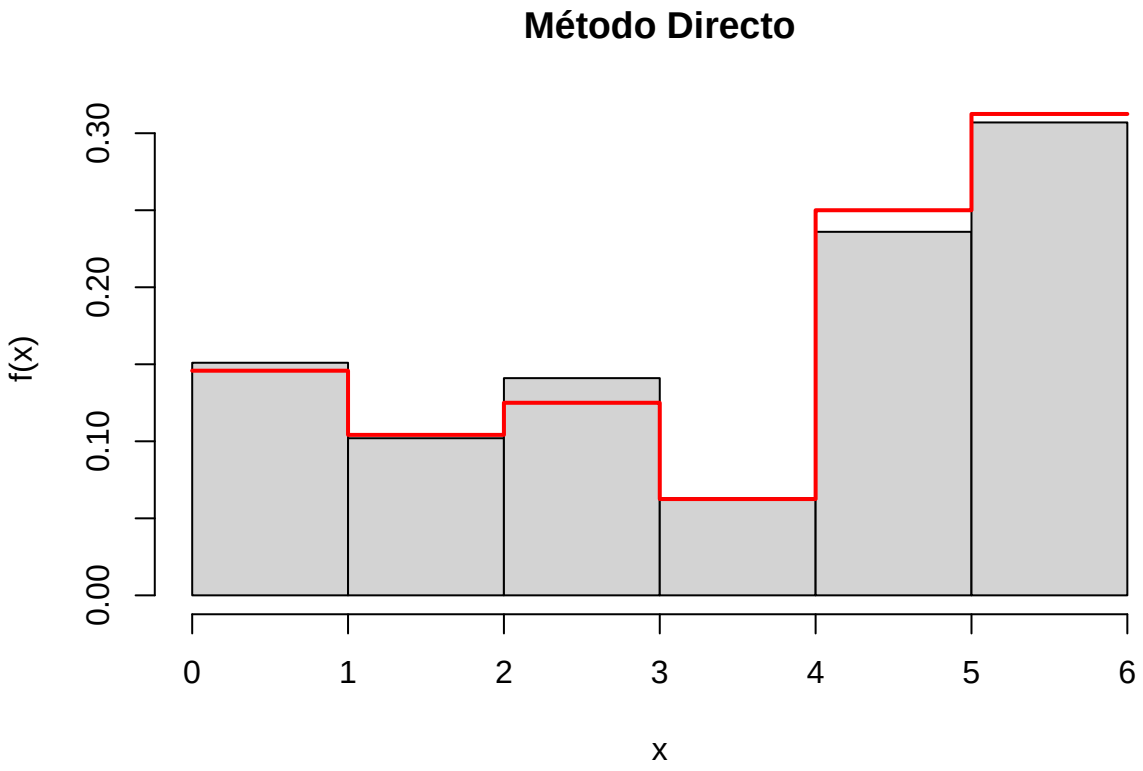
2. Sea X una v.a.tal que X ={1, 2, 3, 4, 5, 6 } y las respectivas probabilidades $\{\frac{7}{48}, \frac{5}{48}, \frac{1}{8}, \frac{1}{16}, \frac{1}{4}, \frac{5}{16}\}$. Simule la distribución de los 6 puntos con

   (a) aplicando un método directo

(b) aplicando un método de rechazo

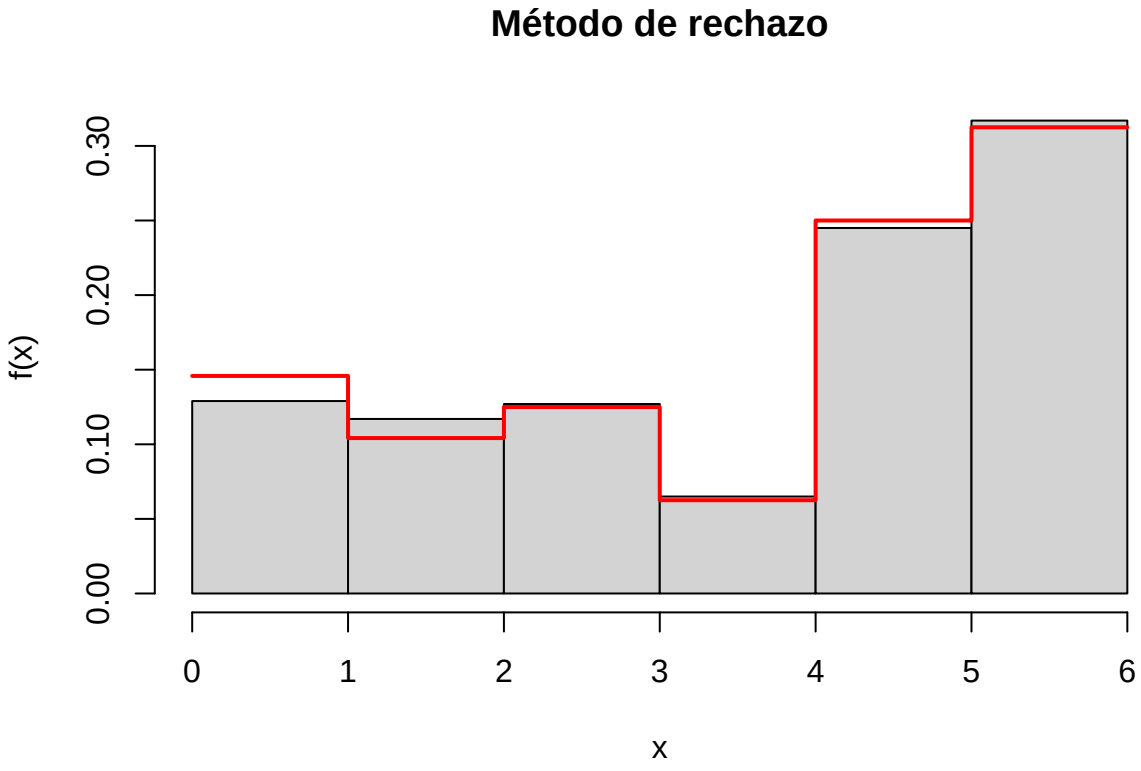
Método directo

```
Directo = function(n)
{
  u = runif(n)
  p = c(7/48,5/48,1/8,1/16,1/4,5/16)
  X=vector()
  for (i in 1:n){
    k=1
    while(1<2){
      pk = sum(p[1:k])
      if(pk > u[i]){
        X[i] = k
        break
      }
      k=k+1
    }
  }
  return(X)
}
X = Directo(1000)
hist(X,breaks=seq(0,6,1),freq=FALSE,main="Método Directo",ylab="f(x)",xlab="x")
x=c(0,1,1,2,2,3,3,4,4,5,5,6)
y = c(7/48,7/48,5/48,5/48,1/8,1/8,1/16,1/16,1/4,1/4,5/16,5/16)
lines(x,y,lwd=2,col="red")
```



Método de rechazo

```r
Rechazo = function(n)
{
  p = c(7/48,5/48,1/8,1/16,1/4,5/16)
  X = vector()
  for (i in 1:n)
    {
    while(1<2)
      {
      u1 = runif(1)
      u2 = runif(1)
      I = ceiling(6*u1)
      if(u2 < p[I])
        {
        X[i] = I
        break
      }
    }

  }
  return(X)
}

X = Rechazo(1000)
# Comparamos
hist(X, breaks=seq(0,6,1),main="Método de rechazo", freq=FALSE,ylab="f(x)",xlab="x")
x=c(0,1,1,2,2,3,3,4,4,5,5,6)
y = c(7/48,7/48,5/48,5/48,1/8,1/8,1/16,1/16,1/4,1/4,5/16,5/16)
lines(x,y,lwd=2,col="red")
```



