Exercici Model Lineal Mixte

Estadística Bayesiana

Guillem Miralles

19/11/2020

INTRODUCCION AL EJERCICIO:

El por qué de los Modelos Lineales Mixtos (LMM):

Los datos biológicos suelen ser complejos y confusos. Podemos tener diferentes factores de agrupación como poblaciones, especies, sitios donde recopilamos los datos, etc... Los tamaños de las muestras también suelen ser complejos si ya que normalmente se trata de ajustar modelos complicados con muchos parámetros. Además de eso, es habitual que las variables recogidas no sean independientes.

Es por eso que se desarrollaron modelos mixtos, para poder trabajar con datos tan desordenados y para permitirnos usar todos ellos, incluso con tamaños de muestra bajos, datos estructurados y muchas covariables para ajustar.

Ejercicio:

Tenemos un conjunto de datos sobre una recopilación de información de la serpiente Nauyaca Real con las siguientes variables:

```
- TestScore: Puntuación de la serpiente en una pruba para medir su inteligencia. Variable numérica
- BodyLenght: Longitud de la serpiente (en cm). Variable numérica
- mountainRange: Cadena montañosa dónde se han recopilado los datos:
        - Bavarian (1)
        - Ligurian (2)
        - Emmental (3)
        - Central (4)
        - Maritime (5)
        - Southern (6)
        - Julian (7)
        - Sarntal (8)
```

Lo que queremos saber es cómo afecta la longitud del cuerpo de la serpiente (BodyLenght) en función de su inteligencia (TestScore). Entonces, tenemos que la variable que queremos explicar (variable respuesta) es la inteligencia, y la variable explicativa es la longitud del cuerpo. En el modelo lineal mixto, entra en juego una variable de agrupamiento, esta será MountainRange (sitio donde se han recopilado los datos).

```
#Lectura de los datos

library(readr)
data <- read_csv("Serps.csv", col_types = cols(X1 = col_skip(),X = col_skip(),site = col_skip()))
```

```
## Warning: Missing column names filled in: 'X1' [1]
```

```
serps <- as.data.frame(data)

head(serps)
```

```
##   testScore bodyLength mountainRange
## 1 16.147309   165.5485      Bavarian
## 2 33.886183   167.5593      Bavarian
## 3  6.038333   165.8830      Bavarian
## 4 18.838821   167.6855      Bavarian
## 5 33.862328   169.9597      Bavarian
## 6 47.043246   168.6887      Bavarian
```

