Homework 3

Davide Berasi

2023-06-15

Exercise 1: Mixture of exponential data

Suppose a company obtains boxes of electronic parts from a particular supplier. It is known that 80% of the lots are acceptable and the lifetimes of the "acceptable" parts follow an exponential distribution with mean λ_A. Unfortunately, 20% of the lots are unacceptable and the lifetimes of the "bad" parts are exponential with mean λ_B, where $\lambda_A > \lambda_B$. Suppose $y_1, \ldots, y_n$, are the lifetimes of n inspected parts that can come from either acceptable and unacceptable lots. The y_is are a random sample from the mixture distribution

$$h(y|\lambda_A, \lambda_B) = p\frac{\exp(-y/\lambda_A)}{\lambda_A} + (1-p)\frac{\exp(-y/\lambda_B)}{\lambda_B} \ ,$$

where $p = 0.8$. Suppose (λ_A, λ_B) are assigned the noninformative prior proportional to $1/(\lambda_A \ \lambda_B)$. The following function computes the log posterior density of the transformed parameters $\theta = (\theta_A, \theta_B) = (\log \lambda_A, \log \lambda_B)$:

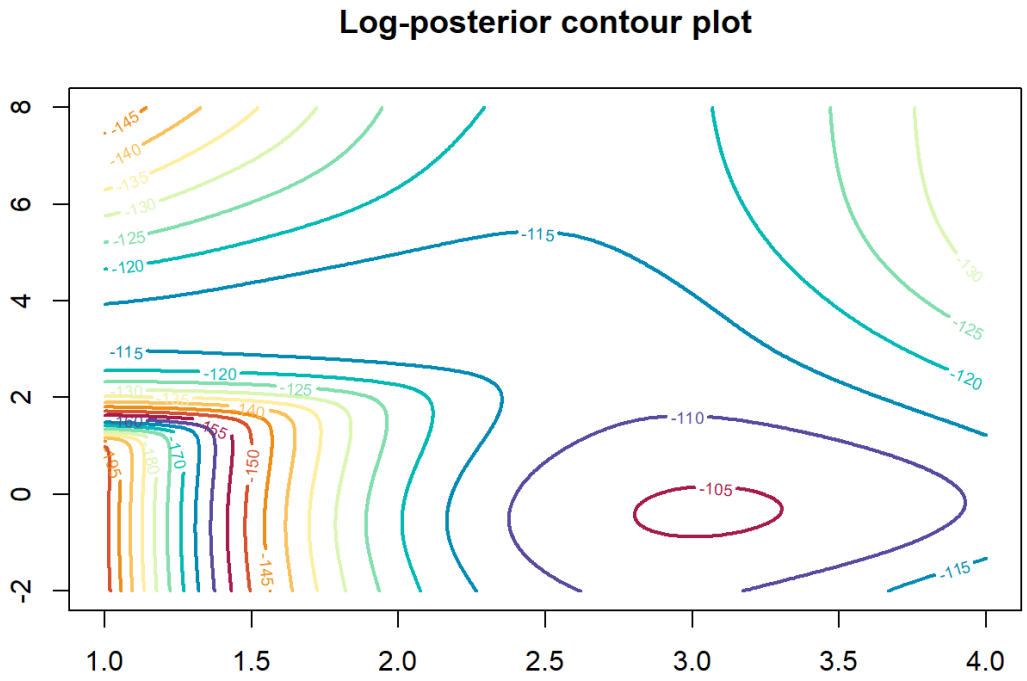
```
log.exponential.mix <- function(theta, y) {
  lambda.A <- exp(theta[1])
  lambda.B <- exp(theta[2])
  sum(log(0.8*dexp(y,1/lambda.A)+(1-0.8)*dexp(y,1/lambda.B)))
}
```

The following lifetimes are observed from a sample of 30 parts:

```
data <- c(0.98, 14.57, 0.08, 0.18, 86.49, 41.79, 0.29, 1.67, 7.08, 32.21, 18.51, 50.38, 36.70, 18.81, 14.89,
0.16, 0.72, 0.77, 10.39, 4.87, 18.64, 1.46, 2.69, 24.60, 39.93, 20.24, 8.69, 0.58, 2.58, 0.91)
```

- Construct a contour plot of (θ_A, θ_B) over the rectangle $(1, 4) \times (-2, 8)$

```
theta.A <- seq(1, 4, 0.01)
theta.B <- seq(-2, 8, 0.01)

z <- matrix(nrow=length(theta.A), ncol=length(theta.B))
for(i in 1:length(theta.A)){
  for(j in 1:length(theta.B))
    z[i,j] <- log.exponential.mix(theta = c(theta.A[i], theta.B[j]), y = data)
}

cols <- hcl.colors(10, "Spectral")
contour(theta.A, theta.B, z, col=cols, lwd=2, main="Log-posterior contour plot", nlevels=15)
```



- Using the function `optim` search for the posterior mode with a starting guess of $(\theta_A, \theta_B) = (3, 0)$.

```
laplace1 <- optim(par=c(3, 0), fn=log.exponential.mix, gr=NULL, hessian=TRUE, control=list(fnscale=-1), y=dat
a)
mode1 <- laplace1$par
mode1
```

```
## [1]  3.0424570 -0.3620477
```

- Search for the posterior mode with a starting guess $(\theta_A, \theta_B) = (2, 4)$.

```
laplace2 <- optim(par=c(2, 4), fn=log.exponential.mix, hessian=TRUE, control=list(fnscale=-1), y=data)
mode2 <- laplace2$par
mode2
```

```
## [1] 2.254041 3.566703
```

- Explain why you obtain different estimates of the posterior mode in the previous two points.

Due to the complex nature of mixture model, the density of the posterior can have multiple local maxima. We obtained two different estimates because the two starting points are close to different local maxima and the algorithm converges to the closest one. That is verified by the fact that the hessian computed in the two estimates are negative definite.

From the contour plot we can guess that the estimate obtained with initial guess $(3, 0)$ is better. We can verify it by comparing the value of the function in the two points.

```
log.exponential.mix(mode1, y=data) > log.exponential.mix(mode2, y=data)
```

```
## [1] TRUE
```

- Use a normal approximation to construct a random walk Metropolis chain for sampling the posterior of $\theta = (\log(\lambda_A), \log(\lambda_B))$. Run the chain for 10000 iterations, and construct density estimates for $\log(\lambda_A)$ and $\log(\lambda_B)$.

As covariance matrix for our normal approximation we use the one given by Laplace approximation, that is

```
V <- - solve(laplace1$hessian)
```

Here is the implementation of the random walk Metropolis chain.

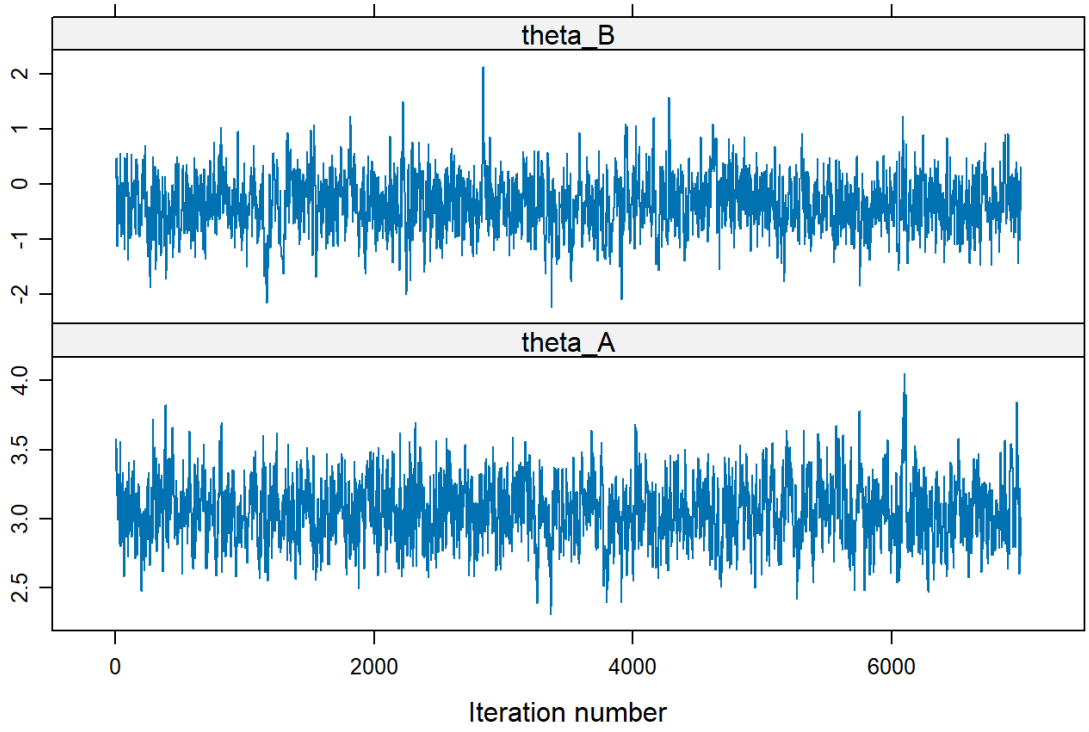
```
library(mvtnorm)

Metropolis <- function(logpost, start, data, V, n.iter, burn.in, n.thin=1) {
  f0 <- logpost(start, data)
  theta <- matrix(NA, nrow=n.iter, ncol=length(start))
  theta[1,] <- start
  accept <- 0
  for (i in 2:n.iter) {
    proposal <- rmvnorm(1, theta[i-1,], V)
    f1 <- logpost(proposal, data)
    r <- f1 - f0
    if (log(runif(1)) < r) {
      theta[i,] <- proposal
      accept <- accept + 1
      f0 <- f1
   } else {
      theta[i,] <- theta[i-1,]
   }
 }
 bin <- theta[(1:burn.in),]
 theta <- theta[seq((burn.in+1), n.iter, n.thin), ]
 result <- list(theta=theta, accept=accept, burn.in=bin)
 return(result)
}
```