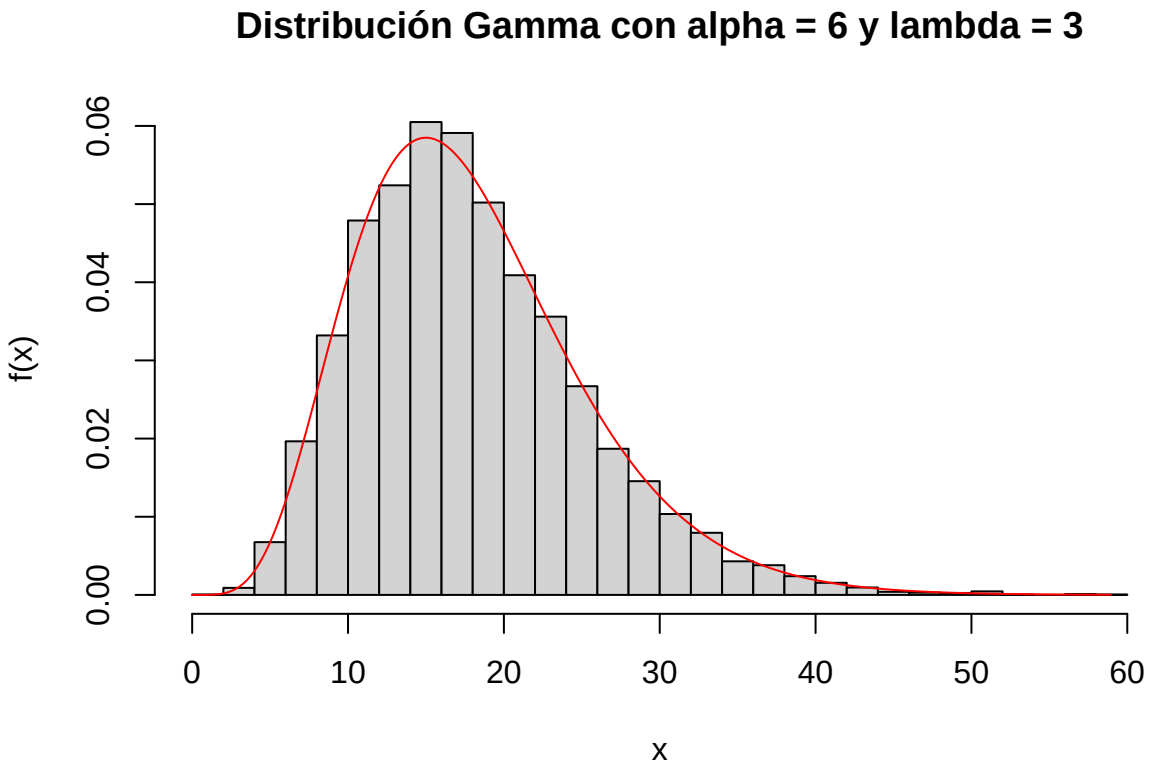
Ejercicios 3.4.8.

Implementar algoritmos para generar $Gama(\alpha, \beta)$ para todo $\alpha, \beta \in \mathbb{R}^+$. Seprar en tres casos:

1. α entero
2. $\alpha < 1$
3. $\alpha > 1$

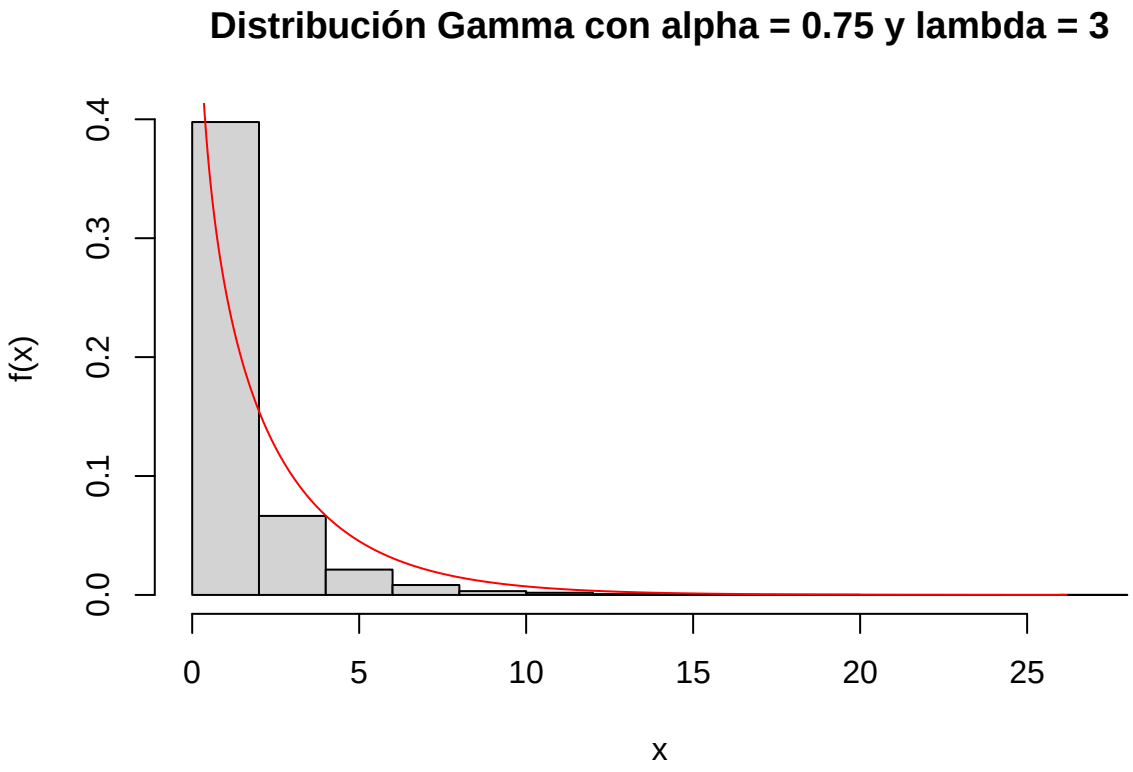
Para α entero

```
GammaEntero = function(n,alpha,lambda)
{
  X = vector()
  for(i in 1:n)
  {
    s = 0
    for(j in 1:alpha)
    {
      u = runif(1)
      s = s - log(u)
    }
    X[i] = s
  }
  X = X*lambda
  return(X)
}
n = 10000
alpha = 6
lambda= 3
X = GammaEntero(n, alpha, lambda)
# Comparamos
hist(X,breaks=25,freq=FALSE, main="Distribución Gamma con alpha = 6 y lambda = 3", ylab="f(x)",xlab="x"
x = seq(0, max(X), 0.01)
y = dgamma(x,shape=alpha,scale=lambda)
lines(x,y,col="red")
```



Para $\alpha < 1$

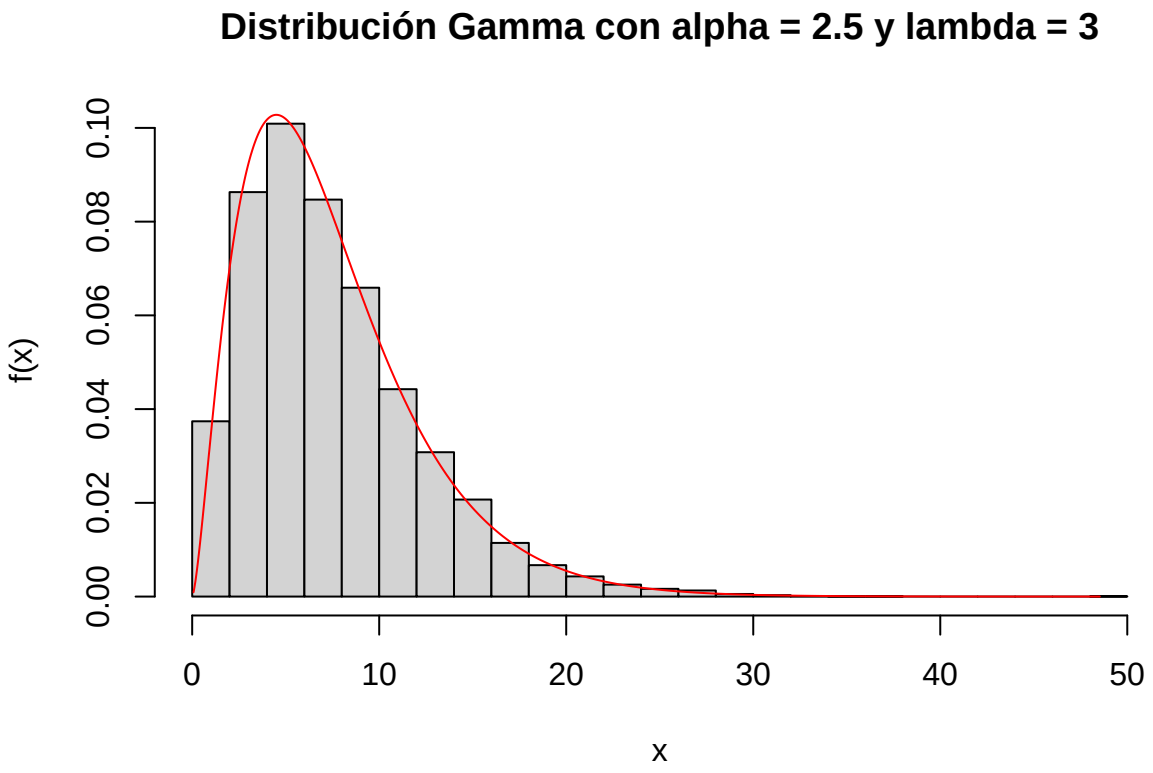
```
GammaMenor = function(n,alpha,lambda)
{
  u = runif(n)
  Y = GammaEntero(n,1+alpha,lambda)
  X = Y*u^{1/alpha}
}
# Comparamos
n = 10000
alpha = 0.75
lambda = 3
X = GammaMenor(n,alpha, lambda)
hist(X,breaks=10, freq=FALSE, main="Distribución Gamma con alpha = 0.75 y lambda = 3",ylab="f(x)",xlab=
x = seq(min(X), max(X), 0.01)
y = dgamma(x,shape=alpha,scale=lambda)
lines(x,y,col="red")
```



