Rats - An Bayesian analysis

Ivan Ferrante (1563390)

```
library(R2jags)
library(ggplot2)
```

Introduction

The data contains 30 young rats whose weights were measured weekly for five weeks. Dependent variable Y_{ij} is the weight of the i^{th} rat at age x_j.

```
rats.data  <- list(x = c(8.0, 15.0, 22.0, 29.0, 36.0), xbar = 22, N = 30, T = 5,
        Y = c(151, 199, 246, 283, 320,
                              145, 199, 249, 293, 354,
                              147, 214, 263, 312, 328,
                              155, 200, 237, 272, 297,
                              135, 188, 230, 280, 323,
                              159, 210, 252, 298, 331,
                              141, 189, 231, 275, 305,
                              159, 201, 248, 297, 338,
                              177, 236, 285, 350, 376,
                              134, 182, 220, 260, 296,
                              160, 208, 261, 313, 352,
                              143, 188, 220, 273, 314,
                              154, 200, 244, 289, 325,
                              171, 221, 270, 326, 358,
                              163, 216, 242, 281, 312,
                              160, 207, 248, 288, 324,
                              142, 187, 234, 280, 316,
                              156, 203, 243, 283, 317,
                              157, 212, 259, 307, 336,
                              152, 203, 246, 286, 321,
                              154, 205, 253, 298, 334,
                              139, 190, 225, 267, 302,
                              146, 191, 229, 272, 302,
                              157, 211, 250, 285, 323,
                              132, 185, 237, 286, 331,
                              160, 207, 257, 303, 345,
                              169, 216, 261, 295, 333,
                              157, 205, 248, 289, 316,
                              137, 180, 219, 258, 291,
                              153, 200, 244, 286, 324))

Y <- matrix(rats.data$Y, nrow = rats.data$N, ncol = rats.data$T, byrow = TRUE)
Y
```

```
##       [,1] [,2] [,3] [,4] [,5]
##  [1,]  151  199  246  283  320
##  [2,]  145  199  249  293  354
##  [3,]  147  214  263  312  328
##  [4,]  155  200  237  272  297
##  [5,]  135  188  230  280  323
##  [6,]  159  210  252  298  331
##  [7,]  141  189  231  275  305
##  [8,]  159  201  248  297  338
##  [9,]  177  236  285  350  376
## [10,]  134  182  220  260  296
## [11,]  160  208  261  313  352
## [12,]  143  188  220  273  314
## [13,]  154  200  244  289  325
## [14,]  171  221  270  326  358
## [15,]  163  216  242  281  312
## [16,]  160  207  248  288  324
## [17,]  142  187  234  280  316
## [18,]  156  203  243  283  317
## [19,]  157  212  259  307  336
## [20,]  152  203  246  286  321
## [21,]  154  205  253  298  334
## [22,]  139  190  225  267  302
## [23,]  146  191  229  272  302
## [24,]  157  211  250  285  323
## [25,]  132  185  237  286  331
## [26,]  160  207  257  303  345
## [27,]  169  216  261  295  333
## [28,]  157  205  248  289  316
## [29,]  137  180  219  258  291
## [30,]  153  200  244  286  324
```

```r
T <- rats.data$T
T
```

```
## [1] 5
```

```r
x <-  rats.data$x
x
```

```
## [1]  8 15 22 29 36
```

```r
xbar <- rats.data$xbar
xbar
```

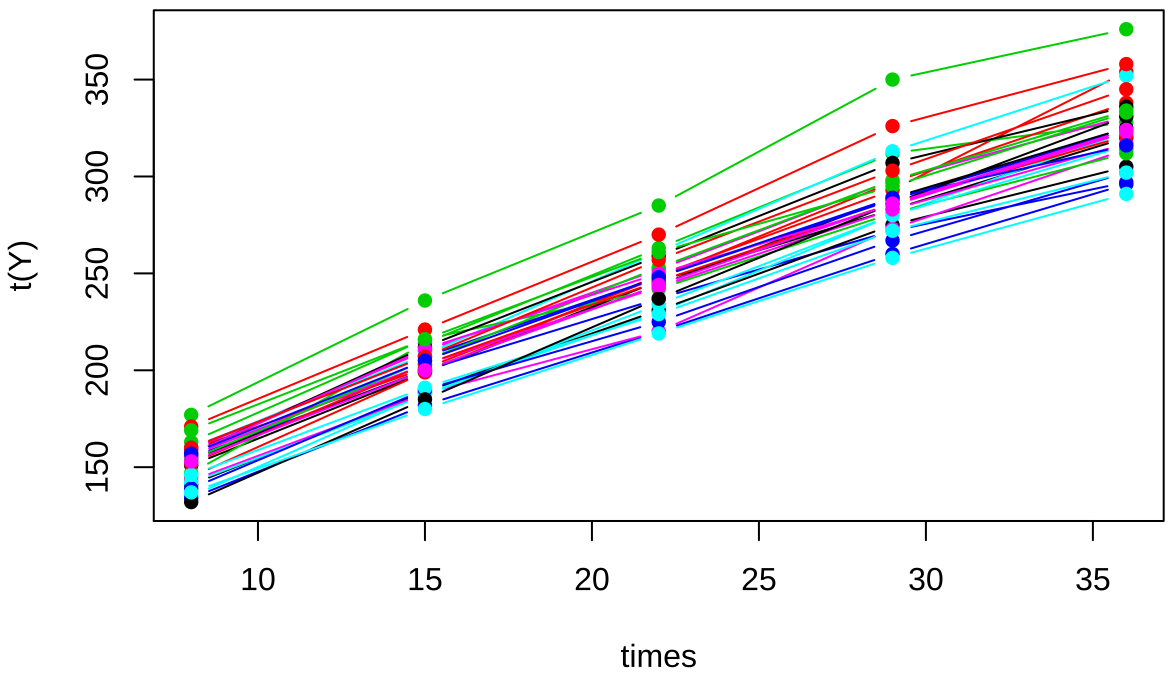
```
## [1] 22
```

```r
N <- rats.data$N
N
```

```
## [1] 30
```