We can use it to get a sample from the posterior distribution.

```
system.time(
  result1 <- Metropolis(n.iter=10000, start=mode1, logpost=log.exponential.mix, V=V, data=data, burn.in=3000)
  )
```

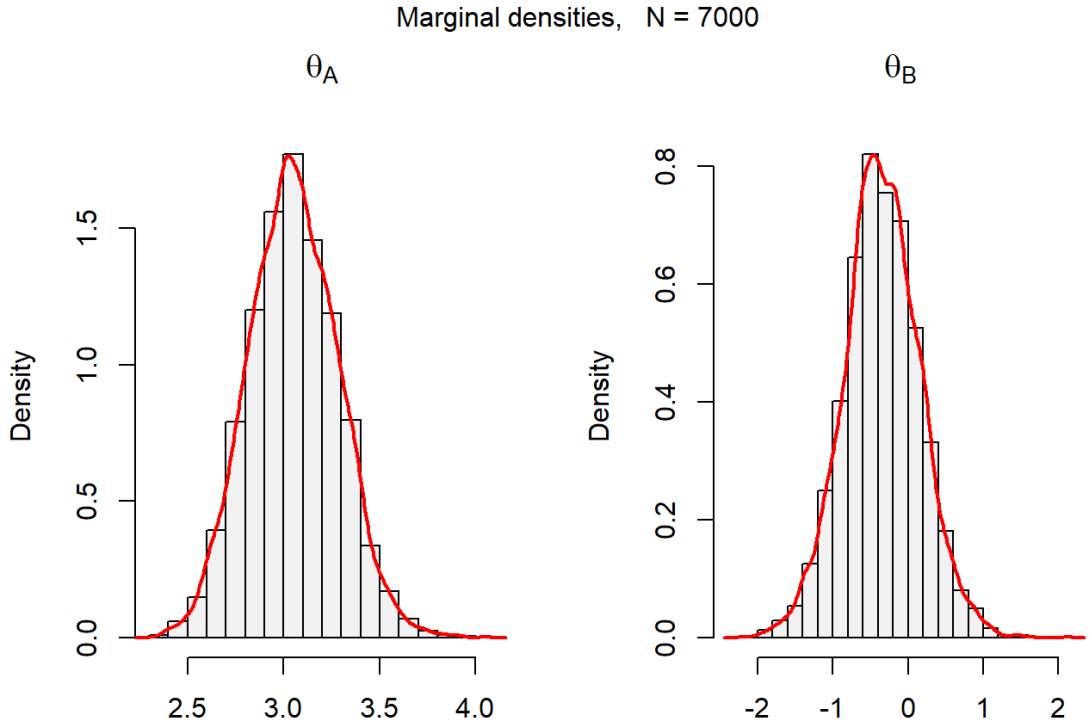
```
##    utente   sistema trascorso
##      2.08      0.01      2.53
```

From the trace plot we can see that the chain behaves quite well.



We can use the sample to approximate the densities of the marginals.

```
# Helper function to plot the marginals' densitiy
plot.marginals <- function(sample, names, mfrow){
  k <- dim(sample)[2]
  par(mfrow = mfrow)
  for (i in 1:k){
    hist(sample[,i], breaks = 20, main=names[i], xlab = "", freq=FALSE, col='gray95')
    lines(density(sample[,i]), col = "red", lwd=2)
  }
  mtext(sprintf("Marginal densities,   N = %s", length(sample[,1])), side = 3, line = -1, outer = TRUE)
}

names <- c(TeX(r"($\theta_A$)"), TeX(r"($\theta_B$)"))
plot.marginals(result1$theta, names, c(1, 2))
```



- Construct a Metropolis within Gibbs samples, i.e., use a Metropolis algorithm to sample from $\log(\lambda_B)|\log(\lambda_A)$ and then do the viceversa. Also run the chain for 10000 iterations and construct density estimates for $\log(\lambda_A)$ and $\log(\lambda_B)$.

We start by constructing the Metropolis algorithm to sample from $\theta_B|\theta_A$. As proposal distribution we can take the conditional distribution of the bivariate normal we used in the previous point. Indeed it can be proved that, if

$$(\theta_A, \theta_B) \sim \mathcal{N}\left((\mu_A, \mu_B), \begin{bmatrix} \sigma_A & \sigma_{AB} \\ \sigma_{AB} & \sigma_B \end{bmatrix}\right),$$

then

$$\theta_A|(\theta_B = t) \sim \mathcal{N}\left(\mu_A + \frac{\sigma_A}{\sigma_B}\rho(t - \mu_B),\ (1-\rho^2)\sigma_A^2\right)$$

where $\rho = \frac{\sigma_{AB}}{\sigma_A \sigma_B}$ is the correlation coefficient.

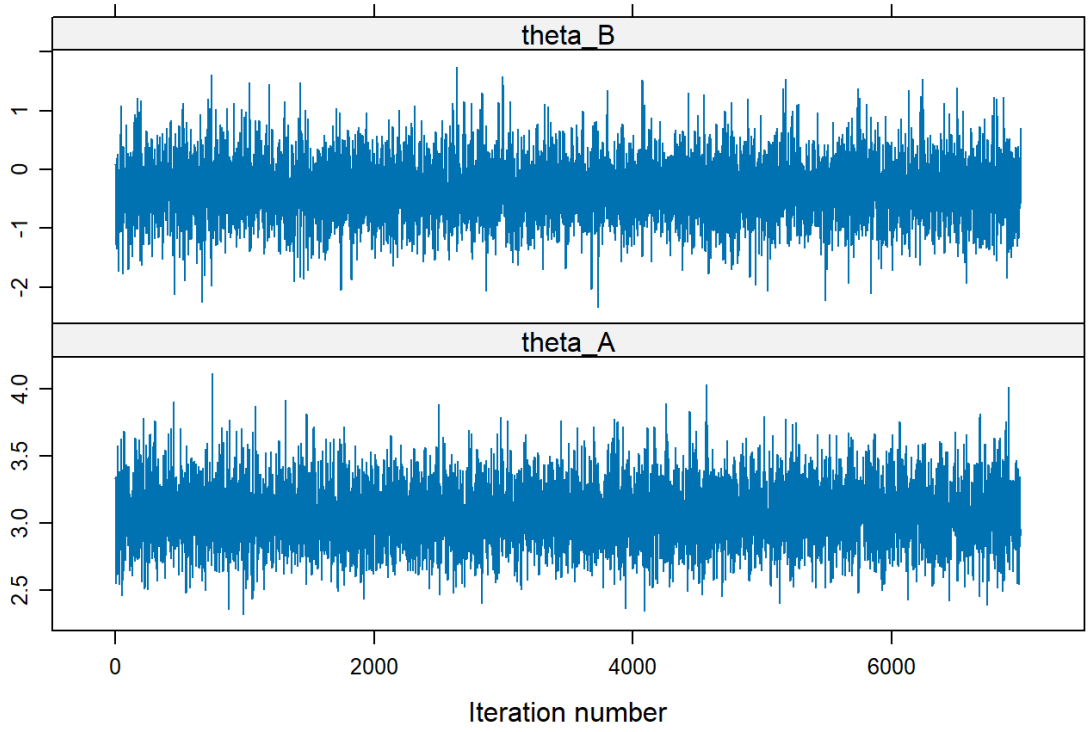
Knowing that, we have everything sample from the two conditional distribution with Metropolis sampler. I decided to use the independent version of the algorithm because it seems to work better.

```
GibbsMetr <- function(n.iter, start, data, burn.in, Metr.burn.in=100){
  rho <- V[1, 2] / (sqrt(V[1,1] * V[2, 2]))
  sd.AgivB <- sqrt((1 - rho**2) * V[1, 1])
  sd.BgivA <- sqrt((1 - rho**2) * V[2, 2])

  indepMetr <- function(start, givenA, t, mu, sd,
                        logpost=log.exponential.mix, burn.in=Metr.burn.in) {
    # givenA has to be a boolean that tells if we want to sample from A|B or B|A
    if(givenA) aux <- c(start, t) else aux <- c(t, start)
    f0 <- logpost(aux, data)
    g0 <- dnorm(start, mu, sd, log=TRUE)

    theta <- rep(NA, burn.in+1)
    theta[1] <- start
    for (i in 2:(burn.in+1)) {
      proposal <- rnorm(1, mu, sd)
      g1 <- dnorm(proposal, mu, sd, log=TRUE)
      if(givenA) aux <- c(proposal, t) else aux <- c(t, proposal)
      f1 <- logpost(aux, data)
      r <- f1 - f0 + (g0-g1)
      if (log(runif(1)) < r) {
        theta[i] <- proposal
        f0 <- f1
        g0 <- g1
     } else {
        theta[i] <- theta[i-1]
      }
   }
   return(theta[length(theta)])
  }

  theta <- matrix(NA, ncol=2, nrow = n.iter)
  theta[1,] <- start
  for(i in 2:n.iter){
    #theta.B | (theta.A=t.A)
    t.A <- theta[i-1, 1]
    mu.BgivA <- mode1[2] + V[1, 1]*rho*(t.A-mode1[1])/V[2,2]
    theta[i, 2] <- indepMetr(start=theta[i-1,2], givenA=FALSE, t=t.A, mu=mu.BgivA, sd=sd.BgivA)

    #theta.A | (theta.B=t.B)
    t.B <- theta[i, 2]
    mu.AgivB <- mode1[1] + V[1, 1]*rho*(t.B-mode1[2])/V[2,2]
    theta[i, 1] <- indepMetr(start=theta[i-1,1], givenA=TRUE, t=t.B, mu=mu.AgivB, sd=sd.AgivB)
  }

  bin <- theta[(1:burn.in),]
  theta <- theta[-(1:burn.in),]
  result <- list(theta=theta, burn.in=bin)
  return(result)
}
```

```
system.time(
  result1G <- GibbsMetr(n.iter=10000, start=mode1, data=data, burn.in=3000)
  )
```

