**UNIVERSIDAD NACIONAL DE ROSARIO**

Facultad de Ciencias Económicas y Estadística


# "Metropolis-Hastings"

Estadística Bayesiana - Trabajo Práctico Nº2

*Alumnas:* Agustina Mac Kay, Ailén Salas y Rocío Canteros

Año 2024

# Introducción

El algoritmo de Metropolis-Hastings (MH) permite generar muestras (pseudo-)aleatorias a partir de una distribución de probabilidad $P$ que no necesariamente pertence a una familia de distribuciones conocida. El único requisito es que se pueda evaluar la función de densidad (o de masa de probabilidad) $p^*(\theta)$ en cualquier valor de $\theta$, incluso cuando $p^*(\theta)$ sea impropia (es decir, incluso aunque sea desconocida la constante de normalización que hace que la integral en el soporte de la función sea igual a uno).

Los pasos del algoritmo son:

1. Durante la iteración $i$, se encuentra en el valor del parámetro $\theta^{(i)}$.
2. En función del valor de parámetro actual $\theta^{(i)} = \theta$, se propone un nuevo valor $\theta'$ en función de $q(\theta'|\theta)$.
3. Se decide si se vá a la nueva ubicación $\theta^{(i+1)} = \theta'$ o si se queda $\theta^{(i+1)} = \theta$:

   - Se calcula la probabilidad de salto:

$$\alpha_{\theta \to \theta'} = \min\left\{1, \frac{f(\theta')}{f(\theta)} \frac{q(\theta|\theta')}{q(\theta'|\theta)}\right\}$$

   - Pasar a $\theta'$ con probabilidad $\alpha_{\theta \to \theta'}$:

$$\theta^{(i+1)} = \begin{cases} \theta & con \quad probabilidad \quad \alpha_{\theta \to \theta'} \\ \theta & con \quad probabilidad \quad (1 - \alpha_{\theta \to \theta'}) \end{cases}$$

A continuación, se presenta la función que implementa el algoritmo de Metropolis-Hastings para tomar muestras de una distribución de probabilidad a partir de su función de densidad. Se otorga flexibilidad al algoritmo permitiendo elegir entre un punto de inicio arbitrario o al azar y permitiendo utilizar distribuciones de propuesta de transición arbitrarias (por defecto, se utiliza una distribución normal estándar).

```r
# Función de Metropolis-Hastings

cant_saltos <- 0  # se inicia en 0 el contador de saltos

sample_mh <- function(d_objetivo, r_propuesta, d_propuesta, p_inicial, n) {
  muestras <- matrix(nrow = n, ncol = length(p_inicial))
  muestras[1, ] <- p_inicial

  for(i in 2:n) {
    p_actual <- muestras[i-1,]
    p_nuevo <- r_propuesta(p_actual)

    f_nuevo <- d_objetivo(p_nuevo)
    f_actual <- d_objetivo(p_actual)

    q_actual <- d_propuesta(p_actual, mean = p_nuevo)
    q_nuevo <- d_propuesta(p_nuevo, mean = p_actual)

    alpha <- min(1, (f_nuevo/f_actual)*(q_actual/q_nuevo))
    aceptar <- rbinom(1, 1, alpha)

    if(aceptar) {
      muestras[i,] <- p_nuevo
      cant_saltos <- cant_saltos + 1

    } else {
      muestras[i,] <- p_actual
    }
  }

  if (ncol(muestras) == 1) {
    muestras <- as.vector(muestras)
  }
  return(list(muestras=muestras,cant_saltos=cant_saltos))
}
```

# Metropolis-Hastings en 1D

## Distribución de Kumaraswamy

La distribución de Kumaraswamy es una distribución de probabilidad continua que se utiliza para modelar variables aleatorias con soporte en el intervalo $(0, 1)$. Si bien graficamente la forma de su función de densidad puede hacer recordar a la distribución beta, vale mencionar que la distribución de Kumaraswamy resulta en una expresión matemática cuyo cómputo es más sencillo:

$$p(x|a, b) = abx^{a-1}(1 - x^a)^{b-1}$$

con $a, b > 0$

A continuación, se grafica la función de densidad de la distribución de Kumaraswamy para las combinaciones de los parámetros:

| *Combinación* | $a$ | $b$ |
|---|---|---|
| 1 | 0.2 | 0.2 |
| 2 | 0.1 | 0.7 |
| 3 | 3 | 3 |
| 4 | 4 | 9 |
| 5 | 10 | 5 |

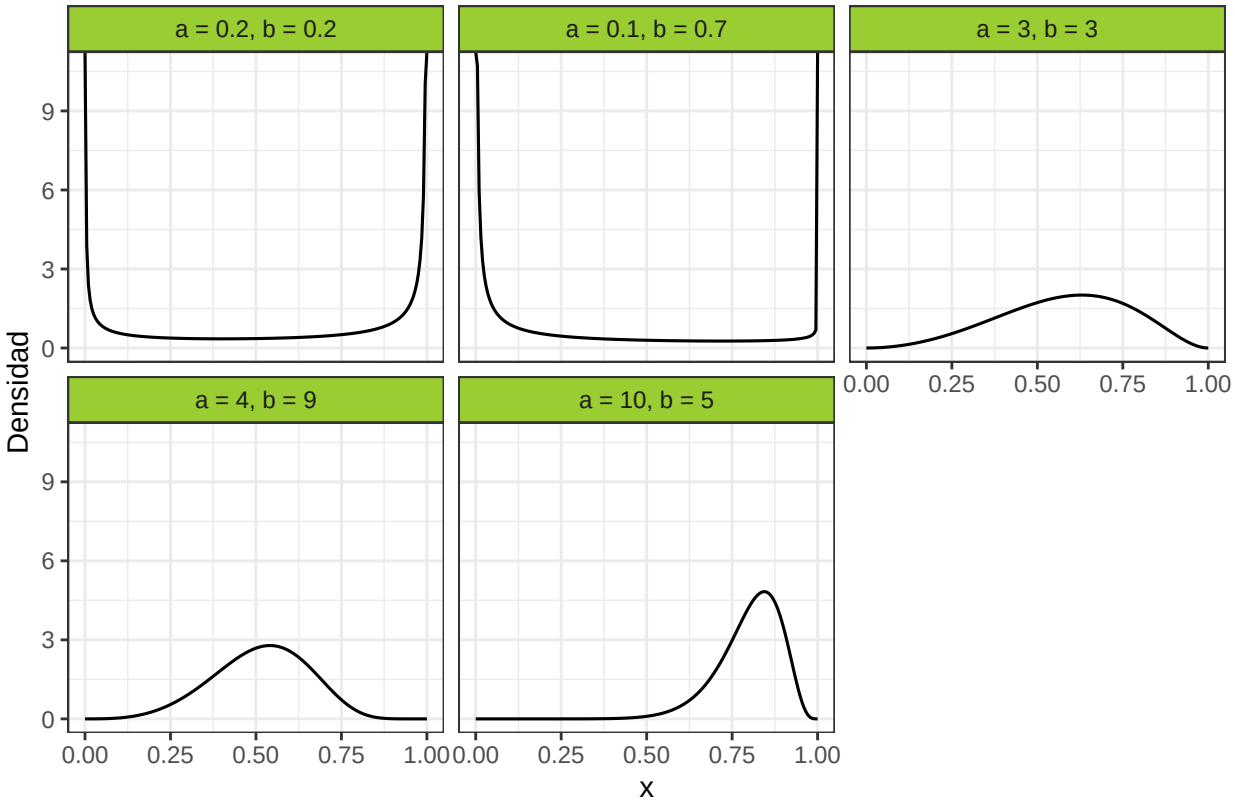



Gráfico 1: distribución Kumaraswamy con distintas combinaciones de los parámetros a y b

