EST-24107: Simulación

Profesor: Alfredo Garbuno Iñigo — Otoño, 2023 — Diagnósticos MCMC.
Objetivo. Estudiaremos diagnósticos típicos de métodos de simulación Markoviana con el objetivo de poder identificar problemas y posibles soluciones para simulaciones deficientes.
Lectura recomendada: Para una revisión de los diagnósticos de MCMC busca la sección 11.4 de [7]. El Capítulo 3 de [3] tiene una revisión muy bonita de series de tiempo que es útil para algunos términos.

1. Introducción

El avance en poder computacional ha permitido la proliferación de métodos Bayesianos. El poder generar cadenas de Markov es múltiples procesadores nos ayuda a relajar los requisitos y ayuda a explotar los recursos computacionales disponibles.

Cuando generamos una muestra de la distribución objetivo usando MCMC, sin importar el método (Metrópolis, Gibbs, HMC), buscamos que:

- Los valores simulados `no estén influenciados` por el valor inicial (arbitrario) y deben explorar todo el rango de la distribución objetivo.
- Debemos tener `suficientes simulaciones` de tal manera que las estimaciones sean precisas y estables.
- Queremos tener `métodos y resúmenes informativos` que nos ayuden diagnosticar correctamente el desempeño de nuestas simulaciones.

En la `práctica` intentamos cumplir lo más posible estos objetivos. Debemos de tener un criterio para considerar cadenas de `longitud finita` y `evaluar la calidad` de las simulaciones.

Primero estudiaremos `diagnósticos generales` para métodos que utilicen MCMC y después mencionaremos particularidades del método de simulación HMC.

2. Diagnósticos generales

Una forma que tenemos de evaluar la (o identificar la falta de) convergencia es considerar distintas secuencias independientes.

2.1. Monitoreo de convergencia

`Burn-in e iteraciones iniciales`. En primer lugar, en muchas ocasiones las condiciones iniciales de las cadenas las escogemos de tal forma que que son *atípicos* en relación a la distribución objetivo.

Estrategias de selección de puntos iniciales pueden ser valores aleatorios de la previa o perturbaciones aleatorias a estimadores MLE.

Correr varias cadenas en puntos dispersos tienen la ventaja de explorar desde distintas regiones de la distribución objetivo. Eventualmente, esperamos que todas las *cadenas* `mezclen`

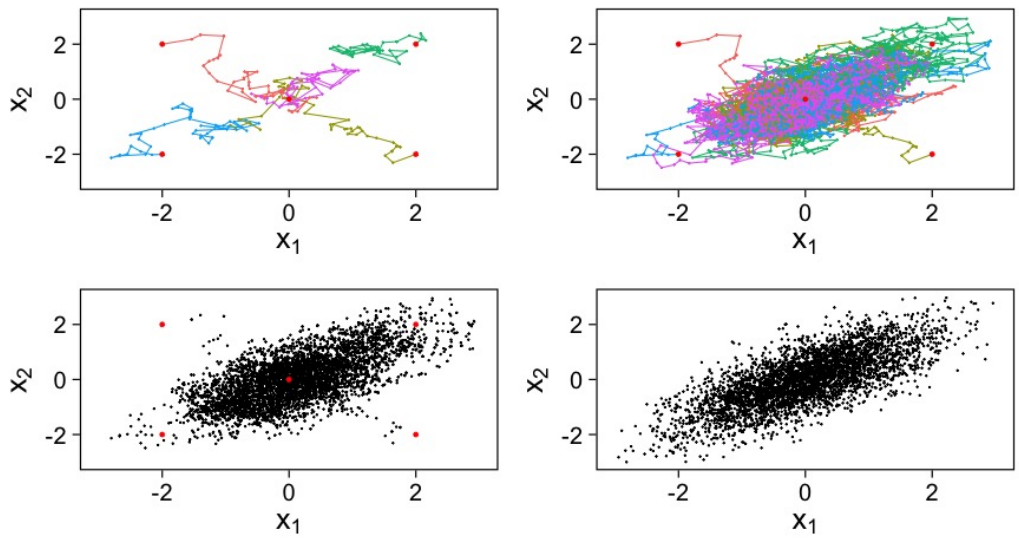


FIGURA 1. *Distintas cadenas de Markov.*

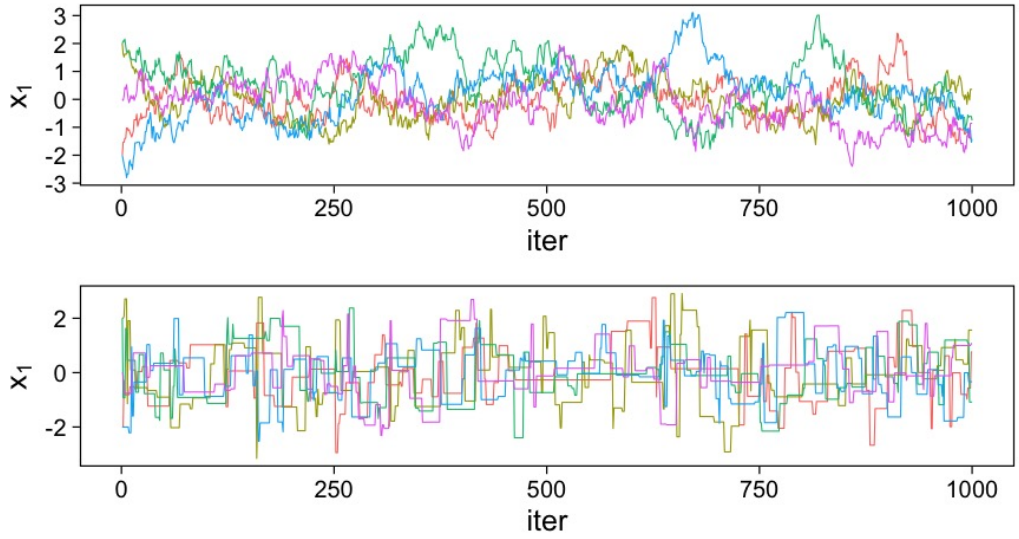


FIGURA 2. *Trayectorias de simulación para* X_1.

