Tarea_II_estadistica

Alejandro Brenes (C21319), Santiago Fernández (C22943), Eyeri Méndez (C24765)

2024-10-20

1)

```r
set.seed(12345)

# Definimos la función a integrar
g <- Vectorize(function(x)
  exp(-x ^ 2) / (1 + x ^ 2))

# Calculamos el valor exacto de la integral
int_exacta <- integrate(g, 0, 1)$value

# Tamaño de la muestra
n <- 10 ^ 6

# Generamos una m.a.s de U[0, 1]
U <- runif(n)

# Calculamos los valores de la función en los valores generados
Y <- g(U)

# Calculamos la aproximación de la integral mediante Monte Carlo
int_aprox <- mean(Y)

# Calculamos el error absoluto entre el valor aproximado y el valor exacto de la integral
error_1 <- abs(int_aprox - int_exacta)

error_1
```

```
## [1] 1.695227e-06
```

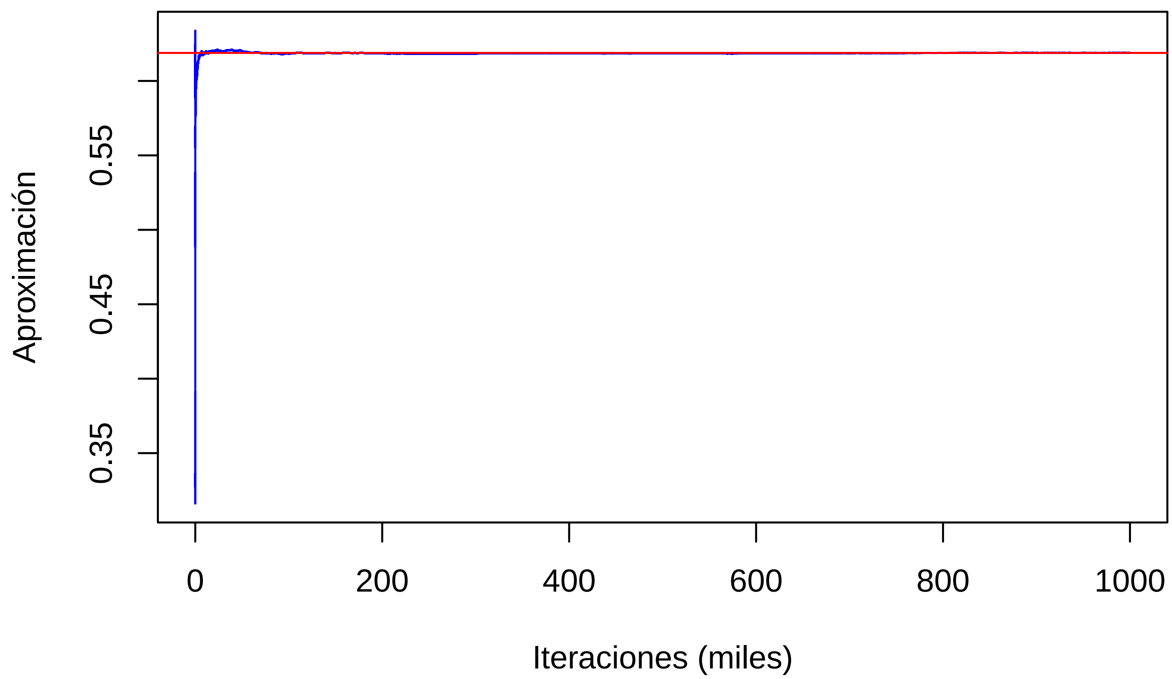
En el siguiente gráfico podemos apreciar la convergencia de la función $\frac{1}{n}\sum g(u_i)$.

```r
prod_acum <- cumsum(Y) / (1:n)

plot(
  (1:n) / 1000,
  prod_acum,
  col = "blue",
  type = "l",
```

```r
  ylab = "Aproximación",
  xlab = "Iteraciones (miles)"
)

abline(h = integrate(g, 0, 1)$value,
       col = "red",
       lwd = 1)
```



2)

a)

Dado que la pérdida $L \sim Exp(\lambda)$, con $\lambda = 1$, entonces su función de densidad está dada por $f_L(L) = e^{-L}$, y además $E[L] = \int_0^\infty L \cdot f_L(L)\, dL = \frac{1}{\lambda} = 1$.

```r
lambda <- 1

f_L <- function(L){

  return(lambda * exp(-lambda * L))

}
```

b)

```r
set.seed(54321)

# Número de muestras
n <- 10 ^ 4

# Generamos las muestras de la distribución auxiliar g(L) ~ N(3, 4)
g_L <- rnorm(n, mean = 3, sd = 2)

# Filtramos para asegurarnos de que las muestras sean positivas
g_L <- g_L[g_L > 0]

n <- length(g_L)

# Definimos una función que calcula el valor correspondiente para cada L
f <- Vectorize(function(L)
  (L * f_L(L)) / (dnorm(L, mean = 3, sd = 2)))

# Calculamos los valores de la función en los valores de la muestra
Y <- f(g_L)

# Calculamos la estimación del valor esperado
estimacion <- mean(Y)

# Calculamos el valor esperado exacto
valor_esperado <- 1 / lambda

# Calculamos el error absoluto entre la estimación y el valor esperado
error_2 <- abs(estimacion - valor_esperado)

# Guardamos lo obtenido en una matriz
A <- matrix(c(estimacion, valor_esperado, error_2), ncol = 3)
colnames(A) <- c("Estimación", "Valor real", "Error absoluto")

A
```