En el gráfico 1 se puede apreciar las distintas formas que toman las curvas de la distribución Kumaraswamy dependiendo de los parámetros a y b que se elijan. El parámetro $a$ controla la asimetría de la curva. El parámetro $b$ controla la curvatura de la gráfica. Se espera que si $a = b$, la curva sea simétrica. Si $a > b$, la

curva se inclina hacia la derecha. Si $a < b$, la curva se inclina hacia la izquierda. Se observa que: - Si los parámetros son iguales y menores a 1, la curva es simétrica y tiene forma de U. - Si los dos parámetros son menores a 1 y $a < b$, la curva tiene forma de U y es más aplastada del lado derecho. - Si los parámetros son iguales y mayores a 1, la curva es simétrica y tiene forma de campana. - Si ambos parámetros son mayores a 1 y $a < b$, la curva tiene forma de campana. - Si ambos parámetros son mayores a 1 y $a > b$, la curva es asimétrica a la izquierda y tiene forma de campana.

Conocer las distintas formas que puede tomar la curva de la distribución de Kumaraswamy según los parámetros $a$ y $b$ es útil en Estadística Bayesiana porque: - Facilita la elección de un prior que refleje adecuadamente las creencias previas sobre los parámetros del modelo. Esto es crucial para obtener inferencias precisas y robustas. - Permite adapar el modelo a diferentes tipos de datos, ya que la distribución puede variar ampliamente de forma dependiendo de los valores de $a$ y $b$. -Ayuda a comprender cómo los valores de los parámetros afectan el posterior y, por lo tanto, las conclusiones que se pueden extraer del análisis.

Utilizando la función construída al comienzo, se obtienen 5000 muestras de una distribución Kumaraswamy con parámetros $a = 6$ y $b = 2$. Como distribución propuesta se utiliza una beta con los siguientes grados de concentración:

$$Concentración \quad 4 \quad 10 \quad 20$$

Como punto inicial del algoritmo de MH, se obtiene un valor aleatorio de una distribución $beta(2, 2)$

La tasa de aceptación en el algoritmo de Metropolis-Hastings indica qué tan frecuentemente se aceptan los nuevos $\theta$ propuestos, en relación al total de $\theta$ propuestos.

| Concentracion | Tasa.de.aceptación |
|---|---|
| 4 | 0.4628 |
| 10 | 0.6220 |
| 20 | 0.7340 |

[a] Tabla 1: Tasa de aceptación

En la tabla 1 se observa que, a mayor concentración de la distribución propuesta beta, mayor es la tasa de aceptación.

A continuación, una representación gráfica que muestra cómo evolucionan las muestras generadas por el algoritmo a lo largo del tiempo.

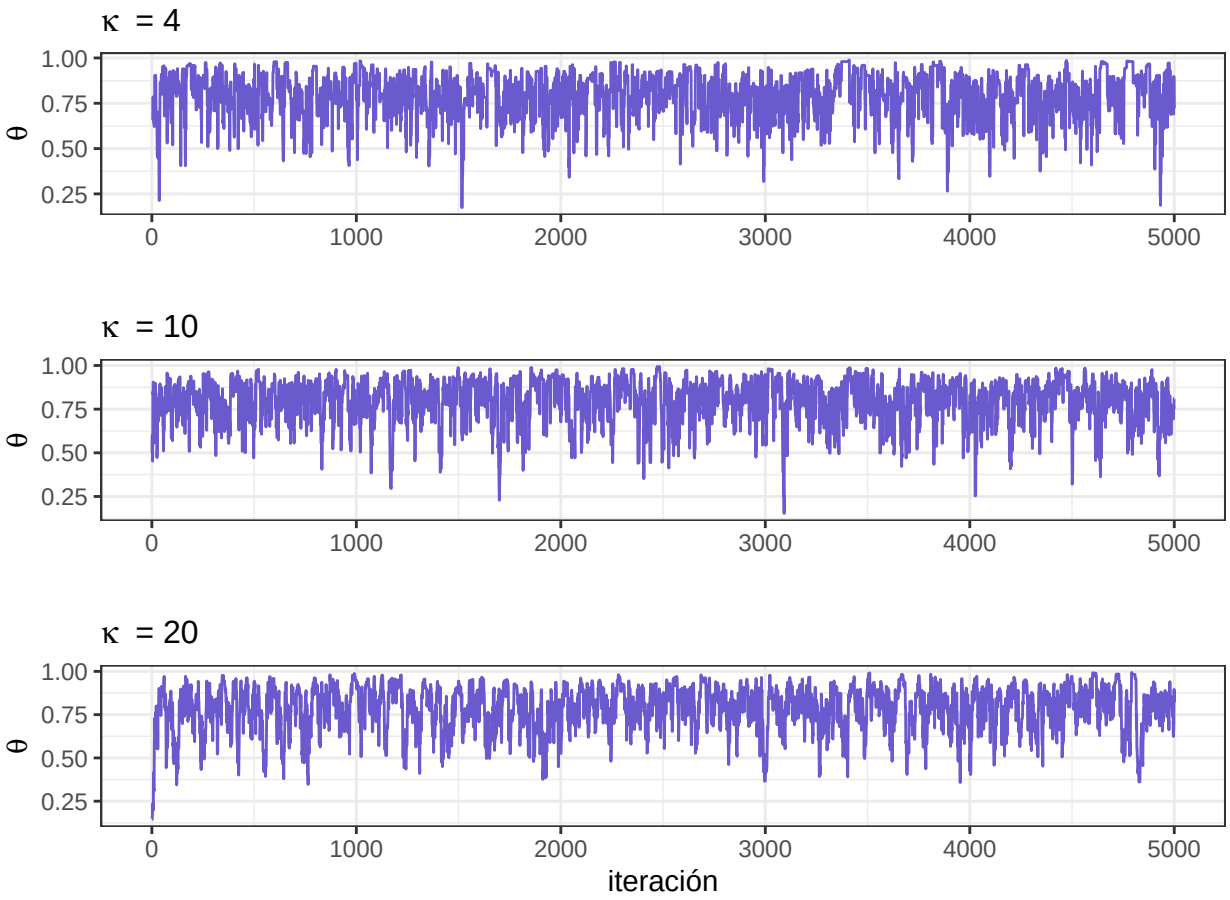



Gráfico 2: Plot trace según concentración

```
## NULL
```

El gráfico 2 muestra que para las 3 concentraciones elegidas, el trace plot resulta ser un ruido blanco sin ningún patrón particular. Sin embargo, aunque es dificil apreciar de manera detallada el comportamiento del algoritmo debido a la cantidad de muestras, pareciera ser que el de concentración $k = 10$ tiene un comportamiento más apropiado. Esto se debe a que se puede apreciar algunos estancamientos del valor de $\theta$ en los de concentraciones $k = 4$ y $k = 20$.

