EST-24107: Simulación

Profesor: Alfredo Garbuno Iñigo — Otoño, 2022 — *Software* de muestreo (intro).
Objetivo: Este sesión está pensada para ver en *acción* alguno de los paquetes de recién creación y versatilidad para realizar modelos de muestreo por cadenas de Markov para realizar estimaciones Monte Carlo.
Lectura recomendada: Los tutoriales introductorios para los paquetes de *software* son muy buenos; tanto el de `Stan` [1] como el de `PyMC` [2].

1. EL PAQUETE LEARNBAYES

Ejemplo tomado de las viñetas de la librería.

```
library(LearnBayes)
minmaxpost <- function(theta, data){
  mu <- theta[1]
  sigma <- exp(theta[2])
  dnorm(data$min, mu, sigma, log = TRUE) +
    dnorm(data$max, mu, sigma, log = TRUE) +
    ((data$n - 2) * log(pnorm(data$max, mu, sigma) - pnorm(data$min, mu, sigma
      )))
}
```

1.1. Aproximación Normal (de Laplace)

```
data <- list(n = 10, min = 52, max = 84)
fit  <- laplace(minmaxpost, c(70, 2), data)
fit
```

```
$mode
[1] 68.000  2.298

$var
             [,1]        [,2]
[1,]  1.921e+01 -1.901e-06
[2,] -1.901e-06  6.032e-02

$int
[1] -8.02

$converge
[1] TRUE
```

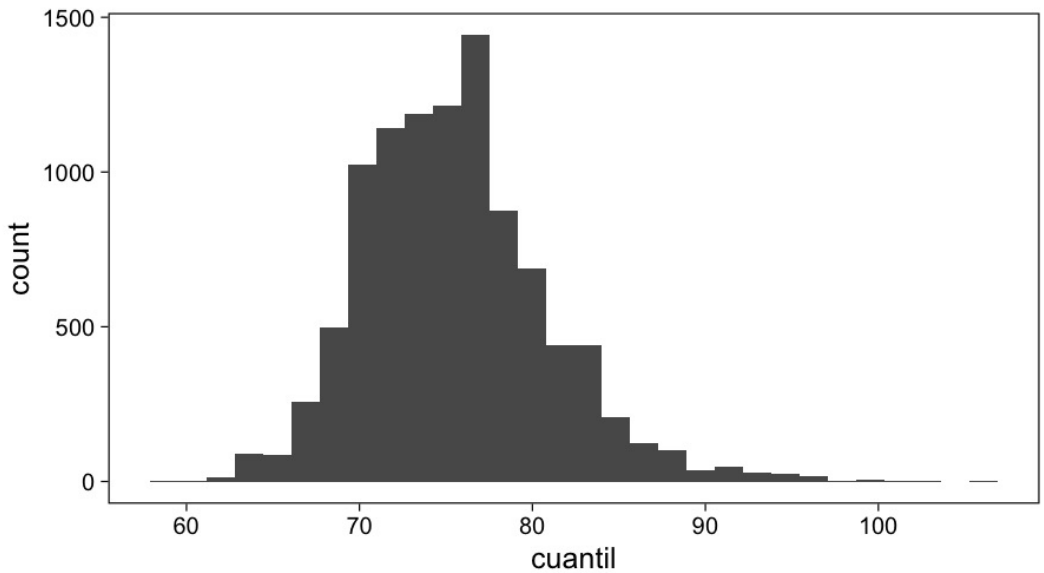
1.2. Muestreo por cadenas de Markov

```
mcmc.fit ← rwmetrop(minmaxpost,
                    list(var = fit$v, scale = 3),
                    c(70, 2),
                    10000,
                    data)
```

```
mcmc.fit$accept
```

```
[1] 0.1735
```

