EST-46115: Modelación Bayesiana

Profesor: Alfredo Garbuno Iñigo — Primavera, 2023 — Introducción a Stan.
Objetivo. Un primer acercamiento a `Stan`, sintaxis, manipulación de objetos y visualizaciones sencillas. Además un caso de estudio donde empezaremos a ver ciertos errores en el ajuste de los modelos.
Lectura recomendada: Sección 6.2.1 de [5].

1. INTRODUCCIÓN

Veremos ejemplos sencillos sobre la sintaxis de `Stan` ([2]) con el cual simularemos realizaciones de parámetros `no observables` para los cuales tenemos un `estado de conocimiento` reflejado en la distribución posterior.

Habíamos visto que los *bayesics* son los componentes:

1. Un conjunto de datos.
2. Supuestos del proceso generador de datos.
3. Conocimiento previo sobre los componentes que rigen el modelo.

Para `Stan` tenemos que seguir algo muy similar. Es decir,

1. Definir el modelo.
2. Ingestar los datos.
3. Darle *play* al botón de inferencia.

Un modelo de `Stan` se escribe en un archivo de texto y es una secuencia de bloques con nombre. En general el esqueleto es como sigue:

```
functions {
    // ... function declarations and definitions ...
}
data {
    // ... declarations ...
}
transformed data {
    // ... declarations ... statements ...
}
parameters {
    // ... declarations ...
}
transformed parameters {
    // ... declarations ... statements ...
}
model {
    // ... declarations ... statements ...
}
generated quantities {
    // ... declarations ... statements ...
}
```

Listing 1. *Estructura de un modelo de `Stan`.*

En general `todos` los bloques `son opcionales`, y **no es necesario** tener todos para compilar un modelo. Para mas información puedes consultar la guía de Stan.

1.1. Estructura del código

Consideremos el modelo Beta-Binomial que hemos trabajado antes.

- Un bloque `data`. Por ejemplo, Y el número observado de éxitos en 10 pruebas.
- Un bloque `parameters`. Por ejemplo, θ la tasa de éxito en la realización de las pruebas.
- Un bloque `model`. Por ejemplo, $Y \sim \mathsf{Binomial}(10, \theta)$ y $\theta \sim \mathsf{Beta}(2, 2)$.

El código es:

```
data {
  int<lower = 0, upper = 10> Y;
}
parameters {
  real<lower = 0, upper = 1> theta;
}
model {
  Y ~ binomial(10, theta);
  theta ~ beta(2, 2);
}
```

```
## Modelo Beta-Binomial
    -------------------------------------------------------
modelos_files <- "modelos/compilados/tutorial"
ruta <- file.path("modelos/tutorial/beta-binomial.stan")
modelo <- cmdstan_model(ruta, dir = modelos_files)
```

Listing 2. *Código necesario para interactuar con* `Stan` *desde* `R`*.*

> Para leer mas sobre la herramienta y sus interacción desde línea de comandos puedes consultar la documentación de `Stan`.

1.1.1. Interacción con ejecutable La instrucción `cmdstan_model` compila el modelo y crea un `ejecutable` con el cual podemos interactuar desde `terminal`. Los datos del modelo los guardamos en un `JSON`

```
{
   "Y" : 7
}
```

Tenemos un ejecutable:

```
ls modelos/compilados/tutorial
```

```
Permissions Size User Date Modified Git Name
.rwxr-xr-x  1.6M user  1 Mar 11:09   -I  beta-binomial*
```

Con el cual podemos interactuar por medio de

```
./beta-binomial sample data file=../../tutorial/datos-bb.json
```

```
method = sample (Default)
  sample
    num_samples = 1000 (Default)
    num_warmup = 1000 (Default)
    save_warmup = 0 (Default)
    thin = 1 (Default)
    adapt
      engaged = 1 (Default)
      gamma = 0.050000000000000003 (Default)
      delta = 0.80000000000000004 (Default)
      kappa = 0.75 (Default)
      t0 = 10 (Default)
      init_buffer = 75 (Default)
      term_buffer = 50 (Default)
      window = 25 (Default)
    algorithm = hmc (Default)
      hmc
        engine = nuts (Default)
          nuts
            max_depth = 10 (Default)
        metric = diag_e (Default)
        metric_file =  (Default)
        stepsize = 1 (Default)
        stepsize_jitter = 0 (Default)
    num_chains = 1 (Default)
id = 1 (Default)
data
  file = ../../tutorial/datos-bb.json
init = 2 (Default)
random
  seed = 2774886018 (Default)
output
  file = output.csv (Default)
  diagnostic_file =  (Default)
  refresh = 100 (Default)
  sig_figs = -1 (Default)
  profile_file = profile.csv (Default)
num_threads = 1 (Default)


Gradient evaluation took 6e-06 seconds
1000 transitions using 10 leapfrog steps per transition would take 0.06 seconds.
Adjust your expectations accordingly!


Iteration:    1 / 2000 [  0%]  (Warmup)
Iteration:  100 / 2000 [  5%]  (Warmup)
Iteration:  200 / 2000 [ 10%]  (Warmup)
Iteration:  300 / 2000 [ 15%]  (Warmup)
Iteration:  400 / 2000 [ 20%]  (Warmup)
Iteration:  500 / 2000 [ 25%]  (Warmup)
Iteration:  600 / 2000 [ 30%]  (Warmup)
Iteration:  700 / 2000 [ 35%]  (Warmup)
Iteration:  800 / 2000 [ 40%]  (Warmup)
Iteration:  900 / 2000 [ 45%]  (Warmup)
Iteration: 1000 / 2000 [ 50%]  (Warmup)
Iteration: 1001 / 2000 [ 50%]  (Sampling)
Iteration: 1100 / 2000 [ 55%]  (Sampling)
```

```
Iteration: 1200 / 2000 [ 60%]  (Sampling)
Iteration: 1300 / 2000 [ 65%]  (Sampling)
Iteration: 1400 / 2000 [ 70%]  (Sampling)
Iteration: 1500 / 2000 [ 75%]  (Sampling)
Iteration: 1600 / 2000 [ 80%]  (Sampling)
Iteration: 1700 / 2000 [ 85%]  (Sampling)
Iteration: 1800 / 2000 [ 90%]  (Sampling)
Iteration: 1900 / 2000 [ 95%]  (Sampling)
Iteration: 2000 / 2000 [100%]  (Sampling)

 Elapsed Time: 0.005 seconds (Warm-up)
               0.012 seconds (Sampling)
               0.017 seconds (Total)
```

1.1.2. Interacción desde R Nota que el objeto `modelo` es parte de una clase (OOB):

```
class(modelo)
```

```
[1] "CmdStanModel" "R6"
```

LISTING 3. *Tipo de objeto que regresa la compilación del modelo.*

Con esto tenemos un `objeto` (OOP) con distintos `métodos` que podemos utilizar. Puedes consultar aquí los métodos disponibles de dichos objetos.

Stan code

Method	Description
`$stan_file()`	Return the file path to the Stan program.
`$code()`	Return Stan program as a string.
`$print()`	Print readable version of Stan program.
`$check_syntax()`	Check Stan syntax without having to compile.

Compilation

Method	Description
`$compile()`	Compile Stan program.
`$exe_file()`	Return the file path to the compiled executable.
`$hpp_file()`	Return the file path to the `.hpp` file containing the generated C++ code.
`$save_hpp_file()`	Save the `.hpp` file containing the generated C++ code.

Model fitting

Method	Description
`$sample()`	Run CmdStan's `"sample"` method, return `CmdStanMCMC` object.
`$sample_mpi()`	Run CmdStan's `"sample"` method with MPI, return `CmdStanMCMC` object.
`$optimize()`	Run CmdStan's `"optimize"` method, return `CmdStanMLE` object.
`$variational()`	Run CmdStan's `"variational"` method, return `CmdStanVB` object.
`$generate_quantities()`	Run CmdStan's `"generate_quantities"` method, return `CmdStanGQ` object.

FIGURA 1. *Métodos de objetos de la clase `CmdStanModel`.*

Por ejemplo, tenemos un método que puede generar muestras del `modelo probabilístico` que se definió en el bloque de modelo.

Necesitamos los datos en un formato muy especial (una lista):

```
data.list <- list(Y = 7)
```

> La interacción desde `R` con `Stan` necesita los datos ordenados en `listas con nombres`. En `Python` éstos son `diccionarios`. Ambos, generalizan a archivos en formato `JSON`.

Vamos a darle *play* al botón de la máquina bayesiana:

```
muestras <- modelo$sample(data = data.list,
                          chains = 1,
                          iter=1500,
                          iter_warmup=500,
                          seed=483892929,
                          refresh=500)
```

```
Running MCMC with 1 chain...

Chain 1 Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 1 Iteration:  500 / 2000 [ 25%]  (Warmup)
Chain 1 Iteration:  501 / 2000 [ 25%]  (Sampling)
Chain 1 Iteration: 1000 / 2000 [ 50%]  (Sampling)
Chain 1 Iteration: 1500 / 2000 [ 75%]  (Sampling)
Chain 1 Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 1 finished in 0.0 seconds.
```

LISTING 4. *Resultados de muestreo.*

El resultado es otro objeto:

```
class(muestras)
```

```
[1] "CmdStanMCMC" "CmdStanFit"  "R6"
```

LISTING 5. *Tipo de objeto que regresa la compilación del modelo.*

Donde se pueden explorar los métodos de estos objetos en la documentación.