Para evaluar la convergencia de las muestras a la distribución objetivo se presenta:

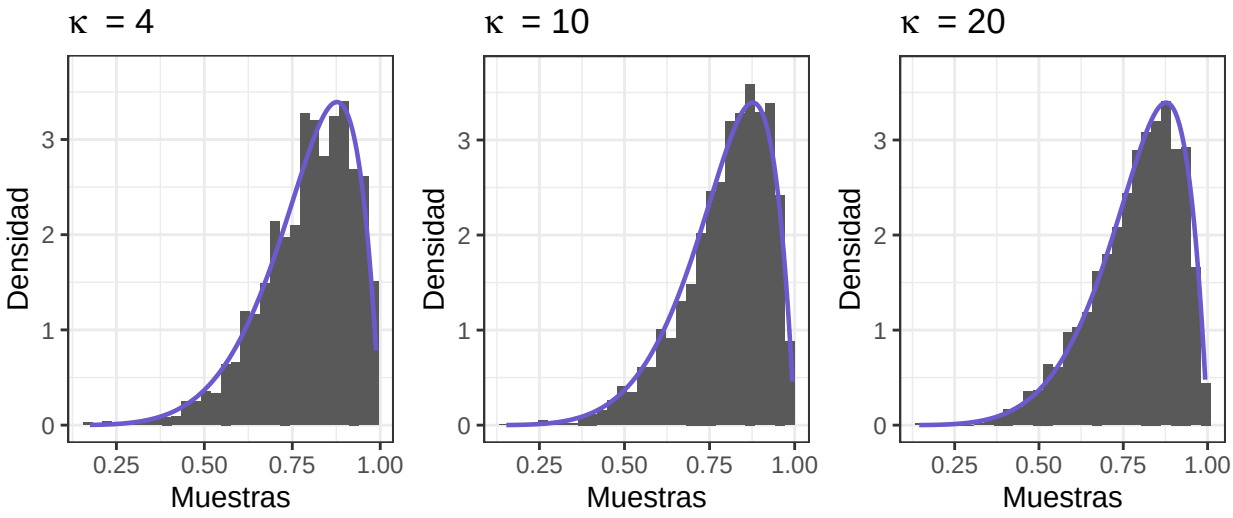



Gráfico 3: muestras obtenidas y distribución de kumaraswamy según concentración

```
## NULL
```

En el gráfico 3 se observa que las muestras generadas para los 3 valores de concentración se ajustan a la distribución objetivo. Las 3 muestras exploran el rango completo de la distribución a posteriori. Sin embargo, pareciera que las concentraciones $k = 10$ y $k = 20$ ajustan mejor (a excepción de una barra muy alta en $k = 10$)

Se calcula la correlación de la serie para cada valor de lag k consigo misma originando la función de autocorrelación:

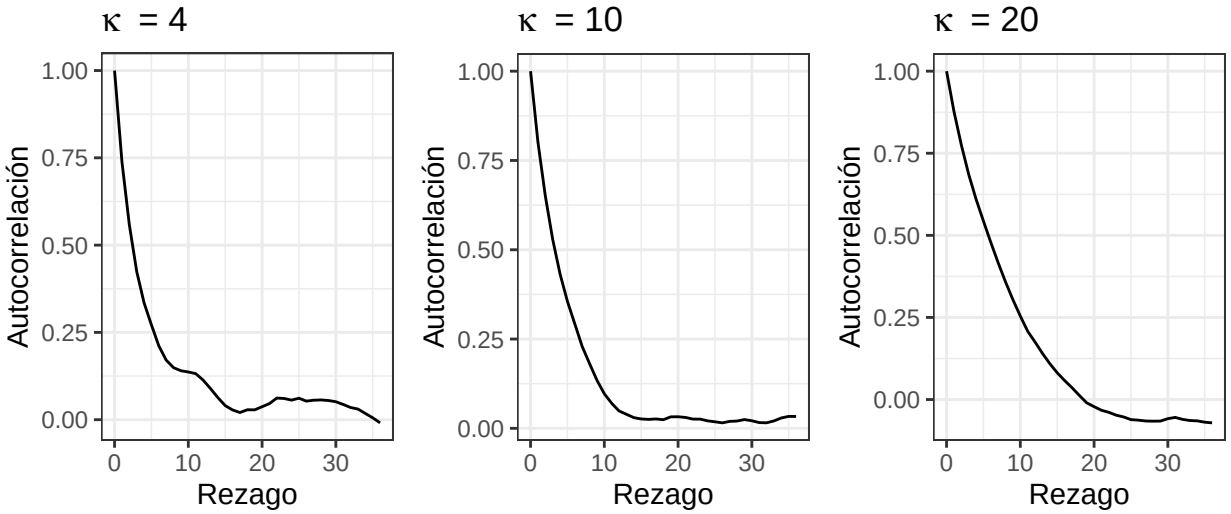



Gráfico 4: Autocorrelación según concentración

```
## NULL
```

Las muestras tienen que ser independientes. La dependencia de valores anteriores tiene que desaparecer rápido. En el gráfico 4 se observa que esto ocurre para los 3 valores de concentración. Sin embargo, a diferencia de los resultados obtenidos anteriormente, este gráfico sugiere que el valor más adecuado para la

concentración es $k = 4$, teniendo una correlación de 0.25 en el rezago 5. Cabe destacar que la diferecia con las otras concentraciones no es grande, siendo éstas de 0.25 en los rezagos 6 y 8 aproximadamente.

Para cada una de las cadenas anteriores, se presentan la media de la distribución y los percentiles de $X$:

| Muestra | Concentración | Medias | Percentil_5 | Percentil_95 |
|---|---|---|---|---|
| 1 | k=4 | 0.7948624 | 0.5545698 | 0.9634114 |
| 2 | k=10 | 0.7980176 | 0.5490458 | 0.9569654 |
| 3 | k=20 | 0.7855301 | 0.5283012 | 0.9547644 |

[a] Tabla 2: Media y percentiles de X

Para los 3 valores de concentración, se obtienen resultados muy similares para la media (0.79) y para los percentiles 5 y 95 de la distribución, siendo éstos 0.55 y 0.96 respectivamente.