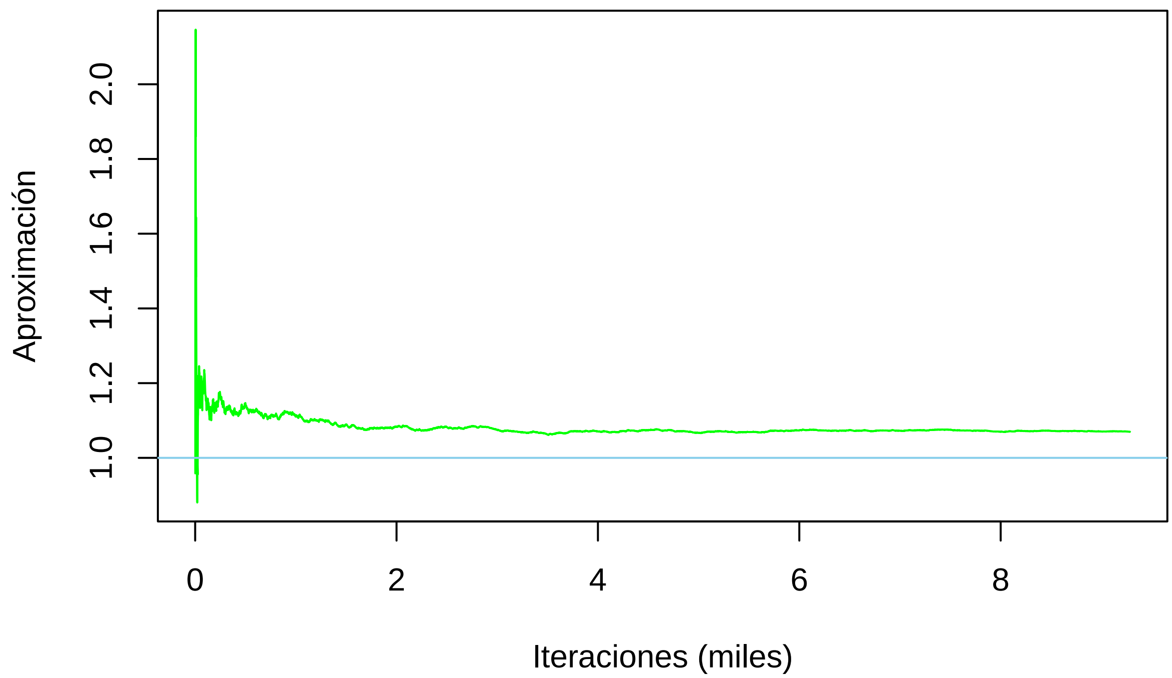
```
##      Estimación Valor real Error absoluto
## [1,]   1.069952          1     0.06995193
```

Podemos ver la convergencia en el siguiente gráfico.

```r
prod_acum <- cumsum(Y) / (1:n)

plot(
  (1:n) / 1000,
  prod_acum,
  col = "green",
  type = "l",
  ylab = "Aproximación",
  xlab = "Iteraciones (miles)"
)
```

```r
abline(h = valor_esperado,
       col = "skyblue",
       lwd = 1)
```



3)

Inicialmente, agregamos la muestra que nos da el problema.

```r
muestra <- c(2.72, 1.93, 1.76, 0.49, 6.12, 0.43, 4.01, 1.71, 2.01, 5.96)
```

a)

Debido a que tenemos una muestra, se debe calcular su función de verosimilitud.

```r
lik <- Vectorize(function(lambda)
  prod(dexp(muestra, rate = lambda)))
```

