Tarea4

Alberto, Ivan, Sara, Valeria

Tarea 4

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

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

```
library(rstan)
```

```
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
```

```
library(V8)
```

```
Using V8 engine 9.6.180.12
```

```
#| Dosis | # muertos | # expuestos |
#|-------|-----------|-------------|
#| 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 $pi \equiv P(muerte|wi)$, pero comparan tres funciones ligas diferentes:

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

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

complementario log-log: $p_i = 1 - \exp\left[-\exp(\alpha + \beta z_i)\right]$

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:

LOGIT

```
dosis <- c(1.6907, 1.7242, 1.7552, 1.7842, 1.8113, 1.8369, 1.8610, 1.8839)
muertos <- c(6, 13, 18, 28, 52, 53, 61, 60)
expuestos <- c(59, 60, 62, 56, 63, 59, 62, 60)

# Prepara los datos en un formato adecuado para Stan
datos_stan <- list(
  k = length(muertos),
  y = muertos,
  n = expuestos,
  w = dosis
```

```r
)

setwd("./tarea4_files/")

fit_logit <- stan(file = 'logit1.stan', data = datos_stan, iter = 2000, chains = 4)
```

```
Trying to compile a simple C file

Running /usr/lib/R/bin/R CMD SHLIB foo.c
using C compiler: ‘gcc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0’
gcc -I"/usr/share/R/include" -DNDEBUG   -I"/home/saraluz/.cache/R/renv/cache/v5/R-4.3/x86_64-
In file included from /home/saraluz/Documents/2Sem/MLG/Tareas/renv/library/R-4.3/x86_64-pc-li
                 from /home/saraluz/Documents/2Sem/MLG/Tareas/renv/library/R-4.3/x86_64-pc-li
                 from /home/saraluz/.cache/R/renv/cache/v5/R-4.3/x86_64-pc-linux-gnu/StanHead
                 from <command-line>:
/home/saraluz/Documents/2Sem/MLG/Tareas/renv/library/R-4.3/x86_64-pc-linux-gnu/RcppEigen/incl
  628 | namespace Eigen {
      | ^~~~~~~~~
/home/saraluz/Documents/2Sem/MLG/Tareas/renv/library/R-4.3/x86_64-pc-linux-gnu/RcppEigen/incl
  628 | namespace Eigen {
      |                 ^
In file included from /home/saraluz/Documents/2Sem/MLG/Tareas/renv/library/R-4.3/x86_64-pc-li
                 from /home/saraluz/.cache/R/renv/cache/v5/R-4.3/x86_64-pc-linux-gnu/StanHead
                 from <command-line>:
/home/saraluz/Documents/2Sem/MLG/Tareas/renv/library/R-4.3/x86_64-pc-linux-gnu/RcppEigen/incl
   96 | #include <complex>
      |          ^~~~~~~~~
compilation terminated.
make: *** [/usr/lib/R/etc/Makeconf:191: foo.o] Error 1

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
Chain 1:
Chain 1: Gradient evaluation took 8e-06 seconds
Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.08 seconds.
Chain 1: Adjust your expectations accordingly!
Chain 1:
Chain 1:
Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
```

```
Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0.019 seconds (Warm-up)
Chain 1:                0.017 seconds (Sampling)
Chain 1:                0.036 seconds (Total)
Chain 1:

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
Chain 2:
Chain 2: Gradient evaluation took 4e-06 seconds
Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.04 seconds.
Chain 2: Adjust your expectations accordingly!
Chain 2:
Chain 2:
Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 2:
Chain 2:  Elapsed Time: 0.021 seconds (Warm-up)
Chain 2:                0.017 seconds (Sampling)
Chain 2:                0.038 seconds (Total)
Chain 2:

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
Chain 3:
Chain 3: Gradient evaluation took 3e-06 seconds
Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.03 seconds.
```

```
Chain 3: Adjust your expectations accordingly!
Chain 3:
Chain 3:
Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 3:
Chain 3:  Elapsed Time: 0.021 seconds (Warm-up)
Chain 3:                0.018 seconds (Sampling)
Chain 3:                0.039 seconds (Total)
Chain 3:

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 4).
Chain 4:
Chain 4: Gradient evaluation took 3e-06 seconds
Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.03 seconds.
Chain 4: Adjust your expectations accordingly!
Chain 4:
Chain 4:
Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 4:
Chain 4:  Elapsed Time: 0.02 seconds (Warm-up)
Chain 4:                0.017 seconds (Sampling)
```

```
Chain 4:                0.037 seconds (Total)
Chain 4:
```

```r
print(fit_logit)
```

```
Inference for Stan model: anon_model.
4 chains, each with iter=2000; warmup=1000; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=4000.

         mean se_mean   sd    2.5%     25%     50%     75%   97.5% n_eff Rhat
alpha    0.75    0.00 0.13    0.49    0.65    0.75    0.84    1.02  2480    1
beta    34.61    0.06 2.89   29.21   32.60   34.49   36.51   40.56  2374    1
p[1]     0.06    0.00 0.02    0.03    0.05    0.06    0.07    0.09  3089    1
p[2]     0.16    0.00 0.03    0.11    0.14    0.16    0.18    0.22  3308    1
p[3]     0.36    0.00 0.03    0.30    0.34    0.36    0.38    0.42  3597    1
p[4]     0.60    0.00 0.03    0.54    0.58    0.61    0.63    0.66  2771    1
p[5]     0.80    0.00 0.03    0.74    0.78    0.80    0.81    0.84  2222    1
p[6]     0.90    0.00 0.02    0.86    0.89    0.90    0.92    0.94  2132    1
p[7]     0.95    0.00 0.01    0.93    0.95    0.96    0.96    0.97  2145    1
p[8]     0.98    0.00 0.01    0.96    0.97    0.98    0.98    0.99  2166    1
lp__  -187.19    0.02 0.96 -189.81 -187.56 -186.90 -186.52 -186.26  1512    1

Samples were drawn using NUTS(diag_e) at Fri Mar  8 20:02:13 2024.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

En el caso con una función liga logit, la media posterior de α se ubica en 0.75 con una desviación estándar de 0.14. Por su parte, la media posterior de β se ubica en 34.59 con una desviación estándar de 3.01. Este parámetro nos idica que existe una relación positiva entre la cantidad utilizada de pesticidad y la mortalidad de los escarabajos. En promedio el modelo tiene una log_p de -187.28, donde un valor más alto indica un mejor ajuste. También destaca al observar los valores de Rhat que las cadenas han convergido.

PROBIT

```r
dosis <- c(1.6907, 1.7242, 1.7552, 1.7842, 1.8113, 1.8369, 1.8610, 1.8839)
muertos <- c(6, 13, 18, 28, 52, 53, 61, 60)
expuestos <- c(59, 60, 62, 56, 63, 59, 62, 60)

# Prepara los datos en un formato adecuado para Stan
```

```r
datos_stan <- list(
  k = length(muertos),
  y = muertos,
  n = expuestos,
  w = dosis
)

setwd("./tarea4_files/")

fit_probit <- stan(file = 'probit1.stan', data = datos_stan, iter = 2000, chains = 4)
```

```
Trying to compile a simple C file


Running /usr/lib/R/bin/R CMD SHLIB foo.c
using C compiler: ‘gcc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0’
gcc -I"/usr/share/R/include" -DNDEBUG   -I"/home/saraluz/.cache/R/renv/cache/v5/R-4.3/x86_64-
In file included from /home/saraluz/Documents/2Sem/MLG/Tareas/renv/library/R-4.3/x86_64-pc-li
                 from /home/saraluz/Documents/2Sem/MLG/Tareas/renv/library/R-4.3/x86_64-pc-li
                 from /home/saraluz/.cache/R/renv/cache/v5/R-4.3/x86_64-pc-linux-gnu/StanHead
                 from <command-line>:
/home/saraluz/Documents/2Sem/MLG/Tareas/renv/library/R-4.3/x86_64-pc-linux-gnu/RcppEigen/incl
  628 | namespace Eigen {
      | ^~~~~~~~~
/home/saraluz/Documents/2Sem/MLG/Tareas/renv/library/R-4.3/x86_64-pc-linux-gnu/RcppEigen/incl
  628 | namespace Eigen {
      |                 ^
In file included from /home/saraluz/Documents/2Sem/MLG/Tareas/renv/library/R-4.3/x86_64-pc-li
                 from /home/saraluz/.cache/R/renv/cache/v5/R-4.3/x86_64-pc-linux-gnu/StanHead
                 from <command-line>:
/home/saraluz/Documents/2Sem/MLG/Tareas/renv/library/R-4.3/x86_64-pc-linux-gnu/RcppEigen/incl
   96 | #include <complex>
      |          ^~~~~~~~~
compilation terminated.
make: *** [/usr/lib/R/etc/Makeconf:191: foo.o] Error 1

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
Chain 1:
Chain 1: Gradient evaluation took 7e-06 seconds
Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.07 seconds.
Chain 1: Adjust your expectations accordingly!
Chain 1:
```

```
Chain 1:
Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0.023 seconds (Warm-up)
Chain 1:                0.019 seconds (Sampling)
Chain 1:                0.042 seconds (Total)
Chain 1:

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
Chain 2:
Chain 2: Gradient evaluation took 4e-06 seconds
Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.04 seconds.
Chain 2: Adjust your expectations accordingly!
Chain 2:
Chain 2:
Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 2:
Chain 2:  Elapsed Time: 0.021 seconds (Warm-up)
Chain 2:                0.017 seconds (Sampling)
Chain 2:                0.038 seconds (Total)
Chain 2:
```

```
SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
Chain 3:
Chain 3: Gradient evaluation took 3e-06 seconds
Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.03 seconds.
Chain 3: Adjust your expectations accordingly!
Chain 3:
Chain 3:
Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 3:
Chain 3:  Elapsed Time: 0.022 seconds (Warm-up)
Chain 3:                0.019 seconds (Sampling)
Chain 3:                0.041 seconds (Total)
Chain 3:

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 4).
Chain 4:
Chain 4: Gradient evaluation took 3e-06 seconds
Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.03 seconds.
Chain 4: Adjust your expectations accordingly!
Chain 4:
Chain 4:
Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
```

```
Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 4:
Chain 4:  Elapsed Time: 0.024 seconds (Warm-up)
Chain 4:                0.021 seconds (Sampling)
Chain 4:                0.045 seconds (Total)
Chain 4:
```

```
print(fit_probit)
```

```
Inference for Stan model: anon_model.
4 chains, each with iter=2000; warmup=1000; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=4000.

         mean se_mean   sd    2.5%     25%     50%     75%   97.5% n_eff Rhat
alpha    0.45    0.00 0.08    0.29    0.39    0.45    0.50    0.60  2509    1
beta    19.87    0.03 1.51   16.98   18.85   19.85   20.87   22.95  2381    1
p[1]     0.06    0.00 0.02    0.03    0.05    0.06    0.07    0.09  3350    1
p[2]     0.18    0.00 0.03    0.12    0.16    0.18    0.20    0.24  3799    1
p[3]     0.38    0.00 0.03    0.32    0.36    0.38    0.40    0.44  3897    1
p[4]     0.60    0.00 0.03    0.55    0.58    0.60    0.62    0.66  2818    1
p[5]     0.79    0.00 0.03    0.74    0.77    0.79    0.81    0.84  2197    1
p[6]     0.90    0.00 0.02    0.86    0.89    0.90    0.92    0.94  2068    1
p[7]     0.96    0.00 0.01    0.93    0.95    0.96    0.97    0.98  2073    1
p[8]     0.99    0.00 0.01    0.97    0.98    0.99    0.99    1.00  2109    1
lp__  -186.69    0.03 1.02 -189.52 -187.09 -186.38 -185.97 -185.70  1481    1

Samples were drawn using NUTS(diag_e) at Fri Mar  8 20:03:21 2024.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

En el caso con una función liga probit, la media posterior de α se ubica en 0.45 con una desviación estándar de 0.08. Por su parte, la media posterior de β se ubica en 19.84 con una desviación estándar de 1.05. Este parámetro nos idica que existe una relación positiva entre la cantidad utilizada de pesticidad y la mortalidad de los escarabajos. Es importante enfatizar que ambos parámetros registran una media y una desviación estándar inferior al obtenido con la función logit. En promedio el modelo tiene una log_p de -186.67, que es más alto que el obtenido con el modelo logit, lo que indica un mejor ajuste. También destaca al observar los valores de Rhat que las cadenas han convergido.

C LOG-LOG

```r
dosis <- c(1.6907, 1.7242, 1.7552, 1.7842, 1.8113, 1.8369, 1.8610, 1.8839)
muertos <- c(6, 13, 18, 28, 52, 53, 61, 60)
expuestos <- c(59, 60, 62, 56, 63, 59, 62, 60)

# Prepara los datos en un formato adecuado para Stan
datos_stan <- list(
  k = length(muertos),
  y = muertos,
  n = expuestos,
  w = dosis
)

setwd("./tarea4_files/")

fit_clog <- stan(file = 'cloglog.stan', data = datos_stan, iter = 2000, chains = 4)
```