We can plot each rats growth in the time, by pick the transpose matrix of our data:

```r
times = as.numeric(rats.data$x)
matplot(times, t(Y), type = "b", pch = 16, lty = 1)
```



We can use the frequentistic approach in order to predict the weight Y from time x_j for each rat.

```
# LINEAR REGRESSION
array_intercept <- rep(NA,N)
array_slope <- rep(NA,N)
for(i in 1:N)
{
  linear_reg <- lm(Y[i,] ~ x)
  array_intercept[i] <- linear_reg$coefficients[[1]]
  array_slope[i] <- linear_reg$coefficients[[2]]
}
array_intercept
```

```
##  [1] 107.17143  87.08571 108.22857 120.31429  84.11429 114.22857  98.08571
##  [8] 105.91429 123.88571  92.05714 105.11429  93.40000 106.94286 118.65714
## [15] 128.71429 116.85714  93.20000 114.05714 111.82857 109.28571 106.42857
## [22]  97.94286 104.48571 117.60000  77.37143 107.94286 126.88571 116.65714
## [29]  95.68571 106.88571
```

```
array_slope
```

```
##  [1] 6.028571 7.314286 6.571429 5.085714 6.685714 6.171429 5.914286
##  [8] 6.485714 7.314286 5.742857 6.985714 6.100000 6.157143 6.842857
## [15] 5.185714 5.842857 6.300000 5.742857 6.471429 6.014286 6.471429
## [22] 5.757143 5.614286 5.800000 7.128571 6.657143 5.814286 5.742857
## [29] 5.514286 6.114286
```

```
# INTERCEPT MEAN
mean_alpha <- mean(array_intercept)
mean_alpha
```

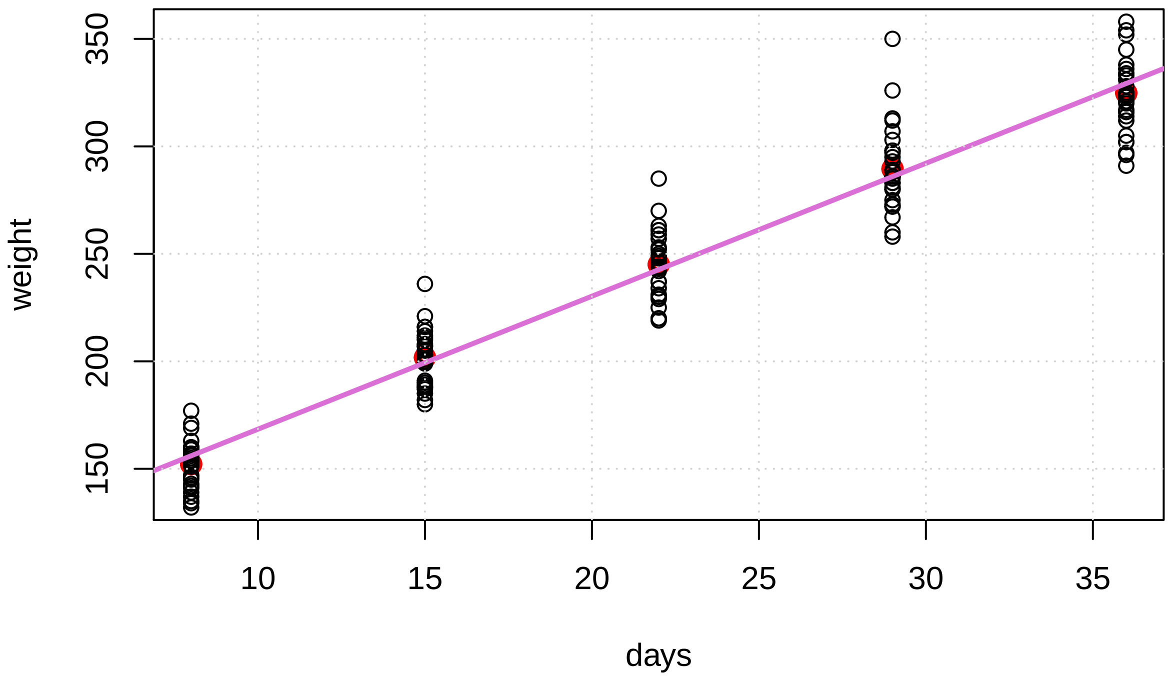
```
## [1] 106.5676
```

```
# SLOPE MEAN
mean_beta <- mean(array_slope)
```

```
mean_beta
```

```
## [1] 6.185714
```

```
plot(x,colMeans(Y), lwd=4, xlab = "days", ylab = "weight",
     col="red", ylim=c(135,355))
points(rep(x[1],N), Y[,1])
points(rep(x[2],N), Y[,2])
points(rep(x[3],N), Y[,3])
points(rep(x[4],N), Y[,4])
points(rep(x[5],N), Y[,5])
abline(mean_alpha, mean_beta, col="orchid", lwd=2.5)
grid()
```



