Tarea4

Alberto, Ivan, Sara, Valeria

Tarea 4

```
library(ggplot2)
library(tidyverse)
library(patchwork)
library(kableExtra)
```

1. Spiegelhalter et al. (1995) analiza la mortalidad del escarabajo del trigo en la siguiente tabla, usando BUGS.

Dosis_wi	muertos_yi	expuestos_ni
1.6907	6	59
1.7242	13	60
1.7552	18	62
1.7842	28	56
1.8113	52	63
1.8369	53	59
1.8610	61	62
1.8839	60	60

Estos autores usaron una parametrización usual en dos parámetros de la forma $p_i \equiv P(\text{muerte}|w_i)$, pero comparan tres funciones ligas diferentes:

$$\text{logit: } p_i = \frac{\exp(\alpha + \beta z_i)}{1 + \exp(\alpha + \beta z_i)}$$

$$\text{probit: } p_i = \Phi(\alpha + \beta z_i)$$

$$\text{complementario log-log: } p_i = 1 - \exp[-\exp(\alpha + \beta z_i)]$$

en donde se usa la covariada centrada $z_i = w_i - \bar{w}$ para reducir la correlación entre la ordenada α y la pendiente β. En OpenBUGS el código para implementar este modelo es el que sigue:

```
modelo1 = "model{
for (i in 1:k){
y[i] ~ dbin(p[i],n[i])
logit(p[i]) <- alpha + beta*(w[i]-mean(w[])
# probit(p[i]) <- alpha + beta*(w[i]-mean(w[])
# cloglog(p[i]) <- alpha + beta*(w[i]-mean(w[])
} #fin del loop i
alpha ~ dnorm(0.0,1.0e-3)
dbeta ~ dnorm(0.0,1.0e-3)
} #fin del código"
```

Lo que sigue al símbolo # es un comentario, así que esta versión corresponde al modelo logit. También *dbin* denota la distribución binomial y *dnorm* denota la distribución normal, donde el segundo argumento denota la precisión, no la varianza (entonces las iniciales normales para α y β tienen precisión 0.001, que son aproximadamente iniciales planas (no informativas). Hacer el análisis en OpenBUGS.

2. Casella y George (1992)

Consideren las siguientes dos distribuciones condicionales completas, analizadas en el artículo de Casella y George (1992) que les incluí como lectura:

$f(x|y) \propto ye^{-yx}$, $0 < x < B < \infty$

$f(y|x) \propto xe^{-xy}$, $0 < y < B < \infty$

- Obtener un estimado de la distribución marginal de X cuando B = 10 usando el Gibbs sampler.

necesitamos simular muestras de las distribuciones condicionales $f(x \mid y)$ y $f(y \mid x)$ para estimar la marginal de X

El algoritmo de Gibbs sampler quedaría de la siguiente manera:

1. Inicializamos x y y con valores dentro del rango de soporte (ejemplo, $x0 = y0 = 1$).
2. Iteramos sobre un gran número de pasos, en cada paso:
    a. Muestreamos x de $f(x \mid y)$ utilizando el valor actual de y.
    b. Muestrear y de $f(y \mid x)$ utilizando el valor actual de x.

3. Después de un número grande de iteraciones, las muestras de x se pueden utilizar para estimar la distribución marginal de X.

Utilizaremos la librería rstan para realizar este ejercicio.

```
Loading required package: StanHeaders
```

```
rstan version 2.32.5 (Stan version 2.32.2)
```

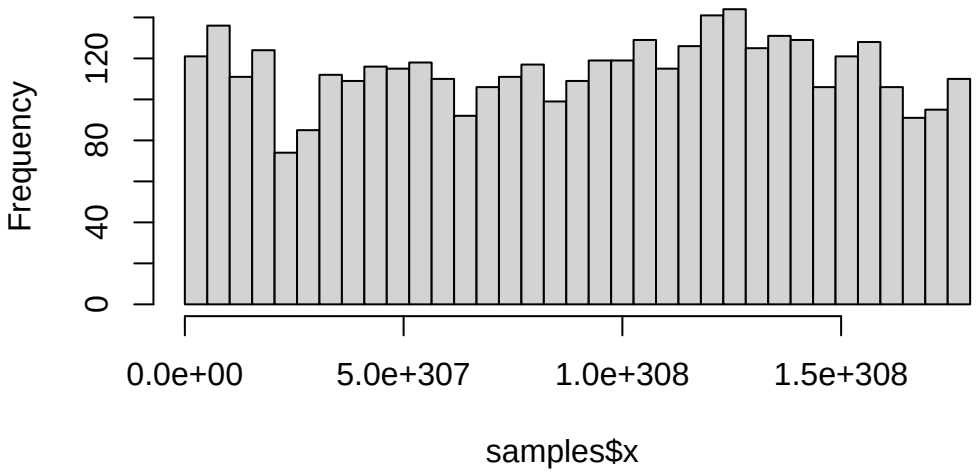
```
For execution on a local, multicore CPU with excess RAM we recommend calling
options(mc.cores = parallel::detectCores()).
To avoid recompilation of unchanged Stan programs, we recommend calling
rstan_options(auto_write = TRUE)
For within-chain threading using `reduce_sum()` or `map_rect()` Stan functions,
change `threads_per_chain` option:
rstan_options(threads_per_chain = 1)
```

```
Attaching package: 'rstan'
```

```
The following object is masked from 'package:tidyr':

    extract
```

```
Trying to compile a simple C file
```

```r
# Obtener los resultados
samples <- extract(fit)

# Visualizar la distribución marginal de x
hist(samples$x, breaks = 30, main = "Distribución Marginal de X")
```



- Ahora supongan que $B = \infty$ así que las distribuciones condicionales completas son ahora las ordinarias distribuciones exponenciales no truncadas. Mostrar analíticamente que $f_x(t) = \frac{1}{t}$ es una solución a la ecuación integral en este caso:
  - $f_x(x) = \int [\int f_{x|y}(x|y) f - y|t(y|t)dy] f_x(t)dt$

Para mostrar analíticamente que $f_X(t) = \frac{1}{t}$ es una solución a la ecuación integral dada, primero necesitamos expresar las distribuciones condicionales $f_{X|Y}(x \mid y)$ y $f_{Y|T}(y \mid t)$ en términos de las distribuciones marginales.

Dado que las distribuciones condicionales ahora son distribuciones exponenciales no truncadas, podemos expresarlas de la siguiente manera:

$f_{X|Y}(x \mid y) = Exp(x \mid y) = ye^{-yx}$

$f_{Y|T}(y \mid t) = Exp(y \mid t) = te^{-ty}$

Ahora, sustituimos estas expresiones en la ecuación integral:

$f_X(x) = \int_0^\infty f_{X|Y}(x \mid y) \cdot f_{Y|T}(y \mid t)dy \cdot f_X(t)dt$

$= \int_0^\infty ye^{-yx} \cdot te^{-ty}dy \cdot \frac{1}{t}dt$

$= \int_0^\infty yt \cdot e^{-(y+t)x}dy \cdot \frac{1}{t}dt$

Para resolver esta integral, primero integramos respecto a y y luego respecto a t:

$= \int_0^\infty \frac{t}{(x+t)^2} dt$

Ahora, para resolver esta integral, podemos hacer un cambio de variable $u = x + t$, entonces $du = dt$, y cuando $t = 0$, $u = x$ y cuando $t = \infty$, $u = \infty$. Así:

$= \int_x^\infty \frac{1}{u2} du$

$= [-\frac{1}{u}]_x^\infty$

$= (-\frac{1}{\infty}) - (-\frac{1}{x})$

$= \frac{1}{x}$

Por lo tanto, $f_X(x) = \frac{1}{x}$, lo cual demuestra que $f_X(t) = \frac{1}{t}$ es una solución a la ecuación integral dada.

- ¿El Gibbs sampler convergerá a esta solución?

No, el método Gibbs sampler no convergerá a la solución $f_X(x) = \frac{1}{x}$ en este caso debido a que se trata de un método para muestrear de distribuciones condicionales específicas, no para converger hacia la distribución marginal exacta.

En el caso en que $B = \infty$, las distribuciones condicionales completas son distribuciones exponenciales no truncadas. Utilizando el Gibbs sampler con estas distribuciones condicionales, se obtendrán muestras de la distribución conjunta, pero no necesariamente convergerán a la distribución marginal de X.

3.Poli Cauchy

- Supongan que una variable aleatoria y se distribuye de acuerdo a la densidad poli-Cauchy:
  - $g(y) = \prod_{i=1}^n \frac{1}{\pi(1+(y-a_i)^2)}$

donde $a = (a_1, ...a_n)$ es un vector de parámetros. Supongan que n = 6 y a = (1, 2, 2, 6, 7, 8). Escriban una función que calcule la log-densidad de y.