```
Trying to compile a simple C file


Running /usr/lib/R/bin/R CMD SHLIB foo.c
using C compiler: ‘gcc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0’
gcc -I"/usr/share/R/include" -DNDEBUG   -I"/home/saraluz/.cache/R/renv/cache/v5/R-4.3/x86_64-
In file included from /home/saraluz/Documents/2Sem/MLG/Tareas/renv/library/R-4.3/x86_64-pc-li
                 from /home/saraluz/Documents/2Sem/MLG/Tareas/renv/library/R-4.3/x86_64-pc-li
                 from /home/saraluz/.cache/R/renv/cache/v5/R-4.3/x86_64-pc-linux-gnu/StanHead
                 from <command-line>:
/home/saraluz/Documents/2Sem/MLG/Tareas/renv/library/R-4.3/x86_64-pc-linux-gnu/RcppEigen/incl
  628 | namespace Eigen {
      | ^~~~~~~~~
/home/saraluz/Documents/2Sem/MLG/Tareas/renv/library/R-4.3/x86_64-pc-linux-gnu/RcppEigen/incl
  628 | namespace Eigen {
      |                 ^
In file included from /home/saraluz/Documents/2Sem/MLG/Tareas/renv/library/R-4.3/x86_64-pc-li
                 from /home/saraluz/.cache/R/renv/cache/v5/R-4.3/x86_64-pc-linux-gnu/StanHead
                 from <command-line>:
/home/saraluz/Documents/2Sem/MLG/Tareas/renv/library/R-4.3/x86_64-pc-linux-gnu/RcppEigen/incl
   96 | #include <complex>
      |          ^~~~~~~~~
compilation terminated.
make: *** [/usr/lib/R/etc/Makeconf:191: foo.o] Error 1

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
```

```
Chain 1:
Chain 1: Gradient evaluation took 9e-06 seconds
Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.09 seconds.
Chain 1: Adjust your expectations accordingly!
Chain 1:
Chain 1:
Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0.029 seconds (Warm-up)
Chain 1:                0.018 seconds (Sampling)
Chain 1:                0.047 seconds (Total)
Chain 1:

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
Chain 2:
Chain 2: Gradient evaluation took 5e-06 seconds
Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.05 seconds.
Chain 2: Adjust your expectations accordingly!
Chain 2:
Chain 2:
Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
```

```
Chain 2:
Chain 2:  Elapsed Time: 0.029 seconds (Warm-up)
Chain 2:                0.022 seconds (Sampling)
Chain 2:                0.051 seconds (Total)
Chain 2:

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
Chain 3:
Chain 3: Gradient evaluation took 5e-06 seconds
Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.05 seconds.
Chain 3: Adjust your expectations accordingly!
Chain 3:
Chain 3:
Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 3:
Chain 3:  Elapsed Time: 0.03 seconds (Warm-up)
Chain 3:                0.02 seconds (Sampling)
Chain 3:                0.05 seconds (Total)
Chain 3:

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 4).
Chain 4:
Chain 4: Gradient evaluation took 5e-06 seconds
Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.05 seconds.
Chain 4: Adjust your expectations accordingly!
Chain 4:
Chain 4:
Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
```

```
Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 4:
Chain 4:  Elapsed Time: 0.027 seconds (Warm-up)
Chain 4:                0.023 seconds (Sampling)
Chain 4:                0.05 seconds (Total)
Chain 4:
```

```
print(fit_clog)
```

```
Inference for Stan model: anon_model.
4 chains, each with iter=2000; warmup=1000; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=4000.

         mean se_mean   sd    2.5%     25%     50%     75%   97.5% n_eff Rhat
alpha   -0.05    0.00 0.08   -0.21   -0.10   -0.05    0.01    0.11  3214    1
beta    22.17    0.03 1.75   18.91   20.97   22.14   23.34   25.67  2931    1
p[1]     0.09    0.00 0.02    0.06    0.08    0.09    0.11    0.13  2883    1
p[2]     0.19    0.00 0.03    0.14    0.17    0.19    0.20    0.24  2860    1
p[3]     0.34    0.00 0.03    0.28    0.32    0.34    0.36    0.40  2928    1
p[4]     0.54    0.00 0.03    0.48    0.52    0.54    0.56    0.60  3104    1
p[5]     0.76    0.00 0.03    0.70    0.74    0.76    0.78    0.81  3509    1
p[6]     0.92    0.00 0.02    0.87    0.90    0.92    0.93    0.95  3766    1
p[7]     0.98    0.00 0.01    0.96    0.98    0.99    0.99    1.00  3642    1
p[8]     1.00    0.00 0.00    0.99    1.00    1.00    1.00    1.00  3155    1
lp__  -183.36    0.02 1.00 -186.02 -183.78 -183.04 -182.63 -182.37  1813    1

Samples were drawn using NUTS(diag_e) at Fri Mar  8 20:04:29 2024.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

En el caso con una función liga c log-log, la media posterior de α se ubica en -0.05 con una desviación estándar de 0.08. Por su parte, la media posterior de β se ubica en 22.14 con una desviación estándar de 1.83. Este parámetro nos idica que existe una relación positiva entre la cantidad utilizada de pesticidad y la mortalidad de los escarabajos. Es importante enfatizar

que ambos parámetros registran una media y una desviación estándar distinta a las obtenidas con los modelos anteriores. En promedio el modelo tiene una log_p de -183.36, que es la más alta de los 3 modelos, lo que indica el mejor ajuste. También destaca al observar los valores de Rhat que las cadenas han convergido.

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

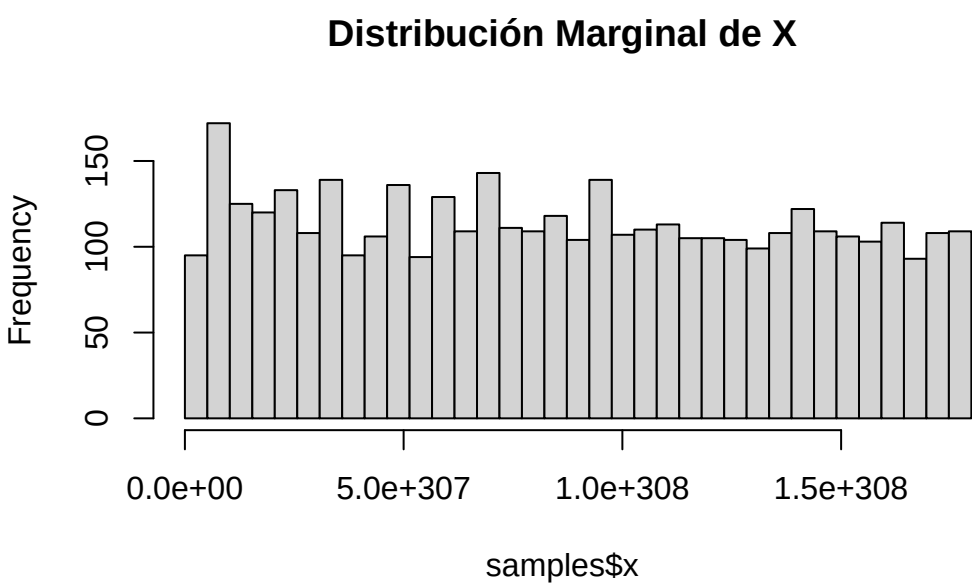
El algoritmo de Gibbs sampler quedaría de la siguiente manera:

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.

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

```
# 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.

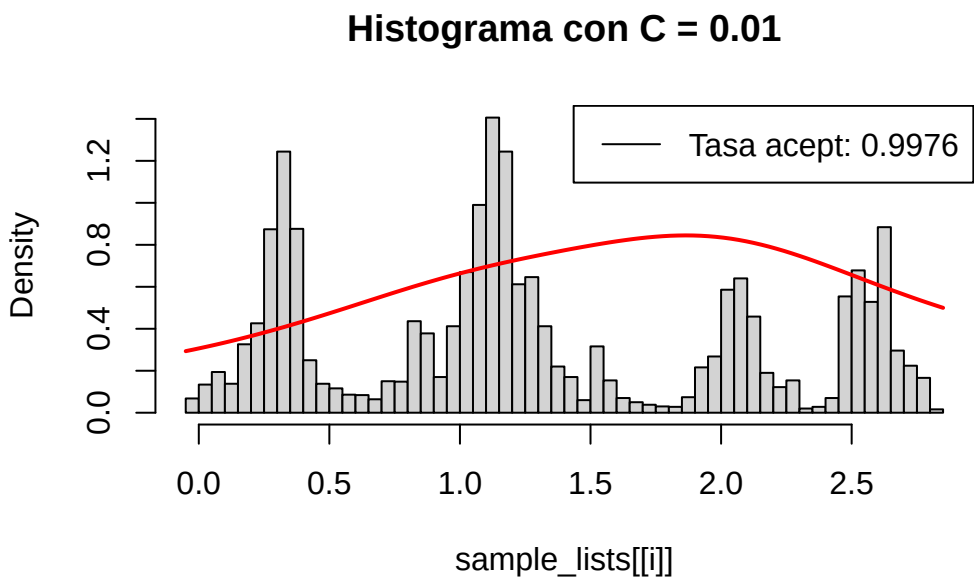
```
log_densidad_poli_cauchy <- function(y, a) {
  n <- length(a)
  log_density <- 0
  for (i in 1:n) {
    log_density <- log_density - log(pi * (1 + (y - a[i])^2))
  }
  return(log_density)
}
```

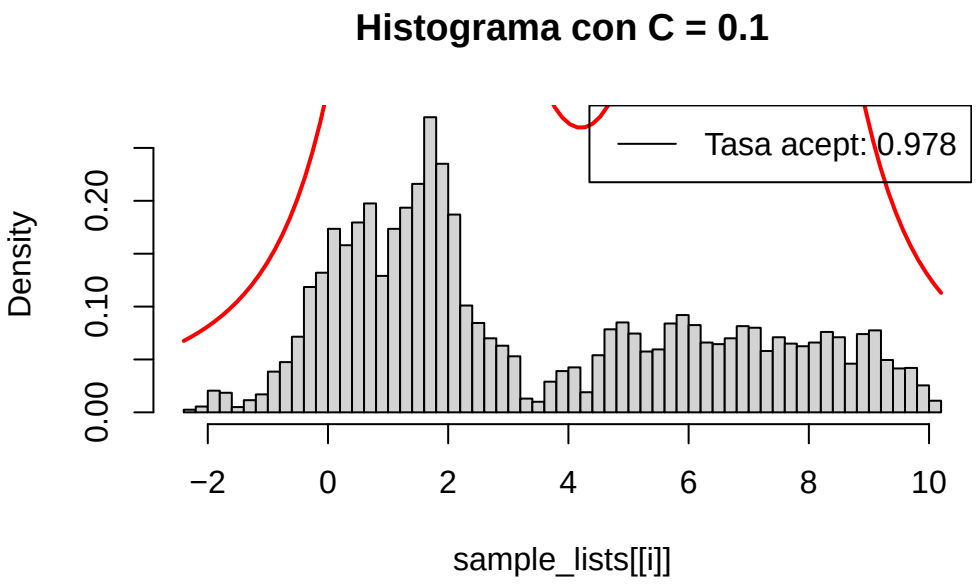
```r
# Parámetros dados
a <- c(1, 2, 2, 6, 7, 8)

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

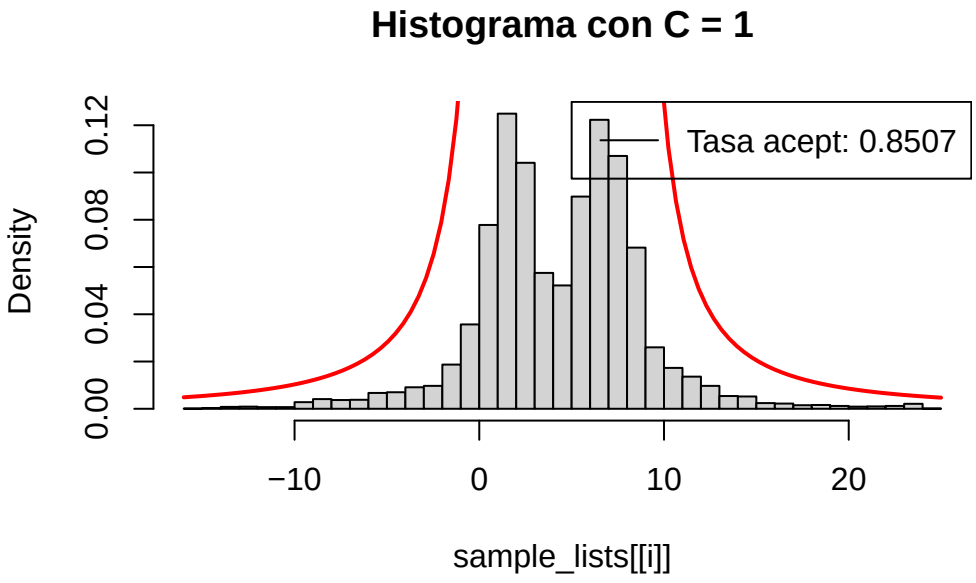
```
[1] "Log-densidad de y: -18.8280466933155"
```

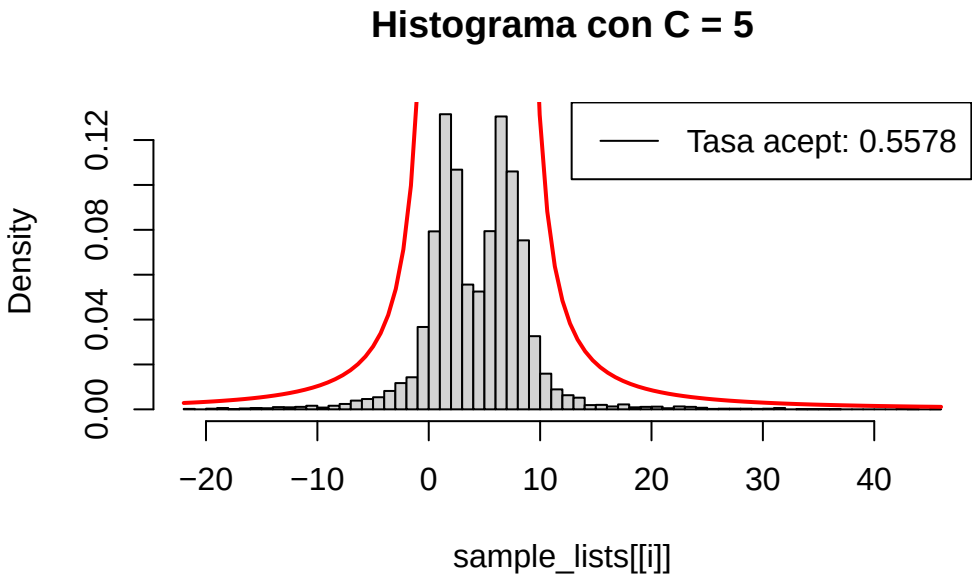
- Escriban una función que tome una muestra de tamaño 10,000 de la densidad de y, usando Metropolis-Hastings con función propuesta una caminata aleatoria con desviación estandar C. Investiguen el efecto de la elección de C en la tasa de aceptación, y la mezcla de la cadena en la densidad.