`bien` y `representen realizaciones independientes` del mismo proceso estócastico (Markoviano).

Para contrarrestar la dependencia en los distintos puntos iniciales se descarta parte de la cadena en un periodo inicial (periodo de calentamiento).

2.1.1. Datos: Cantantes de ópera

```
## Datos: cantantes de opera ------------------------
set.seed(3413)
cantantes ← lattice::singer ▷
  mutate(estatura_cm = round(2.54 * height)) ▷
  filter(str_detect(voice.part, "Tenor")) ▷
  select(voice.part, estatura_cm) ▷
  sample_n(20)
```

```
# A tibble: 20 × 2
  voice.part estatura_cm
  <fct>            <dbl>
1 Tenor 1            178
2 Tenor 2            173
3 Tenor 1            165
#  17 more rows
#  Use ‘print(n = ...)‘ to see more rows
```

Denotamos por x_i la estatura de tenores (cantantas de ópera). Asumimos un modelo Normal con parámetros poblacionales no observados: μ y σ. El modelo previo lo asumimos como

$$\mu|\sigma \sim \text{Normal}\left(\mu_0, \frac{\sigma}{n_0}\right), \tag{1}$$

$$\sigma^{-1} \sim \text{Gamma}(a_0, b_0)\,. \tag{2}$$

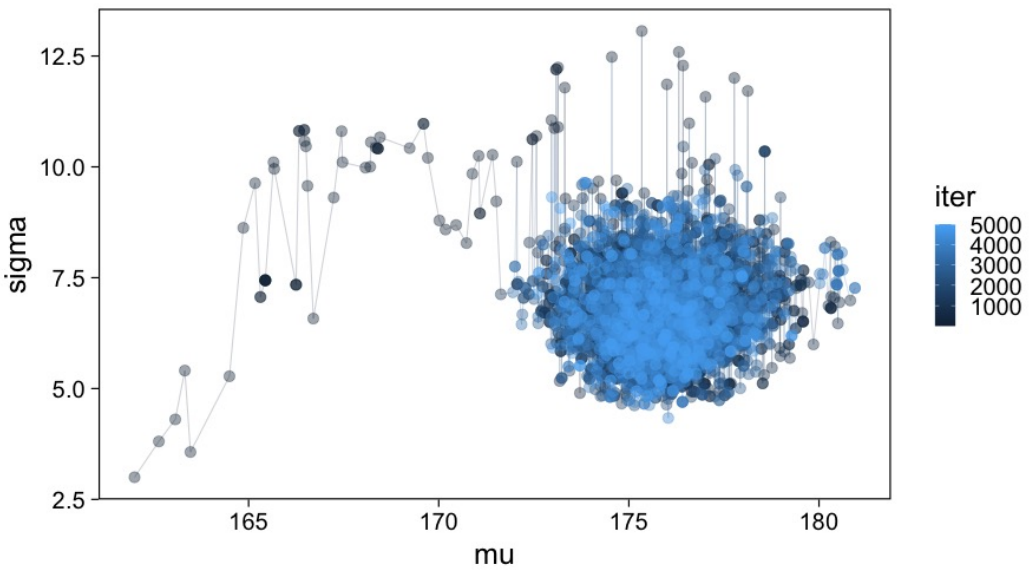


Figura 3. *Cadena de Markov simulando de la posterior como distribución objetivo.*

En esta simulación es evidente[†] que necesitamos descartar una parte inicial de la simulación.

Gelman et al. [7] recomiendan descartar la mitad de las iteraciones de cada una de las cadenas que se simularon. Para problemas en dimensiones altas, incluso se podría esperar descartar hasta un 80 % de simulaciones (en especial para métodos basados en Metropolis-Hastings).