Para cada una de las concentraciones, se presentan la media y los percentiles de la distribución de $logit(X)$:

| Muestra | Concentración | Medias | Percentil_5 | Percentil_95 |
|---|---|---|---|---|
| 1 | k=4 | 1.576201 | 0.219152 | 3.270745 |
| 2 | k=10 | 1.588587 | 0.196816 | 3.101762 |
| 3 | k=20 | 1.503743 | 0.113326 | 3.049580 |

[a] Tabla 3: Media y percentiles de logit(X)

Al hacer uso de todo el eje real mediante el $logit(x)$, se observa que la media para $k = 20$ difiere un poco de las medias correspondientes a las otras 2 concentraciones. A su vez, los percentiles 5 y 95 que difieren más son los de concentración $k = 4$. Cabe destacar que no se consideran relevantes estas diferencias.

**Conclusión**

Al analizar las distintas muestras de distribución Kumaraswamy de parámetros $a = 6$ y $b = 2$ con propuesta beta de distintas concentraciones, no se encontraron grandes diferencias entre éstas. No es posible decidir con certeza cuál de los valores de concentración es mejor, debido a que las diferencias obtenidas han sido muy pequeñas. Sin embargo, si habría que sugerir un valor de concentración, se sugiere el 10. Esto es debido a que tiene una tasa de aceptación moderada (0.65), el algoritmo no se estanca demasiado en los mismos valores de $\theta$, la muestra explora el rango completo de la distribución a posteriori y se ajusta bien a la curva. Además, la autocorrelación decrece rápidamente y es menor a 0.25 a partir del rezago 6.

# Metropolis-Hastings en 2D

La verdadera utilidad del algoritmo de Metropolis-Hastings se aprecia cuando se obtienen muestras de distribuciones en más de una dimensión, incluso cuando no se conoce la constante de normalización. En esta sección se trabaja con ejemplos que permitirán advertir las limitaciones del algoritmo y motivarán la búsqueda de mejores alternativas.

## Normal multivariada

La distribución normal multivariada es la generalización de la distribución normal univariada a múltiples dimensiones (o mejor dicho, el caso en una dimensión es un caso particular de la distribución en múltiples dimensiones). La función de densidad de la distribución normal en $k$ dimensiones es:

$$p(x|\mu, \Sigma) = \frac{1}{(2\pi)^{k/2}|\Sigma|^{1/2}} exp(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu))$$

donde $\mu$ es el vector de medias y $\Sigma$ la matriz de covarianza.

Utilizando la función descrita al principio del informe se obtienen muestras de una distribución normal bivariada con media $\mu^*$ y matriz de covarianza $\Sigma^*$.

$$\mu^* = \begin{bmatrix} 0.4 \\ 0.75 \end{bmatrix}$$

$$\mathbf{\Sigma^*} = \begin{bmatrix} 1.35 & 0.4 \\ 0.4 & 2.4 \end{bmatrix}$$

Con el objetivo de analizar cual es la matriz de covarianza para la distribución propuesta más óptima, se prueba con las siguientes:

$$\mathbf{\Sigma^1} = \begin{bmatrix} 2 & 0.5 \\ 0.5 & 31.35 \end{bmatrix}$$

$$\mathbf{\Sigma^2} = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix}$$

$$\mathbf{\Sigma^3} = \begin{bmatrix} 0.4 & 0 \\ 0 & 0.4 \end{bmatrix}$$

$$\mathbf{\Sigma^4} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\mathbf{\Sigma^5} = \begin{bmatrix} 5 & 0 \\ 0 & 6 \end{bmatrix}$$

