Algoritmos para el método Markov Chain Monte Carlo (MCMC)

Teoría y aplicaciones

Joaquin Cavieres G.

1. Introducción

La aplicación de modelos Bayesianos durante los últimos años ha presentado un fuerte crecimiento debido a la generalidad que tienen estos para ser especificados y gracias a las nuevas tecnologías computacionales que han permitido incrementar el nivel de cómputo de ordenadores tradicionales. La inferencia dentro de la metodología Bayesiana en modelos estadísticos sencillos (modelos lineales, modelos para proporciones o modelos de conteo) o en otro tipo de modelos más complejos (efectos aleatorios, jerárquicos, espaciales, temporales o espacio-temporales) generalmente se hace a través del método Markov Chain Monte Carlo (MCMC).

En primera instancia, vamos a hacer una diferenciación entre estadística frecuentista y estadística Bayesiana, resumido en los siguientes puntos:

I) Estadística frecuentista (o clásica): Se basa principalmente en la función de verosimilitud. La función de verosimilitud es la función de densidad de probabilidad de los datos y dado el vector de parámetros. $\boldsymbol{\theta}$

$$\pi(y \mid \boldsymbol{\theta}) \tag{1}$$

En general se quiere estimar un vector $\hat{\boldsymbol{\theta}}$ mediante algún método de estimación (generalmente con un *Estimador Máximo Verosímil* (MLE siglas en inglés)).

II) Estadística Bayesiana: Dado un conocimiento a priori del vector $\boldsymbol{\theta}$ podemos calcular la distribución de los parámetros dado los datos y

$$\pi(\boldsymbol{\theta} \mid y) \tag{2}$$

La diferencia fundamental es que, dentro del paradigma bayesiano, los parámetros y los datos son tratados como variables aleatorias.

1.1. Notaciones relevantes

El objetivo principal dentro de la inferencia Bayesiana es obtener la distribución posterior y evaluar la incertidumbre en las estimaciones. Para calcular la distribución posterior primero debemos contar con una expresión para la función de verosimilitud (*likelihood*) y asignar una distribución a priori para nuestros parámetros (también conocida como *Prior distribution*, *distribución previa* o *distribución a priori*). Pero, ¿de donde viene esta idea? Esta idea proviene fundamentalmente desde el teorema de Bayes

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

donde:

- $P(A)$ son las probabilidades a priori
- $P(B \mid A)$ es la probabilidad de B dado el evento A
- $P(A \mid B)$ son las probabilidades a posterior

Entonces, a partir de este téorema nosotros podemos escribir la siguiente expresión representando a nuestros datos como y el conjunto de parámetros como θ:

$$\begin{aligned} \pi(y, \theta) &= \pi(y \mid \theta)\, \pi(\theta) \\ \pi(\theta \mid y) &= \frac{\pi(y, \theta)}{\pi(y)} \\ &= \frac{\pi(y \mid \theta)\, \pi(\theta)}{\pi(y)} \end{aligned}$$

obviando el termino $\pi(y)$ (ya que no depende de θ) se tiene que:

$$\pi(\theta|y) \propto \pi(y|\theta)\, \pi(\theta)$$

$$\textit{posterior} \propto \textit{likelihood} \times \textit{prior}$$

La función de verosimilitud incorpora la probabilidad del modelo $\pi(\boldsymbol{y} \mid \theta)$ y del vector de datos $\boldsymbol{y} = (y_1, ..., y_n)$ a través de $\pi(\boldsymbol{y}, \theta) = \prod_{i=1}^{n} \pi(y_i \mid \theta)$ para cuando las observaciones son asumidas como *i.i.t.* Aquí, la prior $\pi(\theta)$ refleja el conocimiento previo sobre los parámetros del modelo.

A continuación se describen algunos tipos de distribuciones a priori que pueden ser utilizadas para describir el conocimiento previo sobre los parámetros:

a) Conjugadas: Son aquellas en que la distribución posterior pertenecen a la misma familia de distribuciones que la distribución a priori cuando es combinada con la función de verosimilitud, por ejemplo: La distribución Beta es conjugada con la distribución Bernoulli y Binomial. Las distribuciones a priori conjugadas son fáciles de manejar mediante cálculos computacionales (debido a su forma cerrada) pero sólo existen en casos puntuales (ver Berger, J., 2000)

b) Prior no informativas (o también conocidas como "vagas"): Es cuando se considera que una distribución a priori tiene un impacto mínimo sobre la distribución posterior debido a que no se tiene ningún tipo de información previa sobre los parámetros θ. Este tipo de distribuciones a priori también se les conoce como difusas o “flat” (mayor descripción puede verse en Box and Tiao, 1973)

c) Prior de Jeffreys: Este tipo de prior (Jeffreys, 1961) satisfacen la propiedad de uniformidad local: “Es una prior que no cambia mucho en la región en la que la probabilidad es significativa y no asume grandes valores fuera de ese rango”. Este tipo de distribuciones también tiene la propiedad de ser invariante a reparametrizaciones. Las prior de Jeffreys están basadas en la matriz de información de Fisher dada por:

$$I(\theta) = \mathbb{E}_{\theta}\left[\left(\frac{\partial log\ f(y \mid \theta)}{\partial \theta}\right)^2\right]$$

y bajo algunas condiciones de regularidad:

$$-\mathbb{E}_\theta\left(\frac{\partial^2 log\ f(y \mid \theta)}{\partial^2 \theta}\right)$$

definimos a la prior de Jeffreys como:

$$\pi(\theta) \propto \mid I(\theta)^{1/2} \tag{3}$$

la prior de Jeffreys es localmente uniforme.

d) Prior informativas: Son aquellas que no están determinadas totalmente por la función de verosimilitud $\pi(y \mid \theta)$ y sí tienen un impacto sobre la distribución posterior $\pi(\theta) \mid y)$. Esta asignación de prior informativa puede estar basada en datos históricos de la distribución de los parámetros, de experimentos realizados previamente o simplemente del conocimiento de expertos sobre el fenómeno de interés.

Ya habiendo hecho un breve resumen sobre la distribución a priori, daremos paso al objetivo de este documento; presentar el método MCMC quien nos permite realizar la inferencia sobre los parámetros del modelo. Gracias a este método nosotros podemos calcular cantidades de interés como la media posterior, la mediana posterior, intervalos de credibilidad o la desviación estándar de los parámetros, sólo muestreando desde la distribución posterior. Este tipo de aproximaciones de les conoce como aproximaciones de Monte Carlo.

2. Método Markov Chain Monte Carlo (MCMC)

2.1. Monte Carlo

Supongamos que tenemos $X_1, X_2...$ una suseción de variables aleatorias independientes e identicamente distribuidas (*i.i.d*) desde alguna distribución de probabilidad cualquiera. Si nosotros queremos calcular la esperanza de la variable aleatoria $g(X)$ mediante la esperanza $\mu = E\{g(X)\}$, entonces la aproximación hacia μ por el método de Monte Carlo estaría representada como:

$$\text{Media posterior} = \hat{\mu}_n \approx \frac{1}{n}\sum_{i=1}^{n} g(X_i)$$

Ya que $\hat{\mu}_n$ es la media de una muestra $g(X_1), ..., g(X_n)$ *i.i.d*, y que a su vez tiene esperanza μ, entonces por la ley de grandes números $\hat{\mu}_n$ converge casi seguramente a μ (cuando el número de simulaciones va hacia el infinito).

$$\hat{\mu}_n \xrightarrow{c.s} \mu, \quad n \to \infty$$

Si la $Var\{g(X)\}$ es finita, digamos σ^2, entonces por Téorema de Límite Central (TLC), $\hat{\mu}_n$ es asintóticamente normal con media μ y varianza σ^2/n,

$$\sqrt{n}(\hat{\mu}_n - \mu) \longrightarrow N(0, \sigma^2)$$

El problema de aproximar mediante el método de Monte Carlo .ºrdinario.es que es muy difícil de implementar para cantidades aleatorias multivariantes (debido a la inmediatez de este método)

2.2. Markov Chain

El método de Monte Carlo por si sólo no es un buen estimador para aproximar hacia un valor de tendencia central, ya que asumiendo que una variable es aleatoria $i.i.d$, la convergencia en n número de iteraciones no es la adecuada. Por lo anterior es que el método de Markov Chain (Cadenas de Markov) es escencial para lograr la aproximación hacia la convergencia. Sin considerar el supuesto de independencia, la forma de dependencia más simple entre variables aleatorias de un proceso estocástico es el proceso de Markov. Pero, ¿que es un proceso estocástico?