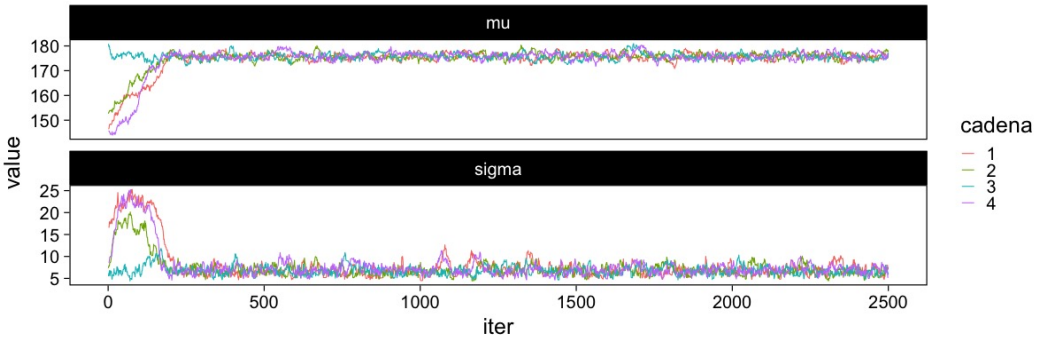


FIGURA 4. *Trayectorias con dependencias iniciales.*

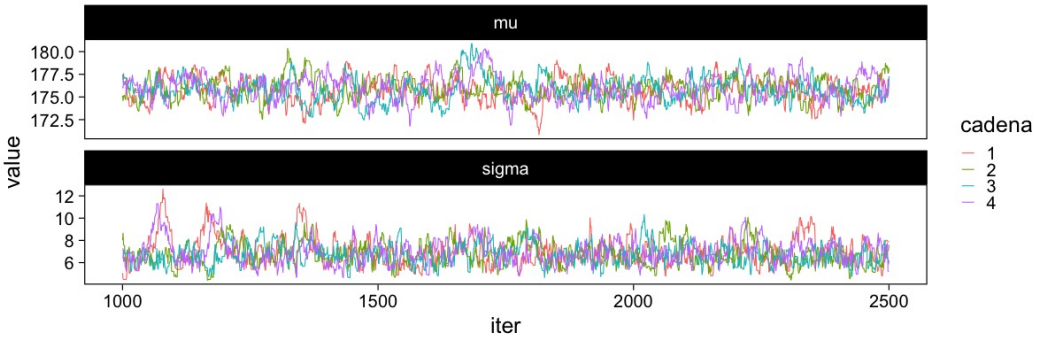


FIGURA 5. *Trayectorias estacionarias.*

2.2. Monitoreo de mezcla dentro y entre cadenas

Podemos utilizar `todas` las simulaciones como si vinieran de una sola cadena (argumentando por estacionariedad)

```
cadenas.cantantes ▷
  unnest(cadenas) ▷
  filter(iter > 100) ▷
  summarise(.estimate = mean(sigma), .variance = var(sigma),
            .error_mc = sqrt(.variance/n()))
```

```
# A tibble: 1 × 3
  .estimate .variance .error_mc
      <dbl>     <dbl>     <dbl>
1      7.11      4.14    0.0208
```

Sin embargo, al calcular la varianza como si fueran 4 cadenas independientes vemos que nuestro estimador del error Monte Carlo es mucho mas elevado de lo que esperamos ¿por qué?

```
cadenas.cantantes ▷
 unnest(cadenas) ▷
 filter(iter > 100) ▷
 group_by(cadena) ▷
 summarise(media = mean(sigma), varianza = var(sigma)) ▷
 summarise(.estimate = mean(media), .error_mc = sd(media))
```

```
# A tibble: 1 × 2
  .estimate .error_mc
      <dbl>     <dbl>
1      7.11     0.272
```

Al inspeccionar cada cadena tenemos los siguientes resúmenes

```
cadenas.cantantes ▷
 unnest(cadenas) ▷
 filter(iter > 100) ▷
 group_by(cadena) ▷
 summarise(media = mean(sigma), varianza = var(sigma))
```

```
# A tibble: 4 × 3
  cadena media varianza
  <fct>  <dbl>    <dbl>
1 1       7.34     7.74
2 2       6.93     2.35
3 3       6.82     1.13
4 4       7.35     5.12
```

Podemos partir cada cadena a la mitad y calcular nuestra estimación del error Monte Carlo. Ahora tenemos 8 cadenas que *esperamos* sean `estacionarias` (*idénticamente distribuidas*).

```
# A tibble: 1 × 2
  .estimate .error_mc
      <dbl>     <dbl>
1      7.11     0.418
```

Nota cómo está sucediendo algo contraintuitivo. Tenemos mas observaciones (pasamos de 1 cadena a 8) y el error Monte Carlo no decrece. Lo cual indica que nuestras cadenas realmente no han terminado de converger y tienen comportamiento distinto aunque en promedio parecen estar cercanas.

Gelman y diversos de sus coatures han desarollado un diagnóstico numérico para evaluar implementaciones de MCMC al considerar múltiples cadenas. Aunque éste estadístico se ha ido refinando con los años, su desarrollo muestra un entendimiento gradual de éstos métodos en la práctica. La medida $\hat{R}$ se conoce como el `factor de reducción potencial de escala`.

El estadístico $\hat{R}$ pretende ser una estimación de la posible `reducción en la longitud` de un intervalo de confianza si las simulaciones continuaran infinitamente.

La $\hat{R}$ estudia de manera simultánea `la mezcla` de todas las cadenas (cada cadena, y fracciones de ella, deberían de haber transitado el soporte de la distribución objetivo) y `estacionalidad` (de haberse logrado cada mitad de una cadena deberían de poseer las mismas estadísticas).

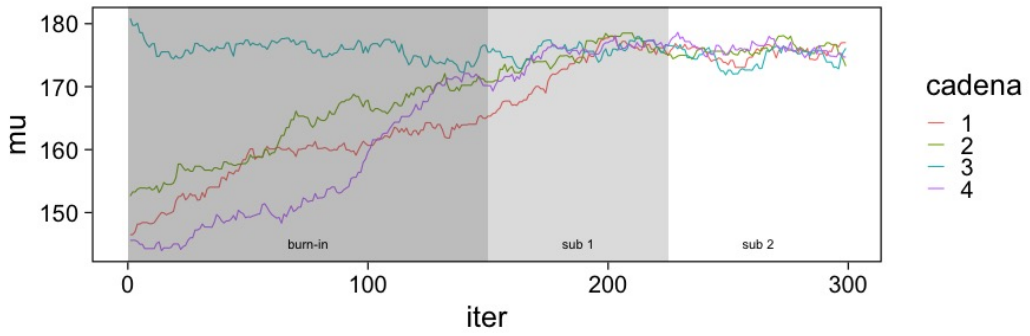


FIGURA 6. *Separación de simulaciones para cálculo de* $\hat{R}$.

