Stat4DS2+DS - Final Project

Tsardanidis Iason (1846834)

Hipparcos star dataset: an in-depth fully Bayesian analysis.

In this project we are dealing with a subset of Hipparcos stars dataset extracted from the following link which include many Hyades members with the selection criterion that the parallax lie between 20 and 25 mas (i.e. Hipparcos stars with distances 40-50 pc). The extraction was made using the Vizier catalog service. Our main purpose is to perform an analysis for the relation between the *colour of the star* $B-V$ and the logarithm of the *luminosity* $logL$.

This dataset has the following columns:

```
1. HIP = Hipparcos star number
2. Vmag = Visual band magnitude.  This is an inverted logarithmic measure of brightness
3. RA = Right Ascension (degrees), positional coordinate in the sky equivalent to longitude on the Earth
4. DE = Declination (degrees), positional coordinate in the sky equivalent to latitude on the Earth
5. Plx = Parallactic angle (mas = milliarcsseconds).  1000/Plx gives the distance in parsecs (pc)
6. pmRA = Proper motion in RA (mas/yr).  RA component of the motion of the star across the sky
7. pmDE = Proper motion in DE (mas/yr). DE component of the motion of the star across the sky
8. e_Plx = Measurement error in Plx (mas)
9. B-V = Color of star (mag)
```

```r
# set seed
set.seed(123)

# import data
df = read.csv(file="HIP_star.dat",sep="")
df = na.omit(df)

#We construct the logarithm of luminosity variable
df$logL = (15 - df$Vmag - 5*log(df$Plx))/2.5

x = df$B.V
y = df$logL
z = df$Vmag
N = length(x)
```

```r
# load the libraries
library(ggplot2)
library(moments)
library(Metrics)
library(R2jags, quietly = T)
library(ggmcmc, quietly = T)
library(corrplot, quietly = T)
library(bridgesampling)
```

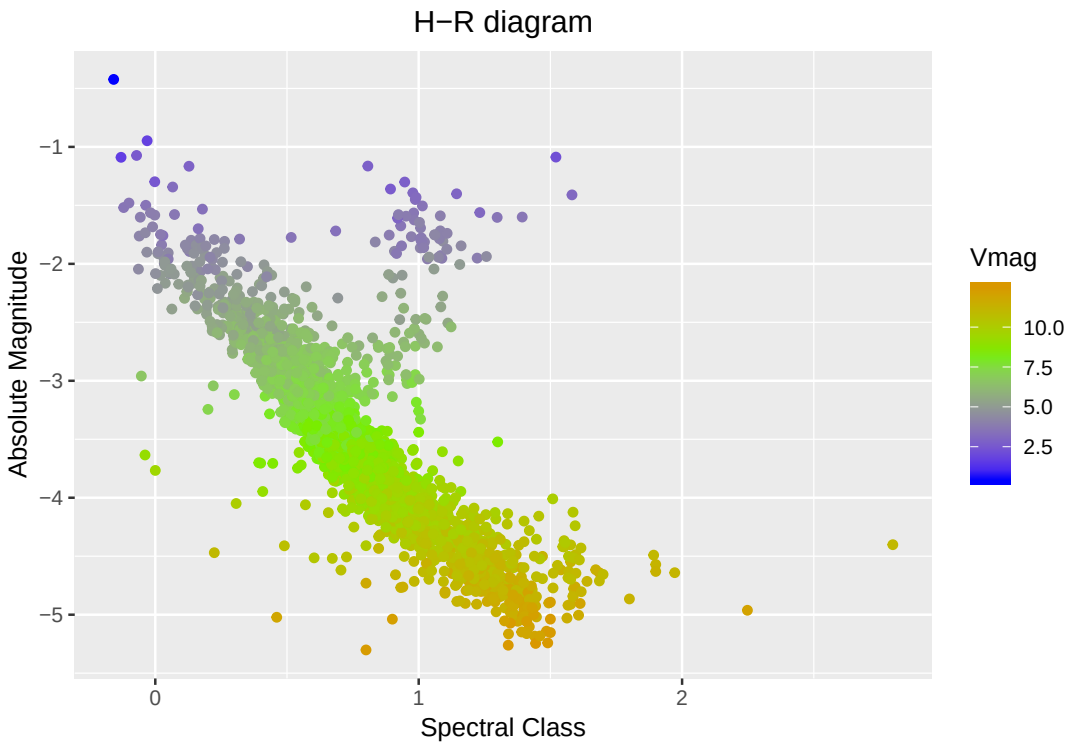
The Hertzsprung–Russell diagram (HR diagram,), is a scatter plot of stars showing the relationship between the stars' absolute magnitudes or luminosities versus their stellar classifications or effective temperatures. More simply, it plots each star on a graph plotting the star's brightness against its temperature (colour-index). The HR diagram can be plotted by plotting logL vs. B-V where (roughly) the log-luminosity in units of solar luminosity is constructed:

$$\log_{10} L = \frac{15 - V_{mag} - 5\log_{10} Plx}{2.5}$$

.