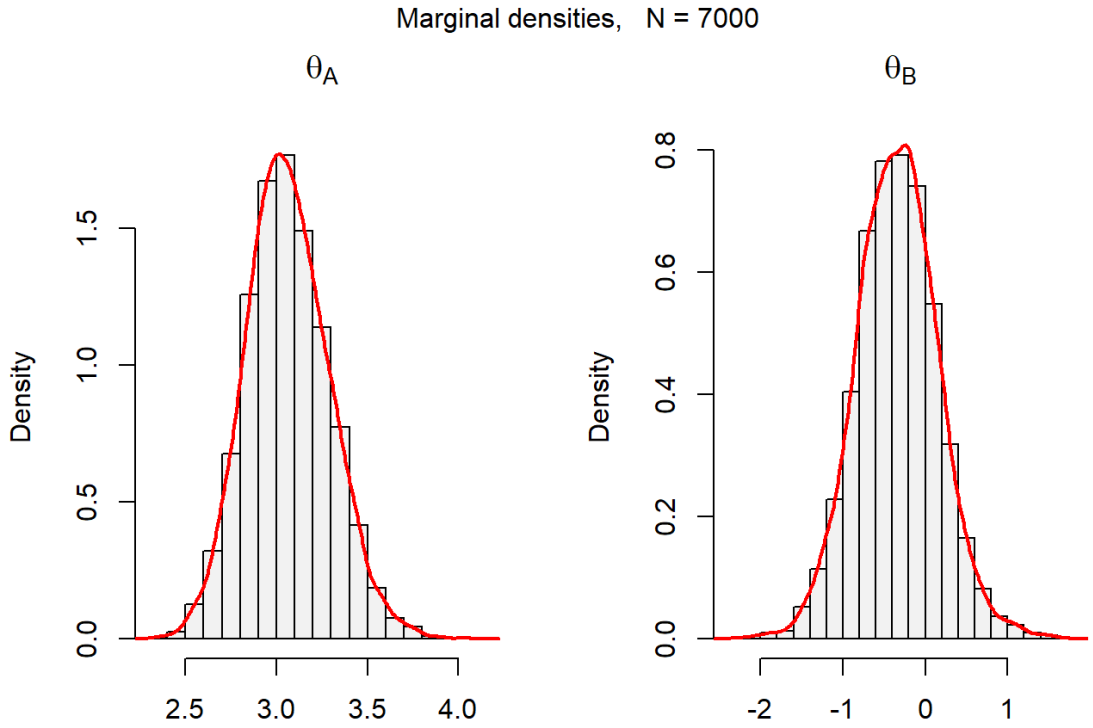
```
##    utente   sistema trascorso
##     24.52      0.04     25.83
```

Also in this case the chain behaves well.



As before, we approximate the density of the marginals.

```
plot.marginals(result1G$theta, names, c(1, 2))
```



Exercise 2: Birthweight regression

Dobson (2001) describes a birthweight regression study. One is interested in predicting a baby's birthweight (in grams) based on the gestational age (in weeks) and the gender of the baby. The data are available as `birthweight` in the `LearnBayes` R package. In the standard linear regression model, we assume that

$$BIRTHWEIGHT_i = \beta_0 + \beta_1 AGE_i + \beta_2 GENDER_i + \epsilon_i$$

- Use the R function `lm` to fit this model by least-squares. From the output, assess if the effects `AGE` and `GENDER` are significant, and if they are significant, describe the effects of each covariate on `BIRTHWEIGHT`.

First of all we have to load the data.

```
library(LearnBayes)
data <- LearnBayes::birthweight
str(data)
```

```
## 'data.frame':    24 obs. of  3 variables:
##  $ age   : int  40 38 40 35 36 37 41 40 37 38 ...
##  $ gender: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ weight: int  2968 2795 3163 2925 2625 2847 3292 3473 2628 3176 ...
```

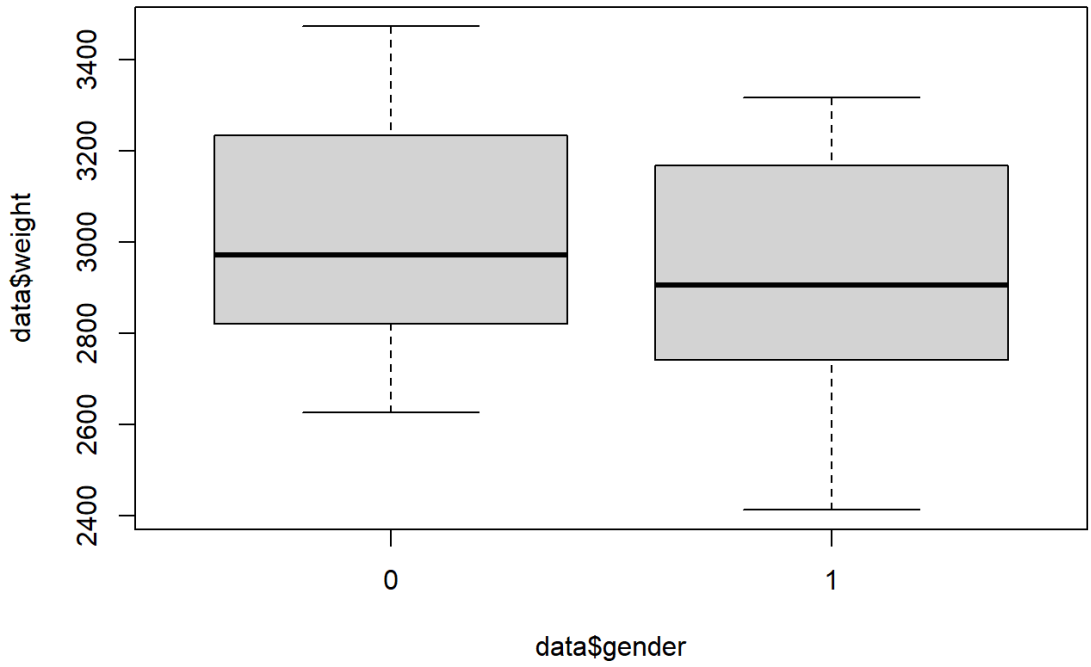
We fit the model by least-squares.

```
linear_model <- lm(weight ~ age + gender, data = data)
summary(linear_model)
```

```
##
## Call:
## lm(formula = weight ~ age + gender, data = data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -258.65 -155.48  -36.17  164.00  463.62
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -924.94     896.09  -1.032  0.31373
## age           103.02      23.32   4.417  0.00024 ***
## gender       -121.25      84.97  -1.427  0.16827
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 208.1 on 21 degrees of freedom
## Multiple R-squared:  0.5032, Adjusted R-squared:  0.4558
## F-statistic: 10.63 on 2 and 21 DF,  p-value: 0.0006461
```

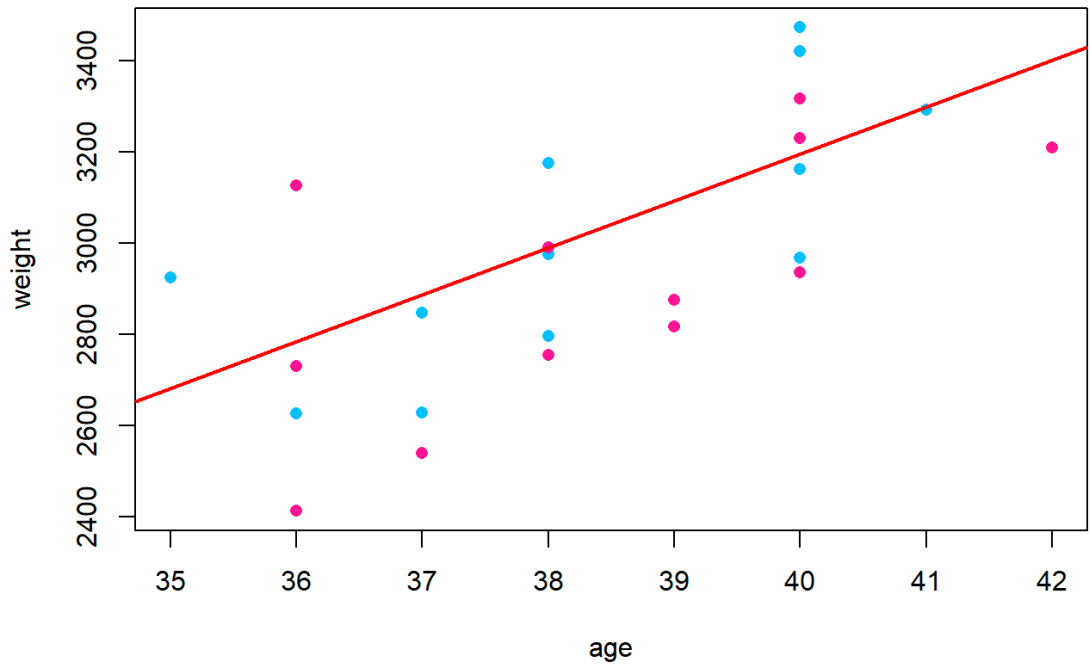
From the p-values, we can see that, in our model, the effect of age is significant while the effect of gender is not very significant. We can use a boxplot to visualize the fact that weight and gender are not strongly related

```
boxplot(data$weight ~ data$gender)
```



We can visualize our linear model in the weight-age plane.

```
col_vec <- c(rep('deepskyblue', sum(data$gender==0)),
             rep('deeppink', sum(data$gender==1)))
plot(weight ~ age, data = data, pch=16, col = col_vec)
abline(a= linear_model$coefficients[1], b= linear_model$coefficients[2], col = 'red', lwd = 2)
```



- Suppose a noninformative Zellner's g prior (https://en.wikipedia.org/wiki/G-prior) is placed on the regression parameter vector $\beta = (\beta_0, \beta_1, \beta_2)$ and assume an Inverse-Gamma prior for σ^2.

We are considering the model

$$y_i = x_i^\top \beta + \epsilon_i$$

where ϵ_i are i.i.d normal of mean 0 and variance σ^2. Let X be the matrix with rows equal to $x_i^\top$. The prior distributions for β and σ^2 are:

- $\beta|\sigma^2 \sim \mathcal{N}(\bar{\beta}, g\sigma^2(X^\top X)^{-1})$, where $g > 0,\ \bar{\beta}$ are hyper parameters and $\sigma^2(X^\top X)^{-1}$ is the inverse of Fisher information matrix;
- $\sigma^2 \sim \text{InverseGamma}(a, b)$.