La estrategia es descartar la `primera mitad` de cada cadena. El resto lo volvemos a dividir en dos y utilizamos cada fracción como si fuera una cadena independiente[†].

Denotemos por m el número de cadenas simuladas y por n el número de simulaciones dentro de cada cadena. Cada una de las `cantidades escalares de interés` las denotamos por ϕ. Éstas pueden ser los parámetros originales θ o alguna otra cantidad derivada $\phi = f(\theta)$.

Ahora denotemos por ϕ_{ij} las simulaciones que tenemos disponibles con $i = 1, \ldots, n$, y $j = 1, \ldots, m$. Calculamos B y W, la variabilidad `entre` (*between*) y `dentro` (*within*) cadenas, respectivamente, por medio de

$$W = \frac{1}{m}\sum_{j=1}^{m} s_j^2, \quad \text{con} \quad s_j^2 = \frac{1}{n-1}\sum_{i=1}^{n}(\phi_{ij} - \bar{\phi}_{\cdot j})^2, \quad \text{donde} \quad \bar{\phi}_{\cdot j} = \frac{1}{n}\sum_{i=1}^{n}\phi_{ij}, \tag{3}$$

$$B = \frac{n}{m-1}\sum_{j=1}^{m}(\bar{\phi}_{\cdot j} - \bar{\phi}_{\cdot\cdot})^2, \quad \text{donde} \quad \bar{\phi}_{\cdot\cdot} = \frac{1}{m}\sum_{j=1}^{m}\bar{\phi}_{\cdot j}. \tag{4}$$

> La varianza entre cadenas, B, se multiplica por n dado que ésta se calcula por medio de promedios y sin este factor de corrección no reflejaría la variabilidad de las cantidades de interés ϕ.

La varianza de ϕ se puede estimar por medio de

$$\hat{\mathbb{V}}(\phi)^+ = \frac{n-1}{n}W + \frac{1}{n}B\,. \tag{5}$$

Este estimador `sobre-estima` la varianza pues los puntos iniciales pueden estar sobre-dispersos, mientras que es un `estimador insesgado` una vez que se haya alcanzado el estado estacionario (realizaciones de la distribución objetivo)

Por otro lado, la varianza estimada por W será un sub-estimador pues podría ser el caso de que cada cadena no ha tenido la oportunidad de recorrer todo el soporte de la distribución. En el límite $n \to \infty$, el valor esperado de W aproxima $\mathbb{V}(\phi)$.

Se utiliza como diagnostico el factor por el cual la escala de la distribución actual de ϕ se puede reducir si se continua con el procedimiento en el límite $n \to \infty$. Esto es,

$$\hat{R} = \sqrt{\frac{\hat{\mathbb{V}}(\phi)^+}{W}}\,,$$

por construcción converge a 1 cuando $n \to \infty$.

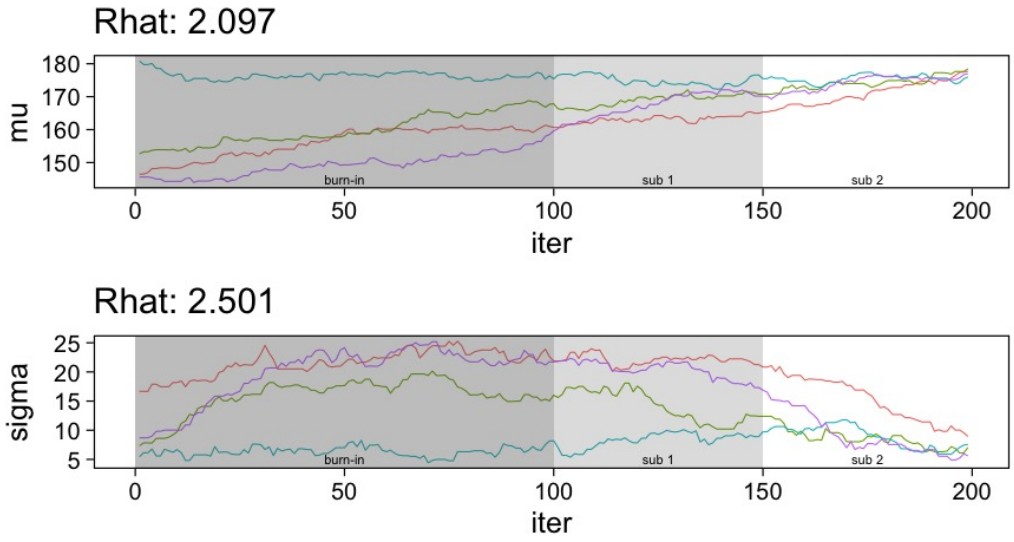


FIGURA 7. *Diágnostico de reducción de escala. Sugerencia: generar mas simulaciones.*

