Tarea 2 Estadistica (C24343_C24459_C20459)

José Eduardo López Corella, Felix Madrigal Mora, Gustavo Amador Fonseca

2024-10-22

Pregunta 1

```r
set.seed(73)
# Función f a integrar
f <- function(x) {
  exp(-x^2) / (1 + x^2)
}

# Usamos integrate para calcular el valor exacto de la integral en [0, 1]
resultado <- integrate(f, lower = 0, upper = 1)
valor_exacto <- resultado$value

n_values <- c(3, 100, 1000, 10000, 100000) # Tomamos solo algunos valores n acá pues el comportamiento

diferencias <- numeric(length(n_values))

# Para cada valor n anterior, calculamos la estimación Monte Carlo y la diferencia
for (i in 1:length(n_values)) {
  n <- n_values[i]
    x <- runif(n, min = 0, max = 1)
    fx <- f(x)
    monte_carlo_integral <- mean(fx)
  diferencia_absoluta <- abs(monte_carlo_integral - valor_exacto)
    diferencias[i] <- diferencia_absoluta
      cat("\nPara n =", n, ":\n")
  cat("Estimación por Monte Carlo:", monte_carlo_integral, "\n")
  cat("Valor de la integral usando integrate:", valor_exacto, "\n")
  cat("Diferencia absoluta entre los resultados:", diferencia_absoluta, "\n")
}
```

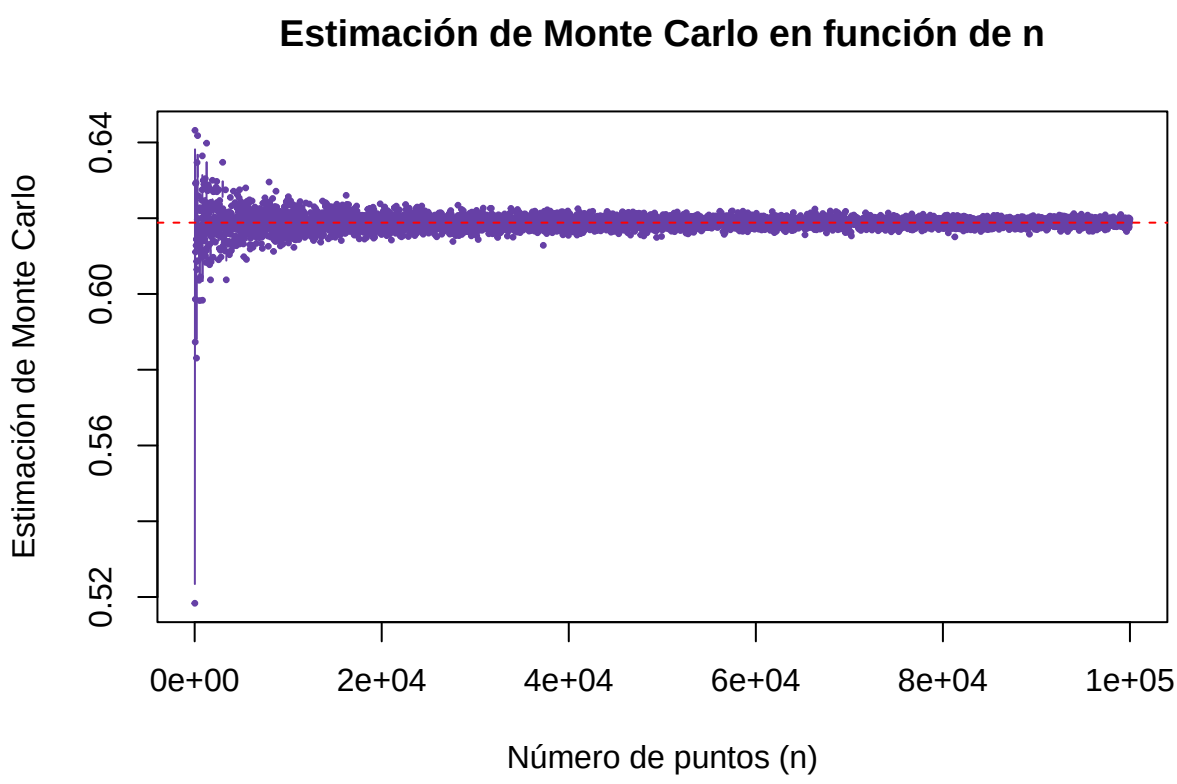
```
##
## Para n = 3 :
## Estimación por Monte Carlo: 0.7238379
## Valor de la integral usando integrate: 0.618822
## Diferencia absoluta entre los resultados: 0.1050159
##
## Para n = 100 :
## Estimación por Monte Carlo: 0.5355188
## Valor de la integral usando integrate: 0.618822
```

```
## Diferencia absoluta entre los resultados: 0.08330315
##
## Para n = 1000 :
## Estimación por Monte Carlo: 0.6362127
## Valor de la integral usando integrate: 0.618822
## Diferencia absoluta entre los resultados: 0.01739072
##
## Para n = 10000 :
## Estimación por Monte Carlo: 0.6231183
## Valor de la integral usando integrate: 0.618822
## Diferencia absoluta entre los resultados: 0.004296331
##
## Para n = 1e+05 :
## Estimación por Monte Carlo: 0.6192179
## Valor de la integral usando integrate: 0.618822
## Diferencia absoluta entre los resultados: 0.000395936
```

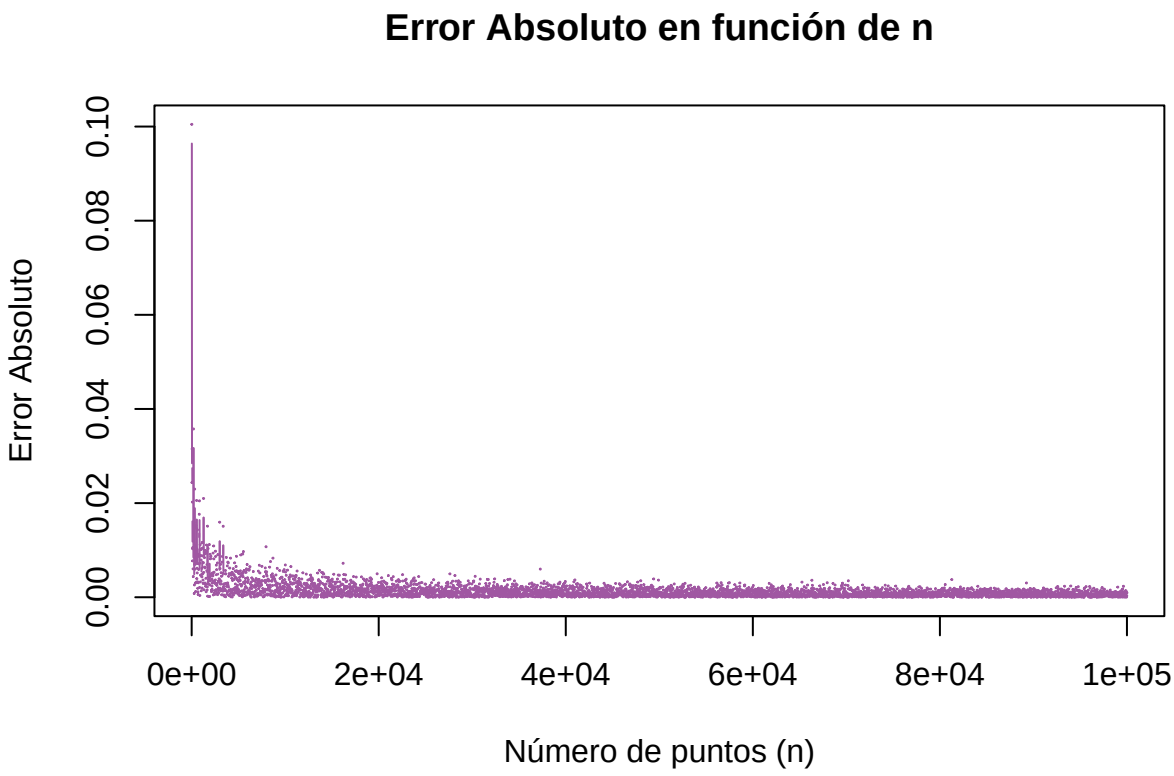
Con n = 1e+05 vemos que se satisface lo deseado.