Escalar y centrar los predictores

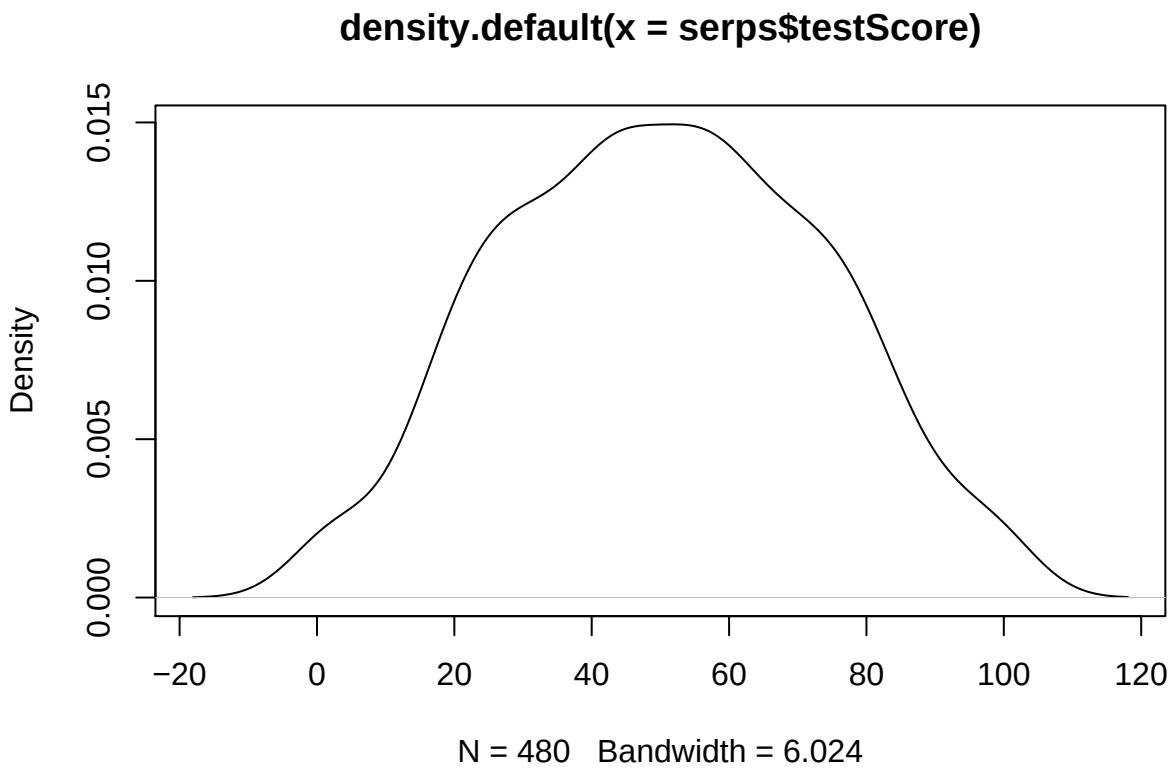
siempre se deben escalar y centrar todos los predictores continuos, en este caso la longitud del cuerpo, antes del análisis (es decir, restar la media y dividir por la desviación estándar, transformándolos así en valores con una media de 0 y una sd de 1) para evitar problemas con ajuste del modelo.

```
serps$bodyLength2 <- as.vector(scale(serps$bodyLength, center = TRUE, scale = TRUE))

#Ahora, la nueva variable creada bodyLength2 sera la que usaremos en el modelo, y sigue una distribució
```

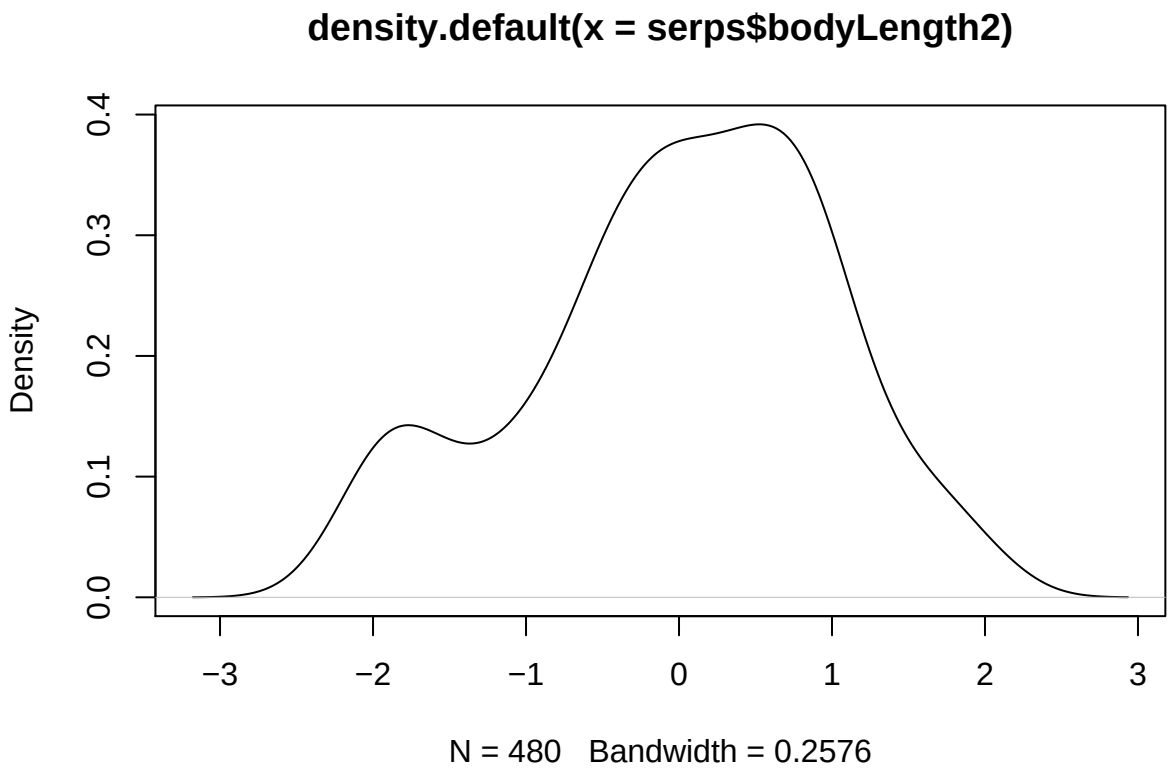
Veamos nuestras variables graficamente:

```
plot(density(serps$testScore))
```