1.2. Visualizaciones

Podemos grafica trayectorias

> Nota: tuvimos que definir qué parámetros queremos en la visualización. Por *default* incluye un misterioso `lp__` que hace referencia a la evaluación de la log-posterior para cada elemento de la simulación. Adicional, nota (en el código fuente) que la sintaxis para el gráfico utiliza la gramática y las funciones de `ggplot2`.

1.3. Modelos conjugados

Se puede aprovechar que el modelo Beta-Binomial es un modelo conjugado. De tal forma que podemos escribirlo

```
data {
  int<lower = 0, upper = 10> Y;
}
generated quantities {
  real<lower=0, upper=1> theta;
  theta = beta_rng(Y + 2, 10 - Y + 2);
}
```

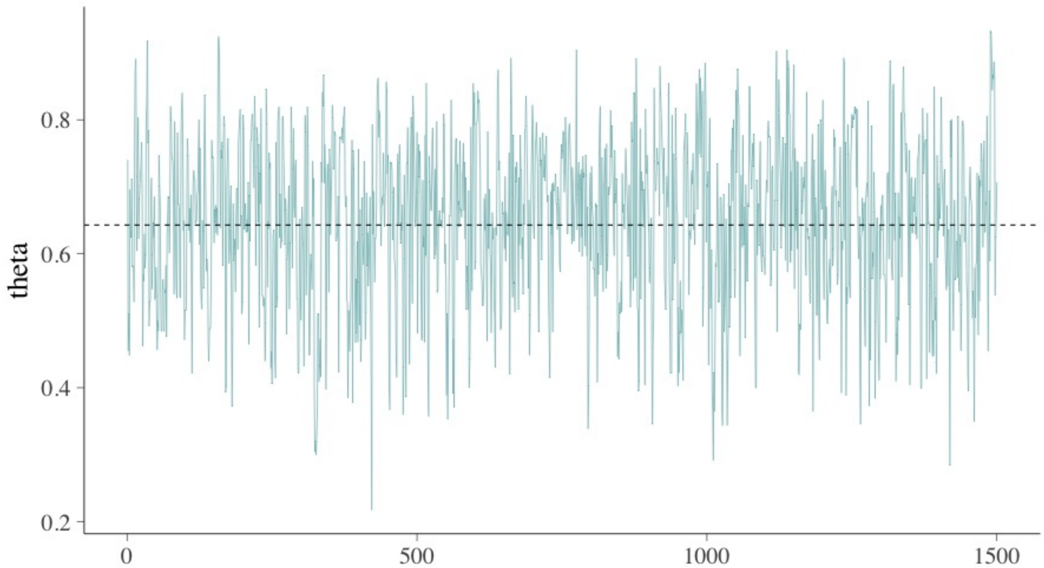


Figura 2. *Trazas (trayectorias) del componente θ en el modelo Beta-Binomial.*

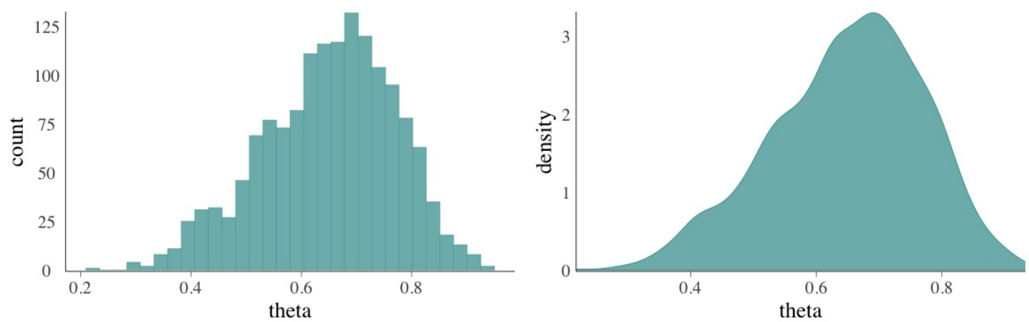


Figura 3. *Histogramas del componente θ en el modelo Beta-Binomial.*

```r
## Modelo BetaBinomial Conjugado -----------------------------------------------
modelos_files <- "modelos/compilados/tutorial"
ruta <- file.path("modelos/tutorial/beta-binomial-conjugado.stan")
modelo <- cmdstan_model(ruta, dir = modelos_files)
```

```r
muestras <- modelo$sample(data   = data.list,
                          chains = 1,
                          iter   = 1500,
                          iter_warmup = 500,
                          seed   = 10101,
                          refresh= 500,
                          fixed_param = TRUE)
```

```
Running MCMC with 1 chain...

Chain 1 Iteration:    1 / 1500 [  0%]  (Sampling)
```

```
Chain 1 Iteration:  500 / 1500 [ 33%]  (Sampling)
Chain 1 Iteration: 1000 / 1500 [ 66%]  (Sampling)
Chain 1 Iteration: 1500 / 1500 [100%]  (Sampling)
Chain 1 finished in 0.0 seconds.
```

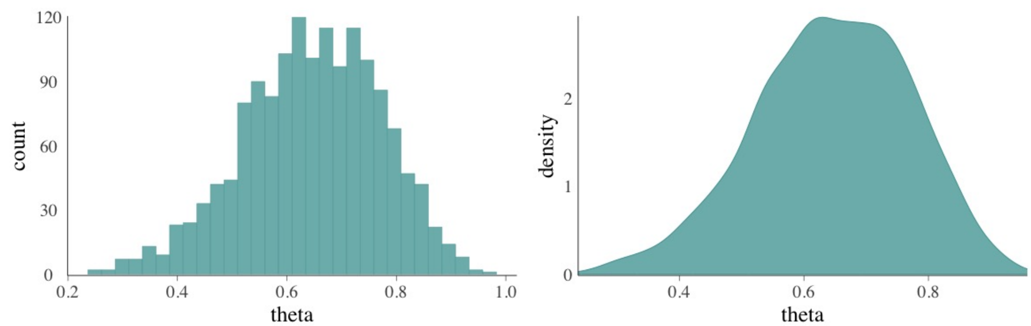


FIGURA 4. *Histogramas del componente θ en el modelo Beta-Binomial.*

1.3.1. Tarea (1) ¿Cómo utilizarías `Stan` para generar números aleatorios de:

1. la distribución previa;
2. la distribución predictiva posterior?

Utiliza el ejemplo Beta-Binomial de arriba para ponerlo en práctica. *Hint*: revisa la documentación del bloque `generated quantities`.

1.3.2. Tarea (2) Repite lo anterior para un modelo Poisson-Gamma. Es decir, para una colección de observaciones $(Y_1, Y_2) = (2, 9)$ donde suponemos que $Y_j \overset{\text{iid}}{\sim} \mathsf{Poisson}(\lambda)$ y $\lambda \sim \mathsf{Exponencial}(3)$.

Hints: Revisa la documentación para definir vectores (en este caso de longitud 2) en el bloque de datos.

1.3.3. Tarea (3) Utiliza el ambiente de `Stan` para encontrar el estimador de Máxima verosimillitud de los dos problemas que hemos trabajado. Es decir, el caso Beta-Binomial y Poisson-Gamma.

2. CASO: ESCUELAS

Utilizaremos los datos de un estudio de desempeño de 8 escuelas ([3, 8]). Los datos consisten en el puntaje promedio de cada escuela `y` y los errores estándar reportados `sigma` la dispersión de los resultados de dicha prueba.

```
## Caso: escuelas
    ------------------------------------------------------------
data <- tibble( id = factor(seq(1, 8)),
                y = c(28, 8, -3, 7, -1, 1, 18, 12),
                sigma = c(15, 10, 16, 11, 9, 11, 10, 18))
```

En este caso se utiliza un modelo normal para los resultados de cada escuela

$$y_j \sim \mathsf{N}(\theta_j, \sigma_j), \qquad j = 1, \ldots, J\,, \tag{1}$$

donde $J = 8$, y θ_j representa el promedio de los alumnos de escuela que no observamos pero del cual tenemos un estimador y_j.

Nota que tenemos J valores distintos para θ_j y σ_j. Dado que esperamos que las escuelas provengan de la misma población de escuelas asumimos que

$$\theta_j \sim \mathsf{N}(\mu, \tau), \qquad j = 1, \ldots, J\,,$$

donde μ representa la media poblacional (el promedio en el sistema escolar) y τ la desviación estándar alrededor de este valor.

Representamos nuestra incertidumbre en estos dos valores por medio de

$$\mu \sim \mathsf{N}(0, 5), \qquad \tau \sim \mathsf{Half\text{-}Cauchy}(0, 5)\,,$$

lo cual representa información poco precisa de estos valores poblacionales.