Estimación de montecarlo de hasta n=100000 y su abs error (tomando cada 20 puntos)

```
plot(seq(20, n, by = 20), monte_carlo_estimaciones, type = "b", col = "#6640A6", pch = 20, cex = 0.5,
     xlab = "Número de puntos (n)", ylab = "Estimación de Monte Carlo",
     main = "Estimación de Monte Carlo en función de n")
abline(h = valor_exacto, col = "red", lty = 2)
```




```r
plot(seq(20, n, by = 20), error_absoluto, type = "b", col = "#A057A2", pch = 20, cex = 0.1,
     xlab = "Número de puntos (n)", ylab = "Error Absoluto",
     main = "Error Absoluto en función de n")
```



Pregunta 2

a)

```r
# Definimos la función de densidad de la pérdida L con distribución exponencial
f_L <- function(L, lambda = 1) {
  return(lambda * exp(-lambda * L))
}
```

b)

```r
set.seed(54321)
n <- 10^4

mu <- 3
sigma <- 2

#sacamos las muestras con distribucion normal
muestras_g <- rnorm(n, mean = mu , sd = sigma)
#para constuir las muestras se usa una distribucion normal, sin embargo la funcion de perdida es para l
#exponenciales, por esa razon se restringe a mayores a 0
```

```r
muestras_g <- muestras_g[muestras_g > 0]

#sacamos la densidad de g y f en base a las muestras
g_L_valores <- dnorm(muestras_g, mean = mu, sd = sigma)
f_L_valores <- f_L(muestras_g)

valor_esperado <-mean( muestras_g *  (f_L_valores / g_L_valores))
valor_esperado
```

```
## [1] 1.069952
```

Pregunta 3

a)

```r
#Dadas las muestras:
muestras <- c(2.72, 1.93, 1.76, 0.49, 6.12, 0.43, 4.01, 1.71, 2.01, 5.96)
```

Note que la función priori está dada por:

$$\pi(\lambda) \sim \gamma(2,1) \quad \Rightarrow \quad \gamma(2,1) = \lambda e^{-\lambda}$$

Y la función de verosimilitud según el enunciado:

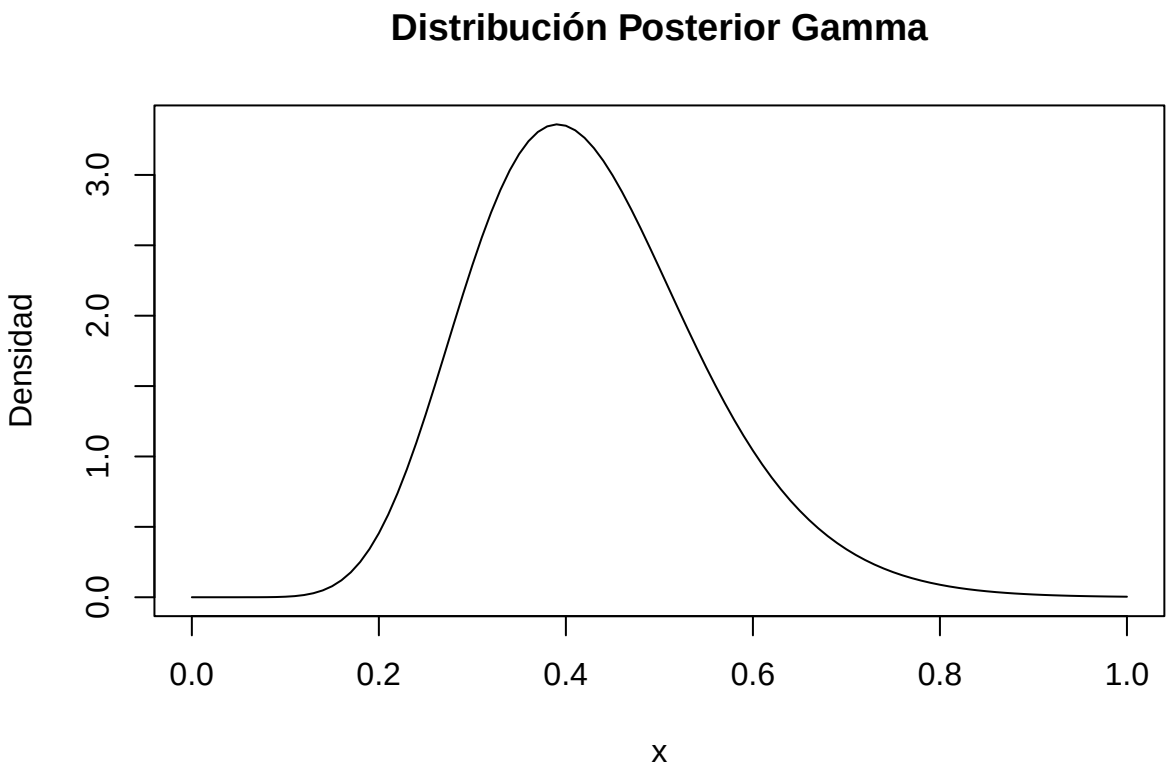
$$L(x|\lambda) \sim \text{Exp}(\lambda) \quad \Rightarrow \quad L(x|\lambda) = \lambda^n e^{-\lambda \sum_{i=1}^{n} x_i}$$

Entonces, por medio de teoría bayesiana se obtiene que la función posteriori:

$$f(\lambda|x) \propto \lambda e^{-\lambda} \cdot \lambda e^{-\lambda \sum_{i=1}^{n} x_i} = \lambda^{(n+1)} e^{-(1+\Sigma x)}$$

donde $n = 10 \quad \wedge \quad \Sigma x = 27.14$.
Por lo tanto, se tiene que la función posteriori es una distribución gamma con parámetros $\alpha = 12$ y $\beta = 28.14$.

```r
#Por teoría bayesiana se optiene que la distribución posteriore es una gamma(12,1+ suma de las muestras
alpha <- 12
beta <- 1 + sum(muestras)

# Función de densidad de la posterior Gamma(alpha, beta) es la función objetivo
f_obj <- Vectorize(function(x) {
  ((beta^alpha) / factorial(alpha - 1)) * x^(alpha - 1) * exp(-beta * x)
})

# Graficar la función de densidad
curve(f_obj, from = 0, to = 1, ylab = "Densidad", main = "Distribución Posterior Gamma")
```




```r
#Calculo del maximo
pto_max <- optimize(f_obj, interval = c(0, 1), maximum = TRUE)
c <- pto_max$objective
print(c)
```

```
## [1] 3.359299
```

Se decidió tomar una función envolvente $g(\lambda) = U[0, 1]$, por ende en el codigo solo se uliza el valor de 1 en la comparación.

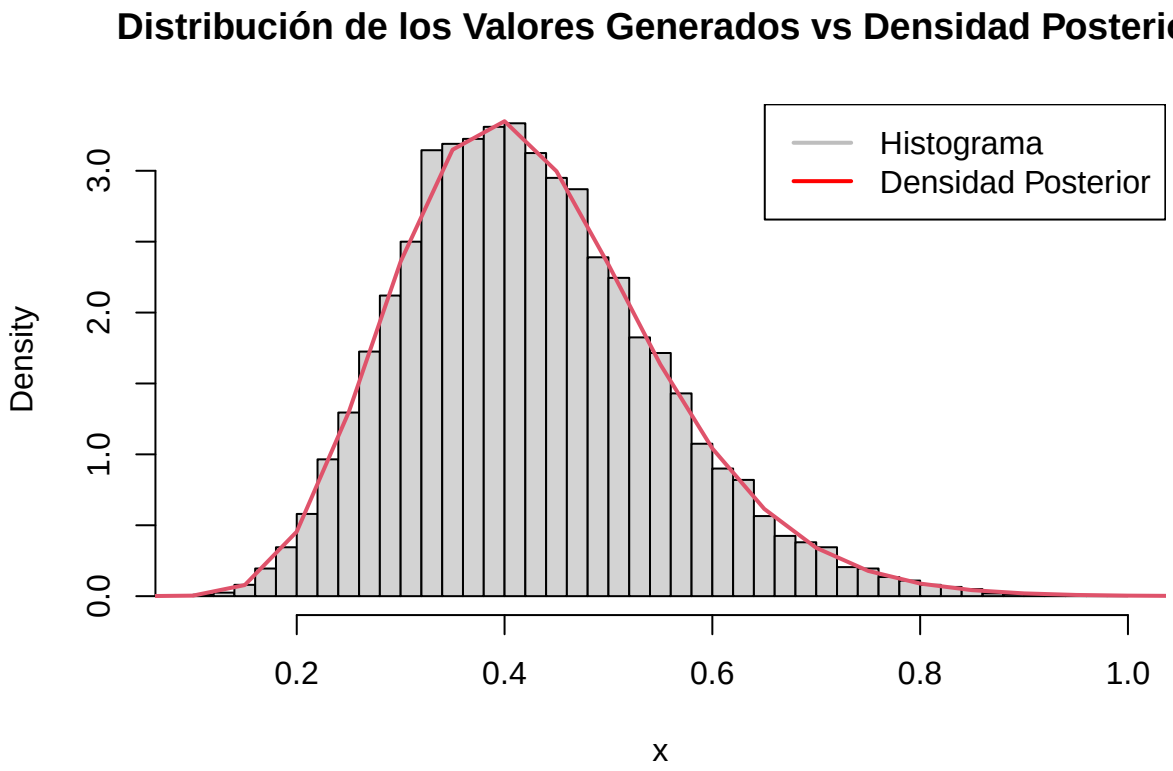
```r
# Algoritmo de Aceptación y Rechazo
set.seed(2023)
U <- runif(10000)
x <- runif(10000, 0, 1) #Entre cero y uno porque es donde se encuentra principalmente la densidad
ngen = length(x)

DIB <- f_obj(x)
for (i in 1:10000) {
  while ((U[i] * c * 1) >= DIB[i]) {  # Sustituimos los rechazados
    U[i] <- runif(1)
    x[i] <- runif(1)
    DIB[i] <- f_obj(x[i])
    ngen <- ngen + 1
  }
}
```

```r
# Estimación de lambda
lambda_est = mean(x)
cat("Valor estimado de lambda:", lambda_est, "\n")
```

```
## Valor estimado de lambda: 0.4252807
```