Proceso estocástico: Es una sucesión de variables aleatorias $X_t, t \in T$ indexadas por un conjunto T y que toma valores en un espacio medible común S y definido bajo un σ - algebra apropiado (Insua, D *et al.*, (2012)). T puede ser el tiempo cuando es un proceso estocástico temporal pero también puede ser un proceso espacial cuando se tienen observaciones de coordenadas espaciales, por ejemplo *latitud - longitud*.

Definición formal Markov Chain: Consideremos un conjunto de $\{t_0, ..., t_n, t\}$, con $t_0 < t_1, ..., t_n < t$ y $t_i \in T$. Un proceso estocástico $\{X_t, t \in T\}$ es Markoviano si la distribución condicional de X_t en los valores de $X_{t1}, ..., X_{tn}$ depende sólo de X_{tn}, el valor mas reciente del proceso (Insua, D *et al.*, (2012)). En simples palabraas, el estado futuro es independiente del pasado dado su estado actual.

Ejemplo: Consideremos una muestra de $\boldsymbol{\theta}^{(t)}$. La siguiente muestra $\boldsymbol{\theta}^{(t+1)}$ sólo depende de $\boldsymbol{\theta}^{(t)}$ (actual) y no de las muestras pasadas. Los anterior satisface las **Propiedades Markovianas**:

$$p(\boldsymbol{\theta}^{(t+1)}|\theta^{(1)} + \theta^{(2)}, ..., \theta^{(t)}) = p(\theta^{(t+1)}|\theta^{(t)})$$

El proceso de Markov Chain debe tener las siguientes propiedades:

Estacionariedad

Un proceso estrictamente estacionario es un proceso estocástico cuya distribución de probabilidad en un instante de tiempo fijo, o una posición fija, es la misma para todos los instantes de tiempo o posiciones.

Una cadena de Markov es estacionaria si esta a su vez es un proceso estocástico estacionario. En consecuencia, parámetros tales como la media y la varianza, si existen, no varían a lo largo del tiempo o el espacio.

Reversibilidad

Una transición de distribución de probabilidad es *reversible* con respecto a una distribución inicial si, para una cadea de Markov $X_1, X_2, ...$, la distribución de pares (X_i, X_{i+1}) es intercambiable. Una cadena de Markov es reversible si, esta transción de probabilidad, es reversible con respecto a la distribución inicial. Reversibilidad implica estacionariedad pero estacionariedad no implica reversibilidad

Funcionalidad

Si $X_1, X_2, ...$ es un proceso estocástico y g es una función de valor real en el espacio de estados, entonces el proceso estocástico $g(X_1), g(X_2), ...$, teniendo un espacio de estados en $\mathcal{R}$ se dice que es funcional de $X_1, X_2, ...$ Si $X_1, X_2, ...$ es una cadena de Markov, entonces $g(X_1), g(X_2), ...$ usualmente no es una cadena de Markov.

Por otra parte, Markov Chains tiene dos propiedades importantes:

a) **Dependencia**: Produce una correlación inherente entre cada iteración y va decayendo iteraciones distantes. Por ejemplo: $\theta^{(t+1)}$ y $\theta^{(t)}$ tiene una mayor correlación que $\theta^{(t)}$ y $\theta^{(t+100)}$

b) **Distribución en equilibrio**: La cadena presenta una estabilización ante una gran cantidad de iteraciones.

Nosotros hemos nombramos algunas caracteristicas de los procesos de Markov Chain y de Monte Carlo pero sin especificar el detalle matemático o teórico de algunos de sus componentes, esto por que nosotros queremos simplicar la comprensión del método MCMC y los algoritmos mas comunes que lo componen, pero para mayor detalle téorico en cadenas de Markov se puede revisar Mikosch, T (1998) o Lindgren G, (2013) y para el método de Monte Carlo se puede revisar Metropolis, N. (1987) o Brooks, S et al., (2011).

3. Método MCMC

La idea principal del método MCMC es simular una cadena de Markov $\theta_1, \theta_2, ...$ cuya distribución estacionaria sea $\pi(\theta|\boldsymbol{y})$. Dentro del contexto de la estadística Bayesiana nosotros queremos construir cadenas que presenten un equilibrio (estacionariedad) a la distribución posterior de las cantidades que estamos interesados en conocer.

- Cada valor simulado, θ_t, depende de únicamente del paso anterior, θ_{t-1}.

Pero también es muy recomendable iterar una gran cantidad de veces para poder garantizar la convergencia aproximada hacia una distribucion posterior mediante algoritmos iterativos que permiten generalizar esta hipótesis de convergencia. Existen una gran cantidad de algoritmos utilizados en el método MCMC pero aquí estudiaremos tres: Metropolis - Hasting, Gibbs Sampling y el Hamiltoniano Monte Carlo.

3.1. Algoritmo de Metropolis - Hasting (M-H)

Supongamos que queremos muestrear desde una distribución posterior $\pi(\theta|\boldsymbol{y})$, pero:

- La $\pi(\theta|\boldsymbol{y})$ no parece tener una distribución probabilística conocida
- $\pi(\theta|\boldsymbol{y})$ tiene más de dos parámetros (lo que puede ser complejo tratarla en forma analítica)
- Algunas (o todas) las condicionales no parecen tener una expresión conocida.

..... Entonces podemos utilizar el algoritmo de M-H.

El algoritmo M-H simula cadenas de Markov desde una distribución estacionaria sobre un conjunto de parámetros $\pi(\boldsymbol{\theta}|\boldsymbol{y})$ y comprende de los siguientes pasos:

Algoritmo 1 Metropolis - Hasting

1: Elegir un valor para $\boldsymbol{\theta^{(0)}}$
2: En cada iteración t, extraemos un candidato $\boldsymbol{\theta^*}$ desde una distribución instrumental $J_t(\boldsymbol{\theta^*} \mid \boldsymbol{\theta^{(t-1)}})$
3: Cálcular la tasa de aceptación (probabilidad):

$$r = \frac{\pi(\boldsymbol{\theta^*} \mid \boldsymbol{y})/J_t(\boldsymbol{\theta^*} \mid \boldsymbol{\theta^{(t-1)}})}{\pi(\boldsymbol{\theta^{(t-1)}} \mid \boldsymbol{y})/J_t(\boldsymbol{\theta^{(t-1)}} \mid \boldsymbol{\theta^*})}$$

4: Aceptar $\boldsymbol{\theta^*}$ como $\boldsymbol{\theta^{(t)}}$ con probabilidad mín$(r, 1)$. Si $\boldsymbol{\theta^{(t)}}$ no se acepta, entonces $\boldsymbol{\theta^{(t)}} = \boldsymbol{\theta^{(t-1)}}$
5: Repetir los pasos 2-4 M veces hasta obtener M muestras desde $\pi(\boldsymbol{\theta} \mid \boldsymbol{y})$.
6: Finalizar

Paso 1: Elegir un valor inicial para $\boldsymbol{\theta}^{(0)}$

I) Elegir un $\boldsymbol{\theta^{(0)}}$ con probabilidad positiva

$$\pi(\boldsymbol{\theta^{(0)}} \mid \boldsymbol{y}) > 0$$

Paso 2: Simular $\boldsymbol{\theta}^*$ de $J_t(\boldsymbol{\theta^*} \mid \boldsymbol{\theta^{(t-1)}})$

II) La distribución $J_t(\boldsymbol{\theta^*} \mid \boldsymbol{\theta^{(t-1)}})$ determina hacia donde debemos movernos en la siguiente iteración de la cadena de Markov.

III) El algoritmo de Metropolis requería que $J_t(\boldsymbol{\theta^*} \mid \boldsymbol{\theta^{(t-1)}})$ tenga distribución simétrica, esto es:

$$J_t(\boldsymbol{\theta^*} \mid \boldsymbol{\theta^{(t-1)}}) = J_t(\boldsymbol{\theta^{(t-1)}} \mid J_t\boldsymbol{\theta^*})$$

IV) Si tenemos una distribución simétrica que es dependiente en $\boldsymbol{\theta^{(t-1)}}$, entonces tenemos lo que se conoce como un **Metropolis Hasting Random Walk**.

