

Simulación

3. Monte Carlo a través de Cadenas de Markov (MCMC) Gibbs Sampler

Jorge de la Vega Góngora

Departamento de Estadística,
Instituto Tecnológico Autónomo de México

Clase 10



Gibbs Sampler

Gibbs Sampler I

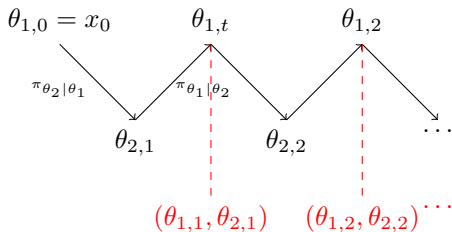
- El Gibbs Sampler es un caso particular del algoritmo de Metropolis-Hastings.
- Se aplica cuando la distribución objetivo $\pi(\boldsymbol{\theta})$ de la que se quiere tomar una muestra es una *distribución multivariada*.
- Idea:
 - $\boldsymbol{\theta}$ se puede particionar en k subvectores: $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_k)$
 - La cadena de Markov se genera a través de muestrear de las *distribuciones condicionales* de la distribución objetivo:

$$\pi(\boldsymbol{\theta}_j | \boldsymbol{\theta}_{-j}, y), \text{ donde } \boldsymbol{\theta}_{-j} = \boldsymbol{\theta} \setminus \boldsymbol{\theta}_j, \quad j = 1, \dots, k$$

iterando sobre los k subvectores de parámetros $\boldsymbol{\theta}_{-j}$.

- En el caso bidimensional, $\boldsymbol{\theta} = (\theta_1, \theta_2) \sim \pi(\theta_1, \theta_2)$. Se genera una cadena de Markov $(\theta_{1t}, \theta_{2t})$ del siguiente modo. Para $n = 1, 2, \dots$:
 - Se inicializa la cadena $\theta_{1,0} = x_0$.
 - Genera $\theta_{2t} \sim \pi_{\theta_2|\theta_1}(\cdot | \theta_{1,t-1})$.
 - Genera $\theta_{1t} \sim \pi_{\theta_1|\theta_2}(\cdot | \theta_{2t})$
 - Incrementa t a $t + 1$ hasta n

- En el caso del Gibbs sampler, a diferencia del Metropolis-Hastings, *cada candidato generado es aceptado*.
- Una representación gráfica del flujo del algoritmo se muestra a continuación:



Ejemplo 1 GS Normal bivariada I

Consideremos una normal bivariada $\mathcal{N}_2(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ con $\boldsymbol{\mu} = (\mu_1, \mu_2)$ y $\boldsymbol{\Sigma} = \begin{pmatrix} \sigma_1^2 & \rho \\ \rho & \sigma_2^2 \end{pmatrix}$. En este caso es fácil conocer las marginales:

$$\begin{aligned} Y_{n+1}|X_n &\sim \mathcal{N}\left(\mu_2 + \rho \frac{\sigma_2}{\sigma_1}(X_n - \mu_1), (1 - \rho^2)\sigma_2^2\right) \\ X_{n+1}|Y_{n+1} &\sim \mathcal{N}\left(\mu_1 + \rho \frac{\sigma_1}{\sigma_2}(Y_{n+1} - \mu_2), (1 - \rho^2)\sigma_1^2\right) \end{aligned}$$

Entonces $(X_n, Y_n) \rightarrow \mathcal{N}_2(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. A continuación se muestra la simulación para el caso $\rho = 0.5$, $\mu_1 = 0$, $\mu_2 = 2$, $\sigma_1 = 1$ y $\sigma_2 = 2$.

Ejemplo 1 GS Normal bivariada II

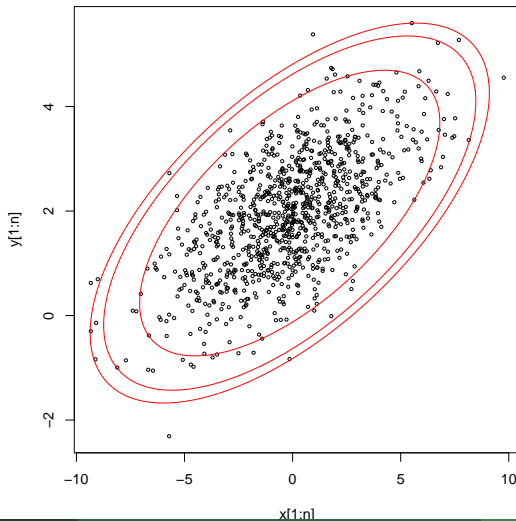
```
library(mixtools) #para la función ellipse.
```

mixtools package, version 1.1.0, Released 2017-03-10

This package is based upon work supported by the National Science Foundation under Grant No. SES-0518772.

```
par(pty="s") #plot cuadrado
n <- 1000; x0 <- 0 #valor inicial
x <- NULL; y <- NULL
x <- append(x0,x)
rho <- 0.5; mu1 <- 0; mu2 <- 2; sigma1 <- 1; sigma2 <- 2 #valores dados
for(i in 1:n){
  y <- append(y, rnorm(1,mean=mu2 + rho*sigma1/sigma2*(x[i]-mu1),sd=sqrt(1-rho^2))*sigma1)
  x <- append(x, rnorm(1,mean=mu1 + rho*sigma2/sigma1*(y[i]-mu2),sd=sqrt(1-rho^2))*sigma2)
}
plot(x[1:n],y[1:n],cex=0.5)
ellipse(mu = colMeans(cbind(x[1:n],y[1:n])), sigma = cov(cbind(x[1:n],y[1:n])),
  alpha = 0.005, npoints = 300, col="red")
ellipse(mu = colMeans(cbind(x[1:n],y[1:n])), sigma = cov(cbind(x[1:n],y[1:n])),
  alpha = 0.01, npoints = 300, col="red")
ellipse(mu = colMeans(cbind(x[1:n],y[1:n])), sigma = cov(cbind(x[1:n],y[1:n])),
  alpha = 0.05, npoints = 300, col="red")
```

Ejemplo 1 GS Normal bivariada III



Si se particiona el vector \mathbf{X} en k componentes $\mathbf{X} = (\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_k)$, y podemos simular de $f_i(\mathbf{x}_i | \mathbf{x}_{-i})$, entonces podemos aplicar el siguiente algoritmo:

Algoritmo Gibbs sampler

- 1 Inicializar $\mathbf{X}^{(0)} = (\mathbf{x}_1^{(0)}, \mathbf{x}_2^{(0)}, \dots, \mathbf{x}_k^{(0)})$ y $j = 1$
- 2 Obtener $\mathbf{X}^{(j)}$ a partir de $\mathbf{X}^{(j-1)}$ generando

$$\begin{aligned}\mathbf{x}_1^{(j)} &\sim f(\mathbf{x}_1 | \mathbf{x}_2^{(j-1)}, \dots, \mathbf{x}_k^{(j-1)}) \\ \mathbf{x}_2^{(j)} &\sim f(\mathbf{x}_2 | \mathbf{x}_1^{(j)}, \mathbf{x}_3^{(j-1)}, \dots, \mathbf{x}_k^{(j-1)}) \\ \mathbf{x}_3^{(j)} &\sim f(\mathbf{x}_3 | \mathbf{x}_1^{(j)}, \mathbf{x}_2^{(j)}, \dots, \mathbf{x}_k^{(j-1)}) \\ &\vdots \\ \mathbf{x}_{k-1}^{(j)} &\sim f(\mathbf{x}_{k-1} | \mathbf{x}_1^{(j)}, \mathbf{x}_2^{(j)}, \dots, \mathbf{x}_k^{(j-1)}) \\ \mathbf{x}_k^{(j)} &\sim f(\mathbf{x}_k | \mathbf{x}_1^{(j)}, \mathbf{x}_2^{(j)}, \dots, \mathbf{x}_{k-1}^{(j)})\end{aligned}$$

- 3 Hacer $j := j + 1$ y repetir el paso 2 hasta que la sucesión $\{\mathbf{X}^{(n)}\}$ converja a la distribución objetivo.

Ejemplo, Casella y George (1992) I

Supongan que X , P y N son tres variables aleatorias con densidad conjunta:

$$\pi(x, p, n) \propto \binom{n}{x} p^x (1-p)^{n-x} \frac{4^n}{n!}$$

para $x \in \{0, 1, \dots, n\}$, $p \in (0, 1)$, $n \in \mathbb{N}$. ¿Cómo podemos obtener una muestra aleatoria de esta distribución? Notar que X y N son dependientes.

