Tarea 2 - CA0307: Estadística Actuarial II

Anthony Mauricio Jiménez Navarro | C24067 Henri Gerard Gabert Hidalgo | B93096
Juan Pablo Morgan Sandí | C15319

2024-10-22

Contents

Librerias

```
#set.seed(2024)
```

Ejercicio 1

Primero se crea la función a integrar, la cual sabemos que es

$$\int_0^1 \frac{e^{-x^2}}{1+x^2} dx$$

```
f <- function(x) {
  exp(-x^2) / (1 + x^2)
}
```

Una vez hecho lo anterior, programos el algoritmo de Montecarlo.

```
set.seed(875)
# Método de Montecarlo para aproximar la integral
montecarlo_integration <- function(N) {
  # Generamos N muestras aleatorias entre 0 y 1
```

```r
  x <- runif(N, 0, 1)

  # Evaluamos la función en los puntos muestreados
  fx <- f(x)

  # Estimar la integral como el promedio de f(x)
  integral_estimate <- mean(fx)

  return(integral_estimate)
}

N <- 100000  # número de muestras

montecarlo_result <- montecarlo_integration(N)

cat("Aproximación de la integral por Montecarlo:", montecarlo_result, "\n")
```

```
## Aproximación de la integral por Montecarlo: 0.6194744
```

Ahora, usando integrate.

```r
integral_exacta <- integrate(f, 0, 1)
cat("Aproximación de la integral por Integrate:", integral_exacta$value, "\n")
```

```
## Aproximación de la integral por Integrate: 0.618822
```

```r
cat("El error absoluto de Integrate:", integral_exacta$abs.error, "\n")
```

```
## El error absoluto de Integrate: 6.870304e-15
```

Ahora, la diferencia entre el resultado de Montecarlo y el de Integrate es:

```r
abs(montecarlo_result - integral_exacta$value)
```

```
## [1] 0.000652426
```

Ejercicio 2

Primero, se crea la función f_L, la cual es

$$f_L(L) = \lambda e^{-\lambda L}$$

La cual, sabiendo que $\lambda = 1$, es $f_L(L) = e^{-L}$

```r
f_L <- function(L) {
  return(exp(-L))
}
```

Por el enunciado sabemos también que $g(L) \sim N(3, 4)$

```
g_L <- function(L) {
  return(dnorm(L, mean = 3, sd = 2))
}
```

Una vez creadas las funciones anteriores, se procede a crear el algoritmo de muestreo por importancia.

```
# Establecemos la semilla
set.seed(54321)

n <- 10^4  # número de muestras
L_samples <- numeric(0)  # Inicializamos el vector de muestras

# Rechazamos valores negativos
while (length(L_samples) < n) {
  samples <- rnorm(n, mean = 3, sd = 2)  # Generamos n muestras de N(3, 2^2)
  L_samples <- c(L_samples, samples[samples > 0])
  # Solo conservamos las positivas por la restricción
  L_samples <- L_samples[1:n]  # Aseguramos que solo tengamos n muestras
}

pesos <- f_L(L_samples) / g_L(L_samples)
# Filtrar valores infinitos o NA en los pesos de importancia
pesos_validos <- is.finite(pesos) & !is.na(pesos)
L_samples <- L_samples[pesos_validos]
pesos <- pesos[pesos_validos]

# Calcular la pérdida esperada
perdidas_esperadas <- mean(L_samples * pesos)

cat("La estimación del valor esperado de la pérdida usando muestreo por importancia es:",
    perdidas_esperadas, "\n")
```

```
## La estimación del valor esperado de la pérdida usando muestreo por importancia es: 1.069952
```

Ejercicio 3

Se definen los tiempos entre accidentes y las funciones exponencial y gamma, ademas se establecen los parametros de la funcion gamma, todo esto segun lo establecido en el enunciado.