3. PRIMER MODELO EN STAN

La forma en que escribimos el modelo en `Stan` es de manera generativa (*bottom up*):

$$\mu \sim \mathsf{N}(0, 5)\,, \tag{2a}$$
$$\tau \sim \mathsf{Half\text{-}Cauchy}(0, 5)\,, \tag{2b}$$
$$\theta_j \sim \mathsf{N}(\mu, \tau), \qquad j = 1, \ldots, J\,, \tag{2c}$$
$$y_j \sim \mathsf{N}(\theta_j, \sigma_j), \qquad j = 1, \ldots, J\,. \tag{2d}$$

```
data {
  int<lower=0> J;
  array[J] real y;
  array[J] real<lower=0> sigma;
}
parameters {
  real mu;
  real<lower=0> tau;
  array[J] real theta;
}
model {
  mu ~ normal(0, 5);
  tau ~ cauchy(0, 5);
  theta ~ normal(mu, tau);
  y ~ normal(theta, sigma);
}
```

LISTING 6. *Código del modelo para el desempeño de las escuelas.*

Nota que `sigma` está definida como *parte del conjunto de datos* que el usuario debe de proveer. Aunque es un parámetro en nuestro modelo (verosimilitud) no está sujeto al proceso de inferencia. Por otro lado, nota que la declaración no se hace de manera componente por componente, sino de forma `vectorizada`.

Una vez escrito nuestro modelo, lo podemos compilar utilizando la librería de `cmdstanr`, que es la interface con `Stan` desde `R`.

```
modelos_files <- "modelos/compilados/caso-escuelas"
ruta <- file.path("modelos/caso-escuelas/modelo-escuelas.stan")
modelo <- cmdstan_model(ruta, dir = modelos_files)
```

Los datos que necesita el bloque `data` se pasan como una *lista con nombres*.

```
data_list ← c(data, J = 8)
```

3.1. Simulación

Contra todas las recomendaciones usuales, corramos sólo una cadena corta:

```
muestras ← modelo$sample(data = data_list,
                          chains = 1,
                          iter=700,
                          iter_warmup=500,
                          seed=483892929,
                          refresh=1200)
```

```
Running MCMC with 1 chain...

Chain 1 Iteration:    1 / 1200 [  0%]  (Warmup)
Chain 1 Iteration:  501 / 1200 [ 41%]  (Sampling)
Chain 1 Iteration: 1200 / 1200 [100%]  (Sampling)
Chain 1 finished in 0.1 seconds.
Warning: 53 of 700 (8.0%) transitions ended with a divergence.
See https://mc-stan.org/misc/warnings for details.

Warning: 1 of 1 chains had an E-BFMI less than 0.2.
See https://mc-stan.org/misc/warnings for details.
```

Listing 7. *Resultados del muestreador en el modelo.*

El muestreador en automático nos regresa ciertas alertas las cuales podemos inspeccionar más a fondo con el siguiente comando:

```
muestras$cmdstan_diagnose()
```

```
Processing csv files: /var/folders/lk/4hdvzkhx269df8zc5xmkqgwr0000gn/T/
    Rtmp5LA0na/modelo-escuelas-202307272149-1-462462.csv

Checking sampler transitions treedepth.
Treedepth satisfactory for all transitions.

Checking sampler transitions for divergences.
53 of 700 (7.57%) transitions ended with a divergence.
These divergent transitions indicate that HMC is not fully able to explore the
     posterior distribution.
Try increasing adapt delta closer to 1.
If this doesn't␣remove␣all␣divergences,␣try␣to␣reparameterize␣the␣model.

Checking␣E-BFMI␣-␣sampler␣transitions␣HMC␣potential␣energy.
The␣E-BFMI,␣0.16,␣is␣below␣the␣nominal␣threshold␣of␣0.30␣which␣suggests␣that␣
    HMC␣may␣have␣trouble␣exploring␣the␣target␣distribution.
If␣possible,␣try␣to␣reparameterize␣the␣model.

Effective␣sample␣size␣satisfactory.

The␣following␣parameters␣had␣split␣R-hat␣greater␣than␣1.05:
```

```
␣␣tau,␣theta[1],␣theta[7]
Such␣high␣values␣indicate␣incomplete␣mixing␣and␣biased␣estimation.
You␣should␣consider␣regularizating␣your␣model␣with␣additional␣prior␣
    information␣or␣a␣more␣effective␣parameterization.

Processing␣complete.
```

Listing 8. *Diagnósticos y resumen.*

Notamos que parece ser que tenemos varias transiciones divergentes, algunos parámetros tienen una $\hat{R}$ tienen un valor que excede la referencia de 1.1 (lo veremos más adelante), y parece ser que los estadisticos de energía también presentan problemas.

Podemos inspeccionar el resultado de las simulaciones utilizando:

```
muestras
```

```
variable    mean median   sd   mad     q5   q95 rhat ess_bulk ess_tail
lp__      -11.62 -11.90 8.04 10.77 -24.85  0.36 1.08       13       34
mu          3.98   3.40 3.46  3.45  -1.71  9.71 1.06       56      135
tau         2.87   1.65 2.96  1.90   0.32  8.93 1.12       10       10
theta[1]    5.44   4.01 5.14  4.66  -1.46 14.62 1.10       64      131
theta[2]    4.43   3.35 4.78  4.43  -2.63 12.43 1.05       69      209
theta[3]    3.44   3.34 5.42  4.29  -4.92 11.38 1.10      102      147
theta[4]    4.11   3.40 4.86  4.23  -3.56 11.98 1.11       73      141
theta[5]    3.48   3.18 4.44  3.97  -3.88 10.65 1.08       87      176
theta[6]    3.67   3.64 4.83  4.30  -4.64 11.10 1.11       92      236
theta[7]    5.44   4.14 4.88  4.21  -1.22 13.57 1.10       58       93

# showing 10 of 11 rows (change via 'max_rows' argument or 'cmdstanr_max_rows'
     option)
```

```
muestras$cmdstan_summary()
```

```
Inference for Stan model: modelo_escuelas_model
1 chains: each with iter=(700); warmup=(0); thin=(1); 700 iterations saved.

Warmup took 0.030 seconds
Sampling took 0.046 seconds

                  Mean     MCSE   StdDev       5%     50%     95%    N_Eff   N_Eff
                      /s     R_hat

lp__               -12      2.0      8.0      -25     -12    0.36       16
    357       1.1
accept_stat__     0.76  1.1e-01  3.7e-01  4.6e-16    0.98    1.00  1.1e+01  2.3e
    +02  1.1e+00
stepsize__       0.086      nan  2.8e-17  8.6e-02   0.086   0.086      nan
    nan       nan
treedepth__        3.9  4.1e-01  1.5e+00  1.0e+00     4.0     6.0  1.3e+01  2.9e
    +02  1.1e+00
n_leapfrog__        28  4.2e+00  2.3e+01  3.0e+00      19      63  3.0e+01  6.4e
    +02  1.1e+00
divergent__      0.076  6.0e-02  2.6e-01  0.0e+00    0.00     1.0  1.9e+01  4.2e
    +02  1.1e+00
```

```
energy__            17  2.0e+00  8.5e+00  4.0e+00     17     30  1.7e+01  3.8e+02  1.1e+00

mu                  4.0     0.47      3.5     -1.7    3.4    9.7       55     1199      1.0
tau                 2.9     0.55      3.0     0.32    1.7    8.9       30     643     1.1
theta[1]            5.4     0.60      5.1     -1.6    4.0     15       74     1606      1.1
theta[2]            4.4     0.56      4.8     -2.6    3.4     12       72     1564      1.0
theta[3]            3.4     0.47      5.4     -5.1    3.3     11      130     2831      1.0
theta[4]            4.1     0.54      4.9     -3.6    3.4     12       82     1790      1.0
theta[5]            3.5     0.46      4.4     -4.1    3.2     11       92     2003      1.0
theta[6]            3.7     0.49      4.8     -4.7    3.6     11       99     2146     1.00
theta[7]            5.4     0.59      4.9     -1.2    4.2     14       68     1483      1.1
theta[8]            4.5     0.53      4.9     -3.0    3.6     12       85     1847      1.0

Samples were drawn using hmc with nuts.
For each parameter, N_Eff is a crude measure of effective sample size,
and R_hat is the potential scale reduction factor on split chains (at
convergence, R_hat=1).
```

Listing 9. *Resumen utilizando los métodos de* `CmdStanModel`*.*

Donde además de los resúmenes usuales para nuestros parámetros de interés encontramos resúmenes internos del simulador (los veremos mas adelante).