Si $J_t(\boldsymbol{\theta^*} \mid \boldsymbol{\theta^{(t-1)}})$ no depende de $\boldsymbol{\theta^{(t-1)}}$,

$$J_t(\boldsymbol{\theta^*} \mid \boldsymbol{\theta^{(t-1)}}) = J_t(\theta^*)$$

entonces tenemos un **Metropolis Hasting de muestreo independiente**.

Básicamente, todos nuestros candidatos son obtenidos de la misma distribución, independiente de la muestra anterior.

Paso 3: Cálcular la tasa de aceptación r

$$r = \frac{\pi(\boldsymbol{\theta^*}|\boldsymbol{y})/J_t(\boldsymbol{\theta^*}|\boldsymbol{\theta^{(t-1)}})}{\pi(\boldsymbol{\theta^{(t-1)}}|\boldsymbol{y})/J_t(\boldsymbol{\theta^{(t-1)}}|\boldsymbol{\theta^*})}$$

y si nuestra distribución es simétrica entonces:

$$r = \frac{\pi(\boldsymbol{\theta^*}|\boldsymbol{y})}{\pi(\boldsymbol{\theta^{(t-1)}}|\boldsymbol{y})}$$

Si el candidato tiene una probabilidad más alta que el valor actual entonces este candidato es mejor que el actual y es aceptado. En caso que la distribución de salto no sea simétrica,

$$r = \frac{\pi(\boldsymbol{\theta}^*|\boldsymbol{y})/J_t(\boldsymbol{\theta}^*|\boldsymbol{\theta}^{(t-1)})}{\pi(\boldsymbol{\theta}^{(t-1)}|\boldsymbol{y})/J_t(\boldsymbol{\theta}^{(t-1)}|\boldsymbol{\theta}^*)}$$

Paso 4: Decidir la aceptación de $\boldsymbol{\theta}^*$

Aceptar $\boldsymbol{\theta}^*$ como $\boldsymbol{\theta}^{(t)}$ con probabilidad $min(r, 1)$. Si $\boldsymbol{\theta}^*$ no es aceptado, entonces $\boldsymbol{\theta}^{(t)} = \boldsymbol{\theta}^{(t-1)}$.

1) Por cada $\boldsymbol{\theta}^*$, obtenemos un valor de u desde una $\mathcal{U} \sim (0, 1)$
2) Si $u \leq r$, entonces se acepta $\boldsymbol{\theta}^*$ como $\boldsymbol{\theta}^{(t)}$. De otra forma, usar $\boldsymbol{\theta}^{(t-1)}$ como $\boldsymbol{\theta}^{(t)}$.

Paso 5: Razón de aceptación

Es importante monitorear la tasa de aceptación del algoritmo M-H (que es la fracción de candidatos muestreados aceptados).

Si la tasa es muy alta, la cadena probablemente no se está mezclando bien (no se está moviendo por el espacio parámetrico rápidamente)

Si la tasa es baja, el algoritmo es ineficiente (rechazando muchos candidatos del muestreo)

¿Que es alto y que es bajo? Esto depende el algoritmo....

- Random walk: entre 0.25 y 0.5 es recomendable
- Independiente: Muy cercano a 1 (preferiblemente)

Como se comentó previamente, es necesario diseñar cadenas que logren estabilidad a lo largo de un número grande de iteraciones, ya que despúes de este "equilibrio" nosotros podemos muestrear desde la distribución posterior de interés (generalmente los parámetros asumidos variables en el tiempo). Más adelante se detallarán algunos diagnósticos de convergencia para estas cadenas.

3.2. Algoritmo Gibbs - Sampling (G-S)

Suponiendo que tenemos una distribución conjunta conocida de parámetros $\pi(\theta_1, ...\theta_k)$. Si nosotros queremos muestrar aleatoriamente desde su distribución posterior con el fin de obtener la **distribución condicional** de cada parámetro, entonces nosotros podemos utilizar el algoritmo G-S para esta finalidad.

Es posible determinar la distribución condicional para cada parámetro dado el resto de parámetros y los datos observados, esto es:

$$\pi(\theta_i|\theta_{-i}, y)$$

El algoritmo G-S tiene la siguiente estructura para una sola iteración:

Algoritmo 2 Gibbs - Sampling (G-S)

1: Fijar un valor inicial $(\theta_1^{(1)}, ..., \theta_k^{(1)})$
2: Simular $\theta_1^{(2)} \sim \pi(\theta_1|\theta_2^{(1)}, ..., \theta_k^{(1)})$
3: Simular $\theta_2^{(2)} \sim \pi(\theta_2|\theta_1^{(2)}, \theta_3^{(1)} ..., \theta_k^{(1)})$
4: Simular $\theta_3^{(2)} \sim \pi(\theta_3|\theta_1^{(2)}, \theta_2^{(2)}, \theta_4^{(1)} ..., \theta_k^{(1)})$
5: y así sucesivamente....
6: Simular $\theta_k^{(n)} \sim \pi(\theta_k|\theta_1^{(n)}, ..., \theta_{k-1}^{(n)})$
7: Finalizar

¿Como podemos saber la $\pi(\theta)$ simplemente sabiendo la distribución condicional conjunta?

Téorema Hammersley - Clifford (para dos bloques)

Si se tiene $f(x, y)$, el téorema de Téorema Hammersley - Clifford prueba que podemos escribir la densidad conjunta en términos de las densidades condicionales $f(x|y)$ y $f(y|x)$ de la siguiente manera:

$$f(x, y) = \frac{f(y|x)}{\int \frac{f(y|x)}{f(x|y)} dy}$$

De la expresión anterior, podemos escribir el denominador como:

$$\begin{aligned} \int \frac{f(y|x)}{f(x|y)} dy &= \int \frac{\frac{(x,y)}{f(x)}}{\frac{(x,y)}{f(y)}} dy \\ &= \int \frac{f(y)}{f(x)} dy \\ &= \frac{1}{f(x)} \end{aligned}$$

Finalmente:

$$\begin{aligned} \frac{f(y|x)}{\frac{1}{f(x)}} &= f(y|x)\ f(x) \\ &= f(x, y) \end{aligned}$$

Este téorema nos ayuda a obtener la densidad conjunta a partir del conocimiento de las densidades condicionales y funciona correctamente para más de dos bloques de parámetros. Pero ¿cómo podemos averiguar los condicionales conjuntas?

Cálculo de las condicionales conjuntas

Supongamos que tenemos la distribución posterior $\pi(\boldsymbol{\theta}|\boldsymbol{y})$, entonces, para calcular la distribución condicional conjunta para cada θ debemos hacer:

a) Escribir la distribución posterior (ignorando la constante de proporcionalidad).

b) Tomar un bloque de parámetros (por ejemplo θ_1) y no considerar lo que no depende de θ_1.
c) Saber cual es la distribución condicional conjunta (dejando fuera la constante de normalización) $\pi(\theta_1 \mid \theta_{-1}, \boldsymbol{y})$
d) Repetir los pasos 2 y 3 para todo el conjunto de parámetros.

A continuación se detalla paso a paso el procecidimiento del algoritmo Gibbs Sampling:

Algoritmo G-S paso a paso

Queremos muestrear desde la $\pi(\boldsymbol{\theta}|\boldsymbol{y})$, donde $\boldsymbol{\theta}$ en este caso es un vector de 3 parámetros, $\theta_1, \theta_2, \theta_3$.