Con esta estructura, podemos establecer el muestreo de Gibbs de la siguiente manera:

- $X|\{P = p, N = n\} \propto \mathbf{Bin}(n, p)$.
- $P|\{X = x, N = n\} \propto \mathcal{Be}(x + 1, n - x + 1)$.
- $N|\{X = x, P = p\} \propto \frac{(1-p)^{n-x} 4^n}{(n-x)!}$ para $n = x, x + 1, \dots$. Esta es una distribución Poisson desplazada con parámetro $4(1-p)$, es decir, $N|\{P = p, X = x\} \sim Z + x$ donde $Z \sim \mathcal{P}(4(1-p))$.

De acuerdo al algoritmo dado, la implementación se realiza de la siguiente manera:

- Inicializar $(x_0, p_0, n_0) \leftarrow (1, 0.5, 2)$

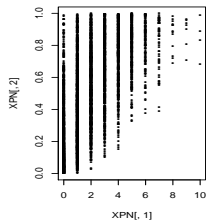
Ejemplo, Casella y George (1992) II

- Para $m = 1, \dots, M$
 - $x_m \sim \mathbf{Bin}(n_{m-1}, p_{m-1})$
 - $p_m \sim \mathcal{Be}(x_m + 1, n_{m-1} - x_m + 1)$
 - $z \sim \mathcal{P}(4(1 - p_m))$ y $n_m = z + x_m$

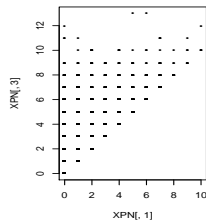
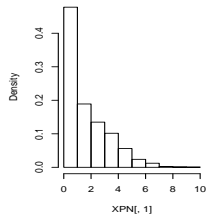
El algoritmo anterior nos da la secuencia de valores $(x_0, p_0, n_0), (x_1, p_1, n_1), \dots, (x_M, p_M, n_M)$. El siguiente código grafica las distribuciones bivariadas y las marginales en base a la muestra.

```
M <- 5000
XPN <- matrix(rep(0,3*M),ncol=3) #Guarda los vectores con variables X, P, N
XPN[1,] <- c(1,0.5,2) #inicialización
for(i in 2:M) {
  XPN[i,1] <- rbinom(1,XPN[i-1,3],XPN[i-1,2])
  XPN[i,2] <- rbeta(1,XPN[i,1]+1, XPN[i-1,3]-XPN[i,1]+1)
  XPN[i,3] <- rpois(1,4*(1-XPN[i,2])) + XPN[i,1]
}
par(mfrow=c(2,3))
plot(XPN[,1],XPN[,2],pch=16,cex=0.5)
plot(XPN[,1],XPN[,3],pch=16,cex=0.5)
plot(XPN[,2],XPN[,3],pch=16,cex=0.5)
hist(XPN[,1],prob=T)
hist(XPN[,2],prob=T)
hist(XPN[,3],prob=T)
```

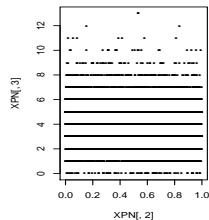
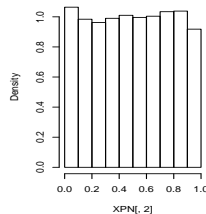
Ejemplo, Casella y George (1992) III



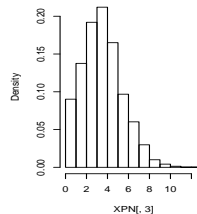
Histogram of $XPN[, 1]$



Histogram of $XPN[, 2]$



Histogram of $XPN[, 3]$



¿Porqué funciona el Gibbs sampler? I

- El Gibbs sampler funciona porque es un caso especial del algoritmo de Metropolis-Hastings.
- Para verlo, consideremos la actualización de uno de los subvectores en la iteración para mostrar la lógica. Todos los pasos en cada variable siguen el mismo razonamiento.
- Sea $i = (x_1, x_2, \dots, x_k) \in \mathcal{S}$ el estado actual.
- La distribución candidata $q(\cdot|\cdot)$ será la distribución condicional de X_1 dados X_2, \dots, X_k . Con esta distribución, se obtiene $j = (x_1^*, x_2, \dots, x_k)$, el estado propuesto.

¿Porqué funciona el Gibbs sampler? II

- La probabilidad de aceptación del paso es $\alpha(i, j) = \frac{\pi_j q(i|j)}{\pi_i q(j|i)} = \frac{\pi_j q_{ji}}{\pi_i q_{ij}}$. Entonces

$$\begin{aligned}\pi_j q_{ji} &= \pi(x_1^*, x_2, \dots, x_k) f_{X_1|X_2, \dots, X_k}(x_1|x_2, \dots, x_k) \\ &= \pi(x_1^*, x_2, \dots, x_k) \left(\frac{\pi(x_1, x_2, \dots, x_k)}{\int \pi(x, x_2, \dots, x_k) dx} \right) \\ &= \pi(x_1, x_2, \dots, x_k) \left(\frac{\pi(x_1^*, x_2, \dots, x_k)}{\int \pi(x, x_2, \dots, x_k) dx} \right) \\ &= \pi(x_1, x_2, \dots, x_k) f_{X_1|X_2, \dots, X_k}(x_1^*|x_2, \dots, x_k) \\ &= \pi_i q_{ij}\end{aligned}$$

Así que $\alpha(i, j) = 1$, lo que implica que con esta elección de función q el estado propuesto siempre se acepta.

Ejemplo 2: GS en modelos jerárquicos I

Modelo jerárquico Binomial-beta

El Gibbs sampler se hizo muy popular en los 90's por sus aplicaciones en estadística Bayesiana, sobre todo en los problemas de estimación de modelos jerárquicos, ya que se cuenta con las marginales de manera explícita.

Ejemplo

- Consideren el modelo jerárquico

$$x|\theta \sim \mathbf{Bin}(n, \theta)$$

$$\theta \sim \mathcal{Be}(a, b)$$

- Los parámetros n , a y b son *parámetros molestos (nuisance)* o *hiperparámetros*, en el sentido de que no son de interés de la estimación. Son muy comunes en los modelos jerárquicos.

Ejemplo 2: GS en modelos jerárquicos II

Modelo jerárquico Binomial-beta

- Entonces se tiene que la densidad conjunta de X y θ se puede escribir como

$$f(X, \theta) = f(X|\theta)f(\theta) = f(\theta|X)f(X) = \binom{n}{x} \theta^x (1 - \theta)^{n-x} \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} \theta^{a-1} (1 - \theta)^{b-1}$$
$$f(\theta|X) = \frac{f(X, \theta)}{f(X)} \propto \binom{n}{x} \theta^{x+a-1} (1 - \theta)^{n-x+b-1}$$

- Entonces $\theta|X \sim \mathcal{Be}(x+a, n-x+b)$ (beta y binomial son familias conjugada). Así que para generar una muestra (X, θ) se puede generar observaciones de las marginales $f(X|\theta)$ y de $f(\theta|X)$.
- Queremos obtener muestras de la conjunta de los datos y el parámetro. Se considera una binomial con $n = 15$, con $a = 1$ y $b = 8$, como ejemplo

Ejemplo 2: GS en modelos jerárquicos III

Modelo jerárquico Binomial-beta

```
nsim <- 5000
n <- 15      #Parámetro dado
a <- 1; b <- 8 #Parámetros de Beta
X <- Theta <- array(0,dim=c(nsim,1))
Theta[1] <- rbeta(1,a,b)
X[1] <- rbinom(1,n,Theta[1])
for(i in 2:nsim){
  X[i] <- rbinom(1,n,Theta[1])
  Theta[i] <- rbeta(1,a+X[i],n-X[i]+b)
}

par(mfrow=c(1,3))
plot(X,Theta, pch = 16, cex = 0.5, ylab = expression(Theta))
hist(X, probability = T, main = "X", ylab = "densidad")
hist(Theta, probability = T, main = expression(Theta),
     xlab = expression(Theta), ylab = "densidad")
```


Modelo jerárquico Binomial-beta



- **Problema:** Considerar el siguiente proceso Poisson con punto de cambio:

$$X_t \sim \begin{cases} \mathcal{P}(\mu t) & 0 < t \leq k \\ \mathcal{P}(\lambda t) & t > k \end{cases}$$

Dada una muestra de n observaciones del proceso anterior, estimar μ , λ y k .

