EST-24107: Simulación

Profesor: Alfredo Garbuno Iñigo — Otoño, 2023 — *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 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 estadisticos 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

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

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

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

$$\pi(\theta) \propto 1\,, \tag{2}$$

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`

```
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)))
}
```

2.1. 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
```

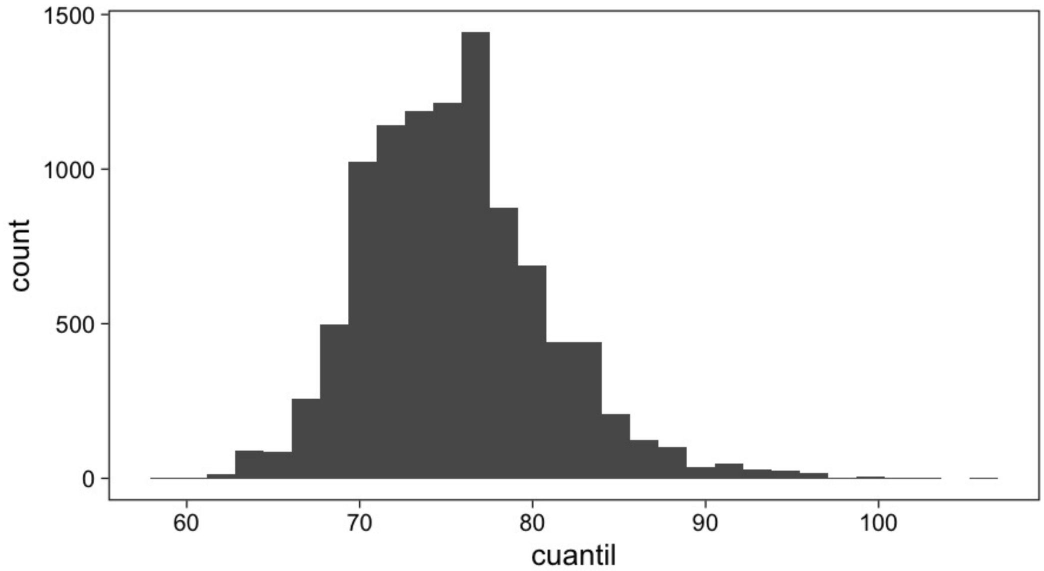
2.2. Muestreo por cadenas de Markov

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

```
mcmc.fit$accept
```

```
[1] 0.1735
```

2.2.1. Estimación Monte Carlo



3. 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$summary()
```

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

```
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())
```

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

```
Finished in  0.1 seconds.
       mu log_sigma     sigma
   68.000     2.298     9.958
```

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

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

4. 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)
```

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

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

```
          mean     sd hdi_3% hdi_97% mcse_mean mcse_sd ess_bulk ess_tail r_hat
mu      67.804  4.753 58.896  76.598     0.078   0.055   3717.0   3940.0    1.
obser   67.712 13.394 41.090  92.514     0.237   0.170   3269.0   3371.0    1.
sigma   12.021  3.701  6.551  19.093     0.071   0.051   2978.0   3534.0    1.
```

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. 1

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