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

# Definición de las funciones de densidad
objetivo <- function(lambda, x) {
  lambda * exp(-lambda * x)
}

previa <- function(lambda) {
  dgamma(lambda, shape = 2, rate = 1)
}
```

Algoritmo de aceptacion-rechazo

```r
n <- 10000
valoresLambda <- numeric(n)

for (i in 1:n) {
  repeat {
    # Generar candidato de la distribución prior
    lambda <- rgamma(1, shape = 2, rate = 1)

    # Generar variable uniforme
    u <- runif(1)

    # Verificar aceptación
    if (u < min(objetivo(lambda, tiempos) / previa(lambda))) {
      valoresLambda[i] <- lambda
      break
    }
  }
}
```

Resultados

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

```
## Número de generaciones =  10000
```

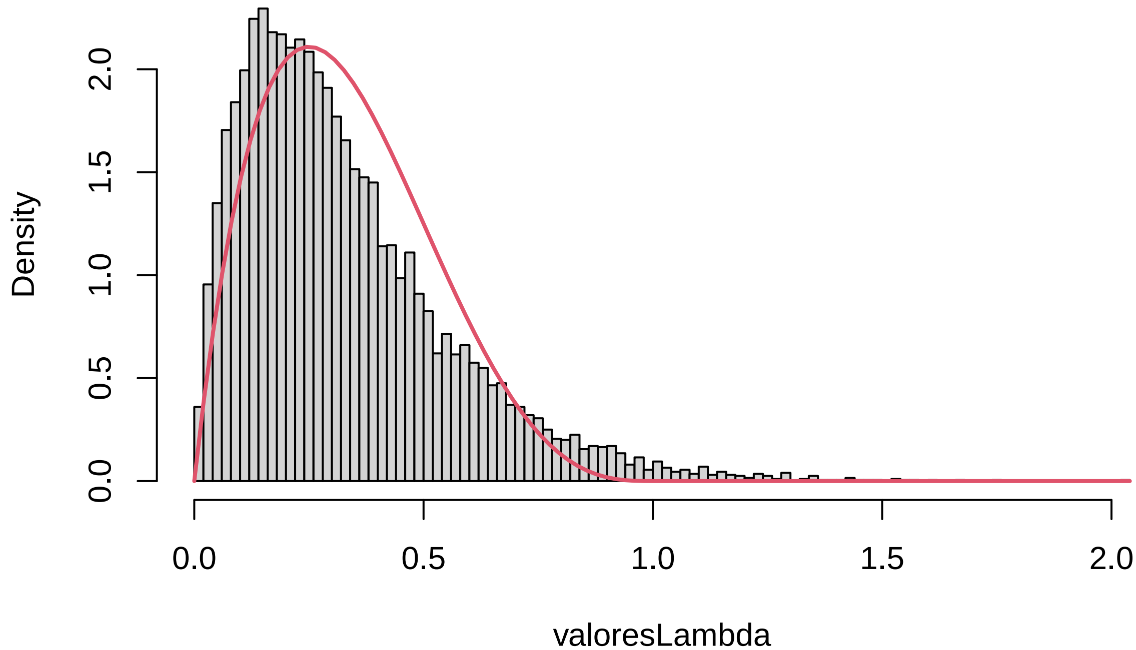
```r
cat("Número medio de aceptados = ", ngen / 10^4, "\n")
```

```
## Número medio de aceptados =  1
```

```r
cat("Proporción de rechazos = ", 1 - 10^4 / ngen, "\n")
```

```
## Proporción de rechazos =  0
```

```r
lambda_est <- 1 / mean(tiempos)

hist(valoresLambda, breaks = "FD", freq = FALSE, main = "")
curve(dbeta(x, 2, 4), col = 2, lwd = 2, add = TRUE)
```



Intervalo de credibilidad al 99%

```
cred_interval <- quantile(valoresLambda, probs = c(0.005, 0.995))
cat("Intervalo de credibilidad al 99%: [", cred_interval[1], ", ", cred_interval[2], "]\n")
```

```
## Intervalo de credibilidad al 99%: [ 0.01597998 ,  1.194903 ]
```

Aceptacion o rechazo de lambda = 5