- Este es un proceso muy común y que ha sido ampliamente estudiado. Hay varias opciones de especificación de modelos Bayesianos para resolverlo.
- Una aplicación concreta del problema anterior se relaciona con los datos publicados en el artículo de R.G. Jarret: A note on the intervals between coal-mining disasters. *Biometrika*, **66:191-193**, 1979. Los datos se encuentran en `coal`, en el paquete de R `boot` y corresponden a las fechas de 191 explosiones en minas de carbón que resultaron en más de 10 fatalidades desde marzo 15, 1851 hasta marzo 22, 1962.

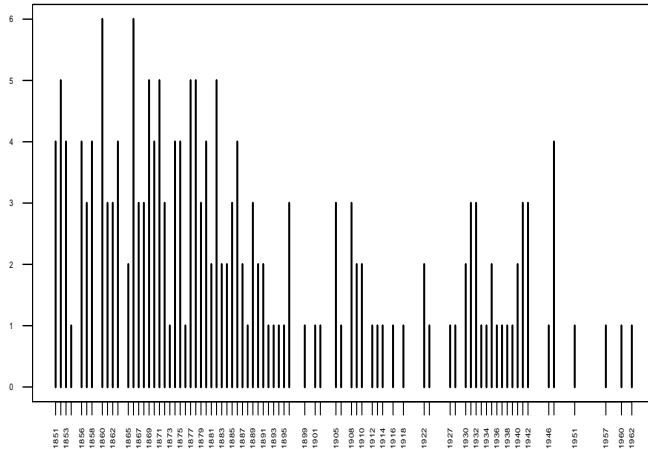
Aplicación: Análisis de punto de cambio II

```
library(boot)
par(mar=c(2,2,3,1))
data(coal)
head(coal) #fecha en numeros consecutivos desde 1851

      date
1 1851.203
2 1851.632
3 1851.969
4 1851.975
5 1852.314
6 1852.347

año <- floor(coal) #tomamos el año del evento
y <- table(año) #número de eventos por año
plot(y,las=2,main="Accidentes en Minas por año 1851-1962",cex.axis=0.4)
```

Accidentes en Minas por año 1851–1962



Aplicación: Análisis de punto de cambio IV

Los datos muestran un cambio en el número promedio de desastres por año alrededor de 1900. Los números anuales de accidentes se crean a continuación:

```
año <- floor(coal[[1]]) #extrae las fechas como vector
y <- tabulate(año) #usa tabulate para obtener los años con frecuencia 0
y <- y[1851:length(y)]
y

[1] 4 5 4 1 0 4 3 4 0 6 3 3 4 0 2 6 3 3 5 4 5 3 1 4 4 1 5 5 3 4 2 5 2 2 3
[36] 4 2 1 3 2 2 1 1 1 1 3 0 0 1 0 1 1 0 0 3 1 0 3 2 2 0 1 1 1 0 1 0 1 0 0
[71] 0 2 1 0 0 0 1 1 0 2 3 3 1 1 2 1 1 1 1 2 3 3 0 0 0 1 4 0 0 0 1 0 0 0 0
[106] 0 1 0 0 1 0 1
```

Modelado:

- Sea Y_i = número de desastres en el año i (1851=1). Entonces, si k es el año punto de cambio,

$$Y_i \sim \mathcal{P}(\mu), \quad i = 1, \dots, k,$$
$$Y_i \sim \mathcal{P}(\lambda), \quad i = k + 1, \dots, n$$

Hay $n = 112$ observaciones terminando en el año 1962. El parámetro a estimar es $\theta = (k, \mu, \lambda)$.

Aplicación: Análisis de punto de cambio V

- Se requiere estimar como distribución objetivo la posterior $\pi(\boldsymbol{\theta}|y) = \pi(k, \mu, \lambda|y)$, y en particular, la distribución posterior de k , $\pi(k|y, \mu, \lambda)$.
- Se puede construir un modelo Bayesiano con las siguientes distribuciones iniciales *independientes*:

$$k \sim U\{1, 2, \dots, n\},$$

$$\mu \sim \mathcal{G}(a_1, b_1),$$

$$\lambda \sim \mathcal{G}(a_2, b_2)$$

donde a_1 , a_2 , b_1 y b_2 son hiperparámetros que se pueden fijar o bien, se pueden considerar a su vez aleatorios. Para este ejercicio se considerarán dados.

- Definimos:
 - $S_k = \sum_{i=1}^k Y_i$ el número de accidentes hasta el periodo de cambio
 - $S_k^c = S_n - S_k$ el número de accidentes posteriores al cambio.

Aplicación: Análisis de punto de cambio VI

Estas variables son sumas de gammas independientes. Para aplicar el GS, se necesita especificar las distribuciones condicionales posteriores. Las densidades condicionales para k , μ , λ , están dadas por:

$$\begin{aligned}\mu|y, k &\sim \mathcal{G}(a_1 + S_k, k + b_1) \\ \lambda|y, k &\sim \mathcal{G}(a_2 + S_k^c, n - k + b_2) \\ k|y, \mu, \lambda &\sim \frac{L(Y|k, \mu, \lambda)}{\sum_{j=1}^n L(Y|j, \mu, \lambda)}\end{aligned}$$

con función de verosimilitud

$$L(Y|k, \mu, \lambda) = e^{k(\lambda - \mu)} \left(\frac{\mu}{\lambda}\right)^{S_k}.$$

Aplicación: Análisis de punto de cambio VII

Por ejemplo, para encontrar la distribución de $\mu|y, k$ usamos la siguiente relación:

$$\begin{aligned}\pi(\mu|y, k) &\propto \pi(y|k, \mu)\pi(k, \mu) = \pi(y|k, \mu)\pi(k)\pi(\mu) \\ &\propto \prod_{i=1}^k e^{-\mu} \mu^{y_i} e^{-b_1 \mu} \mu^{a_1-1} \\ &\propto e^{-k\mu - b_1 \mu} \mu^{S_k + a_1 - 1}\end{aligned}$$

que corresponde al kernel de una distribución $\mathcal{G}(S_k + a_1, k + b_1)$, por lo que

$$\mu|y, k \sim \mathcal{G}(S_k + a_1, k + b_1).$$

Aplicación: Análisis de punto de cambio VIII

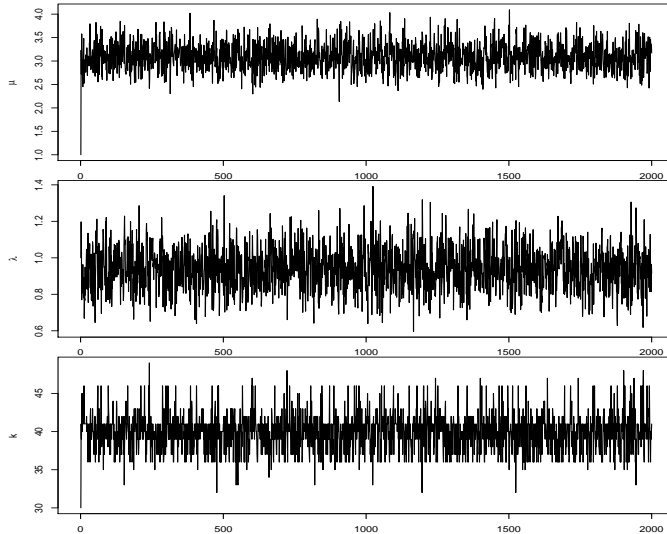
```
# Simulación de los datos para la distribución del punto de cambio.
n <- length(y)           # longitud de los datos
m <- 2000                 # longitud de la cadena
mu <- lambda <- k <- numeric(m) # inicializa mu, lambda y k
L <- numeric(n)
k[1] <- sample(1:n, 1)   # valor inicial del punto de cambio (seleccionado al azar)
mu[1] <- 1
lambda[1] <- 1
b1 <- b2 <- 1
a1 <- a2 <- 2

for (i in 2:m){
  mu[i] <- rgamma(1, shape = a1 + sum(y[1:k[i-1]]), rate = k[i-1] + b1) # genera mu
  lambda[i] <- rgamma(1, shape = a2 + sum(y) - sum(y[1:k[i-1]]), rate = n - k[i-1] + b2) # genera lambda
}

# Construye la función de verosimilitud
for(j in 1:n){
  L[j] <- exp((lambda[i]-mu[i])*j) * (mu[i]/lambda[i])^sum(y[1:j])
}
L <- L/sum(L)
#genera k de la distribución discreta L en 1:n
k[i] <- sample(1:n, prob = L, size = 1)
}
```

```
par(mfrow = c(3,1))
par(mar = c(2,4,0,1))
plot(mu, type = "l", ylab = expression(mu))
plot(lambda, type = "l", ylab = expression(lambda))
plot(k, type = "l", ylab = "k")
```