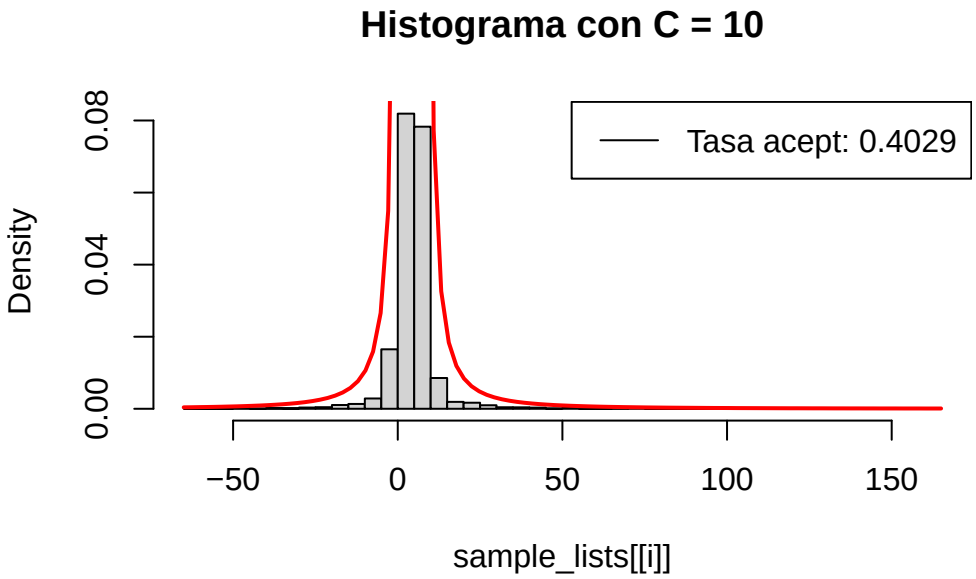
```r
# Mostrar las gráficas para cada valor de C
for (i in seq_along(C)) {
  hist(sample_lists[[i]], breaks = 50, freq = FALSE, main = paste("Histograma con C =", C[
  curve(densidad_poli_cauchy(x, a), col = "red", lwd = 2, add = TRUE)
  legend("topright", legend = paste("Tasa acept:", round(acceptance_rates[i],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

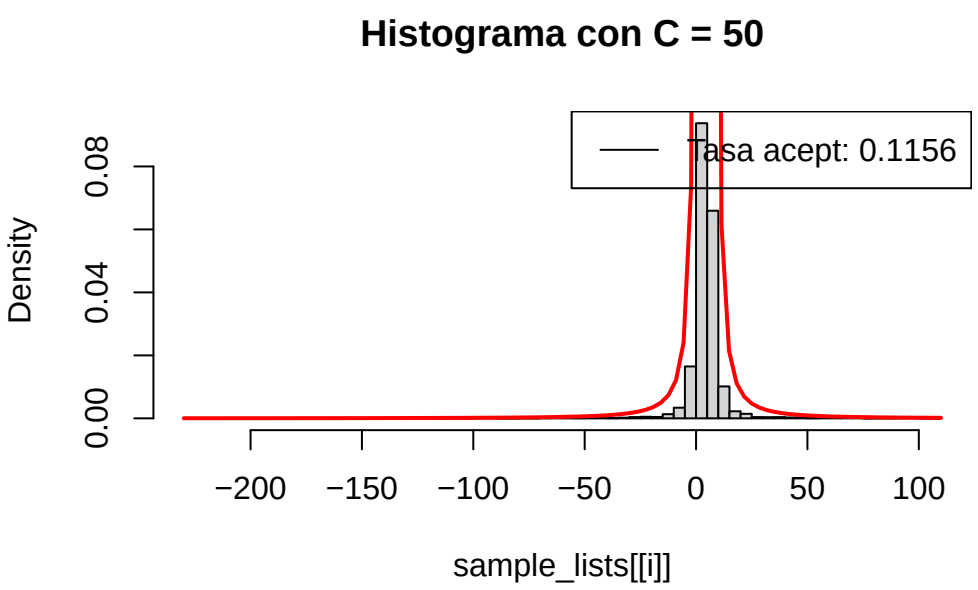



Histograma con C = 0.1
Tasa acept: 0.978
Density
0.00
0.10
0.20
−2
0
2
4
6
8
10
sample_lists[[i]]


Histograma con C = 1
Tasa acept: 0.8507
Density
0.00
0.04
0.08
0.12
−10
0
10
20
sample_lists[[i]]


Histograma con C = 5
Tasa acept: 0.5578
Density
0.00
0.04
0.08
0.12
−20
−10
0
10
20
30
40
sample_lists[[i]]


Histograma con C = 10
Tasa acept: 0.4029
Density
0.00
0.04
0.08
−50
0
50
100
150
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: 0.2072"
```

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 $b = 1$.

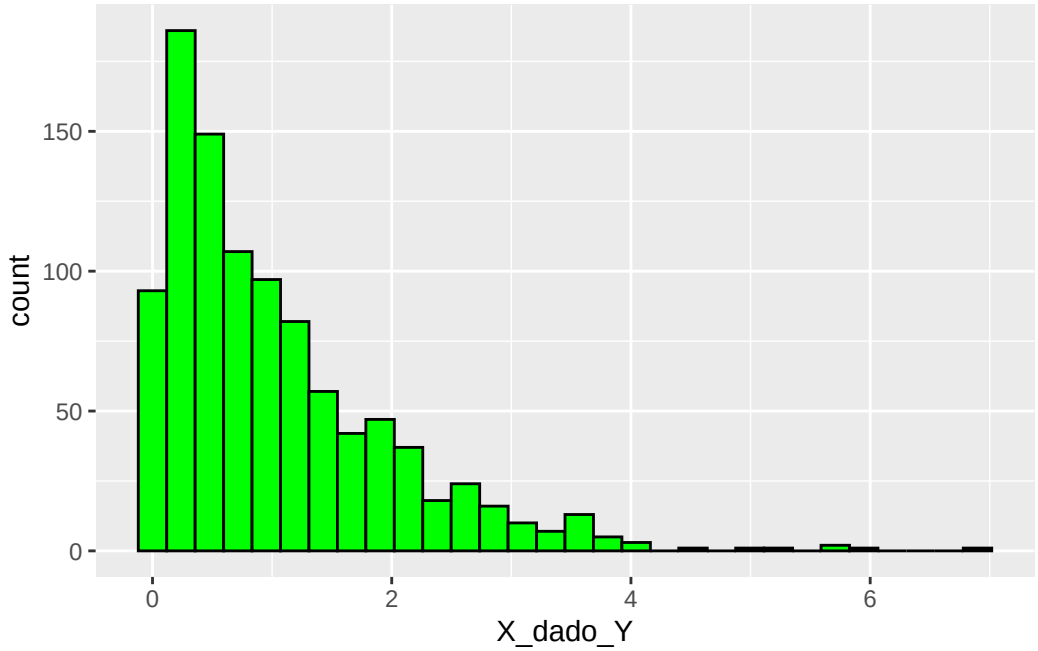
```
a <- 1
b <- 1
nsim <- 1000

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")
```

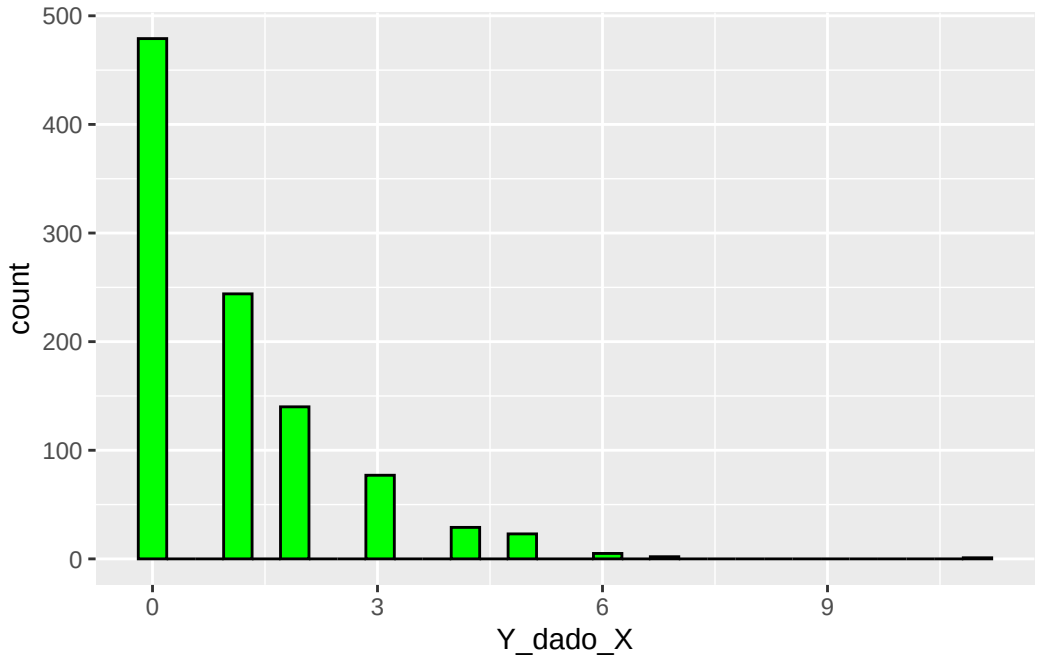
```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



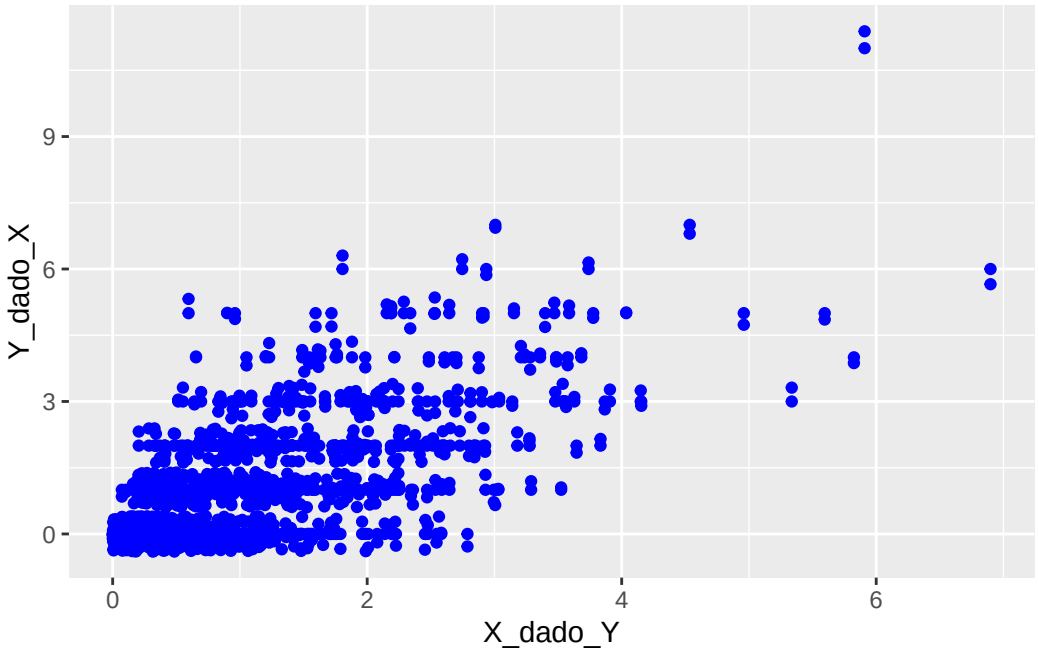