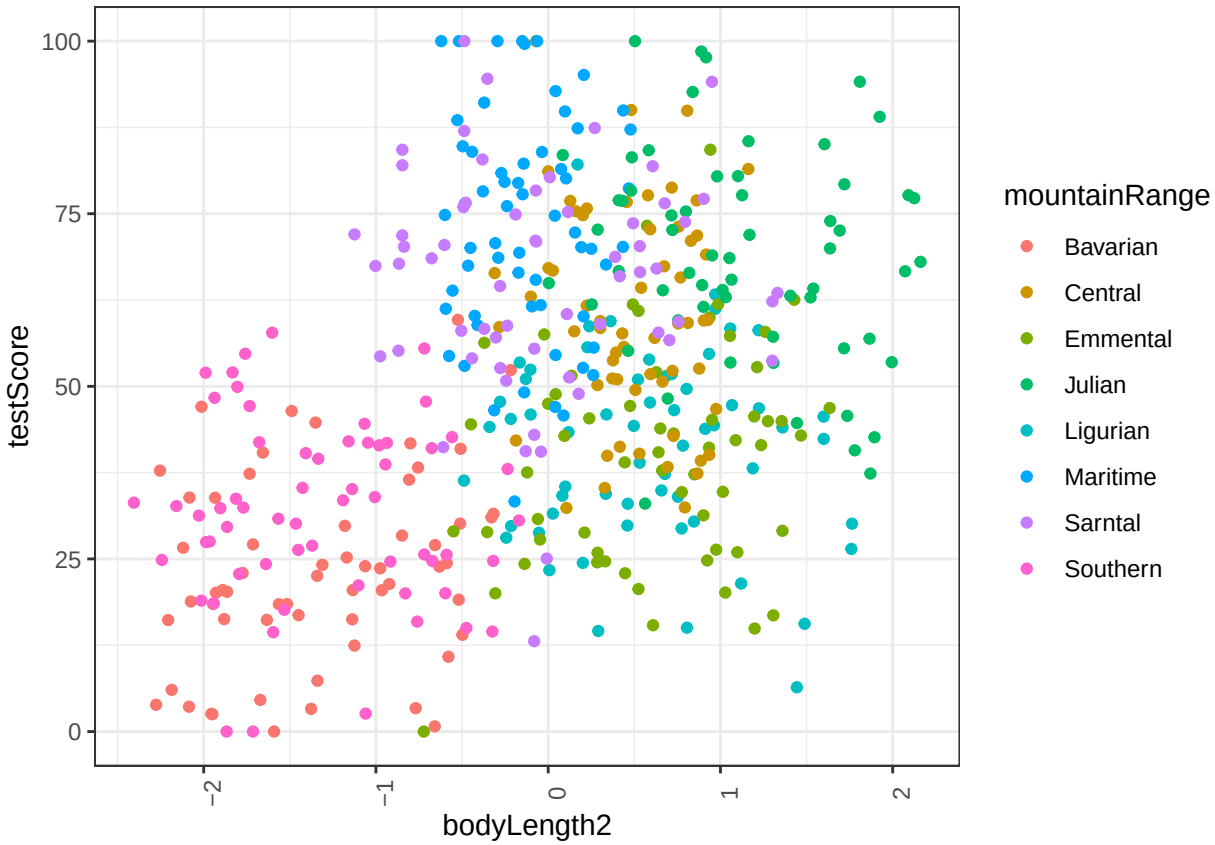

```
#Vemos que la variable respuesta testScore sigue una Noraml.
plot(density(serps$bodyLength2))


library(ggplot2)
```