1) Elegir valores iniciales para el vector $\boldsymbol{\theta}^{(0)}$
2) Partir con cualquier valor de θ (θ_1 por conveniencia). Tomar $\theta_1^{(1)}$ desde la condicional conjunta $\pi(\theta_1|\theta_2^{(0)}, \theta_3^{(0)}, \boldsymbol{y})$
3) Tomar un valor de $\theta_1^{(2)}$ desde la condicional cojunta $\pi(\theta_2|\theta_1^{(1)}, \theta_3^{(0)}, \boldsymbol{y})$. Con el valor actualizado de $\theta_1^{(1)}$
4) Tomar un valor de $\theta_1^{(3)}$ desde la condicional conjunta $\pi(\theta_3|\theta_1^{(1)}, \theta_2^{(1)}, \boldsymbol{y})$. Con valores actualizados de $\theta_1^{(1)}$ y $\theta_2^{(1)}$
5) Sortear $\boldsymbol{\theta}^{(2)}$ utilizando $\boldsymbol{\theta}^{(1)}$ (y así sucesivamente con los valores del vector de $\boldsymbol{\theta}$ actualizado).
6) Repetir hasta que tengamos M muestras con cada muestra siendo un vector de $\boldsymbol{\theta}^{(t)}$

3.3. Algoritmo Hamiltoniano Monte Carlo (HMC)

Este método (también conocido como Hybrid Monte Carlo (Monte Carlo Híbrido)) fue propuesto originalmente por Duane *et al.*, (1987) para simular la dinámica de sistemas moleculares. Téoricamente es aplicado en espacios de estados continuos y utiliza la gradiente para reducir la incertidumbre de la caminata aleatoria. Neal (1995) fue el primer investigador quien lo utilizó para aproximar una solución en problemas estadísticos (espécificamente en redes neuronales Bayesianas) pero MacKay (2003) fue quien lo nombró como Hamiltoniano Monte Carlo (HMC). Desafortunadamente, la teoría de HMC está desarrollada en términos de geometría diferencial, lo que hace que su construcción formal requiera de matemáticas avanzadas para una explicación formal (Betancourt, M. 2017).

Para comenzar vamos a recordar el funcionamiento del algoritmo de M-H. La distribución propuesta está centrada simétricamente en una posición actual, por lo que en un espacio multiparámetrico, esta distribución propuesta generalmente es una Normal multivarada. Si seguimos lo anterior entonces deberíamos especificar una media y una matriz de varianzas-covarianzas que nos permitan una correcta eficiencia computacional en las estimaciones, ya que la distribución elegida (Normal), siempre nos asegura estar centrada en algún punto en particular (en este caso el punto actual). Sin embargo esto nos puede llevar a estimaciones poco robustas si, por ejemplo, nos ubicamos en las colas de la distribución posterior para hacer el muestreo. Esto por que el paso siguiente en el algoritmo (con misma probabilidad entre uno y otro) estaría en función de esta elección (alejandose o acercandose de la moda de la distribución posterior). Ya que en algoritmo M-H la forma de la distribución instrumental es siempre la misma, sin importar dónde se encuentre en el espacio de parámetros, esto nos lleva a realizar una gran cantidad de muestras sin sentido y de ineficiencia

en los cálculos. En cambio el HMC resuelve la dirección en la que aumenta la densidad posterior (*gradiente*) y "moldea" la distribución propuesta en esa dirección.

El HMC se basa en la téoria física de la energía, primero calcula la densidad log-posterior negativa no-normalizada ("*función potencial*" o también conocida como "*energía potencial*") para luego generar una distribución propuesta similar al movimiento de una canica (mólecula) en una superficie de la función potencial sin fricción (que es la ubicación del último valor del parámetro aceptado θ_{act}). A continuación la canica rueda por la superficie de la función potencial en un tiempo determinado para detenerse en algún momento que será nuestro nuevo parámetro propuesto (θ_{prop})

Si este movimiento aleatorio se repitiera "n" cantidad de veces, la misma aleatoriedad produciría "n" trayectorias distintas en que la canica se desplaza en la superficie de la función potencial hasta un punto terminal, entonces estos puntos terminales nosotros podemos conceptualizarlos como la distribución propuesta por el HMC para la posición actual.

Por lo tanto, modificando la probabilidad de "aceptación/rechazo" del algoritmo M-H en el algoritmo HMC, ahora tendremos en cuenta no sola la densidad posterior relativa sino también el *momento* (denotado por ϕ en las posiciones actual y propuesta del algoritmo).

$$r = \text{mín}\left(\frac{\ell(\boldsymbol{y} \mid \theta_{prop})\pi(\theta_{prop})\pi(\phi_{prop})}{\ell(\boldsymbol{y} \mid \theta_{act})\pi(\theta_{act})\pi(\phi_{act})}, 1{,}0\right)$$

En un sistema real la suma de la energía potencial con la energía cinética es constante por lo que la razón en la equación de arriba sería igual a 1 y, en consecuencia, no deberíamos rechazar nunca la propuesta. Así, en la práctica, el sistema continuo se discretiza dentro de pequeños intervalos (en este caso los pasos del MCMC) para que los cálculos sean aproximados y algunas propuestas del algoritmo también sean rechazadas.

4. Diagnósticos de convergencia

Es bueno recordar que el objetivo detrás del método MCMC es lograr una representación de la verdadera pero, analíticamente incalculable, distribución a posterior. La idea es demostrar de alguna manera que las trayectorias de las cadenas han alcanzado la distribución a posterior en equilibrio (convergencia).

Los diagnósticos presentados a continuación pueden ser confiables en terminos de representar númerica y graficamente que las muestras están convergiendo en una representación razonable de la distribución a posterior. Si bien no siempre son suficientes si nos permiten asegurar la representatividad de la convergencia.

a) *Trace plots*: Representan una gráfica que muestra el número de iteraciones junto con los valores generados por la cadena (pueden ser más de una cadena representada gráficamente). Si se aprecia visualmente la no existencia de tendencias o variaciones, entonces la cadena (s) ha convergido hacia la distribución posterior, lo que indicaría la estacionarierad alrededor de un mismo valor modal. Este gráfico se puede obtener mediante la librería `coda` o la librería `bayesplot` mediante el lenguaje de programación estadística Stan (Stan Development Team. 2018)

b) Gráficas de la media ergódica: Cálculo de la media hasta la iteración "n" del algoritmo propuesto. Se puede revisar la librería `coda` para obtener estas gráfica.

c) Gráficos de autocorrelación: De acuerdo a los algoritmos presentados anteriormente, sabemos que estos están correlacionados ya que todas los valores propuestos para θ no siempre son aceptados. Por ejemplo, en el HMC el valor propuesto para θ_{prop} se adapta a la dirección que lleva la distribución a posterior, el valor del parámetro actual influye en la distribución propuesta para el siguiente. Este tipo de diagnóstico nos ayudará a verificar la eficiencia de nuestro algoritmo propuesto en el MCMC (`ggMCMC`, `bayesplot`)

d) Estadísticos de convergencia:

Gelman-Rubin

Compara las varianzas entre k número de cadenas. El valor de este estadístico debe ser cercano a 1 para probar un correcto funcionamiento de las cadenas dentro del algoritmo (`coda`, `bayesplot`)

$$\hat{R} = \sqrt{\widehat{Var}(\theta)}/W$$

donde $W = 1/n = \sum_{i=1}^{m} s_i^2$ es la varianza dentro de las cadenas. Además,

$$\widehat{Var}(\theta) = \left(1 - \frac{1}{n}\right) W + \frac{1}{n} B$$

donde B es la varianza entre cadenas expresada como $B = \frac{1}{m-1} \sum_{i=1}^{m} (\bar{\theta}_i - \bar{\theta})^2$

Estadístico de Geweke

Comprueba si la media del primer 10 % de la cadena es muy distina de la media del 50 % de la cadena restante. El valor representa a un z que es evaluado en una hipótesis de comparación de dos medias, y para valores fuera de -2.5 y 2.5, indican que la cadena no es estacionaria (`boa`, `ggmcmc`)

Tasa de aceptación (Acceptance Ratio)

Se puede definir como la relación entre el número de valores únicos en la cadena y el número total de muestras en la cadena. Esto es equivalente al porcentaje de valores de parámetros propuestos, θ_{prop}, que son aceptados mediante el método MCMC (`bayesImageS`)