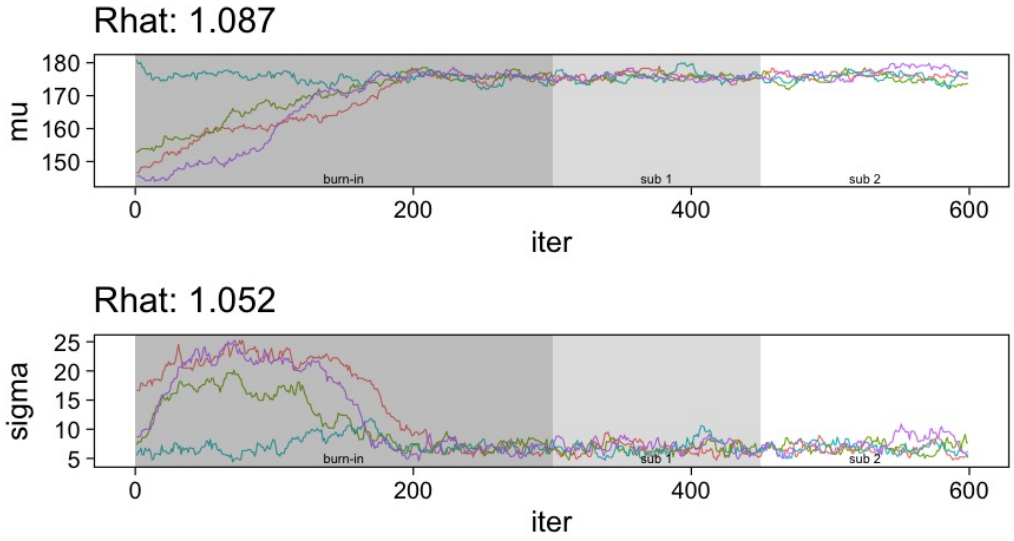


FIGURA 8. *Diágnostico de reducción de escala. Observaciones: parece estar bien.*

Problemas con $\hat{R}$. El estimador de reducción de escala funciona bien para monitorear estimadores y cantidades de interés basados en medias y varianzas, o bien, cuando la distribución es simétrica y cercana a una Gaussiana. Es decir, colas ligeras. Sin embargo, para percentiles, o distribuciones lejos del supuesto de normalidad no es un buen indicador. Es por esto que también se recomienda incorprorar transformaciones que nos permitan generar un buen estimador. Puedes leer mas de esto aqui: [**?**].

2.3. Número efectivo de simulaciones

Queremos que los recursos que hemos asignado a generar simulaciones sean representativos de la distribución objetivo. Si las n simulaciones dentro de cada cadena en verdad son realizaciones independientes entonces la estimación de B sería un estimador insesgado de $\mathbb{V}(\phi)$.

En esta situación tendríamos $n \cdot m$ realizaciones de la distribución que queremos simular. Sin embargo, la correlación entre las muestras hacen que B sea mayor que $\mathbb{V}(\phi)$ en promedio.

Una manera para definir el tamaño efectivo de simulaciones es por medio del estudio del estimador

$$\bar{\phi}_{\cdot\cdot} \approx \mathbb{E}(\phi)\,. \tag{6}$$

Del cual podemos derivar que

$$\mathbb{V}(\bar{\phi}_{\cdot\cdot}) = \frac{\mathbb{V}(\phi)}{m \cdot n}\,.$$

El problema es que la correlación en las cadenas implica el denominador $(m \cdot n)$ realmente sea una fracción del total de muestras, digamos λ. De tal forma que el número efectivo de simulaciones es

$$\mathsf{ESS} = \lambda \cdot (m\, n)\,,$$

donde

$$\lambda = \frac{1}{\sum_{t=-\infty}^{\infty} \rho_t} = \frac{1}{1 + 2\sum_{t=1}^{\infty} \rho_t}\,.$$

El término ρ_t denota la **auto-correlación** con diferencia en t unidades de tiempo.

Definición 2.1 (Autocorrelación)**.** La autocovarianza y autocorrelación de una serie temporal **estacionaria** $\{Y_t : t = 0, \ldots\}$ están definidas (respectivamente) como

$$C_\tau = \mathbb{E}[(Y_{t+\tau} - \mu)(Y_t - \mu)], \qquad \rho_\tau = \frac{\mathbb{E}[(Y_{t+\tau} - \mu)(Y_t - \mu)]}{\sigma^2}\,. \tag{7}$$

Definición 2.2 (Estimador de autocorrelación)**.** La función de autocorrelación se estima utilizando

$$\hat{C}_\tau = \frac{1}{T - \tau}\sum_{t=1}^{T-\tau}(Y_{t+\tau} - \hat{\mu})(Y_t - \hat{\mu}), \qquad \hat{\rho}_\tau = \frac{\hat{C}_\tau}{\hat{C}_0}\,. \tag{8}$$

Definición 2.3 (Variograma)**.** El variograma de una serie temporal **estacionaria** $\{Y_t : t = 0, \ldots\}$ está definido como

$$V_\tau = \mathbb{E}[(Y_{t+\tau} - Y_t)^2]\,. \tag{9}$$

Nota que $V_\tau = C_0 - C_\tau$.