First Model: Normal hierarchical model

The first model, suggested by WinBugs, is essentially a random effects linear growth curve

$$Y_{ij} \sim Normal(\alpha_i + \beta_i(x_j), \tau_c)$$

$$\alpha_i \sim Normal(\alpha_c, \tau_\alpha)$$

$$\beta_i \sim Normal(\beta_c, \tau_\beta)$$

$$\alpha_c \sim Normal(0, 1.0E-6)$$

$$\beta_c \sim Normal(0, 1.0E-6)$$

$$\tau_\alpha \sim Gamma(1.0E-3, 1.0E-3)$$

$$\tau_\beta \sim Gamma(1.0E-3, 1.0E-3)$$

$$\tau_c \sim Gamma(1.0E-3, 1.0E-3)$$

where $\bar{x}$ and τ represent the precision of Normal distribution. α_c , β_c , τ_α , τ_β , τ_c are "non-informative" priors. In the WinBugs guidelines of this dataset, the x_j is standadized around their mean in order to reduce dependence between α_i and β_i in their likelihood. We'll not do it and we will use the priors mentioned above.

```
rats.model <- function()  {

  for (i in 1:N) {

      for (j in 1:T) {

        Y[i,j]    ~ dnorm(mu[i,j],tau.c)
        mu[i,j] <- alpha[i] + beta[i]*(x[j]);

        }

      alpha[i] ~ dnorm(alpha.c,tau.alpha);
      beta[i]  ~ dnorm(beta.c,tau.beta); }

    alpha.c    ~ dnorm(0,1.0E-6);
    beta.c     ~ dnorm(0,1.0E-6);
    tau.c      ~ dgamma(1.0E-3,1.0E-3);
    tau.alpha ~ dgamma(1.0E-3,1.0E-3);
    tau.beta  ~ dgamma(1.0E-3,1.0E-3);
    sigma <- 1.0 / sqrt(tau.c)
    x.bar    <- mean(x[]);
    alpha0 <- alpha.c - beta.c * x.bar
    }
```

Now we can define the vectors of the data matrix, the starting values and the name of the parameter for JAGS:

```
rats.data  <- list("Y", "x", "T", "N")
rats.params <- c("tau.c", "alpha.c", "beta.c", "tau.alpha", "tau.beta")

## Define the starting values for JAGS

rats.inits <- function(){
  list(alpha = c(250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250,
                 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250),
       beta  = c(6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
                 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6),
       alpha.c = 150, beta.c = 10,
       tau.c = 1, tau.alpha = 1, tau.beta = 1)
}
```

At this point we can run our JAGS function:

```
ratsfit <- jags(data=rats.data, inits=rats.inits, rats.params, n.chains=2, n.iter=10000, n.burnin=1000,
```

```
## module glm loaded
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 150
##    Unobserved stochastic nodes: 65
##    Total graph size: 532
##
## Initializing model
```

```r
ratsfit.mcmc <- as.mcmc(ratsfit)

summary(ratsfit.mcmc)
```

```
##
## Iterations = 1001:10000
## Thinning interval = 1
## Number of chains = 2
## Sample size per chain = 9000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##                 Mean        SD  Naive SE Time-series SE
## alpha.c   106.53098  2.283557 1.702e-02      3.506e-02
## beta.c      6.18762  0.105316 7.850e-04      1.445e-03
## deviance  969.21493 14.400278 1.073e-01      2.947e-01
## tau.alpha   0.01005  0.003790 2.825e-05      7.833e-05
## tau.beta    4.28993  1.489280 1.110e-02      2.526e-02
## tau.c       0.02700  0.004012 2.990e-05      6.311e-05
##
## 2. Quantiles for each variable:
##
##                 2.5%       25%       50%       75%     97.5%
## alpha.c   1.020e+02 1.050e+02 1.065e+02 108.02361 111.06482
## beta.c    5.980e+00 6.120e+00 6.189e+00   6.25767   6.39527
## deviance  9.434e+02 9.591e+02 9.683e+02 978.42834 999.37132
## tau.alpha 4.634e-03 7.374e-03 9.403e-03   0.01192   0.01923
## tau.beta  2.076e+00 3.252e+00 4.076e+00   5.07471   7.72017
## tau.c     1.967e-02 2.420e-02 2.683e-02   0.02960   0.03541
```

We can compare the results of our first model with the frequentist approach used above:

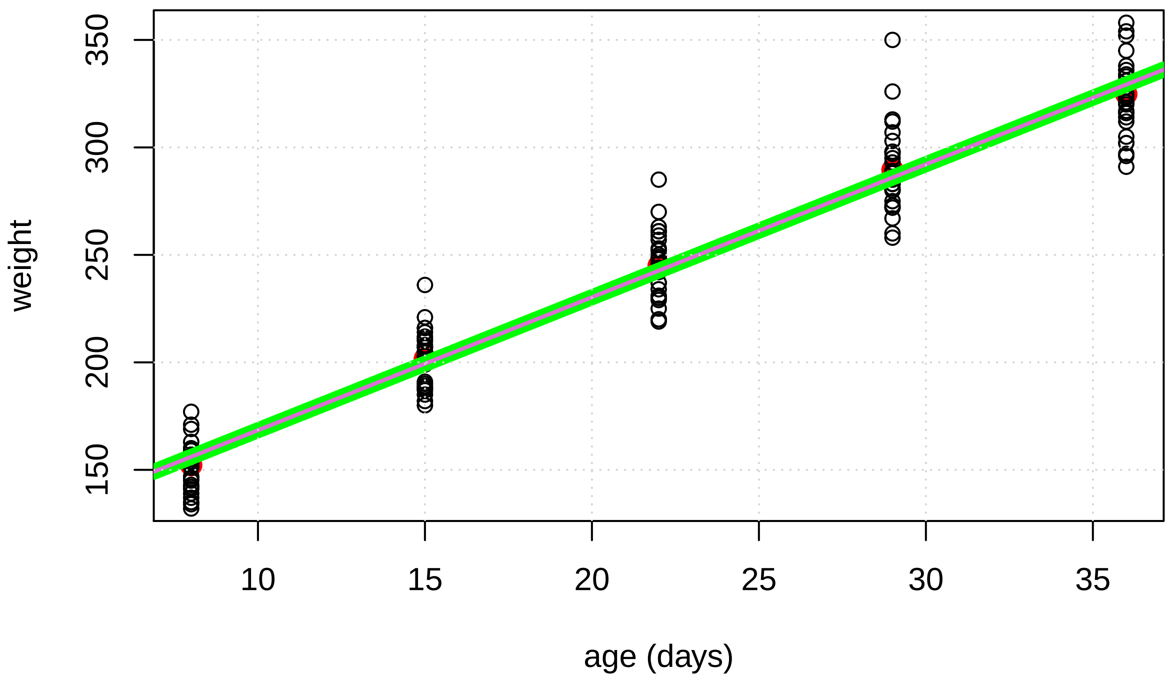
```r
# INTERCEPT MEAN MODEL 1
ratsfit$BUGSoutput$mean$alpha.c
```

```
## [1] 106.531
```

```r
# SLOP MEAN MODEL 2
ratsfit$BUGSoutput$mean$beta.c
```