If we indicate with $\hat{\beta}$ the ML estimator of β and we define $q = g/(1+g)$, then the posterior distribution for β is

$$\beta \,|\, \sigma^2,\, y \sim \mathcal{N}(q\hat{\beta} + (1-q)\bar{\beta},\ q\sigma^2(X^\top X)^{-1})$$

We are interested also in the joint posterior distribution of (β, σ^2):

$$f(\beta, \sigma^2 \,|\, y) \propto f(y \,|\, \beta,\, \sigma^2) f(\beta \,|\, \sigma^2) f(\sigma^2)$$

The only term that we don't already know is the likelihood of the linear model $f(y \,|\, \beta,\, \sigma^2)$. To compute it we notice that $y_i|(\beta,\, \sigma^2) \sim \mathcal{N}(x_i^\top \beta,\, \sigma^2)$, and so we have that

$$f(y|\beta,\, \sigma^2) = \prod_{i=1}^{n} (2\sigma^2)^{-1} \exp\left(-\frac{1}{2\sigma^2}(y_i - x_i^\top \beta)^2\right) \propto (\sigma^2)^{-\frac{n}{2}} \exp\left(-\frac{1}{2\sigma^2}\sum_{i=0}^{n}(y_i - x_i^\top \beta)^2\right) \propto (\sigma^2)^{-\frac{n}{2}} \exp\left(-\frac{r^\top r}{2\sigma^2}\right)$$

where $r = (y_1 - x_1^\top \beta,\, \ldots,\, y_n - x_n^\top \beta)$ is the vector of the residuals.

Here is the implementation of a function that computes the log of the posterior $f(y|\beta,\, \sigma^2)$.

```
library(mvtnorm)
library(invgamma)

n <- length(data$age)
X <- cbind(rep(1, n), data$age, data$gender)
y <- data$weight
data4post <- list(n=n, y=y, X=X, fisher.aux=solve(t(X)%*%X))

log.posterior <- function(theta, data, params){
  # theta = (beta, sigma2)
  beta <- theta[1:3]
  sigma2 <- theta[4]

  fisher.inv <- sigma2 * data$fisher.aux

  log.yGIVsigma2beta <-dmvnorm(data$y, mean=data$X %*% beta, sigma=diag(sigma2, data$n), log= TRUE)
  log.betaGIVsigma2 <- dmvnorm(beta, mean=params$beta.bar, sigma= params$g * fisher.inv, log= TRUE)
  log.sigma2 <- dinvgamma(sigma2, shape=params$a, scale=params$b, log = TRUE)

  return(log.yGIVsigma2beta + log.betaGIVsigma2 + log.sigma2)
}
```

- Simulate a sample of 5000 draws from the joint posterior distribution of (β, σ). Explore different values of the prior parameters. To help you on doing this point you might want to read this slides (./bayes-varsel.pdf)

We can sample from the posterior distribution of (β, σ) using Metropolis algorithm. As proposal distribution we use the multivariate normal given by the Laplace approximation. As usual, the mean and the covariance matrix for this approximation can be computed with the function `optim`. As initial guess we consider the ML estimators: for β it is the vector of coefficients of the linear model, for σ^2 it is the sample variance of the residuals $\hat{\sigma}^2 = \frac{r^\top r}{n}$.

We also need to fix the hyper parameters. We have no prior knowledge on β and σ^2, thus it is natural to set $\bar{\beta} = (0, 0, 0)$ and to choose the other parameters in such a way that the two prior distribution have high variance. This means to set a small and $g,\ b$ large, because

$$Var(\sigma^2) = \frac{b}{(a-1)^2(a-2)} \qquad \text{and} \qquad Var(\beta) \propto g$$

.

I explored different values for the hyper parameters and a reasonable choice is for example $g = 1000,\ a = 3,\ b = 100$.

```
# Initial guess
beta.hat <- unname(linear_model$coefficients)
res <- linear_model$residuals
sigma2.hat <- sum(res^2) / n

theta.start <- c(beta.hat, sigma2.hat)

# Hyperparameters
params <- list(beta.bar=c(0, 0, 0), g=1000, a=5, b=100)

# Proposal parameters
laplace <- optim(theta.start, fn=log.posterior, gr=NULL, hessian=TRUE, control=list(fnscale=-1),
                 params=params, data=data4post)
mode <- laplace$par
V <- - solve(laplace$hessian)
mode
```

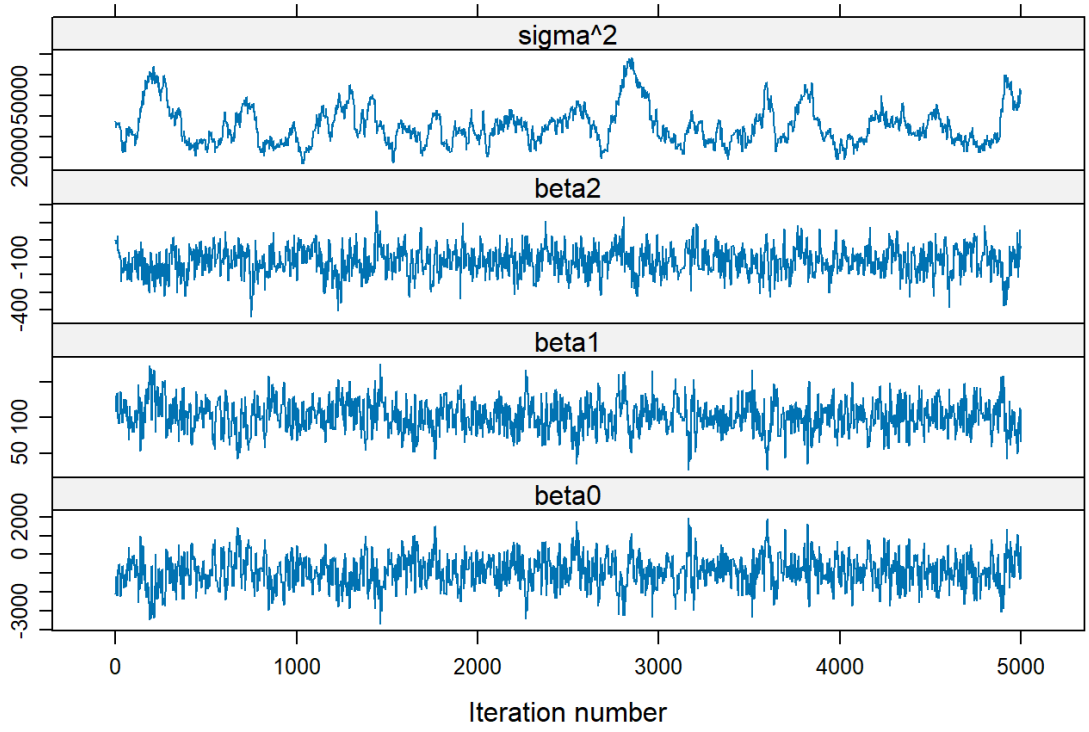
```
## [1]  -923.8983   102.9085  -121.0983 28744.9037
```

We can use the `Metropolis` function that we already implemented.

```
N <- 5000 # Sample size
burn.in <- 3000

system.time(
  result2 <- Metropolis(logpost=function(theta, data) log.posterior(theta, data, params=params),
                        start=mode, data=data4post, V=V, n.iter=N+burn.in, burn.in=burn.in)
  )
```

```
##    utente   sistema trascorso
##      5.26      0.04      5.89
```

Looking at the trace plots we can see that the chain moves slowly, in particular σ^2.



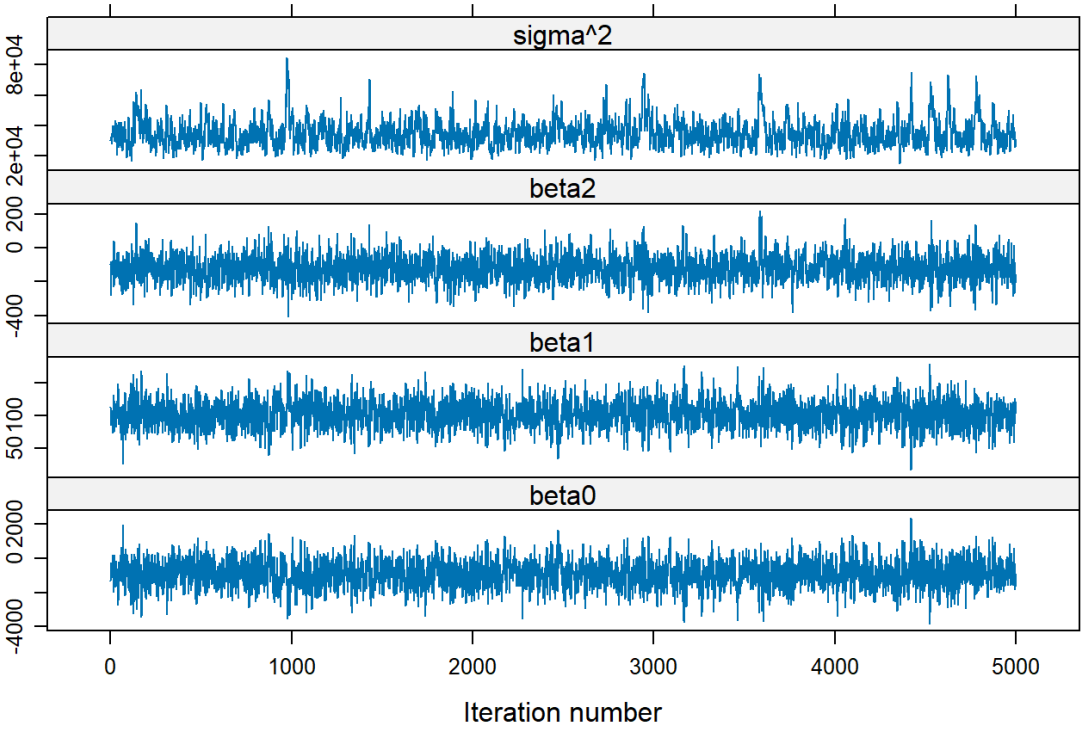
To mitigate this problem, we can make the proposal distribution a little bit wider. To do it we take a diagonal matrix D (with elements on the diagonal bigger or equal than 1) and we consider the new covariance matrix $\tilde{V} = DVD$.