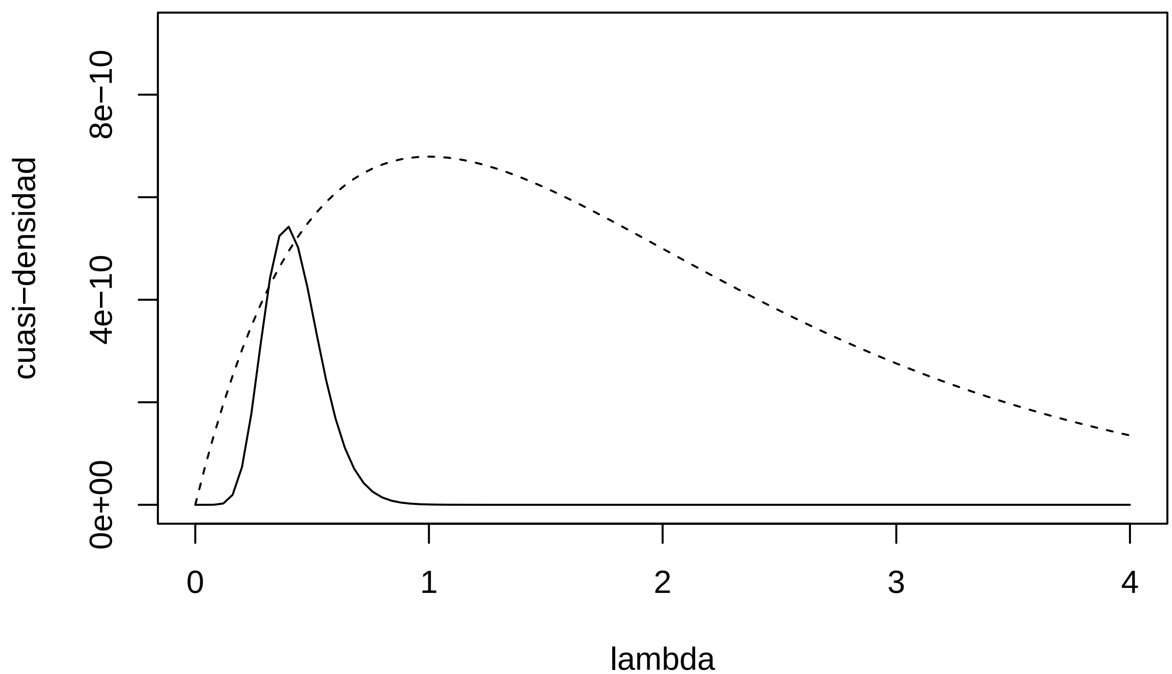
Para la constante c óptima, según la muestra, se maximiza numéricamente la verosimilitud.

```r
# Se maximiza la función
emv <- optimize(lik , int = range(muestra), maximum = TRUE)
```

```
# Se extrae la constante c
c <- emv$objective
```

Para asegurarnos que el algoritmo será eficaz, se tiene el siguiente gráfico, en el cual se observa que la función punteada recubre casi completamente a la otra (no la recubre completamente, probablemente, debido a que se toma una muestra pequeña).

```
f.cuasi <- function(lambda)
  lik(lambda) * dgamma(lambda, shape = 2, scale = 1)
curve(
  c * dgamma(x, shape = 2, scale = 1),
  xlim = c(0, 4),
  ylim = c(0, c / 2),
  lty = 2,
  xlab = "lambda",
  ylab = "cuasi-densidad"
)
curve(f.cuasi, add = TRUE)
```



Aplicando el algoritmo de aceptación-rechazo.

```
# Cantidad de simulaciones
nsim <- 10 ^ 4

# Muestras correspondientes
```

```r
U <- runif(nsim)
rc <- rgamma(nsim, shape = 2, scale = 1)

# Número de generaciones inicial
ngen <- length(rc)

# Verosimilitud de la muestra de gamma
Ver <- lik(rc)

# Ciclo para realizar las simulaciones requieridas
for (i in 1:nsim) {
  # Ciclo para el algoritmo de aceptación-rechazo
  while ((U[i] * c) > (Ver[i])) {
    U[i] <- runif(1)
    rc[i] <- rgamma(1, shape = 2, scale = 1)
    Ver[i] <- lik(rc[i])
    ngen <- ngen + 1
  }
}
```

El estimador de máxima verosimilitud (estimación bayesiana) de una distribución exponencial es el promedio, por lo cual, con los resultados del algoritmo anterior obtenemos el valor estimado de λ.