```
#hist(Y_dado_X, probability = T, main = "Y", ylab = "densidad")
```

```
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
```

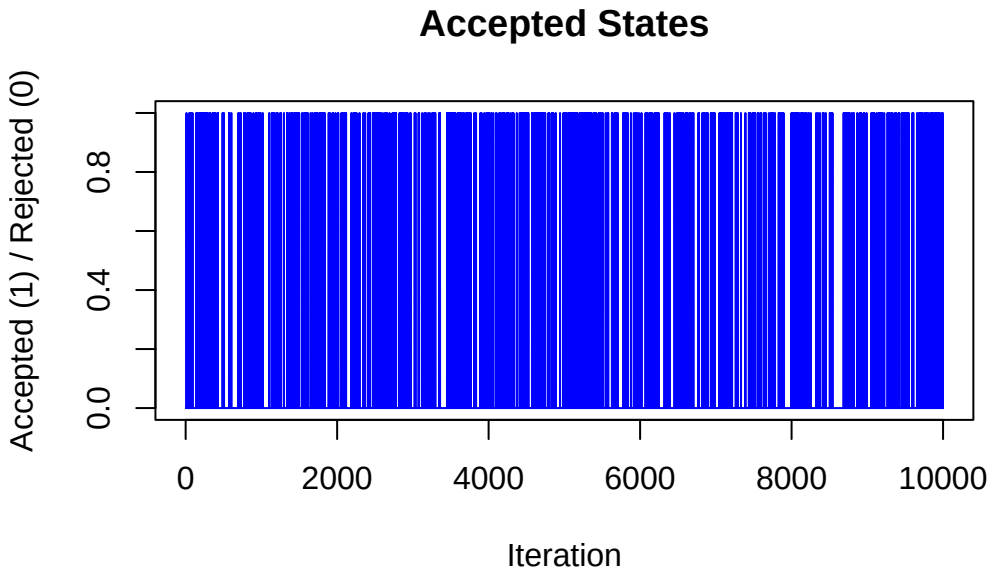
Esto se puede hacer calculando la fracción acumulada de veces que la cadena tiene 1's sobre la diagonal conforme se muestrea de la cadena. Comparar los resultados de las 2 cadenas. Tienen el mismo valor asintótico, pero ¿una cadena llega más rápido que la otra? ¿Alguna cadena muestra más autocorrelación 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)

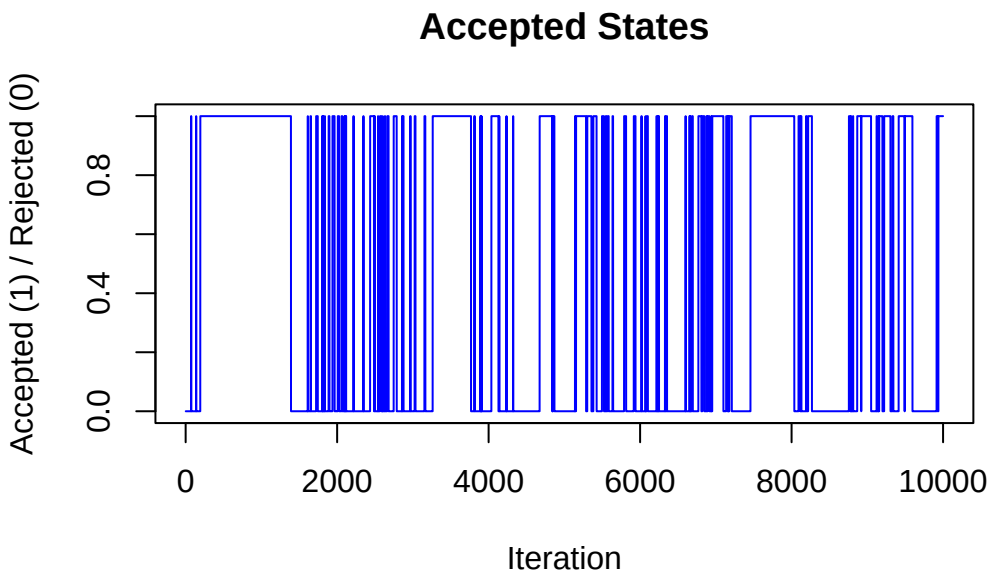
```
metropolis_hastings <- function(initial_vector, iterations, p, lambda) {
  current_vector <- initial_vector
  accepted_states <- numeric(iterations)

  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)")
```

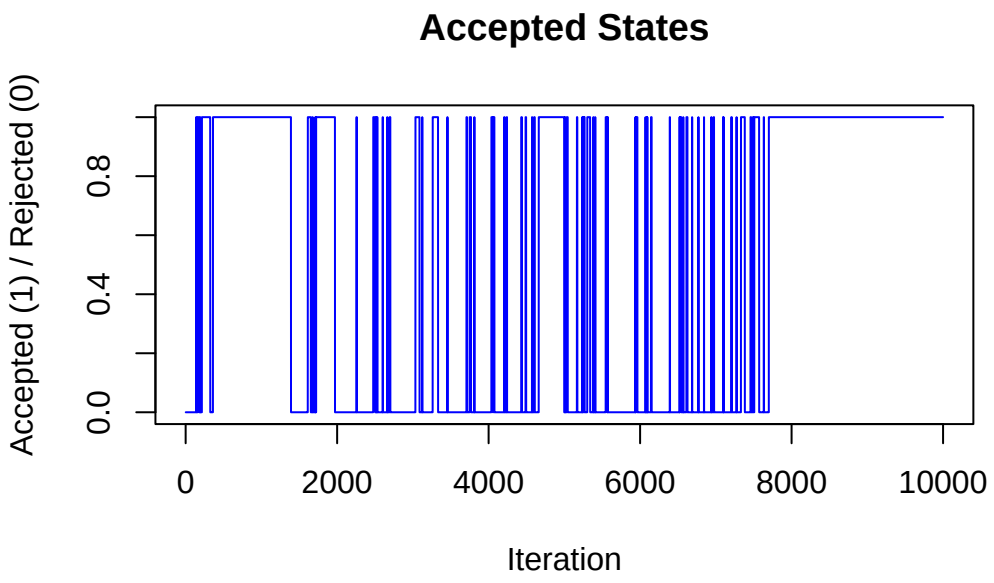


```r
# lambda_1, p_2,
set.seed(123)  # Set seed for reproducibility
initial_vector <- sample(0:1, 16, replace = TRUE)  # Initial random vector
iterations <- 10000
p <- p_2
lambda <- lambda_1

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)")
```

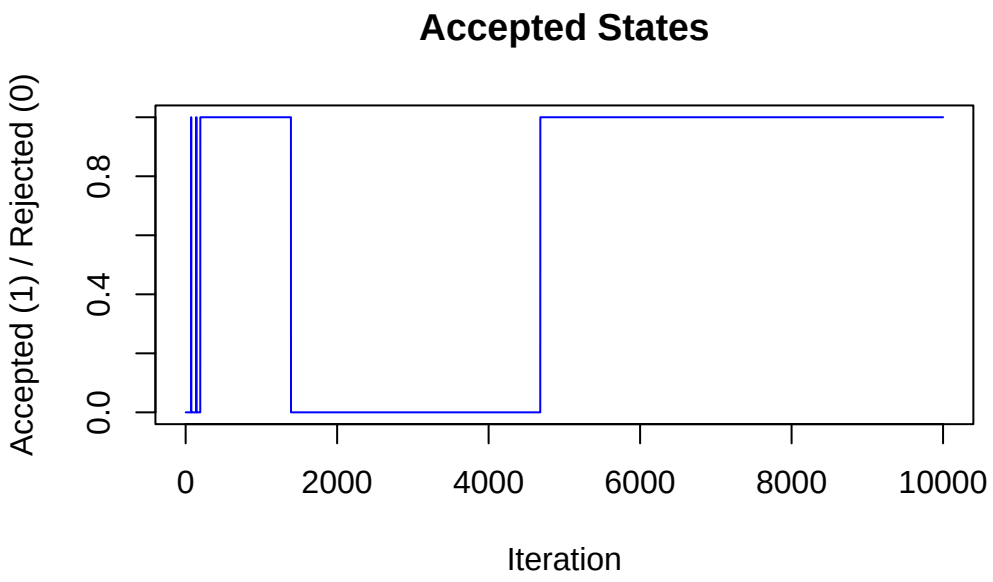



```r
# lambda_2, p_1,
set.seed(123)  # Set seed for reproducibility
initial_vector <- sample(0:1, 16, replace = TRUE)  # Initial random vector
iterations <- 10000
p <- p_1
lambda <- lambda_2

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

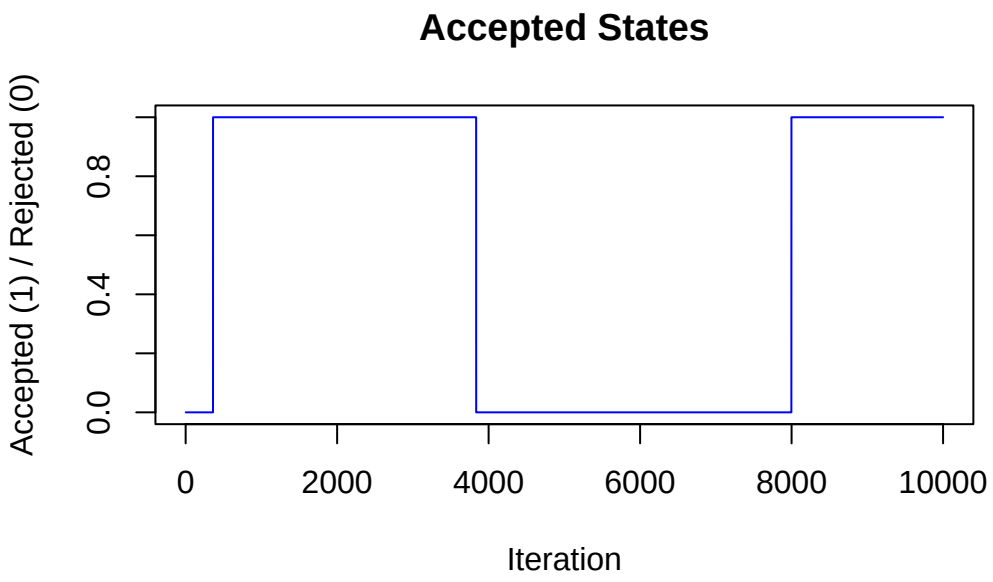
```r
# Print the result (1 if all positions 4, 8, 12, 16 have 1, 0 otherwise)
plot(accepted_states, type = "l", col = "blue", main = "Accepted States",
       xlab = "Iteration", ylab = "Accepted (1) / Rejected (0)")
```



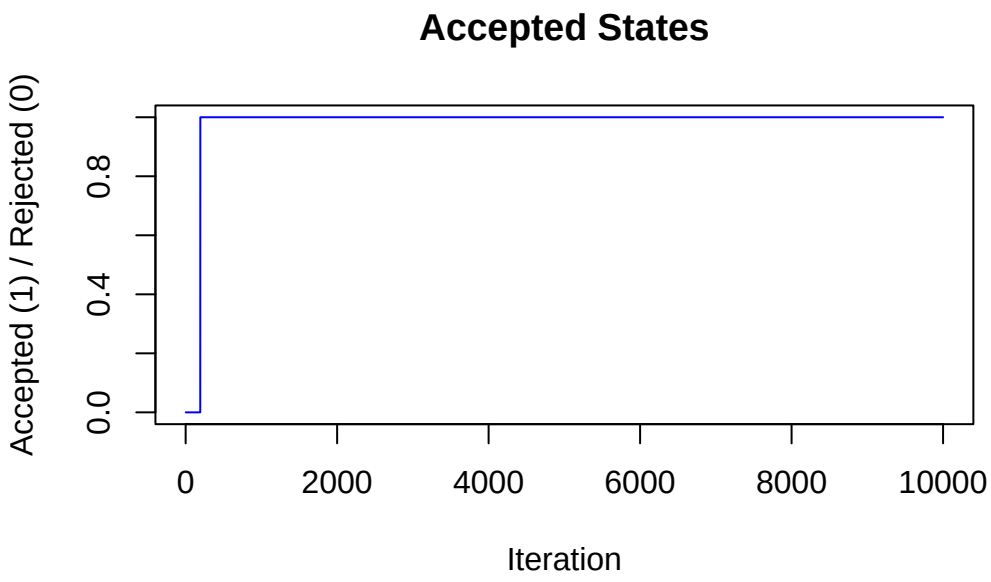

```r
# lambda_1, p_2,
set.seed(123)  # Set seed for reproducibility
initial_vector <- sample(0:1, 16, replace = TRUE)  # Initial random vector
iterations <- 10000
p <- p_2
lambda <- lambda_2

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)")
```



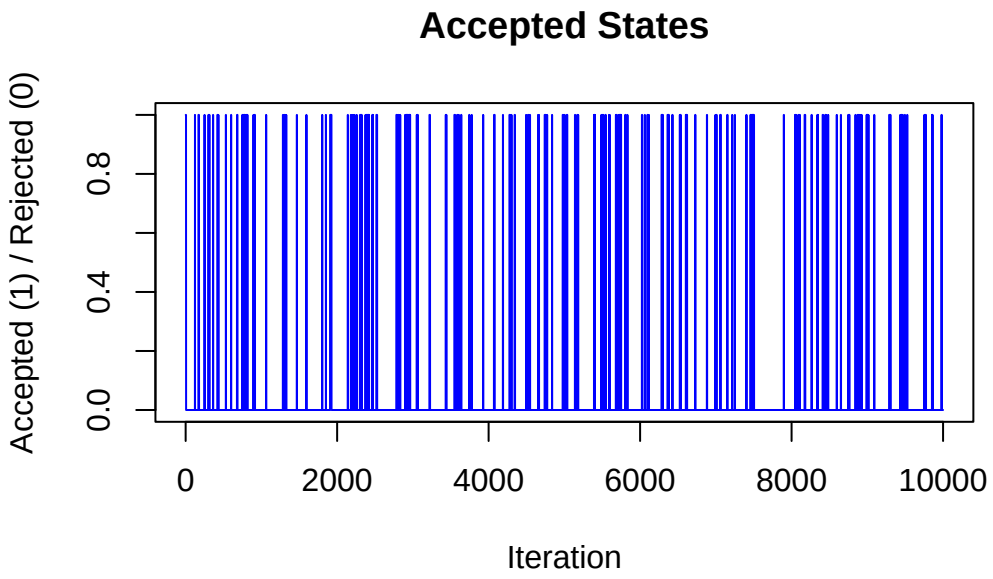

```r
# lambda_1, p_2,
set.seed(123)  # Set seed for reproducibility
initial_vector <- sample(0:1, 16, replace = TRUE)  # Initial random vector
iterations <- 10000
p <- p_2
lambda <- lambda_3

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)")
```