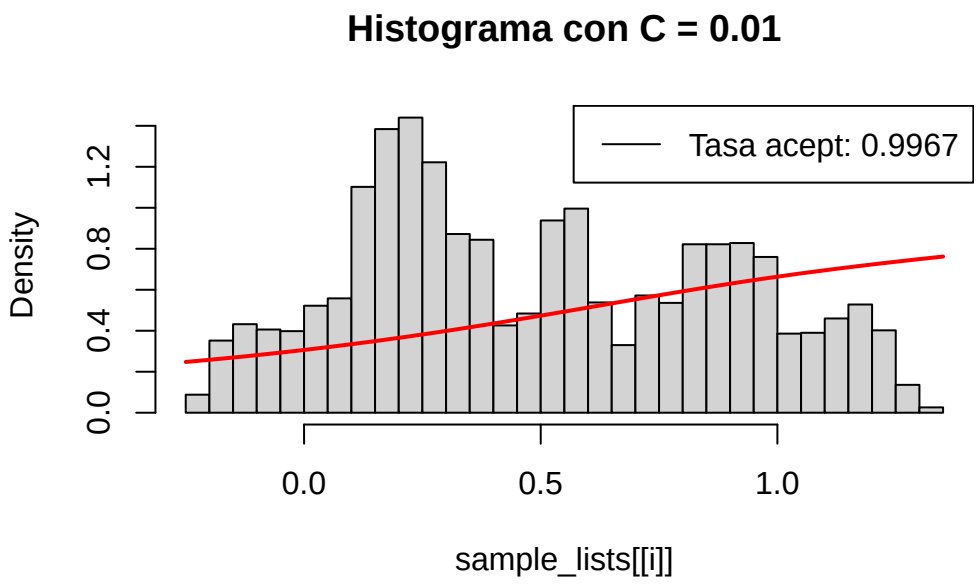
```
log_densidad_poli_cauchy <- function(y, a) {
  n <- length(a)
  log_density <- 0
  for (i in 1:n) {
    log_density <- log_density - log(pi * (1 + (y - a[i])^2))
  }
  return(log_density)
}
```

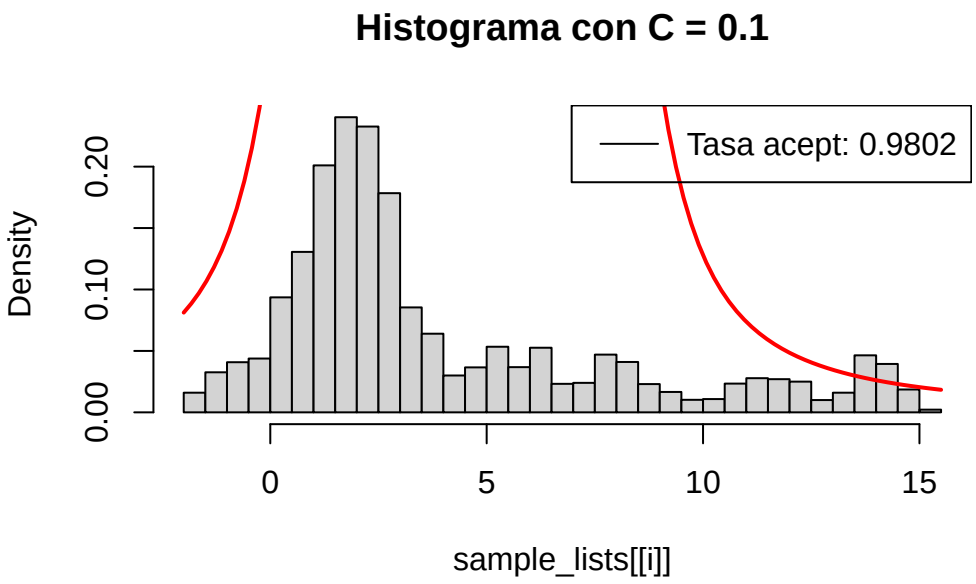
```r
# Parámetros dados
a <- c(1, 2, 2, 6, 7, 8)

# Ejemplo
y <- 3.5
print(paste("Log-densidad de y:", log_densidad_poli_cauchy(y, a)))
```

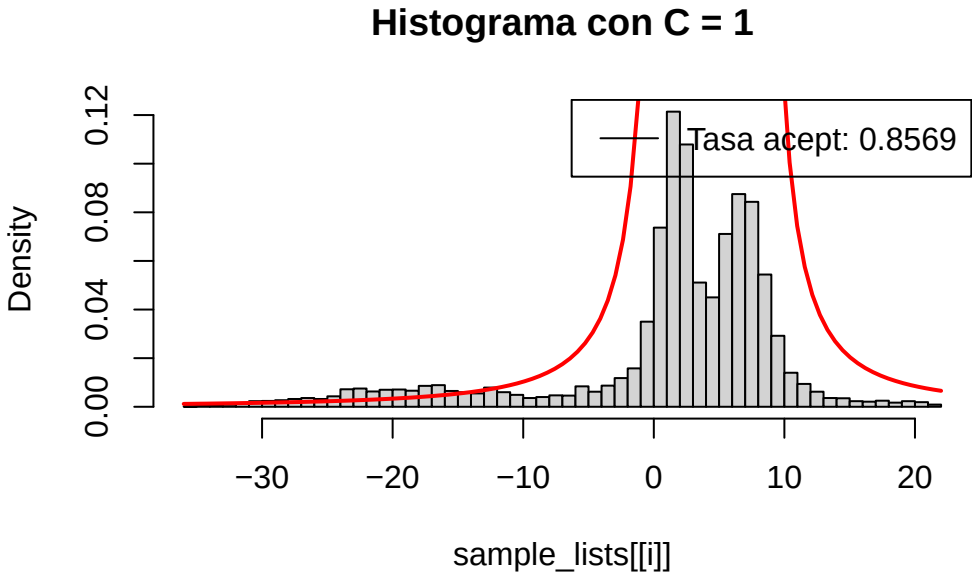
```
[1] "Log-densidad de y: -18.8280466933155"
```

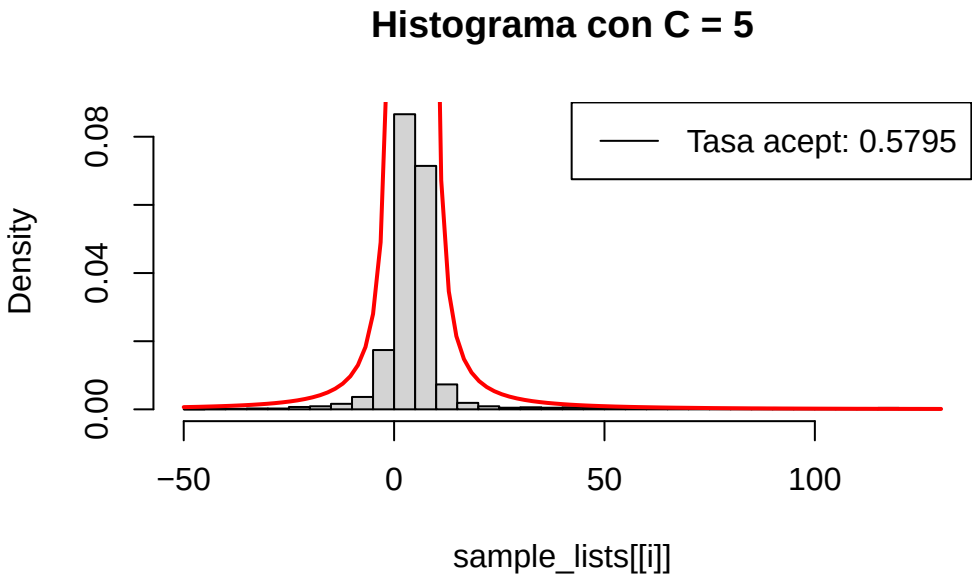
- Escriban una función que tome una muestra de tamaño 10,000 de la densidad de y, usando Metropolis-Hastings con función propuesta una caminata aleatoria con desviación estandar C. Investiguen el efecto de la elección de C en la tasa de aceptación, y la mezcla de la cadena en la densidad.

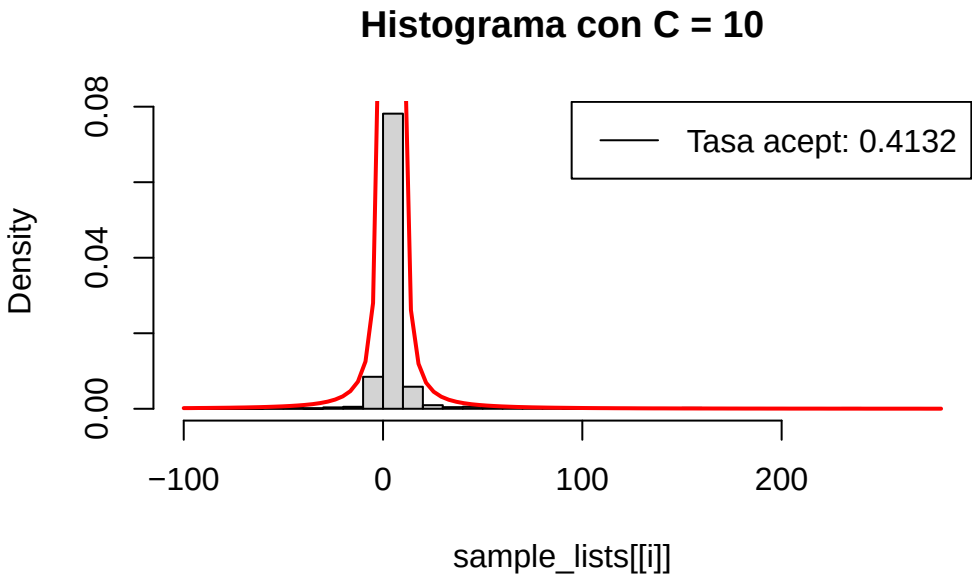
```r
# Mostrar las gráficas para cada valor de C
for (i in seq_along(C)) {
  hist(sample_lists[[i]], breaks = 50, freq = FALSE, main = paste("Histograma con C =", C[
  curve(densidad_poli_cauchy(x, a), col = "red", lwd = 2, add = TRUE)
  legend("topright", legend = paste("Tasa acept:", round(acceptance_rates[i], 4)), col = "
}
```