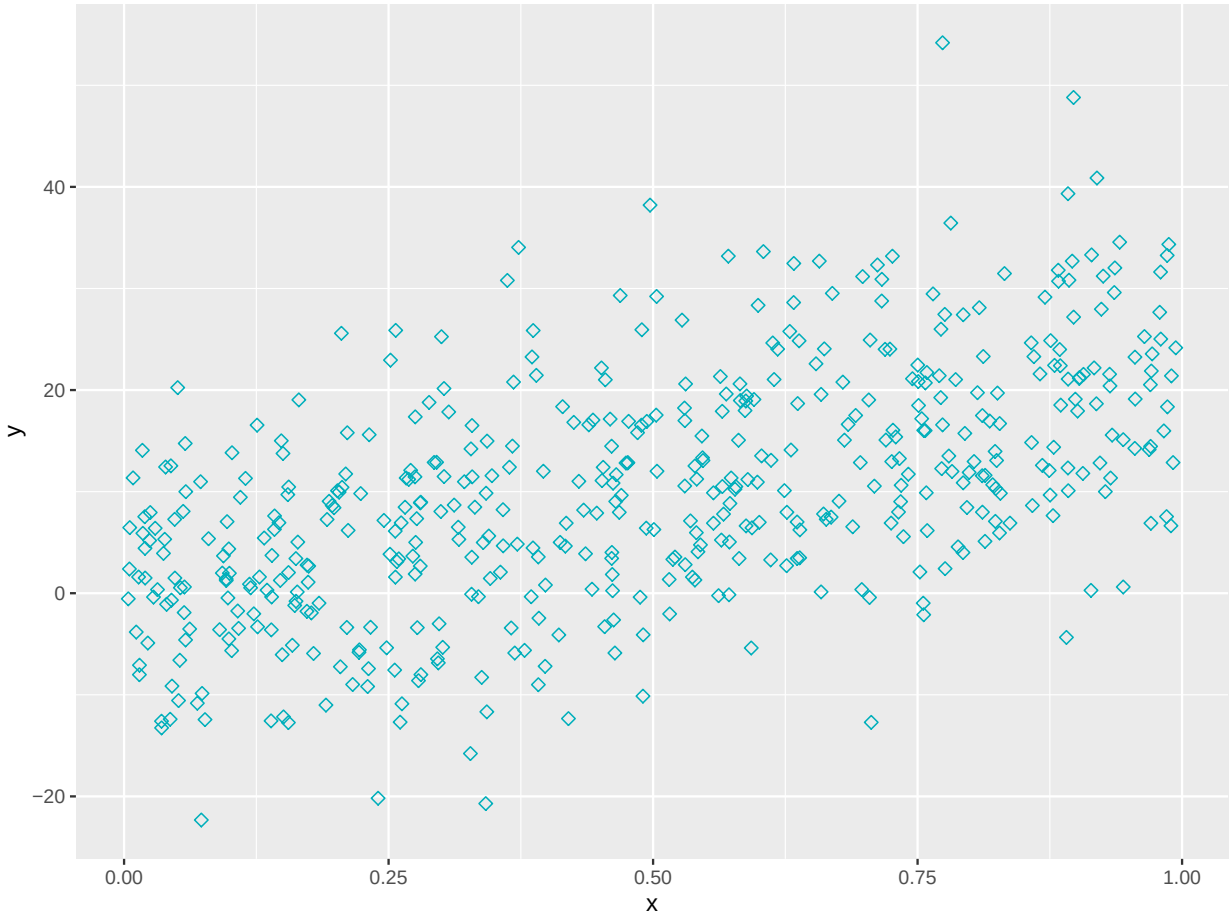
5. Ejemplos

Se presentan dos ejemplos de estimación bayesiana, uno mediante el algoritmo Gibb Sampling y otro mediante el lenguaje de programación Stan.

El modelo a evaluar es un clásico modelo de regresión lineal:

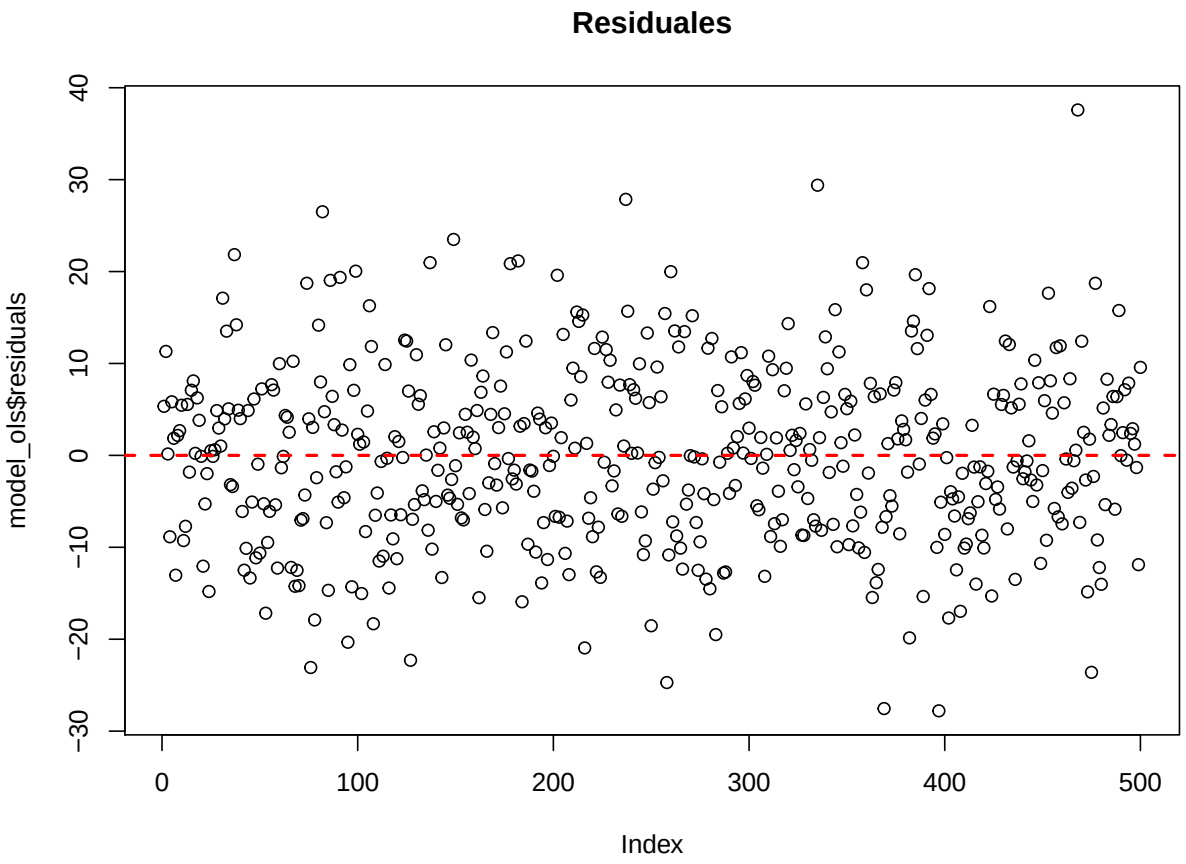
$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon} \tag{4}$$

```
#=============================================================================
#                       Simulación de datos para el ejercicio
#=============================================================================
library(ggplot2)
mu0 = 0
sigma0 = 10
n = 500
e = rnorm(n, mu0, sigma0)
x = runif(n)
y = 20*x + e

data = data.frame(x, y)

plot = ggplot(data, aes(x = x, y = y))
plot + geom_point(color = "#00AFBB", size = 2, shape = 23)
```