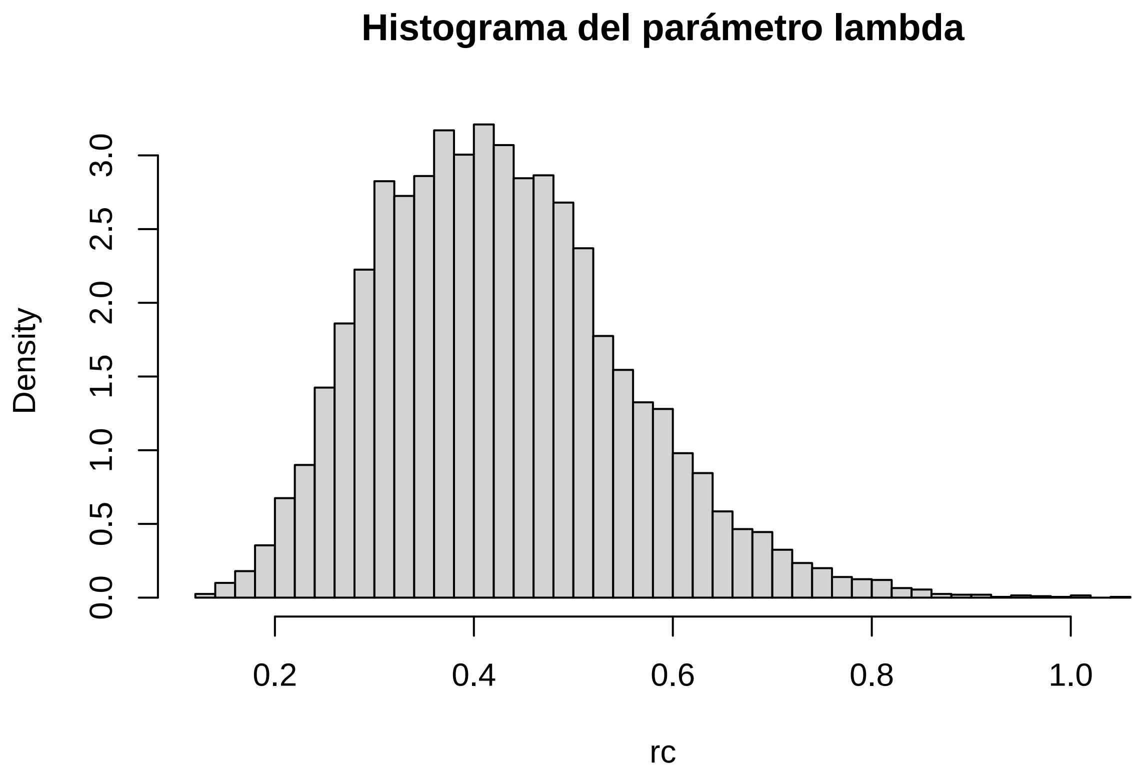
```r
cat("Lambda estimado = ", mean(rc))
```

```
## Lambda estimado =  0.4274934
```

b)

El histograma es el siguiente.

```r
hist(rc,
     freq = FALSE,
     main = "Histograma del parámetro lambda",
     breaks = "FD")
```



c)

```r
{
  cat("Número de generaciones = ", ngen)
  cat("\nNúmero medio de generaciones = ", ngen / nsim)
  cat("\nProporción de rechazos = ", 1 - (nsim / ngen), "\n")
}
```

```
## Número de generaciones =  117411
## Número medio de generaciones =  11.7411
## Proporción de rechazos =  0.9148291
```

d)

El intervalo de credibilidad puede construirse de múltiples maneras para que acumule el %99 de masa, en este caso se toma un intervalo "centrado".

```r
quantile(rc, c(0.005, 0.995))
```

```
##      0.5%     99.5%
## 0.1766837 0.8177066
```

e)

En el caso del intervalo de credibilidad anterior, el valor 0.5 se encuentra dentro del intervalo, es decir, se aceptaría la hipótesis de que $\lambda = 0.5$.

4)

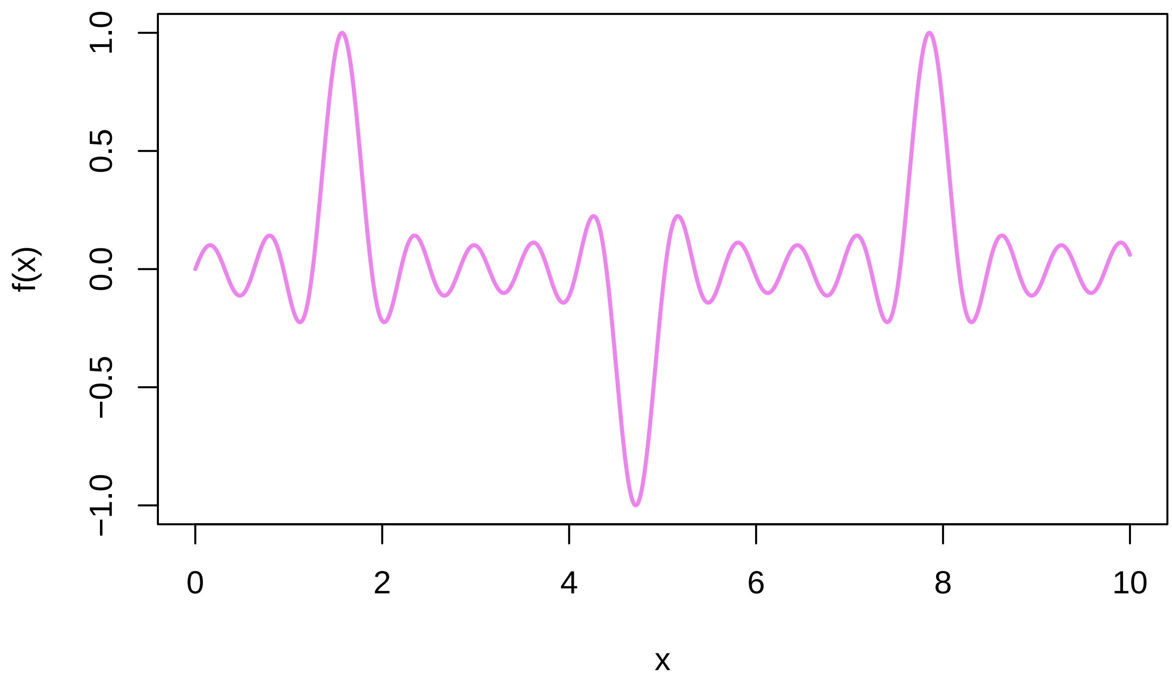
En primer lugar, hace falta definir la función correspondiente.

```
f <- function(x) {
  (sin(10 * x)) / (10 * cos(x))
}
```