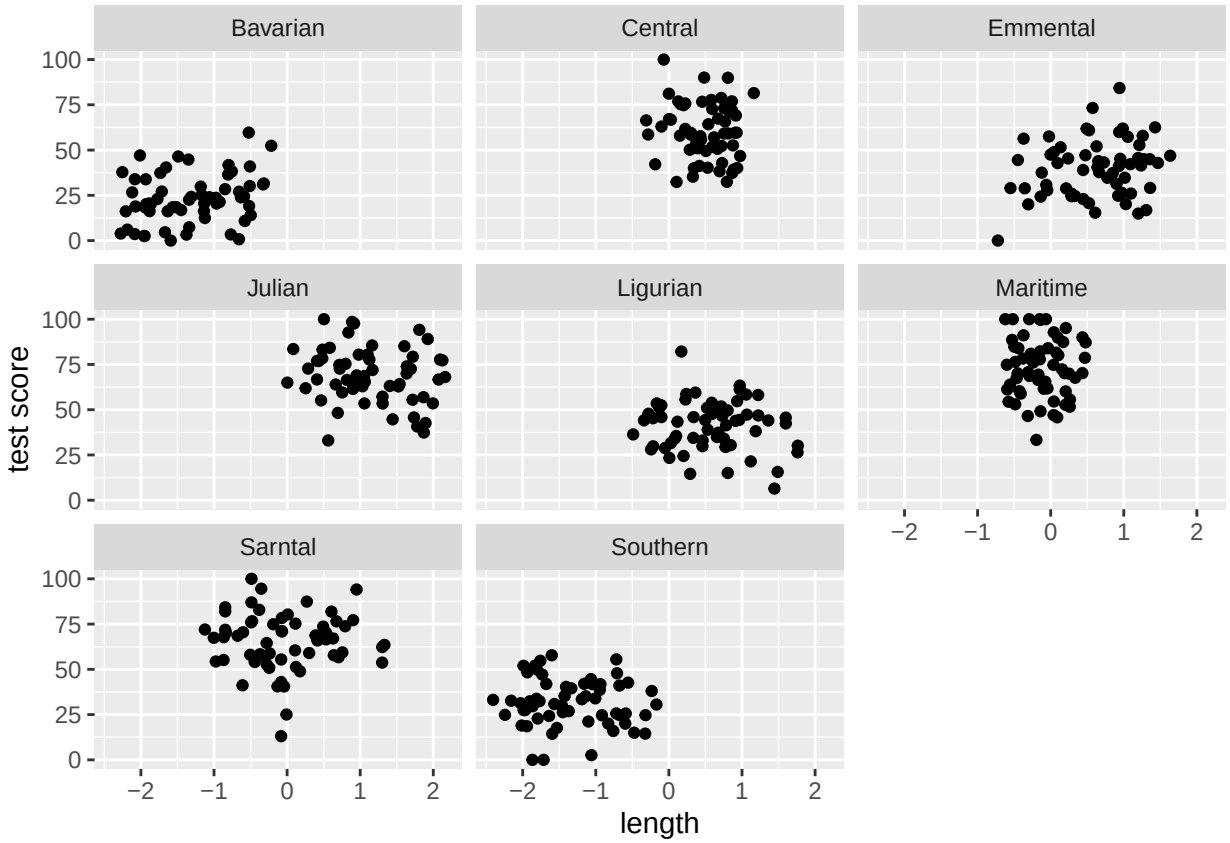

```
grafic1 <-ggplot(serps, aes(x = bodyLength2, y = testScore, col = mountainRange)) + geom_point()+ theme_
     y        = "testScore",
     fill     = "mountainRange")

plot(grafic1)
```




```
plot2 <- ggplot(aes(bodyLength2, testScore), data = serps) + geom_point() +
  facet_wrap(~ mountainRange) +   xlab("length") + ylab("test score")

plot2
```



Ajustemos el modelo lineal mitxto:

Tenemos una variable de respuesta, la puntuación de la prueba, y estamos intentando explicar parte de la variación en la puntuación de la prueba mediante el ajuste de la longitud del cuerpo como un efecto fijo. Pero la variable de respuesta tiene alguna variación residual (es decir, variación inexplicable) asociada con las cadenas montañosas. Mediante el uso de efectos aleatorios, estamos modelando esa variación inexplicable a través de la varianza .