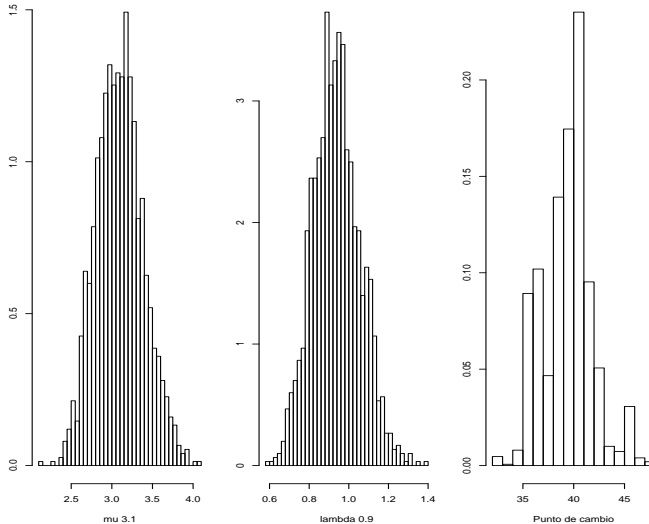
Aplicación: Análisis de punto de cambio IX



Aplicación: Análisis de punto de cambio X

```
# histogramas de las muestras del GS
b <- 500 # burn-in
par(mar=c(4,3,0,1))
par(mfrow=c(1,3))
label1 <- paste("mu", round(mean(mu[b:m]), 1))
label2 <- paste("lambda", round(mean(lambda[b:m]), 1))
hist(mu[b:m], main="", xlab=label1, breaks = 30, prob=TRUE) #mu posterior
hist(lambda[b:m], main="", xlab=label2, breaks = 30, prob=TRUE) #lambda posterior
hist(k[b:m], breaks = min(k[b:m]):max(k[b:m]), prob=TRUE, main="", xlab = "Punto de cambio")
```

Aplicación: Análisis de punto de cambio XI



Entonces el punto de cambio está alrededor de $\hat{k} = 40$, que corresponde al año $1851 + 40 = 1891$. De 1851 a 1890 la media Poisson es alrededor de $\hat{\mu} \approx 3.1$ y del año 1891 hacia adelante la media es $\hat{\lambda} \approx 0.93$

GS Software

- [BUGS \(Bayesian Analysis using Gibbs Sampler\)](#) es un programa que permite realizar el análisis Bayesiano de modelos estadísticos complicados, utilizando métodos MCMC y extraer muestras de distribuciones posteriores. Fue desarrollado del 89 al 97 por un equipo en Cambridge dirigido por David Spiegelhalter.
- De este modo, podemos enfocarnos en el modelado estadístico, que es el principal interés, y dejar los cálculos a la computadora. La sintaxis de `*BUGS` es similar a la de R, pero hay diferencias que hay que notar.
- BUGS tiene dos sabores: [WinBUGS](#), versión para Windows con interfaz gráfica y [OpenBUGS](#), que es la versión de consola, abierta y extendida a más plataformas. Ahora también tiene su versión GUI.
- WinBUGS ya no se actualiza y se ha sustituido principalmente por OpenBUGS. BRugs y R2OpenBUGS son paquetes interfaz entre OpenBUGS y R.
- Una buena manera de aprender lo básico de WinBUGS es ver: [WinBUGS - The Movie](#)
- [JAGS \(Just Another Gibbs Sampler\)](#) es otro proyecto independiente de BUGS para hacer análisis de modelos jerárquicos bayesianos usando MCMC. Desarrollado por [Martyn Plummer](#)

- [Stan](#) Stan es un lenguaje y librerías para obtener inferencia Bayesiana utilizando muestreo MCMC, utilizando como interfase a R, pero también puede ser Python, MATLAB, JULIA y Stata. A la fecha es uno de las implementaciones con más interés en desarrollo. La documentación se puede encontrar [aquí](#).

Aún cuando hay muchas implementaciones, el lenguaje básico es muy similar en todas ellas, y es muy parecida a la de BUGS, por lo que nos concentraremos en esa versión.

- Para llevar a cabo un análisis en BUGS, se requiere realizar el siguiente procedimiento:
 - 1 Definir el modelo
 - 2 Incluir los datos observados
 - 3 Dar valores iniciales para los parámetros.
 - 4 Estimar el modelo
 - 5 Realizar evaluación de diagnósticos.
 - 6 Analizar los resultados
- A continuación seguiremos el procedimiento en WinBUGS a través de un ejemplo sencillo, y lo haremos también a través de la interfase a R utilizando BRugs.

Ejemplo 3: Modelo normal conjugado I

- Paso 1: Consideremos un modelo normal conjugado:

$$y|\theta \sim \mathcal{N}(\theta, \sigma^2)$$

donde σ^2 es conocido, y la prior de la media θ también es normal,

$$\theta|\eta \sim \mathcal{N}(m, \tau^2)$$

donde m y τ son hiperparámetros conocidos, donde $\eta = (m, \tau)$; entonces la distribución posterior, $\theta|y$ también es normal:

$$\begin{aligned}\theta|y &\sim \mathcal{N}\left(\frac{\sigma^2 m + \tau y}{\sigma^2 + \tau^2}, \frac{\sigma^2 \tau^2}{\sigma^2 + \tau^2}\right) \\ &= \mathcal{N}(Bm + (1 - B)y, B\tau^2)\end{aligned}$$

donde $B = \frac{\sigma^2}{\sigma^2 + \tau^2} \in (0, 1)$, se puede interpretar como un *factor de compresión* de la media.

Ejemplo 3: Modelo normal conjugado II

- En estadística Bayesiana, es común pensar en términos del concepto de *precisión* más que de varianza.

Precisión

La precisión es el recíproco de la varianza: si σ^2 es la varianza, la precisión es $1/\sigma^2$.

- Tomando una muestra aleatoria y_1, \dots, y_n , la información de toda la muestra se puede combinar en la *estadística suficiente* \bar{y} , y entonces la posterior en términos de una muestra queda como:

$$\begin{aligned}\theta|\bar{y} &\sim \mathcal{N}\left(\frac{\sigma^2 m + \tau^2 \bar{y}}{\sigma^2/n + \tau^2}, \frac{\frac{\sigma^2}{n} \tau^2}{\sigma^2/n + \tau^2}\right) \\ &= \mathcal{N}\left(\frac{\sigma^2 m + n\tau^2 \bar{y}}{\sigma^2 + n\tau^2}, \frac{\sigma^2 \tau^2}{\sigma^2 + n\tau^2}\right)\end{aligned}$$

Ejemplo 3: Modelo normal conjugado III

- Pasos 2 y 3: Por ejemplo, si $m = 2$, $\tau = \sigma = 1$ y $\bar{y} = 6$, podemos obtener la distribución posterior para diferentes tamaños n de la muestra.

Aunque este modelo está resuelto de manera analítica, se puede usar como referencia para aprender WinBUGS.

Para ejecutar este modelo desde R usando [BRugs](#), seguimos el mismo proceso.

- 1 Paso 4: Escribimos el modelo a un archivo, y se “checa” el modelo:

```
library(BRugs) #Paquete que establece la conexión de OpenBUGS y R
Welcome to BRugs connected to OpenBUGS version 3.2.3
setwd("~/Dropbox/Academia/ITAM/SimS18-II/bugs/modelo1/")
Modelo <- "
model{
  prec.ybar <- n/sigma2 #Los modelos normales en BUGS usan precision, no varianza
  prec.theta <- 1/tau2
  ybar ~ dnorm(theta,prec.ybar)
  theta ~ dnorm(mu,prec.theta)
}"
writeLines(Modelo, con = "Modelo.txt")
modelCheck("Modelo.txt")

model is syntactically correct
```

Ejemplo 3: Modelo normal conjugado IV

- 2 Cargamos los datos y los valores iniciales a un archivo. Cargamos los datos del modelo. Aquí suponemos que $n = 10$.

```
datos <- "list(ybar=6, mu=2, sigma2=1, tau2=1, n=10)"
writeLines(datos, con = "datos.txt")

vinit <- "list(theta=0)"
writeLines(vinit, con = "vinit.txt")

modelData("datos.txt")

data loaded
```

- 3 A continuación se compila el modelo y se inicializa la cadena.

```
modelCompile() #compilación del modelo
model compiled

modelInits("vinit.txt") #Se da un valor inicial para el modelo o se puede usar modelGenInits() para valores generados
Initializing chain 1:
model is initialized
```