b)

```r
# Graficar el histograma de los valores generados y la densidad teórica
hist(x, breaks = "FD", freq = FALSE, main = "Distribución de los Valores Generados vs Densidad Posterio
curve(f_obj, from = 0, to = 5, col = 2, lwd = 2, add = TRUE)
legend("topright", legend = c("Histograma", "Densidad Posterior"), col = c("gray", "red"), lty = 1, lwd
```



c)

```r
# Proporción de rechazos
density_values = f_obj(x)
cat("Número medio de aceptados =", ngen / 10000, "\n")
```

```
## Número medio de aceptados = 3.3525
```

```r
cat("Número de generaciones =", ngen, "\n")
```

```
## Número de generaciones = 33525
```

```r
cat("Proporción de rechazos =", 1 - 10000 / ngen, "\n")
```

```
## Proporción de rechazos = 0.7017151
```

d)

```r
q <- quantile(x, c(0.005, 0.9955))
q
```

```
##      0.5%    99.55%
## 0.1748033 0.8064021
```

e)

Si lambda fuera 0,5 no se rechazaria la hipotesis porque el valor estaria dentro del intervalo de confianza, por lo que no se tiene la suficiente informacion para poder descartar la hipotesis y mas bien es un valor de lambda que puede ser admisible ya que cae dentro del intervalo de confianza del 99%

Pregunta 4

a)

```r
set.seed(73)
resim <- function(f, alpha, s0, niter, mini=0, maxi=10) {
  s_n <- s0
  estados <- rep(0, niter)

  for (k in 1:niter) {
    estados[k] <- s_n
    T <- (1 - alpha)^k # Se reduce el T según el número de iteraciones
    s_new <- rnorm(1, s_n, 1)

    s_new <- max(mini, min(maxi, s_new))

    dif <- f(s_new) - f(s_n)
    if (dif < 0) {
      s_n <- s_new
    } else {
      random <- runif(1)
      if (random < exp(-dif / T)) {
        s_n <- s_new
      }
```

```
    }
  }

  return(list(r=s_n, e=estados))
}

# Función a minimizar
f4 <- function(x) {
  return(exp(sin(10 * x)) / (10 * cos(x)))
}

# Ejecutar el algoritmo
Resultado <- resim(f4, alpha=0.1, s0=5, niter=1000, mini=0, maxi=10)
x_min <- Resultado$r
min_f <- exp((sin(10 * x_min)) / (10 * cos(x_min)))

cat("Estado final alcanzado:", x_min, "\n")
```

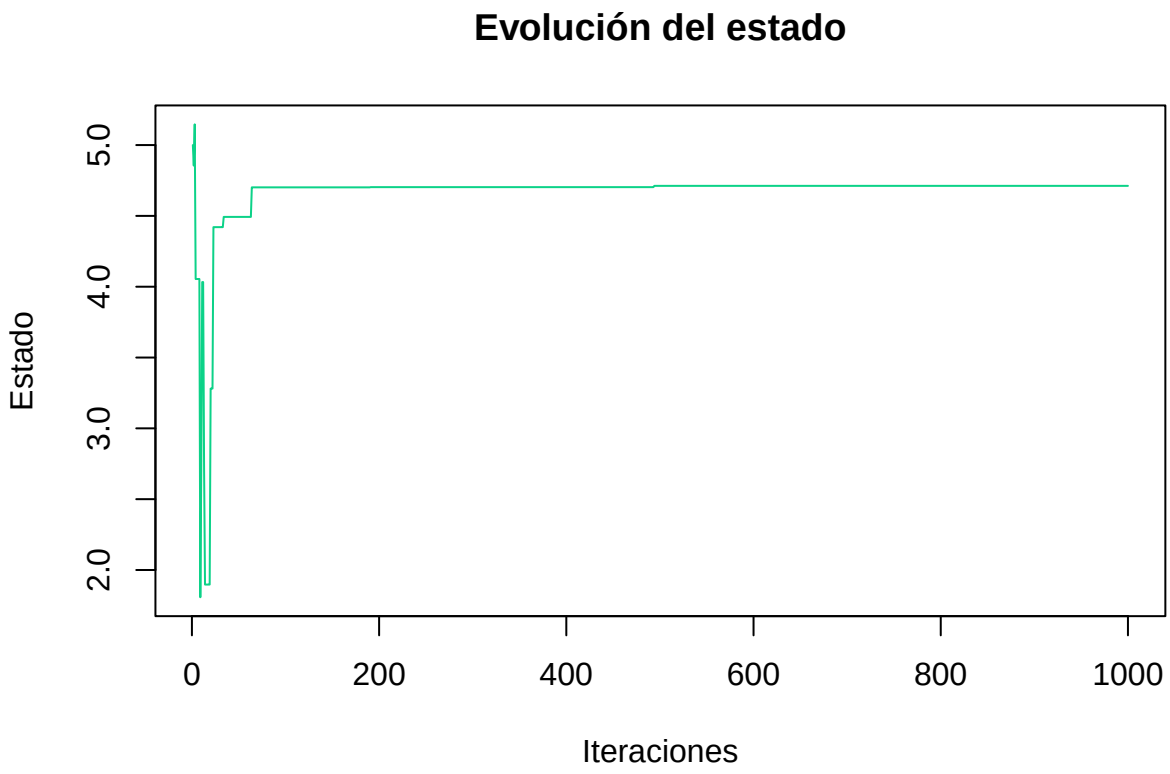
```
## Estado final alcanzado: 4.712077
```

```
cat("Estimacion del min de f(x):", min_f, "\n")
```

```
## Estimacion del min de f(x): 0.36788
```

b)

```
plot(Resultado$e, type='l', main='Evolución del estado', xlab='Iteraciones', ylab='Estado', col='#0CD288
```



Pregunta 5

a)

```
#La muestra de los siniestros observados
mx <- c(4,2,5,6,3,4,7,5,6,4)
```

```
set.seed(54321)
nsim <- 10^4 #numero de simulaciones

# Distribución de verosimilitud
vero <- function(lambda) {prod(dpois(mx, lambda))}

# Distribución priori
alpha <- 3
beta <- 2
priori <- function(lambda) {
  dgamma(lambda, alpha, 1/beta)
}
```

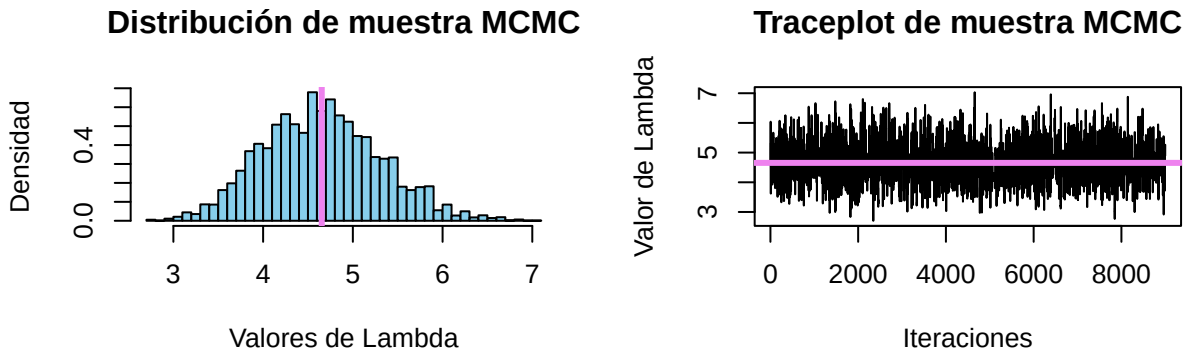
```
# Parámetros del algoritmo
L <- 1000  # Periodo quemado (burn in)
```