We can illustrate the dataset below:

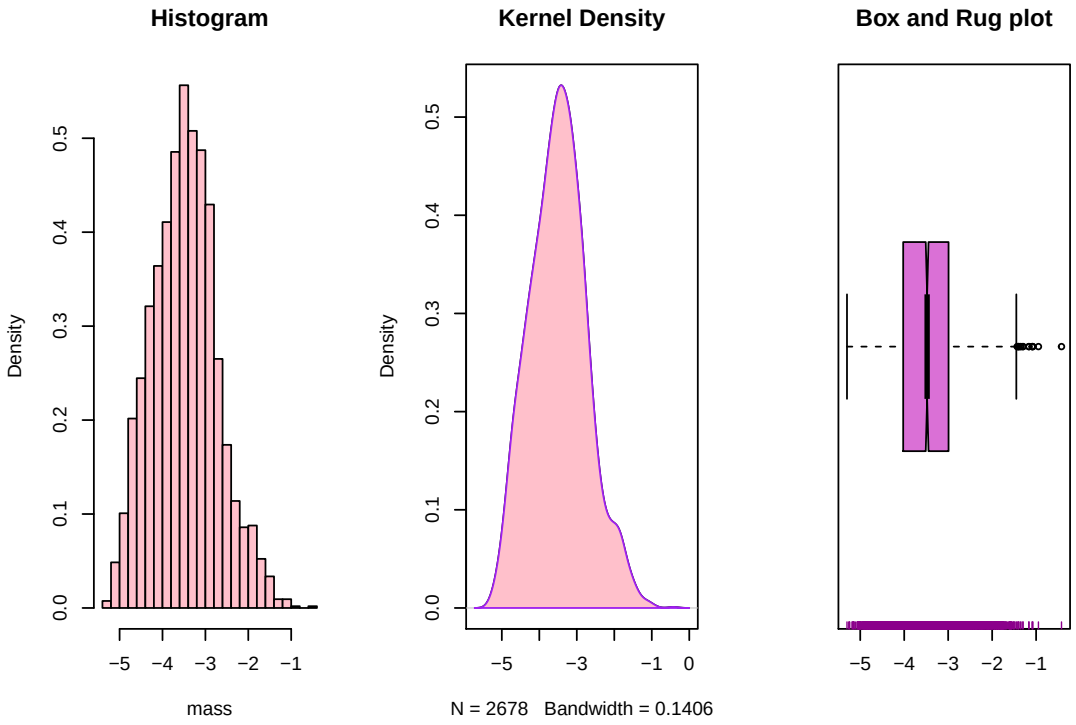
```r
# Color by Vmag values
p =ggplot(df, aes(x=B.V, y=logL, color=Vmag)) + geom_point()
mid=mean(df$Vmag)
p + ggtitle("H-R diagram") +
  xlab("Spectral Class") + ylab("Absolute Magnitude") +
  scale_color_gradient2(midpoint=mid,low="blue", mid="chartreuse2",high="red" ) +
  theme(plot.title = element_text(hjust = 0.5))
```



The dataset above, composed of stars of the **main sequence** of low luminosity. The main sequence is the point that stars spend about 90% of their lives burning hydrogen into helium in their cores and stretching from the upper left (hot, luminous stars) to the bottom right (cool, faint stars) dominates the HR diagram. As next step, we can illustrate some summary statistics for the $B-V$ and $logL$ variables:

```r
summary.list = function(mass){

par(mfrow=c(1,3))

hist(mass,col='pink',breaks=25,prob=T,main="Histogram")

d = density(mass)
plot(d, main="Kernel Density")
polygon(d, col='pink', border="purple")

boxplot(x=mass,notch=TRUE,col='orchid',main='Box and Rug plot',horizontal=TRUE)

rug(mass,col = "darkmagenta",lwd=.8)

list(
  Number.of.data.with.NA.removed= length(mass[!is.na(mass)]),
  Mean=mean(mass, na.rm=TRUE),
  Median=median(mass, na.rm=TRUE),
  Min.Max=range(mass, na.rm=TRUE),
  Range=max(mass, na.rm=TRUE) - min(mass, na.rm=TRUE),
  Variance=var(mass, na.rm=TRUE),
```