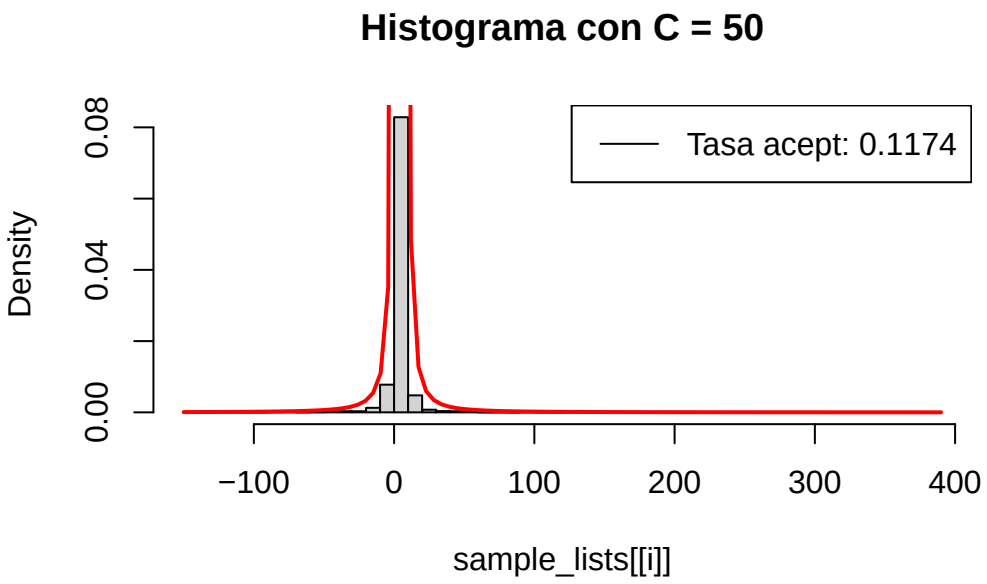



Histograma con C = 0.1
Tasa acept: 0.9802
Density
0.00
0.10
0.20
0
5
10
15
sample_lists[[i]]


Histograma con C = 1
Tasa acept: 0.8569
Density
0.00
0.04
0.08
0.12
−30
−20
−10
0
10
20
sample_lists[[i]]


Histograma con C = 5
Tasa acept: 0.5795
Density
0.00
0.04
0.08
−50
0
50
100
sample_lists[[i]]


Histograma con C = 10
Tasa acept: 0.4132
Density
0.00
0.04
0.08
−100
0
100
200
sample_lists[[i]]



Como podemos ver, el parámetro C en el algoritmo de Metropolis-Hastings controla la desviación estándar de la caminata aleatoria propuesta.

- Impacto general: Un valor mayor de C significa que las propuestas aceptadas se alejarán más de la muestra actual. Por lo tanto, un valor mayor de #C$ permitirá una exploración más rápida del espacio de parámetros, mientras que un valor menor de C limitará el movimiento de las propuestas aceptadas a regiones cercanas a la muestra actual.
- Tasa de aceptación: Si C es demasiado grande, las propuestas serán muy diferentes de la muestra actual, lo que puede resultar en una baja tasa de aceptación, ya que muchas propuestas serán rechazadas. Por otro lado, si C es demasiado pequeño, las propuestas serán muy similares a la muestra actual, lo que también puede conducir a una baja tasa de aceptación debido a que pocas propuestas son aceptadas.
- Mezcla de la cadena: La mezcla de la cadena se refiere a qué tan eficientemente la cadena de Markov generada por el algoritmo de Metropolis-Hastings explora el espacio de parámetros. Si C es demasiado grande, la cadena puede tener dificultades para converger a la distribución objetivo y puede deambular demasiado por el espacio de parámetros. Por otro lado, si C es demasiado pequeño, la cadena puede tardar mucho en explorar diferentes regiones del espacio de parámetros y puede quedarse atascada en óptimos locales.
- Usando la muestra simulada de una "buena" elección de C, aproximar la probabilidad $P(6 < Y < 8)$.

Podemos utilizar una muestra generada con una "buena" elección de C y contar cuántas veces los valores de la muestra caen dentro del intervalo $(6, 8)$. Luego, dividimos este conteo por el tamaño total de la muestra para obtener una aproximación de la probabilidad.

```
# Utilizar la muestra generada con una "buena" elección de C
sample_good_C <- sample_lists[[5]]  # Por ejemplo, tomamos la muestra generada con C = 10

# Calcular la probabilidad P(6 < Y < 8)
prob_6_to_8 <- sum(sample_good_C > 6 & sample_good_C < 8) / length(sample_good_C)

# Imprimir el resultado
print(paste("La probabilidad P(6 < Y < 8) es:", prob_6_to_8))
```

```
[1] "La probabilidad P(6 < Y < 8) es: 0.2123"
```

4. Supongan que el vector (X, Y) tiene función de distribución conjunta:

$$f(x,y) = \frac{x^{a+y-1}e^{-(1+b)x}b^a}{y!\Gamma(a)}, x > 0, y = 0, 1, 2, ...$$

y deseamos simular de la densidad conjunta.

- Mostrar que la densidad condicional $f(x|y)$ es una Gamma e identificar los parámetros.

Primero obtenemos $f(y)$

$$\begin{aligned} f(y) = & \int_0^\infty \frac{x^{a+y-1}e^{-(1+b)x}b^a}{y!\Gamma(a)}dx \\ = & \frac{b^a}{y!\Gamma(a)}\int_0^\infty x^{a+y-1}e^{-(1+b)x} \\ = & \frac{b^a}{y!\Gamma(a)}\Big(\frac{\Gamma(a+y)}{(1+b)^{a+y}}\Big) \end{aligned}$$

Luego aplicamos la definición de desidad condicional:

$$\begin{aligned} f(x|y) = & \frac{f(x,y)}{f(y)} \\ = & \frac{\frac{x^{a+y-1}e^{-(1+b)x}b^a}{y!\Gamma(a)}}{\frac{b^a}{y!\Gamma(a)}\Big(\frac{\Gamma(a+y)}{(1+b)^{a+y}}\Big)} \\ = & \frac{x^{a+y-1}e^{-(1+b)x}}{\Gamma(a+y)}(1+b)^{a+y} \end{aligned}$$

Con lo cual podemos ver que se trata de una $\mathcal{G}(a+y, 1+b)$

- Mostrar que la densidad condicional $f(y|x)$ es Poisson.

De nuevo obtenemos como primer paso la marginal $f(x)$

$$\begin{aligned} f(x) = & \sum_{y=0}^{\infty} \frac{x^{a+y-1} e^{-(1+b)x} b^a}{y!\Gamma(a)} \\ = & \frac{e^{-(1+b)x} b^a}{\Gamma(a)} \sum_{y=0}^{\infty} \frac{x^{a+y-1}}{y!} \\ = & \frac{e^{-(1+b)x} b^a}{\Gamma(a)} \sum_{y=0}^{\infty} \frac{x^y x^{a-1}}{y!} \\ = & \frac{e^{-(1+b)x} b^a x^{a-1}}{\Gamma(a)} \sum_{y=0}^{\infty} \frac{x^y}{y!} \\ = & \frac{e^{-(1+b)x} b^a x^{a-1}}{\Gamma(a)} e^x \end{aligned}$$

Y usamos la definición de densidad condicional:

$$\begin{aligned} f(x|y) = & \frac{f(x,y)}{f(y)} \\ = & \frac{\frac{x^{a+y-1} e^{-(1+b)x} b^a}{y!\Gamma(a)}}{\frac{e^{-(1+b)x} b^a x^{a-1}}{\Gamma(a)} e^x} \\ = & \frac{x^y e^x}{y!} \end{aligned}$$

Con lo cual vemos que es $\mathcal{P}(\lambda = x, k = y)$

- Escriban una función para implementar el Gibbs sampler cuando las constantes son dadas con valores $a = 1$ y $b = 1$.