Vamos a describir en detalle el modelo que nos permite explicar la inteligencia de la serpiente en función de la longitud del cuerpo:

- $Y_{ij} \sim N(\mu_i, \tau)$, siendo $\tau = \frac{1}{\sigma^2}$ y $sigma \sim Unif(0, 100)$
- $bodyLength2 -> X \sim N(0, 1)$
- $\mu_i = \beta_0 + A_j + \beta_1 x_i + \epsilon$
- $A_j \sim N(0, \tau_a)$
- $\beta_0 \sim$ -$N(0, 0.01)$ y $beta1 \sim N(0, 0.01)$

Para ajustar una regresión lineal a este conjunto de datos vamos a usar JAGS:

```
library(R2jags)
```

```
## Warning: package 'R2jags' was built under R version 3.6.3
```

```
## Loading required package: rjags
```

```
## Warning: package 'rjags' was built under R version 3.6.3

## Loading required package: coda

## Warning: package 'coda' was built under R version 3.6.3

## Linked to JAGS 4.3.0

## Loaded modules: basemod,bugs

##
## Attaching package: 'R2jags'

## The following object is masked from 'package:coda':
##
##     traceplot
```

Antes de que podamos analizar el conjunto de datos en JAGS, es necesario convertirlo a un formato diferente. Esto crea una lista con tres elementos: la respuesta y el predictor como vectores y el tamaño de la muestra como un solo número, i incluimos el número de sitios como un elemento adicional en la lista

```
Nsites <- length(levels(as.factor(serps$mountainRange)))
jagsdata <- with(serps, list(testScore = testScore, bodyLength2 = bodyLength2,
                             mountainRange = rep(1:8,rep(60,8)),
                             N = length(testScore), Nsites = Nsites))
```

Además del beta0 de intersección general , estimamos una intersección beta0 para cada sitio.

```
library(R2jags)

cat(file="modelo.jags", "
model{
   for (i in 1:N){
        testScore[i] ~ dnorm(mu[i], tau)
        mu[i] <- beta0 + a[mountainRange[i]] + beta1*bodyLength2[i]
    }
    # Prioris:
    beta0 ~ dnorm(0, 0.01)
    sigma_a ~ dunif(0, 100) # Desviación estándar del efecto aleatorio (variación entre sitios)
    tau_a <- 1 / (sigma_a * sigma_a)
    for (j in 1:Nsites){
        a[j] ~ dnorm(0, tau_a) # random intercept para cada sitio
    }
    beta1 ~ dnorm(0, 0.01)
    sigma ~ dunif(0, 100) # Desviación estándar del efecto fijo (variación entre sitios)
    tau <- 1 / (sigma * sigma)
}")
```

El siguiente código especifica los valores de los parámetros iniciales para el muestreador MCMC, elige los parámetros cuyas distribuciones posteriores se informarán y ejecuta el modelo en JAGS:

```
init_values <- function(){
    list(beta0 = rnorm(1), sigma_a = runif(1), beta1 = rnorm(1), sigma = runif(1))
}

params <- c("beta0", "beta1", "sigma", "sigma_a")

iteraciones <- 10000
burnin <- 1000
cadenas <- 3

modelo <- jags(
model.file= "modelo.jags", # aqui ponemos el modelo
inits = init_values,
data = jagsdata,
parameters.to.save = params,
n.chains= cadenas,
n.iter = iteraciones,
n.burnin = burnin
)
```

```
## module glm loaded

## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 480
##    Unobserved stochastic nodes: 12
##    Total graph size: 2422
##
## Initializing model
```