3.2. Alternativas: `Rstan`

Podemos utilizar las funciones de `RStan` (otra interfase con `Stan` desde `R`) para visualizar los resúmenes de manera alternativa.

```
## Ejemplo de ócdigo utilizando Rstan
stanfit ← rstan::read_stan_csv(muestras$output_files())
stanfit
```

```
Inference for Stan model: modelo-escuelas-202202231948-1-817561.
1 chains, each with iter=1200; warmup=500; thin=1;
post-warmup draws per chain=700, total post-warmup draws=700.

          mean se_mean  sd  2.5%   25%  50%  75% 97.5% n_eff Rhat
mu         4.0    0.47 3.5 -2.42  1.66  3.4  6.6  11.1    55  1.0
tau        2.9    0.55 3.0  0.32  0.59  1.6  4.3  11.1    29  1.1
theta[1]   5.4    0.60 5.1 -3.50  2.50  4.0  8.4  17.2    73  1.1
theta[2]   4.4    0.57 4.8 -3.99  1.62  3.4  7.5  14.3    71  1.0
theta[3]   3.4    0.48 5.4 -8.36  0.83  3.3  6.7  14.5   129  1.0
theta[4]   4.1    0.54 4.9 -5.79  1.39  3.4  7.3  13.6    82  1.0
theta[5]   3.5    0.46 4.4 -6.08  1.16  3.2  6.6  11.8    91  1.0
theta[6]   3.7    0.49 4.8 -6.97  1.04  3.6  7.0  12.7    98  1.0
theta[7]   5.4    0.59 4.9 -2.64  2.65  4.1  8.1  16.7    67  1.1
theta[8]   4.5    0.53 4.9 -4.63  1.84  3.6  7.6  14.5    84  1.0
```

```
lp__      -11.6    2.01 8.0 -25.98 -18.30 -11.9 -3.8   1.4    16  1.1

Samples were drawn using NUTS(diag_e) at Wed Feb 23 19:48:39 2022.
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).
```

Listing 10. *Resumen obtenido con librería de* `Rstan`*.*

3.3. Consulta de resultados de muestreo

En caso de necesitarlo podemos extraer las muestras en una tabla para poder procesarlas y generar visualizaciones. Por ejemplo, un gráfico de traza con τ que es el parámetro donde más problemas parecemos tener.

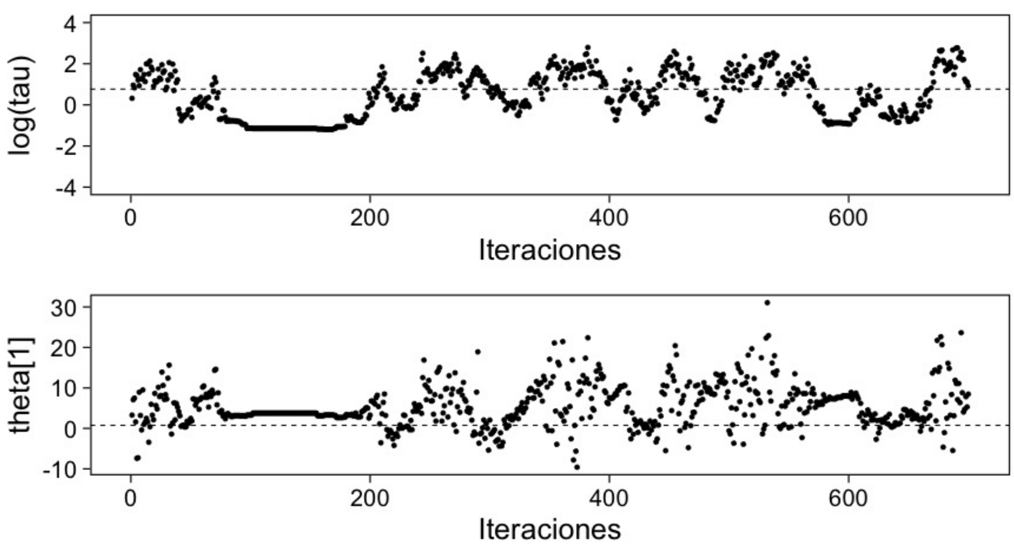


Figura 5. *Trayectorías de las muestras del modelo para los componentes* $\log \tau$ *y* θ_1*.*

Claramente no podemos afirmar que el muestreador está explorando bien la posterior. Hay correlaciones muy altas. Si usáramos la media acumulada no seríamos capaces de diagnosticar estos problemas.

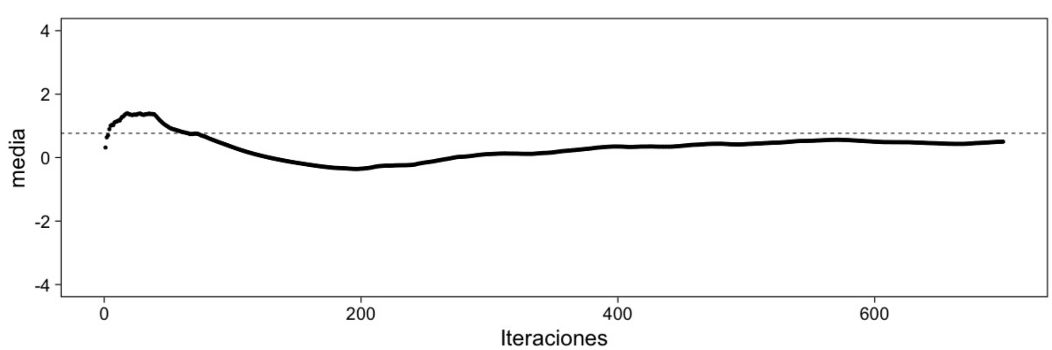


Figura 6. *Media acumulada de* $\log \tau$*.*

Utilizar gráficos de dispersión bivariados nos ayuda a identificar mejor el problema. En color salmón apuntamos las muestras con transiciones *divergentes* (mas adelante lo explicaremos).

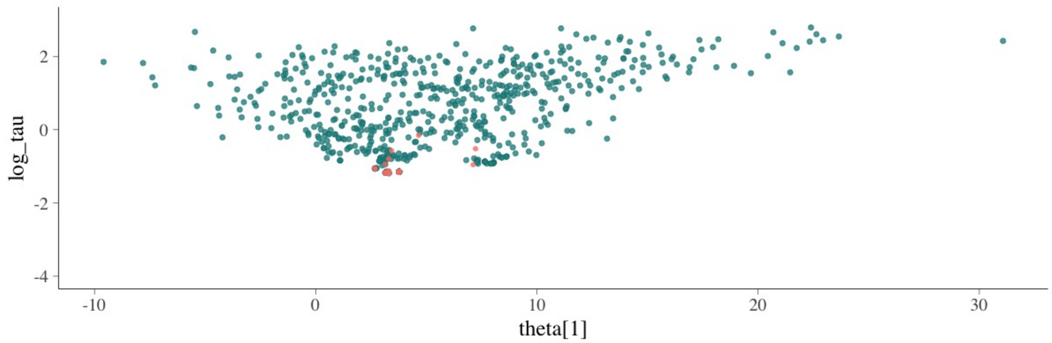


FIGURA 7. *Gráfico de dispersión. Muestras en color salmón representan simulaciones* problemáticas.

Otra visualización muy conocida es la de coordenadas paralelas. En este tipo de gráficos podemos observar de manera simultánea ciertos patrones en todos los componentes.

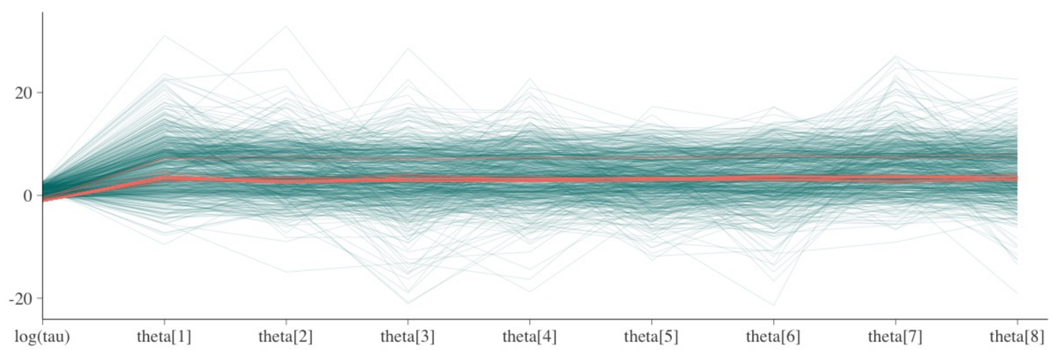


FIGURA 8. *Gráfico de coordenadas paralelas. Permiten* conectar *los distintos componentes de un vector. Color salmón representa simulaciones* problemáticas.

Y por último, también podemos explorar la autocorrelación de la cadena.

3.4. Generando mas simulaciones

Hasta ahora los resultados parecen no ser buenos. Tenemos muestras con transiciones *divergentes* y una *correlación muy alta* entre las muestras. Podríamos aumentar el número de simulaciones con la esperanza que esto permita una mejor exploracion de la posterior:

```
muestras ← modelo$sample(data        = data_list,
                          chains      = 1,
                          iter        = 5000,
                          iter_warmup = 5000,
                          seed        = 483892929,
                          refresh     = 10000)
```

Como vemos, seguimos teniendo problemas con la exploración del espacio parametral (donde está definida nuestra distribución de θ) y tenemos dificultades en explorar esa zona con τ pequeña. Esto lo confirmamos en la siguiente gráfica.