Moreover, we thin the chain by increasing the `n.thin` parameter. Indeed, the algorithm obtains a new sample every `n.thin` iterations of the chain. The price to pay is that the computational time increases, because, in order to have `N` samples, we have to perform `n.iter = n.thin * N + burn.in` iterations.

```
n.thin <- 5
D <- diag(c(1, 1, 1, 2))
V.tilde <- D %*% V %*% D
system.time(result2 <- Metropolis(logpost=function(theta, data) log.posterior(theta, data, params=params),
                     start=mode, data=data4post, V=V.tilde,
                     n.iter=burn.in+n.thin*5000, burn.in=burn.in, n.thin=n.thin)
            )
```

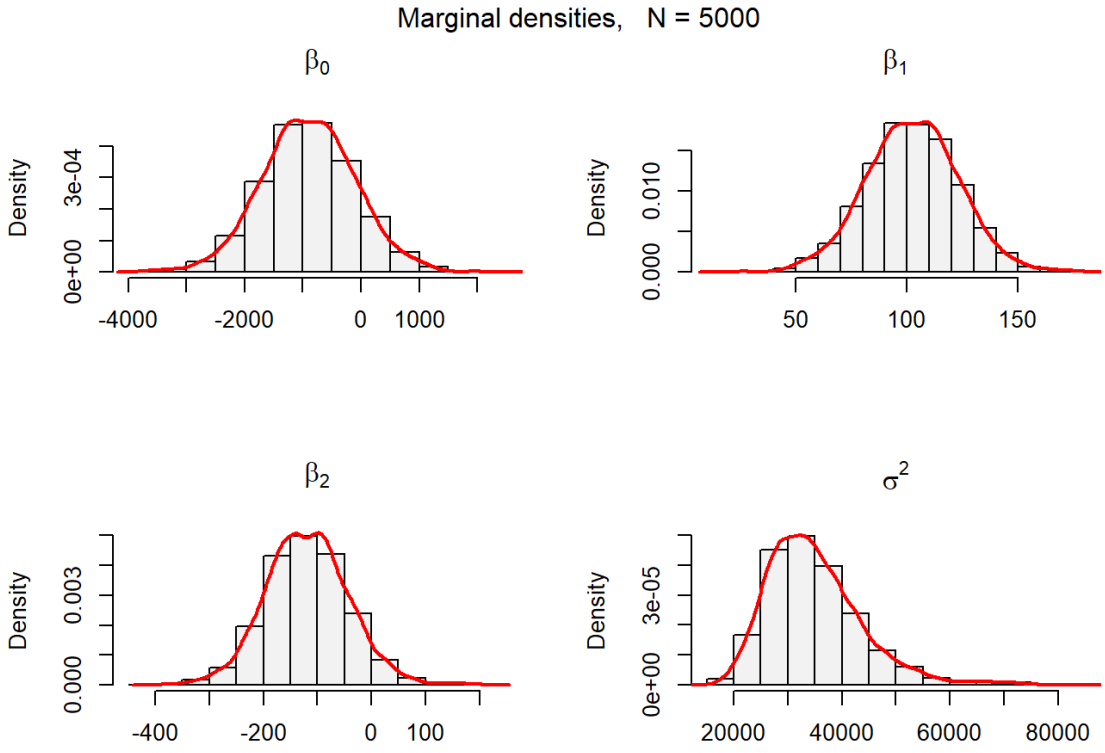
```
##    utente   sistema trascorso
##     19.19      0.14     20.37
```

```
my.sample <- result2$theta
```



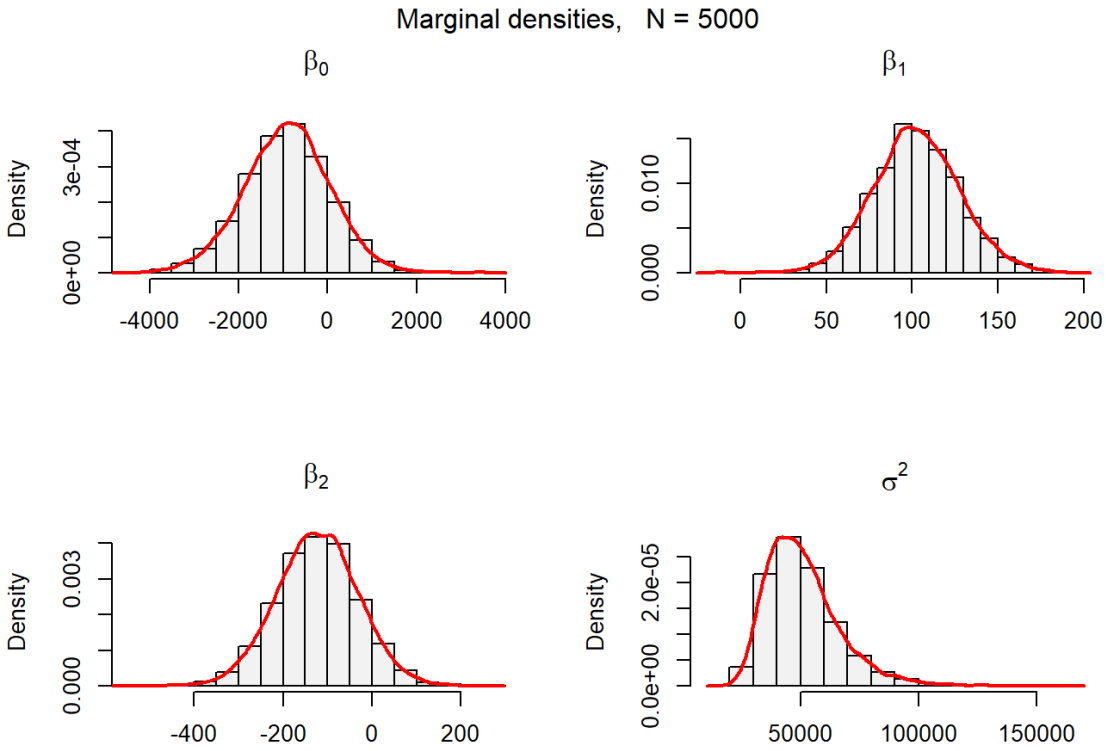
We can use the sample to approximate the densities of the four marginals.

```
names <- c(TeX(r"($\beta_0$)"), TeX(r"($\beta_1$)"), TeX(r"($\beta_2$)"), TeX(r"($\sigma^2$)"))
plot.marginals(my.sample, names, c(2, 2))
```



- Use the function `blinreg` in package `LearnBayes` to redo the simulation and compare the results.

```
package.sample <- blinreg(y = y, X = X, m = N, prior = list(c0 = 1000, b0 = c(0,0,0)))

plot.marginals(cbind(package.sample$beta, package.sample$sigma**2), names, c(2, 2))
```



We can see that our estimates are similar to the estimates obtained with `blinreg` .

- From the simulated samples, compute the posterior means and standard deviations of β_1 and β_2.

Using the samples simulated with `Metropolis` we have that:

- $\mathbb{E}[\beta_1 \mid y] = 102.38$ and $sd[\beta_1 \mid y] = 20.79$
- $\mathbb{E}[\beta_2 \mid y] = -119.22$ and $sd[\beta_2 \mid y] = 76.82$

- Check the consistency of the posterior means and standard deviations with the least-squares estimates and associated standard errors from the `lm` run.

```
means <- colMeans(my.sample)
MLE <- theta.start
mean_table <- cbind(means, MLE, abs(means-MLE))
rownames(mean_table) <- c('beta0', 'beta1', 'beta2', 'sigma^2 ')
colnames(mean_table) <- c('sample mean', 'MLE', 'abs error')

print(mean_table)
```

```
##          sample mean        MLE    abs error
## beta0      -903.9656  -924.9354   20.9698118
## beta1       102.3764   103.0157    0.6393037
## beta2      -119.2242  -121.2513    2.0271145
## sigma^2   34875.9491 37881.3985 3005.4493528
```

We can see that the posterior means and the MLE estimates are quite similar. Only the two values for σ^2 are not very close to each other. Also the standard deviation and the standard errors are quite similar.

```
std_dev <- apply(my.sample[,1:3], 2, sd)
std_errors <- c(sqrt(diag(vcov(linear_model))))
sd_table <- cbind(std_dev, std_errors, abs(std_dev-std_errors))
rownames(sd_table) <- c('beta0', 'beta1', 'beta2')
colnames(sd_table) <- c('sample sd', 'se lm', 'error')

print(sd_table)
```

```
##       sample sd     se lm     error
## beta0 798.83309 896.09144 97.258349
## beta1  20.79104  23.32373  2.532684
## beta2  76.81935  84.96636  8.147011
```

- Suppose one is interested in estimating the expected `birthweight` for `male` and `female` babies of gestational weeks `36` and `40`. From the simulated draws of the posterior distribution construct 90% interval estimates for 36-week males, 36-week females, 40-week males, and 40-week females.

In the classical framework, the expected `birthweight` given x_i is $\mathbb{E}[y_i] = \mathbb{E}[x_i\beta + \epsilon_i] = x_i\beta$. In the Bayesian framework this quantity is a random variable and we can compute a confidence interval for it using the posterior distribution.

```
X1 <- cbind(1, c(36,36,40,40), c(0,1,0,1)) #X for which we want to sample y

# E[y_i] = x_i * beta
my.sample.exp_Y <- X1%*%t(my.sample[,1:3]) # dim = (4, 5000)
rownames(my.sample.exp_Y) <- c('(36, 0)', '(36, 1)', '(40, 0)', '(40, 1) ')

# For each row of my.sample.exp_y we compute the extremes of the 90% confidence interval
apply(my.sample.exp_Y, 1, function(x) return(quantile(x, probs= c(0.05, 0.95)) ))
```

```
##      (36, 0)  (36, 1)  (40, 0) (40, 1)
## 5%  2663.513 2541.564 3088.473 2967.247
## 95% 2897.948 2785.268 3294.948 3177.302
```

- Compare the results you obtained with those you can have from function `blinregexpected`.