Regresando a nuestro contexto... para estimar ρ_t partimos de nuestro estimador $\hat{\mathbb{V}}(\phi)^+$; y utilizamos el **variograma** V_t para `cada retraso` t

$$V_t = \frac{1}{m(n-t)} \sum_{j=1}^{m} \sum_{i=t+1}^{n} (\phi_{i,j} - \phi_{i-t,j})^2 \,.$$

Utilizando la igualdad $\mathbb{E}(\phi_i - \phi_{i-t})^2 = 2(1-\rho_t)\mathbb{V}(\phi)$, podemos estimar

$$\hat{\rho}_t = 1 - \frac{V_t}{2\,\hat{\mathbb{V}}(\phi)^+} \,.$$

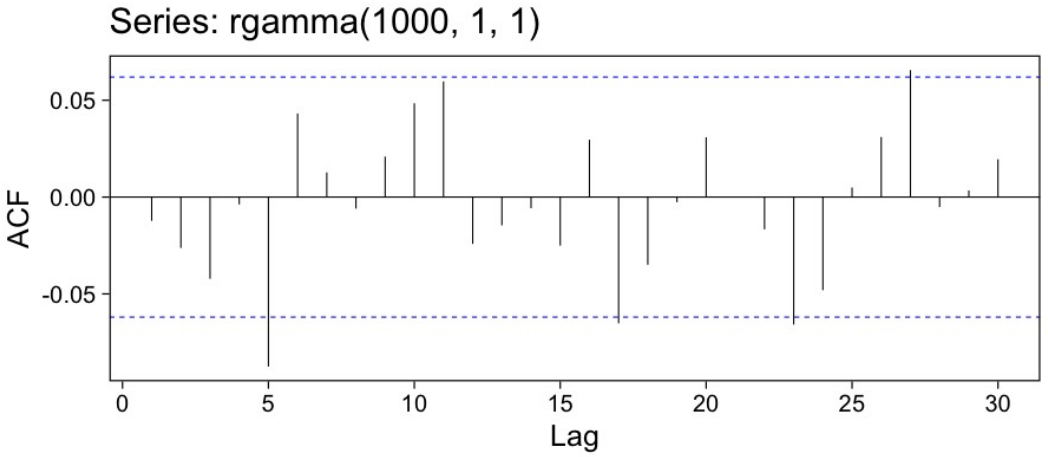
La mayor dificultad que presenta el estimador es considerar **todos** los retrasos posibles. Eventualmente agotaremos la longitud de las cadenas para ello. Por otro lado, para t eventualmente grande nuestros estimadores del variograma V_t serán muy ruidosos (¿por qué?). En la práctica truncamos la serie de acuerdo a las observaciones [8]. La serie tiene la propiedad de que para cada par $\rho_{2t} + \rho_{2t+1} > 0$. Por lo tanto, la serie se trunca cuando observamos $\hat{\rho}_{2t} + \hat{\rho}_{2t+1} < 0$ para dos retrasos sucesivos.

Si denotamos por T el **tiempo de paro**, el estimador para el número efectivo de simulaciones es

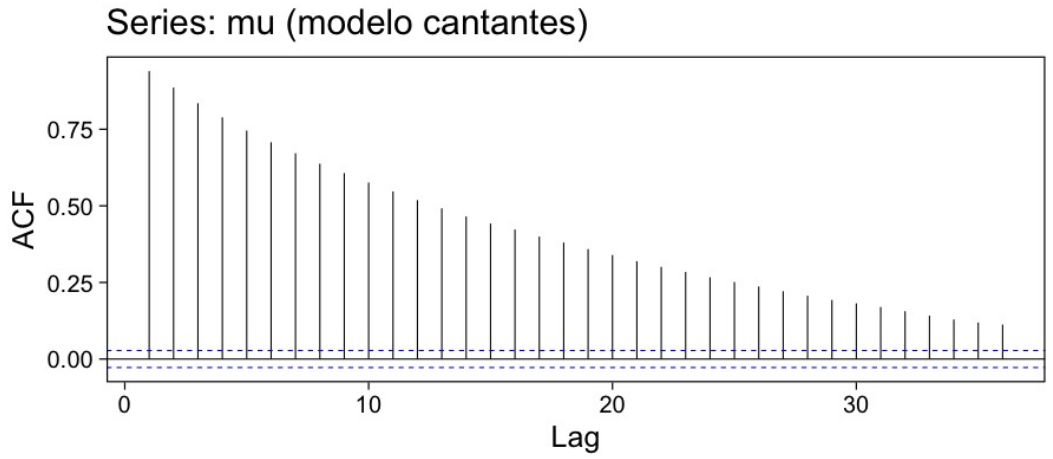
$$\widehat{\mathsf{ESS}} = \frac{m\,n}{1 + 2\sum_{t=1}^{T} \hat{\rho}_t} \,.$$

El `tamaño efectivo de simulaciones` nos ayuda a monitorear lo siguiente. Si las simulaciones fueran independientes ESS sería el número total de simulaciones; sin embargo, las simulaciones de MCMC suelen estar correlacionadas, de modo que cada iteración de MCMC es menos informativa que si fueran independientes.