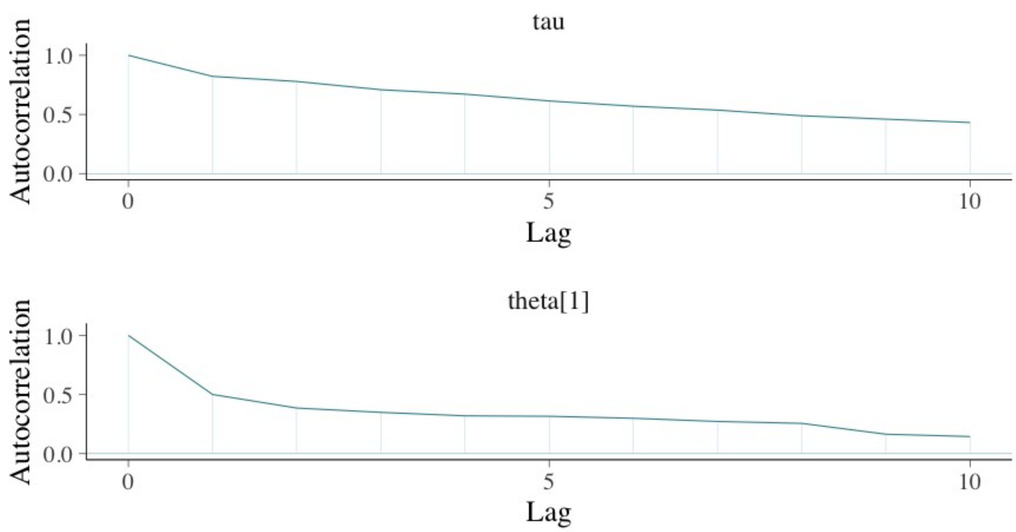


Figura 9. *Autocorrelaciones en las simulaciones.*

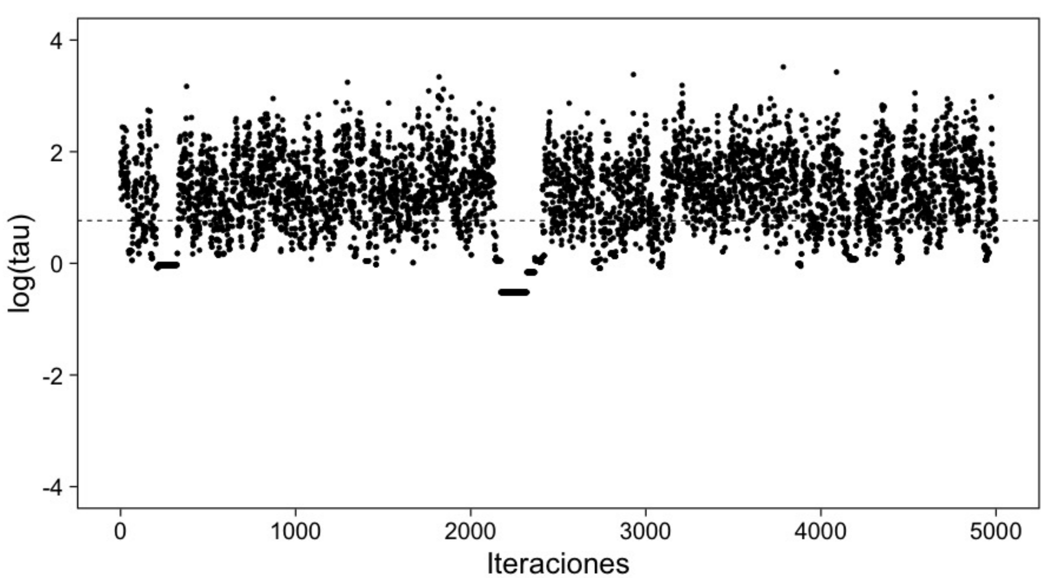


Figura 10. *Trayectorías de simulaciones.*

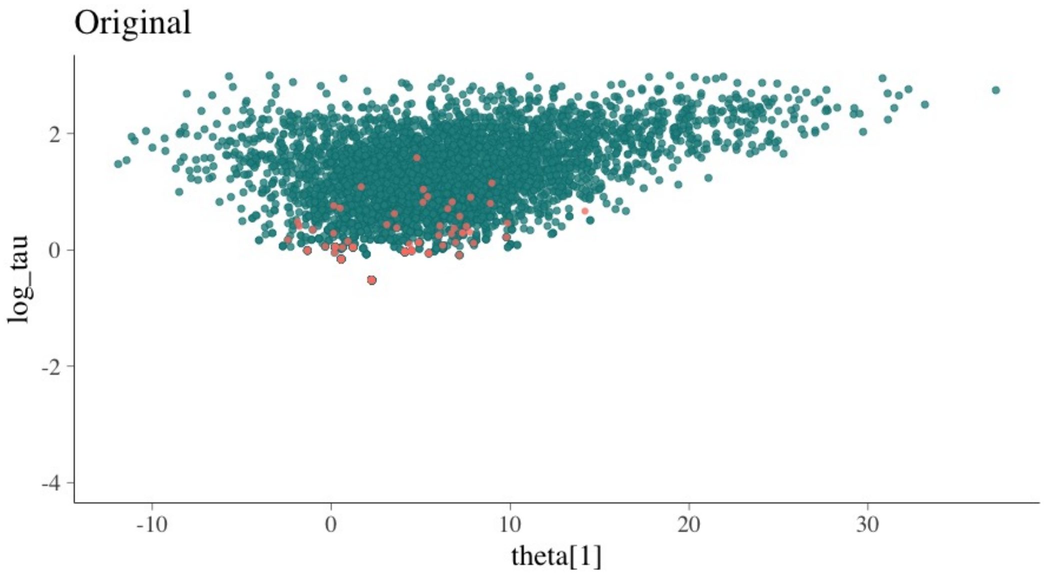


FIGURA 11. *Gráficos de dispersión.*

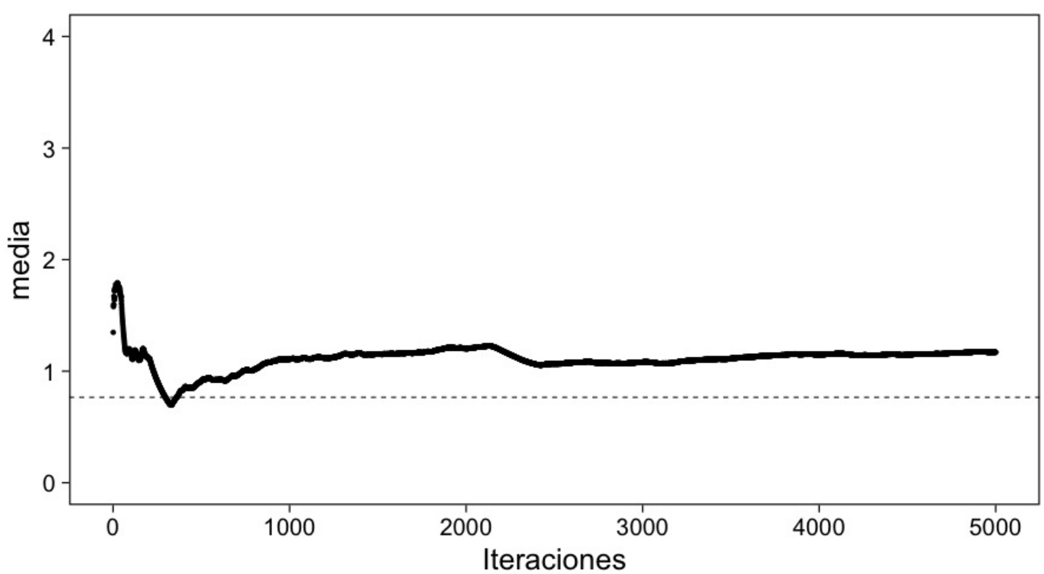


FIGURA 12. *Media acumulada de* $\log \tau$.

3.5. Haciendo tweaks en el simulador

Podríamos correr una cadena con algunas opciones que permitan la exploracion mas segura de la distribución.

```
muestras ← modelo$sample(data        = data_list,
                          chains      = 1,
                          iter        = 5000,
                          iter_warmup = 5000,
                          seed        = 483892929,
                          refresh     = 10000,
                          adapt_delta = .90)
```

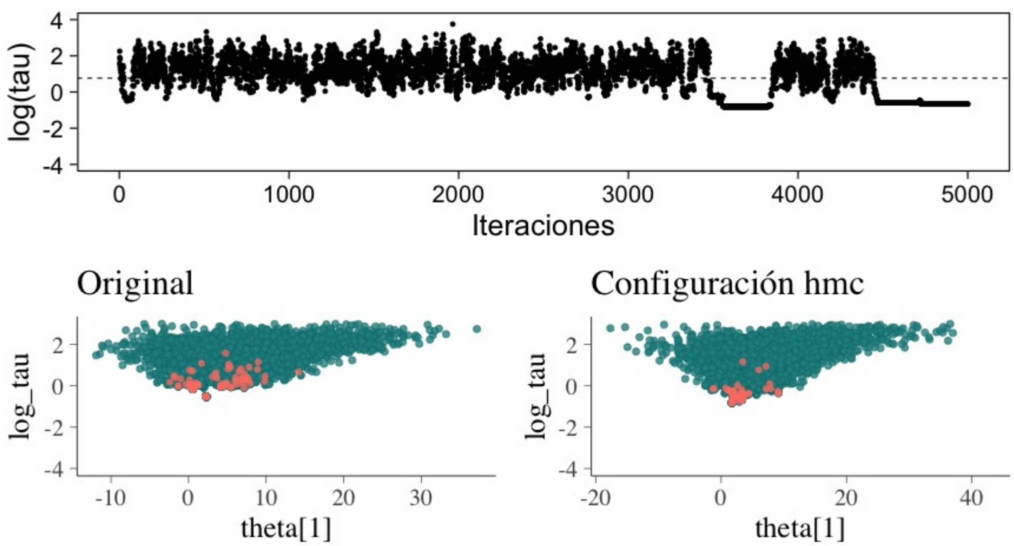


FIGURA 13. *Gráficos de comparación.*

4. CAMBIANDO LIGERAMENTE EL MODELO

Tener cuidado en la simulación del sistema Hamiltoniano nos ayuda hasta cierto punto. Seguimos teniendo problemas y no hay garantías que nuestra simulación y nuestros estimadores Monte Carlo no estén sesgados.