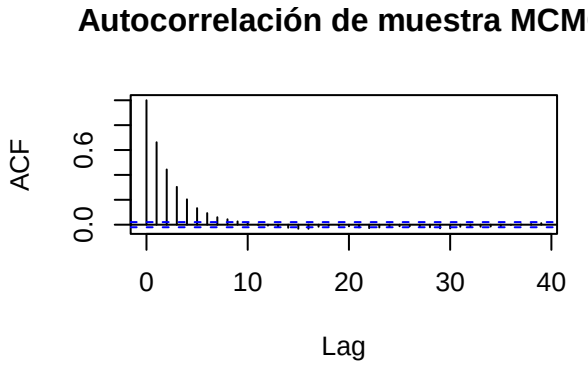
```r
MCMC <- matrix(data = 0, nrow = nsim, ncol = 5)  # Cambiado para tener `nsim` filas
colnames(MCMC) <- c("x", "PIx", "PIy", "Fxy", "Salto")

x <- runif(1, 0, 10)  # Se inicia con un valor aleatorio para lambda

for (i in 1:nsim) {

  y <- rgamma(1,alpha, 1/beta)  # Genera un valor aleatorio de funcion  gamma(3,2)

  # Calcular las distribuciones
  PIx <- vero(x)
  PIy <- vero(y)
  Kxy <- priori(x)
  Kyx <- priori(y)

  Rxy <- (PIy * Kyx) / (PIx * Kxy)

  # Generar un número aleatorio uniforme
  Fxy <- runif(1)

  # Registrar los resultados en la matriz MCMC
  MCMC[i, ] <- c(x, PIx, PIy, Fxy, 0)

  # Verificar la aceptación
  if (Fxy < Rxy) {
    x <- y
    lsalto <- 1
  } else {
    lsalto <- 0
  }

  MCMC[i, 5] <- lsalto
}

# Extraer las muestras después del periodo quemado
mcmc <- MCMC[(L + 1):nsim, "x"]

# Mostrar parte de la muestra obtenida
head(mcmc, 50)
```

```
##  [1] 4.576059 4.576059 4.576059 4.576059 6.040566 6.040566 6.040566 4.605108
##  [9] 3.618269 4.068736 5.320962 5.320962 5.320962 5.320962 5.320962 4.533651
## [17] 4.533651 4.522519 4.522519 4.522519 4.522519 4.522519 4.522519 4.171500
## [25] 4.171500 4.171500 4.171500 4.171500 4.171500 4.171500 4.171500 4.171500
## [33] 4.171500 4.171500 4.679408 4.679408 4.894455 4.039974 5.213855 5.213855
## [41] 5.213855 5.389783 3.923199 3.923199 5.515621 5.515621 5.515621 5.515621
## [49] 4.018215 4.022317
```

b), c) y d)

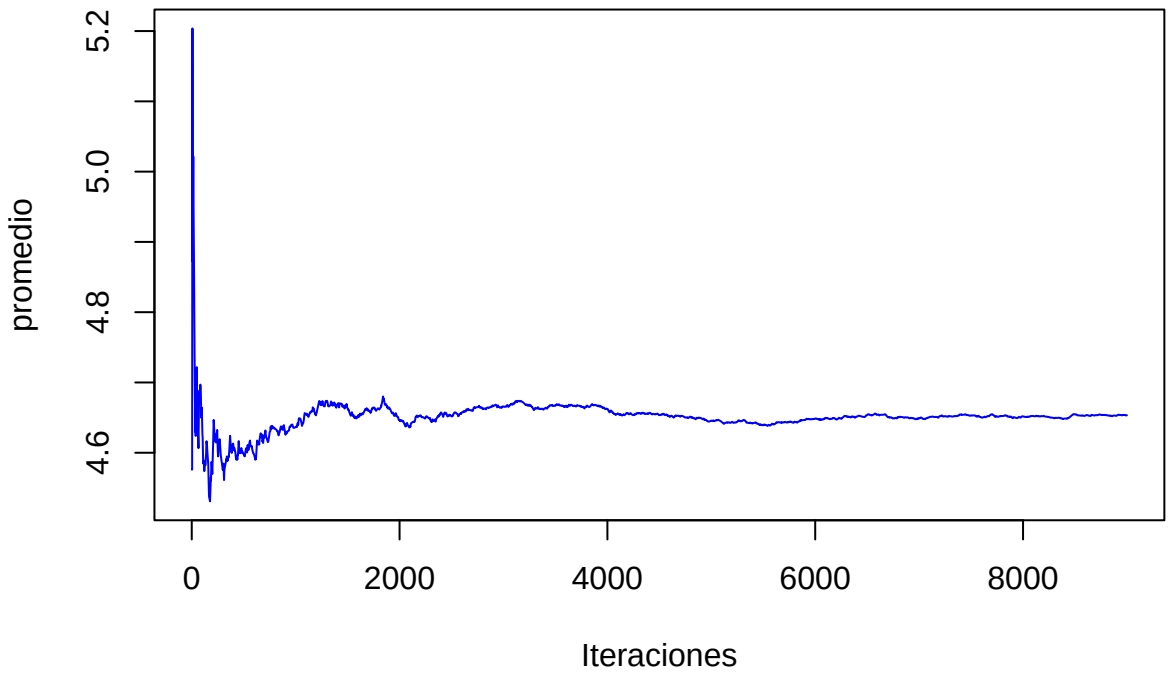
```r
# Cargar la librería ggplot2
library(ggplot2)

# Definir el espacio de gráficos en una ventana de 2x2
par(mfrow = c(2, 2))

# Calcular lambda a partir de la media de la muestra MCMC
lambda <- mean(mcmc)

# Histograma de la muestra MCMC
hist(mcmc, freq = FALSE, main = "Distribución de muestra MCMC",
     xlab = "Valores de Lambda", ylab = "Densidad", breaks = 50,
     col = "skyblue", border = "black")
abline(v = lambda, col = 'violet', lwd = 3)

# Traceplot de la muestra MCMC
plot(mcmc, type = "l", xlab = "Iteraciones", ylab = "Valor de Lambda",
     main = "Traceplot de muestra MCMC")
abline(h = lambda, col = 'violet', lwd = 3)

# Gráfico de autocorrelación
acf(mcmc, main = "Autocorrelación de muestra MCMC")
```





e)

```
#Grafico convergencia de la media
m=nsim-L
acumulado<-cumsum(mcmc)/(1:m)
plot(1:m,acumulado,col="blue",type="l",ylab="promedio",xlab="Iteraciones")
```




```
cat("Estimacion de la media/lambda:", lambda)
```

```
## Estimacion de la media/lambda: 4.653378
```

f)

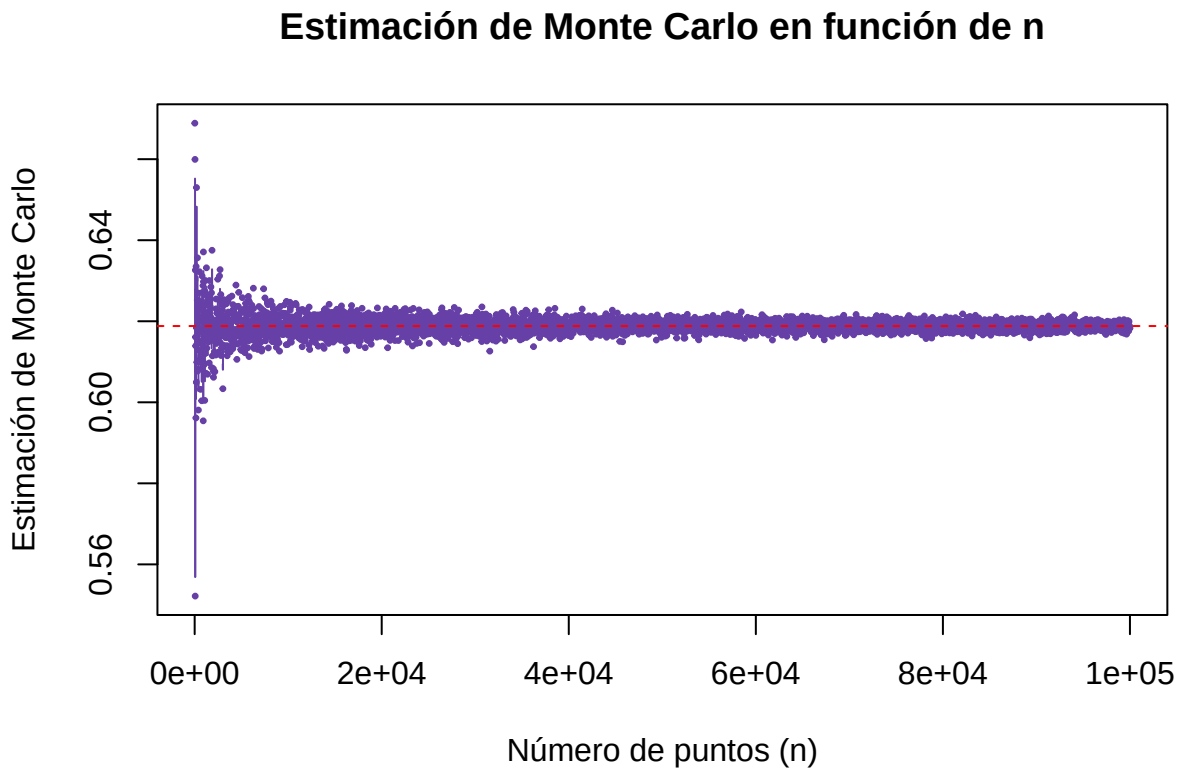
```
cat("Tasa de aceptacion \n",
"NumeroSaltos/TotalIteraciones :" , mean(MCMC[,"Salto"]) ,"\n")
```

```
## Tasa de aceptacion
##  NumeroSaltos/TotalIteraciones : 0.265
```

