

# EST-24107: Simulación

**Profesor:** Alfredo Garbuno Iñigo — Otoño, 2022 — *Software* de muestreo (intro).

**Objetivo:** Esta 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 MODELO

Ejemplo tomado de las [viñetas de la librería](#). El problema es poder estimar los parámetros de un modelo Normal condicional en que sólo observamos los estadísticos de orden de una muestra de tamaño  $N = 10$ .

La función de verosimilitud conjunta para el mínimo y el máximo puede probarse que se escribe como

$$\pi(x_{(1)}, x_{(N)} | \mu, \sigma) \propto \phi(x_{(1)} | \theta) \phi(x_{(N)} | \theta) [\Phi(x_{(1)} | \theta) - \Phi(x_{(N)} | \theta)]^{N-2}, \quad (1)$$

donde  $\phi(\cdot | \theta)$  denota la función de densidad de una  $N(\mu, \sigma)$ . Es decir,  $\theta \in \mathbb{R}^2$ .

El problema asume una función de densidad impropia para los parámetros *a priori*. Es decir,

$$\pi(\theta) \propto 1, \quad (2)$$

en cualquier punto  $\theta \in \Theta \subset \mathbb{R}^2$ .

Nota que esta elección es una mala elección desde el punto de vista bayesiano. Pero nos deja en una situación donde el máximo de la distribución posterior coincide con el MLE.

## 2. EL PAQUETE LEARNBAYES

```
1 library(LearnBayes)
2 minmaxpost <- function(theta, data){
3   mu <- theta[1]
4   sigma <- exp(theta[2])
5   dnorm(data$min, mu, sigma, log = TRUE) +
6     dnorm(data$max, mu, sigma, log = TRUE) +
7     ((data$n - 2) * log(pnorm(data$max, mu, sigma) -
8                           pnorm(data$min, mu, sigma)))
9 }
```

**2.1. Aproximación Normal (de Laplace)**

```

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

```

```

1 $mode
2 [1] 68.000  2.298
3
4 $var
5           [,1]      [,2]
6 [1,]  1.921e+01 -1.901e-06
7 [2,] -1.901e-06  6.032e-02
8
9 $int
10 [1] -8.02
11
12 $converge
13 [1] TRUE

```

**2.2. Muestreo por cadenas de Markov**

```

1 mcmc.fit <- rwmetrop(minmaxpost,
2                     list(var = fit$v, scale = 3),
3                     c(70, 2),
4                     10000,
5                     data)