Esta situación es muy común en *modelos jerárquicos*. El cual hemos definido como

$$\mu \sim \mathsf{N}(0,5)\,, \tag{3a}$$
$$\tau \sim \textsf{Half-Cauchy}(0,5)\,, \tag{3b}$$
$$\theta_j \sim \mathsf{N}(\mu,\tau), \qquad j = 1,\ldots,J\,, \tag{3c}$$
$$y_j \sim \mathsf{N}(\theta_j,\sigma_j), \qquad j = 1,\ldots,J\,. \tag{3d}$$

El problema es la geometría de la distribución posterior. La ventaja es que existe una solución sencilla para hacer el problema de muestreo mas sencillo. Esto es al escribir el modelo en términos de una variable auxiliar:

$$
\begin{aligned}
\mu &\sim \mathsf{N}(0,5)\,, && (4a)\\
\tau &\sim \textsf{Half-Cauchy}(0,5)\,, && (4b)\\
\tilde{\theta}_j &\sim \mathsf{N}(0,1), \qquad j=1,\ldots,J\,, && (4c)\\
\theta_j &= \mu + \tau\cdot\tilde{\theta}_j, \qquad j=1,\ldots,J\,, && (4d)\\
y_j &\sim \mathsf{N}(\theta_j,\sigma_j), \qquad j=1,\ldots,J\,. && (4e)
\end{aligned}
$$

El modelo en `Stan` es muy parecido. La nomenclatura que se utiliza es: **modelo centrado** para el primero, y para la reparametrización presentada en la ecuación de arriba nos referimos a un **modelo no centrado**.

```
data {
  int<lower=0> J;
  array[J] real y;
  array[J] real<lower=0> sigma;
}

parameters {
  real mu;
  real<lower=0> tau;
  array[J] real theta_tilde;
}

transformed parameters {
  array[J] real theta;
  for (j in 1:J)
    theta[j] = mu + tau * theta_tilde[j];
}

model {
  mu ~ normal(0, 5);
  tau ~ cauchy(0, 5);
  theta_tilde ~ normal(0, 1);
  y ~ normal(theta, sigma);
}
```

> Nota que la definición de nuevos parametros se hace desde el bloque `transformed parameters` en donde la asignación se ejecuta componente por componente mientras que la definición del modelo de probabilidad conjunto se puede hacer de manera vectorizada.

Igual que antes lo necesitamos compilar para hacerlo un objeto ejecutable desde `R`.

```
## Cambio de óparametrizacin
    -----------------------------------------------------
ruta_ncp <- file.path("modelos/caso-escuelas/modelo-escuelas-ncp.stan")
modelo_ncp <- cmdstan_model(ruta_ncp, dir = modelos_files)
```

Muestreamos de la posterior

```
muestras_ncp <-
  modelo_ncp$sample(
             data = data_list,
             chains = 1,
```

```
                iter=5000,
                iter_warmup=5000,
                seed=483892929,
                refresh=10000
              )
```

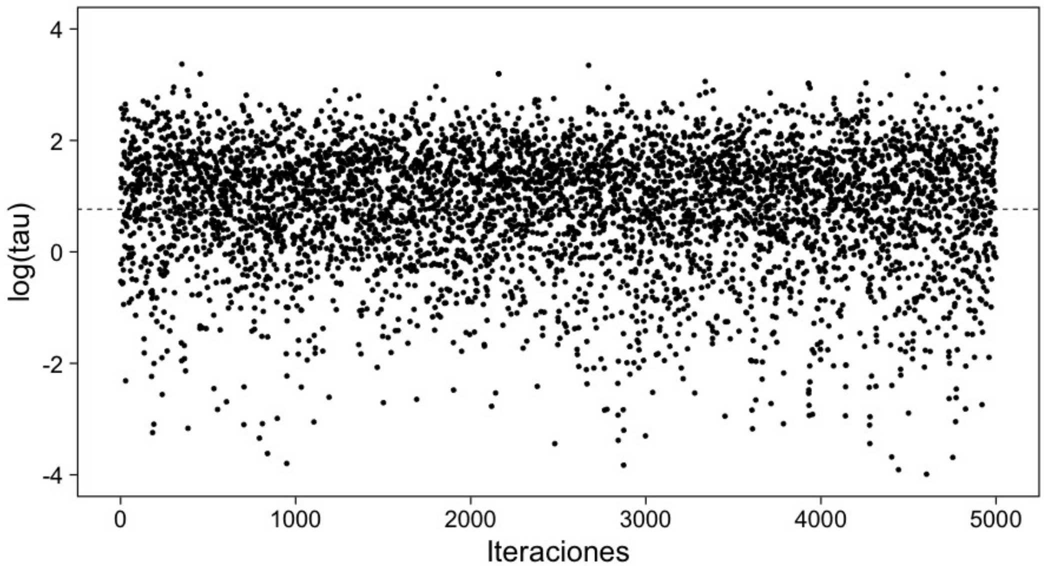
```
Running MCMC with 1 chain...

Chain 1 Iteration:    1 / 10000 [  0%]  (Warmup)
Chain 1 Iteration: 5001 / 10000 [ 50%]  (Sampling)
Chain 1 Iteration: 10000 / 10000 [100%]  (Sampling)
Chain 1 finished in 0.3 seconds.
```

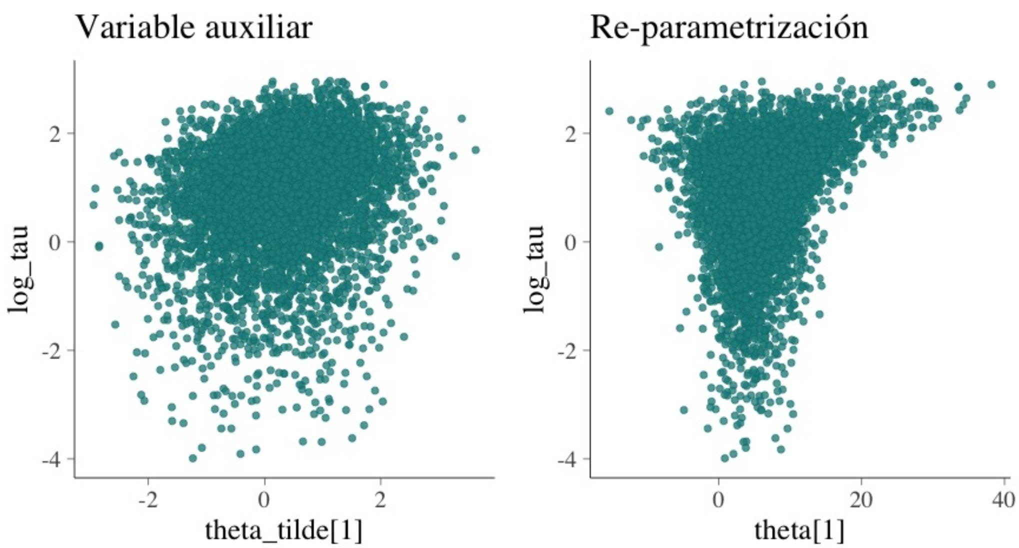
```
print(muestras_ncp, max_rows = 19)
```