```
lambda_hip <- 0.5
if(lambda_hip >= cred_interval[1] && lambda_hip <= cred_interval[2]) {
  cat("No se rechaza la hipótesis lambda = 0.5, está dentro del intervalo de credibilidad.\n")
} else {
  cat("Se rechaza la hipótesis lambda = 0.5, está fuera del intervalo de credibilidad.\n")
}
```

```
## No se rechaza la hipótesis lambda = 0.5, está dentro del intervalo de credibilidad.
```

Ejercicio 4

Funcion

$$f(x) = exp(\frac{sen(10x)}{10cos(x)})$$

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

Funcion de recalentamiento simulado

```r
resim <- function(f, alpha = 0.5, s0 = 5, niter = 1000, mini = 0, maxi = 10) {
  s_n <- s0
  estados <- rep(0, niter)
  iter_count <- 0
  for (k in 1:niter) {
    estados[k] <- s_n
    T <- (1 - alpha)^k  # Enfriamiento
    s_new <- rnorm(1, s_n, 1)

    # Asegurarse de que la nueva solución esté dentro de los límites
    if (s_new < mini) { s_new <- mini }
    if (s_new > maxi) { s_new <- maxi }

    dif <- f(s_new) - f(s_n)
    if (dif < 0) {
      s_n <- s_new
    } else {
      random <- runif(1, 0, 1)
      if (random < exp(-dif / T)) {
        s_n <- s_new
      }
    }
    iter_count <- iter_count + 1
  }
  return(list(r = s_n, e = estados))
}
```

Aplicacion

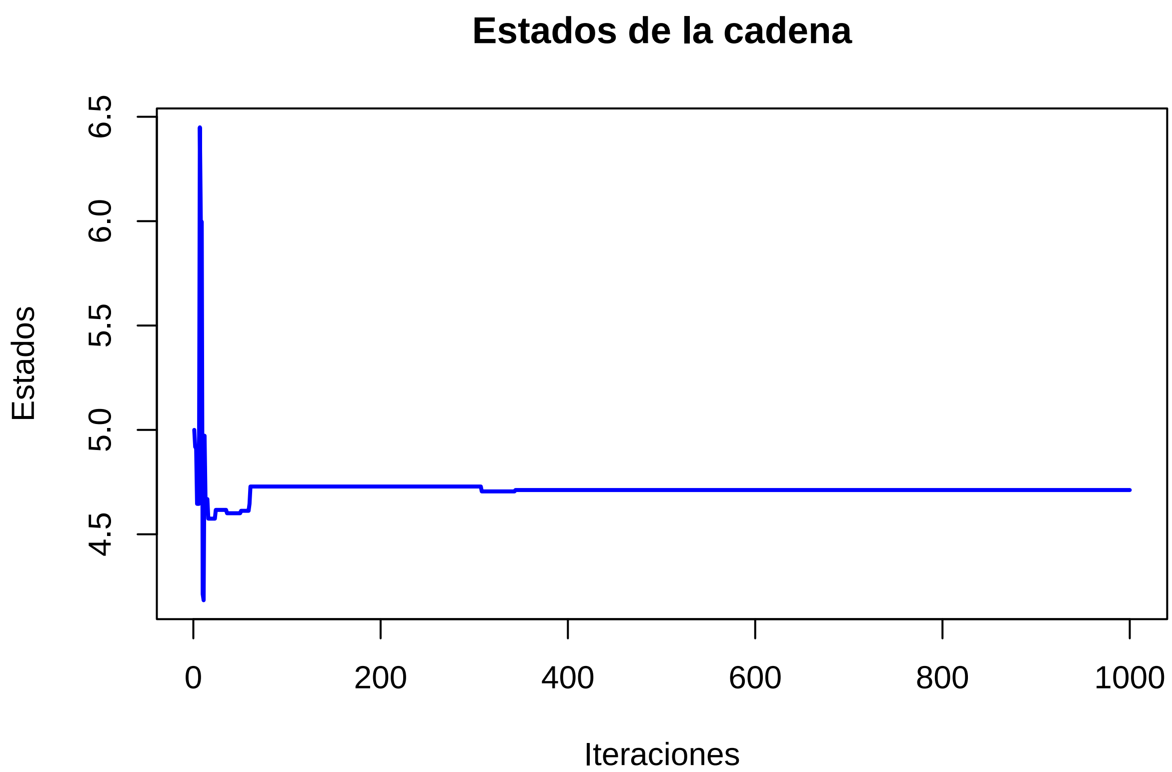
```r
Resultado <- resim(f, 0.1, 5, 1000, 0, 10)
```

Resultados