Para $\alpha > 1$

```
GammaMayor = function(n,alpha,lambda)
{
  a = floor(alpha)
  b = a/alpha
  c = b^(-a)*((alpha - a )/(exp(1)*(1-b)))^(alpha-a)
  X = vector()
  i=1
  while (i<=n)
  {
    x = GammaEntero(1,a,1/b)
    h=x^(alpha-a)*b^(-a)*exp(-x*(1-b))*factorial(a-1)
    Y = runif(1)
    if (Y<=h/c)
    {
      X[i] = x
      i = i+1
    }
  }
  X = X*lambda
  return(X)
}
# Comparamos
n = 10000
alpha = 2.5
lambda = 3
X = GammaMayor(n,alpha, lambda)
hist(X,breaks=25, freq=FALSE, main="Distribución Gamma con alpha = 2.5 y lambda = 3",ylab="f(x)",xlab="
x = seq(min(X), max(X), 0.01)
```

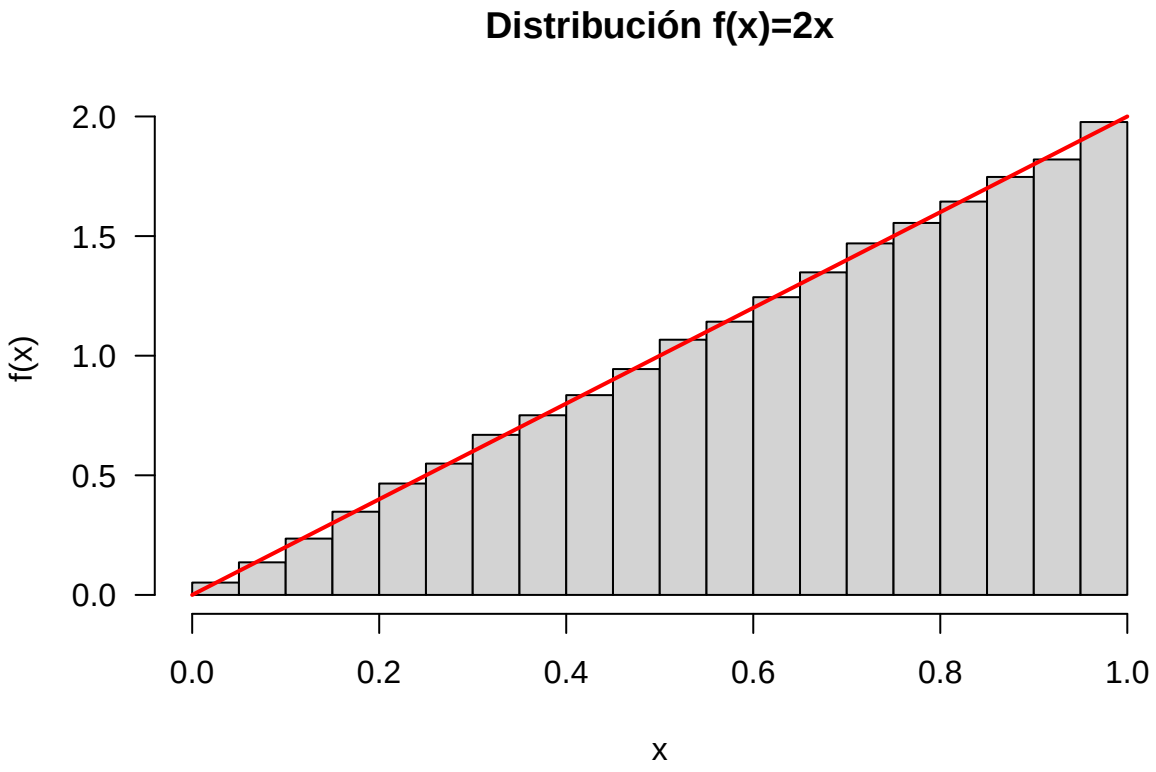
```
y = dgamma(x,shape=alpha,scale=lambda)
lines(x,y,col="red")
```



Ejercicios 3.4.10

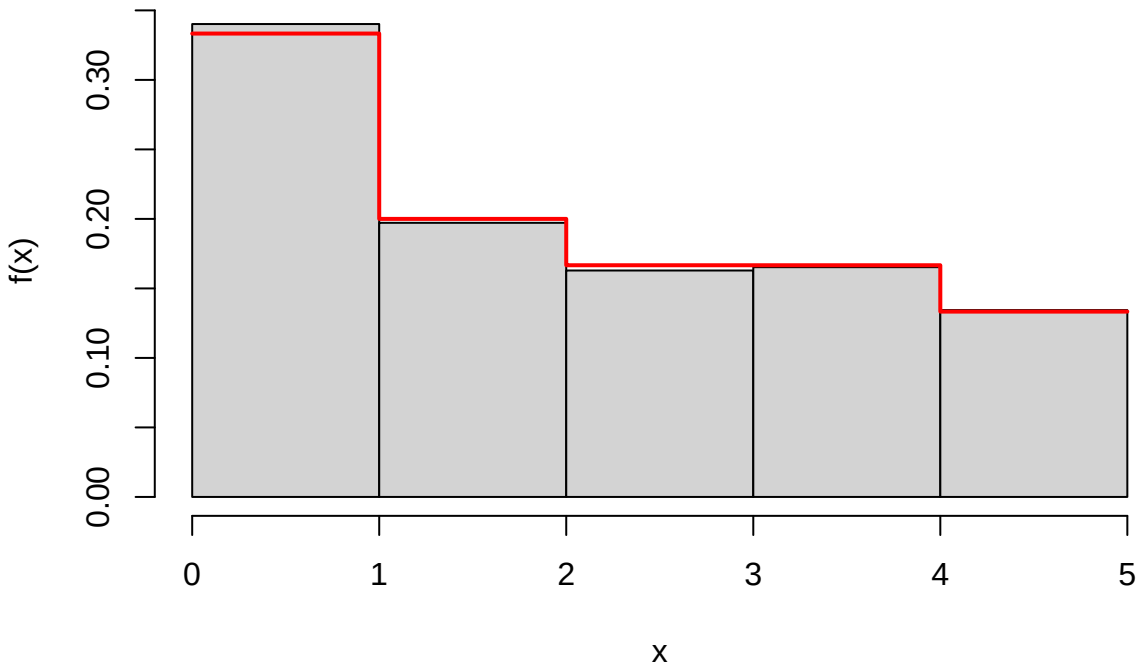
1. Considere una v.a. X con función de densidad $f(x) = 2xI_{(0,1)}(x)$. Escribir e implementar un algoritmo para generar valores de dicha variable.

```
n =100000
u=runif(n)
X=sqrt(u)

# Comparamos
hist(X,freq=FALSE,main="Distribución f(x)=2x",ylab="f(x)", xlab="x",las=1)
x=seq(0,1,0.0001)
y=2*x
lines(x,y,lwd=2,col="red")
```



2. Considere una v.a. X, con soporte en el conjunto $\{1, 2, 3, 4, 5\}$ y distribución $\{1/3, 1/5, 1/6, 1/6, 4/30\}$. Implementar un algoritmo para generar una muestra de 1000 elementos de dicha distribución.

```
N = 10000
p=c(1/3,1/5,1/6,1/6,4/30)
X=c()
for (i in 1:N)
{
  while(1<2)
  {
    u1 = runif(1)
    u2 = runif(1)
    j = ceiling(5*u1)
    if(u2 < p[j])
    {
     X[i] = j
     break
    }
  }
}
# Comparamos
hist(X, breaks=seq(0,5,1), main="Distribución discreta", freq=FALSE, ylab="f(x)",xlab="x")
x=c(0,1,1,2,2,3,3,4,4,5)
y = c(1/3,1/3,1/5,1/5,1/6,1/6,1/6,1/6,4/30,4/30)
lines(x,y,lwd=2,col="red")
```