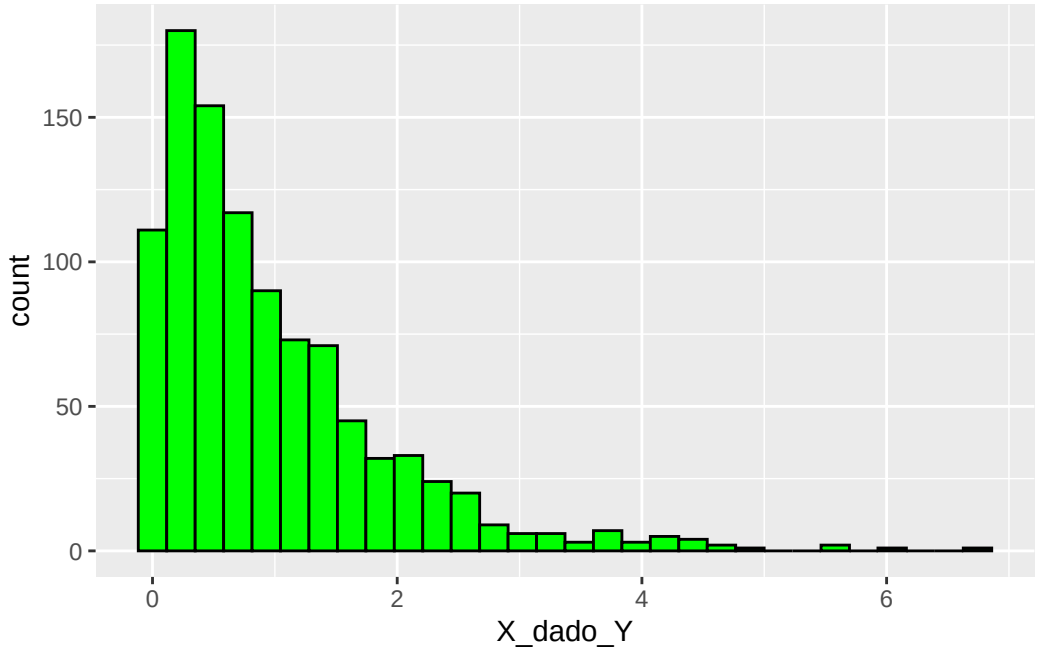
```r
a <- 1
b <- 1
nsim <- 1000

X_dado_Y <- Y_dado_X <- array(0, dim = c(nsim, 1))

X_dado_Y[1] <- rgamma(1, shape = a, rate = 1 + b)
Y_dado_X[1] <- rpois(1, lambda = X_dado_Y[1])
```

```r
ejercicio_4 <- function(n, X_dado_Y, Y_dado_X) {
  for (i in 2:n) {
    X_dado_Y[i] <- rgamma(1, shape = a + Y_dado_X[i - 1], rate = 1 + b)
    Y_dado_X[i] <- rpois(1, lambda = X_dado_Y[i])
  }

  return(list(X_dado_Y = X_dado_Y, Y_dado_X = Y_dado_X))
}
```

- Con su función, escriban 1000 ciclos del Gibbs sampler y de la salida, hacer los histogramas y estimar $E(Y)$.

```r
result <- ejercicio_4(nsim, X_dado_Y, Y_dado_X)
# X_dado_Y <- result$X_dado_Y
# Y_dado_X <- result$Y_dado_X

result<- data.frame(result)

ggplot(result, aes(x=X_dado_Y)) +
  geom_histogram(color="black", fill="green")
```

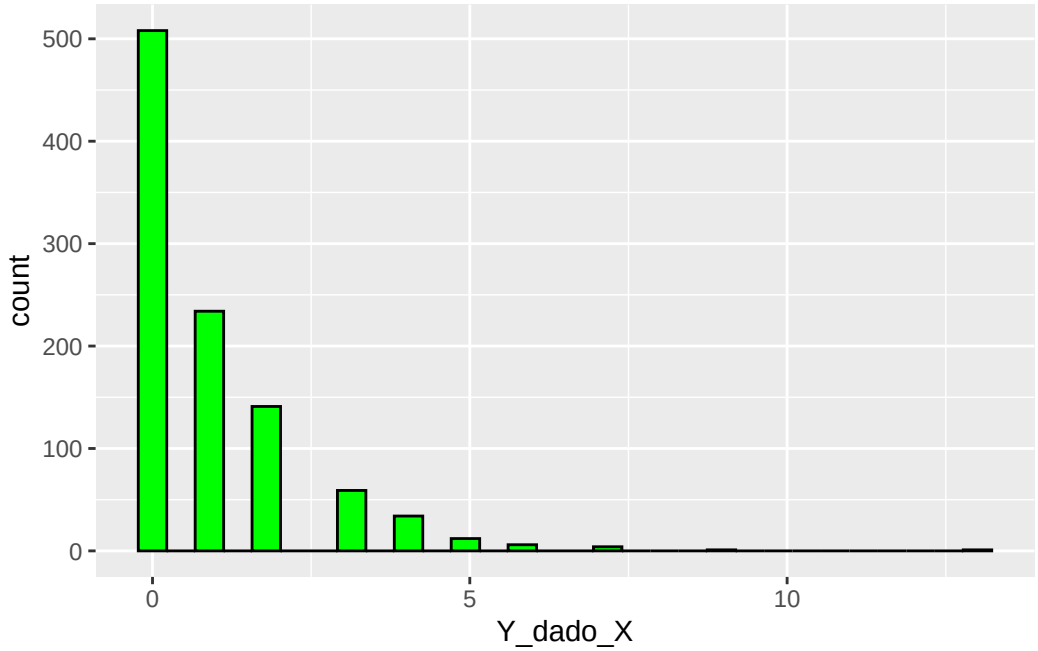
```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



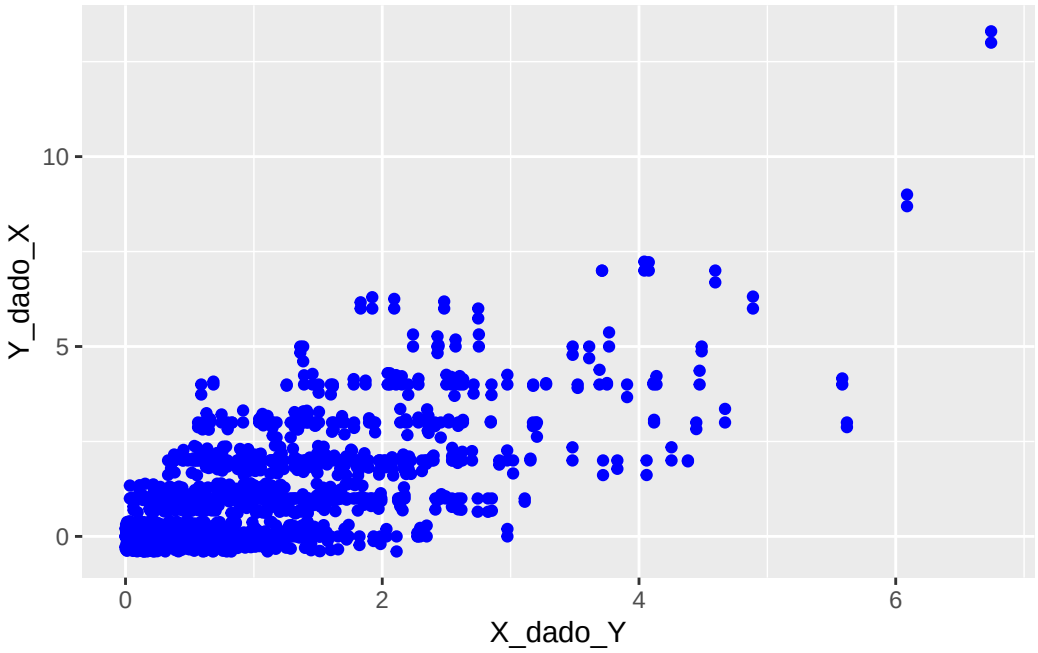

```
#hist(Y_dado_X, probability = T, main = "Y", ylab = "densidad")
```

```
ggplot(result, aes(x=Y_dado_X)) +
  geom_histogram(color="black", fill="green")
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
ggplot(result, aes(X_dado_Y, Y_dado_X))+ geom_point(color = 'blue')+ geom_jitter(color =
```



5. Supongan que se tiene una matriz de 4×4 de variables aleatorias Bernoulli, y la denotamos por $[X_{ij}]$, y sea $N(X)$ el número total de éxitos en X (la suma de X) y $D(X)$ es el total de vecinos de dichos éxitos (horizontales o verticales) que difieren. Por ejemplo,

X	$N(X)$	$D(X)$
$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$	1	2
$\begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$	3	5

Noten que si se escriben los elementos de X en un vector V , entonces existe una matriz M de 24×16 tal que $D(X)$ es la suma de los valores absolutos de MV . El 24 surge porque hay 24 pares de vecinos a revisar.

```
#              1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16
M <- matrix(c(-1,  1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  #1
              0, -1,  1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  #2
              0,  0, -1,  1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  #3
              0,  0,  0,  0, -1,  1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  #4
              0,  0,  0,  0,  0, -1,  1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  #5
              0,  0,  0,  0,  0,  0, -1,  1,  0,  0,  0,  0,  0,  0,  0,  0,  #6
              0,  0,  0,  0,  0,  0,  0,  0, -1,  1,  0,  0,  0,  0,  0,  0,  #7
              0,  0,  0,  0,  0,  0,  0,  0,  0, -1,  1,  0,  0,  0,  0,  0,  #8
              0,  0,  0,  0,  0,  0,  0,  0,  0,  0, -1,  1,  0,  0,  0,  0,  #9
              0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, -1,  1,  0,  0,  #10
              0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, -1,  1,  0,  #11
              0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, -1,  1,  #12
             -1,  0,  0,  0,  1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  #13
              0,  0,  0,  0, -1,  0,  0,  0,  1,  0,  0,  0,  0,  0,  0,  0,  #14
              0,  0,  0,  0,  0,  0,  0,  0, -1,  0,  0,  0,  1,  0,  0,  0,  #15
              0, -1,  0,  0,  0,  1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  #16
              0,  0,  0,  0,  0, -1,  0,  0,  0,  1,  0,  0,  0,  0,  0,  0,  #17
```

```
                0,  0,  0,  0,  0,  0,  0,  0,  0, -1,  0,  0,  0,  1,  0,  0,  #18
                0,  0, -1,  0,  0,  0,  1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  #19
                0,  0,  0,  0,  0,  0, -1,  0,  0,  0,  1,  0,  0,  0,  0,  0,  #20
                0,  0,  0,  0,  0,  0,  0,  0,  0,  0, -1,  0,  0,  0,  1,  0,  #21
                0,  0,  0, -1,  0,  0,  0,  1,  0,  0,  0,  0,  0,  0,  0,  0,  #22
                0,  0,  0,  0,  0,  0,  0, -1,  0,  0,  0,  1,  0,  0,  0,  0,  #23
                0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, -1,  0,  0,  0,  1), nrow=24, by
print(M)
```