```r
Resultado$r  # Minimo global
```

```
## [1] 4.711989
```

```r
plot(Resultado$e, type = "l", col = "blue", lwd = 2,
     ylab = "Estados", xlab = "Iteraciones", main = "Estados de la cadena")
```



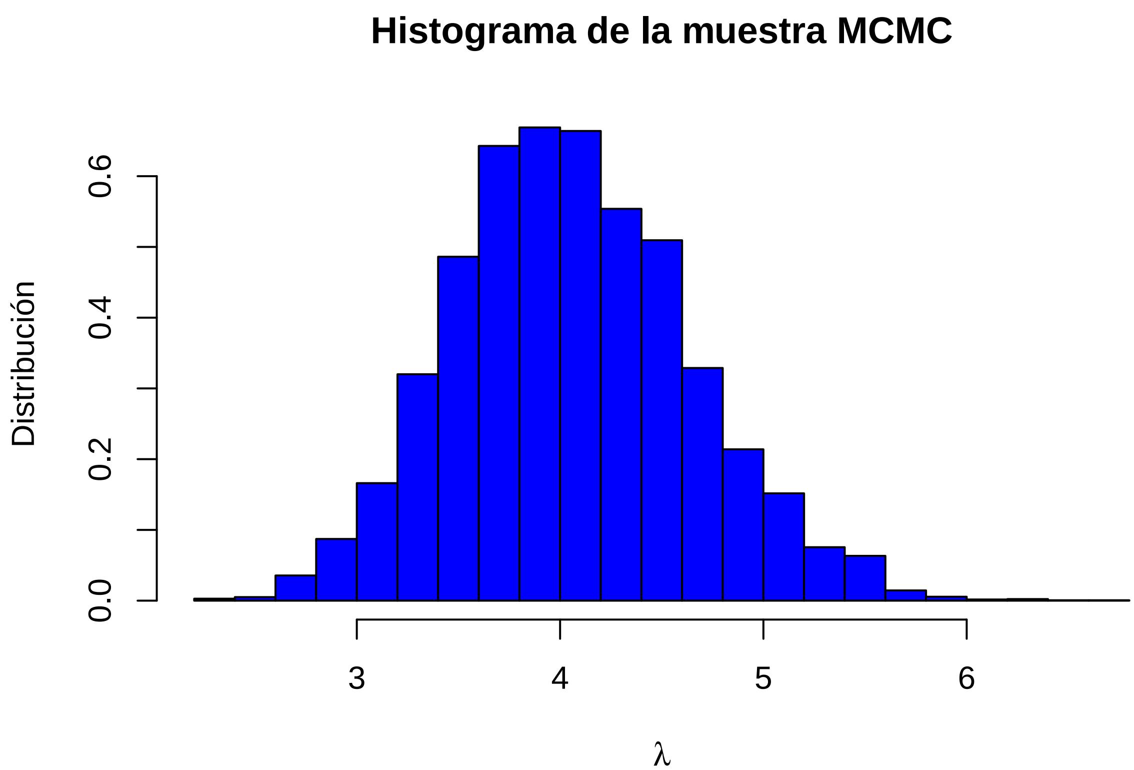
#Ejercicio 5 A)

```
set.seed(77)
# Muestra
data <- c(4, 2, 5, 6, 3, 4, 7, 5, 6, 4)
#Numero de iteraciones
n<-10000
#Periodo quemado
L<-1000
#Lambda artibtrario de inicio
lambda_inicio<-runif(1,0,10)