Ejemplo 3: Modelo normal conjugado V

- 4 Ahora ejecutamos el modelo. Por ejemplo, consideramos un burn-in de 1,000 observaciones y generamos 10,000 datos adicionales. Necesitamos monitorear a θ :

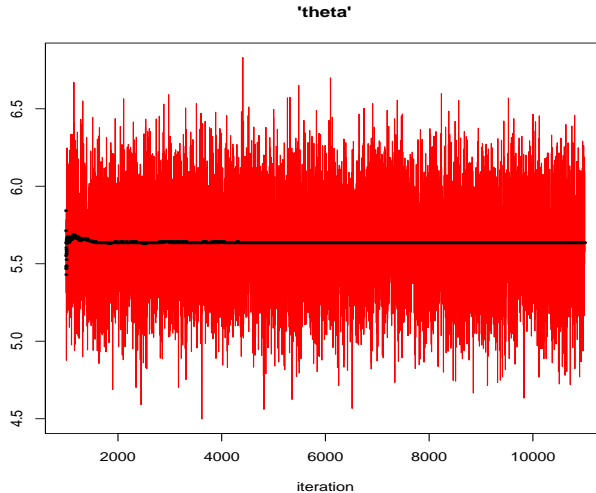
```
modelUpdate(1000) #burn-in
1000 updates took 0 s
samplesSet("theta") #Define el monitor de la variable
monitor set for variable 'theta'
modelUpdate(10000) #Generamos n iteraciones
10000 updates took 0 s
samplesStats("theta") #podemos usar "*" o un vector con los nombres de las variables a monitorear entre comillas
      mean      sd MC_error val2.5pc median val97.5pc start sample
theta 5.635 0.3028 0.003174    5.046  5.638      6.228  1001  10000
```

`samplesStats` devuelve los valores de la muestra obtenida

Paso 5: Podemos ahora generar las gráficas del modelo

```
a <- samplesHistory("a",mfrow=c(1,1),ask = F)
points(1001:11000,cumsum(a$theta)/(1:length(a$theta)),pch=16,cex=0.5)
```

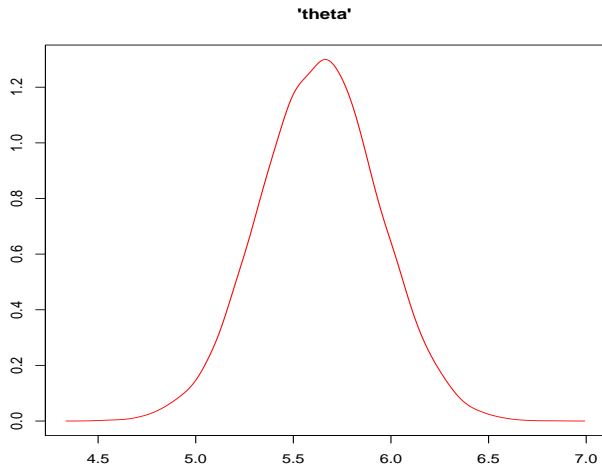
Ejemplo 3: Modelo normal conjugado VI



Ejemplo 3: Modelo normal conjugado VII

```
samplesDensity("theta",mfrow=c(1,1))
```


Ejemplo 3: Modelo normal conjugado VIII



Ejemplo 3 con Modelo normal conjugado con R2openBUGS I

Ejecutamos el mismo ejemplo previo usando el paquete R2openBUGS. El modelo se escribe con la misma sintaxis que antes y está en el archivo `Modelo.txt`. Este programa da una salida ligeramente diferente a la de OpenBUGS con información adicional.

Ejemplo 3 con Modelo normal conjugado con R2OpenBUGS II

```
library(R2OpenBUGS)
setwd("~/Dropbox/Academia/ITAM/SimS18-II/bugs/modelo1/") #define el directorio de trabajo.
datos <- list(ybar=6, mu=2, sigma2=1, tau2=1, n=10) #define los valores de los datos

#también se requiere especificar valores iniciales
inits <- function(){list(theta = 0)} #se requiere función para tener inicializadas múltiples cadenas
mod.sim <- bugs(datos, inits, model.file = "Modelo.txt",
               parameters = c("theta"), n.chains = 20,
               n.burnin = 1000, n.iter = 5000)

print(mod.sim)
```

Inference for Bugs model at "Modelo.txt",
Current: 20 chains, each with 5000 iterations (first 1000 discarded)
Cumulative: n.sims = 80000 iterations saved

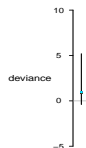
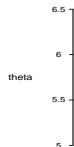
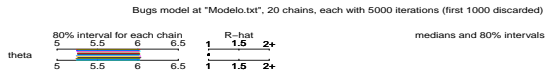
	mean	sd	2.5%	25%	50%	75%	97.5%	Rhat	n.eff
theta	5.6	0.3	5.0	5.4	5.6	5.8	6.2	1	80000
deviance	1.8	2.6	-0.5	-0.1	0.9	2.8	8.7	1	80000

For each parameter, n.eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor (at convergence, Rhat=1).

DIC info (using the rule, $pD = Dbar - Dhat$)
 $pD = 0.9$ and $DIC = 2.7$
DIC is an estimate of expected predictive error (lower deviance is better).

```
plot(mod.sim)
```

Ejemplo 3 con Modelo normal conjugado con R2openBUGS III



Explicación de los elementos de la salida de OpenBUGS I

- La interpretación de esta salida es muy similar a la que se obtendría en Stan también.
- En la salida de los resultados del modelo, se pueden ver los siguientes elementos para cada parámetro estimado:
 - `mean` es la media posterior estimada, calculada como el promedio de las salidas de la cadena.
 - `sd` es la desviación estándar de las observaciones, que conforme el tamaño de muestra tiende a infinito, se acerca a la desviación estándar posterior del parámetro.
 - 2.5 %, 25 %, 50 %, 75 %, 97.5 % son cuantiles de la distribución posterior
 - `Rhat` corresponde al factor de reducción potencial de escala \hat{R} que se mencionó antes, que vale $\hat{R} = 1$ cuando todas las cadenas se han mezclado
 - `n.eff` es el tamaño de muestra efectivo, que tiene una fórmula dada por:

$$\hat{n}_{eff} = \frac{mn}{1 + 2 \sum_{i=1}^T \hat{\rho}_t}$$

ρ_t es la autocorrelación de la sucesión $\{X_n\}$ en el rezago t , n es el número de simulaciones de cada cadena y m es el número de cadenas simuladas, y T es el primer entero impar para el cual $\hat{\rho}_{T+1} + \hat{\rho}_{T+2} < 0$. (ver detalles en Gelman, et.al BDA3, sección 11.5) Basta tomar como referencia que $\hat{n}_{eff} \geq 10m$ es un síntoma de que la cadena se ha mezclado adecuadamente.

- En la práctica, la selección estadística apropiada de un mejor modelo en una colección de modelos jerárquicos es problemática, debido a la ambigüedad del tamaño de esos modelos.
- Spiegelhalter et al. (2002) sugiere una generalización del Criterio de Información de Akaike (AIC) que se basa en la distribución posterior de la estadística *devianza*:

$$D(\theta) = -2 \log f(y|\theta) + 2 \log h(y)$$

donde $f(y|\theta)$ es la verosimilitud de los datos observados, dado el parámetro θ y $h(y)$ es alguna función estandarizada que sólo depende de los datos (y que no tiene importancia en la selección del modelo).

- En este enfoque, el ajuste de un modelo queda resumido por la esperanza posterior de la devianza, $\bar{D} = E_{\theta|y}[D]$, mientras que la complejidad del modelo se captura en el número efectivo de parámetros p_D que hay en el modelo:

$$p_D = E_{\theta|y}[D] - D(E_{\theta|y}[\theta]) = \bar{D} - D(\bar{\theta})$$

- El *Criterio de Información de la Devianza* (DIC) se define como

$$DIC = \bar{D} + p_D = 2\bar{D} - D(\bar{\theta})$$

- Valores pequeños de DIC indican un modelo que ajusta mejor.
- Comparando modelos, los que tienen las máximas diferencias son los que tienen mayor cambio y mejoran más.

Ejemplo: Modelo normal conjugado y coda. I

- coda significa [Convergence and diagnostics analysis for MCMC](#). El paquete [coda](#), permite hacer análisis de salida y diagnósticos para MCMC. coda trata de responder a la pregunta: ¿ha tenido el sampler suficiente adaptación (burn-in) para justificar que la muestra proviene de la posterior de interés?
- Un paquete similar es boa que sirve para evaluar la convergencia de cadenas MCMC. boa surge de reescribir de las funciones y la interfase de CODA, que es la versión original del paquete de R coda. El orden es:

CODA \rightarrow boa \rightarrow coda

- Usualmente tenemos que analizar las muestras *después* del periodo burn-in para detectar algún tipo de anomalía. El paquete coda ofrece algunas funciones de diagnóstico para facilitar el análisis.