1.2.1. Estimación Monte Carlo



2. USANDO STAN DESDE R

```
data {
  real xmin;
  real xmax;
  int N;
}
parameters {
  real mu;
  real log_sigma;
}
transformed parameters {
  real sigma = exp(log_sigma);
}
model {
  target += normal_lpdf(xmin | mu, sigma);
  target += normal_lpdf(xmax | mu, sigma);
  target += (N-2) * log(normal_cdf(xmax | mu, sigma) - normal_cdf(xmin | mu,
     sigma));
}
```

```
muestras ← modelo$sample(data = list(N = 10, xmin = 52, xmax = 84),
                chains = 4,
                iter = 1500,
                iter_warmup = 500,
                seed = 108727,
                refresh = 500)
```

```
muestras$cmdstan_summary()
```

```
Inference for Stan model: minmax_model
4 chains: each with iter=(1500,1500,1500,1500); warmup=(0,0,0,0); thin
    =(1,1,1,1); 6000 iterations saved.

Warmup took (0.0070, 0.0070, 0.0070, 0.0060) seconds, 0.027 seconds total
Sampling took (0.026, 0.022, 0.024, 0.023) seconds, 0.095 seconds total

                Mean     MCSE  StdDev     5%    50%    95%    N_Eff  N_Eff/s    R
                    _hat

lp__             -11  2.3e-02     1.1    -13    -11  -10.0     2131    22435
         1.0
accept_stat__   0.92  8.8e-03    0.11   0.72   0.96    1.0  1.5e+02  1.6e+03  1.0
    e+00
stepsize__      0.82  6.9e-02   0.098   0.74   0.79   0.99  2.0e+00  2.1e+01  2.4
    e+13
treedepth__      2.1  1.2e-01    0.64    1.0    2.0    3.0  3.0e+01  3.2e+02  1.0
    e+00
n_leapfrog__     4.2  3.6e-01     2.0    1.0    3.0    7.0  3.2e+01  3.3e+02  1.0
    e+00
divergent__     0.00      nan    0.00   0.00   0.00   0.00      nan      nan
         nan
energy__          12  3.1e-02     1.5     10     12     15  2.1e+03  2.3e+04  1.0
    e+00

mu                68  7.8e-02     4.7     60     68     76     3664    38567
    1.00
log_sigma        2.4  4.6e-03    0.27    2.0    2.4    2.8     3521    37068
         1.0
sigma             11  5.9e-02     3.3    7.1     11     17     3063    32243
         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).
```

```
muestras
```

```
 variable   mean median   sd  mad     q5   q95 rhat ess_bulk ess_tail
lp__      -11.01 -10.68 1.07 0.77 -13.20 -9.99 1.00     3886     4639
mu         68.12  68.18 4.68 4.47  60.46 75.65 1.00     6207     5125
log_sigma   2.38   2.36 0.27 0.27   1.96  2.86 1.00     6525     4679
sigma      11.22  10.60 3.31 2.80   7.13 17.45 1.00     6525     4679
```

```
muestras$draws(format = "df") ▷
  pivot_longer(cols = 2:4, names_to = "parameter") ▷
  group_by(parameter) ▷
  summarise(media = mean(value), std.dev = sd(value), error.mc = std.dev/(n())
      , samples = n())
```

3. USANDO PYMC

```
import aesara.tensor as at
import arviz as az
import matplotlib.pyplot as plt
import numpy as np
import pymc as pm
import scipy.stats as stats

RANDOM_SEED = 108727
rng = np.random.default_rng(RANDOM_SEED)
```

```
def minmaxpost(base, *args):
    loglik = pm.logp(base, 52) + pm.logp(base, 84) + (10 - 2) * \
             at.log(at.exp(pm.logcdf(base, 84)) - at.exp(pm.logcdf(base, 52)))
    return loglik
```

```
with pm.Model() as model:
    mu=pm.Normal("mu",␣0,␣100);
␣␣␣␣sigma=pm.HalfNormal("sigma",␣100);
␣␣␣␣base=pm.Normal("observations",␣mu,␣sigma)
␣␣␣␣like=pm.Potential("likelihood",␣minmaxpost(base))

␣␣␣␣idata=pm.sample(1500,␣progressbar␣=␣False)
```

```
az.summary(idata)
```

```
                mean       sd   hdi_3%  hdi_97%  mcse_mean  mcse_sd  ess_bulk
                  ess_tail  r_hat
mu            67.773    4.847   58.735   76.735      0.079    0.056    3765.0
      3481.0     1.0
observations  67.952   12.984   42.344   91.496      0.213    0.151    3766.0
      3541.0     1.0
sigma         12.015    3.621    6.305   18.594      0.063    0.045    3199.0
      2930.0     1.0
```

REFERENCIAS

[1] 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

[2] J. Salvatier, T. V. Wiecki, and C. Fonnesbeck. Probabilistic programming in Python using PyMC3. *PeerJ Computer Science*, 2:e55, 2016. 1