```
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
 [1,]   -1    1    0    0    0    0    0    0    0     0     0     0     0
 [2,]    0   -1    1    0    0    0    0    0    0     0     0     0     0
 [3,]    0    0   -1    1    0    0    0    0    0     0     0     0     0
 [4,]    0    0    0    0   -1    1    0    0    0     0     0     0     0
 [5,]    0    0    0    0    0   -1    1    0    0     0     0     0     0
 [6,]    0    0    0    0    0    0   -1    1    0     0     0     0     0
 [7,]    0    0    0    0    0    0    0    0   -1     1     0     0     0
 [8,]    0    0    0    0    0    0    0    0    0    -1     1     0     0
 [9,]    0    0    0    0    0    0    0    0    0     0    -1     1     0
[10,]    0    0    0    0    0    0    0    0    0     0     0     0    -1
[11,]    0    0    0    0    0    0    0    0    0     0     0     0     0
[12,]    0    0    0    0    0    0    0    0    0     0     0     0     0
[13,]   -1    0    0    0    1    0    0    0    0     0     0     0     0
[14,]    0    0    0    0   -1    0    0    0    1     0     0     0     0
[15,]    0    0    0    0    0    0    0    0   -1     0     0     0     1
[16,]    0   -1    0    0    0    1    0    0    0     0     0     0     0
[17,]    0    0    0    0    0   -1    0    0    0     1     0     0     0
[18,]    0    0    0    0    0    0    0    0    0    -1     0     0     0
[19,]    0    0   -1    0    0    0    1    0    0     0     0     0     0
[20,]    0    0    0    0    0    0   -1    0    0     0     1     0     0
[21,]    0    0    0    0    0    0    0    0    0     0    -1     0     0
[22,]    0    0    0   -1    0    0    0    1    0     0     0     0     0
[23,]    0    0    0    0    0    0    0   -1    0     0     0     1     0
[24,]    0    0    0    0    0    0    0    0    0     0     0    -1     0
      [,14] [,15] [,16]
 [1,]     0     0     0
 [2,]     0     0     0
 [3,]     0     0     0
 [4,]     0     0     0
 [5,]     0     0     0
 [6,]     0     0     0
 [7,]     0     0     0
```

```
 [8,]     0     0     0
 [9,]     0     0     0
[10,]     1     0     0
[11,]    -1     1     0
[12,]     0    -1     1
[13,]     0     0     0
[14,]     0     0     0
[15,]     0     0     0
[16,]     0     0     0
[17,]     0     0     0
[18,]     1     0     0
[19,]     0     0     0
[20,]     0     0     0
[21,]     0     1     0
[22,]     0     0     0
[23,]     0     0     0
[24,]     0     0     1
```

```
X2 <- c(1,  1,  0,  0,
        0,  1,  0,  0,
        0,  0,  0,  0,
        0,  0,  0,  0)
sum( abs( M %*% X2 ) )
```

```
[1] 5
```

Supongan que $\pi(X)$, la distribución de X, es proporcional a

$$\pi(X) \propto p^{N(X)}(1-p)^{16-N(X)} \exp(-\lambda D(X))$$

```
problema5 <- function(p, X, lambda){
  D <- sum(abs(M %*% X))
  N <- sum(X)
  f <- p^N * (1 - p)^(16 - N) * exp(-lambda * D)
  return(f)
}
```

Si $\lambda = 0$, las variables son iid Bernoulli (p).

Usar el método de Metropolis-Hastings usando los siguientes kerneles de transición. Hay 2^{16} posibles estados, uno por cada posible valor de X.

a) Sea q_1 tal que cada transición es igualmente plausible con probabilidad $1/2^{16}$. Esto es, el siguiente estado candidato para X es simplemente un vector de 16 iid Bernoulli (p).

b) Sea q_2 tal que se elige una de las 16 entradas en X con probabilidad 1/16, y luego se determina el valor de la celda a ser 0 o 1 con probabilidad 0.5 en cada caso. Entonces sólo un elemento de X puede cambiar en cada transición a lo más.

Ambas q's son simétricas, irreducibles y tienen diagonales positivas. La primera se mueve más rápido que la segunda.

Estamos interesados en la probabilidad de que todos los elementos de la diagonal sean 1. Usen las dos q's para estimar la probabilidad de 1's en la diagonal para $\lambda = 0, 1, 3$ y $p = 0.5, 0.8$.

```
lambda_1 <- 0
lambda_2 <- 1
lambda_3 <- 3

p_1 <- 0.5
p_2 <- 0.8
```

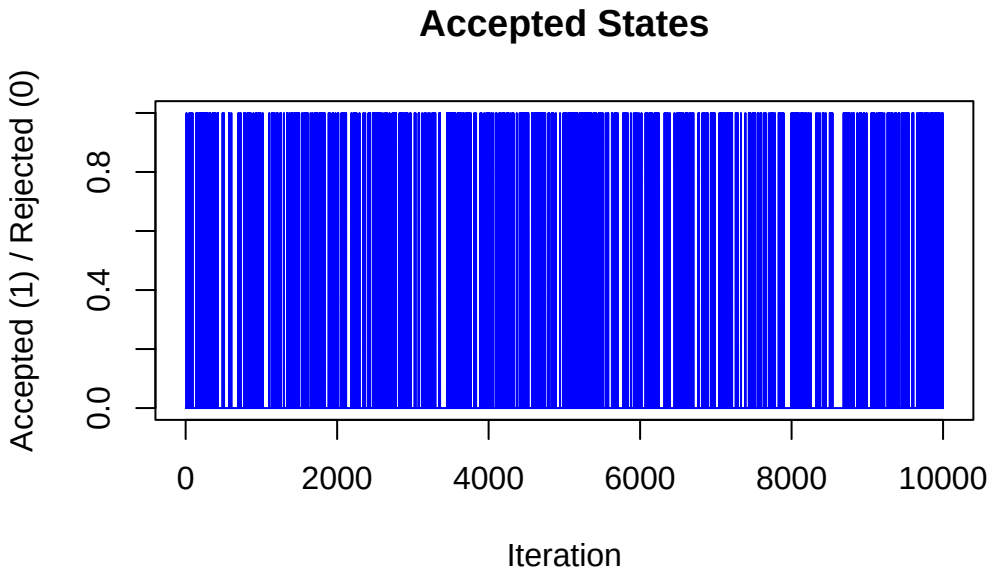
Esto se puede hacer calculando la fracción acumulada de veces que la cadena tiene 1's sobre la diagonal conforme se muestrea de la cadena. Comparar los resultados de las 2 cadenas. Tienen el mismo valor asintótico, pero ¿una cadena llega más rápido que la otra? ¿Alguna cadena muestra más autocorrelación que la otra (por ejemplo, estimados de la probabilidad en 100 pasos sucesivos muestran más correlación para una cadena que para la otra?).

Para a)

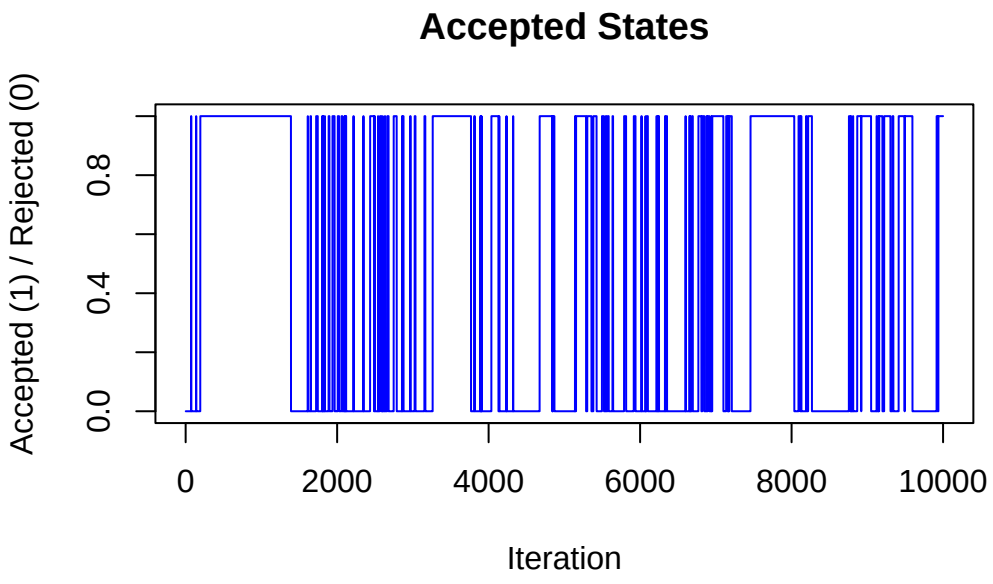
```
metropolis_hastings <- function(initial_vector, iterations, p, lambda) {
  current_vector <- initial_vector
  accepted_states <- numeric(iterations)

  for (iter in 1:iterations) {
    # Calculate current probability without proposing a new sample
    current_probability <- problema5(p, current_vector, lambda)

    # Propose a new random vector
    proposed_vector <- sample(0:1, length(initial_vector), replace = TRUE)

    # Calculate probability for the proposed vector
    proposed_probability <- problema5(p, proposed_vector, lambda)

    # Accept or reject the proposal based on Metropolis-Hastings acceptance ratio
    if (runif(1) < proposed_probability / current_probability) {
```