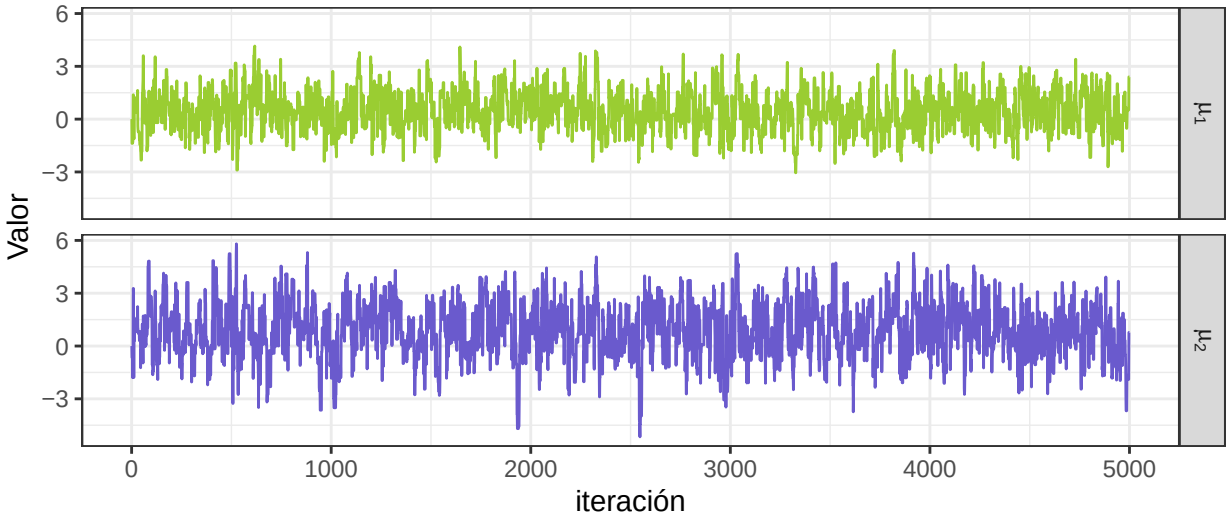



Gráfico 5: Trace plot de la matriz 1

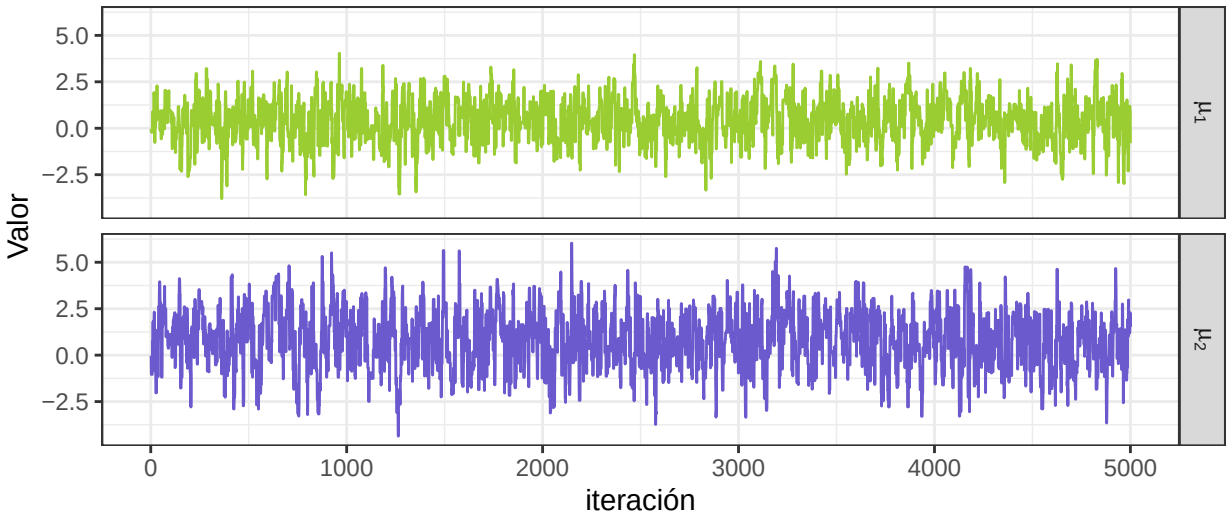



Gráfico 6: Trace plot de la matriz 2

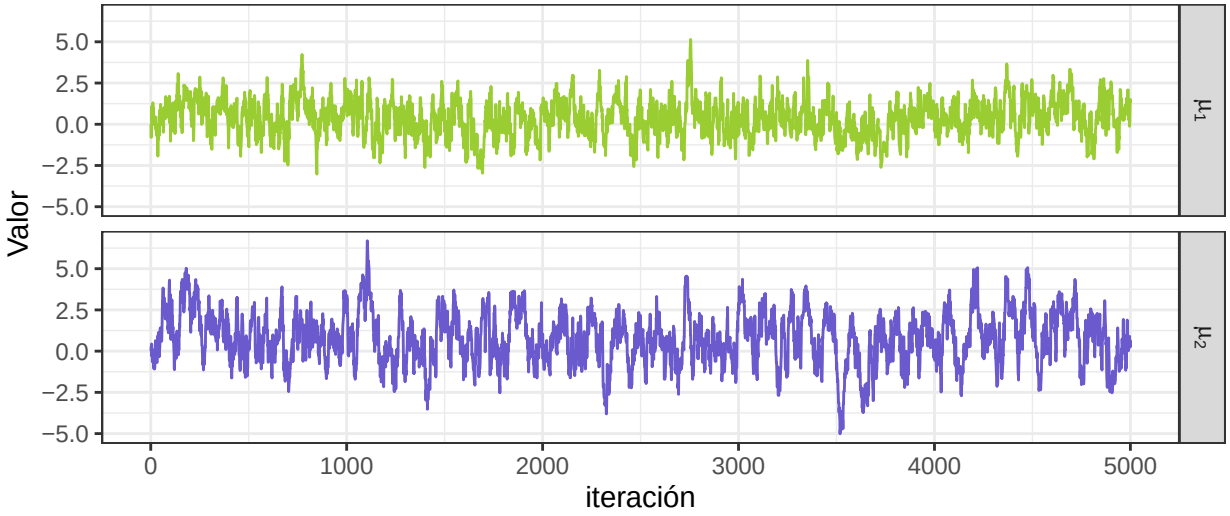



Gráfico 7: Trace plot de la matriz 3

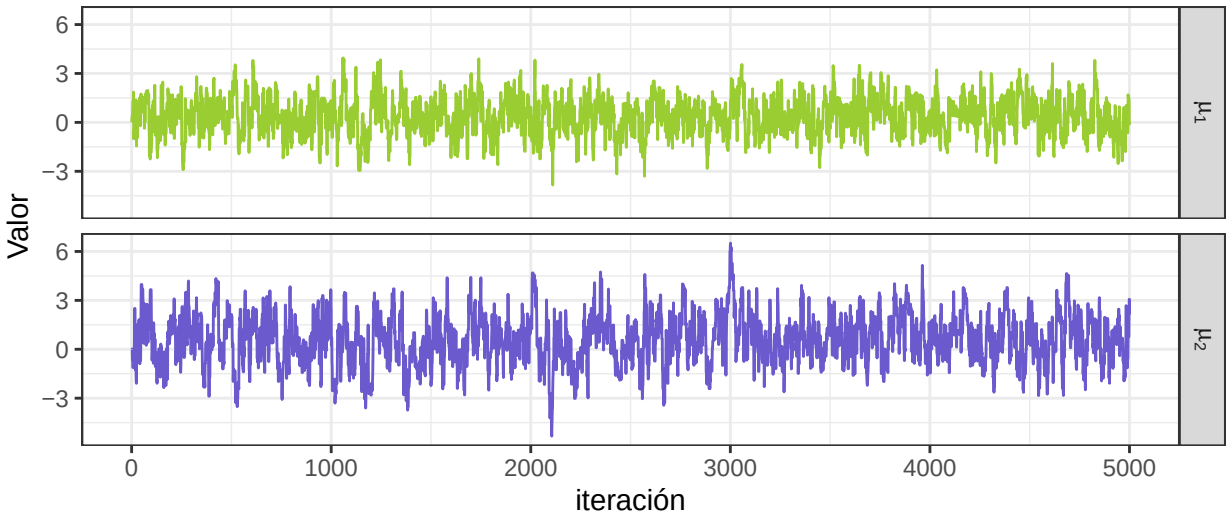



Gráfico 8: Trace plot de la matriz 4

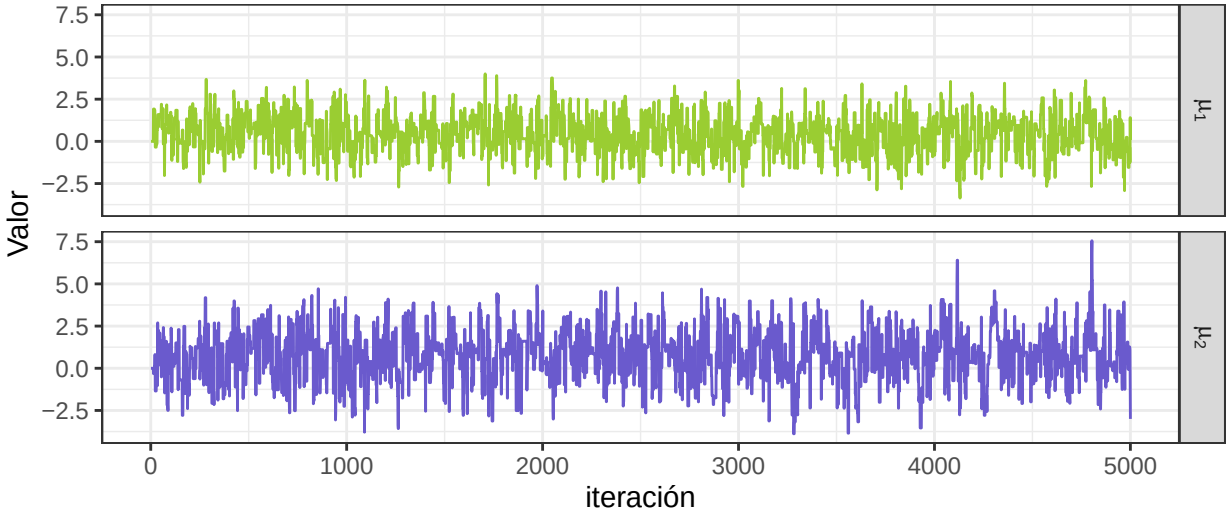



Gráfico 9: Trace plot de la matriz 5

En los gráficos 5 a 9 se observa que, para los dos parámetros, las iteraciones oscilan alrededor del cero. Puesto que la cantidad de observaciones es elevada, no se aprecian con claridad los estancamientos del valor de $\theta$. Pareciera que con la matriz 3 es con la cual más se estanca.

| Matriz | Num_efectivo_p1 | Num_efectivo_p2 |
|---|---|---|
| 1 | 633 | 488 |
| 2 | 586 | 564 |
| 3 | 159 | 167 |
| 4 | 458 | 309 |
| 5 | 647 | 572 |

[a] Tabla 4: Números efectivos de muestras para cada matriz y cada parámetro

En la tabla 4 se observa que la matriz de covarianza de la distribución propuesta que devuelve un valor más óptimo de muestras efectivas es la matriz 5. Teniendo en cuenta que la cantidad de muestras es 5000, este resulta ser un valor bajo. De todas maneras, al ser el más alto, se elige esta matriz.