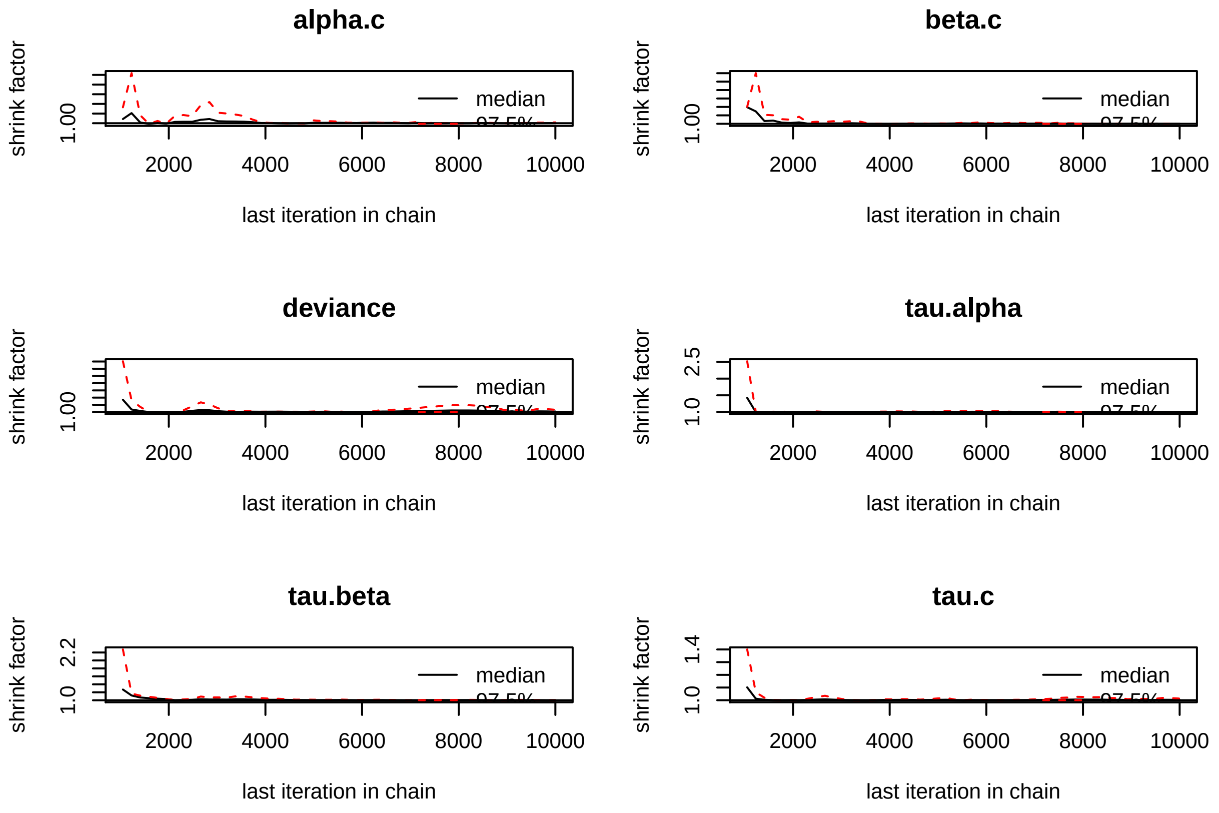
```
## [1] 6.187619
```

```r
# MODEL 1 vs. FREQ APPROACH
plot(x,colMeans(Y), lwd=4, xlab = "age (days)", ylab = "weight",
     col="red", ylim=c(135,355))
points(rep(x[1],N), Y[,1])
points(rep(x[2],N), Y[,2])
points(rep(x[3],N), Y[,3])
points(rep(x[4],N), Y[,4])
points(rep(x[5],N), Y[,5])
abline(ratsfit$BUGSoutput$mean$alpha.c,
       ratsfit$BUGSoutput$mean$beta.c, col="green", lwd=8)
abline(mean_alpha, mean_beta, col="orchid", lwd=2)
grid()
```



Another graphic method of diagnostic for convergence that we have seen at lessons, is **Gelman's plot**. This method can be applied when we have more than one chain and consists in calculating the 'within variance' of each chain, subtract from this value the 'between variance' of two chain and then multiply by a scaling factor.

```
gelman.plot(ratsfit.mcmc)
```



Second Model: Uniform priors.