```
      current_vector <- proposed_vector
    }

    # Save 1 if all specified positions have 1, otherwise save 0
    accepted_states[iter] <- all(current_vector[c(4, 8, 12, 16)] == 1)
  }

  return(accepted_states)
}
```

```
# lambda_1, p_1,
set.seed(123)  # Set seed for reproducibility
initial_vector <- sample(0:1, 16, replace = TRUE)  # Initial random vector
iterations <- 10000
p <- p_1
lambda <- lambda_1

accepted_states <- metropolis_hastings(initial_vector, iterations, p, lambda)

# Print the result (1 if all positions 4, 8, 12, 16 have 1, 0 otherwise)
plot(accepted_states, type = "l", col = "blue", main = "Accepted States",
       xlab = "Iteration", ylab = "Accepted (1) / Rejected (0)")
```

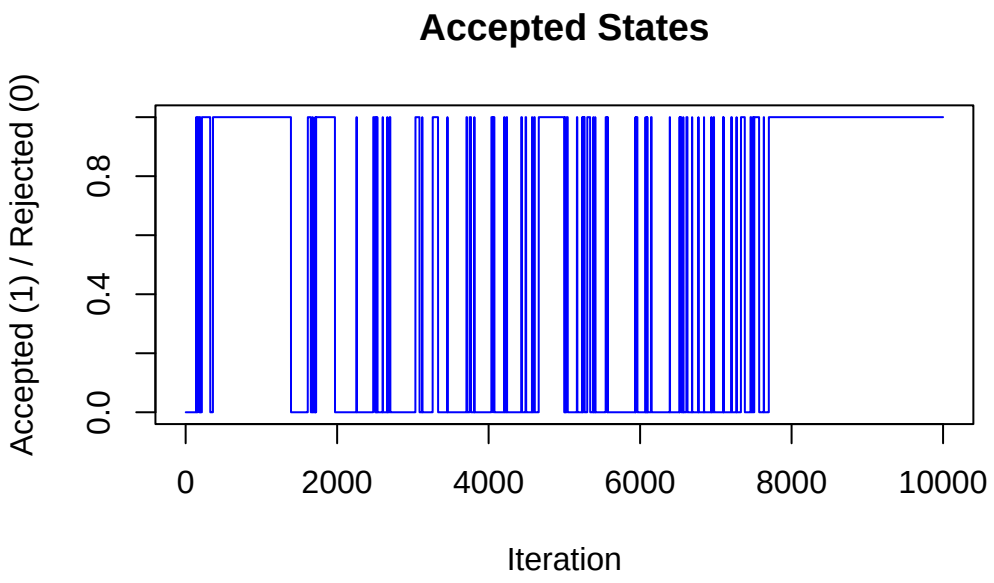


```r
# lambda_1, p_2,
set.seed(123)  # Set seed for reproducibility
initial_vector <- sample(0:1, 16, replace = TRUE)  # Initial random vector
iterations <- 10000
p <- p_2
lambda <- lambda_1

accepted_states <- metropolis_hastings(initial_vector, iterations, p, lambda)

# Print the result (1 if all positions 4, 8, 12, 16 have 1, 0 otherwise)
plot(accepted_states, type = "l", col = "blue", main = "Accepted States",
       xlab = "Iteration", ylab = "Accepted (1) / Rejected (0)")
```

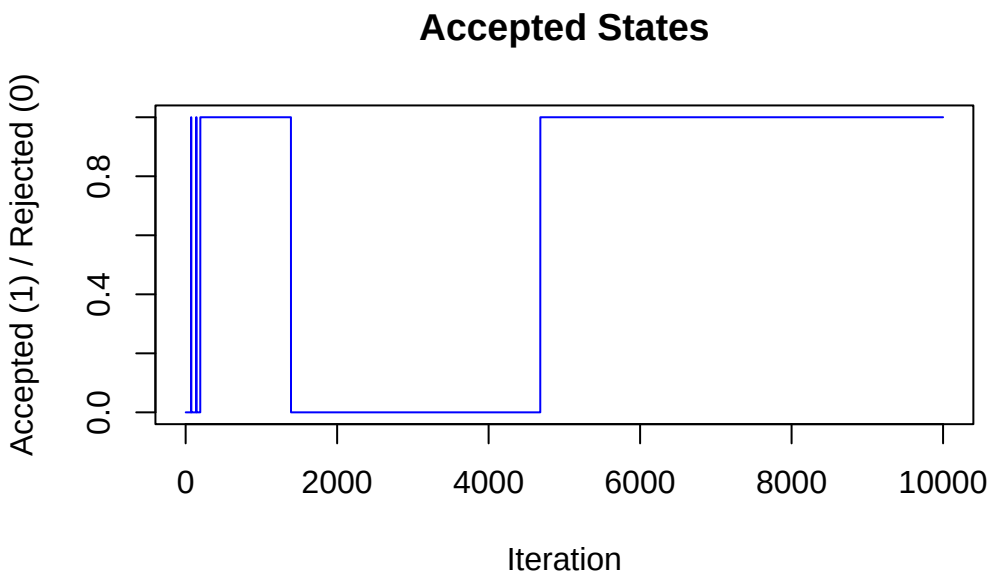



```r
# lambda_2, p_1,
set.seed(123)  # Set seed for reproducibility
initial_vector <- sample(0:1, 16, replace = TRUE)  # Initial random vector
iterations <- 10000
p <- p_1
lambda <- lambda_2

accepted_states <- metropolis_hastings(initial_vector, iterations, p, lambda)
```

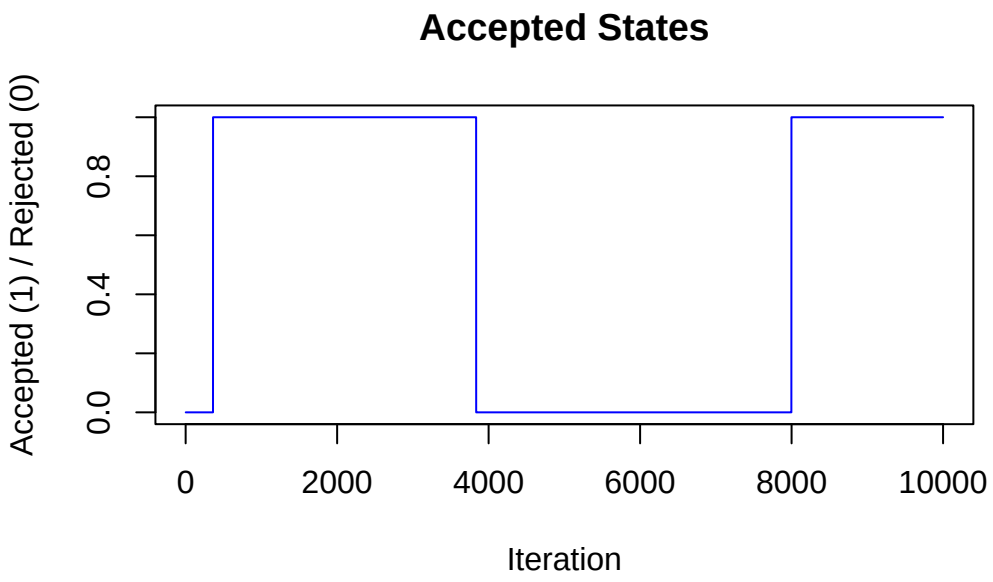
```r
# Print the result (1 if all positions 4, 8, 12, 16 have 1, 0 otherwise)
plot(accepted_states, type = "l", col = "blue", main = "Accepted States",
       xlab = "Iteration", ylab = "Accepted (1) / Rejected (0)")
```



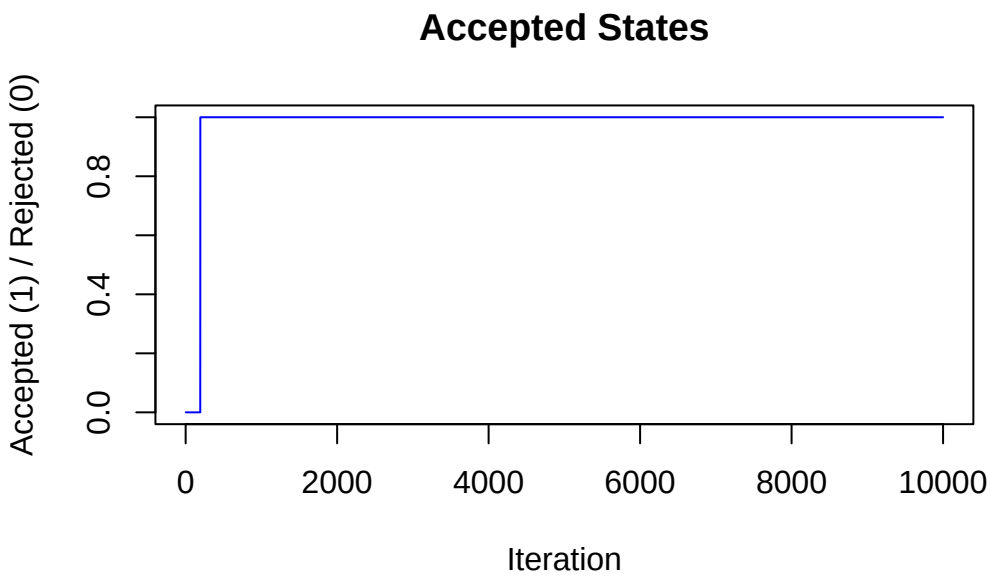