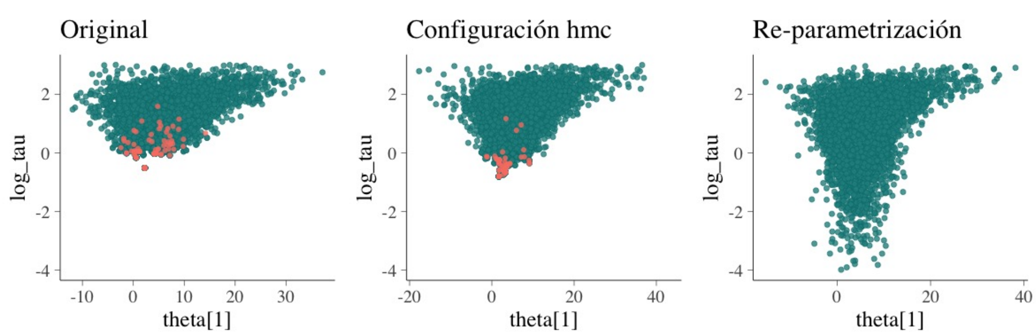
```
      variable  mean median   sd  mad     q5   q95 rhat ess_bulk ess_tail
lp__           -6.99  -6.70 2.30 2.22 -11.09 -3.77 1.00     2199     2984
mu              4.33   4.30 3.38 3.28  -1.17  9.99 1.00     4660     3210
tau             3.60   2.78 3.20 2.55   0.27  9.84 1.00     3321     2377
theta_tilde[1]  0.31   0.32 0.99 1.02  -1.33  1.91 1.00     5289     3976
theta_tilde[2]  0.10   0.11 0.95 0.93  -1.45  1.65 1.00     5112     3495
theta_tilde[3] -0.08  -0.10 0.97 0.97  -1.67  1.52 1.00     4731     3522
theta_tilde[4]  0.07   0.06 0.93 0.95  -1.45  1.59 1.00     5773     3865
theta_tilde[5] -0.16  -0.17 0.93 0.94  -1.67  1.38 1.00     5730     4068
theta_tilde[6] -0.08  -0.08 0.94 0.94  -1.62  1.47 1.00     5664     3710
theta_tilde[7]  0.37   0.39 0.97 0.96  -1.27  1.92 1.00     4720     3688
theta_tilde[8]  0.09   0.10 0.99 1.02  -1.53  1.70 1.00     5050     3175
theta[1]        6.10   5.52 5.60 4.71  -1.83 16.12 1.00     4807     3956
theta[2]        4.89   4.69 4.68 4.25  -2.37 12.74 1.00     4901     3803
theta[3]        3.88   4.01 5.35 4.48  -4.91 12.00 1.00     4777     3577
theta[4]        4.74   4.63 4.81 4.41  -2.88 12.63 1.00     5645     4062
theta[5]        3.55   3.71 4.80 4.27  -4.61 11.00 1.00     4982     4180
theta[6]        3.88   4.04 4.97 4.36  -4.62 11.63 1.00     5494     4553
theta[7]        6.29   5.79 5.16 4.45  -1.10 15.61 1.00     5017     3604
theta[8]        4.87   4.70 5.35 4.51  -3.34 13.49 1.00     4872     3874
```

Si graficamos la dispersión de τ ($\log \tau$), vemos un mejor comportamiento (del cual ya teníamos indicios por los diagnósticos del modelo).



Si regresamos a los gráficos de dispersión para verificar que se hayan resuelto los problemas observamos lo siguiente:





Como lo hemos mencionado antes. Este caso ilustra uno de los casos de uso mas conocidos de la inferencia Bayesiana, `modelos jerárquicos`. Estos modelos surgen en diversas aplicaciones, como regresión, análisis de series de tiempo, datos estratificados, etc.

5. REGULARIZACIÓN BAYESIANA

Otro caso de uso bastante común y con el cual *podrían* estar altamente familiarizados es con el concepto de `regularización`. Por ejemplo, en modelos predictivos donde buscamos una regla de asociación $y = f_\theta(x)$. En dichos modelos θ son los parámetros que no conocemos y que ajustamos minimizando una función de pérdida *adecuada*

$$\hat{\theta} = \arg\min_{\theta} \mathcal{J}(y, f_\theta(x))\,. \tag{5}$$

El problema de optimización está usualmente `mal formulado` en el sentido de que pequeñas perturbaciones en el conjunto de datos utilizado para *entrenar* lleva a soluciones radicalmente distintas. En este contexto se busca `regularizar` el problema utilizando una función que permita restringir la solución y de esta manera tener soluciones `estables`. Esto lo formulamos como

$$\hat{\theta}_R = \arg\min_{\theta} \left(\mathcal{J}(y, f_\theta(x)) + R(\theta)\right)\,. \tag{6}$$

Esto es bastante usual en la solución de problemas inversos ([6, 9]) y la estimación de modelos predictivos por medio de Ridge o Lasso ([4]). Por ejemplo, se pueden considerar regularizadores como penalizaciones en **norma 2** (`Ridge`, $\|\theta\|_2^2 = \sum (\theta_i)^2$) o **norma 1** (`Lasso`, $\|\theta\|_1 = \sum |\theta_i|$).

5.1. Formulación probabilística

En términos probabilísticos esto correspondería a plantear un modelo

$$\pi(\theta|y) \propto \exp\left(-\mathcal{J}(y, f_\theta(x))\right) \exp\left(-R(\theta)\right)\,. \tag{7}$$

Por ejemplo, considerar `regresión Ridge` implica considerar un modelo `Gaussiano` para la `verosimilitud` y un modelo Gaussiano para la `previa`.

Una variable aleatoria Gaussiana tiene conexiones interesantes con la `descomposición espectral` de una señal (descomposición en valores singulares y series de Fourier). Si pensamos que en un problema de regresión queremos estimar coeficientes. La solución sin restricciones nos puede dar algunos coeficientes con `errores estándar muy altos` y en consecuencia `estadísticamente no significativos` (alta varianza y centrados en cero). La regularización elimina la alta variabilidad (las frecuencia altas de una señal) y rápidamente centra los valores de aquellos valores alrededor del cero para tener una señal con una frecuencia mas *suave*. ¡La conexión la podemos trazar a los coeficientes de Fourier ([1])!

La solución de este problema de optimización se traduce en encontrar el punto `máximo posterior`.

Ahora, el problema es que tanto para Ridge (previa Gaussiana) como para LASSO (previa `doble-exponencial` o Laplace), la *moda* –el punto que maximiza la posterior– es muy distinto de lo que nos darían simulaciones de ese modelo.

En el contexto Bayesiano nos interesaría poder utilizar una distribución previa de la cual podamos extraer muestras donde algunos componentes son cero. Con este propósito se han estudiado y propuesto previas de la familia `horseshoe` (presiento que es un modismo finlandés) [7].

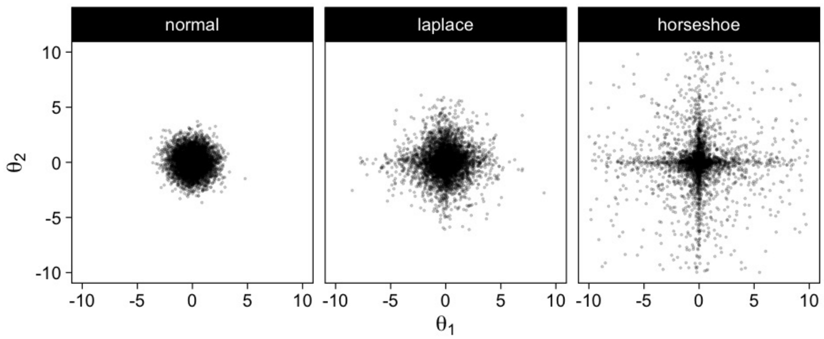


FIGURA 14. *Distintas previas y efectos de regularización.*

5.2. Regularización en regresión (diabetes)

Veamos lo siguiente para comparar los distintos modelos probabilísticos (Normal, Laplace, Horseshoe).

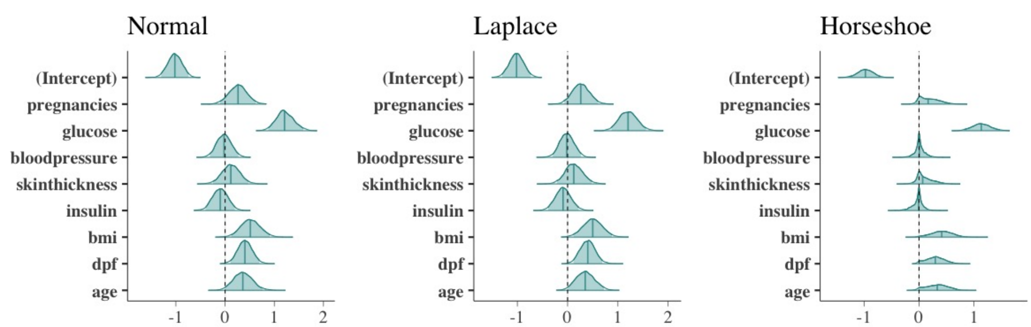


FIGURA 15. *Ajuste posterior bajo distinas previas.*

5.3. Regularización en regresión (carros)

Notemos como el modelo tiene dos zonas de alta probabilidad.