We can do the same using the functions of the package `LearnBayes`.

```
package.sample_exp_y <- t(blinregexpected(X1, package.sample)) # dim = (4,5000)
rownames(package.sample_exp_y) <- c('(36, 0)', '(36, 1)', '(40, 0)', '(40, 1) ')

apply(package.sample_exp_y, 1, function(x) return(quantile(x, probs= c(0.05, 0.95)) ))
```

```
##      (36, 0)  (36, 1)  (40, 0) (40, 1)
## 5%  2644.564 2509.198 3068.220 2945.330
## 95% 2924.970 2802.993 3317.365 3193.348
```

These confidence intervals are similar to the ones computed with our estimates.

- Suppose instead that one wishes to predict the birthweight for a 36- week male, a 36-week female, a 40-week male, and a 40-week female. Use the simulated data from the posterior to construct 90% prediction intervals for the birthweight for each type of baby.

```
# y_i = x_i * beta + e_i
my.sample_y <- X1%*%t(my.sample[,1:3]) + rnorm(N, mean=0, sd=sqrt(my.sample[,4]))  # dim = (4, 5000)
rownames(my.sample_y) <- c('(36, 0)', '(36, 1)', '(40, 0)', '(40, 1) ')

apply(my.sample_y, 1, function(x) return(quantile(x, probs= c(0.05, 0.95)) ))
```

```
##      (36, 0)  (36, 1)  (40, 0) (40, 1)
## 5%  2448.489 2320.292 2872.899 2727.469
## 95% 3111.434 2998.001 3537.341 3378.311
```

- Compare the results you obtained with those you can have from function `blinregpred` .

```
package.sample_y <- t(blinregexpected(X1, package.sample) + rnorm(N, mean=0, sd=package.sample$sigma))
rownames(package.sample_y) <- c('(36, 0)', '(36, 1)', '(40, 0)', '(40, 1) ')

apply(package.sample_y, 1, function(x) return(quantile(x, probs= c(0.05, 0.95)) ))
```

```
##      (36, 0)  (36, 1)  (40, 0) (40, 1)
## 5%  2378.032 2264.392 2795.628 2681.840
## 95% 3183.220 3062.280 3595.423 3471.798
```

Also in this case the CI are quite similar.

Exercise 3: Athlete

For a given professional athlete, his or her performance level will tend to increase until midcareer and then deteriorate until retirement. Let y_i denote the number of home runs hit by the professional baseball player Mike Schmidt in n_i at-bats (opportunities) during the i-th season. #The datafile `schmidt` in `LearnBayes` package gives Schmidt's age, y_i, and n_i for all 18 years of his baseball career. If y_i s assumed #to be $binomial(n_i, p_i)$, where p_i denotes the probability of hitting a home run during the i-th season, then a reasonable model or the #p_i is the logit quadratic model of the form:

$$\log\left(\frac{p_i}{1-p_i}\right) = \beta_0 + \beta_1 AGE_i + \beta_2 AGE_i^2$$

where AGE_i is Schmidt's age during the ith session. Assume that the regression vector $\boldsymbol{\beta} = (\beta_0, \beta_1, \beta_2)^\top$ has a uniform non-informative prior.

- Write a short R function to compute the logarithm of the posterior density of β.

First of all we import the data.

```
data <- LearnBayes::schmidt
str(data)
```

```
## 'data.frame':    18 obs. of  14 variables:
##  $ Year: int  1972 1973 1974 1975 1976 1977 1978 1979 1980 1981 ...
##  $ Age : int  22 23 24 25 26 27 28 29 30 31 ...
##  $ G   : int  13 132 162 158 160 154 145 160 150 102 ...
##  $ AB  : int  34 367 568 562 584 544 513 541 548 354 ...
##  $ R   : int  2 43 108 93 112 114 93 109 104 78 ...
##  $ H   : int  7 72 160 140 153 149 129 137 157 112 ...
##  $ X2B : int  0 11 28 34 31 27 27 25 25 19 ...
##  $ X3B : int  0 0 7 3 4 11 2 4 8 2 ...
##  $ HR  : int  1 18 36 38 38 38 21 45 48 31 ...
##  $ RBI : int  3 52 116 95 107 101 78 114 121 91 ...
##  $ SB  : int  0 8 23 29 14 15 19 9 12 12 ...
##  $ CS  : int  0 2 12 12 9 8 6 5 5 4 ...
##  $ BB  : int  5 62 106 101 100 104 91 120 89 73 ...
##  $ SO  : int  15 136 138 180 149 122 103 115 119 71 ...
```

The logarithm of the posterior density of β is

$$\log f(\beta \mid y) \propto \log f(y \mid \beta) + \log f(\beta) \propto \log \prod_{i=1}^{18} \binom{n_i}{y_i} p_i^{y_i} (1-p_i)^{n_i - y_i}$$

$$\propto \sum_{i=1}^{18} y_i \log(p_i) + (n_i - y_i) \log(1-p_i)$$

$$= \sum_{i=1}^{18} y_i \log\left(\frac{p_i}{1-p_i}\right) + n_i \log(1-p_i)$$

$$\propto \sum_{i=1}^{18} y_i h_i(\beta) - n_i \log(1 + \exp(h_i(\beta)))$$

Where $h_i(\beta) = \beta_0 + \beta_1 AGE_i + \beta_2 AGE_i^2$.

```
log.posterior2 <- function(beta, data){
  age <- data$Age
  n <- data$AB # at-bats
  y <- data$HR # number of home runs

  h <- beta[1] + beta[2]*age + beta[3]*age**2

  return( sum(y*h - n*log(1+exp(h))) )
}
```

- Find the laplace approximation of the posterior for β, build an R function to compute it.

As always we use the function `optim` to compute the parameters of the Laplace approximation. We can obtain a good starting value by fitting a linear model to our data.

```
p <- data$HR / data$AB
data$h <- log(p/(1-p))
data$Age2 <- data$Age**2

linear_model <- lm(h ~ Age + Age2, data = data)
linear_model$coefficients
```

```
##  (Intercept)          Age         Age2
## -12.48222240   0.65591666  -0.01074155
```

```
beta.start <- unname(linear_model$coefficients)
laplace <- optim(beta.start, log.posterior2, hessian=TRUE, control=list(fnscale=-1), data = data)
mode <- laplace$par
V <- - solve(laplace$hessian)

mode
```

```
## [1] -12.41400733   0.65589039  -0.01078346
```

We have a problem because the covariance matrix V is not definite positive.

```
print(eigen(V)$values)
```

```
## [1]  5.134072e-03  1.638338e-09 -1.188221e-03
```

To solve this problem we substitute V with a matrix U that is positive definite matrix and is close to V w.r.t L-2 norm.

```
U <- as.matrix(Matrix::nearPD(V)$mat)
Matrix::norm(U - V, type = "2")
```

```
## [1] 0.001188221
```

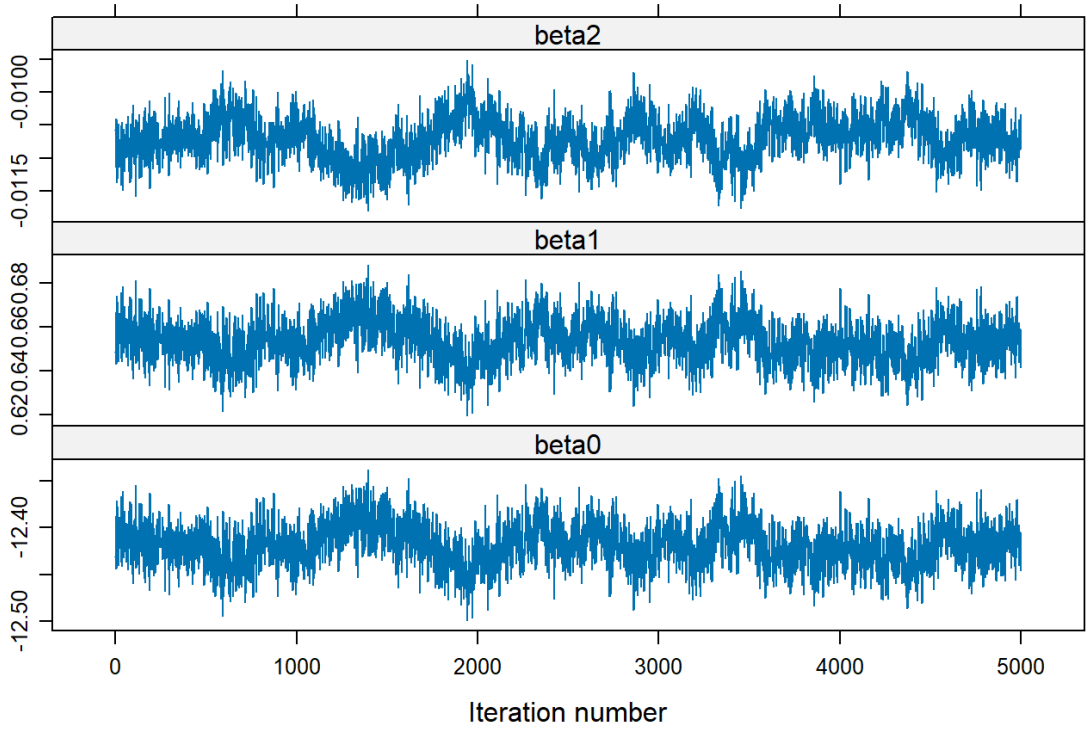
- Build a Metropolis-Hasting using the laplace approximation as proposal to simulate 5000 draws from the posterior distribution of β.