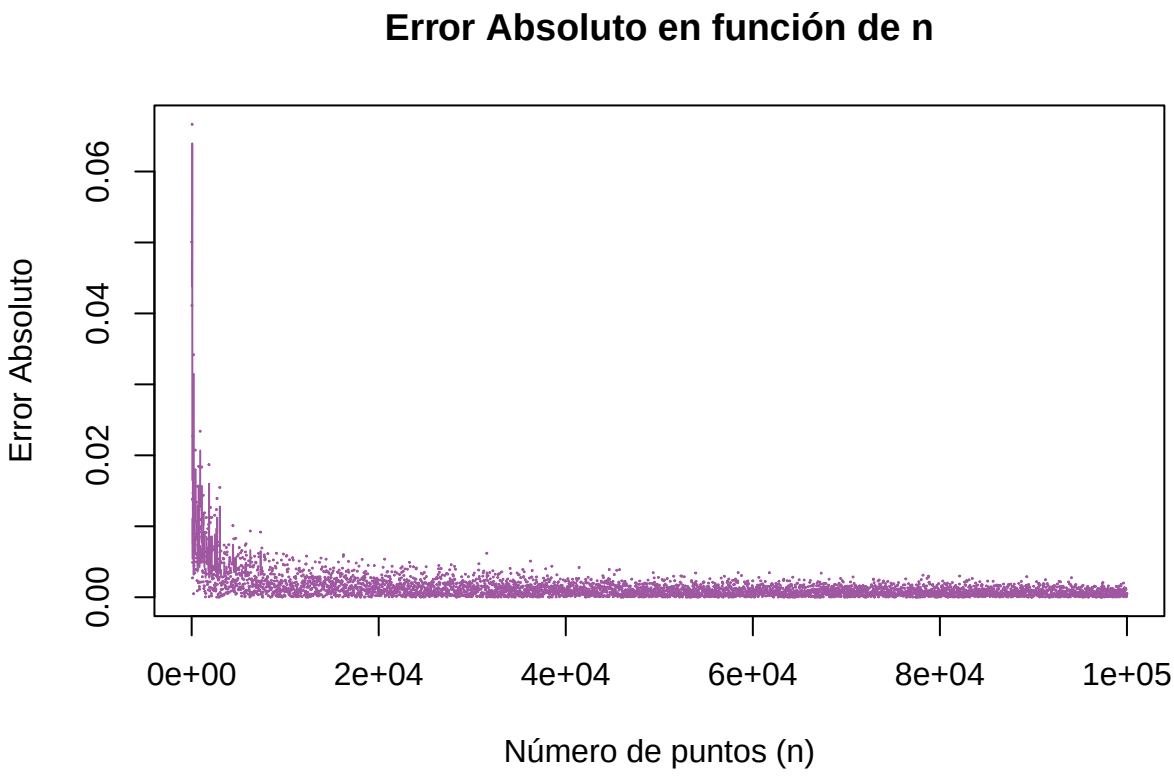
Con n = 1e+05 vemos que se satisface lo deseado.

Estimación de montecarlo de hasta n=100000 y su abs error (tomando cada 20 puntos)

```
plot(seq(20, n, by = 20), monte_carlo_estimaciones, type = "b", col = "#6640A6", pch = 20, cex = 0.5,
     xlab = "Número de puntos (n)", ylab = "Estimación de Monte Carlo",
     main = "Estimación de Monte Carlo en función de n")
abline(h = valor_exacto, col = "red", lty = 2)
```




```
plot(seq(20, n, by = 20), error_absoluto, type = "b", col = "#A057A2", pch = 20, cex = 0.1,
     xlab = "Número de puntos (n)", ylab = "Error Absoluto",
     main = "Error Absoluto en función de n")
```



Pregunta 2

a)

```r
# Definimos la función de densidad de la pérdida L con distribución exponencial
f_L <- function(L, lambda = 1) {
  return(lambda * exp(-lambda * L))
}
```

b)

```r
set.seed(54321)
n <- 10^4

mu <- 3
sigma <- 2

#sacamos las muestras con distribucion normal
muestras_g <- rnorm(n, mean = mu , sd = sigma)
#para constuir las muestras se usa una distribucion normal, sin embargo la funcion de perdida es para l
#exponenciales, por esa razon se restringe a mayores a 0
```

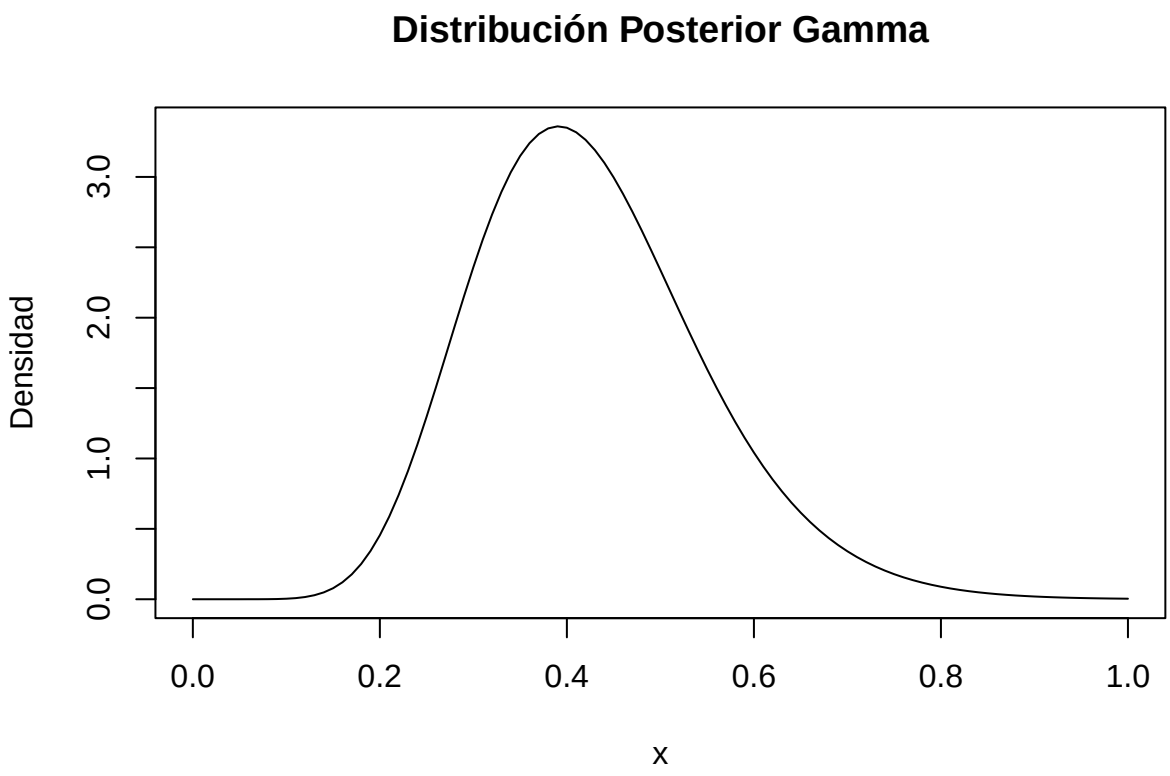
```r
muestras_g <- muestras_g[muestras_g > 0]

#sacamos la densidad de g y f en base a las muestras
g_L_valores <- dnorm(muestras_g, mean = mu, sd = sigma)
f_L_valores <- f_L(muestras_g)

valor_esperado <-mean( muestras_g *  (f_L_valores / g_L_valores))
valor_esperado
```

```
## [1] 1.069952
```