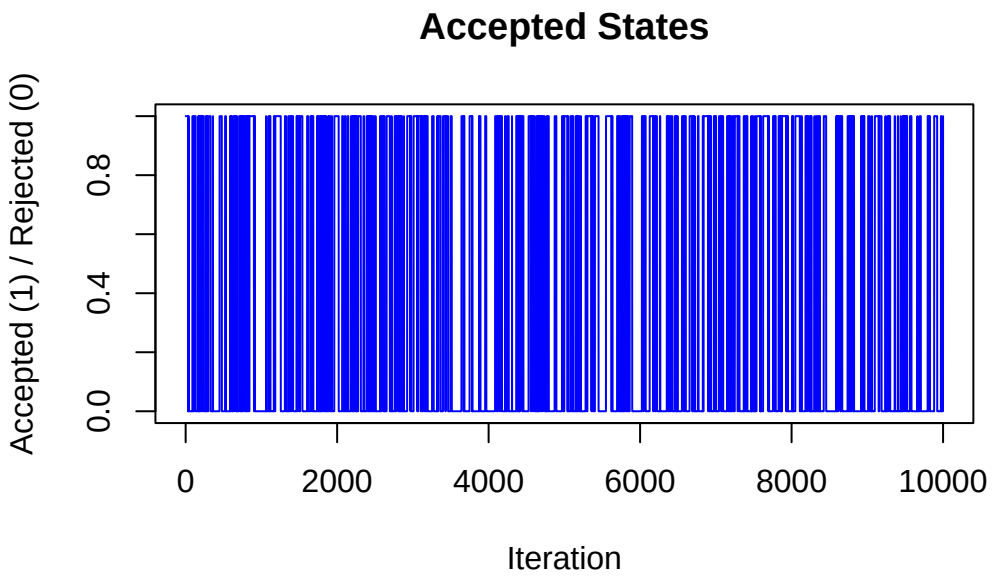

```r
metropolis_hastings2 <- function(initial_vector, iterations, p, lambda) {
  current_vector <- initial_vector
  accepted_states <- numeric(iterations)

  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)")
```

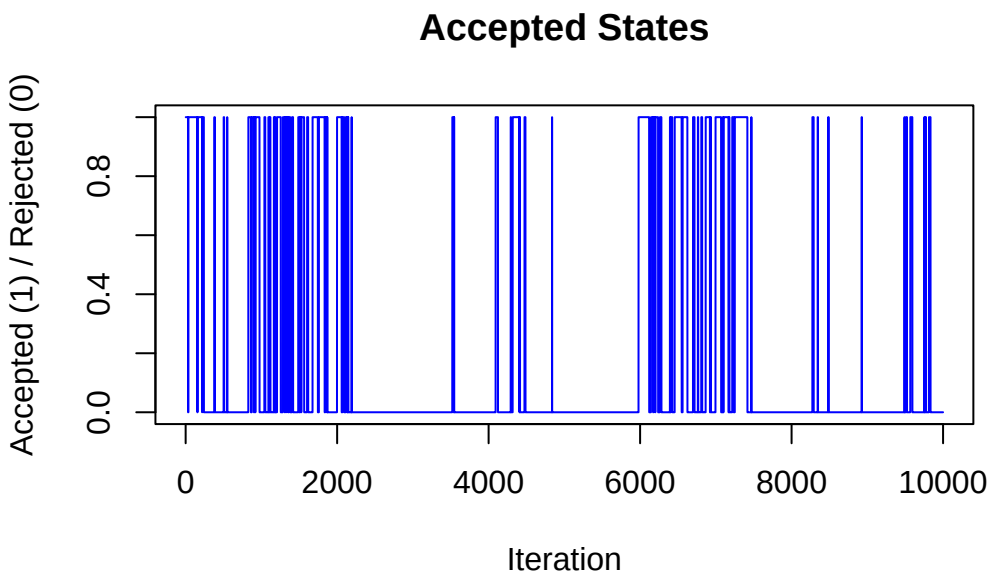


```r
# lambda_1, p_2,
set.seed(123)  # Set seed for reproducibility
initial_vector <- sample(0:1, 16, replace = TRUE)  # Initial random vector
iterations <- 10000
p <- p_2
lambda <- lambda_1

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)")
```

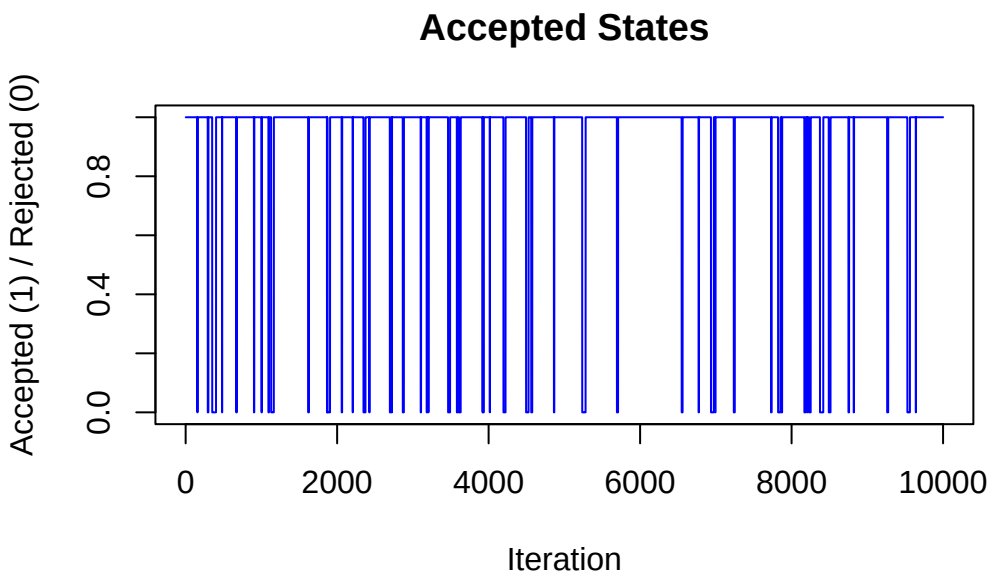



```r
# lambda_2, p_1,
set.seed(123)  # Set seed for reproducibility
initial_vector <- sample(0:1, 16, replace = TRUE)  # Initial random vector
iterations <- 10000
p <- p_1
lambda <- lambda_2

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

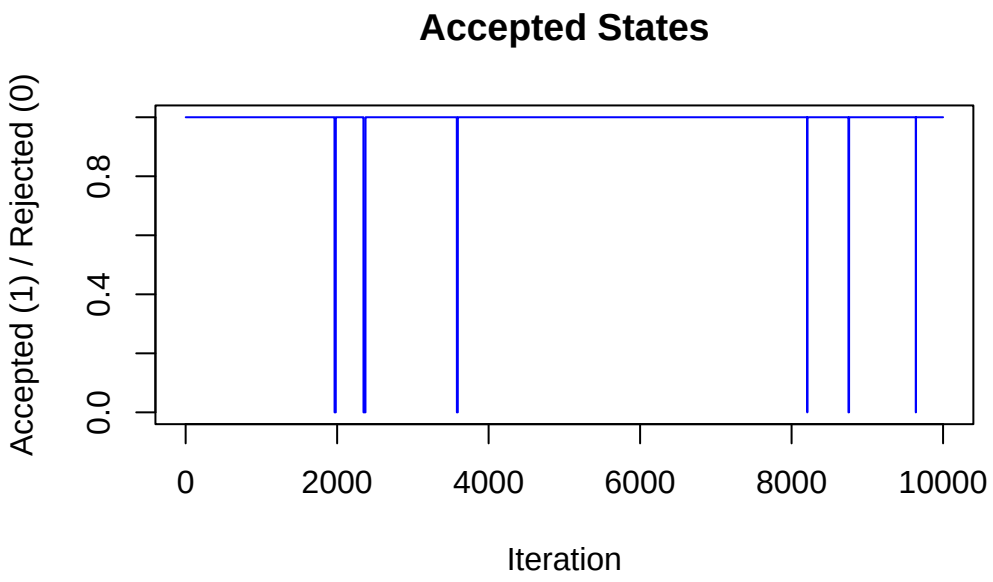
```r
# Print the result (1 if all positions 4, 8, 12, 16 have 1, 0 otherwise)
plot(accepted_states, type = "l", col = "blue", main = "Accepted States",
       xlab = "Iteration", ylab = "Accepted (1) / Rejected (0)")
```



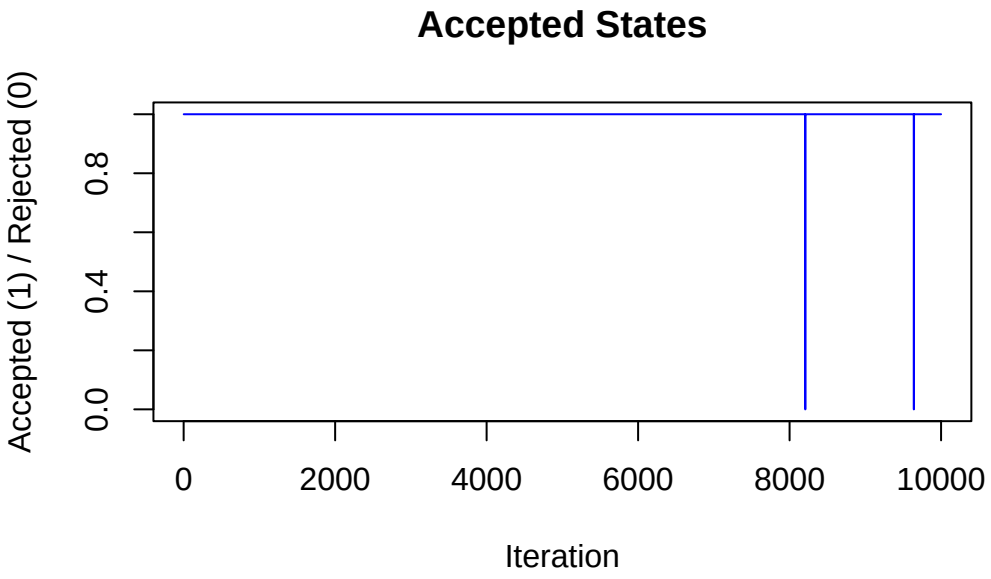

```r
# lambda_1, p_2,
set.seed(123)  # Set seed for reproducibility
initial_vector <- sample(0:1, 16, replace = TRUE)  # Initial random vector
iterations <- 10000
p <- p_2
lambda <- lambda_2

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. Exponencial y más

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 = 1 como valor inicial.
b) Compilen el modelo y obtengan una muestra de 1000 iteraciones después de descartar las 500 iteraciones iniciales como burnin.

```
library(bayesplot)
```

```
This is bayesplot version 1.11.1
```

```
- Online documentation and vignettes at mc-stan.org/bayesplot
```

```
- bayesplot theme set to bayesplot::theme_default()
```

```
   * Does _not_ affect other ggplot2 plots
```

```
   * See ?bayesplot_theme_set for details on theme setting
```

```
# Opciones recomendadas para evitar problemas de compilación
rstan_options(auto_write = TRUE)
options(mc.cores = parallel::detectCores())

# Definimos los datos para el modelo Stan
datos <- list(N = 10,
              y = c(0.4, 0.01, 0.2, 0.1, 2.1, 0.1, 0.9, 2.4, 0.1, 0.2))

setwd("./tarea4_files/")

# Compilamos y muestreamos el modelo desde el archivo
fit_exp <- stan(file = 'exp.stan', data = datos,
             chains = 4,
             warmup = 500,
             iter = 2000,
             seed = 41290
            )
```

```
recompiling to avoid crashing R session
```

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

```
Running /usr/lib/R/bin/R CMD SHLIB foo.c
using C compiler: ‘gcc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0’
gcc -I"/usr/share/R/include" -DNDEBUG   -I"/home/saraluz/.cache/R/renv/cache/v5/R-4.3/x86_64-
In file included from /home/saraluz/Documents/2Sem/MLG/Tareas/renv/library/R-4.3/x86_64-pc-li
                 from /home/saraluz/Documents/2Sem/MLG/Tareas/renv/library/R-4.3/x86_64-pc-li
                 from /home/saraluz/.cache/R/renv/cache/v5/R-4.3/x86_64-pc-linux-gnu/StanHead
                 from <command-line>:
/home/saraluz/Documents/2Sem/MLG/Tareas/renv/library/R-4.3/x86_64-pc-linux-gnu/RcppEigen/incl
  628 | namespace Eigen {
      | ^~~~~~~~~
/home/saraluz/Documents/2Sem/MLG/Tareas/renv/library/R-4.3/x86_64-pc-linux-gnu/RcppEigen/incl
  628 | namespace Eigen {
      |                 ^
In file included from /home/saraluz/Documents/2Sem/MLG/Tareas/renv/library/R-4.3/x86_64-pc-li
                 from /home/saraluz/.cache/R/renv/cache/v5/R-4.3/x86_64-pc-linux-gnu/StanHead
                 from <command-line>:
/home/saraluz/Documents/2Sem/MLG/Tareas/renv/library/R-4.3/x86_64-pc-linux-gnu/RcppEigen/incl
   96 | #include <complex>
      |          ^~~~~~~~~
compilation terminated.
make: *** [/usr/lib/R/etc/Makeconf:191: foo.o] Error 1
```

```r
print(fit_exp)
```

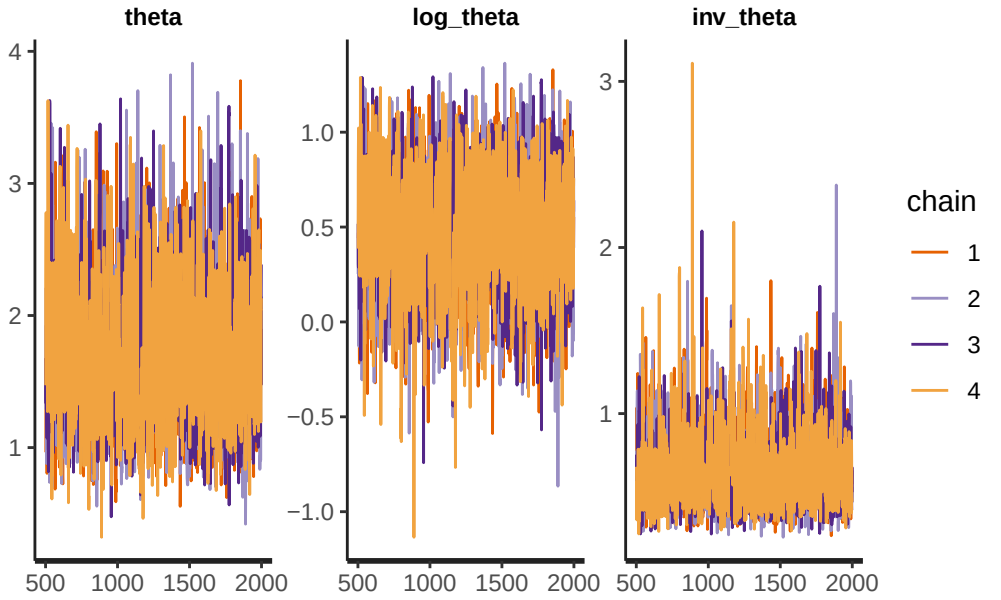
```
Inference for Stan model: anon_model.
4 chains, each with iter=2000; warmup=500; thin=1;
post-warmup draws per chain=1500, total post-warmup draws=6000.

           mean se_mean   sd  2.5%   25%   50%   75% 97.5% n_eff Rhat
theta      1.70    0.01 0.51  0.85  1.34  1.64  2.02  2.83  2145    1
log_theta  0.49    0.01 0.31 -0.17  0.29  0.50  0.70  1.04  2010    1
inv_theta  0.65    0.00 0.21  0.35  0.50  0.61  0.75  1.18  1949    1
lp__      -5.75    0.01 0.70 -7.74 -5.91 -5.48 -5.30 -5.24  2564    1

Samples were drawn using NUTS(diag_e) at Fri Mar  8 20:08:15 2024.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

b) Escriban el código para el modelo.