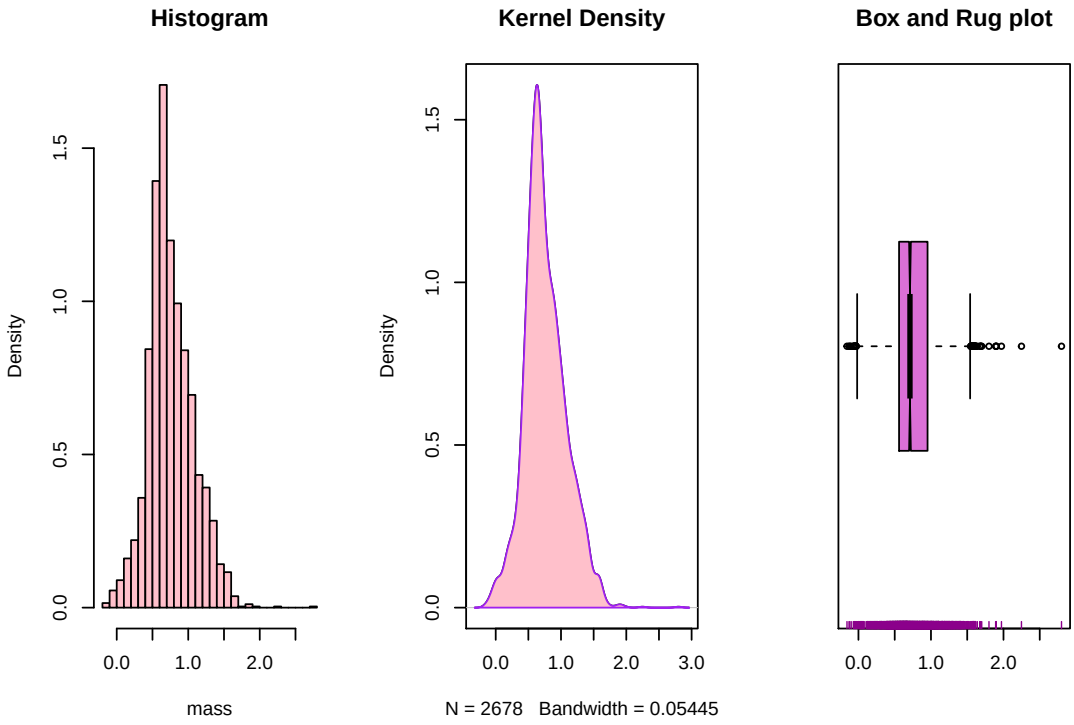
```
  Std.Dev=sd(mass, na.rm=TRUE),
  Coeff.Variation.Prcnt=sd(mass, na.rm=TRUE)/mean(mass, na.rm=TRUE)*100,
  Std.Error=sd(mass, na.rm=TRUE)/sqrt(length(mass[!is.na(mass)])),
  Quantile=quantile(mass, na.rm=TRUE),
  Skewness = skewness(mass),
  Kurtosis = kurtosis(mass)
)

}

# summary of logL
summary.list(y)
```




```
## $Number.of.data.with.NA.removed
## [1] 2678
##
## $Mean
## [1] -3.481624
##
## $Median
## [1] -3.47873
##
## $Min.Max
## [1] -5.301397 -0.422967
##
## $Range
## [1] 4.87843
##
## $Variance
## [1] 0.5734009
##
## $Std.Dev
## [1] 0.7572324
##
```

```
## $Coeff.Variation.Prcnt
## [1] -21.7494
##
## $Std.Error
## [1] 0.01463268
##
## $Quantile
##        0%       25%       50%       75%      100%
## -5.301397 -4.022996 -3.478730 -2.991553 -0.422967
##
## $Skewness
## [1] 0.2532049
##
## $Kurtosis
## [1] 2.951996
```

```
# summary of B-V
summary.list(x)
```




```
## $Number.of.data.with.NA.removed
## [1] 2678
##
## $Mean
## [1] 0.7615299
##
## $Median
## [1] 0.7105
##
## $Min.Max
## [1] -0.158  2.800
##
## $Range
## [1] 2.958
##
## $Variance
```

```
## [1] 0.1012434
##
## $Std.Dev
## [1] 0.3181876
##
## $Coeff.Variation.Prcnt
## [1] 41.78268
##
## $Std.Error
## [1] 0.006148625
##
## $Quantile
##      0%     25%     50%     75%    100%
## -0.1580  0.5600  0.7105  0.9530  2.8000
##
## $Skewness
## [1] 0.4874228
##
## $Kurtosis
## [1] 3.966495
```

Frequentistic approach:

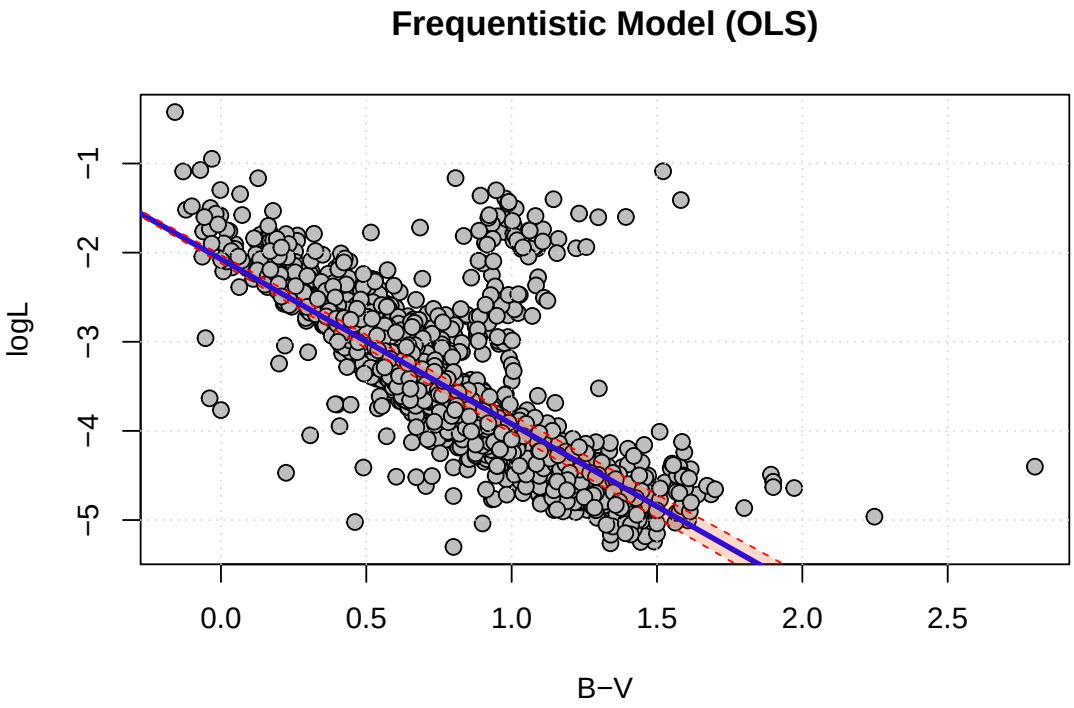
We can use a frequentistic approach and fit a linear model in our data Y_i from every x_i. First we will compute a linear model using *Ordinary Least Squares* method:

```
# frequentistic approach using OLS

n = length(y) # Find length of y to use as sample size
lm.model = lm(y ~ x) # Fit linear model

# Extract fitted coefficients from model object
beta = c(rep(NA,2))
beta[1] = lm.model$coefficients[[1]]
beta[2] = lm.model$coefficients[[2]]

print(list(beta_0=round(beta[1],4),beta_1=round(beta[2],4)))
```

```
## $beta_0
## [1] -2.073
##
## $beta_1
## [1] -1.8498
```

```
y.fit =  beta[1] + beta[2]*x

rmse.lm = rmse(y, y.fit)


# Plot the fitted linear regression line and the computed confidence bands
x_vals = seq(min(x)-0.15,max(x)+0.15,0.01)
y_vals = beta[1] + beta[2]*x_vals
plot(x,y,xlab = "B-V", ylab = "logL", main = "Frequentistic Model (OLS)",
     pch = 21,cex = 1.2,bg = 'gray')
grid(nx=NULL, ny=NULL, lty="dotted", equilogs=FALSE)
lines(x_vals,y_vals,col="blue", lwd=3)
low.ci= confint(lm.model)[1:2,1]  #lower confidence interval
```

```
high.ci=confint(lm.model)[1:2,2]  #upper confidence interval
lower = low.ci[1] + low.ci[2]*x_vals
upper = high.ci[1] + high.ci[2]*x_vals
lines(x_vals,lower,col="red", lty="dashed")
lines(x_vals,upper,col="red", lty="dashed")
polygon(c(x_vals,rev(x_vals)),c(lower,rev(upper)),
        border=NA,col=adjustcolor("orangered", alpha.f = 0.20))
```