```r
# lambda_1, p_2,
set.seed(123)  # Set seed for reproducibility
initial_vector <- sample(0:1, 16, replace = TRUE)  # Initial random vector
iterations <- 10000
p <- p_2
lambda <- lambda_2

accepted_states <- metropolis_hastings(initial_vector, iterations, p, lambda)

# Print the result (1 if all positions 4, 8, 12, 16 have 1, 0 otherwise)
plot(accepted_states, type = "l", col = "blue", main = "Accepted States",
       xlab = "Iteration", ylab = "Accepted (1) / Rejected (0)")
```




```r
# lambda_3, p_1,
set.seed(123)  # Set seed for reproducibility
initial_vector <- sample(0:1, 16, replace = TRUE)  # Initial random vector
iterations <- 10000
p <- p_1
lambda <- lambda_3

accepted_states <- metropolis_hastings(initial_vector, iterations, p, lambda)

# Print the result (1 if all positions 4, 8, 12, 16 have 1, 0 otherwise)
plot(accepted_states, type = "l", col = "blue", main = "Accepted States",
       xlab = "Iteration", ylab = "Accepted (1) / Rejected (0)")
```



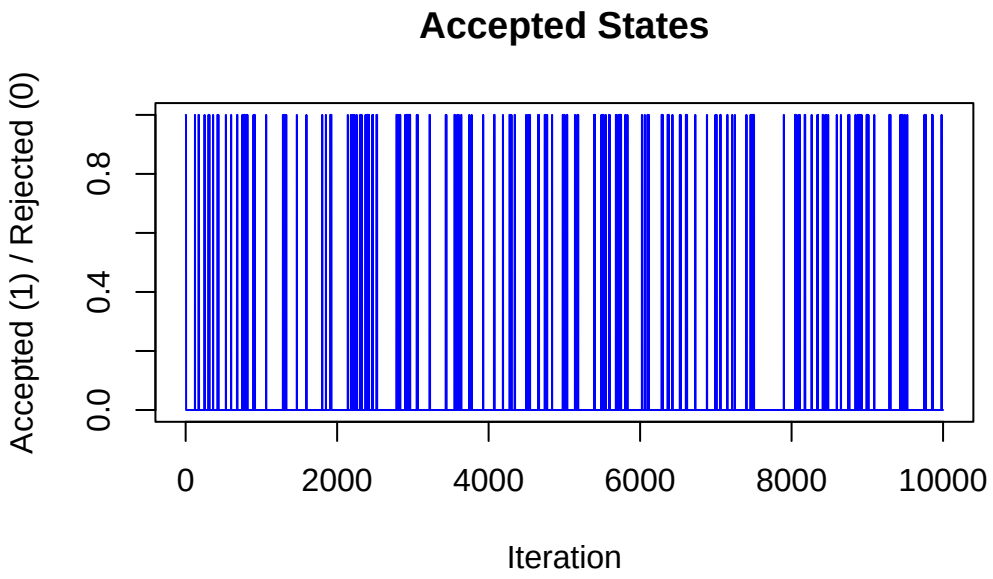

```r
# lambda_1, p_2,
set.seed(123)  # Set seed for reproducibility
initial_vector <- sample(0:1, 16, replace = TRUE)  # Initial random vector
iterations <- 10000
p <- p_2
lambda <- lambda_3

accepted_states <- metropolis_hastings(initial_vector, iterations, p, lambda)

# Print the result (1 if all positions 4, 8, 12, 16 have 1, 0 otherwise)
plot(accepted_states, type = "l", col = "blue", main = "Accepted States",
       xlab = "Iteration", ylab = "Accepted (1) / Rejected (0)")
```



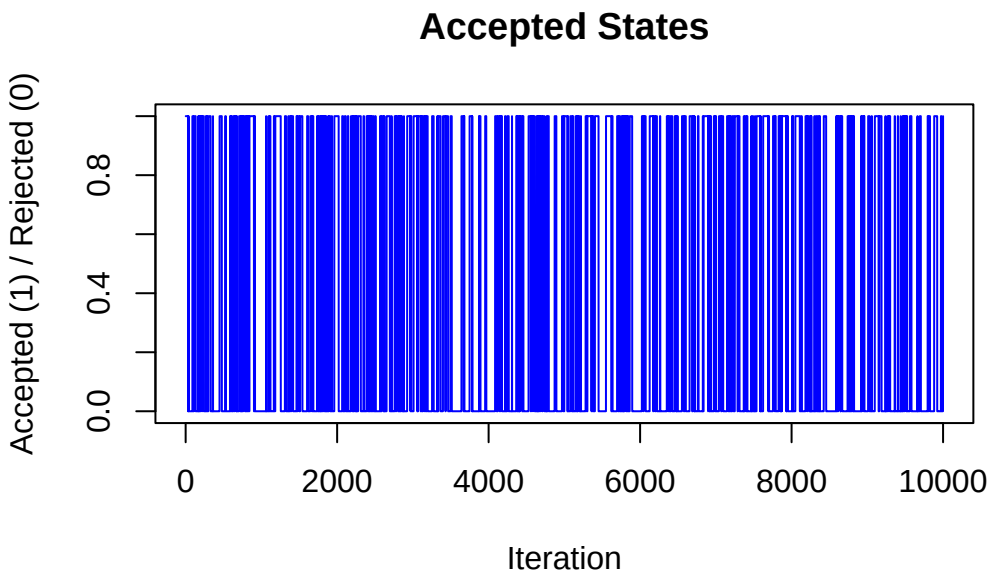

```r
metropolis_hastings2 <- function(initial_vector, iterations, p, lambda) {
  current_vector <- initial_vector
  accepted_states <- numeric(iterations)

  for (iter in 1:iterations) {
    # Calculate current probability without proposing a new sample
    current_probability <- problema5(p, current_vector, lambda)

    # Propose a new vector by changing one element randomly
    proposed_vector <- current_vector
    random_position <- sample(1:length(initial_vector), 1)
    proposed_vector[random_position] <- 1 - proposed_vector[random_position]

    # Calculate probability for the proposed vector
    proposed_probability <- problema5(p, proposed_vector, lambda)

    # Accept or reject the proposal based on Metropolis-Hastings acceptance ratio
    if (runif(1) < proposed_probability / current_probability) {
      current_vector <- proposed_vector
    }
```

```
    # Save 1 if all specified positions have 1, otherwise save 0
    accepted_states[iter] <- all(current_vector[c(4, 8, 12, 16)] == 1)
  }

  return(accepted_states)
}
```

```
# lambda_1, p_1,
set.seed(123)  # Set seed for reproducibility
initial_vector <- sample(0:1, 16, replace = TRUE)  # Initial random vector
iterations <- 10000
p <- p_1
lambda <- lambda_1

accepted_states <- metropolis_hastings2(initial_vector, iterations, p, lambda)

# Print the result (1 if all positions 4, 8, 12, 16 have 1, 0 otherwise)
plot(accepted_states, type = "l", col = "blue", main = "Accepted States",
       xlab = "Iteration", ylab = "Accepted (1) / Rejected (0)")
```

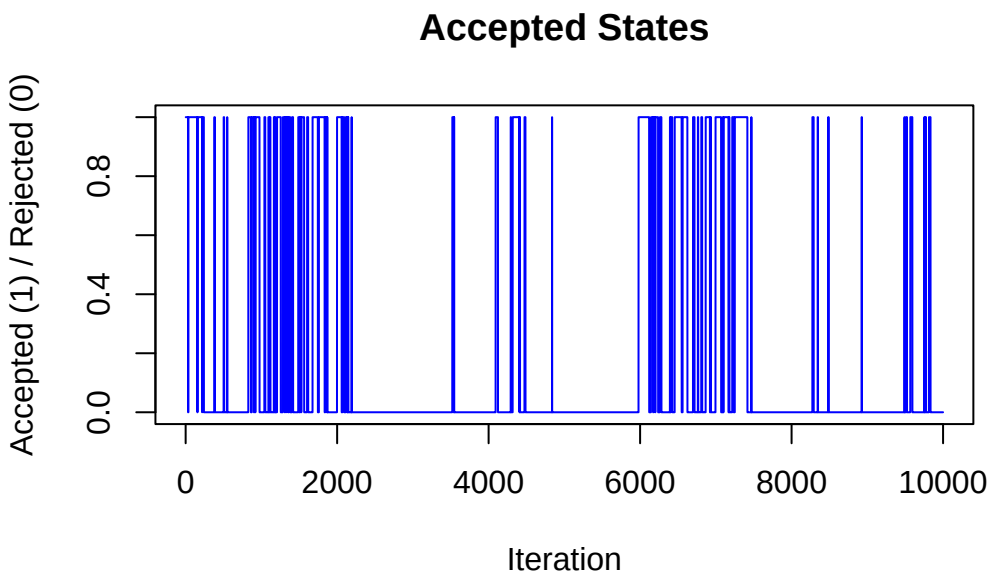


```r
# lambda_1, p_2,
set.seed(123)  # Set seed for reproducibility
initial_vector <- sample(0:1, 16, replace = TRUE)  # Initial random vector
iterations <- 10000
p <- p_2
lambda <- lambda_1

accepted_states <- metropolis_hastings2(initial_vector, iterations, p, lambda)