# Función de verosimilitud de Poisson
poisson <- function(lambda, data) {
  prod(dpois(data, lambda))
}

# Función de distribución gamma a priori
gamma_prior <- function(lambda, alpha , beta ) {
  dgamma(lambda, shape = alpha, rate = beta)
}

# Algoritmo de Metropolis-Hastings
metropolis_hastings <- function(data, N = n, lambda_inicial = lambda_inicio,
                                alpha = 3, beta = 2) {


  Intentos_lambda <- numeric(N)
  Intentos_lambda[1] <- lambda_inicial
  lambda_actual <- lambda_inicial
```

```
  Saltos <- 0

  for (i in 2:N) {
    propuesta <- rnorm(1, mean = lambda_actual, sd = 1)  # Propuesta

    if (propuesta > 0) {  # Para evitar valores negativos de lambda, pues es una poisson

      #Usando el factor de bayes, la propuesta/la actual
      Aceptacion <- (poisson(propuesta, data) * gamma_prior(propuesta, alpha, beta)) /
                    (poisson(lambda_actual, data) * gamma_prior(lambda_actual,
                                                                alpha, beta))

      #Criterio de Aceptacion o Rechazo
      if (runif(1) < Aceptacion) {
        lambda_actual <- propuesta
        Saltos <- Saltos + 1
      }
    }
    Intentos_lambda[i] <- lambda_actual
  }
  return(list(Intentos_lambda = Intentos_lambda[(L+1):N], Saltos = Saltos))
}

# Ejecutar el algoritmo con n = 10000
Muestras_MCMC <- metropolis_hastings(data)$Intentos_lambda
```

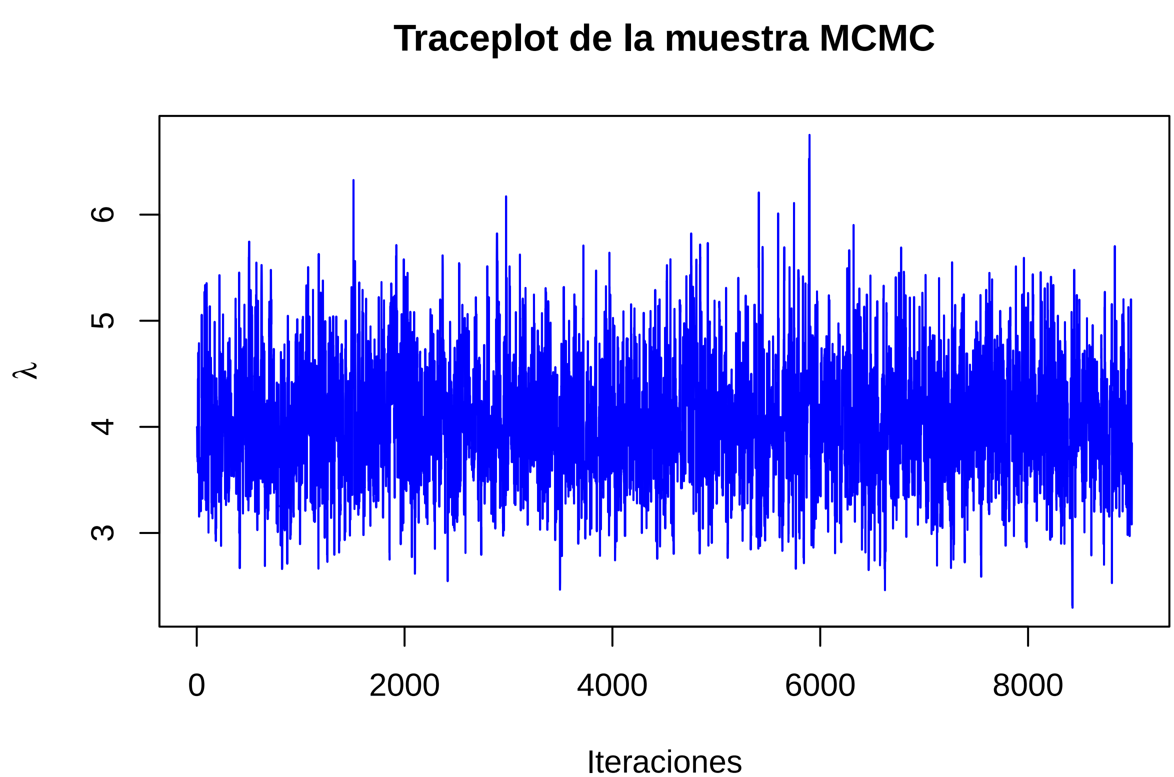
B)