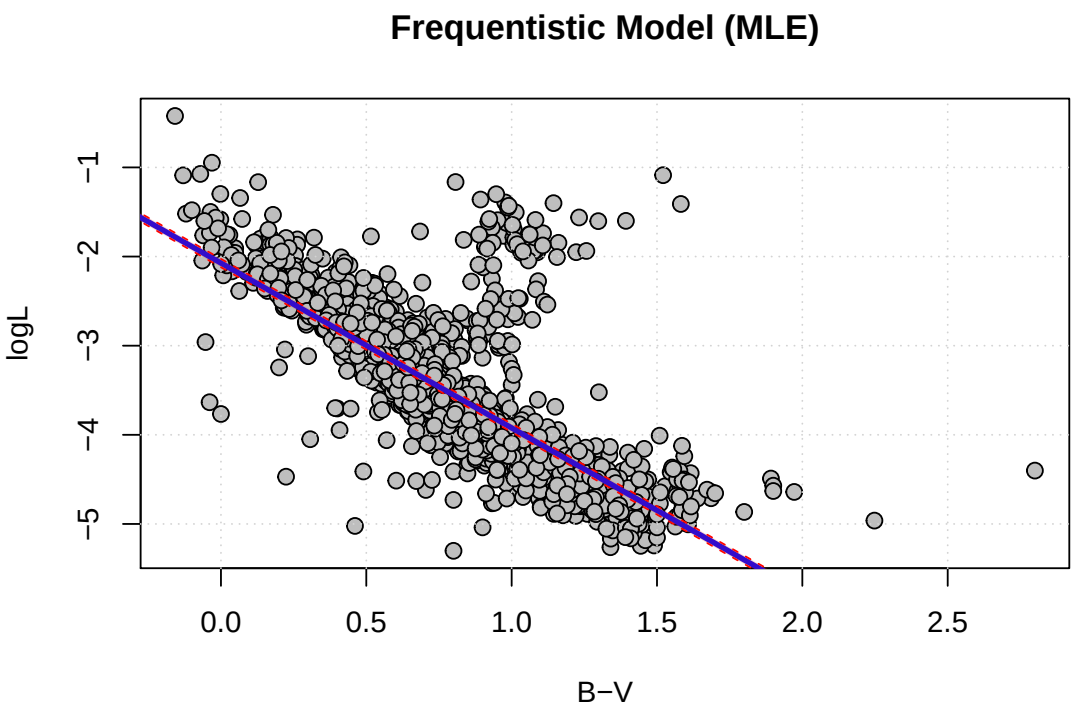


Secondly using Maximum Likelihood Estimator:

```
#frequentistic approach using MLE

init_parameters = c(-2,-2,1)
# compute the log-likelihood function
log_L = function(pars) {
  beta_0 = pars[1]
  beta_1 = pars[2]
  tau = pars[3]
  mu = beta_0 + beta_1*(x)
  out = sum(dnorm(y, mu, tau, log=T))
  return(out)
}
# fit maximum likelihood estimates
fit = optim(par = init_parameters, log_L, control = list(fnscale=-1),hessian=TRUE)

# find maximum likelihood estimates for the vector of parameters of interest
beta = fit$par
print(list(beta_0=round(beta[1],4),beta_1=round(beta[2],4)))
```

```
## $beta_0
## [1] -2.0729
##
## $beta_1
## [1] -1.8498
```

```r
# find maximum likelihood estimates for the vector of parameters confidence interval
fisher_info = solve(-fit$hessian)
prop_sigma = sqrt(diag(fisher_info))
prop_sigma = diag(prop_sigma)
high.ci = (beta + 1.96*prop_sigma)[1:2]
low.ci = (beta - 1.96*prop_sigma)[1:2]

#Plot the fitted linear regression line and the computed confidence bands
x_vals = seq(min(x)-0.15,max(x)+0.15,0.01)
y_vals = beta[1] + beta[2]*x_vals
plot(x,y,xlab = "B-V", ylab = "logL", main = "Frequentistic Model (MLE)",
     pch = 21,cex = 1.2,bg = 'gray')
grid(nx=NULL, ny=NULL, lty="dotted", equilogs=FALSE)
lines(x_vals,y_vals,col="blue", lwd=3)
lower = low.ci[1] + low.ci[2]*x_vals
upper = high.ci[1] + high.ci[2]*x_vals
lines(x_vals,lower,col="red", lty="dashed")
lines(x_vals,upper,col="red", lty="dashed")
polygon(c(x_vals,rev(x_vals)),c(lower,rev(upper)),
        border=NA,col=adjustcolor("orangered", alpha.f = 0.20))
```



We can see that both frequentistic approaches provide similar results and coincide, something that was expected since error terms are normally distributed.