Visualización del modelo:

```
modelo
```

```
## Inference for Bugs model at "modelo.jags", fit using jags,
##  3 chains, each with 10000 iterations (first 1000 discarded), n.thin = 9
##  n.sims = 3000 iterations saved
##           mu.vect sd.vect     2.5%      25%      50%      75%    97.5%  Rhat
## beta0      16.645  11.106   -4.565    8.789   16.452   24.438   38.332 1.001
## beta1       0.280   1.269   -2.255   -0.597    0.302    1.098    2.767 1.001
## sigma      14.999   0.485   14.080   14.676   14.988   15.317   15.966 1.001
## sigma_a    45.031  16.589   19.607   32.631   42.386   54.689   84.755 1.001
## deviance 3960.219   4.570 3953.408 3956.852 3959.538 3962.731 3970.873 1.001
##          n.eff
## beta0     3000
## beta1     3000
## sigma     3000
## sigma_a   3000
## deviance  3000
```

```
##
## 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 = var(deviance)/2)
## pD = 10.4 and DIC = 3970.7
## DIC is an estimate of expected predictive error (lower deviance is better).
```

El AIC lo deberiamos comparar con un modelo sin estos efectos aleatorios para ver si de verdad són relevantes. La sigma_a tiene un valor grande que nos indica que és bastante influyente.

```
library(coda)
mean(modelo$BUGSoutput$sims.list$beta0)
```

```
## [1] 16.64487
```

```
mean(modelo$BUGSoutput$sims.list$beta1)
```

```
## [1] 0.2796714
```

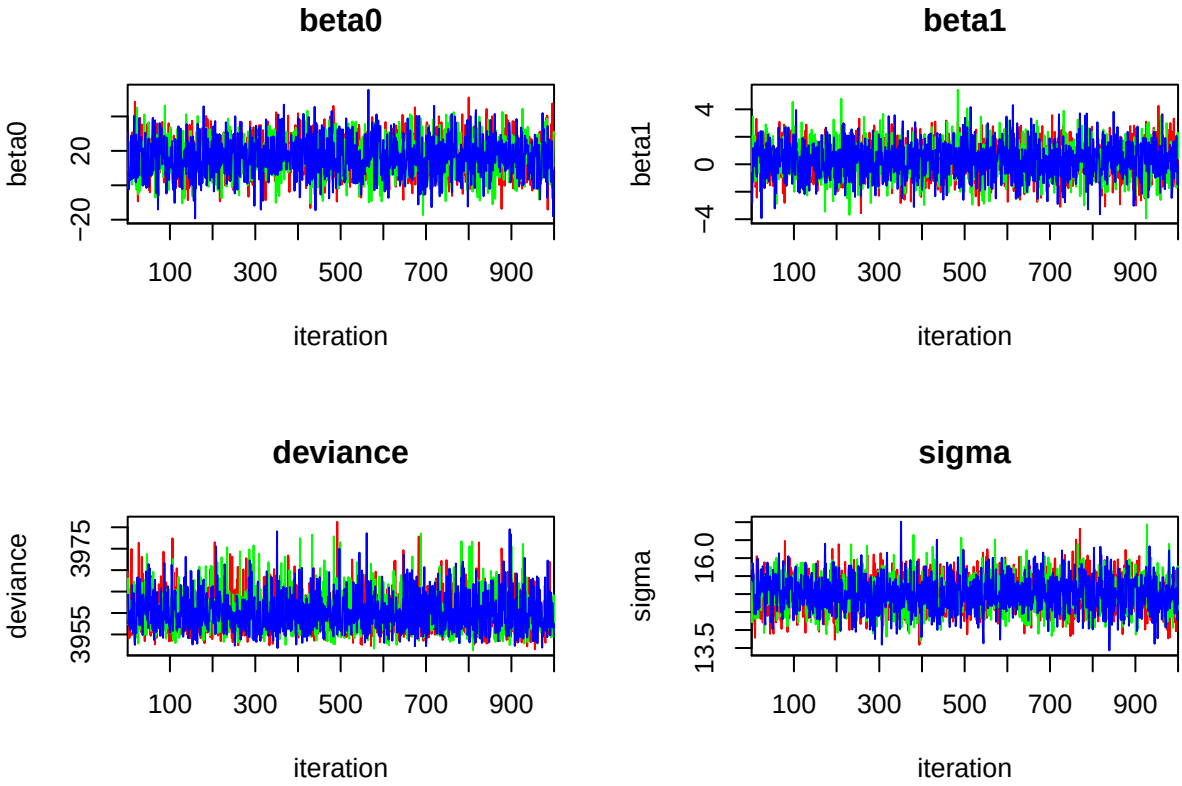
```
mean(modelo$BUGSoutput$sims.list$sigma)
```