a)

Seguidamente se tiene el gráfico de la función, en el cual se puede ver que la función tiene múltiples mínimos en el intervalo requerido.

```
curve(
  f,
  col = "violet",
  lwd = 2,
  from = 0,
  to = 10,
  n = 1000,
  ylab = "f(x)"
)
```



El algoritmo de recalentamiento simulado es el siguiente.

```
resim <- function(f,
                  alpha = 0.5,
                  s0 = 0,
                  niter,
                  mini = -Inf,
                  maxi = Inf) {
  # Valor inicial
  s_n <- s0

  # Vector para guardar los estados
  estados <- rep(0, niter)

  # Contador de iteraciones
  iter_count <- 0

  # Ciclo para encontrar el mínimo
  for (k in 1:niter) {
    # Estado de esta iteración
    estados[k] <- s_n

    # Se reduce el T según el número de iteraciones
    t <- (1 - alpha) ^ k

    # Nuevo estado para comparar con el actual
```

```r
    s_new <- rnorm(1, s_n, 1)

    # Se acorta el intervalo según los estados
    if (s_new < mini) {
      s_new <- mini
    }
    if (s_new > maxi) {
      s_new <- maxi
    }

    # Se define la diferencia de los estados, evaluados en sus funciones
    dif <- f(s_new) - f(s_n)

    # Se actualiza el estado según la condición
    if (dif < 0) {
      s_n <- s_new
    }
    else {
      # Número aleatorio para comparar
      random <- runif(1, 0, 1)

      # Se compara el aleatorio con p
      if (random < exp(-dif / t)) {
        s_n <- s_new
      }
    }

    # Aumenta el contador de iteraciones
    iter_count <- iter_count + 1
  }
  return(list(minimo = s_n, estados = estados))
}
```

Se estima el mínimo usando el algoritmo anterior.

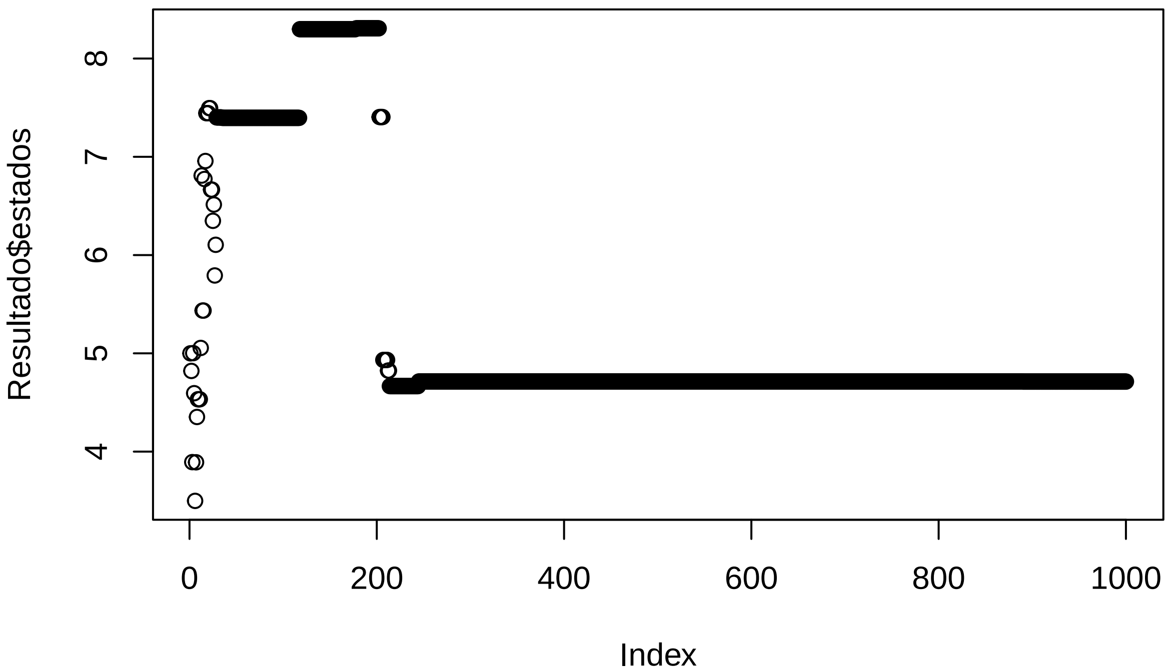
```r
set.seed(54321)
Resultado <- resim(f, 0.1, 5, 1000, 0, 10)
cat("El mínimo de la función f en [0, 10] es: ", Resultado$minimo)
```

```
## El mínimo de la función f en [0, 10] es:  4.712711
```

b)