```
#modelo
#data {
#  int<lower=0> N;          // Número de observaciones
#  vector[N] y;             // Vector de tamaño N de observaciones/datos
#}
#
#parameters {
#  real<lower=0> theta;     // Parámetro de la distribución exponencial
#}
#
#model {
#  // Prior distribución para theta
#  theta ~ normal(0, 10);  // Distribución previa no informativa para log(theta)
#
#  // Likelihood de los datos
#  for (n in 1:N) {
#    y[n] ~ exponential(theta);
#  }
#}
#
#generated quantities {
#  real log_theta = log(theta); // Logaritmo de theta
#  real inv_theta = 1 / theta;  // Inverso de theta
#}
```

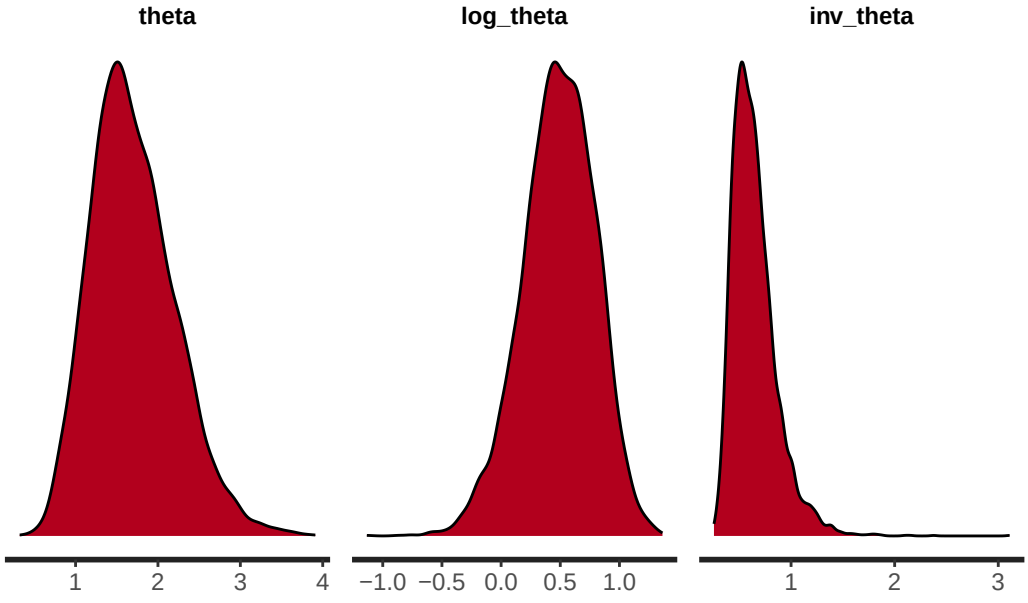
d) Monitoreen la convergencia gráficamente usando gráficas 'trace' y de autocorrelación.

```
stan_trace(fit_exp)
```

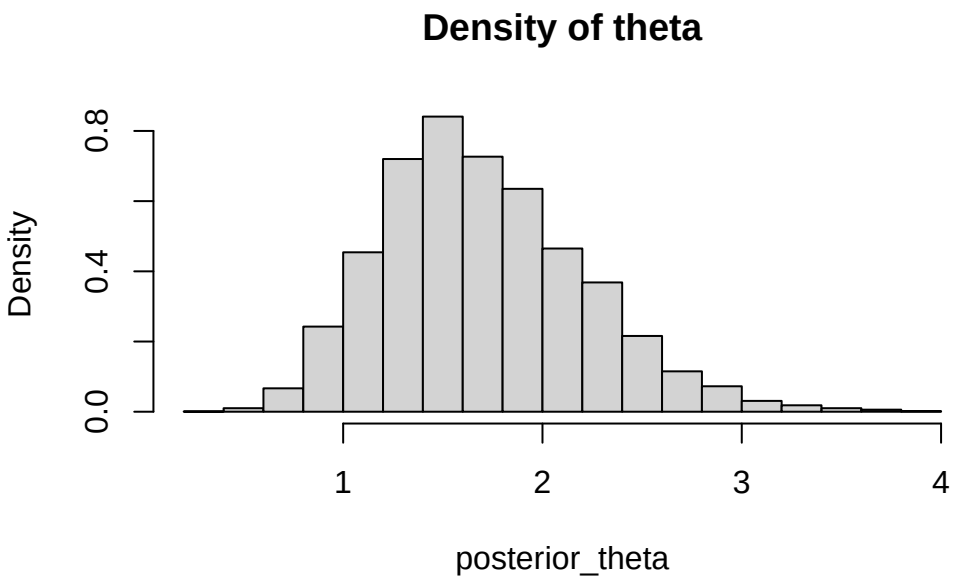


e) Obtengan estadísticas sumarias posteriores y densidades para θ, $1/\theta$ y $log\theta$

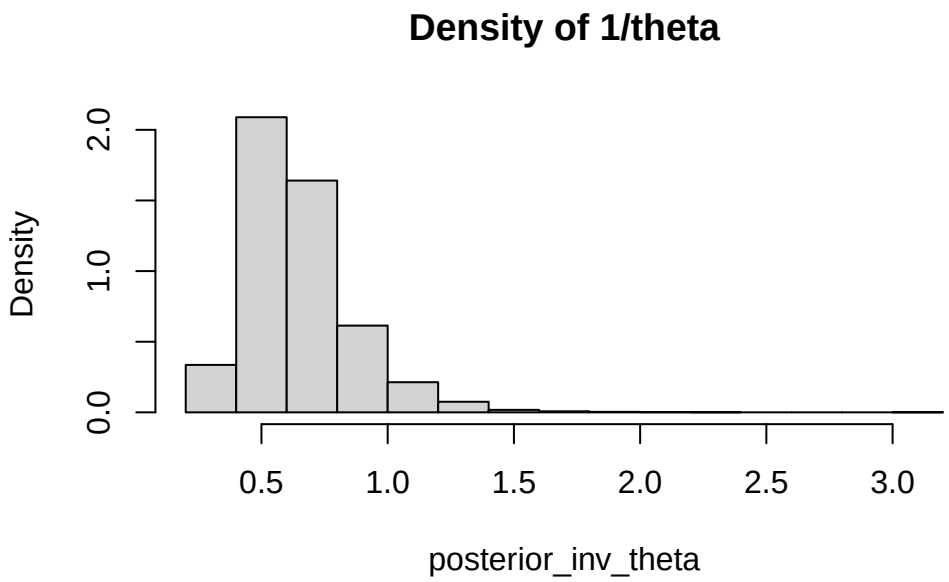
```
stan_dens(fit_exp)
```



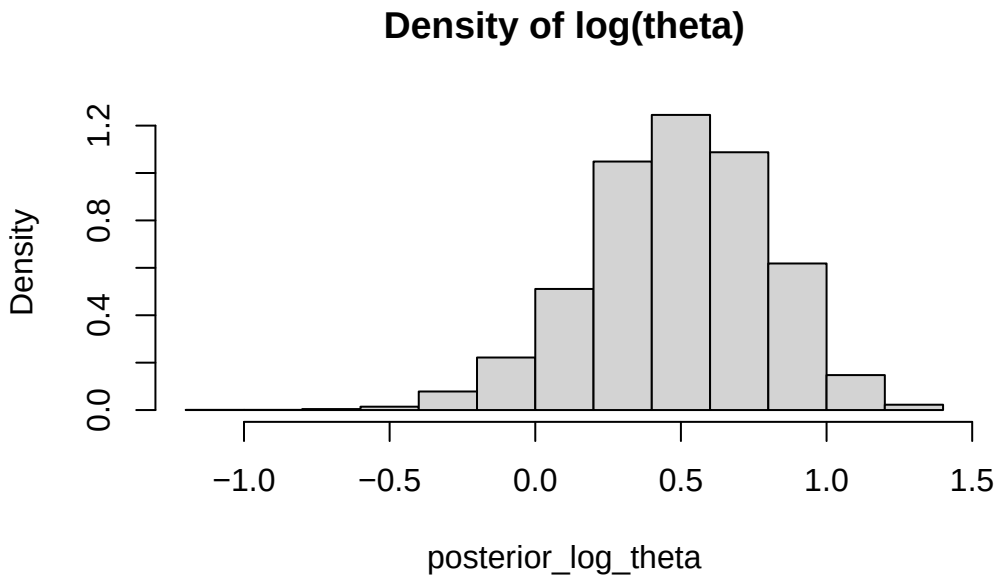
```r
posterior_theta <- extract(fit_exp)$theta
posterior_inv_theta <- extract(fit_exp)$inv_theta
posterior_log_theta <- extract(fit_exp)$log_theta

# Graficar las densidades
hist(posterior_theta, probability = TRUE, main = 'Density of theta')
```




```r
hist(posterior_inv_theta, probability = TRUE, main = 'Density of 1/theta')
```




```r
hist(posterior_log_theta, probability = TRUE, main = 'Density of 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.

(i) gamma

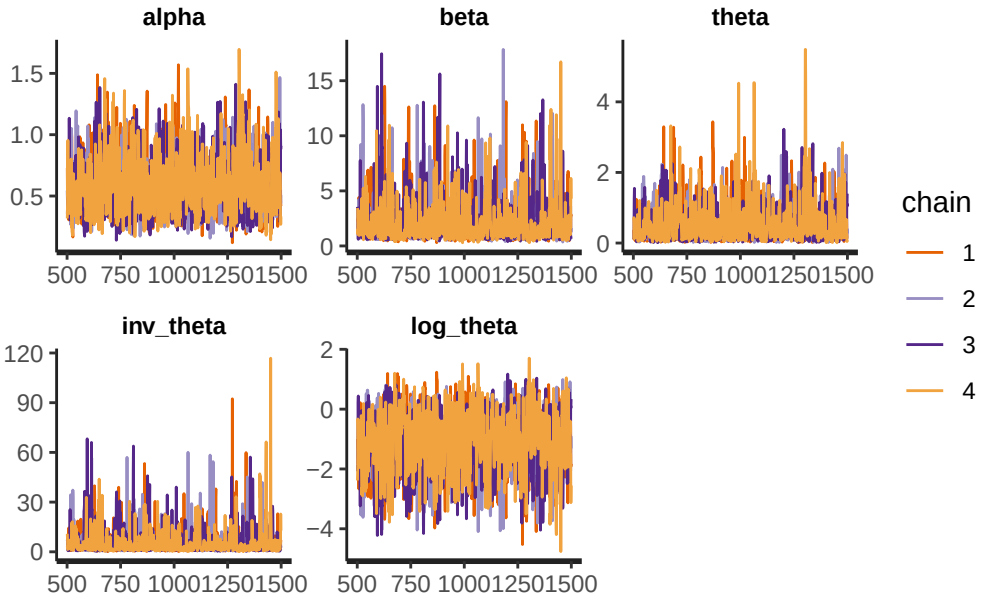
```
setwd("./tarea4_files/")

# Compilar y muestrear el modelo
fit_gamma <- stan(file = 'gamma.stan',
                  data = datos,
                  chains = 4,
                  warmup = 500,
                  iter = 1500,  # Total de iteraciones
                  seed = 41290)
```

```
recompiling to avoid crashing R session

Trying to compile a simple C file

Running /usr/lib/R/bin/R CMD SHLIB foo.c
using C compiler: ‘gcc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0’
gcc -I"/usr/share/R/include" -DNDEBUG   -I"/home/saraluz/.cache/R/renv/cache/v5/R-4.3/x86_64-
In file included from /home/saraluz/Documents/2Sem/MLG/Tareas/renv/library/R-4.3/x86_64-pc-li
                 from /home/saraluz/Documents/2Sem/MLG/Tareas/renv/library/R-4.3/x86_64-pc-li
                 from /home/saraluz/.cache/R/renv/cache/v5/R-4.3/x86_64-pc-linux-gnu/StanHead
                 from <command-line>:
/home/saraluz/Documents/2Sem/MLG/Tareas/renv/library/R-4.3/x86_64-pc-linux-gnu/RcppEigen/incl
  628 | namespace Eigen {
      | ^~~~~~~~~
/home/saraluz/Documents/2Sem/MLG/Tareas/renv/library/R-4.3/x86_64-pc-linux-gnu/RcppEigen/incl
  628 | namespace Eigen {
      |                 ^
In file included from /home/saraluz/Documents/2Sem/MLG/Tareas/renv/library/R-4.3/x86_64-pc-li
                 from /home/saraluz/.cache/R/renv/cache/v5/R-4.3/x86_64-pc-linux-gnu/StanHead
                 from <command-line>:
/home/saraluz/Documents/2Sem/MLG/Tareas/renv/library/R-4.3/x86_64-pc-linux-gnu/RcppEigen/incl
   96 | #include <complex>
      |          ^~~~~~~~~
compilation terminated.
make: *** [/usr/lib/R/etc/Makeconf:191: foo.o] Error 1
```