A partir de las muestras obtenidas con la matriz 5, se estiman las siguientes probabilidades:

i. $Pr(X_1 > 1, X_2 > 0)$

ii. $Pr(X_1 > 1, X_2 > 2)$

iii. $Pr(X_1 > 0.4, X_2 > 0.75)$

Luego, mediante la función de la distribución de la normal bivariada se obtienen las probabilidades reales con el objetivo de compararlas y ver si se obtuvo una buena muestra con el método anterior.

| Probabilidad | Estimada | Real |
|---|---|---|
| i | 0.0826 | 0.0682514 |
| ii | 0.0898 | 0.0870087 |
| iii | 0.3014 | 0.2856655 |

[a] Tabla 5: Probabilidades estimadas y reales

Según lo observado en la tabla 5, se podría considerar que se tiene una buena muestra de una normal bivariada con media $\mu^*$ y matriz de covarianza $\Sigma^*$ a través del algoritmo de Metrópolis-Hastings en 2D, con la matriz de covarianza para la distribución propuesta $\Sigma^5$.

**Conclusión**

Utilizando el algoritmo de Metrópolis-Hastings en 2D para generar muestras de una distribución normal bivariada con la media y la matriz de varianza y covarianza especificadas, se observa que la elección de la matriz de varianza y covarianza de la distribución propuesta ($\mathbf{\Sigma^5}$) influye en la eficiencia del muestreo. Al comparar con las otras matrices de varianza y covarianza, $\mathbf{\Sigma^5}$ presenta un número efectivo de muestras más alto, lo que indica una exploración más eficiente del espacio de parámetros.

## Función de Rosenbrock

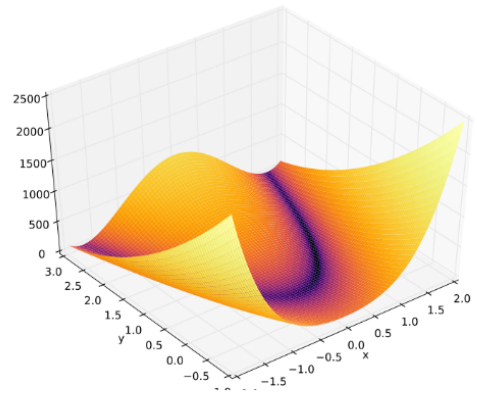
La función de Rosenbrock, a veces llamada el "valle de Rosenbrock", y comunmente conocida como la "banana de Rosenbrock", es una función matemática utilizada frecuentemente como un problema de optimización y prueba para algoritmos de optimización numérica.

La función está definida por:

$$f(x, y) = (a - x)^2 + b(y - x^2)^2$$

y cuenta con un mínimo global en $(x, y) = (a, a^2)$, que satisface $f(a, a^2) = 0$.

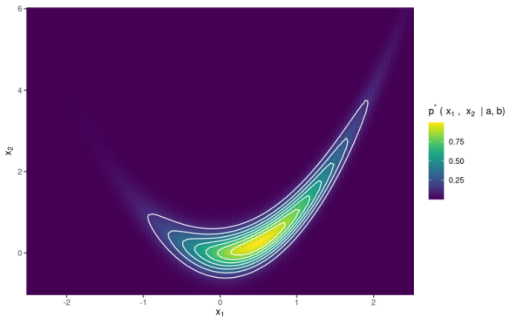
Debido a su forma peculiar, la función de Rosenbrock presenta desafíos particulares para los algoritmos de optimización, ya que tiene un valle largo y estrecho en el que la convergencia puede ser lenta.




Función de Rosenbrock

Esta forma de banana popularizada por Rosenbrock es también muy conocida en el campo de la estadística bayesiana, ya que en ciertos escenarios, la densidad del posterior toma una forma que definitivamente se asemeja a la banana de Rosenbrock. Un ejemplo de este fenómeno es la función $p^*$:

$$p^*(x_1, x_2|a, b) = exp(-[(a - x_1)^2 + b(x_2 - x_1^2)^2])$$

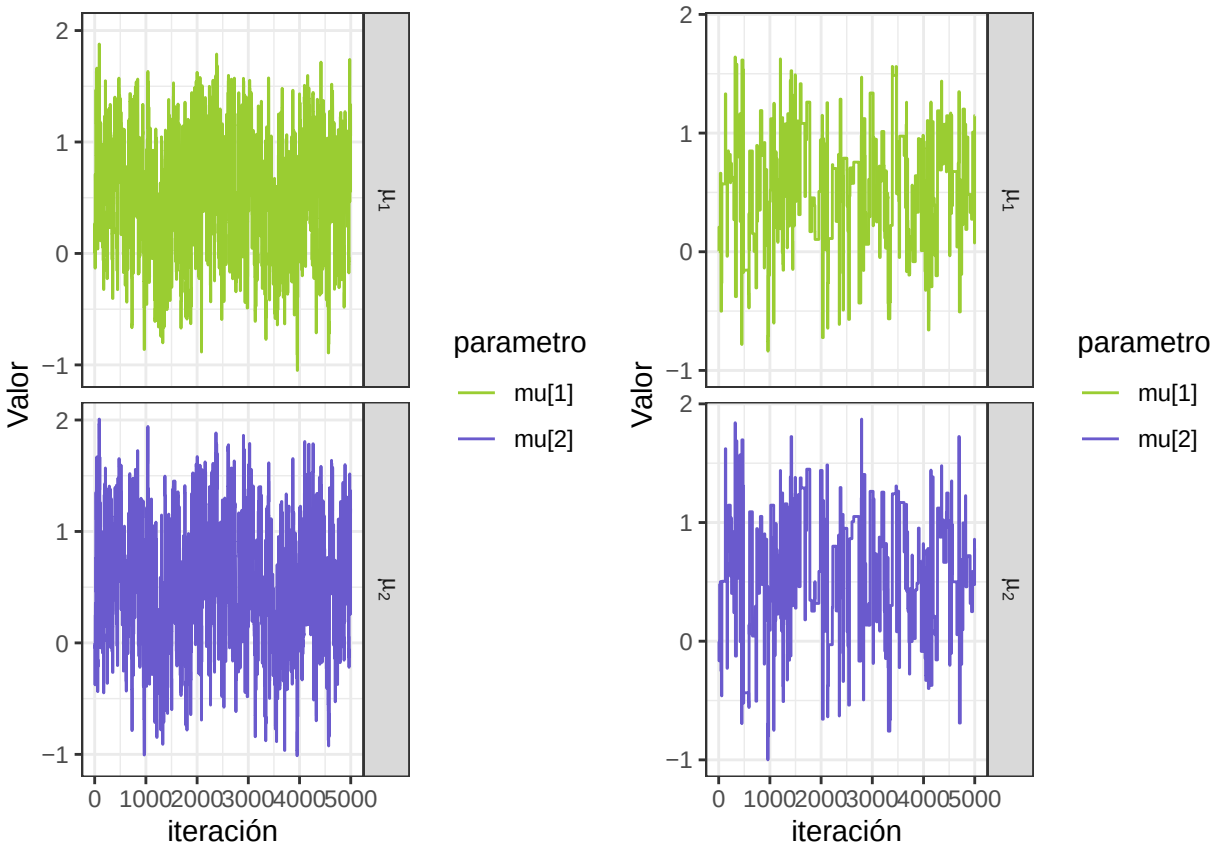



Función de densidad de la que se desean obtener muestras con $a = 0.5$ y $b = 5$

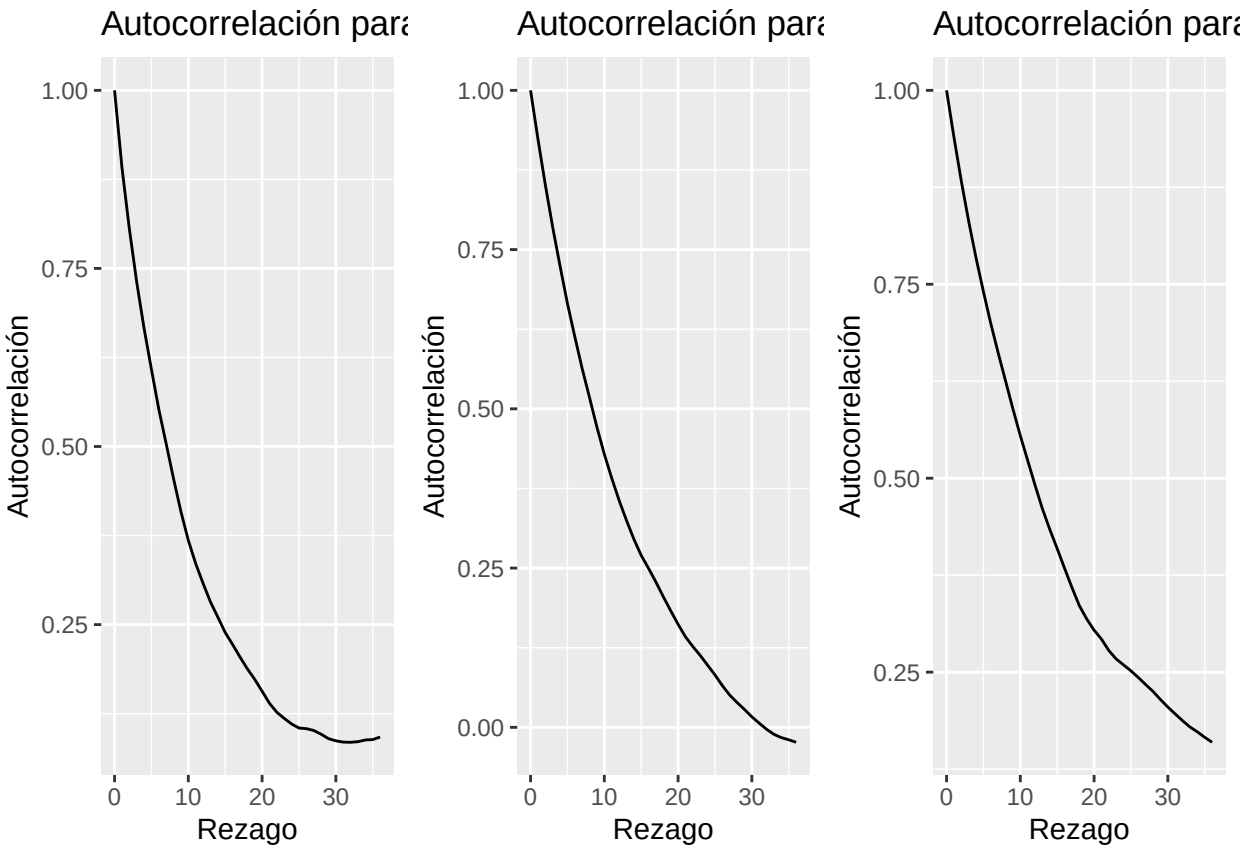
```r
#Punto 9

p_estrella <- function(x, a, b){
  exp(-((((a - x[1])^2) + b * ((x[2] - x[1])^2))^2))
}



n9 <- 5000

funcion9_1 <- function(matriz_cov) {
    d_objetivo9 <- function(x) p_estrella(x, 0.5, 5)
    d_propuesta9 <- function(x, mean) dmvnorm(x, mean = mean, sigma = matriz_cov)
    r_propuesta9 <- function(x) rmvnorm(1, mean = x, sigma = matriz_cov)

    funcion9 <- sample_mh(d_objetivo9,r_propuesta9,d_propuesta9,c(0,0),n9)
    muestra9 <- funcion9$muestras
    grafico9_1 <- plot_trace(muestra9[,1],muestra9[,2])
```

```
    n_eff9_1 <- n_eff(muestra9[,1])
    n_eff9_2 <- n_eff(muestra9[,2])
    tasa9 <- funcion9$cant_saltos/n9
    autocorr_1 <- acf(muestra9[,1], plot = F)
    autocorr_2 <- acf(muestra9[,2], plot = F)

    return(list(grafico = grafico9_1, n1 = n_eff9_1, n2 = n_eff9_2,
             muestra9 = muestra9, tasa9 = tasa9, autocorr_p1 = autocorr_1,
             autocorr_p2 = autocorr_2))

}


matriz1 <- funcion9_1(matrix(c(0.3, 0.1, 0.1, 0.2), nrow = 2))
matriz2 <- funcion9_1(matrix(c(2, 0, 0, 2), nrow = 2))
matriz3 <- funcion9_1(matrix(c(4, 1, 1, 4), nrow = 2))



grafico9_1_1 <- ggplot()+
  geom_line(aes(y = matriz1$autocorr_p1$acf, x = matriz1$autocorr_p1$lag)) +
  labs (title = "Autocorrelación para 'a' ", x = "Rezago", y = "Autocorrelación")

grafico9_1_2 <- ggplot()+
  geom_line(aes(y = matriz1$autocorr_p2$acf, x = matriz1$autocorr_p2$lag)) +
  labs (title = "Autocorrelación para 'b' ", x = "Rezago", y = "Autocorrelación")


grafico9_2_1 <- ggplot()+
  geom_line(aes(y = matriz2$autocorr_p1$acf, x = matriz2$autocorr_p1$lag)) +
  labs (title = "Autocorrelación para 'a' ", x = "Rezago", y = "Autocorrelación")

grafico9_2_2 <- ggplot()+
  geom_line(aes(y = matriz2$autocorr_p2$acf, x = matriz2$autocorr_p2$lag)) +
  labs (title = "Autocorrelación para 'b' ", x = "Rezago", y = "Autocorrelación")

grafico9_3_1 <- ggplot()+
  geom_line(aes(y = matriz3$autocorr_p1$acf, x = matriz3$autocorr_p1$lag)) +
  labs (title = "Autocorrelación para 'a' ", x = "Rezago", y = "Autocorrelación")


grafico9_3_2 <- ggplot()+
  geom_line(aes(y = matriz3$autocorr_p2$acf, x = matriz3$autocorr_p2$lag)) +
  labs (title = "Autocorrelación para 'b' ", x = "Rezago", y = "Autocorrelación")
```