Los estados se pueden observar en el siguiente gráfico.

```r
plot(Resultado$estados)
```



5)

a)

Como primer paso, almacenamos los datos de la muestra.

```
muestra <- c(4, 2, 5, 6, 3, 4, 7, 5, 6, 4)
```

Si cada $x_i, i \in \{1, ..10\}$ representa la cantidad de siniestros por periodo, se sabe que $x_i \sim Poisson(\lambda)$. Es decir, su verosimilitud viene dada por:

$$\prod_{i=1}^{n} \mathbb{P}(x_i|\lambda) = \prod_{i=1}^{n} \frac{e^{-\lambda}\lambda^{x_i}}{(x_i)!}$$

Establecemos así la verosimilitud.

```
verosimilitud <- Vectorize(function(lambda) {
  prod(dpois(muestra, lambda))
})
```

La "propuesta" para λ es que $\lambda \sim \gamma(3, 2)$. Así, procedemos con el algoritmo de Metropolis-Hastings:

```r
simulaciones <- 10 ^ 4
quemados <- 1000 #se usan 1000 dado que es el estandar

# Matriz de resultados
MCMC <- matrix(data = 0, nrow = simulaciones, ncol = 5)
colnames(MCMC) <- c("x", "PIx", "PIy", "Fxy", "Salto")

# Valor inicial de x
x <- runif(1, 0, 10)

for (i in 1:simulaciones) {
  y = rgamma(1, 3, 2)

  PIx <- verosimilitud(x)
  PIy <- verosimilitud(y)

  Kxy = dgamma(x, 3, 2)
  Kyx = dgamma(y, 3, 2)

  Rxy = (PIy * Kyx) / (PIx * Kxy)

  Fxy <- runif(1)

  MCMC[i, ] <- c(x, PIx, PIy, Fxy, 0)

  if (Fxy < Rxy) {
    x <- y
    lsalto <- 1
  } else {
    lsalto <- 0
  }
  MCMC[i, 5] <- lsalto
}

mcmc <- MCMC[(quemados + 1):simulaciones, "x"]

head(mcmc, 25)
```

```
##  [1] 3.852037 3.852037 3.852037 3.852037 3.852037 3.852037 3.852037 3.852037
##  [9] 3.852037 3.852037 3.852037 3.852037 3.852037 3.852037 3.852037 3.852037
## [17] 3.852037 3.852037 3.852037 3.596872 3.596872 3.596872 3.596872 3.596872
## [25] 3.596872
```

Se calcula también la media de la muestra.

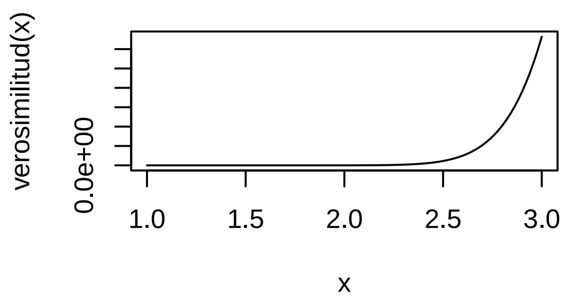
```r
media <- mean(mcmc)
print(media)
```

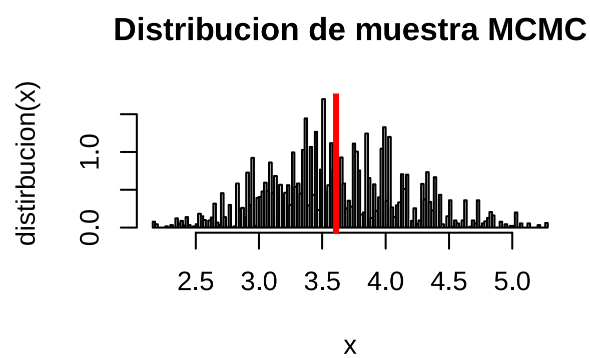
```
## [1] 3.608792
```

b)

Ahora se realiza la distribución de la muestra MCMC del algoritmo.

```r
par(mfrow = c(2, 2))
media = mean(mcmc)
curve(verosimilitud(x),
      from = 1,
      to = 3,
      type = "l")
abline(v = media, col = 'red', lwd = 3)
hist(
  mcmc,
  freq = FALSE,
  main = "Distribucion de muestra MCMC",
  xlab = "x",
  ylab = "distirbucion(x)",
  breaks = 200
)
abline(v = media, col = 'red', lwd = 3)
```