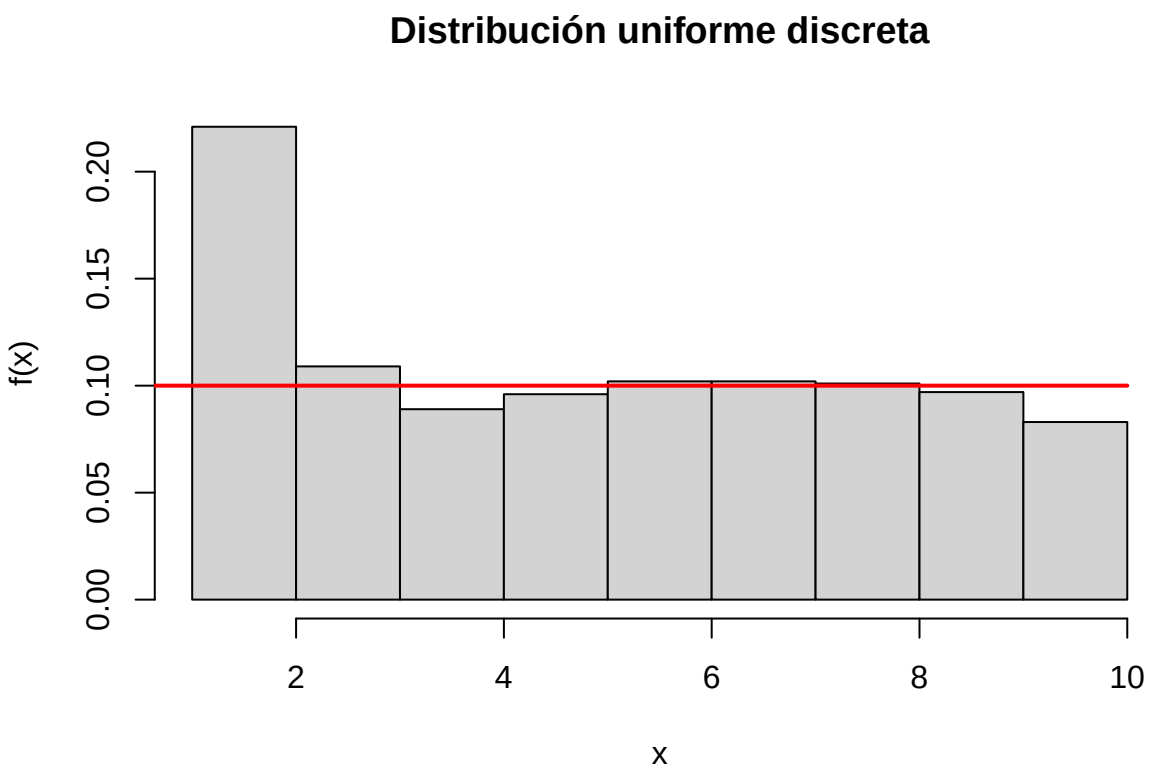


3. Implementar los algoritmos de las siguientes v.a.:
   (a) Uniforme discreta con soporte en los primeros 10 naturales
   (b) Binomial con $n = 10$ y $p = \frac{1}{2}$
   (c) Poisson(9)

Para la distribución uniforme discreta con soporte en los primeros 10 naturales se tiene el siguiente código.

```
Uniforme = function(n,a,b){
  X = vector()
  for (i in 1:n){
    u = runif(1)
    for(j in a:b){
      min = (j-1)/b
      max = j/b
      if(min<=u & max>u){
        X[i] = j
        break
      }
    }
  }
  return(X)
}

X = Uniforme(1000,1,10)
#Comaparamos
hist(X, main = "Distribución uniforme discreta", freq=FALSE, ylab="f(x)", xlab="x")
 x=c(0,10)
```

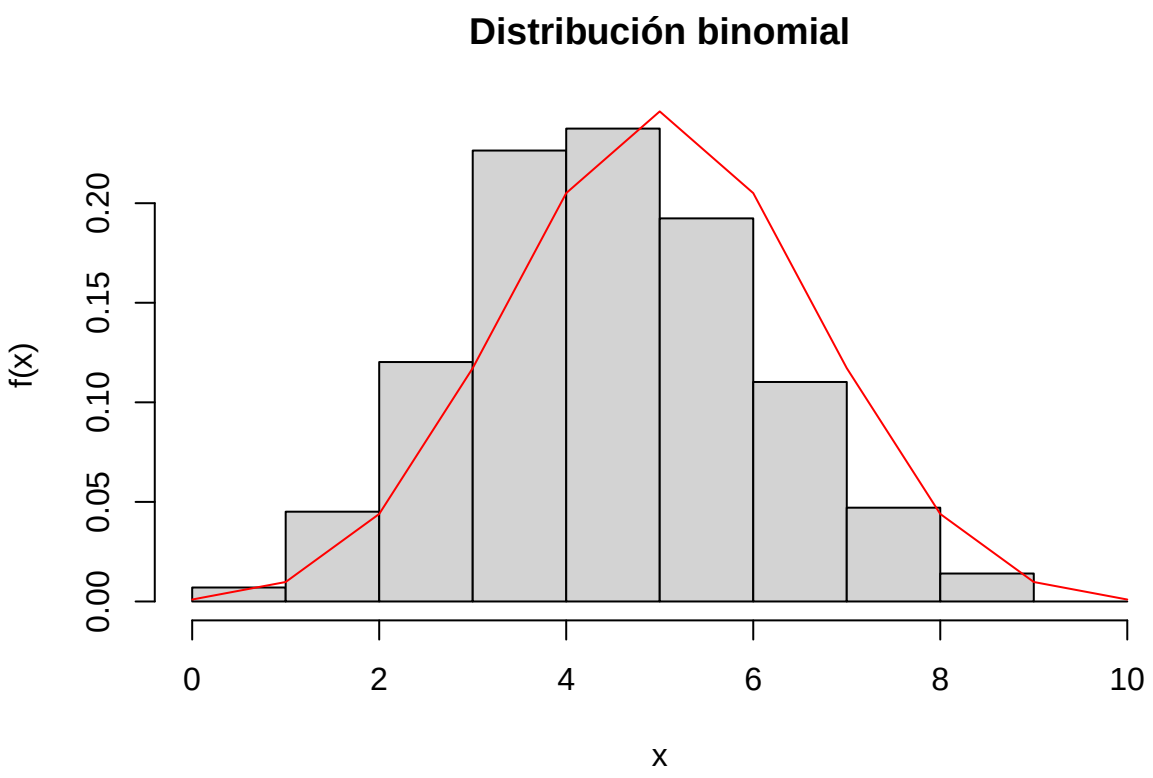
```
y=c(1/10,1/10)
lines(x,y,lwd=2,col="red")
```



Distribución binomial con $n = 10$ y $p = \frac{1}{2}$.

```
binomial=function(k,n,p)
{
  X=c()
  c = p/(1-p)
  for(j in 1:k){
    u = runif(1)
    theta = (1-p)^n
    f = theta
    for(i in 0:(n-1)){
      if(u<=f){
        X[j] = i
        break
      }
      theta = (theta*c*(n-i))/(i+1)
      f = f+theta
    }
  }
  return(X)
}

X = binomial(1000, 10, 0.5)
# Comparamos
hist(X, breaks=seq(0,10,1), main="Distribución binomial", freq=FALSE, ylab="f(x)", xlab="x")
x = seq(0, 10, 1)
```