Por ejemplo si graficaramos simulaciones independientes, esperaríamos valores de autocorrelación chicos:



Sin embargo, los valores que simulamos tienen el siguiente perfil de autocorrelación:



Podemos utilizar la función `posterior::ess_basic`.

```
# A tibble: 1 × 5
    ESS   N.M  ratio  sigma accept
  <dbl> <int>  <dbl>  <dbl>  <dbl>
1   146  5000 0.0292 0.0482   0.69
```

Listing 1. *Fracción* ESS/*nm para la simulación de la posterior los cantantes de ópera.*

```
# A tibble: 1 × 5
    ESS   N.M  ratio sigma accept
  <dbl> <int>  <dbl> <dbl>  <dbl>
1   441  5000 0.0882 0.146  0.380
```

Listing 2. *Fracción* ESS/*nm para la simulación (calibrada) de la posterior los cantantes de ópera.*

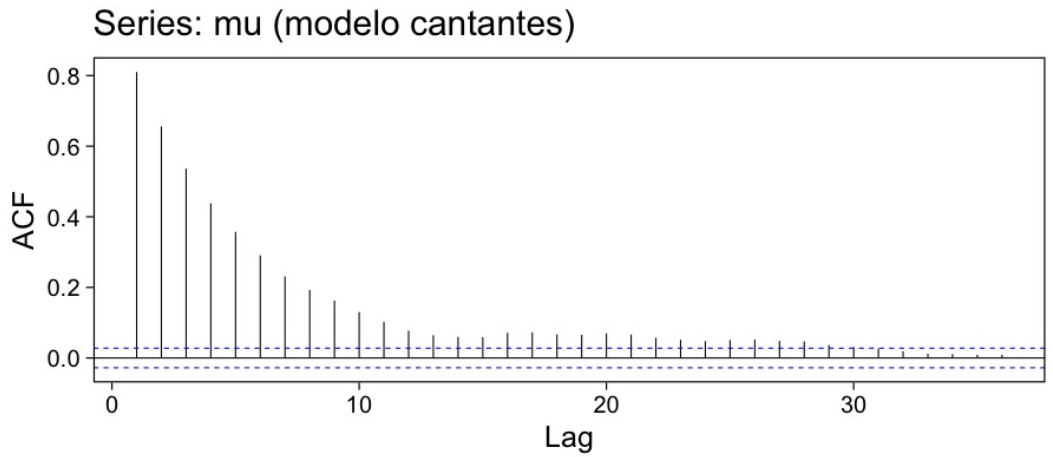


Figura 9. *Perfil de correlación para la simulación calibrada.*

2.4. Relación con error Monte Carlo

Conocer el número efectivo de simulaciones nos permite calcular el error estándar de una aproximación Monte Carlo por medio de expresiones como

$$\mathsf{SE}\left(\hat{\pi}_{\mathsf{ESS}}^{\mathsf{MCMC}}(f)\right) \approx \left(\frac{\hat{\mathbb{V}}_{\pi}(f)}{\widehat{\mathsf{ESS}}}\right)^{\frac{1}{2}}. \tag{10}$$

Geyer [8] menciona que realizar estimaciones puntuales sin medidas de incertidumbre es el *deber* ser de un analista/estadístico. Esto es, independientemente de si la técnica es Bayesiana o frecuentista. El artículo de Flegal et al. [4] también discute la importancia de reportar errores estándar.

3. El estado del arte

El método de Metropolis-Hastings es muy flexible y existe una colección numerable de versiones que pueden ser empleadas en contextos muy particulares. Una buena referencia que incluye métodos de simulación por medio de cadenas de Markov se encuentra en [11], donde incluso se pueden encontrar generalizaciones con `transiciones Markovianas asimétricas` y extensiones a `problemas de dimensión variable`. El libro [1] presenta el estado del arte al 2010.

El cómputo Bayesiano se popularizó con el muestreador de Gibbs. En particular, el avance en teoría de grafos para representar una distribución conjunta como un Grafo Acíclico Dirigido (DAG) que se implementó en software como `BUGS` o WinBUGS. Pueden consultar el libro de Kruschke [10] para su explicación.

La desventaja del muestreador de Gibbs es que tiende a ser muy lento en problemas de tamaño grande. Ha habido estrategias que aceleran la simulación aunque al `costo de utilizar aproximaciones`. Estas estrategias han sido materializadas en lenguajes de programación mas generales como Infer.NET.

JAGS (Just Another Gibbs Sampler), es una generalización donde se implementan métodos MCMC para generar simulaciones de distribuciones posteriores. Los paquetes `rjags` y `R2jags` permiten ajustar modelos en JAGS desde `R` [9]. Es muy fácil utilizar estos programas pues uno simplemente debe especificar las distribuciones iniciales, la verosimilitud y los datos observados. Igual el libro de Kruschke [10].

Al depender de gradientes para construir propuestas para las cadenas de Markov ha sido natural el desarrollo de herramientas de muestreo basadas en diferenciadores automáticos. Por ejemplo, Pyro utiliza PyTorch. Tenemos también Tensorflow Probability que utiliza `Tensorflow`. Pymc (antes Pymc3) utiliza Theano (ahora llamado Aesara). NumPyro utiliza `numpy` y JAX como *backend*.

Pymc es un muestreador `híbrido` que permite utilizar Metropolis-Hastings, Gibbs y HMC para la simulación de la posterior [12]. También es mucho más flexible y brinda muestreadores más modernos basados en particulas e información de primer orden (gradientes).

Además, hay herramientas que utilizan las librerías de muestreo para análisis específicos. Por ejemplo, tenemos `cmdstanarm` ajusta **modelos de regresión** utilizando `Stan` como *backend*. La herramienta de Facebook, `Prophet`, utiliza `Stan` (ver aqui) como *backend* y se especializa en **series de tiempo**. fenics-pymc3 se especializa en soluciones de **ecuaciones diferenciales** escritas en `FEniCS`. También tenemos beat para **análisis probabilístico de**

terremotos y exoplanet para series de tiempo en **astronomía**. Por supuesto, no podía faltar una integración `scikit` que se llama Pymc-Learn.