Ejemplo: Modelo normal conjugado y coda. II

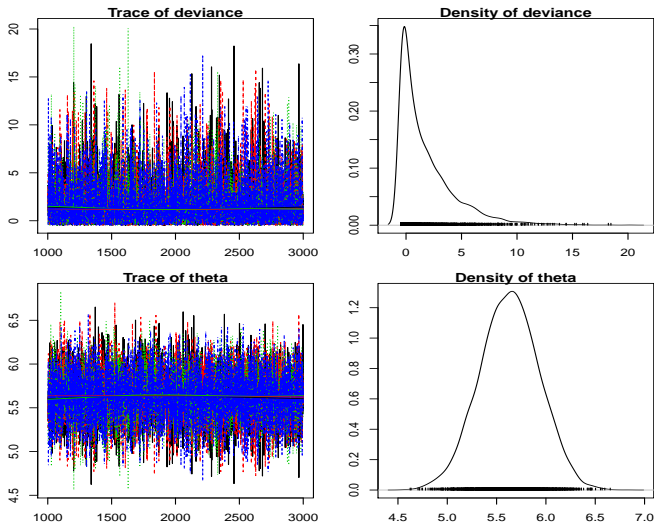
- Podemos obtener la traza y las densidades de las variables de interés del modelo. Estas ya las obteníamos desde antes.

```
par(mar=c(3,3,1,1))
library(coda)
#agrega la opción {\tt codaPkg = T} para poder leer la salida a coda
mod.sim2 <- bugs(datos, inits, model.file = "Modelo.txt",
                parameters = c("theta"), n.chains = 4,
                n.burnin = 1000, n.iter = 3000, codaPkg = T)

plot(read.bugs(mod.sim2))
```

Abstracting deviance ... 2000 valid values
Abstracting theta ... 2000 valid values
Abstracting deviance ... 2000 valid values
Abstracting theta ... 2000 valid values
Abstracting deviance ... 2000 valid values
Abstracting theta ... 2000 valid values
Abstracting deviance ... 2000 valid values
Abstracting theta ... 2000 valid values

Ejemplo: Modelo normal conjugado y coda. III



Ejemplo: Modelo normal conjugado y coda. IV

- Otra herramienta de diagnóstico que ya mencionamos antes son los gráficos de Gelman, que grafican el valor de \hat{R} .
- La gráfica de Gelman mide si hay una diferencia significativa entre la varianza al interior de varias cadenas y la varianza entre las cadenas.
- `gelman.diag` devuelve el factor de reducción para cada parámetro. Un factor de 1 significa que la intravarianza y la intervianza son iguales. Valores grandes denotan que hay diferencias notables entre las cadenas.

Ejemplo: Modelo normal conjugado y coda. V

```
gelman.diag(read.bugs(mod.sim2))
```

```
Abstracting deviance ... 2000 valid values
Abstracting theta ... 2000 valid values
Abstracting deviance ... 2000 valid values
Abstracting theta ... 2000 valid values
Abstracting deviance ... 2000 valid values
Abstracting theta ... 2000 valid values
Abstracting deviance ... 2000 valid values
Abstracting theta ... 2000 valid values
Potential scale reduction factors:
```

	Point est.	Upper C.I.
deviance	1	1
theta	1	1

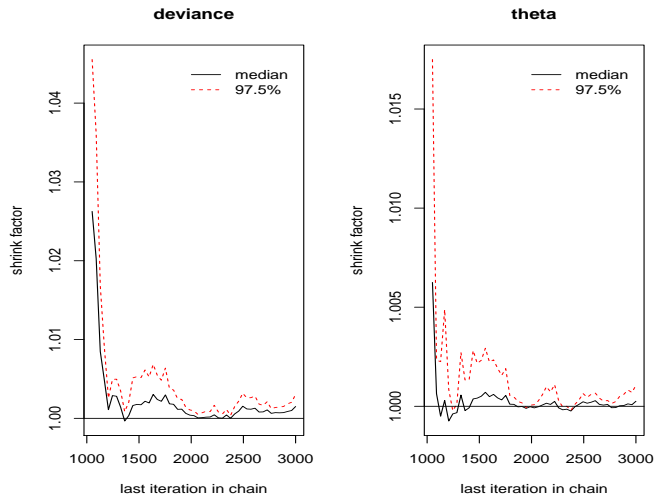
```
Multivariate psrf
```

```
1
```

```
gelman.plot(read.bugs(mod.sim2))
```

```
Abstracting deviance ... 2000 valid values
Abstracting theta ... 2000 valid values
Abstracting deviance ... 2000 valid values
Abstracting theta ... 2000 valid values
Abstracting deviance ... 2000 valid values
Abstracting theta ... 2000 valid values
Abstracting deviance ... 2000 valid values
Abstracting theta ... 2000 valid values
```

Ejemplo: Modelo normal conjugado y coda. VI



Ejemplo: Modelo normal conjugado y coda. VII

De acuerdo al profesor Charles Geyer, de la Universidad de Minnesota, el periodo de burn-in es completamente **innecesario**

Ejemplo 3: Modelo normal conjugado con JAGS y rjags I

Para ejecutar el programa ahora usamos JAGS El archivo con el modelo sigue teniendo la misma especificación y está en `Modelo.txt`. Obviamente primero se requiere tener instalado JAGS en la computadora.

```
library(rjags)

Linked to JAGS 4.3.0
Loaded modules: basemod,bugs

setwd("~/Dropbox/Academia/ITAM/SimS18-II/bugs/modelo1/") #define el directorio de trabajo.
jags <- jags.model("Modelo.txt",
  data = list(ybar=6, mu=2, sigma2=1, tau2=1, n=10), #define los valores de los datos
  n.chains = 20,
  n.adapt = 1000) # burn-in

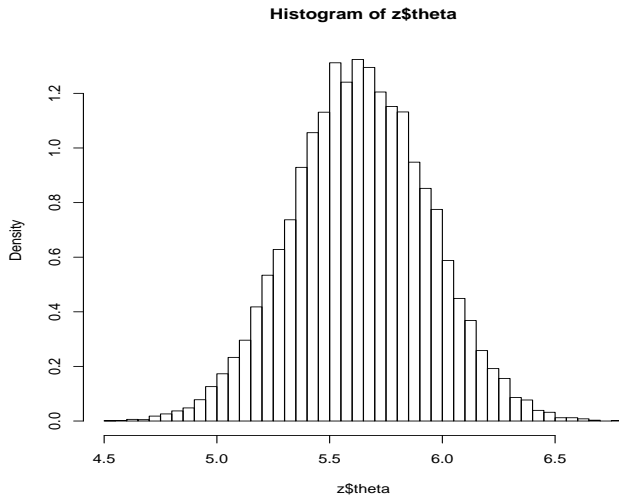
Compiling model graph
  Resolving undeclared variables
  Allocating nodes
Graph information:
  Observed stochastic nodes: 1
  Unobserved stochastic nodes: 1
  Total graph size: 9

Initializing model

update(jags,1000)

z <- jags.samples(jags,c("theta"),1000)
hist(z$theta,breaks = 50,prob=T)
```

Ejemplo 3: Modelo normal conjugado con JAGS y rjags II



Ejemplo 3: Modelo normal conjugado con JAGS y rjags III

Información adicional y ejemplos se pueden encontrar [en este tutorial](#).

Ejemplo 4: Simulación de transformación de variable aleatoria I

Problema: Si tenemos $Z \sim N(0, 1)$ y transformamos a $Y = (2Z + 1)^3$. Queremos obtener la distribución de Y , su media y $P(Y > k)$ para alguna k , por ejemplo $k = 10$. El problema es difícil de resolver de manera analítica pero es fácil de simular.