The random walk does not change state frequently enough. To obtain a better sample, as in previous exercise, we thin the chain.

```
N <- 5000
burn.in <- 3000
n.thin <- 4
system.time(
  result3 <- Metropolis(log.posterior2, mode, data, U, n.iter=burn.in+n.thin*N, burn.in=burn.in, n.thin=n.thi
n)
  )
```

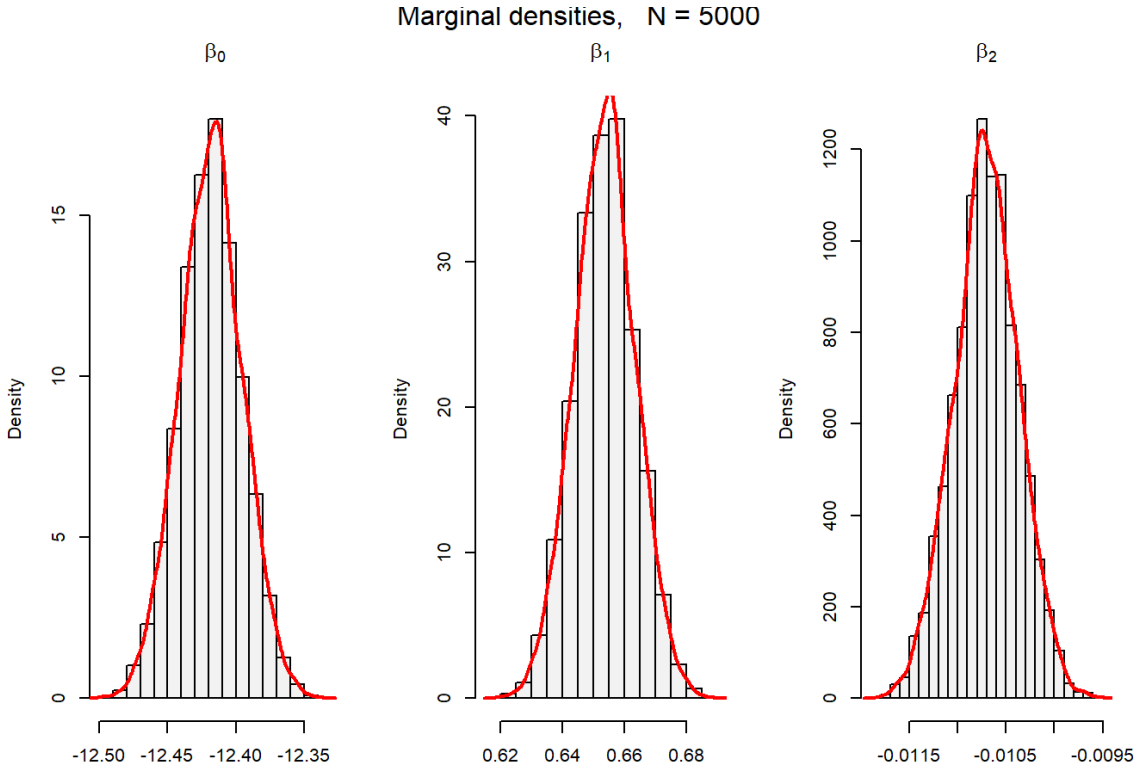
```
##    utente   sistema trascorso
##      5.72      0.09      6.38
```

In this way the chain behaves quite well.



We can use the sample approximate the densities of the marginal distributions.

```
names2 <- c(TeX(r"($\beta_0$)"), TeX(r"($\beta_1$)"), TeX(r"($\beta_2$)"))
plot.marginals(my.sample3, names2, c(1, 3))
```



- One would expect the fitted parabola to have a concave down shape where $\beta_2 < 0$. Use the simulation output from the previous point to find the posterior probability that the fitted curve is concave down.

From the plot of the maginal we can already see that $\mathbb{P}(\beta_2 < 0) \approx 1$. Anyway we compute it:

```
prob <- sum(my.sample3[, 3]<0) / N
prob
```

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

Exercise 4 (Hepatitis): a normal hierarchical model with measurement error

This example is taken from Spiegelhalter et al (1996) (chapter in Markov Chain Monte Carlo in Practice) and concerns $N = 106$ children whose post-vaccination anti Hb titre was measured 2 or 3 times. Both measurements and times have been transformed to a log scale. One covariate y_0= log titre at baseline, is available. The model is essentially a random effects linear growth curve

$$
\begin{aligned}
Y_{ij} &\sim Normal(\alpha_i + \beta_i(t_{ij} - \bar{t}\,) + \gamma(y_{0i} - \bar{y}_0), \tau) \\
\alpha_i &\sim Normal(\alpha_0, \tau_\alpha) \\
\beta_i &\sim Normal(\beta_0, \tau_\beta)
\end{aligned}
$$

where, $\bar{t}$ is the time mean among the observations for which the value of Y_{ij} is available, $\bar{y}_0 = \sum_{i=1}^{N} y_{0i}/N$ and τ represents the precision (1/variance) of a normal distribution. We note the absence of a parameter representing correlation between α_i and β_i unlike in Gelfand et al 1990. α_0, τ_α, β_0, τ_β, τ are given independent ``noninformative'' priors.

The file with the data is available here (./hepatitis.Rdata).

Try to use openBUGS (see eg www.openbugs.net/Examples/Hepatitis.html (http://www.openbugs.net/Examples/Hepatitis.html)) or JAGS (http://mcmc-jags.sourceforge.net/ (http://mcmc-jags.sourceforge.net/)) and their R interfaces to

- Run the chain using the following initial values: $\alpha_0 = 4$, $\beta_0 = 0$, $\gamma = 0$, $\tau_\alpha = 1$, $\tau_\beta = 1$ and $\tau = 1$.
- Run a burn in of 1000 updates followed by a futher 10000 updates.

Since we want non-informative priors, for α_0, β_0, γ we can consider a normal distribution with high variance (10^3), while for τ_α, τ_β, τ (that have to be positive) we can consider a gamma distribution with high variance (10^3).

Given that, we can define our model.

```
library(rjags)
```

```
## Linked to JAGS 4.3.1
```

```
## Loaded modules: basemod,bugs
```

```
library(R2jags)
```

```
##
## Caricamento pacchetto: 'R2jags'
```

```
## Il seguente oggetto è mascherato da 'package:coda':
##
##     traceplot
```

```
# Define the model in BUGS syntax.
model <- "model{
  # Likelihood
  for( i in 1:N) {
     alpha[i] ~ dnorm(alpha0,tau.alpha)
     beta[i] ~ dnorm(beta0,tau.beta)
     for( j in 1:3) {
        mu[i , j] <- alpha[i] + beta[i] * (t[i,j] - t.bar) + gamma * (y0[i] - y.bar)
        y[i , j] ~ dnorm(mu[i , j],tau)
     }
  }

  # Priors
  tau ~ dgamma(0.001,0.001)
  alpha0 ~ dnorm(0.0,1.0E-3)
  tau.alpha ~ dgamma(0.001,0.001)
  beta0 ~ dnorm(0.0,1.0E-3)
  tau.beta ~ dgamma(0.001,0.001)
  gamma ~ dnorm(0.0, 1.0E-3)
  }"

# Load the data
load("C:/Users/davib/OneDrive/Documents/hepatitis.Rdata")
y0 <- hepatitis$y0
t <- hepatitis$t
y <- hepatitis$Y
N <- 106
y.bar <- sum(y0) / N
t.bar <- mean(t[is.na(y) == FALSE])

# Set initial values
inits <- function (){
  list (alpha0=4, beta0=0, gamma=0, tau.alpha=1, tau.beta=1, tau=1)
}

data <- list(y = y, t = t, N = 106, y0 = y0, t.bar= t.bar, y.bar = y.bar)
parameters <- c("alpha","beta", "tau", "gamma")
```

Now we have everything to run the chain.

```
N <- 10000
burn.in <- 1000
n.thin <- 2

jagsfit <- jags(
  data = data,
  inits = inits,
  parameters.to.save = parameters,
  model.file = textConnection(model),
  n.burnin = burn.in,
  n.iter = n.thin * N + burn.in,
  n.chains = 1,
  n.thin = n.thin
)
```

```
## module glm loaded
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 288
##    Unobserved stochastic nodes: 248
##    Total graph size: 1985
##
## Initializing model
```

- Construct a posterior confidence intervals of size 0.95 for α_i and β_i parameters.

In `jagsfit$BUGSoutput$summary` we already have the quantiles that we need. The confidence interval for the first five α_i are

```
CI.alpha <- jagsfit$BUGSoutput$summary[1:106, c("2.5%", "97.5%")]

t(CI.alpha[1:5, ])
```

```
##       alpha[1] alpha[2] alpha[3] alpha[4] alpha[5]
## 2.5%  5.715097 5.745106 5.722066 5.688318 5.677969
## 97.5% 6.553482 6.577399 6.556333 6.511897 6.516412
```

and for the first five β_i are

```
CI.beta <- jagsfit$BUGSoutput$summary[-(1:106), c("2.5%", "97.5%")]

t(CI.beta[1:5, ])
```

```
##          beta[1]   beta[2]     beta[3]   beta[4]   beta[5]
## 2.5%  -2.8533275 -1.949946 -3.33540887 -3.144694 -2.241996
## 97.5%  0.6257819  1.565703  0.06105375  0.138564  1.118858
```