(Non-Conjugate) Bayesian Analysis:

We will try to analyze and fit the dataset we have with the following models:

1. A linear model:
$$y = \beta_0 + \beta_1 x$$
2. A quadratic polynomial model of degree $n = 2$:
$$y = \sum_{i=0}^{2} \beta i x^i$$
3. A super-polynomial exponential model:
$$y = \beta_0 + \beta_1 \gamma^x$$

The model is essentially a random normal effects growth curve that according to our previous prior hypothesis could be *linear*,*polynomial* or *exponential*:
$$Y_i \sim Normal(\mu_i, \tau)$$

More precisely assuming the our data produced following a *normal distribution* we will try to find the appropriate parameters μ and τ in order to construct a stationary and robost model, capable to explain our existed data and moreover to be generalized in new unseen data using *bayesian analysis* and its R-packages.

We may follow **different linear approaches** such that for fixed intercept or slope respectively. We won't concern ourselves with that thought, in this project.

First model - Linear Model with gamma distributed variance

We are dealing with a linear model, so the *mean* of Y_i can be expressed as:

$$\mu_i = \beta_0 + \beta_1 x_i$$

where priors of parameters are:

$$\tau \sim Gamma(0.001, 0.001)$$

and

$$\beta_0 \sim Normal(0, 0.001)$$

$$\beta_1 \sim Normal(0, 0.001)$$

The likelihood function can be derived in this way:

$$L(y_i|\mu_i, \tau, x_i) = L(y_i|\beta_0, \beta_1, \tau, x_i) = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi\tau^2}} \exp\left\{\frac{-(y_i - \mu_i)^2}{2\tau^2}\right\} \propto \prod_{i=1}^{N} \exp\left\{\frac{-(y_i - \mu_i)^2}{2\tau^2}\right\}$$

$$\propto \exp\left\{-\frac{1}{2\tau^2}\sum_{i=1}^{N}(y_i - \mu_i)^2\right\} \propto \exp\left\{-\frac{1}{2\tau^2}\sum_{i=1}^{N}(y_i - (\beta_0 + \beta_1 x_i))^2\right\}$$

Full conditional for every β_i for $i \in \{0, 1\}$ can be derived as:

$$\pi(\beta_i|\beta_{j\neq i}, \tau, X, Y) = \frac{L(Y|\beta_0, \beta_1, \tau, X)\pi(\beta_i)}{f(Y)} \propto L(Y|\beta_0, \beta_1, \tau, X)\pi(\beta_i)$$

and for variance respectively:

$$\pi(\tau|\beta_0, \beta_1, X, Y) = \frac{L(Y|\beta_0, \beta_1, \tau, X)\pi(\tau)}{f(Y)} \propto L(Y|\beta_0, \beta_1, \tau, X)\pi(\tau)$$

where

$$\pi(\beta_i) \sim Normal(0, 0.001)$$

and

$$\pi(\tau) \sim Gamma(0.001, 0.001)$$

Below is the code using RJAGS package in order to estimate the most appropriate factors for this model:

```
# first model -- linear mu -- with gamma tau prior

JAGS.data = list(
  Y = y,
  x = x,
  N = N)


model = function()
{
  for(i in 1:N){
        #Likelihood
```

```
        Y[i] ~ dnorm(mu[i], tau)
        mu[i] = beta[1] + beta[2]*x[i]
                }
  for(i in 1:2){
        #Prior for coefficients
        beta[i] ~ dnorm(0.0, 0.001)
    }
        #Prior for variance
        tau ~ dgamma(0.001,0.001)
        sigma = sqrt(1.0 / tau)  # precision
}



## Name the JAGS parameters
JAGS.params = c("beta", "sigma")

## Define the starting values for JAGS
JAGS.inits = function(){
  list("beta" = rnorm(2, mean = 0.0, sd = 10.0), "tau" = rgamma(1, shape = 1.0, rate = 1.0))
}

mod1 = jags(data=JAGS.data, inits=JAGS.inits, JAGS.params, n.chains=3, n.iter=15000,
             n.burnin=1000, n.thin = 1, model.file=model, DIC=TRUE)
```

```
## module glm loaded

## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 2678
##    Unobserved stochastic nodes: 3
##    Total graph size: 7427
##
## Initializing model
```

```
mod1
```

```
## Inference for Bugs model at "/tmp/RtmpXBvZB9/model9ac3d66a009.txt", fit using jags,
##  3 chains, each with 15000 iterations (first 1000 discarded)
##  n.sims = 42000 iterations saved
##           mu.vect sd.vect     2.5%      25%      50%      75%    97.5%
## beta[1]    -2.073   0.024   -2.120   -2.089   -2.073   -2.057   -2.026
## beta[2]    -1.850   0.029   -1.906   -1.869   -1.850   -1.830   -1.793
## sigma       0.477   0.007    0.464    0.472    0.477    0.481    0.490
## deviance 3630.586   2.457 3627.815 3628.798 3629.955 3631.695 3636.857
##           Rhat n.eff
## beta[1]  1.001 42000
## beta[2]  1.001 42000
## sigma    1.001 42000
## deviance 1.001 17000
##
## 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 = 3.0 and DIC = 3633.6
## DIC is an estimate of expected predictive error (lower deviance is better).
```

```
# model parameters
beta[1] = mod1$BUGSoutput$summary[,"mean"]["beta[1]"]
beta[2] = mod1$BUGSoutput$summary[,"mean"]["beta[2]"]
sigma = mod1$BUGSoutput$summary[,"mean"]["sigma"]

print(list(beta_0=round(beta[1],4),beta_1=round(beta[2],4)))
```

```
## $beta_0
## [1] -2.073
##
## $beta_1
## [1] -1.8497
```