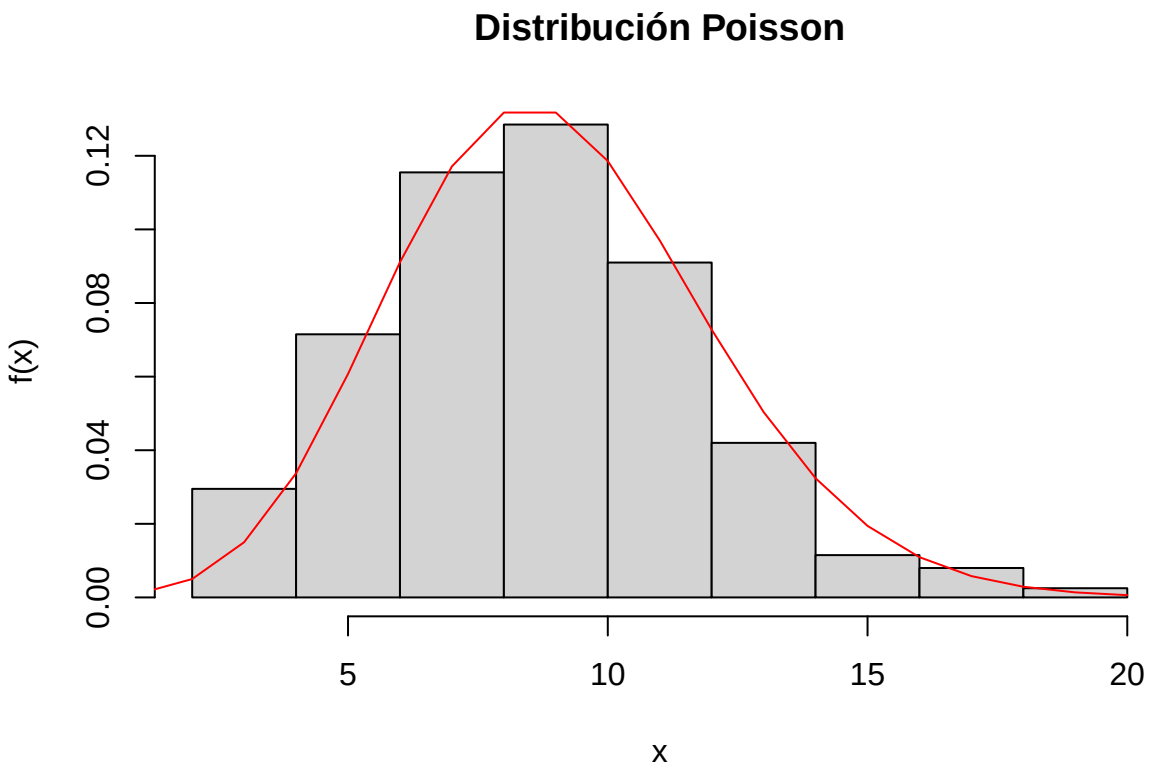
```
y = dbinom(x,10, 0.5)
lines(x,y, col="red")
```



Distribución Poisson con parámetro 9

```
DistPoisson = function(n,lambda)
{
  u = runif(n)
  X=vector()
  for (j in 1:n){
    i=0
    cont = 0
    p=exp(-lambda)
    f = p
    while(cont==0){
      if(u[j]<f)
      {
        X[j] = i
        cont = 1
      }
      p =lambda*p/(i+1)
      f = f+p
      i = i+1
      if(i>10000){
        cont= 1
        X[j]=i
      }
    }
  }
```

```
  return(X)
}

X = DistPoisson(1000, 9)
#Comparamos
hist(X, main="Distribución Poisson", freq=FALSE, ylab="f(x)",xlab="x")
x = seq(0, max(X),1)
y = dpois(x,9)
lines(x,y,col="red")
```

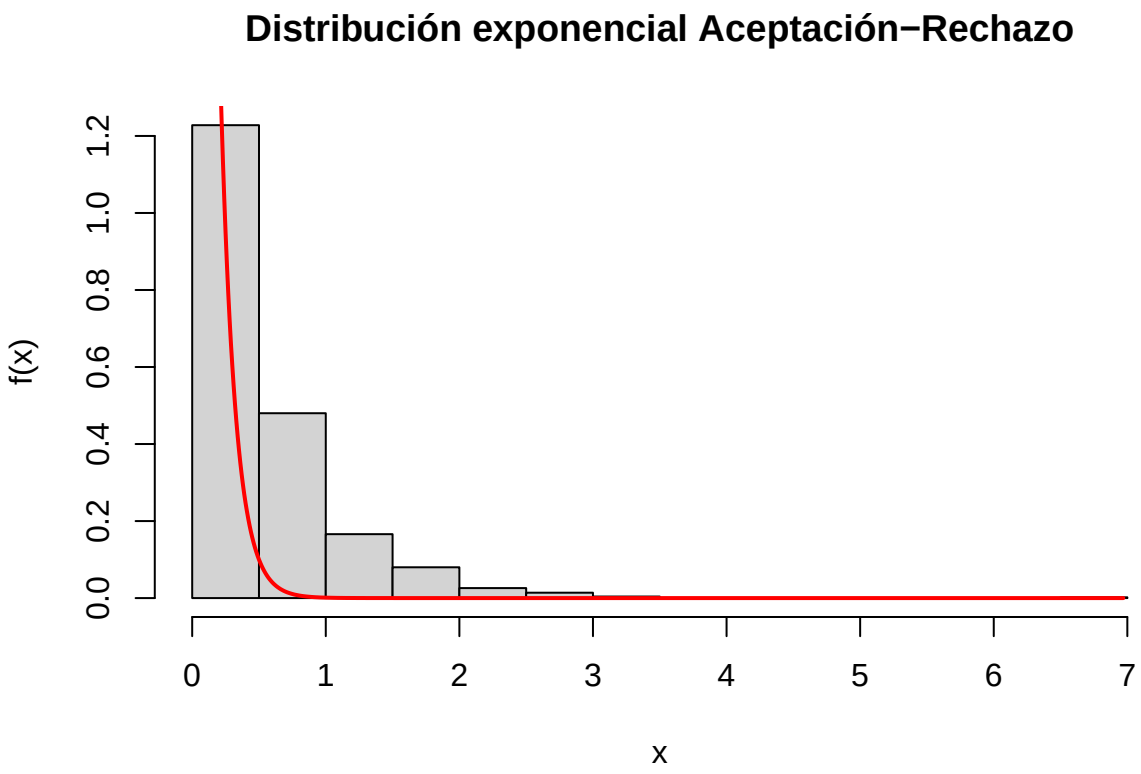


4. Generar muestras de tamaño 1000 para X con distribución $exp(9)$ usando:
   (a) Aceptación-rechazo
   (b) Transformada inversa
   (c) Cociente de uniforme

Aceptación-rechazo distribución exponencial

```
ExpAceptRech = function(n,lambda){
  lambda = 1/lambda
  X = vector()
  for(i in 1:n){
    c=1/exp(1)
    while(1<2)
      {
      u1=runif(1)
      u2=runif(1)
      Y=-log(u1)
```

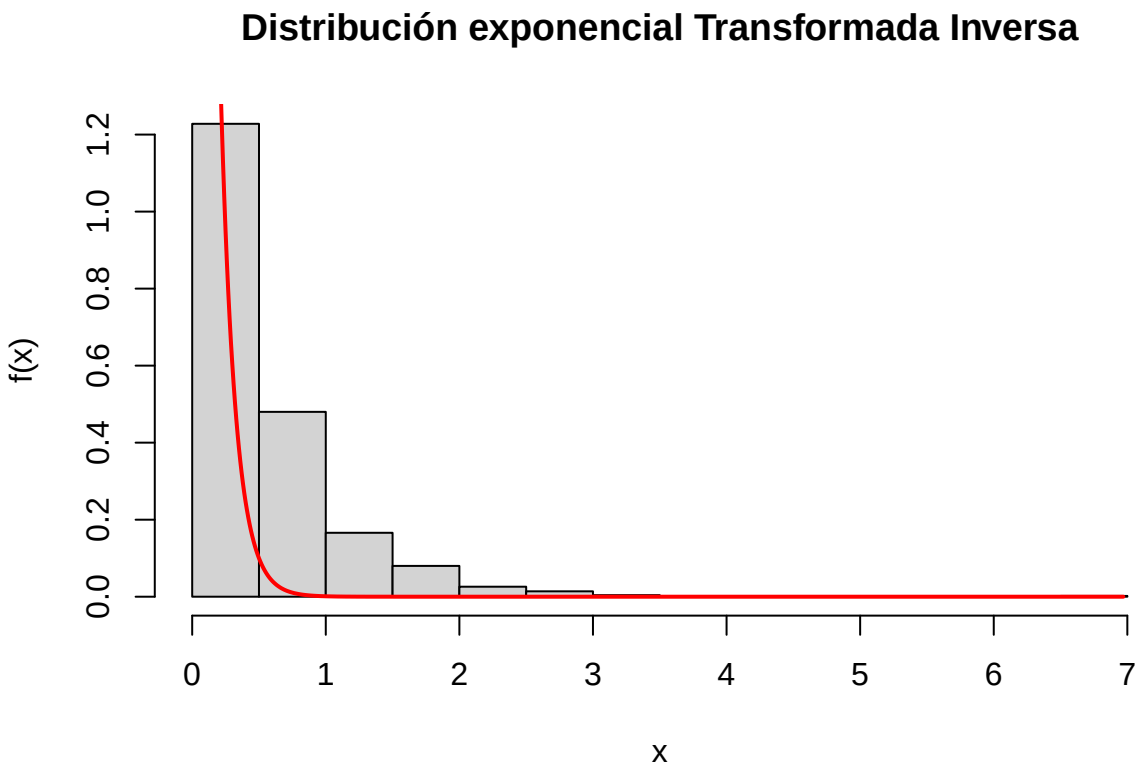
```
      if(u2<=(lambda*exp(-lambda*u1)/c*u1)){
        X[i] = Y
        break
      }
    }
  }
  return(X)
}

X = ExpAceptRech(1000,9)
hist(X, freq=FALSE,main="Distribución exponencial Aceptación-Rechazo",ylab="f(x)",xlab = "x")

x=seq(0,max(X),0.01)
y = dexp(x,9)
lines(x,y,lwd=2,col="red")
```