Ejemplo 4: Simulación de transformación de variable aleatoria II

```
Modelo2 <- "  
model {  
  Z ~ dnorm(0,1) #Hay que considerar que en BUGS se lee la precisión, no la varianza  
  Y <- pow(2*Z+1,3)  
  P10 <- step(Y-10) #step es la indicadora de la condición > 0 Suponemos k=10  
}"  
  
writeLines(Modelo2,con="Modelo2.txt")  
modelCheck("Modelo2.txt")  
  
model is syntactically correct  
  
modelCompile()  
  
model compiled  
  
modelGenInits() #inicializa al azar  
  
initial values generated, model initialized  
  
modelUpdate(10000) #burn-in  
  
10000 updates took 0 s  
  
samplesSet(c("Y","P10"))  
  
monitor set for variable 'Y'  
monitor set for variable 'P10'  
  
modelUpdate(50000)  
  
50000 updates took 0 s
```

Simplificación de proceso I

Como pueden ver, varios pasos se repiten con frecuencia. Entonces, podemos tratar de escribir una función que haga todo el proceso sin tener que reescribirlo todo el tiempo.

```
run.model <- function(modelo, muestras, datos = list(), longcadena = 10000, burnin = 0.10,
vinit, nchains=1,thin=1) {
  writeLines(modelo, con="modelg.txt")
  modelCheck("modelg.txt") #Envía el modelo a BUGS, para verificar sintaxis
  if(length(datos)>0)      #Si hay datos disponibles,
  modelData(bugsData(datos)) #BRugs los pone en un archivo y los envía a BUGS
  modelCompile(nchains)    #BRugs compila el modelo

  if(missing(vinit)) {
    modelGenInits()         #Inicializa la cadena al azar si no hay valores iniciales
  } else {
    for(chain in 1:nchains) modelInits(bugsInits(vinit))
  }

  modelUpdate(longcadena*burnin) #porcentaje de las simulaciones a descartarse
  samplesSet(muestras)
  samplesSetThin(thin)
  modelUpdate(longcadena)
}
```

Entonces el ejercicio previo lo pudimos haber ejecutado de la siguiente manera:

Simplificación de proceso II

```
run.model(Modelo2, muestras=c("Y","P10"),longcadena = 50000)
```

```
model is syntactically correct  
model compiled  
initial values generated, model initialized  
5000 updates took 0 s  
monitor set for variable 'Y'  
monitor set for variable 'P10'  
50000 updates took 0 s
```

Ejemplo 5: Estimación de número de reparaciones I

Problema: Tenemos 100 computadoras que requieren reparación. El costo unitario de reparación depende de las piezas que tenga descompuesta la computadora, y usualmente el costo es una variable aleatoria que se puede modelar como una gamma con media 100 y desviación estándar 50. Si tenemos un presupuesto de 10,000 para reparaciones, ¿cuántas reparaciones en promedio podemos hacer?

Recuerden que la distribución $\mathcal{G}(a, b)$ tiene media $\frac{a}{b}$ y varianza $\frac{a}{b^2}$. Entonces la costo en el ejercicio tiene distribución $\mathcal{G}(4, 0.04)$.

Para simular el ejercicio, generamos 100 datos de costos con la distribución dada. Entonces obtenemos una muestra Y_1, \dots, Y_{100} donde Y_i es el costo de reparar la computadora i . El problema consiste en estimar M tal que $\sum_{i=1}^M Y_i \leq 10,000$. Como podemos ver M es aleatorio completamente.

Ejemplo 5: Estimación de número de reparaciones II

```
modelo3 <- "  
model {  
  for(i in 1:100) {Y[i] ~ dgamma(4,0.04)}  
  sumacosto[1] <- Y[1]  
  for(i in 2:100) {sumacosto[i] <- sumacosto[i-1] + Y[i]}  
  for(i in 1:100) {cum.step[i] <- i * step(10000 - sumacosto[i]) } #1,2,...,M,0,...  
  M <- ranked(cum.step[],100)  
}"
```

```
run.model(modelo3, muestras=c("M","sumacosto[100]"),longcadena = 50000)
```

```
model is syntactically correct  
model compiled  
initial values generated, model initialized  
5000 updates took 0 s  
monitor set for variable 'M'  
monitor set for variable 'sumacosto[100]'  
50000 updates took 7 s
```

Para ver los resultados

Ejemplo 5: Estimación de número de reparaciones III

```
samplesStats(c("M", "sumacosto[100]"))
```

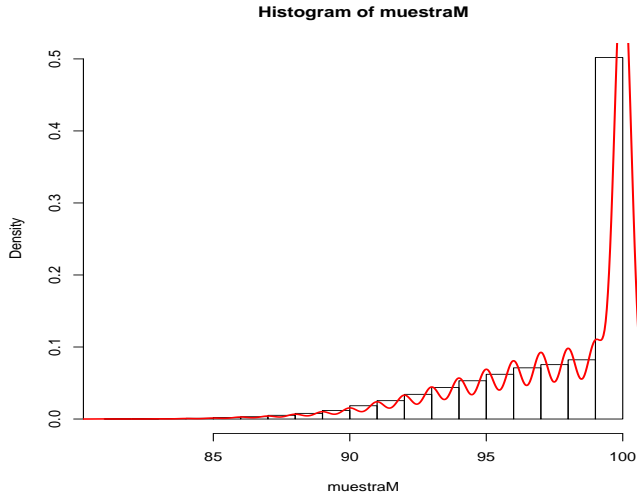
	mean	sd	MC_error	val2.5pc	median	val97.5pc	start
M	97.8	3.033	0.01244	90	100	100	5001
sumacosto[100]	10000.0	498.700	2.10000	9055	9998	11000	5001

sample

M	50000
sumacosto[100]	50000

```
muestraM <- samplesSample("M")  
hist(muestraM, probability = T)  
lines(density(muestraM), col="red", lwd=2)
```


Ejemplo 5: Estimación de número de reparaciones IV



Ejemplo 6: Modelo de Regresión Lineal I

En el siguiente modelo será $\tau = \frac{1}{\sigma^2}$. Se supone que la precisión tiene una distribución inicial de la forma $\text{gamma}(\epsilon, \epsilon)$, en la parametrización de la Gamma en BUGS, tiene media $\frac{\epsilon}{\epsilon} = 1$ y varianza $\frac{\epsilon}{\epsilon^2} = \frac{1}{\epsilon} = 10$.

Ejemplo 6: Modelo de Regresión Lineal II

```
modelo4 <- "  
model {  
  for (i in 1:n) {  
    logx[i] <- log(x[i])  
    y[i] ~ dnorm(mu[i], tau)  
    mu[i] <- beta0 + beta1*logx[i]  
  }  
  
  # priors para los parámetros de regresión  
  beta0 ~ dnorm(0,100)  
  beta1 ~ dnorm(0,100)  
  
  # prior para la precisión del parámetro  
  tau ~ dgamma(0.1,0.1)  
  sigma <- 1/sqrt(tau)  
}"  
  
run.model(modelo4, muestras=c("beta0","beta1","sigma"),  
          datos = list(x = c(1.0, 1.5, 1.5, 1.5, 2.5, 4.0, 5.0, 5.0, 7.0,  
                             8.0, 8.5, 9.0, 9.5, 9.5, 10.0, 12.0, 12.0, 13.0,  
                             13.0, 14.5, 15.5, 15.5, 16.5, 17.0, 22.5, 29.0,31.5),  
                        y = c(1.80, 1.85, 1.87, 1.77, 2.02, 2.27, 2.15, 2.26, 2.47,  
                             2.19, 2.26, 2.40, 2.39, 2.41, 2.50, 2.32, 2.32, 2.43,  
                             2.47, 2.56, 2.65, 2.47, 2.64, 2.56, 2.70, 2.72, 2.57),  
                        n = 27),  
          longcadena = 30000, burnin = 0.20)  
  
model is syntactically correct  
data loaded  
model compiled  
initial values generated, model initialized  
6000 updates took 0 s  
monitor set for variable 'beta0'  
monitor set for variable 'beta1'  
monitor set for variable 'sigma'
```

Modelo de Regresión Lineal I

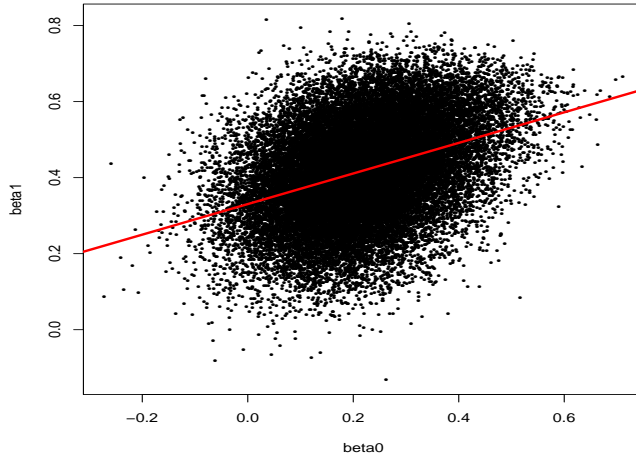
Después de obtener las iteraciones analizamos los resultados. Noten en la gráfica conjunta de (β_0, β_1) la fuerte correlación que hay entre los dos parámetros. La correlación se puede eliminar si la variable x se centra alrededor de su propia media.

```
samplesStats("*")
```

	mean	sd	MC_error	val2.5pc	median	val97.5pc	start	sample
beta0	0.2179	0.1191	0.001299	-0.009314	0.2158	0.4592	6001	30000
beta1	0.4181	0.1289	0.001911	0.161000	0.4209	0.6571	6001	30000
sigma	1.3090	0.3666	0.005506	0.689300	1.2790	2.1100	6001	30000

```
beta0 <- samplesSample("beta0")
beta1 <- samplesSample("beta1")
sigma <- samplesSample("sigma")
plot(beta0,beta1,pch=16,cex=0.5)
abline(reg=lm(beta1~beta0),col="red",lwd=3)
```