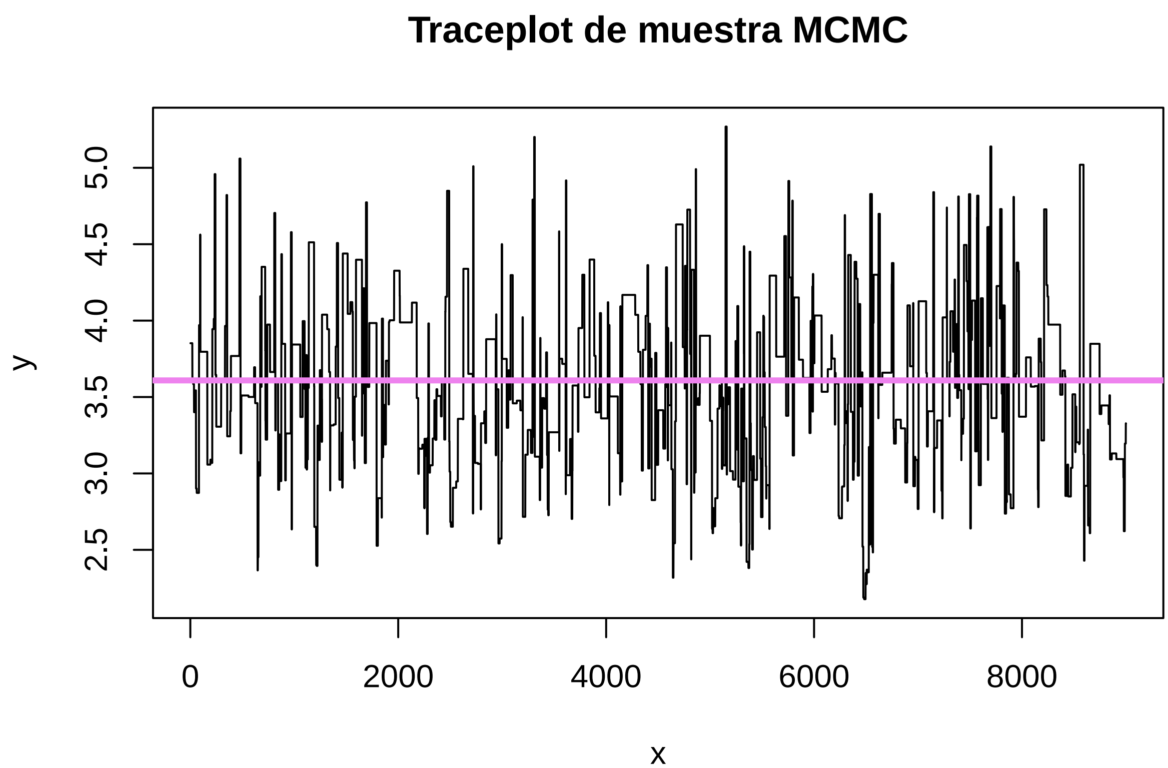


c)

Realizamos un gráfico Traceplot.

```r
plot(
  mcmc,
  type = "l",
```

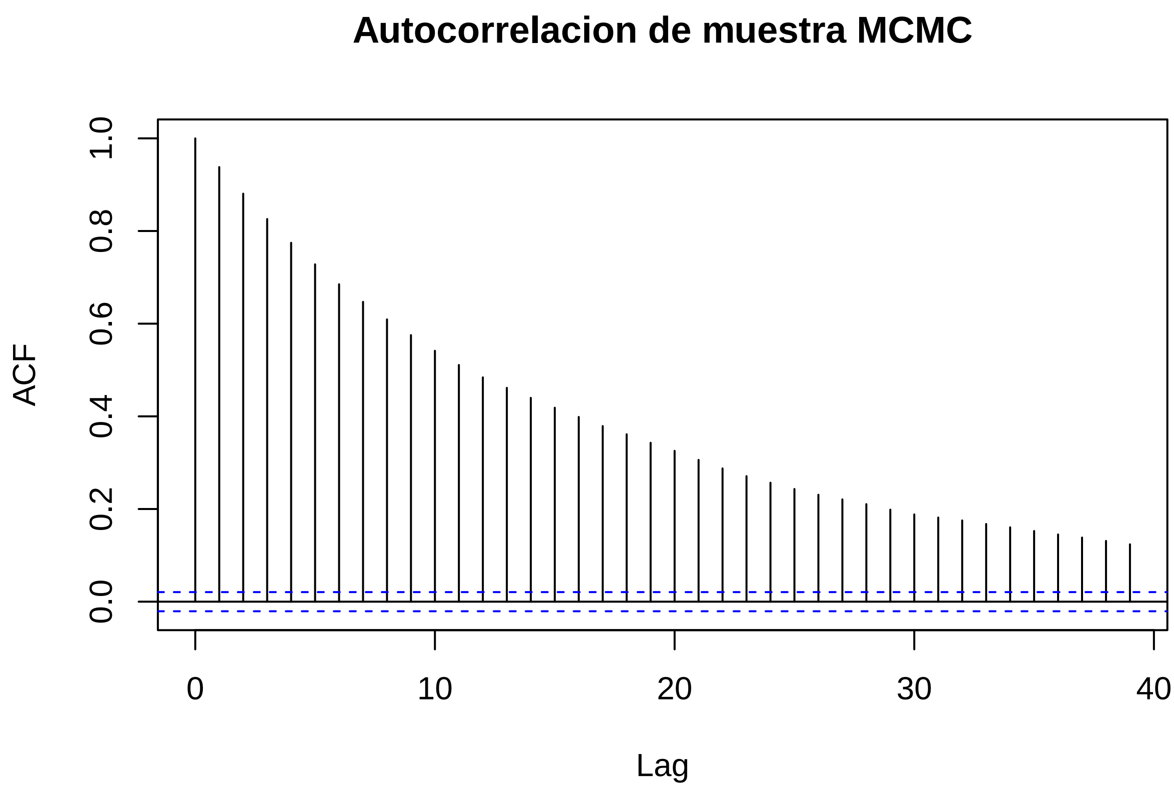
```
  xlab = "x",
  ylab = "y",
  main = "Traceplot de muestra MCMC"
)
abline(h = media,
       col = 'violet',
       lwd = 3)
```