In this second model we want try to change some prior parameters and see what happen at our model. Our prior sigma.alpha and sigma.beta are distributed as uniform. So we will be:

$$\alpha_i \sim Normal(\alpha_c, \tau_\alpha)$$

$$\beta_i \sim Normal(\beta_c, \tau_\beta)$$
$$\alpha_c \sim Normal(0, 1.0E-6)$$

$$\beta_c \sim Normal(0, 1.0E-6)$$
$$\tau_\alpha, \tau_\beta, \tau_c \sim Unif(0, 100)$$

```
rats.model2 <- function()  {

  for (i in 1:N) {

      for (j in 1:T) {

        mu[i,j] <- alpha[i] + beta[i]*(x[j]);
        Y[i,j]   ~ dnorm(mu[i,j],tau.c)

        }

      alpha[i] ~ dnorm(alpha.c,tau.alpha);
      beta[i]  ~ dnorm(beta.c,tau.beta); }

    alpha.c    ~ dnorm(0,1.0E-6);
    beta.c     ~ dnorm(0,1.0E-6);
    tau.c     <- 1.0 / (sigma*sigma);
    sigma  ~ dunif(0,100)
    tau.alpha <- 1.0 / (sigma.alpha*sigma.alpha);
    sigma.alpha ~ dunif(0,100)
    tau.beta <- 1/(sigma.beta*sigma.beta)
    sigma.beta ~ dunif(0,100)
    x.bar    <- mean(x[]);
    alpha0 <- alpha.c - beta.c * x.bar
    }



rats.data2  <- list("Y", "x", "T", "N")
rats.params2 <- c("alpha.c", "beta.c", "sigma", "sigma.alpha", "sigma.beta")

## Define the starting values for JAGS

rats.inits2 <- function(){
  list(alpha = c(250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250,
                 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250, 250),
       beta  = c(6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
                 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6),
       alpha.c = 150, beta.c = 10,
       sigma = 1, sigma.alpha = 1, sigma.beta = 1)
}

ratsfit2 <- jags(data=rats.data2, inits=rats.inits2, rats.params2, n.chains=2, n.iter=10000, n.burnin=1
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 150
##    Unobserved stochastic nodes: 65
##    Total graph size: 536
##
## Initializing model
```

```
ratsfit2.mcmc <- as.mcmc(ratsfit2)

summary(ratsfit2.mcmc)
```

```
##
## Iterations = 1001:10000
## Thinning interval = 1
## Number of chains = 2
## Sample size per chain = 9000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##                 Mean      SD  Naive SE Time-series SE
## alpha.c     106.5238  2.3499 0.0175149       0.035242
## beta.c        6.1875  0.1083 0.0008073       0.001585
## deviance    969.8290 14.5171 0.1082042       0.310270
## sigma         6.1720  0.4679 0.0034878       0.008088
## sigma.alpha  10.7490  1.9566 0.0145839       0.039816
## sigma.beta    0.5153  0.0887 0.0006611       0.001587
##
## 2. Quantiles for each variable:
##
##                 2.5%      25%      50%      75%     97.5%
## alpha.c     101.8581 104.9824 106.5360 108.0898  111.1189
## beta.c        5.9769   6.1149   6.1872   6.2586    6.4025
## deviance    943.6416 959.4107 969.1324 979.3029 1000.4433
## sigma         5.3411   5.8415   6.1410   6.4706    7.1690
## sigma.alpha   7.3373   9.3898  10.6034  11.9566   15.0640
## sigma.beta    0.3634   0.4537   0.5082   0.5673    0.7103
```

Again we can compare the results obtained by our second model with the frequentist approach computed above:

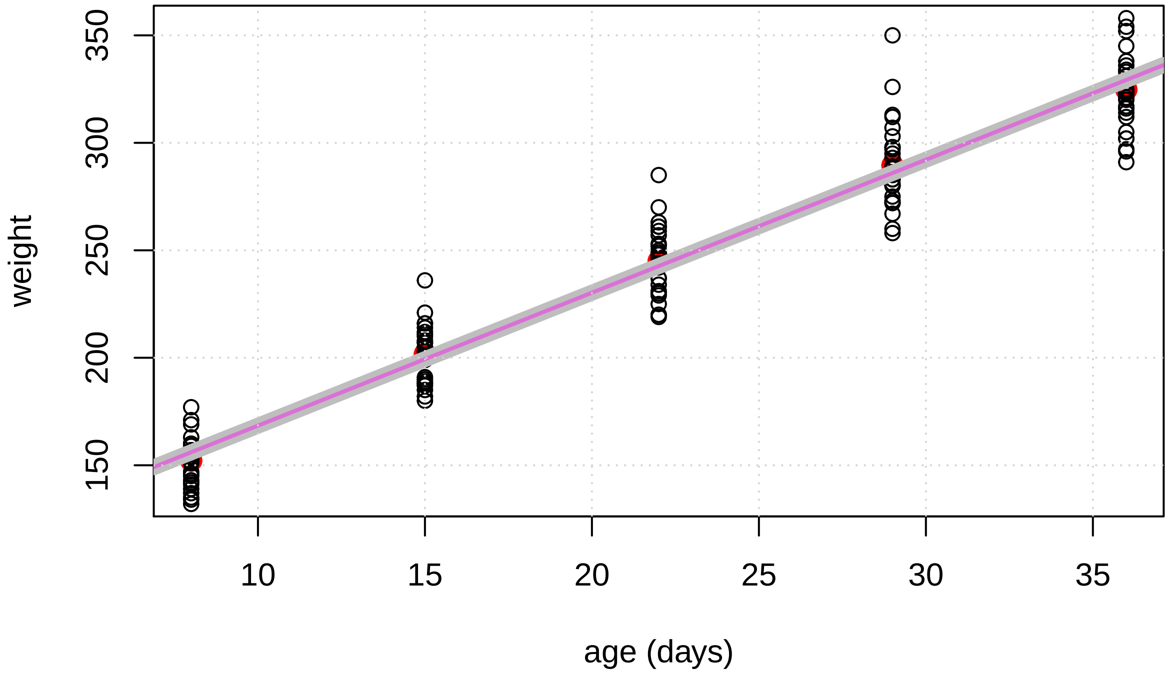
```
# INTERCEPT MEAN MODEL 2
ratsfit2$BUGSoutput$mean$alpha.c
```

```
## [1] 106.5238
```

```r
# SLOPE MEAN MODEL 2
ratsfit2$BUGSoutput$mean$beta.c
```