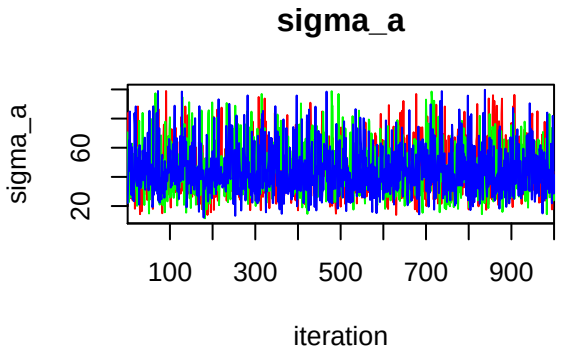
```
## [1] 14.99944
```

```
mean(modelo$BUGSoutput$sims.list$sigma_a)
```

```
## [1] 45.03062
```

```
traceplot(modelo, mfrow = c(2, 2), ask = F)
```


beta0
beta0
−20
20
100
300
500
700
900
iteration
beta1
beta1
−4
0
4
100
300
500
700
900
iteration
deviance
deviance
3955
3975
100
300
500
700
900
iteration
sigma
sigma
13.5
16.0
100
300
500
700
900
iteration



Vemos que la convergencia és bastante buena.

Modelo para comparar:

```
jagsdata1 <- with(serps, list(testScore = testScore,
                              bodyLength2= bodyLength2,
                              N = length(testScore)))


cat(file="modelo.jags1", "
model{
    for (i in 1:N){
        testScore[i] ~ dnorm(mu[i], tau)
        mu[i] <- beta0 + beta1 * bodyLength2[i]
    }

    beta0 ~ dnorm(0, 0.01)
    beta1 ~ dnorm(0, 0.01)
    sigma ~ dunif(0, 100)
    tau <- 1 / (sigma * sigma)
}")

init_values <- function(){
```

```
    list(beta0 = rnorm(1), beta1 = rnorm(1),sigma =runif(1))}

params <- c("beta0", "beta1","sigma")

iteraciones <- 10000
burnin <- 1000
cadenas <- 3

modelo1 <- jags(
model.file= "modelo.jags1",
inits = init_values,
data = jagsdata,
parameters.to.save = params,
n.chains= cadenas,
n.iter = iteraciones,
n.burnin = burnin
)
```

Modelo lineal simple. Sin efectos fijos.

```
## Warning in jags.model(model.file, data = data, inits = init.values, n.chains =
## n.chains, : Unused variable "mountainRange" in data

## Warning in jags.model(model.file, data = data, inits = init.values, n.chains =
## n.chains, : Unused variable "Nsites" in data

## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 480
##    Unobserved stochastic nodes: 3
##    Total graph size: 1930
##
## Initializing model
```

```
modelo1
```

```
## Inference for Bugs model at "modelo.jags1", fit using jags,
##  3 chains, each with 10000 iterations (first 1000 discarded), n.thin = 9
##  n.sims = 3000 iterations saved
##           mu.vect sd.vect     2.5%      25%      50%      75%    97.5%  Rhat
## beta0      49.930   1.006   47.884   49.264   49.961   50.594   51.875 1.001
## beta1       8.901   0.961    7.000    8.259    8.898    9.559   10.823 1.001
## sigma      21.270   0.684   20.026   20.798   21.237   21.702   22.731 1.001
## deviance 4295.268   2.751 4292.222 4293.270 4294.560 4296.429 4302.427 1.001
##          n.eff
## beta0     3000
## beta1     3000
## sigma     3000
## deviance  3000
##
```

```
## 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 = var(deviance)/2)
## pD = 3.8 and DIC = 4299.1
## DIC is an estimate of expected predictive error (lower deviance is better).
```

Podemos observar que el AIC es mayor que el obtenido con los efectos aleatorios, entonces, estos efectos mejoran el modelo.

```
library(coda)
mean(modelo1$BUGSoutput$sims.list$beta0)
```

```
## [1] 49.92991
```

```
mean(modelo1$BUGSoutput$sims.list$beta1)
```

```
## [1] 8.900718
```

```
traceplot(modelo1, mfrow = c(2, 2), ask = F)
```