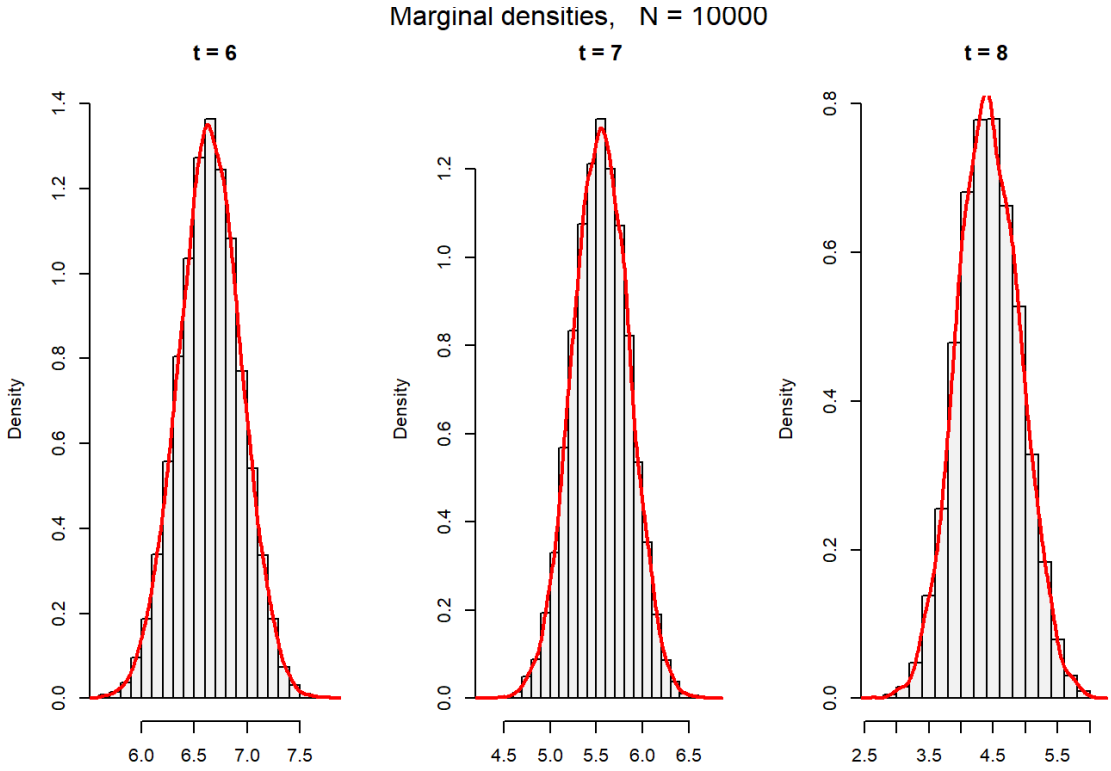
- Sample 10000 values from the posterior predictive distribution of Y for $t = (6, 7, 8)^\top$ and $y_0 = 6.5$.

In order to sample from the posterior predictive of Y, I use the samples from the posterior distributions of the parameters that we computed in the previous point. In particular, for α and β I take the mean over the 106 children.

```
t.pred <- c(6, 7, 8)
y0.pred <- 6.5

sims <- jagsfit$BUGSoutput$sims.list

sample.pred <- matrix(NA, nrow=N, ncol=3)
alpha <- rowMeans(sims$alpha)
beta <- rowMeans(sims$beta)
for (j in 1:3){
  sample.pred[ ,j] <- rnorm(N, alpha + beta*(t.pred[j] - t.bar) + sims$gamma*(y0.pred-y.bar), sims$tau)
}
```

We can plot the marginal distribution of the posterior predictive Y at the three different times.

```
plot.marginals(sample.pred, c("t = 6", "t = 7", "t = 8" ), c(1, 3))
```



- Construct posterior predictive intervals for Y of size 0.95 for the previous case.

```
colnames(sample.pred) <- c("t = 6", "t = 7", "t = 8")
apply(sample.pred, 2, function(x) return(quantile(x, probs= c(0.025, 0.975)) ) )
```

```
##          t = 6    t = 7    t = 8
## 2.5%  6.056408 4.957261 3.487573
## 97.5% 7.227487 6.129942 5.393767
```

In the previous setting we assumed that the baseline y_{0i} was measured without errors. Construct a model where,

$$y_{0i} \sim Normal(\mu_{0i}, \tau)$$
$$\mu_{0i} \sim Normal(\theta, \psi)$$

and a ``non informative'' prior for θ and ψ.

- Redo the previous points with the new model, using initial values $\theta = 6$ and $\psi = 1$.

Reasoning as before, as non-informative priors for μ_{01} and θ we take two normal distributions with high variance, while for ψ we take a gamma distribution with high variance.

```
model2 <- "model{
  # Likelihood
  for( i in 1:N) {
    alpha[i] ~ dnorm(alpha0, tau.alpha)
    beta[i] ~ dnorm(beta0, tau.beta)

    y0[i] ~ dnorm(mu0[i], tau)
    mu0[i] ~ dnorm(theta, psi)

    for(j in 1:3) {
       mu[i, j] <- alpha[i] + beta[i] * (t[i, j] - t.bar) + gamma * (y0[i] - y.bar)
       y[i , j] ~ dnorm(mu[i, j], tau)
      }
  }

  tau.alpha ~ dgamma(0.001,0.001)
  alpha0 ~ dnorm( 0.0,1.0E-6)
  beta0 ~ dnorm( 0.0,1.0E-6)
  tau.beta ~ dgamma(0.001,0.001)

  tau ~ dgamma(0.001,0.001)
  sigma <- 1 / sqrt(tau)
  gamma ~ dnorm( 0.0,1.0E-6)

  theta ~ dnorm( 0.0,1.0E-6)
  psi ~ dgamma(0.001,0.001)
  }"
```

```
# Set initial values
inits2 <- function (){
  list (alpha0=4, beta0=0, gamma=0, tau.alpha=1, tau.beta=1, tau=1, theta=6, psi=1)
}

parameters2 <- c("alpha","beta", "tau", "gamma")

jagsfit2 <- jags(
  data = data,
  inits = inits2,
  parameters.to.save = parameters2,
  model.file = textConnection(model2),
  n.burnin = burn.in,
  n.iter = n.thin * N + burn.in,
  n.chains = 1,
  n.thin = n.thin
)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 394
##    Unobserved stochastic nodes: 356
##    Total graph size: 2097
##
## Initializing model
```

The confidence interval for the first five α_i are

```
CI.alpha2 <- jagsfit2$BUGSoutput$summary[1:106, c("2.5%", "97.5%")]

t(CI.alpha2[1:5, ])
```

```
##       alpha[1] alpha[2] alpha[3] alpha[4] alpha[5]
## 2.5%  5.687612 5.743986 5.695070 5.641782 5.655504
## 97.5% 6.557281 6.572347 6.544861 6.498367 6.498220
```

and for the first five β_i are

```
CI.beta2 <- jagsfit2$BUGSoutput$summary[-(1:106), c("2.5%", "97.5%")]

t(CI.beta2[1:5, ])
```

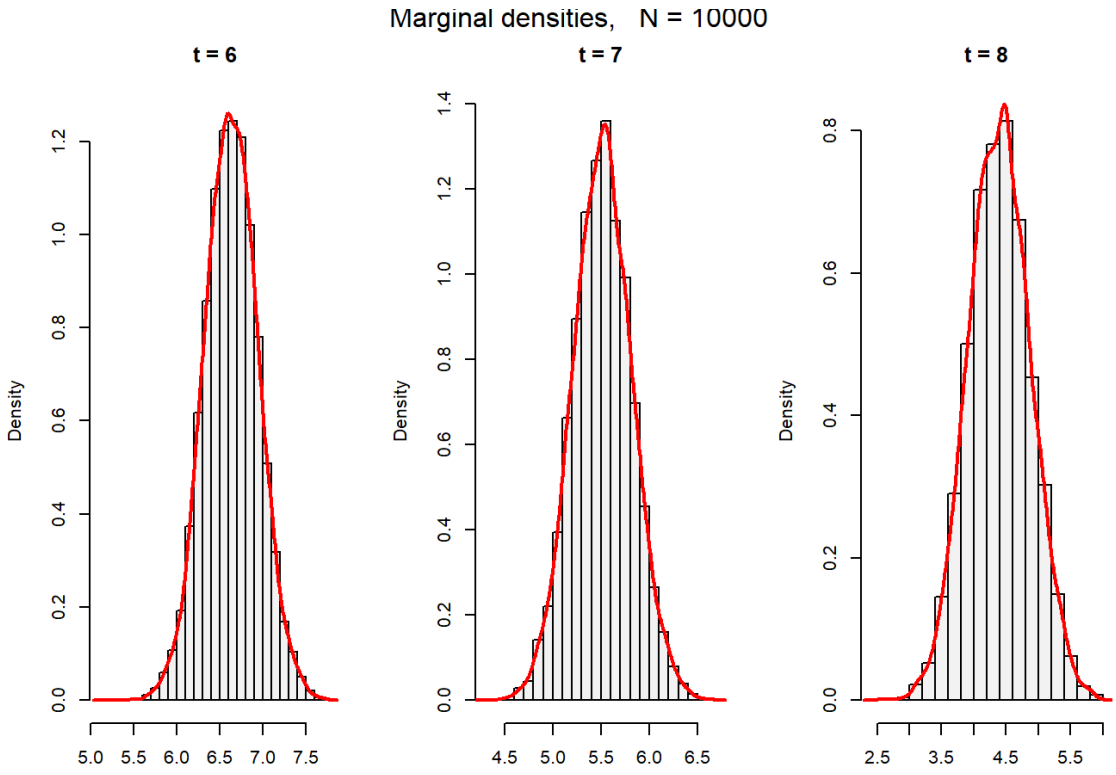
```
##         beta[1]   beta[2]    beta[3]    beta[4]   beta[5]
## 2.5%  -3.414692 -2.033611 -3.9850340 -3.6671147 -2.547817
## 97.5%  1.246373  2.201013  0.3464042  0.3562697  1.766798
```

As expected, the confidence intervals are wider.

Now we sample from the posterior predictive.

```
sims2 <- jagsfit2$BUGSoutput$sims.list

sample.pred2 <- matrix(NA, nrow=N, ncol=3)
alpha <- rowMeans(sims2$alpha)
beta <- rowMeans(sims2$beta)
for (j in 1:3){
  sample.pred2[ ,j] <- rnorm(N, alpha + beta*(t.pred[j] - t.bar) + sims2$gamma*(y0.pred-y.bar), sims2$tau)
}
```

```
plot.marginals(sample.pred2, c("t = 6", "t = 7", "t = 8" ), c(1, 3))
```



- Construct posterior predictive intervals for Y of size 0.95 for the previous case.

```
colnames(sample.pred2) <- c("t = 6", "t = 7", "t = 8")
apply(sample.pred2, 2, function(x) return(quantile(x, probs= c(0.025, 0.975)) ) )
```

```
##          t = 6    t = 7    t = 8
## 2.5%  6.024950 4.913338 3.481078
## 97.5% 7.252112 6.129908 5.340711
```