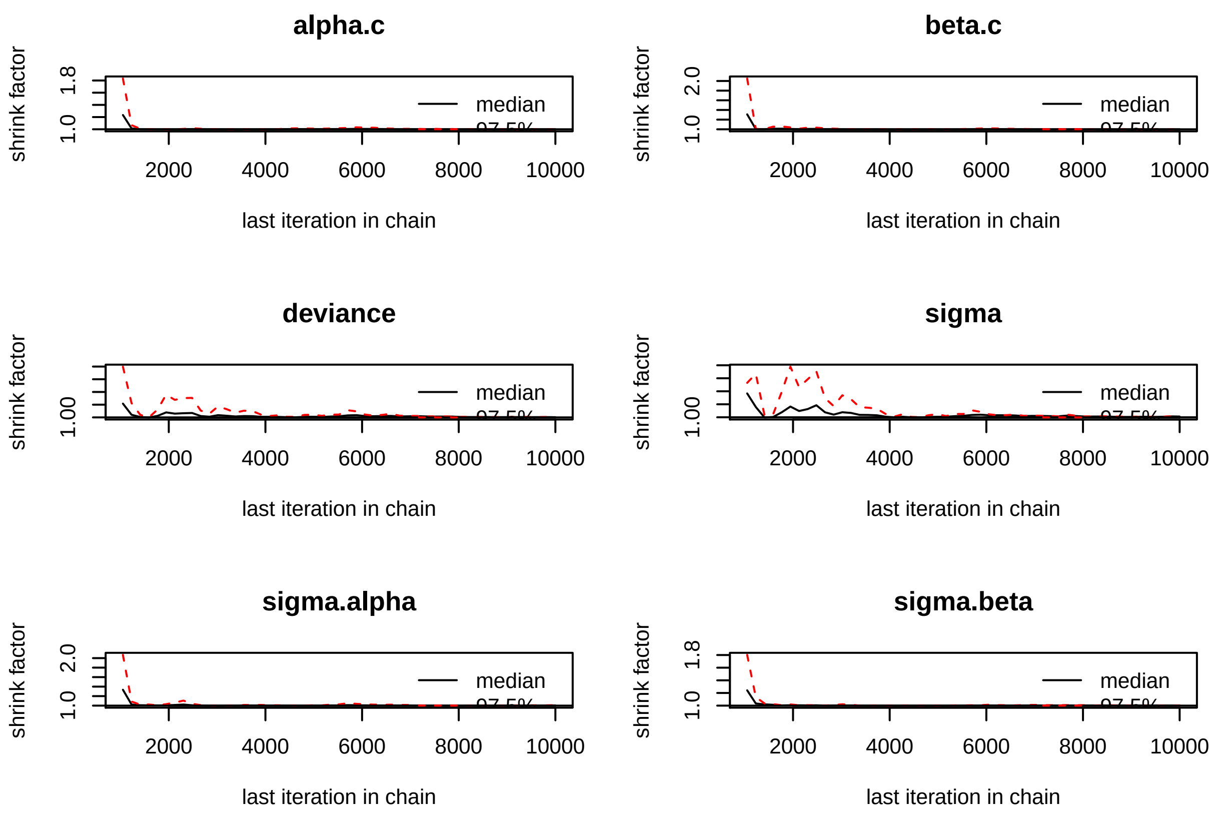
```
## [1] 6.187475
```

```r
# MODEL 2 vs. FREQ APPROACH
plot(x,colMeans(Y), lwd=4, xlab = "age (days)", ylab = "weight",
     col="red", ylim=c(135,355))
points(rep(x[1],N), Y[,1])
points(rep(x[2],N), Y[,2])
points(rep(x[3],N), Y[,3])
points(rep(x[4],N), Y[,4])
points(rep(x[5],N), Y[,5])
abline(ratsfit2$BUGSoutput$mean$alpha.c,
       ratsfit2$BUGSoutput$mean$beta.c, col="grey", lwd=8)
abline(mean_alpha, mean_beta, col="orchid", lwd=2)
grid()
```



and again we can compute the Gelman's plot:

```r
gelman.plot(ratsfit2.mcmc)
```



Global comparision between models:

One way to analyze and compare our models is the **deviance information criterion** that analyze the model in terms of number of parameters and deviance. In general we prefer a model with **lower DIC**. We can consider the DIC a kind of penalized deviance, computed as follows:

$$DIC = pD + \hat{D}$$

Where:

$$pD = penality$$

$$\hat{D} = mean\ \ deviance$$

In the previous models we have:

```r
ratsfit$BUGSoutput$DIC
```

```
## [1] 1072.826
```

```r
ratsfit2$BUGSoutput$DIC
```

```
## [1] 1075.149
```