5.4. Regularización y previas

```

library("rstanarm")
x ← seq(-2,2,1)
y ← c(50, 44, 50, 47, 56)
sexratio ← tibble(x, y)
```

```
fit ← lm(y ∼ x, data = sexratio)
fit ▷ broom::tidy()
fit ▷ broom::glance() ▷ select(1:5)
```

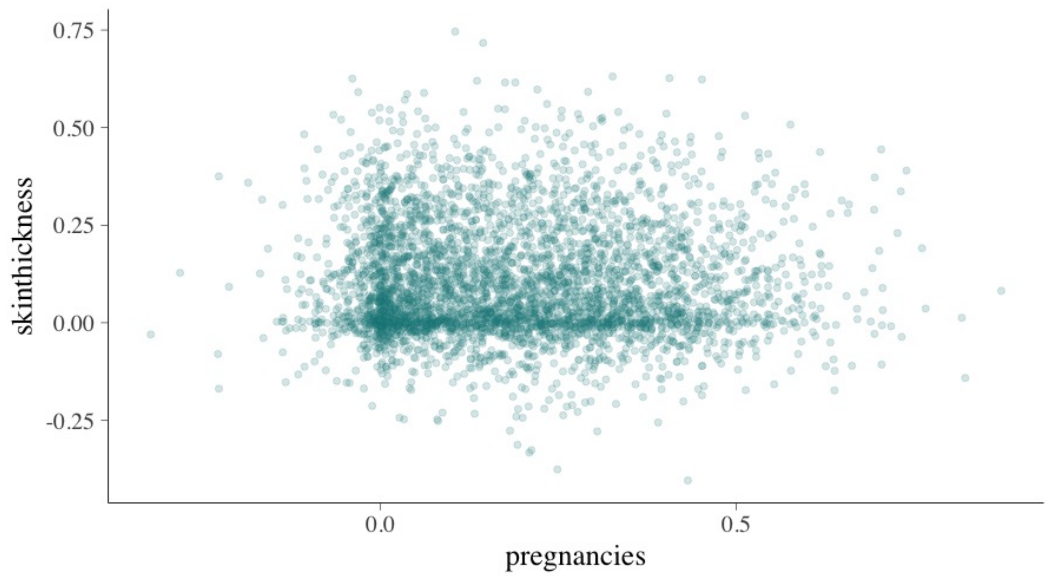


FIGURA 16. *Efecto de la regularización en un par de coeficientes.*

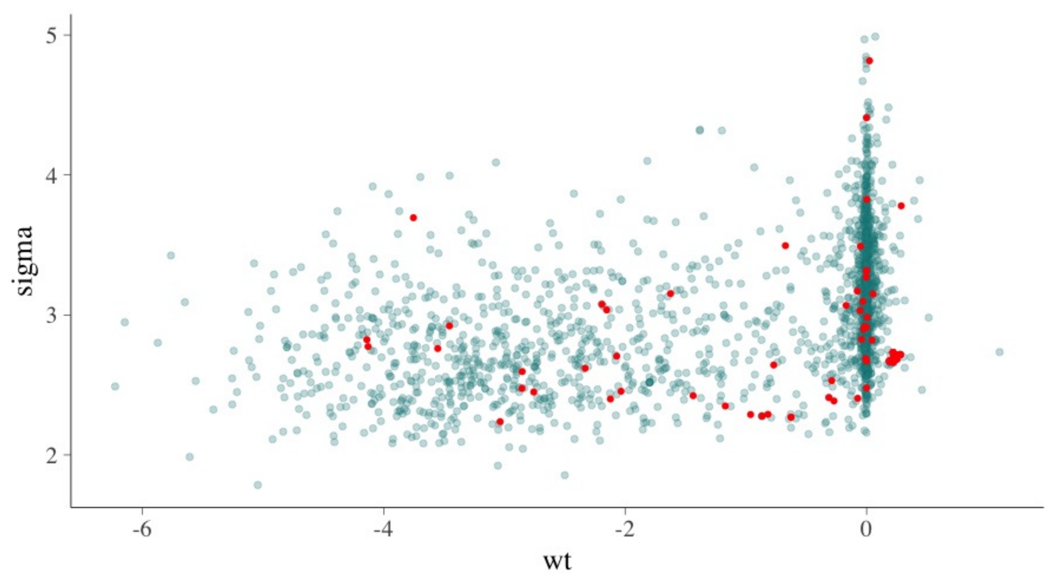
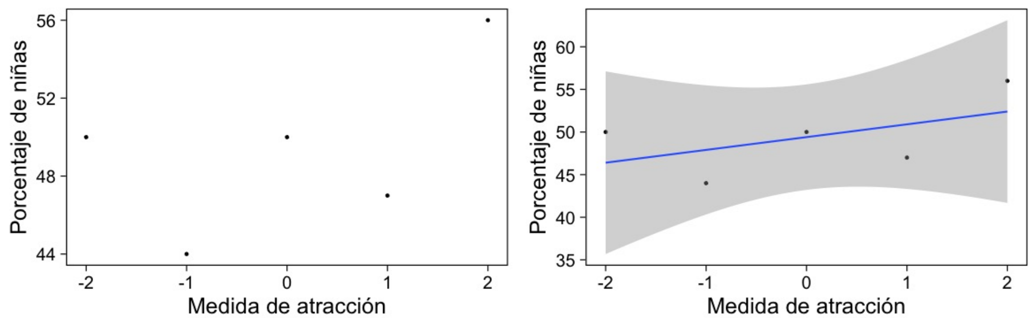


FIGURA 17. *Efecto de regularización en dos parámetros de un modelo.*

```
# A tibble: 2 × 5
  term        estimate std.error statistic  p.value
  <chr>          <dbl>     <dbl>     <dbl>    <dbl>
1 (Intercept)     49.4      1.94     25.4  0.000134
2 x                1.50     1.37      1.09 0.355
# A tibble: 1 × 5
  r.squared adj.r.squared sigma statistic p.value
      <dbl>         <dbl> <dbl>     <dbl>   <dbl>
1     0.284        0.0455  4.35      1.19   0.355
```




```
fit_post <- stan_glm(y ~ x, data = sexratio,
                     prior = normal(0, 0.2),
                     prior_intercept = normal(48.8, 0.5),
                     refresh = 0)
prior_summary(fit_post)
```

```
Priors for model 'fit_post'
------
Intercept (after predictors centered)
 ~ normal(location = 49, scale = 0.5)

Coefficients
 ~ normal(location = 0, scale = 0.2)

Auxiliary (sigma)
  Specified prior:
    ~ exponential(rate = 1)
  Adjusted prior:
    ~ exponential(rate = 0.22)
------
See help('prior_summary.stanreg') for more details
```

```
print(fit_post)
```

```
stan_glm
 family:       gaussian [identity]
 formula:      y ~ x
 observations: 5
 predictors:   2
------
```

```
            Median MAD_SD
(Intercept) 48.8   0.5
x            0.0   0.2

Auxiliary parameter(s):
      Median MAD_SD
sigma 4.3    1.3

------
* For help interpreting the printed output see ?print.stanreg
* For info on the priors used see ?prior_summary.stanreg
```

```
fit_default <- stan_glm(y ~ x, data = sexratio, refresh = 0)
prior_summary(fit_default)
```

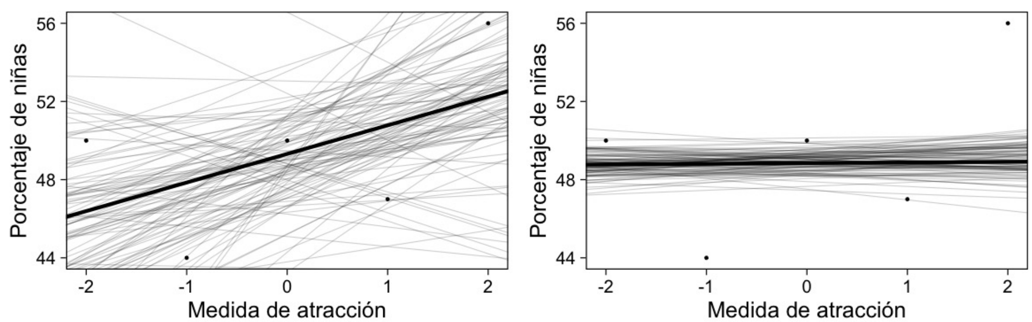


FIGURA 18. *Incorporar información en la previa permite regularizar el problema.*

REFERENCIAS

[1] J. B. J. Baron Fourier. The analytical theory of heat. *Cambridge*, 1878. 20

[2] B. Carpenter, A. Gelman, M. D. Hoffman, D. Lee, B. Goodrich, M. Betancourt, M. Brubaker, J. Guo, P. Li, and A. Riddell. Stan: a probabilistic programming language. *Journal of Statistical Software*, 76(1): nil, 2017. . URL `https://doi.org/10.18637/jss.v076.i01`. 1

[3] A. Gelman, J. B. Carlin, H. S. Stern, D. B. Dunson, A. Vehtari, and D. B. Rubin. *Bayesian Data Analysis*, volume 2. CRC press Boca Raton, FL, 2014. 7

[4] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York, New York, NY, 2009. ISBN 978-0-387-84857-0 978-0-387-84858-7. . 20

[5] A. Johnson, M. Ott, and M. Dogucu. *Bayes Rules! An Introduction to Applied Bayesian Modeling*. CRC Press, 2021. 1

[6] J. Kaipio and E. Somersalo. *Statistical and Computational Inverse Problems*. Number v. 160 in Applied Mathematical Sciences. Springer, New York, 2005. ISBN 978-0-387-22073-4. 20

[7] J. Piironen and A. Vehtari. Sparsity information and regularization in the horseshoe and other shrinkage priors. *Electronic Journal of Statistics*, 11(2), jan 2017. ISSN 1935-7524. . 21

[8] D. B. Rubin. Estimation in Parallel Randomized Experiments. *Journal of Educational Statistics*, 6(4): 377–401, 1981. ISSN 0362-9791. . 7

[9] A. Tarantola. *Inverse Problem Theory and Methods for Model Parameter Estimation*. Society for Industrial and Applied Mathematics, jan 2005. ISBN 978-0-89871-572-9 978-0-89871-792-1. . 20