Existen otras alternativas para construir cadenas de Markov. Por ejemplo, hay algoritmos que buscan evolucionar una colección de muestras de θ como un enjambre que se comunican entre si para generar una caminata aleatoria en el espacio del soporte de la distribución. Ejemplos de éstos son el `t-walk` [2] o un ensamble de cadenas linealmente relacionadas como en la herramienta de `emcee` [5],

Finalmente, hay muchos mas mecanismos que tienen como objetivo aproximar la distribución posterior. En problemas donde la verosimilitud es `computacionalmente costosa` existen alternativas para crear aproximaciones. El artículo [6] provee de una alternativa utilizando una combinación de técnicas bien establecidas (difusiones Langevin, ensamble de partículas interactivas y filtros de Kalman).

Referencias

[1] S. Brooks, A. Gelman, G. Jones, and X.-L. Meng. *Handbook of Markov Chain Monte Carlo*. CRC press, 2011. 11

[2] J. A. Christen and C. Fox. A general purpose sampling algorithm for continuous distributions (the t-walk). *Bayesian Analysis*, 5(2):263–281, jun 2010. 12

[3] N. Cressie and C. K. Wikle. *Statistics for Spatio-Temporal Data*. John Wiley & Sons, 2015. 1

[4] J. M. Flegal, M. Haran, and G. L. Jones. Markov Chain Monte Carlo: Can We Trust the Third Significant Figure? *Statistical Science*, 23(2):250–260, may 2008. ISSN 0883-4237, 2168-8745. . 11

[5] D. Foreman-Mackey, D. W. Hogg, D. Lang, and J. Goodman. Emcee: The MCMC Hammer. *Publications of the Astronomical Society of the Pacific*, 125(925):306–312, mar 2013. 12

[6] A. Garbuno-Inigo, F. Hoffmann, W. Li, and A. M. Stuart. Interacting Langevin Diffusions: Gradient Structure And Ensemble Kalman Sampler. *arXiv:1903.08866 [math]*, oct 2019. 12

[7] 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. 1, 4

[8] C. J. Geyer. Introduction to Markov Chain Monte Carlo. *Handbook of Markov Chain Monte Carlo*, (1990):3–48, 2002. 9, 11

[9] K. Hornik, F. Leisch, and A. Zeileis. JAGS: A program for analysis of Bayesian graphical models using Gibbs sampling. In *Proceedings of DSC*, volume 2, pages 1–1, 2003. 11

[10] J. Kruschke. *Doing Bayesian Data Analysis: A Tutorial with R, JAGS, and Stan*. Academic Press, 2014. 11

[11] J. S. Liu. *Monte Carlo Strategies in Scientific Computing*. Springer Series in Statistics. Springer New York, New York, NY, 2004. 11

[12] J. Salvatier, T. V. Wiecki, and C. Fonnesbeck. Probabilistic programming in Python using PyMC3. *PeerJ Computer Science*, 2:e55, 2016. 11


```
R version 4.3.1 (2023-06-16)
Platform: x86_64-apple-darwin20 (64-bit)
Running under: macOS Sonoma 14.1

Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/4.3-x86_64/Resources/lib/
    libRblas.0.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/4.3-x86_64/Resources/lib/
    libRlapack.dylib;  LAPACK version 3.11.0

locale:
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

time zone: America/Mexico_City
tzcode source: internal

attached base packages:
[1] stats     graphics  grDevices datasets  utils     methods   base
```

```
other attached packages:
 [1] posterior_1.4.1 forecast_8.21   R6_2.5.1        mvtnorm_1.2-2
 [5] scales_1.2.1    patchwork_1.1.2 lubridate_1.9.2 forcats_1.0.0
 [9] stringr_1.5.0   dplyr_1.1.2     purrr_1.0.1     readr_2.1.4
[13] tidyr_1.3.0     tibble_3.2.1    ggplot2_3.4.2   tidyverse_2.0.0

loaded via a namespace (and not attached):
 [1] tensorA_0.36.2       utf8_1.2.3           generics_0.1.3
 [4] renv_1.0.0           stringi_1.7.12       lattice_0.21-8
 [7] hms_1.1.3            magrittr_2.0.3       grid_4.3.1
[10] timechange_0.2.0     backports_1.4.1      nnet_7.3-19
[13] fansi_1.0.4          abind_1.4-5          cli_3.6.1
[16] rlang_1.1.1          crayon_1.5.2         munsell_0.5.0
[19] withr_2.5.0          tools_4.3.1          parallel_4.3.1
[22] tzdb_0.4.0           checkmate_2.2.0      colorspace_2.1-0
[25] curl_5.0.1           vctrs_0.6.3          matrixStats_1.0.0
[28] zoo_1.8-12           lifecycle_1.0.3      tseries_0.10-54
[31] urca_1.3-3           pkgconfig_2.0.3      pillar_1.9.0
[34] gtable_0.3.3         glue_1.6.2           quantmod_0.4.24
[37] Rcpp_1.0.11          lmtest_0.9-40        tidyselect_1.2.0
[40] farver_2.1.1         nlme_3.1-162         xts_0.13.1
[43] labeling_0.4.2       timeDate_4022.108    fracdiff_1.5-2
[46] compiler_4.3.1       quadprog_1.5-8       distributional_0.3.2
[49] TTR_0.24.3
```