```
hist(Muestras_MCMC, breaks = 30, prob = TRUE, main = "Histograma de la muestra MCMC",
     xlab = expression(lambda),ylab = "Distribución", col = "blue")
```



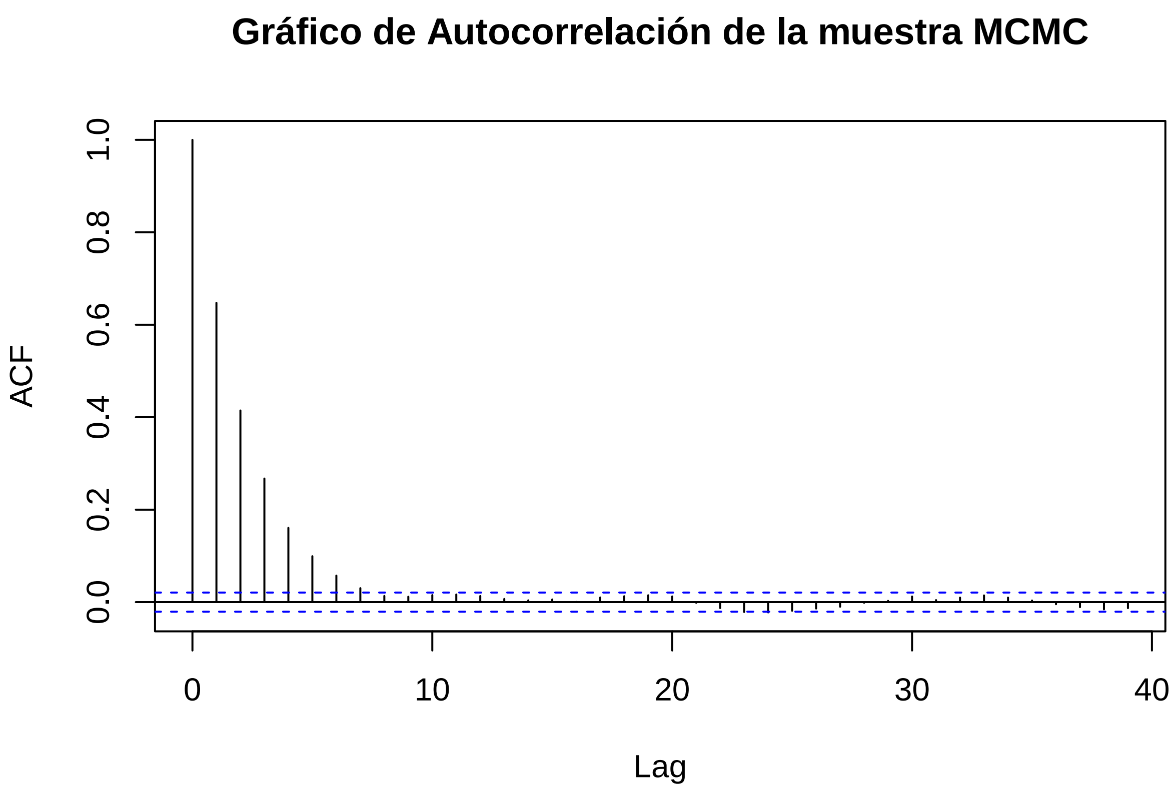
C)