d)

Continuamos con el gráfico de autocorrelación.

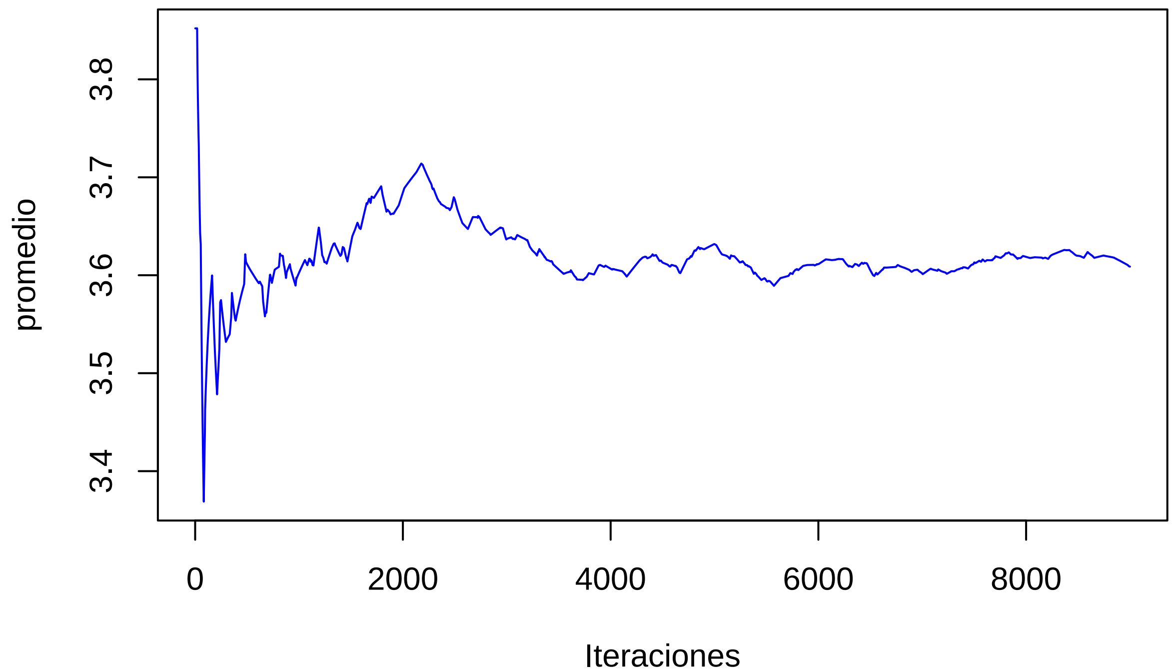
```
acf(mcmc, main = "Autocorrelacion de muestra MCMC")
```



e)

Ahora el gráfico de la convergencia (promedios ergódicos) de la media de la muestra MCMC del algoritmo.

```r
par(mfrow = c(1, 1))
m = simulaciones - quemados
acumulado <- cumsum(mcmc) / (1:m) # media acumulada
plot(
  1:m,
  acumulado,
  col = "blue",
  type = "l",
  ylab = "promedio",
  xlab = "Iteraciones"
)
```



f)

Finalmente, se calcula la tasa de aceptación del algoritmo.

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

```
## Tasa de aceptacion
##  NumeroSaltos/TotalIteraciones : 0.06
```