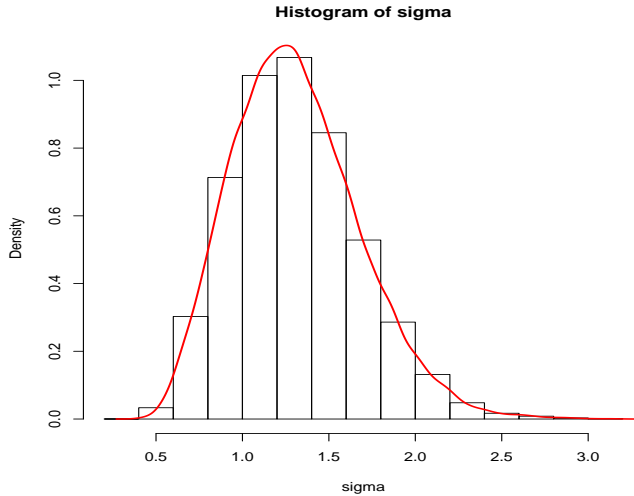
Modelo de Regresión Lineal II



Modelo de Regresión Lineal III

```
hist(sigma, probability = T)  
lines(density(sigma), col="red", lwd=2)
```

Modelo de Regresión Lineal IV



Modelo de Regresión Lineal V

Versión con OpenBUGS para incluir análisis de coda

```
#previamente cargado coda y R2OpenBUGS
writeLines(modelo4, con = "Modelo4.txt")

inits <- function(){list(beta0=1,beta1=1,tau=1)}

datos = list(x = c(1.0, 1.5, 1.5, 1.5, 2.5, 4.0, 5.0, 5.0, 7.0, 8.0, 8.5, 9.0, 9.5, 9.5,
                  10.0, 12.0, 12.0, 13.0, 13.0, 14.5, 15.5, 15.5, 16.5, 17.0, 22.5,
                  29.0,31.5),
            y = c(1.80, 1.85, 1.87, 1.77, 2.02, 2.27, 2.15, 2.26, 2.47, 2.19,
                  2.26, 2.40, 2.39, 2.41, 2.50, 2.32, 2.32, 2.43, 2.47, 2.56, 2.65,
                  2.47, 2.64, 2.56, 2.70, 2.72, 2.57),
            n = 27)

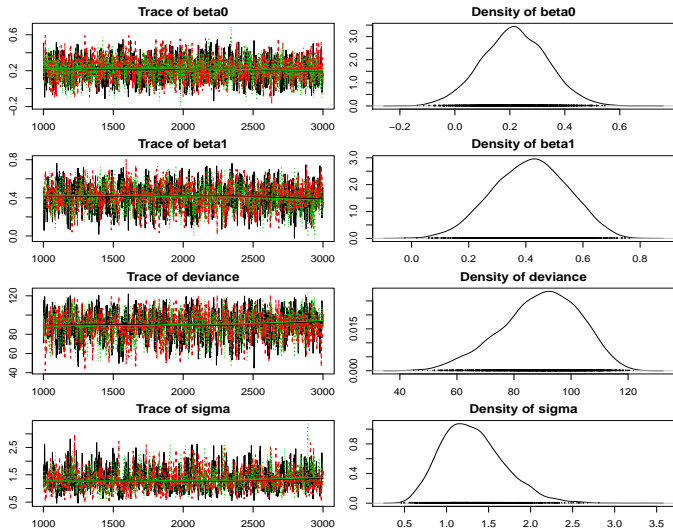
mod.sim1 <- bugs(datos,inits,model.file="Modelo4.txt",
               parameters=c("beta0","beta1","sigma"),
               n.burnin = 1000, n.iter = 3000, codaPkg = T)
```


Modelo de Regresión Lineal VI

```
par(mar=c(2,2,2,1))  
plot(read.bugs(mod.sim1))
```

```
Abstracting beta0 ... 2000 valid values  
Abstracting beta1 ... 2000 valid values  
Abstracting deviance ... 2000 valid values  
Abstracting sigma ... 2000 valid values  
Abstracting beta0 ... 2000 valid values  
Abstracting beta1 ... 2000 valid values  
Abstracting deviance ... 2000 valid values  
Abstracting sigma ... 2000 valid values  
Abstracting beta0 ... 2000 valid values  
Abstracting beta1 ... 2000 valid values  
Abstracting deviance ... 2000 valid values  
Abstracting sigma ... 2000 valid values
```

Modelo de Regresión Lineal VII



Ejemplo 6: Modelo de Regresión Lineal I

```
gelman.diag(read.bugs(mod.sim1))
```

```
Abstracting beta0 ... 2000 valid values
Abstracting beta1 ... 2000 valid values
Abstracting deviance ... 2000 valid values
Abstracting sigma ... 2000 valid values
Abstracting beta0 ... 2000 valid values
Abstracting beta1 ... 2000 valid values
Abstracting deviance ... 2000 valid values
Abstracting sigma ... 2000 valid values
Abstracting beta0 ... 2000 valid values
Abstracting beta1 ... 2000 valid values
Abstracting deviance ... 2000 valid values
Abstracting sigma ... 2000 valid values
Potential scale reduction factors:
```

	Point est.	Upper C.I.
beta0	1	1.00
beta1	1	1.02
deviance	1	1.01
sigma	1	1.01

Multivariate psrf

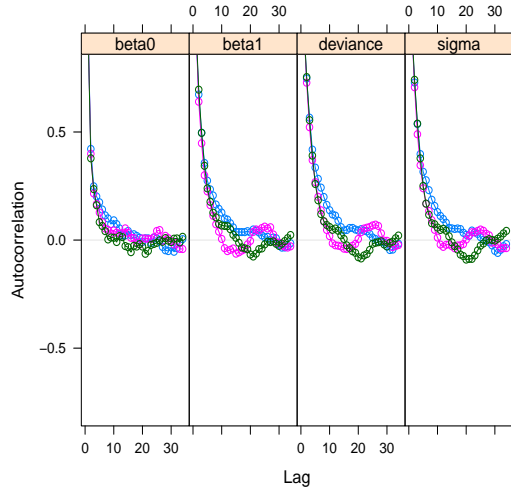
1

Ejemplo 6: Modelo de Regresión Lineal II

```
acfplot(read.bugs(mod.sim1))
```

```
Abstracting beta0 ... 2000 valid values  
Abstracting beta1 ... 2000 valid values  
Abstracting deviance ... 2000 valid values  
Abstracting sigma ... 2000 valid values  
Abstracting beta0 ... 2000 valid values  
Abstracting beta1 ... 2000 valid values  
Abstracting deviance ... 2000 valid values  
Abstracting sigma ... 2000 valid values  
Abstracting beta0 ... 2000 valid values  
Abstracting beta1 ... 2000 valid values  
Abstracting deviance ... 2000 valid values  
Abstracting sigma ... 2000 valid values
```

Ejemplo 6: Modelo de Regresión Lineal III



Ejemplo 6: Modelo de Regresión Lineal IV

```
par(mar = c(3,3,1,1))  
gelman.plot(read.bugs(mod.sim1))
```

```
Abstracting beta0 ... 2000 valid values  
Abstracting beta1 ... 2000 valid values  
Abstracting deviance ... 2000 valid values  
Abstracting sigma ... 2000 valid values  
Abstracting beta0 ... 2000 valid values  
Abstracting beta1 ... 2000 valid values  
Abstracting deviance ... 2000 valid values  
Abstracting sigma ... 2000 valid values  
Abstracting beta0 ... 2000 valid values  
Abstracting beta1 ... 2000 valid values  
Abstracting deviance ... 2000 valid values  
Abstracting sigma ... 2000 valid values
```

Ejemplo 6: Modelo de Regresión Lineal V