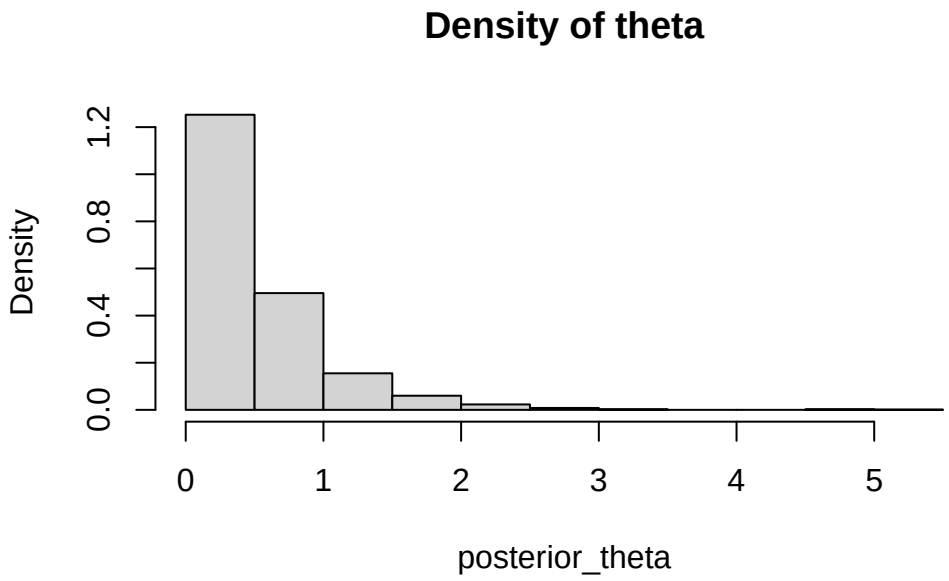
```
print(fit_gamma)
```

```
Inference for Stan model: anon_model.
4 chains, each with iter=1500; warmup=500; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=4000.

           mean se_mean   sd  2.5%   25%   50%   75% 97.5% n_eff Rhat
alpha      0.58    0.01 0.22  0.25  0.42  0.55  0.71  1.10  1080    1
beta       2.11    0.05 1.87  0.54  0.99  1.50  2.47  7.59  1209    1
theta      0.51    0.02 0.50  0.04  0.18  0.36  0.68  1.87   910    1
inv_theta  5.14    0.20 7.22  0.53  1.47  2.76  5.61 24.93  1303    1
log_theta -1.10    0.03 0.99 -3.22 -1.72 -1.01 -0.38  0.63   988    1
lp__      -6.16    0.02 0.95 -8.59 -6.59 -5.89 -5.46 -5.19  1446    1

Samples were drawn using NUTS(diag_e) at Fri Mar  8 20:09:27 2024.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

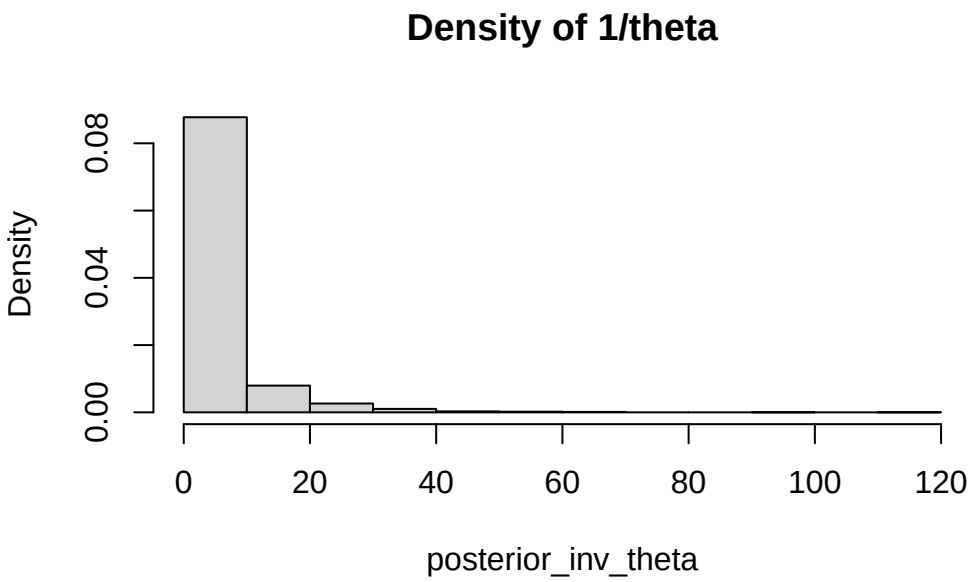
```
stan_trace(fit_gamma)
```



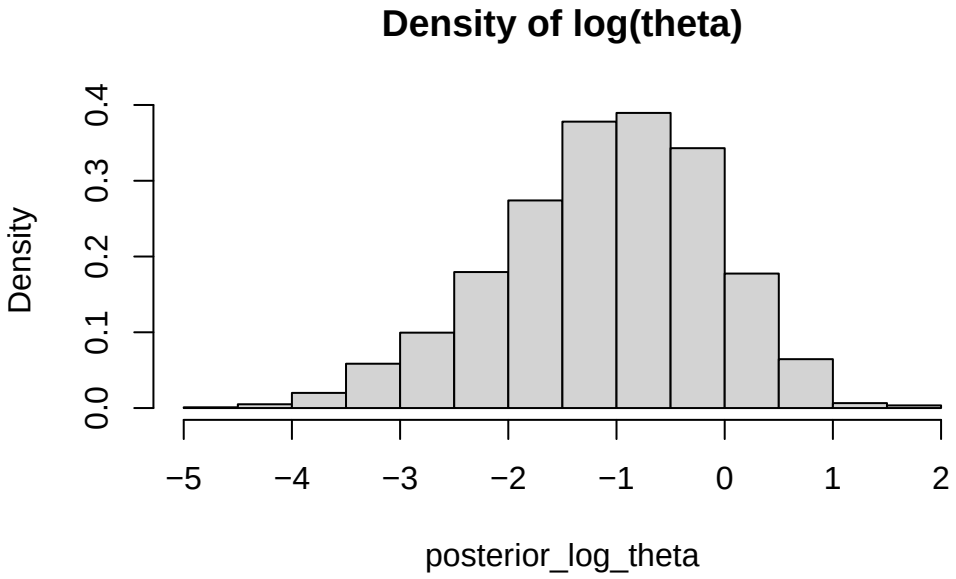
```r
# Obtener y graficar la densidad posterior para theta, 1/theta y log(theta)
posterior_theta <- extract(fit_gamma)$theta
posterior_inv_theta <- extract(fit_gamma)$inv_theta
posterior_log_theta <- extract(fit_gamma)$log_theta

# Graficar las densidades
hist(posterior_theta, probability = TRUE, main = 'Density of theta')
```




```r
hist(posterior_inv_theta, probability = TRUE, main = 'Density of 1/theta')
```




```r
hist(posterior_log_theta, probability = TRUE, main = 'Density of log(theta)')
```



(ii) log-normal

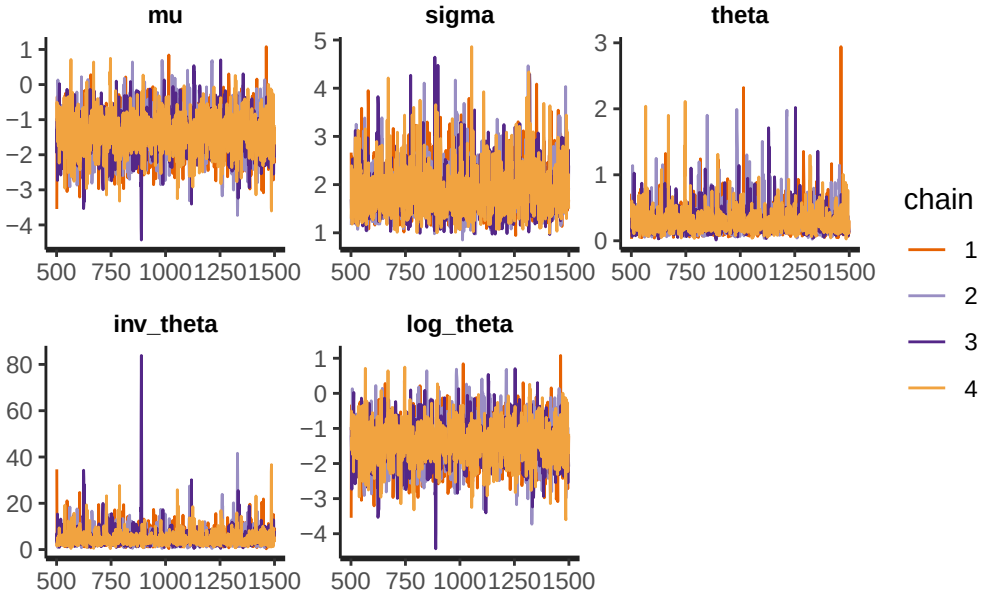
```r
setwd("./tarea4_files/")

# Compilar y muestrear el modelo
fit_lognormal <- stan(file = 'lognormal.stan',
                      data = datos,
                      chains = 4,
                      warmup = 500,
                      iter = 1500,  # Total de iteraciones
                      seed = 41290)
```

```
recompiling to avoid crashing R session

Trying to compile a simple C file

Running /usr/lib/R/bin/R CMD SHLIB foo.c
using C compiler: ‘gcc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0’
gcc -I"/usr/share/R/include" -DNDEBUG   -I"/home/saraluz/.cache/R/renv/cache/v5/R-4.3/x86_64-
In file included from /home/saraluz/Documents/2Sem/MLG/Tareas/renv/library/R-4.3/x86_64-pc-li
                 from /home/saraluz/Documents/2Sem/MLG/Tareas/renv/library/R-4.3/x86_64-pc-li
                 from /home/saraluz/.cache/R/renv/cache/v5/R-4.3/x86_64-pc-linux-gnu/StanHea
                 from <command-line>:
/home/saraluz/Documents/2Sem/MLG/Tareas/renv/library/R-4.3/x86_64-pc-linux-gnu/RcppEigen/incl
  628 | namespace Eigen {
      | ^~~~~~~~~
/home/saraluz/Documents/2Sem/MLG/Tareas/renv/library/R-4.3/x86_64-pc-linux-gnu/RcppEigen/incl
  628 | namespace Eigen {
      |                 ^
In file included from /home/saraluz/Documents/2Sem/MLG/Tareas/renv/library/R-4.3/x86_64-pc-li
                 from /home/saraluz/.cache/R/renv/cache/v5/R-4.3/x86_64-pc-linux-gnu/StanHea
                 from <command-line>:
/home/saraluz/Documents/2Sem/MLG/Tareas/renv/library/R-4.3/x86_64-pc-linux-gnu/RcppEigen/incl
   96 | #include <complex>
      |          ^~~~~~~~~
compilation terminated.
make: *** [/usr/lib/R/etc/Makeconf:191: foo.o] Error 1
```

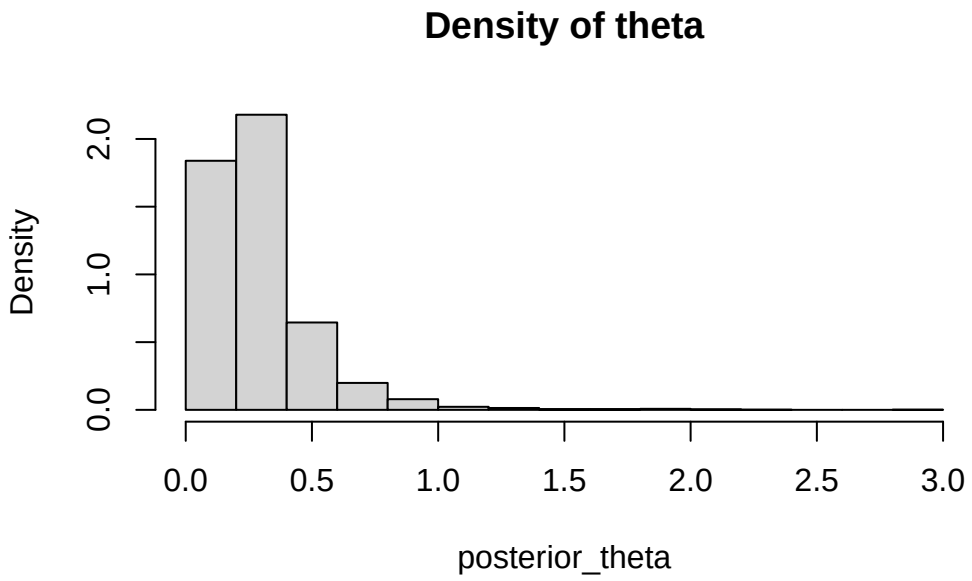
```r
print(fit_lognormal)
```

```
Inference for Stan model: anon_model.
4 chains, each with iter=1500; warmup=500; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=4000.