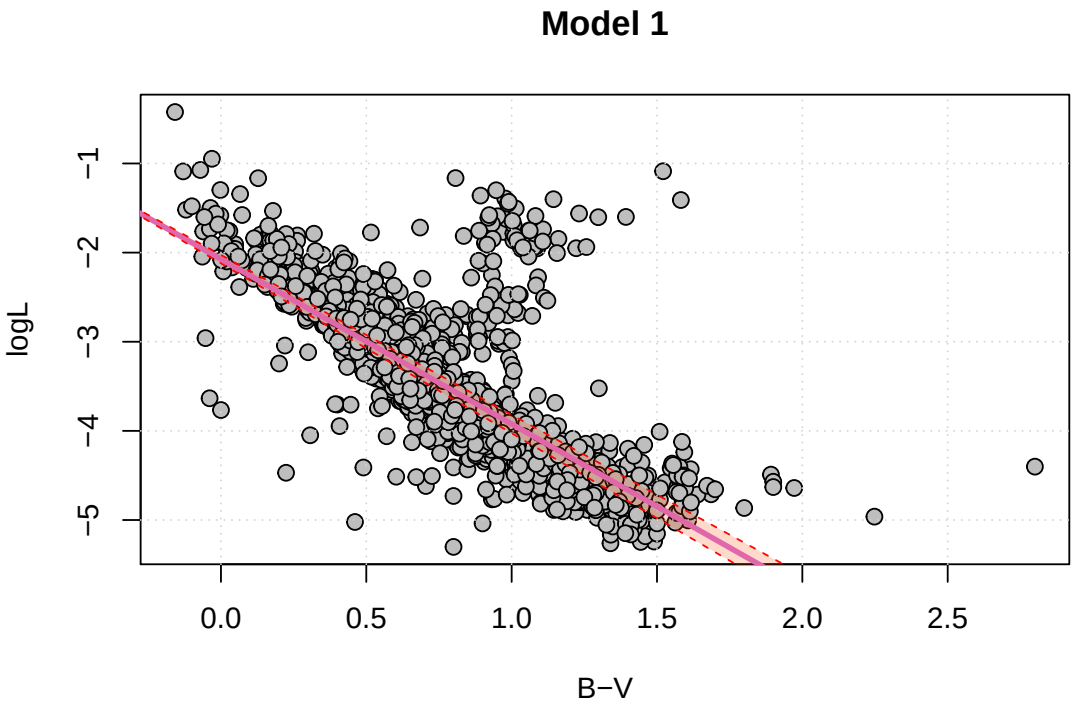
```
# predicted_data
y.fit =  beta[1] + beta[2]*x

# DIC
DIC.mod1 = mod1$BUGSoutput$DIC

# rmse
rmse.mod1 = rmse(y, y.fit)


### marginal likelihood analysis


# model samples
samples_mod1 = mod1$BUGSoutput$sims.matrix


# evaluating the unnormalized posteriors on log scale
log_posterior_mod1 = function(data,samples) {

  sum(dnorm(data$Y, beta[1] + beta[2]*data$x,(1/sigma)^2, log = TRUE)) +
    dnorm(beta[1],0,0.001, log = TRUE) +
    dnorm(beta[2],0,0.001, log = TRUE) +
    dgamma((1/sigma)^2,0.001, 0.001, log = TRUE)

}

# lower and upper bounds
cn = colnames(samples_mod1)
cn = cn[cn != "deviance"]
lb_mod1 = rep(-Inf, length(cn))
ub_mod1 = rep(Inf, length(cn))
names(lb_mod1) = names(ub_mod1) = cn
lb_mod1[[ "sigma" ]] = 0

# compute log marginal likelihood via bridge sampling
mod1.bridge = bridge_sampler(samples = mod1, data = JAGS.data,
                             log_posterior = log_posterior_mod1, lb = lb_mod1,
                             ub = ub_mod1, silent = TRUE)
print(mod1.bridge)
```

```
## Bridge sampling estimate of the log marginal likelihood: -3865809
## Estimate obtained in 15 iteration(s) via method "normal".
```

```r
# compute percentage errors
mod1.error = error_measures(mod1.bridge)$percentage
error.mod1 = error_measures(mod1.bridge)$cv
print(mod1.error)
```

```
## [1] "0.333%"
```

```r
# plot the fit model
x_vals = seq(min(x)-0.15,max(x)+0.15,0.01)
y_vals = beta[1] + beta[2]*x_vals
plot(x,y,xlab = "B-V", ylab = "logL", main = "Model 1",
     pch = 21,cex = 1.2,bg = 'gray')
grid(nx=NULL, ny=NULL, lty="dotted", equilogs=FALSE)
lines(x_vals,y_vals,col="orchid", lwd=3)
low.ci = mod1$BUGSoutput$summary[1:2,3]
high.ci = mod1$BUGSoutput$summary[1:2,7]
lower = low.ci[1] + low.ci[2]*x_vals
upper = high.ci[1] + high.ci[2]*x_vals
lines(x_vals,lower,col="red", lty="dashed")
lines(x_vals,upper,col="red", lty="dashed")
polygon(c(x_vals,rev(x_vals)),c(lower,rev(upper)),
        border=NA,col=adjustcolor("orangered", alpha.f = 0.20))
```