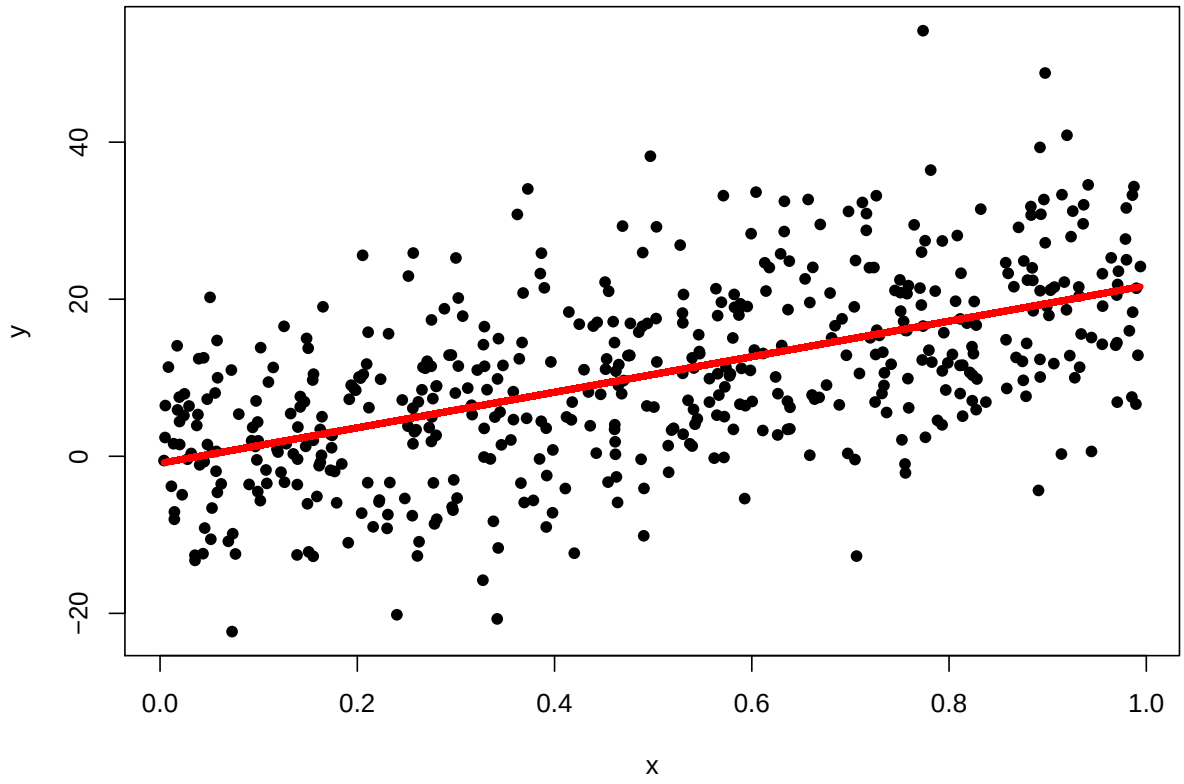



```r
#=============================================================================
#                Mínimos Cuadrados Ordinarios (OLS, siglas en inglés)
#-----------------------------------------------------------------------------
model_ols = lm(y ~ x)
names(model_ols)

model_ols$coefficients              # Coeficientes del modelo (parámetros)
model_ols$residuals                 # Residuales del modelo
resid = data.frame(model_ols$residuals)
```

```r
#=============================================================================
#              Gráficamos los residuales para ver supuesto de normalidad
#-----------------------------------------------------------------------------
plot(model_ols$residuals, main ="Residuales")
abline(h=0, lty=2, col="red", lwd=2)
```


Residuales
model_ols$residuals
40
30
20
10
0
−10
−20
−30
0
100
200
300
400
500
Index

```r
#=============================================================================
#                              Ajuste del modelo
#-----------------------------------------------------------------------------
plot(x, y, pch=16)
lines(x,  model_ols$fitted, col=2,lwd=4)
```



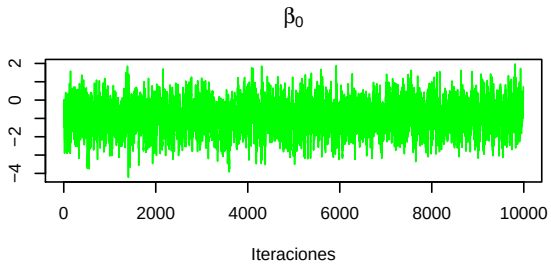

```r
#=============================================================================
#                  Matriz de diseño para utilizar luego con Gibbs Sampling
#-----------------------------------------------------------------------------
X = cbind(1,x)


#=============================================================================
#                              Algoritmo Gibbs Sampling
#=============================================================================

#===================================
#                  Priors
#===================================
mu_0 = 0
sigma2_0 = 100
```

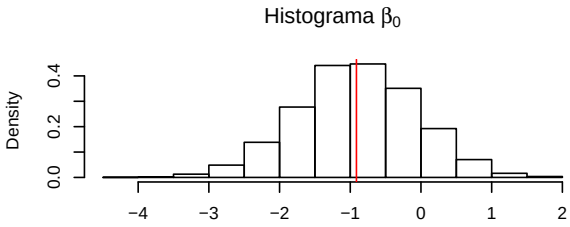
```r
a = 0.01      # parámetro a Gamma inversa para sigma2
b = 0.01      # parámetro b Gamma inversa para sigma2

#==================================
#         Valores iniciales
#==================================
b0 = model_ols$coef[1]
b1 = model_ols$coef[2]
s2 = var(model_ols$residuals)

#==================================
#   Matriz de valores del MCMC
#==================================
n_sims = 10000
mat_ini = matrix(0, n_sims, 3)
colnames(mat_ini) = c("beta0", "beta1", "sigma2")
mat_ini[1,] = c(b0, b1, s2) # Valores iniciales

#============================================
#              MCM Gibbs Sampling
#============================================
for(iter in 2:n_sims){

#--------------------------------------------
#             (b0 | b1, s2, y)
#--------------------------------------------
step1  =  sum(y - x*b1)/s2 + 1/sigma2_0
step2  =  n/s2 + mu_0/sigma2_0
b0 =  rnorm(1, step1/step2, 1/sqrt(step2))

#--------------------------------------------
#             (b1 | b0, s2, y)
#--------------------------------------------
step1  = sum(x*(y - b0))/s2 + 1/sigma2_0
step2  = sum(x^2)/s2 + mu_0/sigma2_0
b1 = rnorm(1, step1/step2, 1/sqrt(step2))

#--------------------------------------------
#             (s2 | b0, b1, y)
#--------------------------------------------
step1  = n/2 + a
step2  = sum((y - b0 - x*b1)^2)/2 + b
s2 = 1/rgamma(1, step1, step2)

  mat_ini[iter,] = c(b0, b1, s2)
} ##
```

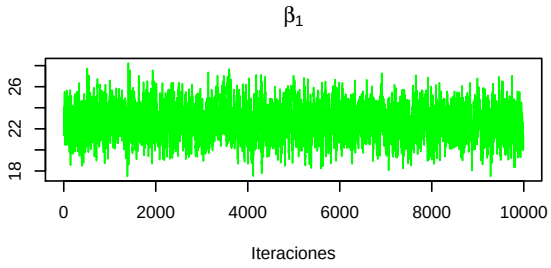
```r
salidas <- matrix(0, 3, 4)
rownames(salidas) <- c("Intercept", "x", "sigma2")
colnames(salidas) <- c("Mean", "SD", "Q025", "Q975")

salidas[,1] = apply(mat_ini, 2, mean)
salidas[,2] = apply(mat_ini, 2, sd)
salidas[,3] = apply(mat_ini, 2, quantile, 0.025)
salidas[,4] = apply(mat_ini, 2, quantile, 0.975)

library(knitr)
kable(salidas, digits = 3)
```

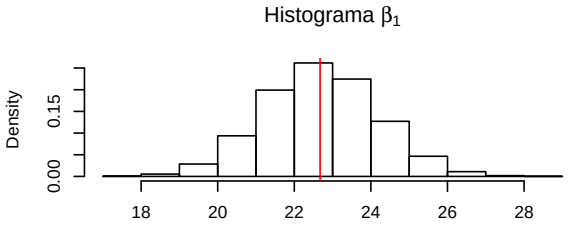
	Mean	SD	Q025	Q975
Intercept	-0.913	0.844	-2.585	0.704
x	22.674	1.485	19.719	25.607
sigma2	93.580	5.898	82.770	105.714

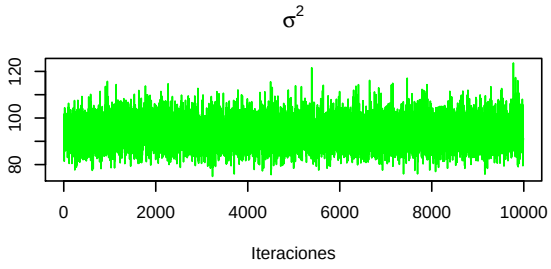
```r
# Aplicando sqrt(salidas[3, ]) se obtiene sigma

#=============================================================================
par(mfrow=c(3,2))
plot(mat_ini[10:n_sims,1], xlab ="Iteraciones", ylab="", main = expression(beta[0]),
     type="l", col = "green")
hist(mat_ini[10:n_sims,1], xlab = "", main = expression(paste("Histograma", " ",
     beta[0])), prob=TRUE)
abline(v = mean(mat_ini[,1]), col = 2)

plot(mat_ini[10:n_sims,2], xlab ="Iteraciones", ylab="", main = expression(beta[1]),
     type="l", col = "green")
hist(mat_ini[10:n_sims,2], xlab = "", main = expression(paste("Histograma", " ",
     beta[1])),prob=TRUE)
abline(v = mean(mat_ini[,2]), col = 2)

plot(mat_ini[10:n_sims,3], xlab ="Iteraciones",ylab="", main = expression(sigma^2),
     type="l", col = "green")
hist(sqrt(mat_ini[10:n_sims,3]), xlab = "", main = expression(paste("Histograma", " ",
     sigma)), prob=TRUE)
abline(v = mean(mat_ini[,3]), col = 2)
```