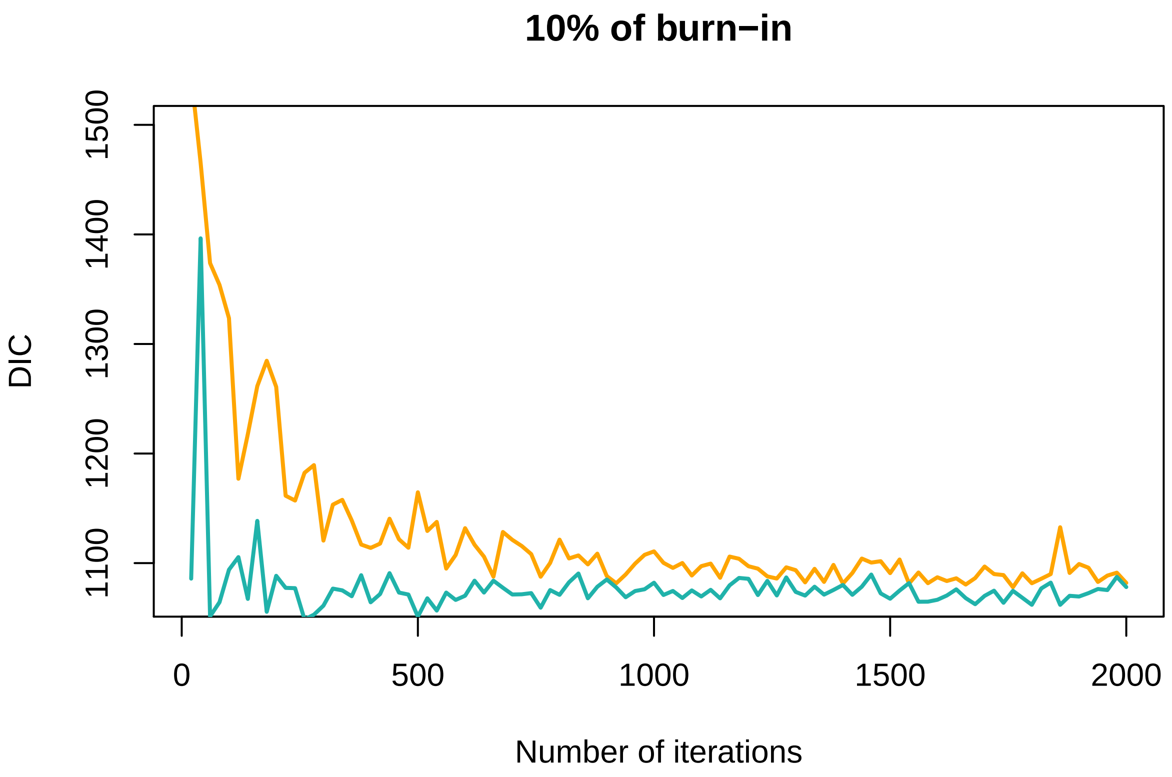
The performances of each model are very influenced by number of iteration and by the burn-in parameter. We can simulate how DIC vary as the number of iteration increase with 3 different percentage of burn-in: 10%, 25% and 50%

```r
# For this section of code, output is disabled in order to avoid printing the same JAGS initialization

n_rep <- 100
iter <- 2000
```

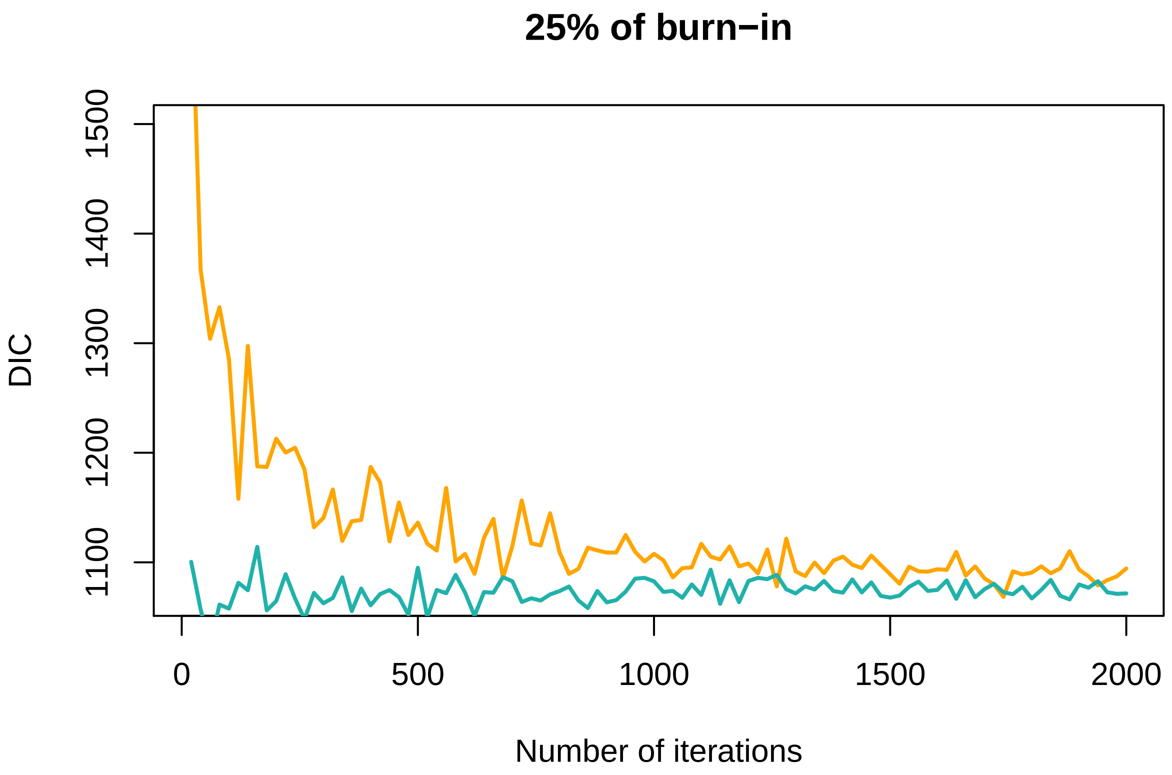
```
DICmodel1_M = list()
DICmodel2_M = list()
DICmodel1 = rep(0,n_rep)
DICmodel2 = rep(0,n_rep)


for (j in 1:3){

  burn_in = c(10,25,50)
  perc = burn_in[j]/100
  DICmodel1 = rep(0,n_rep)
  DICmodel2 = rep(0,n_rep)

  for (i in 1:n_rep){

    ni=iter*i/n_rep
    nb = perc*ni

    ratsfit_SIM <- jags(data=rats.data, inits=rats.inits, rats.params, n.chains=2, n.iter=ni, n.burnin=

    ratsfit2_SIM <- jags(data=rats.data2, inits=rats.inits2, rats.params2, n.chains=2, n.iter=ni, n.burn

    DICmodel1[i] = ratsfit_SIM$BUGSoutput$DIC
    DICmodel2[i] = ratsfit2_SIM$BUGSoutput$DIC

  }

  DICmodel1_M = c(DICmodel1_M,list(DICmodel1))
  DICmodel2_M = c(DICmodel2_M,list(DICmodel2))
}


xfit = (1:n_rep)*iter/n_rep
```