Pregunta 3

Función objetivo

```r
#Dadas las muestras:
muestras <- c(2.72, 1.93, 1.76, 0.49, 6.12, 0.43, 4.01, 1.71, 2.01, 5.96)

#Por teoría bayesiana se optiene que la distribución posteriore es una gamma(12,1+ suma de las muestras
alpha <- 12
beta <- 1 + sum(muestras)

# Función de densidad de la posterior Gamma(alpha, beta) es la función objetivo
f_obj <- Vectorize(function(x) {
  ((beta^alpha) / factorial(alpha - 1)) * x^(alpha - 1) * exp(-beta * x)
})

# Graficar la función de densidad
curve(f_obj, from = 0, to = 1, ylab = "Densidad", main = "Distribución Posterior Gamma")
```




```r
#Calculo del maximo
pto_max <- optimize(f_obj, interval = c(0, 1), maximum = TRUE)
c <- pto_max$objective
print(c)
```

```
## [1] 3.359299
```

```r
# Algoritmo de Aceptación y Rechazo
set.seed(2023)
U <- runif(10000)
x <- runif(10000, 0, 1) #Entre cero y uno porque es donde se encuentra principalmente la densidad
ngen = length(x)

DIB <- f_obj(x)
for (i in 1:10000) {
  while ((U[i] * c) >= DIB[i]) {  # Sustituimos los rechazados
    U[i] <- runif(1)
    x[i] <- runif(1)
    DIB[i] <- f_obj(x[i])
    ngen <- ngen + 1
  }
}
```

```r
# Proporción de rechazos
density_values = f_obj(x)
cat("Número de generaciones =", ngen, "\n")
```

```
## Número de generaciones = 33525
```

```r
cat("Número medio de aceptados =", ngen / 10000, "\n")
```

```
## Número medio de aceptados = 3.3525
```

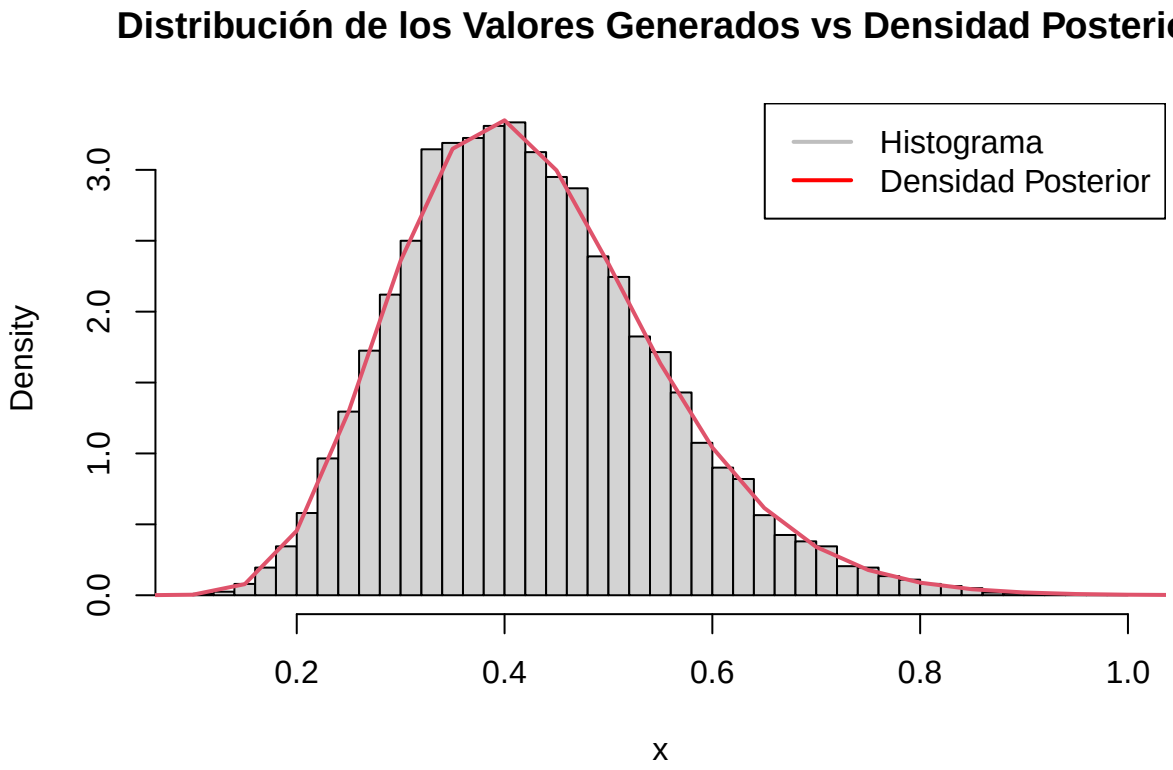
```r
cat("Proporción de rechazos =", 1 - 10000 / ngen, "\n")
```

```
## Proporción de rechazos = 0.7017151
```

```r
# Estimación de lambda
lambda_est = mean(x)
cat("Valor estimado de lambda:", lambda_est, "\n")
```

```
## Valor estimado de lambda: 0.4252807
```

```r
# Graficar el histograma de los valores generados y la densidad teórica
hist(x, breaks = "FD", freq = FALSE, main = "Distribución de los Valores Generados vs Densidad Posterio
curve(f_obj, from = 0, to = 5, col = 2, lwd = 2, add = TRUE)
legend("topright", legend = c("Histograma", "Densidad Posterior"), col = c("gray", "red"), lty = 1, lwd
```



```r
q <- quantile(x, c(0.005, 0.9955))
q
```

```
##      0.5%    99.55%
## 0.1748033 0.8064021
```

Si lambda fuera 0,5 no se rechazaria la hipotesis porque el valor estaria dentro del intervalo de confianza, por lo que no se tiene la suficiente informacion para poder descartar la hipotesis y mas bien es un valor de lambda que puede ser admisible ya que cae dentro del intervalo de confianza del 99%

Pregunta 4

a)

```r
set.seed(73)
resim <- function(f, alpha, s0, niter, mini=0, maxi=10) {
  s_n <- s0
  estados <- rep(0, niter)

  for (k in 1:niter) {
    estados[k] <- s_n
    T <- (1 - alpha)^k # Se reduce el T según el número de iteraciones
    s_new <- rnorm(1, s_n, 1)

    s_new <- max(mini, min(maxi, s_new))

    dif <- f(s_new) - f(s_n)
    if (dif < 0) {
      s_n <- s_new
    } else {
      random <- runif(1)
      if (random < exp(-dif / T)) {
        s_n <- s_new
      }
    }
  }

  return(list(r=s_n, e=estados))
}

# Función a minimizar
f4 <- function(x) {
  return(exp(sin(10 * x)) / (10 * cos(x)))
}

# Ejecutar el algoritmo
Resultado <- resim(f4, alpha=0.1, s0=5, niter=1000, mini=0, maxi=10)
x_min <- Resultado$r
min_f <- exp((sin(10 * x_min)) / (10 * cos(x_min)))

cat("Estado final alcanzado:", x_min, "\n")
```

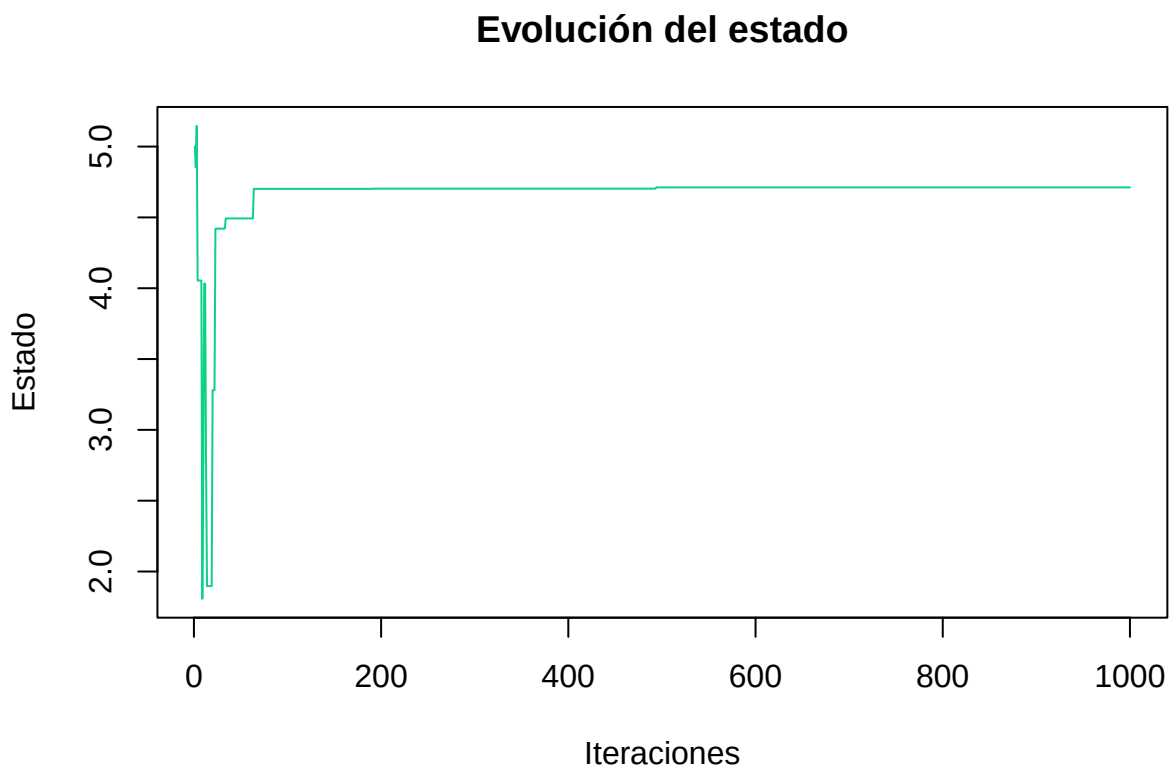
```
## Estado final alcanzado: 4.712077
```

```
cat("Estimacion del min de f(x):", min_f, "\n")
```

```
## Estimacion del min de f(x): 0.36788
```

b)