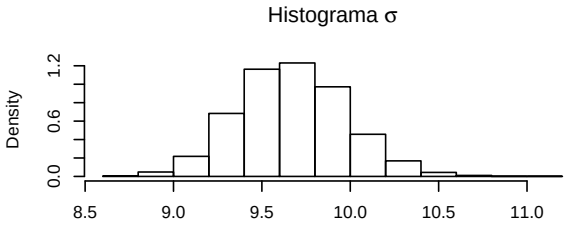













```r
#=============================================================================================
#                                Hamiltoniano Monte Carlo (HMC)
##============================================================================================
# Cargamos las librerías
library("rstan") # observe startup messages
options(mc.cores = parallel::detectCores())
rstan_options(auto_write = TRUE)
Sys.setenv(LOCAL_CPPFLAGS = '-march=native')

library(dplyr)
library(ggplot2)
library(reshape)
```

```r
#=============================================================================
#                    Repetimos data.frame del anterior algoritmo
#_____________________________________________________________________________
n = 500
B0 = as.numeric(model_ols$coefficients[1])
B1 = as.numeric(model_ols$coefficients[2])
mu0 = 0
sigma0 = 10
```

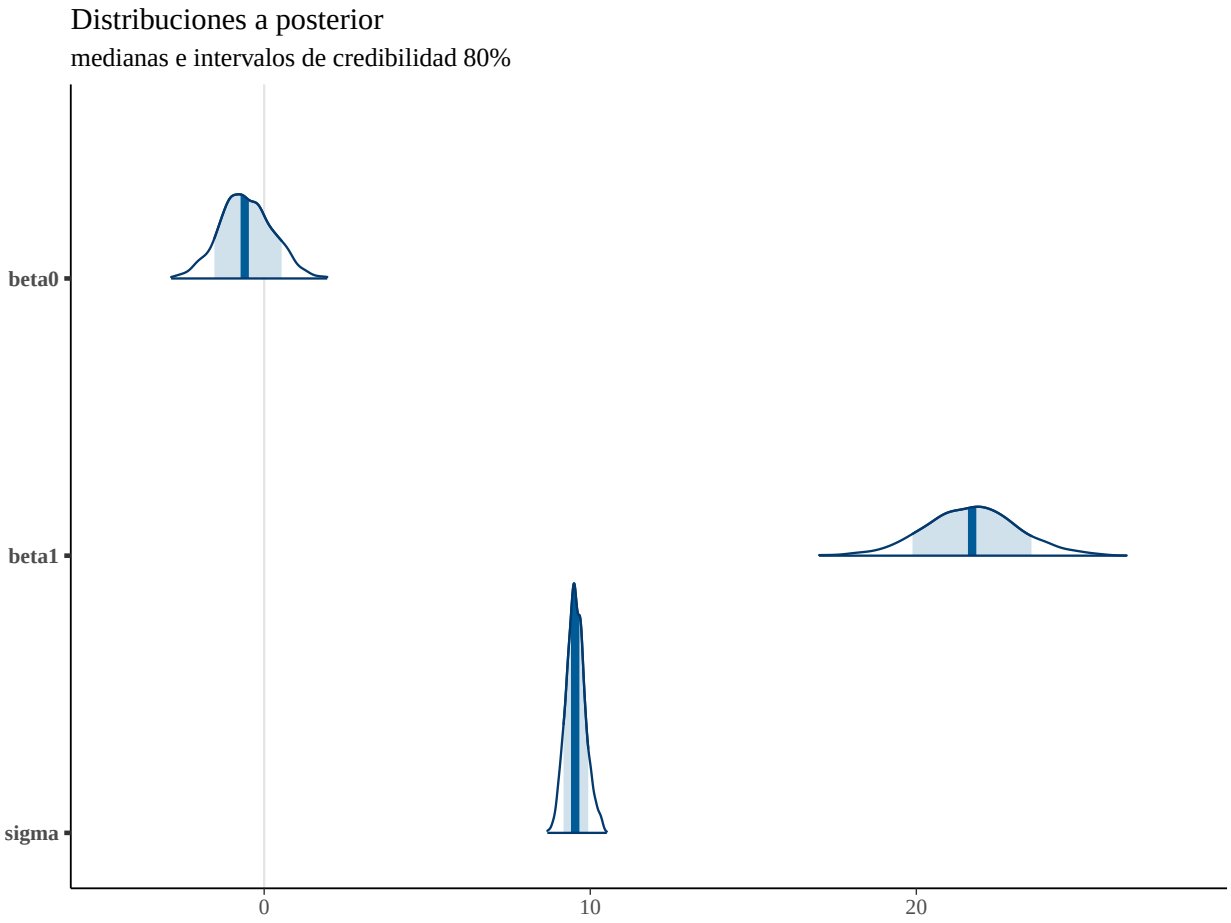
```r
e = rnorm(n, mu0, sigma0)

x = runif(n)
y = B0 + B1 * x + e
#=============================================================================


#=============================================================================
#                              MODELO LINEAL con Stan
#=============================================================================
lm_stan = '
  data {
  int<lower=0> N;
  vector [N] y;
  vector [N] x;
  }
  parameters {
  real beta0;
  real beta1;
  real<lower=0> sigma;
  }
  model {
  vector [N] mu;

  #Priors
  beta0 ~ normal(0,10000);
  beta1 ~ normal(0,10000);
  sigma ~ cauchy(0,5);

  mu = beta0 + beta1*x;

  //Likelihood
  y~normal(mu,sigma);
  }
'
#=============================================================================
#                       Hacemos una lista para nuestros datos
#-----------------------------------------------------------------------------
data_stan = list(N = length(x), x = x, y = y)
```

```r
#=============================================================================
#             Ajuste del modelo con priors en los hyperparametros
#-----------------------------------------------------------------------------
fit_stan <- stan(model_code = lm_stan, data = data_stan,
                 iter = 1000, chains = 3, warmup = 500)
```