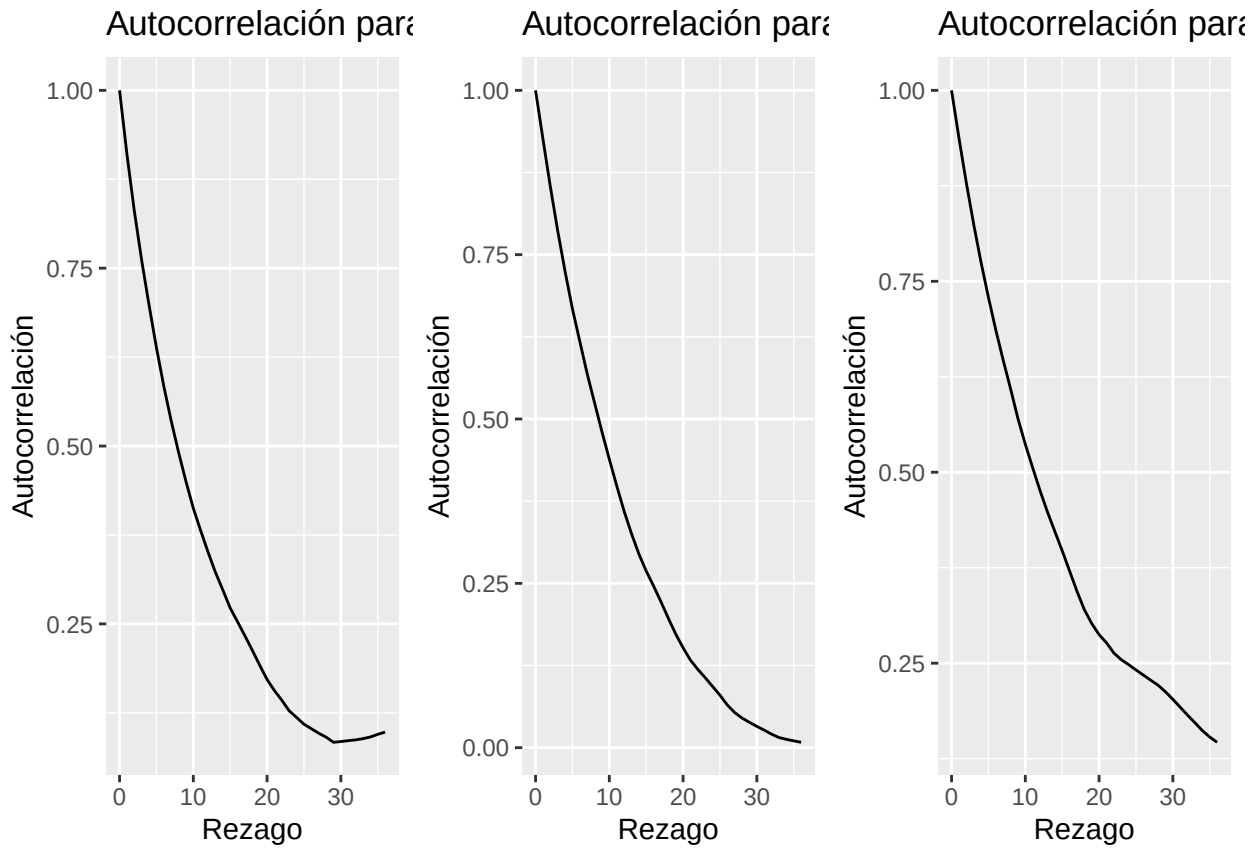
```
grid.arrange(matriz1$grafico, matriz3$grafico, ncol= 2)
```




```
tasas <- data.frame(
  Matriz = c("1", "2", "3"),
  Tasa = c(matriz1$tasa9, matriz2$tasa9, matriz3$tasa9)
)


grid.arrange(grafico9_1_1, grafico9_2_1, grafico9_3_1, ncol = 3)
```




```
grid.arrange(grafico9_1_2, grafico9_2_2, grafico9_3_2, ncol = 3)
```




```
#Punto 10
muestra9 <- as.data.frame(matriz1$muestra9)
colnames(muestra9) <- c("X", "Y")

#Probabilidades estimadas
#Probabilidad n°1:
prob1 <- muestra9 %>%
  filter(X > 0 & X < 1 ) %>%
  count(Y < 1 & Y > 0)

prob1$n[2]/(n9)
```

```
## [1] 0.4666
```

```
#Probabilidad n°2:
prob2 <- muestra9 %>%
  filter(X > -1 & X < 0 ) %>%
  count(Y < 1 & Y > 0)

prob2$n[2]/(n9)
```

```
## [1] 0.0404
```

```r
#Probabilidad n°3:
prob3 <- muestra9 %>%
  filter(X > 1 & X < 2 ) %>%
  count(Y < 3 & Y > 2)

prob3$n[2]/(n9)
```

```
## [1] 0.001
```

```r
#Punto 11
#Aproximación por el método de la grilla:

#Creación de grillas para los parámetros
grid_a <- seq()
```

Se vuelven a calcular las probabilidades anteriores, esta vez utilizando la integración de Monte Carlo:

```r
# Punto 11
# Calculo P(0<X1<1, 0<X2<1)

k <- 10000
u <- matrix(runif(k*2), ncol = 2)
p_estrella_eval <- numeric(k)

for (i in 1:k) {
  p_estrella_eval[i] <- p_estrella(u[i,], 0.5, 5)
}

rdo_prob1 <- mean(p_estrella_eval)

# Calculo P(-1<X1<0, 0<X2<1)

u <- matrix(c(runif(k, min = -1, max = 0),
              runif(k, min = 0, max = 1)),
            ncol = 2)

for (i in 1:k) {
  p_estrella_eval[i] <- p_estrella(u[i,], 0.5, 5)
}

rdo_prob2 <- mean(p_estrella_eval)

# Calculo P(1<X1<2, 2<X2<3)

u <- matrix(c(runif(k, min = 1, max = 2),
              runif(k, min = 2, max = 3)),
            ncol = 2)

for (i in 1:k) {
  p_estrella_eval[i] <- p_estrella(u[i,], 0.5, 5)
}

rdo_prob3 <- mean(p_estrella_eval)
```