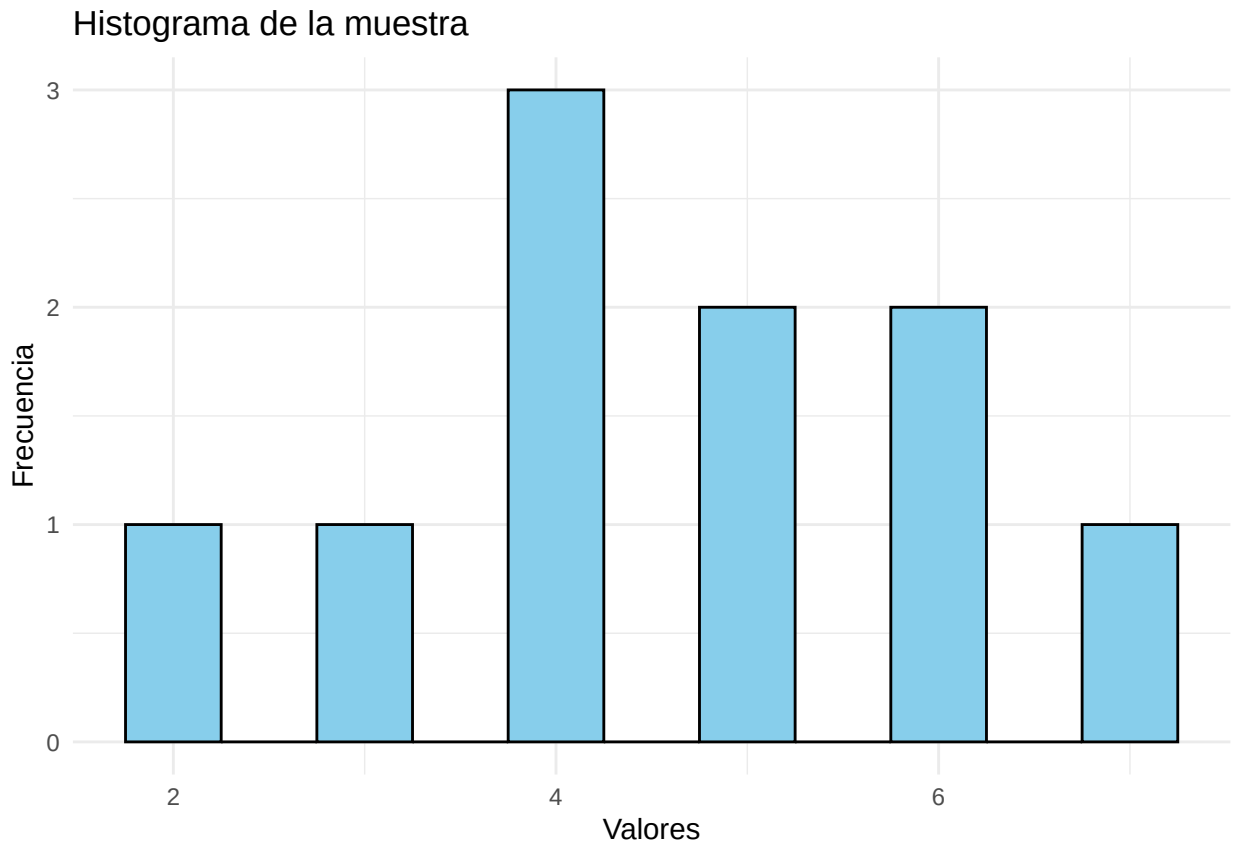
```
plot(Resultado$e, type='l', main='Evolución del estado', xlab='Iteraciones', ylab='Estado', col='#0CD288
```



Pregunta 5

```
#La muestra de los siniestros observados
mx <- c(4,2,5,6,3,4,7,5,6,4)

# Crear el histograma

mxdf <- data.frame(mx)
ggplot(mxdf, aes(x = mx)) +
  geom_histogram(binwidth = 0.5, fill = "skyblue", color = "black") +
```

```r
  theme_minimal() +  # Estética preferida
  labs(title = "Histograma de la muestra", x = "Valores", y = "Frecuencia")
```



Generación de las propuestas con una distribución arbitraria

```r
set.seed(54321)
nsim <- 10^4 #numero de simulaciones

# Distribución de verosimilitud
vero <- function(lambda) {prod(dpois(mx, lambda))}

# Distribución priori
alpha <- 3
beta <- 2
priori <- function(lambda) {
  dgamma(lambda, alpha, 1/beta)
}
```

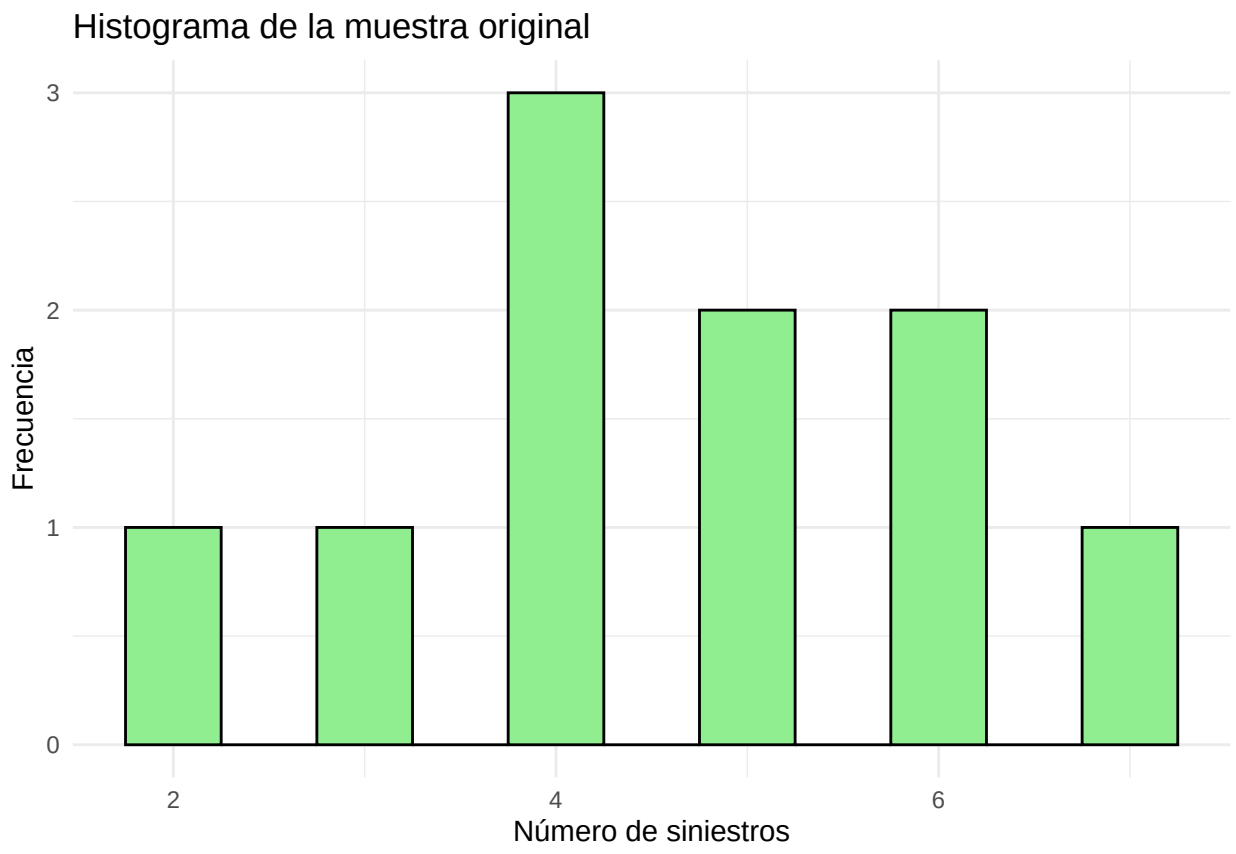
```r
# Parámetros del algoritmo
L <- 1000  # Periodo quemado (burn in)
MCMC <- matrix(data = 0, nrow = nsim, ncol = 5)  # Cambiado para tener `nsim` filas
colnames(MCMC) <- c("x", "PIx", "PIy", "Fxy", "Salto")
```

```r
x <- runif(1, 0, 10)  # Se inicia con un valor aleatorio para lambda

for (i in 1:nsim) {

  y <- rgamma(1,alpha, 1/beta)  # Genera un valor aleatorio de funcion  gamma(3,2)

  # Calcular las distribuciones
  PIx <- vero(x)
  PIy <- vero(y)
  Kxy <- priori(x)
  Kyx <- priori(y)

  Rxy <- (PIy * Kyx) / (PIx * Kxy)

  # Generar un número aleatorio uniforme
  Fxy <- runif(1)

  # Registrar los resultados en la matriz MCMC
  MCMC[i, ] <- c(x, PIx, PIy, Fxy, 0)

  # Verificar la aceptación
  if (Fxy < Rxy) {
    x <- y
    lsalto <- 1
  } else {
    lsalto <- 0
  }

  MCMC[i, 5] <- lsalto
}

# Extraer las muestras después del periodo quemado
mcmc <- MCMC[(L + 1):nsim, "x"]

# Mostrar parte de la muestra obtenida
head(mcmc, 50)
```