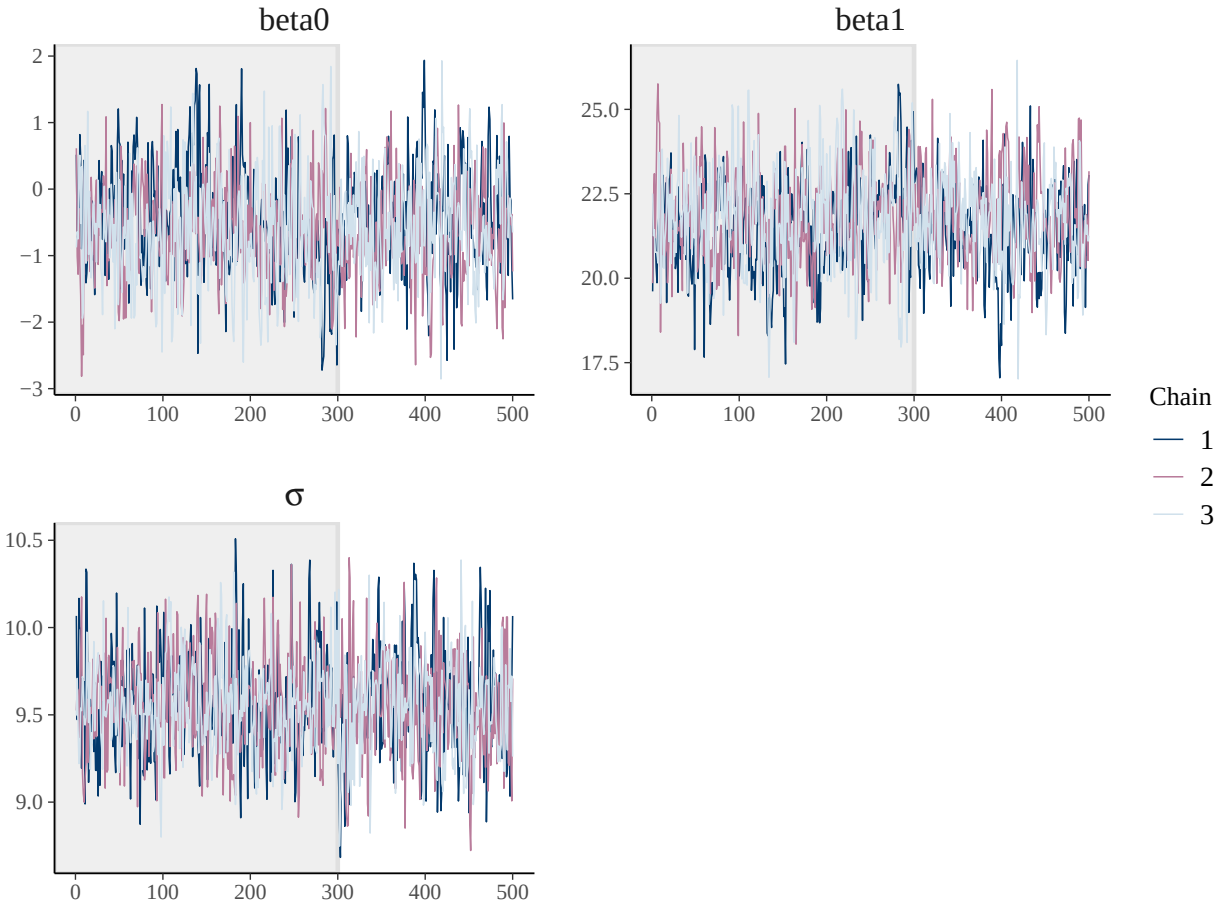
```r
print(fit_stan, digits = 2)
```

```
## Inference for Stan model: dda515e7b6a2d0a443cbab5448726183.
## 3 chains, each with iter=1000; warmup=500; thin=1;
## post-warmup draws per chain=500, total post-warmup draws=1500.
##
##           mean se_mean   sd     2.5%      25%      50%      75%    97.5%
## beta0    -0.55    0.04 0.81    -2.09    -1.10    -0.60    -0.01     1.07
## beta1    21.70    0.07 1.43    18.86    20.75    21.71    22.64    24.62
## sigma     9.55    0.01 0.29     9.01     9.36     9.54     9.73    10.17
## lp__  -1377.65    0.05 1.20 -1380.89 -1378.18 -1377.36 -1376.76 -1376.31
##       n_eff Rhat
## beta0   471 1.01
## beta1   407 1.01
## sigma   966 1.00
## lp__    556 1.00
##
## Samples were drawn using NUTS(diag_e) at Fri Nov 22 12:12:30 2019.
## 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).
```

```
#=============================================================================
#                    Traceplot con la librería bayesplot
#-----------------------------------------------------------------------------
library(bayesplot)
results = as.array(fit_stan)
plot_title <- ggtitle("Distribuciones a posterior",
                      "medianas e intervalos de credibilidad 80% ")
mcmc_areas(results,
           pars = c("beta0", "beta1", "sigma"),
           prob = 0.8) + plot_title
```

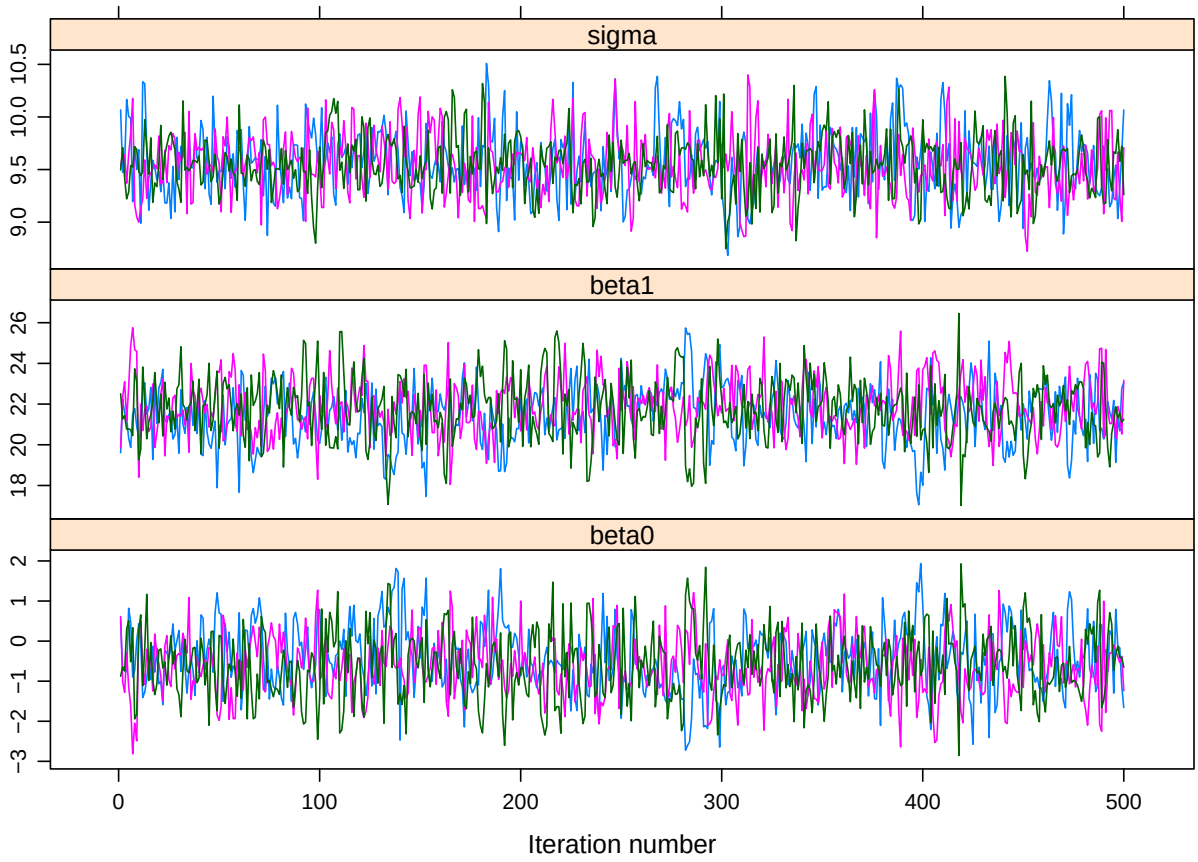



```
color_scheme_set("mix-blue-pink")
p <- mcmc_trace(results,  pars = c("beta0", "beta1", "sigma"), n_warmup = 300,
                facet_args = list(nrow = 2, labeller = label_parsed))
p + facet_text(size = 15)
```




```
require(lattice)
stan2coda = function(fit, pars = names(fit)) {
fit.list <- lapply(1:ncol(fit), function(x) mcmc(as.array(fit)[,x,which(pars %in% names(fit))])
mcmc.list(fit.list)
}
```

```
# Con librería coda
library(coda)
fit_stan_mcmc <- stan2coda(fit_stan, c("beta0", "beta1","sigma"))
xyplot(fit_stan_mcmc)
```




```
#================================================================================
##                        Comparación de modelos
#================================================================================

## Minimos cuadrados ordinarios (OLS)
library(magrittr)
library(tableone)
tableone::ShowRegTable(model_ols, exp = FALSE)
```

```
##             coef [confint]       p
## (Intercept) -0.90 [-2.55, 0.76]   0.289
## x           22.63 [19.71, 25.54] <0.001
```

```
summary(model_ols)$sigma
```

```
## [1] 9.651519
```

```
## Bayesian
print(fit_stan, pars = c("beta0", "beta1", "sigma"))
```

```
## Inference for Stan model: dda515e7b6a2d0a443cbab5448726183.
## 3 chains, each with iter=1000; warmup=500; thin=1;
## post-warmup draws per chain=500, total post-warmup draws=1500.
```

```
##
##        mean se_mean   sd  2.5%   25%   50%   75% 97.5% n_eff Rhat
## beta0 -0.55    0.04 0.81 -2.09 -1.10 -0.60 -0.01  1.07   471 1.01
## beta1 21.70    0.07 1.43 18.86 20.75 21.71 22.64 24.62   407 1.01
## sigma  9.55    0.01 0.29  9.01  9.36  9.54  9.73 10.17   966 1.00
##
## Samples were drawn using NUTS(diag_e) at Fri Nov 22 12:12:30 2019.
## 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).
```