```

```

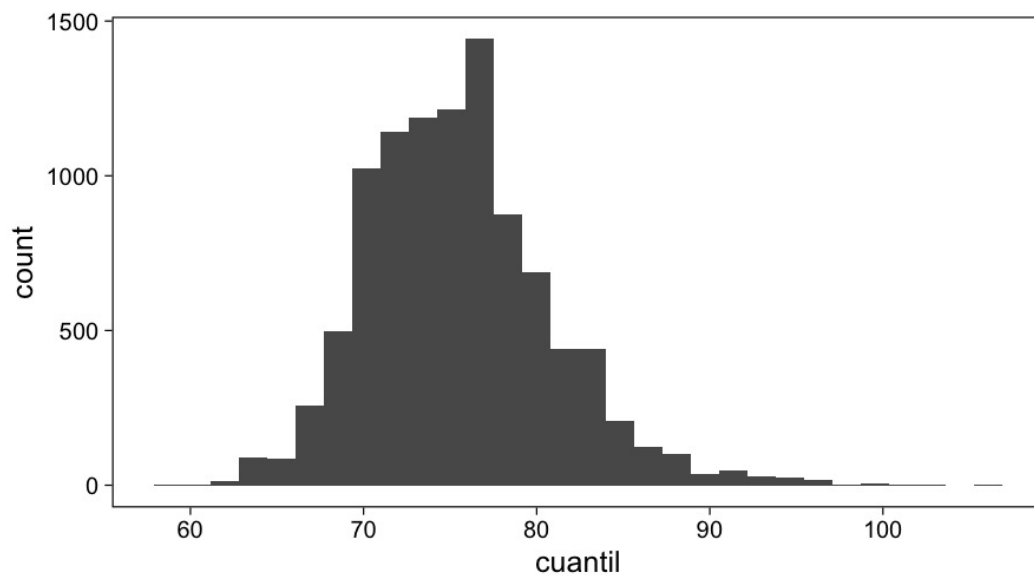
1 mcmc.fit$accept

```

```

1 [1] 0.1735

```

*2.2.1. Estimación Monte Carlo*

## 3. USANDO STAN DESDE R

```

1 data {
2   real xmin;
3   real xmax;
4   int N;
5 }
6 parameters {
7   real mu;
8   real log_sigma;
9 }
10 transformed parameters {
11   real sigma = exp(log_sigma);
12 }
13 model {
14   target += normal_lpdf(xmin | mu, sigma);
15   target += normal_lpdf(xmax | mu, sigma);
16   target += (N-2) * log(normal_cdf(xmax | mu, sigma) - normal_cdf(xmin | mu,
17   sigma));
17 }

```

```

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

```

```
1 muestras$summary()
```

```

1 # A tibble: 4 × 10
2   variable    mean median    sd    mad     q5    q95  rhat ess_bulk ess_tail
3   <chr>      <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>    <dbl>    <dbl>
4 1 lp__      -11.0  -10.7  1.07  0.776 -13.2  -9.99  1.00    2348.    2474.
5 2 mu        68.1   68.1  4.72  4.51  60.3  75.6   1.00    3749.    3021.
6 3 log_sigma  2.38   2.37  0.272 0.266  1.96  2.85   1.00    3795.    2446.
7 4 sigma     11.2   10.7  3.28  2.79  7.09 17.3   1.00    3795.    2446.

```

```

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

```

```

1 modelo$optimize(data = list(N = 10, xmin = 52, xmax = 84),
2   refresh = 0)$mle()

```

```

1 Finished in 0.1 seconds.
2      mu log_sigma      sigma
3  68.000    2.298    9.958

```

```

1 modelo$variational(data = list(N = 10, xmin = 52, xmax = 84),
2   refresh = 0, seed = 108727)

```

```

1 Finished in 0.1 seconds.
2   variable    mean median    sd    mad     q5    q95
3   lp__        -29.11 -28.86  2.99  2.63 -34.18 -24.90
4   lp_approx__ -0.99  -0.67  0.96  0.68  -2.92  -0.06
5   mu           4.32   4.80 19.60 20.40 -27.98  35.38
6   log_sigma    4.28   4.27  0.24  0.24   3.88   4.70
7   sigma        74.59  71.76 18.47 16.96  48.43 109.69

```

#### 4. USANDO PYMC

```

1 import aesara.tensor as at
2 import arviz as az
3 import matplotlib.pyplot as plt
4 import numpy as np
5 import pymc as pm
6 import scipy.stats as stats
7
8 RANDOM_SEED = 108727
9 rng = np.random.default_rng(RANDOM_SEED)

```

```

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

```

```

1 with pm.Model() as model:
2     mu=pm.Normal("mu",0,100);
3     sigma=pm.HalfNormal("sigma",100);
4     base=pm.Normal("observations",mu,sigma)
5     like=pm.Potential("likelihood",minmaxpost(base))
6
7     idata=pm.sample(1500,progressbar=False)

```

```

1 Auto-assigning NUTS sampler...
2 INFO:pymc:Auto-assigning NUTS sampler...
3 Initializing NUTS using jitter+adapt_diag...
4 INFO:pymc:Initializing NUTS using jitter+adapt_diag...
5 Multiprocess sampling (4 chains in 4 jobs)
6 INFO:pymc:Multiprocess sampling (4 chains in 4 jobs)
7 NUTS: [mu, sigma, observations]
8 INFO:pymc:NUTS: [mu, sigma, observations]
9 Sampling 4 chains for 1_000 tune and 1_500 draw iterations (4_000 + 6_000
   draws total) took 15 seconds.
10 INFO:pymc:Sampling 4 chains for 1_000 tune and 1_500 draw iterations (4_000 +
   6_000 draws total) took 15 seconds.

```

```

1 az.summary(idata)

```

		mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk
		ess_tail	r_hat					
mu	3730.0	67.846	4.815	58.660	76.786	0.081	0.057	3586.0
observations	2858.0	67.979	13.518	42.029	93.418	0.236	0.167	3458.0
sigma	3466.0	11.953	3.503	6.257	18.342	0.065	0.046	2984.0

## 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