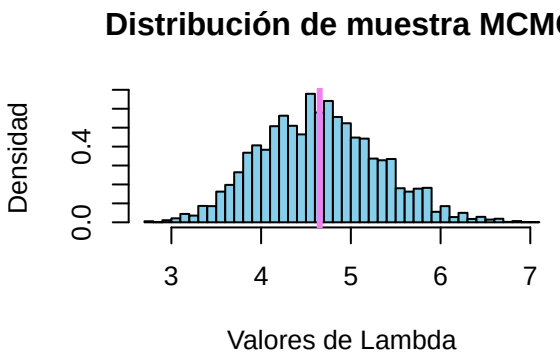
```
##  [1] 4.576059 4.576059 4.576059 4.576059 6.040566 6.040566 6.040566 4.605108
##  [9] 3.618269 4.068736 5.320962 5.320962 5.320962 5.320962 5.320962 4.533651
## [17] 4.533651 4.522519 4.522519 4.522519 4.522519 4.522519 4.522519 4.171500
## [25] 4.171500 4.171500 4.171500 4.171500 4.171500 4.171500 4.171500 4.171500
## [33] 4.171500 4.171500 4.679408 4.679408 4.894455 4.039974 5.213855 5.213855
## [41] 5.213855 5.389783 3.923199 3.923199 5.515621 5.515621 5.515621 5.515621
## [49] 4.018215 4.022317
```

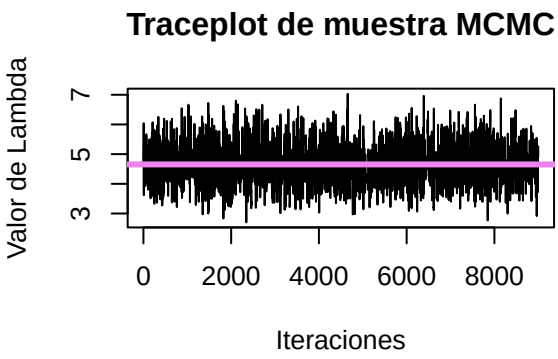
```r
# Cargar la librería ggplot2
library(ggplot2)

# Definir el espacio de gráficos en una ventana de 2x2
par(mfrow = c(2, 2))

# Calcular lambda a partir de la media de la muestra MCMC
lambda <- mean(mcmc)
```

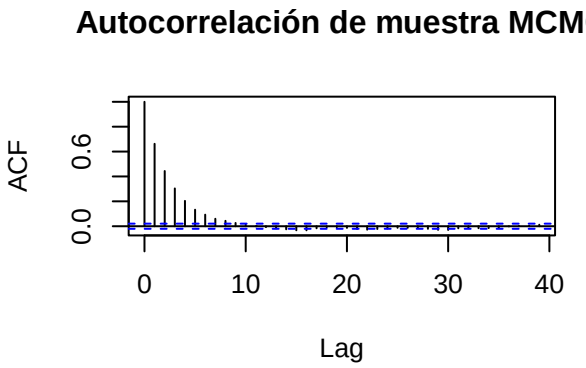
```r
# Histograma de la muestra original (`mx`) usando ggplot2
mxdf <- data.frame(mx)
p <- ggplot(mxdf, aes(x = mx)) +
  geom_histogram(binwidth = 0.5, fill = "lightgreen", color = "black") +
  theme_minimal() +  # Estética preferida
  labs(title = "Histograma de la muestra original", x = "Número de siniestros", y = "Frecuencia")
print(p)  # Mostrar el histograma con ggplot2
```



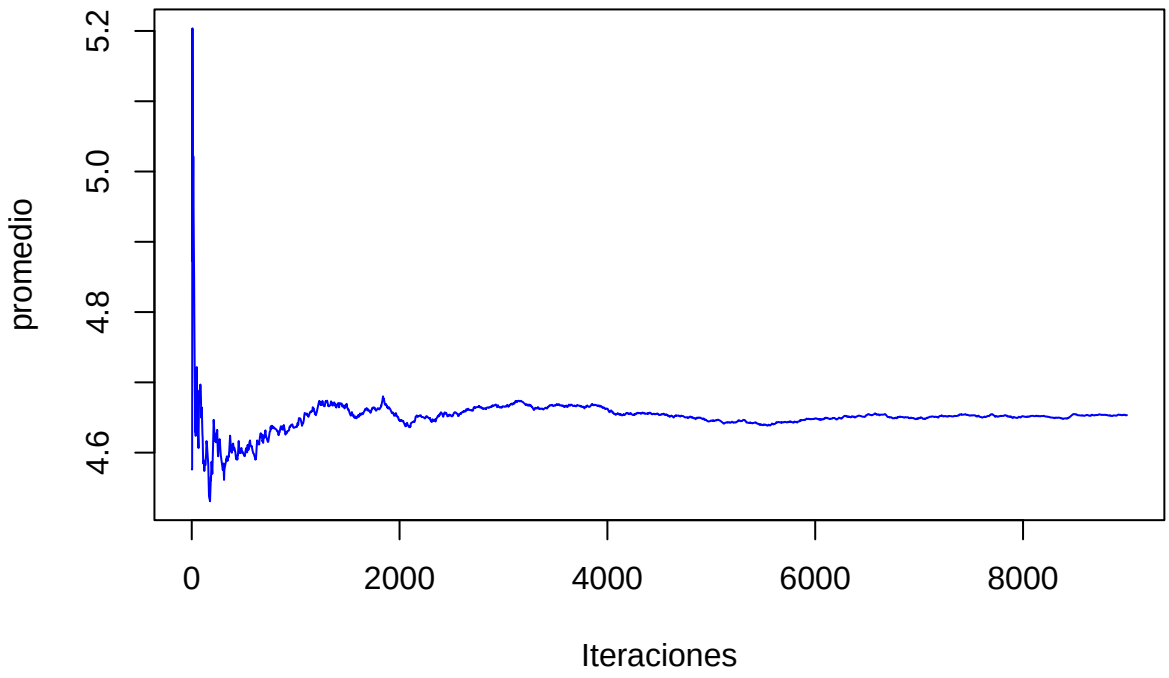

```r
# Histograma de la muestra MCMC
hist(mcmc, freq = FALSE, main = "Distribución de muestra MCMC",
     xlab = "Valores de Lambda", ylab = "Densidad", breaks = 50,
     col = "skyblue", border = "black")
abline(v = lambda, col = 'violet', lwd = 3)

# Traceplot de la muestra MCMC
plot(mcmc, type = "l", xlab = "Iteraciones", ylab = "Valor de Lambda",
     main = "Traceplot de muestra MCMC")
abline(h = lambda, col = 'violet', lwd = 3)

# Gráfico de autocorrelación
acf(mcmc, main = "Autocorrelación de muestra MCMC")
```








```
#Grafico convergencia de la media
m=nsim-L
acumulado<-cumsum(mcmc)/(1:m)
plot(1:m,acumulado,col="blue",type="l",ylab="promedio",xlab="Iteraciones")
```




```
cat("Estimacion de la media/lambda:", lambda)
```

```
## Estimacion de la media/lambda: 4.653378
```

```
cat("Tasa de aceptacion \n",
"NumeroSaltos/TotalIteraciones :" , mean(MCMC[,"Salto"]) ,"\n")
```

```
## Tasa de aceptacion
##  NumeroSaltos/TotalIteraciones : 0.265
```