            mean se_mean   sd   2.5%    25%   50%   75% 97.5% n_eff Rhat
mu         -1.41    0.01 0.61  -2.62  -1.79 -1.41 -1.02 -0.19  1732    1
sigma       1.88    0.01 0.50   1.16   1.53  1.79  2.12  3.09  1917    1
theta       0.30    0.01 0.21   0.07   0.17  0.24  0.36  0.83  1795    1
inv_theta   4.96    0.10 3.84   1.21   2.78  4.10  5.99 13.73  1517    1
log_theta  -1.41    0.01 0.61  -2.62  -1.79 -1.41 -1.02 -0.19  1732    1
lp__      -10.19    0.03 1.05 -13.03 -10.60 -9.87 -9.42 -9.14  1441    1

Samples were drawn using NUTS(diag_e) at Fri Mar  8 20:10:37 2024.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

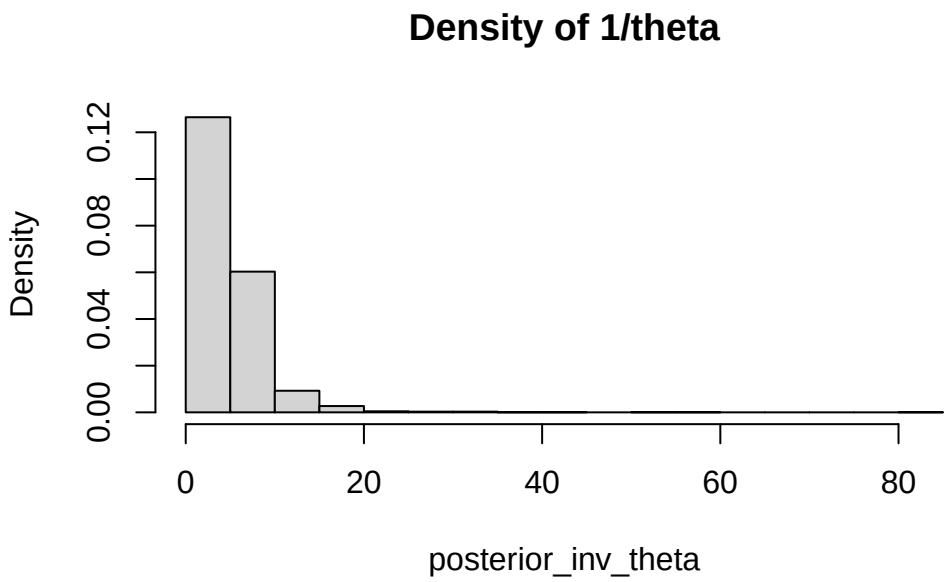
```
stan_trace(fit_lognormal)
```



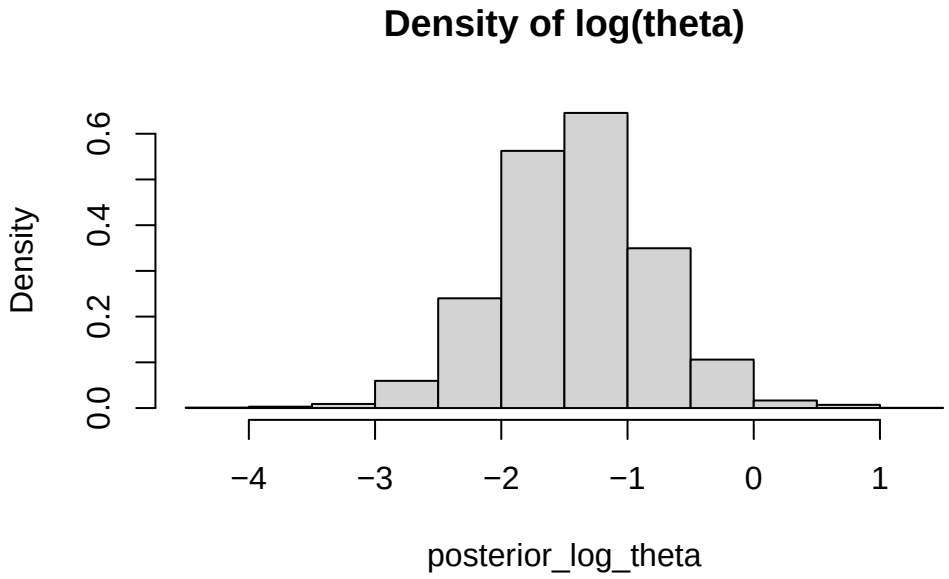
```r
# Obtener y graficar la densidad posterior para theta, 1/theta y log(theta)
posterior_theta <- extract(fit_lognormal)$theta
posterior_inv_theta <- extract(fit_lognormal)$inv_theta
posterior_log_theta <- extract(fit_lognormal)$log_theta

# Graficar las densidades
hist(posterior_theta, probability = TRUE, main = 'Density of theta')
```




```r
hist(posterior_inv_theta, probability = TRUE, main = 'Density of 1/theta')
```




```r
hist(posterior_log_theta, probability = TRUE, main = 'Density of log(theta)')
```



(iii) normal

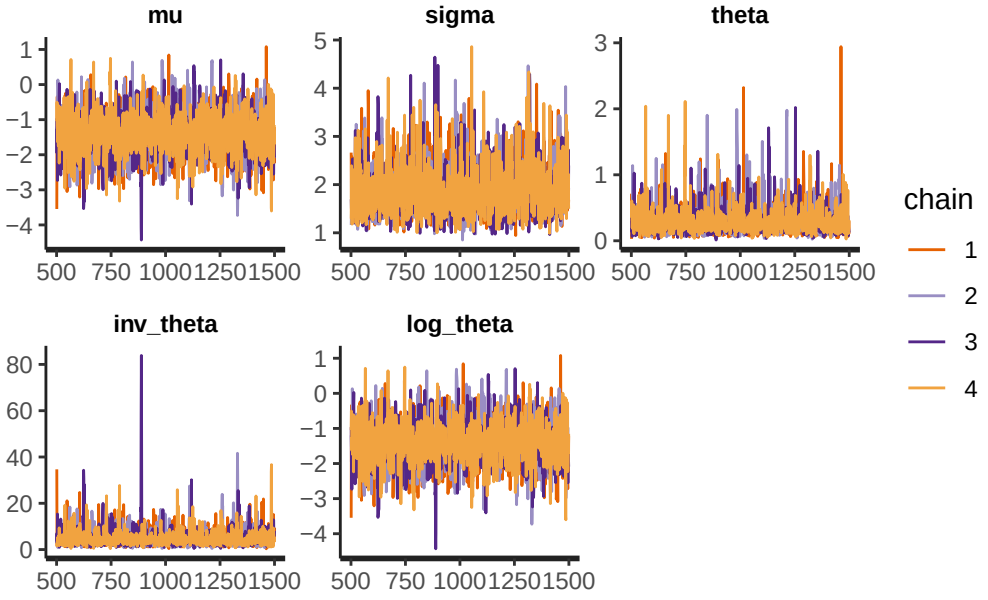
```
setwd("./tarea4_files/")

# Compilar y muestrear el modelo
fit_normal_log <- stan(file = 'normal.stan',
                       data = datos,
                       chains = 4,
                       warmup = 500,
                       iter = 1500,  # Total de iteraciones = warmup + iteraciones deseada
                       seed = 41290)
```

```
recompiling to avoid crashing R session

Trying to compile a simple C file

Running /usr/lib/R/bin/R CMD SHLIB foo.c
using C compiler: ‘gcc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0’
gcc -I"/usr/share/R/include" -DNDEBUG   -I"/home/saraluz/.cache/R/renv/cache/v5/R-4.3/x86_64-
In file included from /home/saraluz/Documents/2Sem/MLG/Tareas/renv/library/R-4.3/x86_64-pc-li
                 from /home/saraluz/Documents/2Sem/MLG/Tareas/renv/library/R-4.3/x86_64-pc-li
                 from /home/saraluz/.cache/R/renv/cache/v5/R-4.3/x86_64-pc-linux-gnu/StanHead
                 from <command-line>:
/home/saraluz/Documents/2Sem/MLG/Tareas/renv/library/R-4.3/x86_64-pc-linux-gnu/RcppEigen/incl
  628 | namespace Eigen {
      | ^~~~~~~~~
/home/saraluz/Documents/2Sem/MLG/Tareas/renv/library/R-4.3/x86_64-pc-linux-gnu/RcppEigen/incl
  628 | namespace Eigen {
      |                 ^
In file included from /home/saraluz/Documents/2Sem/MLG/Tareas/renv/library/R-4.3/x86_64-pc-li
                 from /home/saraluz/.cache/R/renv/cache/v5/R-4.3/x86_64-pc-linux-gnu/StanHead
                 from <command-line>:
/home/saraluz/Documents/2Sem/MLG/Tareas/renv/library/R-4.3/x86_64-pc-linux-gnu/RcppEigen/incl
   96 | #include <complex>
      |          ^~~~~~~~~
compilation terminated.
make: *** [/usr/lib/R/etc/Makeconf:191: foo.o] Error 1
```

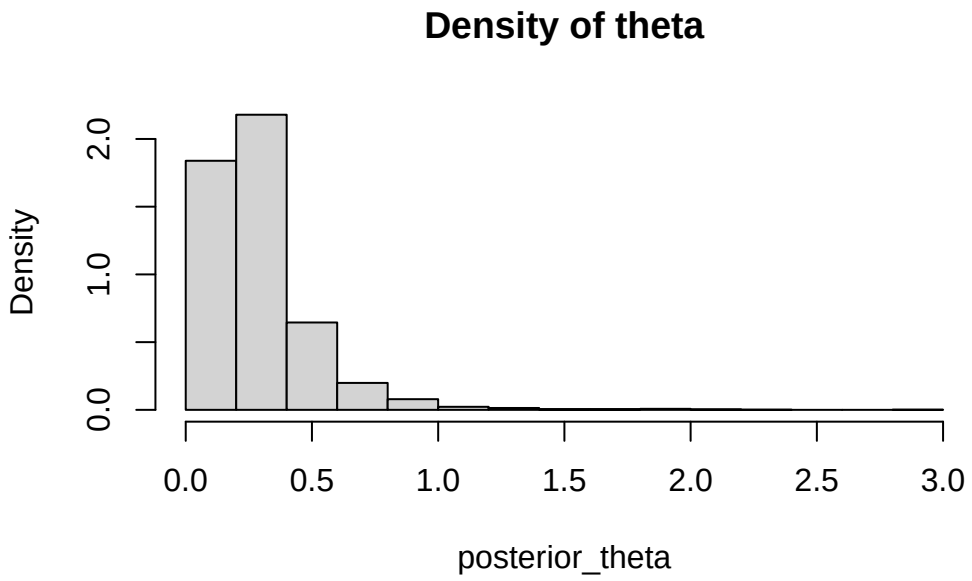
```
print(fit_normal_log)
```

```
Inference for Stan model: anon_model.
4 chains, each with iter=1500; warmup=500; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=4000.

            mean se_mean   sd   2.5%    25%   50%   75% 97.5% n_eff Rhat
mu         -1.41    0.01 0.61  -2.62  -1.79 -1.41 -1.02 -0.19  1732    1
sigma       1.88    0.01 0.50   1.16   1.53  1.79  2.12  3.09  1917    1
theta       0.30    0.01 0.21   0.07   0.17  0.24  0.36  0.83  1795    1
inv_theta   4.96    0.10 3.84   1.21   2.78  4.10  5.99 13.73  1517    1
log_theta  -1.41    0.01 0.61  -2.62  -1.79 -1.41 -1.02 -0.19  1732    1
lp__      -10.19    0.03 1.05 -13.03 -10.60 -9.87 -9.42 -9.14  1441    1

Samples were drawn using NUTS(diag_e) at Fri Mar  8 20:11:49 2024.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

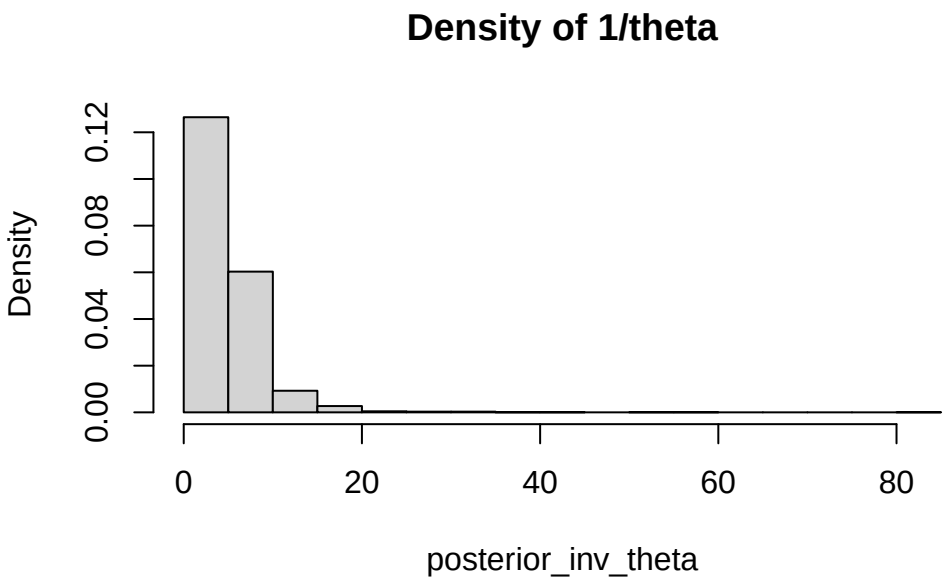
```
stan_trace(fit_normal_log)
```



```r
# Obtener y graficar la densidad posterior para theta, 1/theta y log(theta)
posterior_theta <- extract(fit_normal_log)$theta
posterior_inv_theta <- extract(fit_normal_log)$inv_theta
posterior_log_theta <- extract(fit_normal_log)$log_theta

# Graficar las densidades
hist(posterior_theta, probability = TRUE, main = 'Density of theta')
```




```r
hist(posterior_inv_theta, probability = TRUE, main = 'Density of 1/theta')
```




```r
hist(posterior_log_theta, probability = TRUE, main = 'Density of log(theta)')
```