Transformada inversa

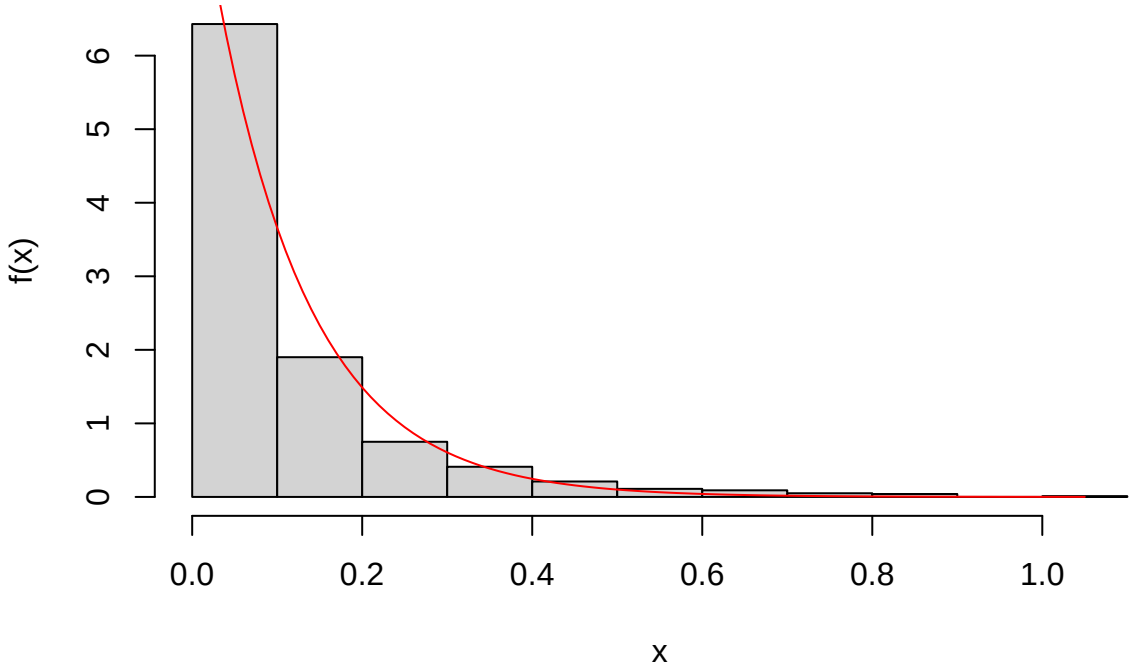
```
ExpTransfInv = function(n,lambda)
{
  u = runif(n,0,1)
  X = (-1/lambda)* log(1-u)
  return (X)
}
lambda = 9
n = 1000
hist(X,freq=FALSE, main="Distribución exponencial Transformada Inversa", ylab = "f(x)", xlab="x")
x=seq(0,max(X),0.01)
y = dexp(x,9)
```

```
lines(x,y,lwd=2,col="red")
```



Cociente uniforme

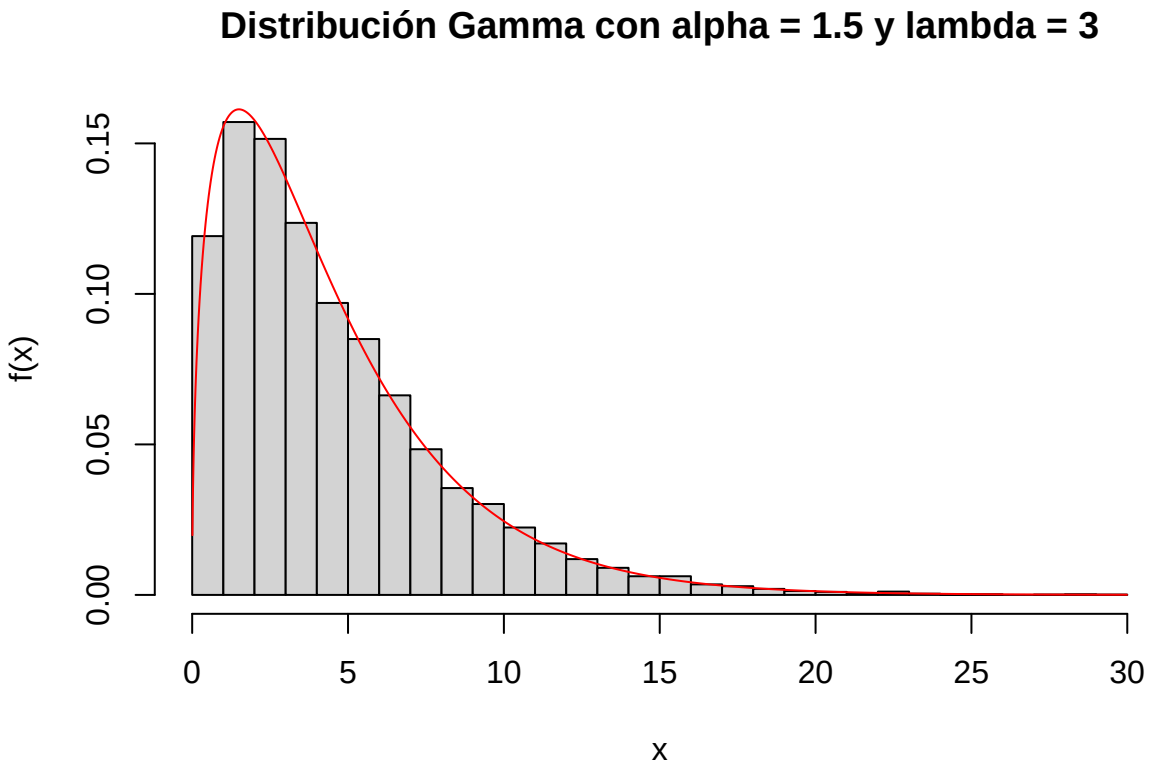
```
ExpCoUnif = function(n, lambda)
{
  u = runif(n,0,sqrt(lambda))
  v = runif(n,0,-(u/lambda)*(2*log(u) - log(lambda)))
  X = v/u
  return (X)
}

X = ExpCoUnif(1000, 9)
#Comparando
hist(X,freq=FALSE, main="Distribución exponencial Cociente de uniformes",ylab="f(x)",xlab="x")
x=seq(0,max(X),0.01)
y=dgamma(x,shape=1,rate=9)
lines(x,y,col="red")
```



5. Implementar un algoritmo para la distribución Gamma con parámetros:
   (a) (1.5, 3)
   (b) (0.5, 6)

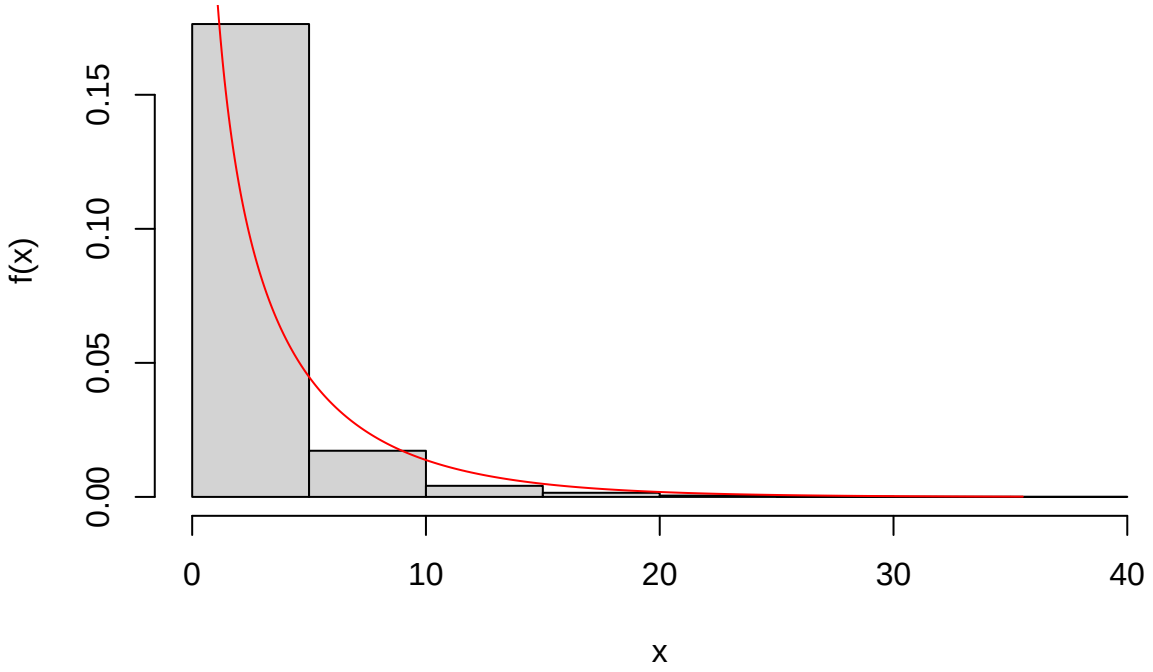
Para el primer caso tenemos que $\alpha = 1.5$, por lo tanto utilizamos la función GammaMayor que definimos previamente

```
# Hallamos la solución
n=10000
alpha = 1.5
lambda = 3
X = GammaMayor(n,alpha,lambda)
# Comparamos
hist(X,breaks=25, freq=FALSE, main="Distribución Gamma con alpha = 1.5 y lambda = 3",ylab="f(x)",xlab="
x = seq(min(X), max(X), 0.01)
y = dgamma(x,shape=alpha,scale=lambda)
lines(x,y,col="red")
```