```r
plot(Muestras_MCMC, type = "l", main = "Traceplot de la muestra MCMC",
     xlab = "Iteraciones", ylab = expression(lambda), col = "blue")
```



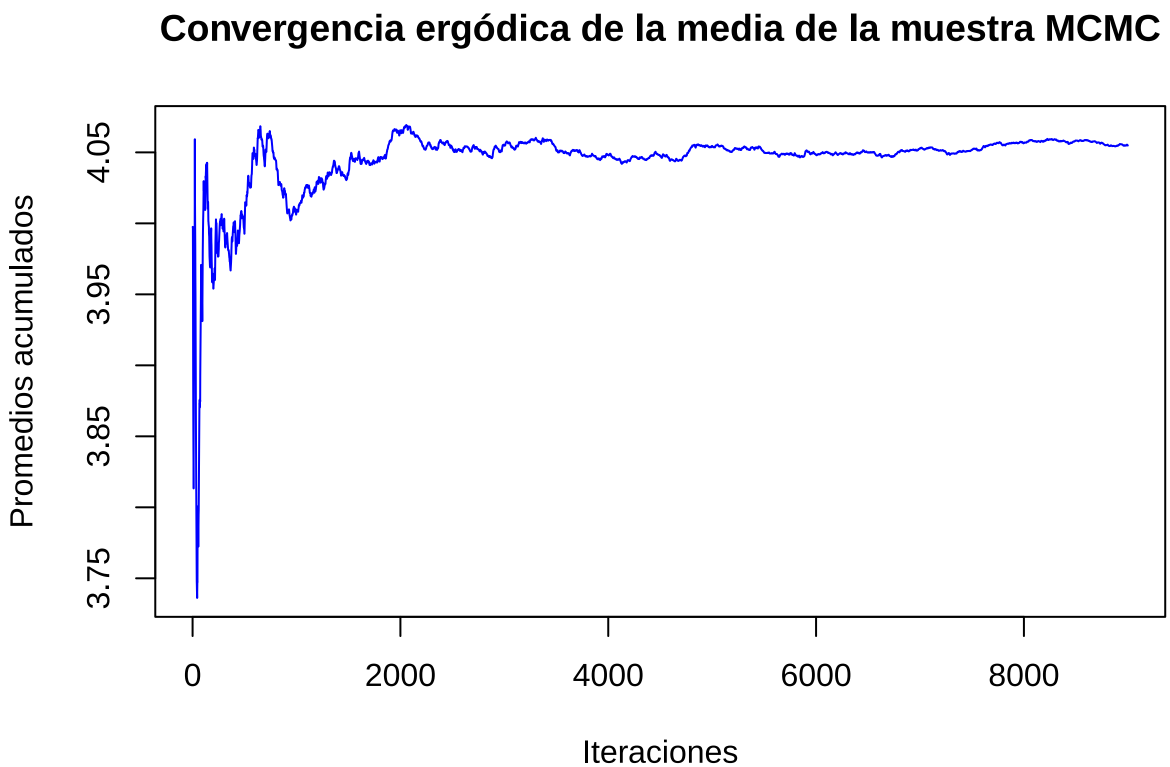
D)

```r
acf(Muestras_MCMC, main = "Gráfico de Autocorrelación de la muestra MCMC")
```



E)

```r
#Normalmente es la suma acumulada/(Iteracion-periodo_quemado)

mean_muestras <- cumsum(Muestras_MCMC) / seq_along(Muestras_MCMC)
plot(mean_muestras, type = "l", main = "Convergencia ergódica de la media de la muestra MCMC",
     xlab = "Iteraciones", ylab = "Promedios acumulados", col = "blue")
```



F)

```
mean_lambda <- mean(Muestras_MCMC)
cat("Estimación de lambda:", mean_lambda, "\n")
```

```
## Estimación de lambda: 4.054824
```

```
cat("Tasa de aceptación \n","NumeroSaltos/TotalIteraciones:" ,
    (metropolis_hastings(data)$Saltos)/(n-L) ,"\n")
```

```
## Tasa de aceptación
##  NumeroSaltos/TotalIteraciones: 0.6001111
```