Finally we can plot the results! Our DIC simulation of first model is represented by orange line and our DIC simulation of second model is represented by light blue line. Although the DIC computed in the previous step by BUGSoutput for our models is very similar (around 1070 for both models), from these simulations we can see how the value of the DIC changes in different way for our models when the number of iteration and burn-in values change.

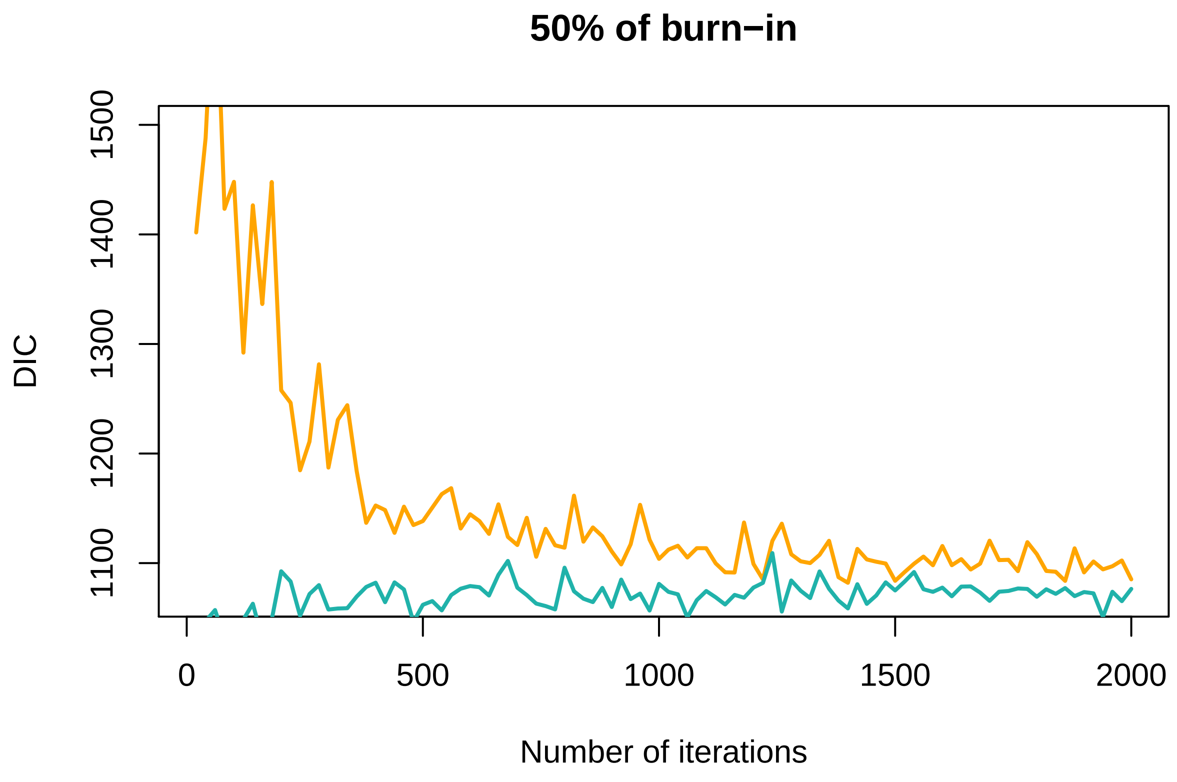
```
yrange<-c(min(DICmodel1_M[[1]],DICmodel1_M[[2]],DICmodel1_M[[3]]),1500)


plot(xfit,DICmodel1_M[[1]],lwd=2,type="l",ylim=yrange,col='orange',ylab='DIC', xlab = 'Number of iterat
lines(xfit,DICmodel2_M[[1]],lwd=2,type="l",col='lightseagreen')
```



```
plot(xfit,DICmodel1_M[[2]],lwd=2,type="l",ylim=yrange,col='orange',ylab='DIC', xlab = 'Number of iterat
lines(xfit,DICmodel2_M[[2]],lwd=2,type="l",col='lightseagreen')
```



```
plot(xfit,DICmodel1_M[[3]],lwd=2,type="l",ylim=yrange,col='orange',ylab='DIC', xlab = 'Number of iterat
lines(xfit,DICmodel2_M[[3]],lwd=2,type="l",col='lightseagreen')
```


50% of burn−in
DIC
1100
1200
1300
1400
1500
0
500
1000
1500
2000
Number of iterations