Por otro lado, para el inciso 2) $\alpha < 1$, por lo tanto se utiliza la función GammaMenor.

```
# Hallamos la solución
n = 10000
alpha = 0.5
lambda = 6
X = GammaMenor(n,alpha, lambda)
# Comparamos
hist(X,breaks=10, freq=FALSE, main="Distribución Gamma con alpha = 0.5 y lambda = 6",ylab="f(x)",xlab="
x = seq(min(X), max(X), 0.01)
y = dgamma(x,shape=alpha,scale=lambda)
lines(x,y,col="red")
```



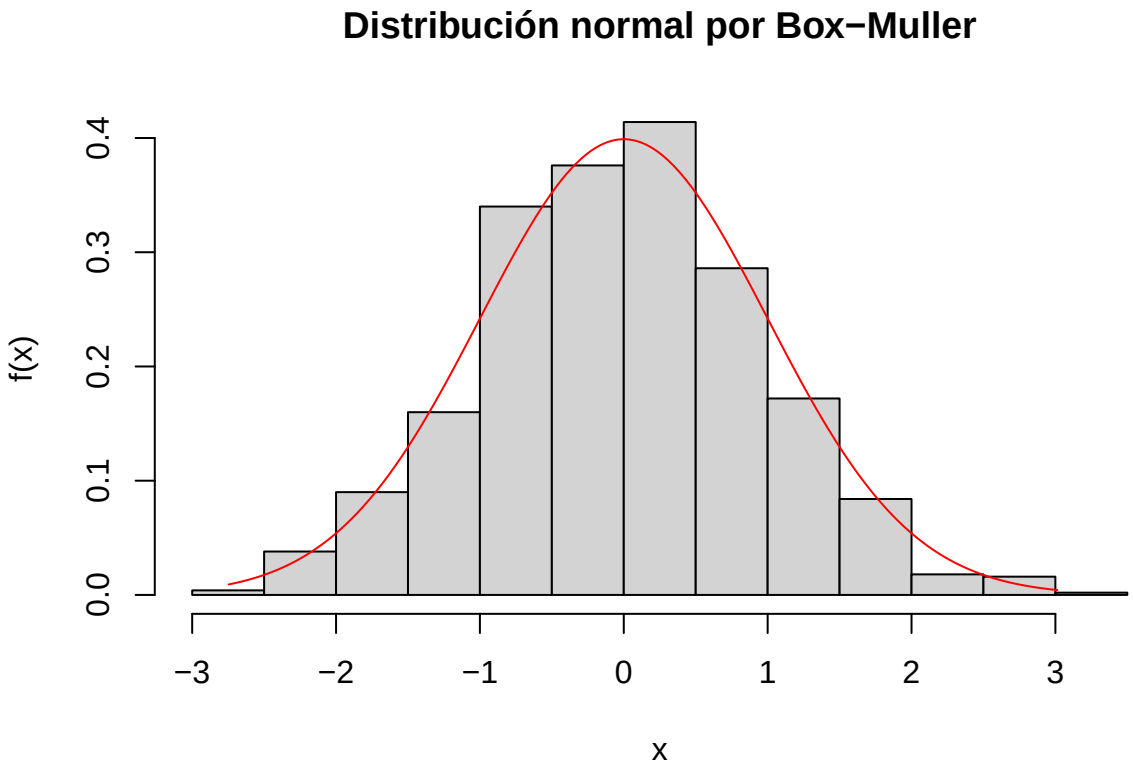
6. Implementar el algoritmo Box-Muller.

El algoritmo de Box-Muller es el siguiente

```
BoxMuller = function(N, media, varianza)
{
  U1 = runif(N,0,1)
  U2 = runif(N,0,1)
  X = ((-2*log(U1))^(1/2))*cos(2*pi*U2)
  X = media + varianza*X
  return (X)
}
```

Para verificar que nuestro algoritmo de Box-Muller da buenas aproximaciones, hagamos una prueba generando una distribución normal estandar.

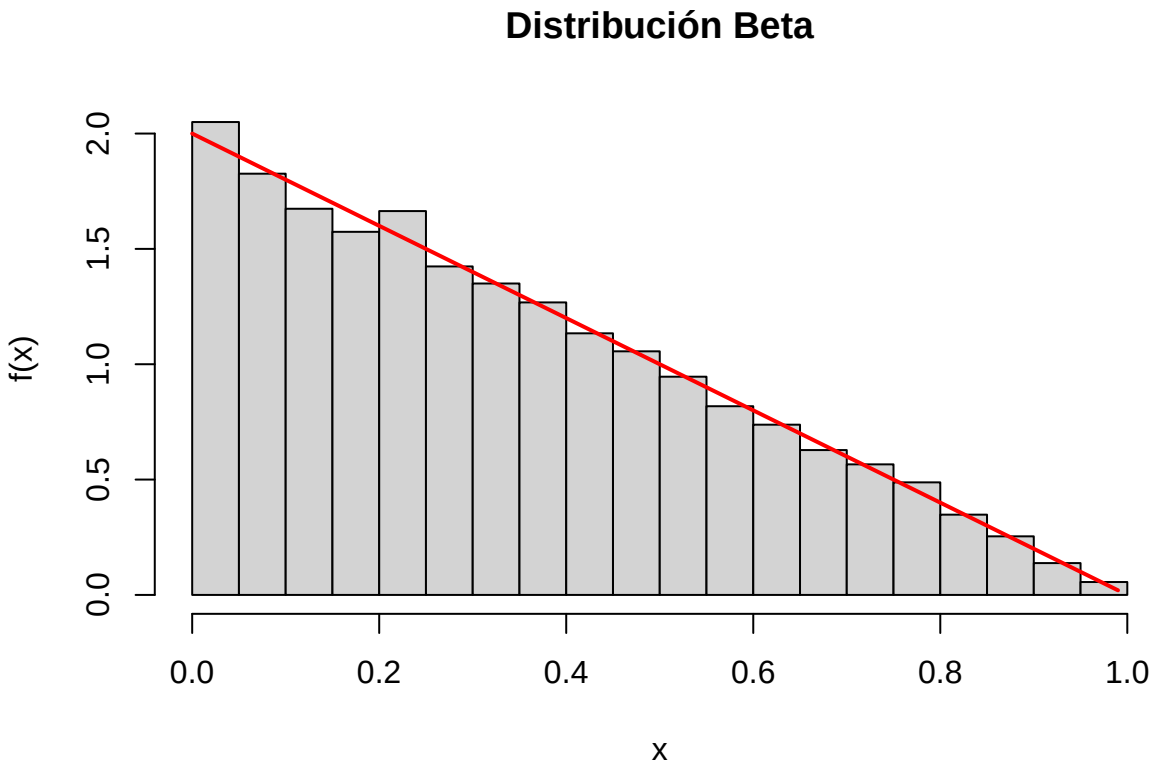
```
N= 1000
media = 0
varianza =1
X = BoxMuller(N, media, varianza)
# Comparamos
hist(X, fre=FALSE, main="Distribución normal por Box-Muller",ylab="f(x)",xlab="x")
x = seq(min(X), max(X), 0.01)
y = dnorm(x,media,varianza)
lines(x,y,col="red")
```



7. Implementar el algoritmo para generar una muestra de v.a. $Beta(1, 2)$

A continuación se muestra el código para generar distribuciones Beta.