# Print the result (1 if all positions 4, 8, 12, 16 have 1, 0 otherwise)
plot(accepted_states, type = "l", col = "blue", main = "Accepted States",
       xlab = "Iteration", ylab = "Accepted (1) / Rejected (0)")
```

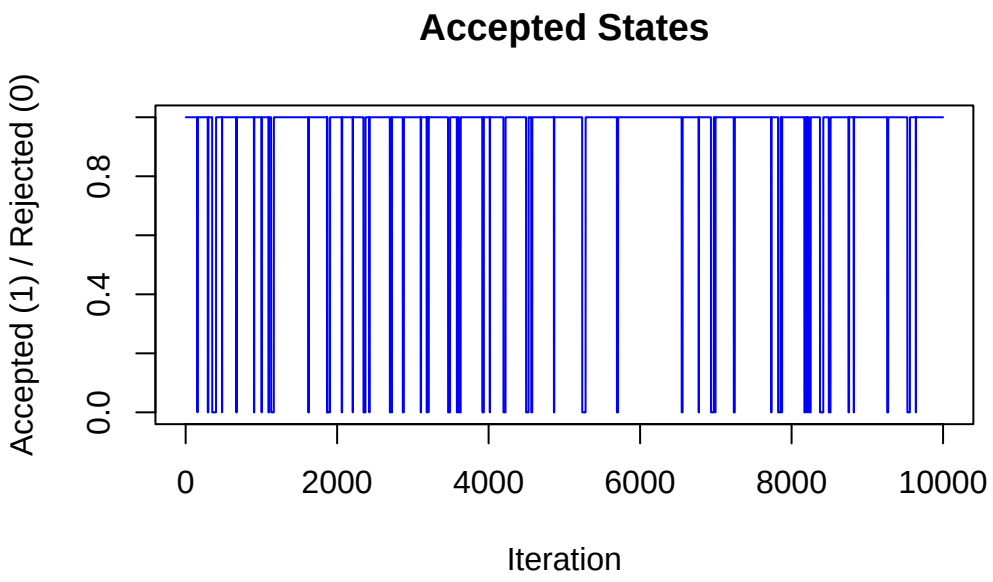



```r
# lambda_2, p_1,
set.seed(123)  # Set seed for reproducibility
initial_vector <- sample(0:1, 16, replace = TRUE)  # Initial random vector
iterations <- 10000
p <- p_1
lambda <- lambda_2

accepted_states <- metropolis_hastings2(initial_vector, iterations, p, lambda)
```

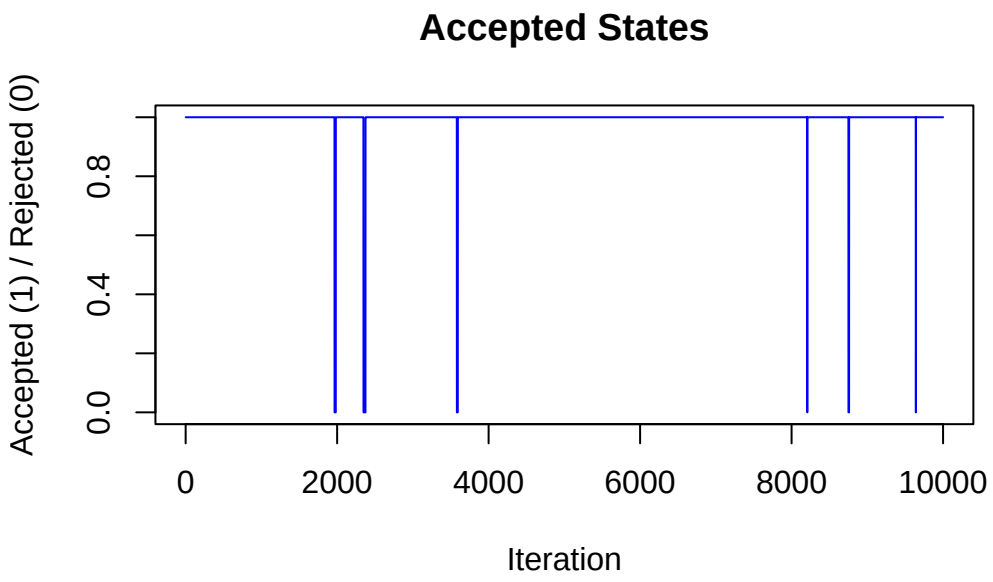
```r
# Print the result (1 if all positions 4, 8, 12, 16 have 1, 0 otherwise)
plot(accepted_states, type = "l", col = "blue", main = "Accepted States",
       xlab = "Iteration", ylab = "Accepted (1) / Rejected (0)")
```



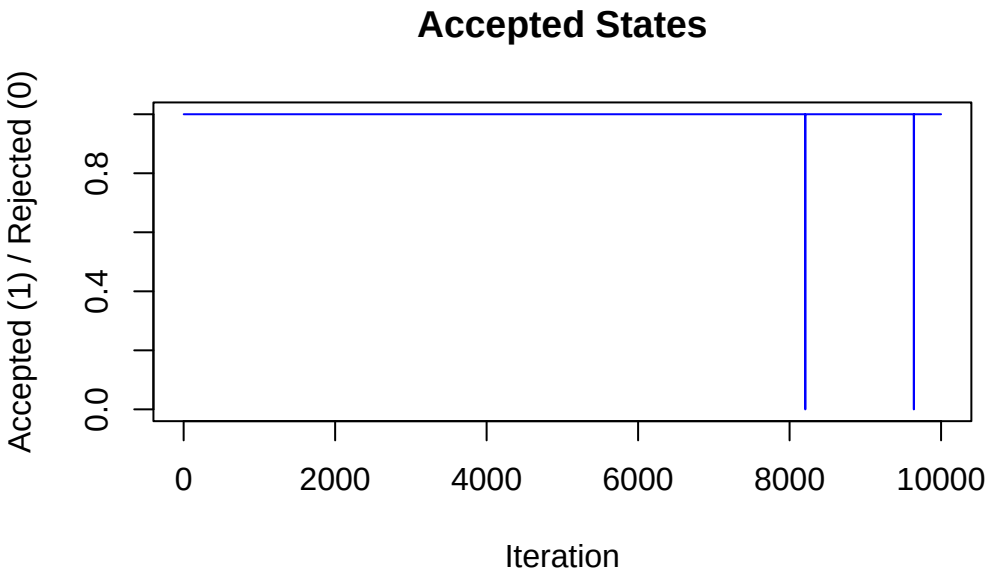

```r
# lambda_1, p_2,
set.seed(123)  # Set seed for reproducibility
initial_vector <- sample(0:1, 16, replace = TRUE)  # Initial random vector
iterations <- 10000
p <- p_2
lambda <- lambda_2

accepted_states <- metropolis_hastings2(initial_vector, iterations, p, lambda)

# Print the result (1 if all positions 4, 8, 12, 16 have 1, 0 otherwise)
plot(accepted_states, type = "l", col = "blue", main = "Accepted States",
       xlab = "Iteration", ylab = "Accepted (1) / Rejected (0)")
```




```r
# lambda_3, p_1,
set.seed(123)  # Set seed for reproducibility
initial_vector <- sample(0:1, 16, replace = TRUE)  # Initial random vector
iterations <- 10000
p <- p_1
lambda <- lambda_3

accepted_states <- metropolis_hastings2(initial_vector, iterations, p, lambda)

# Print the result (1 if all positions 4, 8, 12, 16 have 1, 0 otherwise)
plot(accepted_states, type = "l", col = "blue", main = "Accepted States",
       xlab = "Iteration", ylab = "Accepted (1) / Rejected (0)")
```




```r
# lambda_1, p_2,
set.seed(123)  # Set seed for reproducibility
initial_vector <- sample(0:1, 16, replace = TRUE)  # Initial random vector
iterations <- 10000
p <- p_2
lambda <- lambda_3

accepted_states <- metropolis_hastings2(initial_vector, iterations, p, lambda)

# Print the result (1 if all positions 4, 8, 12, 16 have 1, 0 otherwise)
plot(accepted_states, type = "l", col = "blue", main = "Accepted States",
       xlab = "Iteration", ylab = "Accepted (1) / Rejected (0)")
```



En este problema, también determinen cuántas simulaciones hay que hacer, para desechar el periodo de calentamiento.

a)
- λ_1, p_1, no converge
- λ_1, p_2, no converge
- λ_2, p_1, 8000
- λ_2, p_2, 5000
- λ_3, p_1, no converge
- λ_3, p_2, 500

b)
- λ_1, p_1, no converge
- λ_1, p_2, no converge
- λ_2, p_1, no converge
- λ_2, p_2, no converge
- λ_3, p_1, no converge
- λ_3, p_2, no converge

6. Considera los siguientes números:

$$0.4, 0.01, 0.2, 0.1, 2.1, 0.1, 0.9, 2.4, 0.1, 0.2$$

Usen la distribución exponencial $Y_i \sim \exp(\theta)$ para modelar estos datos y asignen una inicial sobre $\log \theta$.

a) Definan los datos en WinBUGS (u OpenBUGS). Usen $\theta = 1$ como valor inicial.
b) Escriban el código para el modelo.
c) Compilen el modelo y obtengan una muestra de 1000 iteraciones después de descartar las 500 iteraciones iniciales como burnin.
d) Monitoreen la convergencia gráficamente usando gráficas 'trace' y de autocorrelación.
e) Obtengan estadísticas sumarias posteriores y densidades para θ, $1/\theta$ y $\log \theta$
f) Con los datos el primer inciso, ahora usen las distribuciones (i) gamma (ii) log-normal para modelar los datos, así como (iii) normal para los logaritmos de los valores originales. En todos los modelos hagan un ejercicio similar al de los numerales previos, y comparen los resultados obtenidos bajo todos los modelos. Hagan todos los supuestos que tengan que hacer.