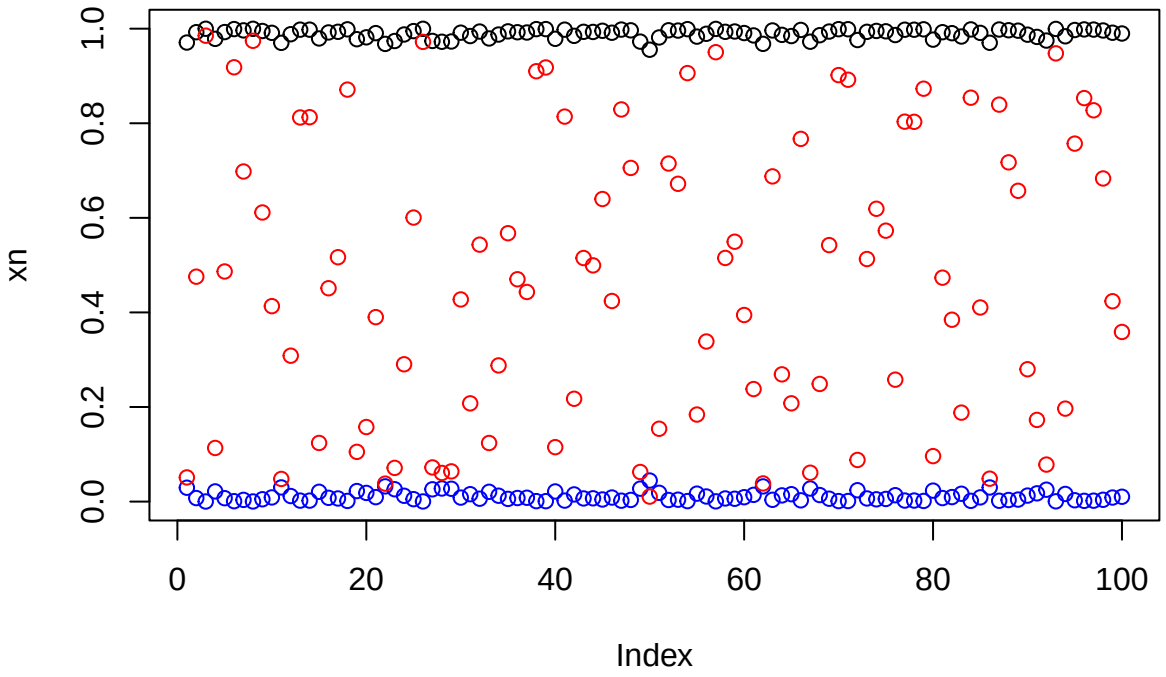
```
DistBeta = function(n,alpha,beta)
{
  Y1 = GammaEntero(n,alpha,1)
  Y2 = GammaEntero(n,beta,1)
  X = Y1/(Y1+Y2)
  return(X)
}

# Hallamos la dis Beta
n=10000
alpha=1
beta=2
X = DistBeta(n,alpha,beta)
#Comparamos
hist(X,main="Distribución Beta",freq=FALSE,ylab="f(x)",xlab="x")
x = seq(min(X),max(X),0.01)
y = dbeta(x,alpha,beta)
lines(x,y,col="red",lwd=2)
```



Otros códigos

Por último, se anexan algunos de los códigos realizados en clase.

```
n = 100
set.seed(1234)
set.seed(NULL) #reestablecer semilla
#Generar estadisticos de orden
u = runif(n,0, 1)
x1 = 1 - u^(1/n)
xn = u^(1/n)
plot(xn,ylim=c(0,1))
points(x1, col="blue")
points(u, col="red")
```



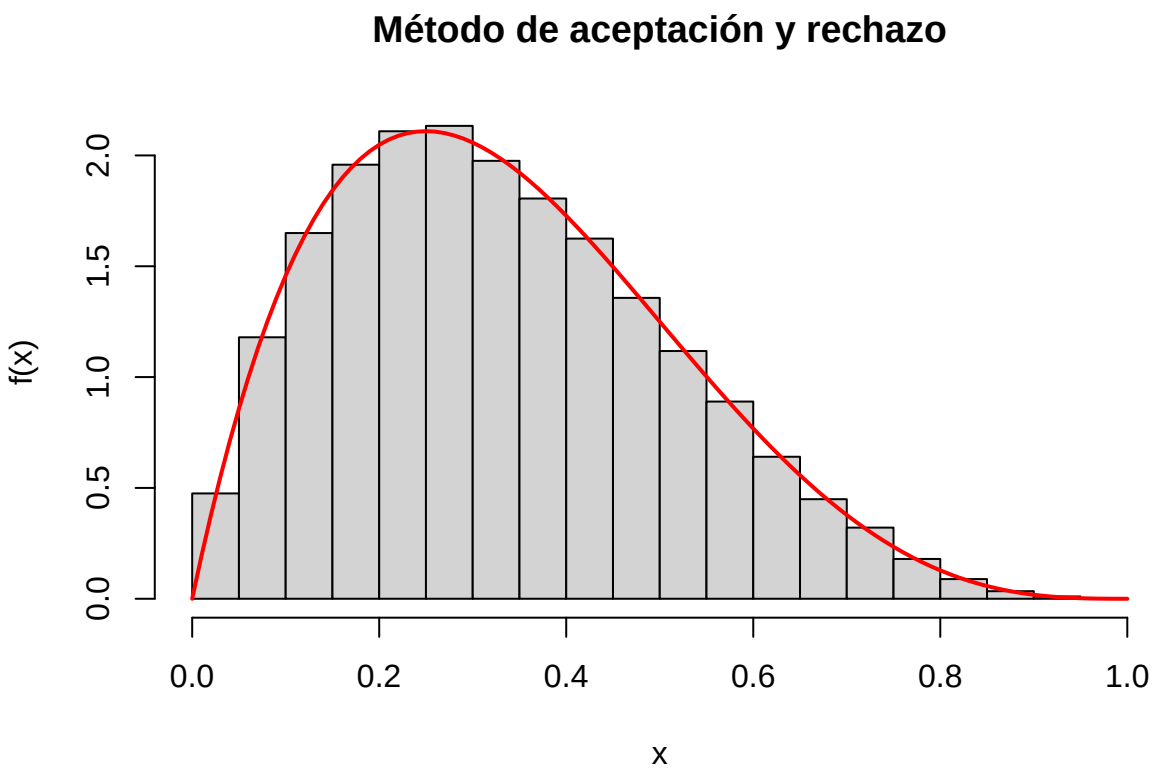
Distribución multinomial

```
DMulti = function(N,n ,p)
{
  #N es el número de elementos de la muestra que queremos hallar
  #n es el número de sucesos
  #r es el número de variables aleatorias
  #p es el vector que contiene las probabilidades p_i con i=1, . .,r
  # se debe cumplir que x_1 + . . . + x_r = n
  #Creamos la matriz de la muestra
  r = length(p)
  X = matrix(data = NA, nrow= N, ncol = r, byrow =  TRUE)
  #El primer for es para obtener el numero de muestras
  for (i in 1:N)
  {
    #Generamos la v.a Y_j de P(Y_j = i) con i=1,. . .,r y j=1, . . .,n
    Y = c()
    s = replicate(r,0)
    for (j in 1:n)
    {
      Y[j] = sample(1:r, size=1)
    }
    for (k in 1:r)
    {
      for (l in 1:n)
      {
        if (Y[l] == k)
        {
          s[k] = s[k]+1
        }
      }
    }
    X[k] = s[k]
```

```
  }
  return(X)
}
```

Método de aceptación y rechazo para $f(x) = 20x(1-x)^3$

```
k = 100000
X = c()
for (j in 1:k){
  aux=0
  while(aux==0){
    U1 = runif(1)
    U2 = runif(1)
    if (U2 <= (256*U1*(1-U1)^3)/27){
      X[j] = U1
      break
    }
  }
}

hist(X,main="Método de aceptación y rechazo",freq=FALSE,ylab="f(x)",xlab="x")
s=seq(0,1,0.01)
lines(s,20*s*(1-s)^3,lwd=2,col="red")
```



Método de composición para $f(x) = \frac{5}{12}(1+(x-1)^4)$

```
MetComp = function(n)
{
  X = vector()
```

```
  for (i in 1:n)
  {
    u = runif(1)
    if (u > 1/6 & u<5/6)
    {
      X[i] = 2*runif(1)
    }
    else
    {
      if (u < 1/6)
      {
       X[i] = -((2*runif(1) - 1)^(1/5)) + 1
      }
      else
      {
       X[i] = -((2*runif(1) - 1)^(1/5)) + 1
      }
    }
  }
  return (X)
}
X = MetComp(1000)
hist(X, freq = FALSE,main="Método de composición",ylab="f(x)",xlab="x")
x=seq(0,2,0.001)
y=(5/12)*(1+(x-1)^4)
lines(x,y,col="red4",lwd=3)
```