We see some more deep analysis on the parameters of the model:

From the theory of Markov chains, we expect our chains to eventually converge to the stationary distribution, which is also our target distribution. However, there is no guarantee that our chain has converged after a given number of iterations. We will check more closely the convergence through some tools.

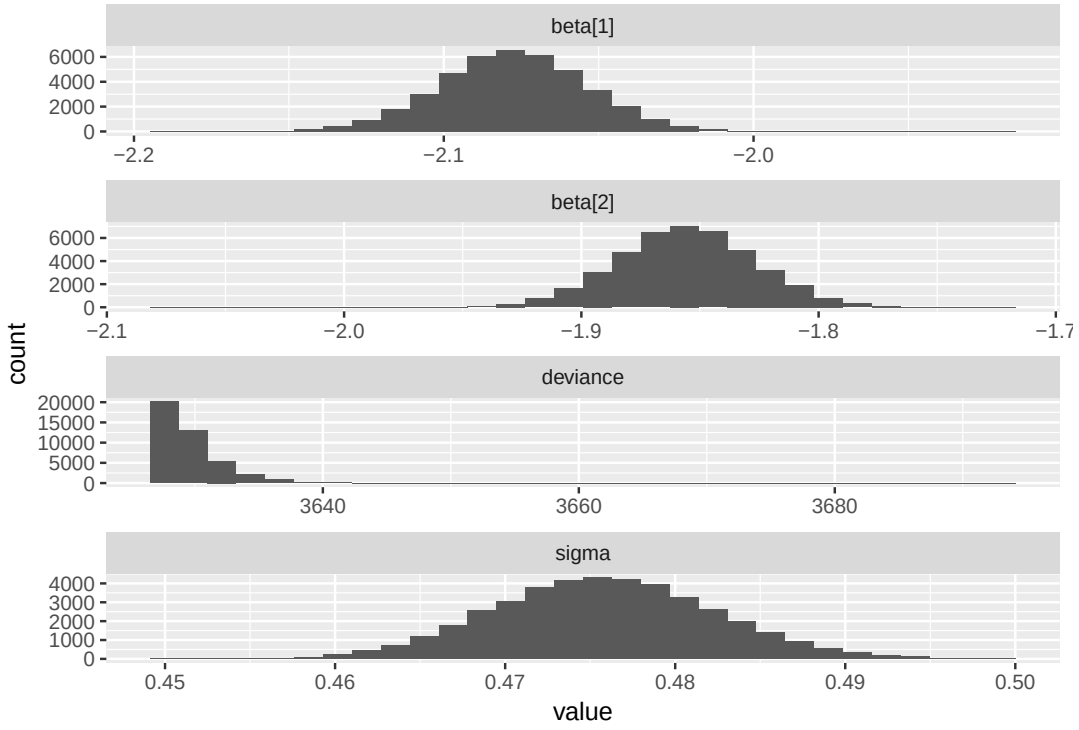
Monitoring traceplots and autocorrelations is also very useful since low or high values indicate fast or slow convergence, respectively.

The lag k autocorrelation ρ_k is the correlation between every draw and its kth lag:

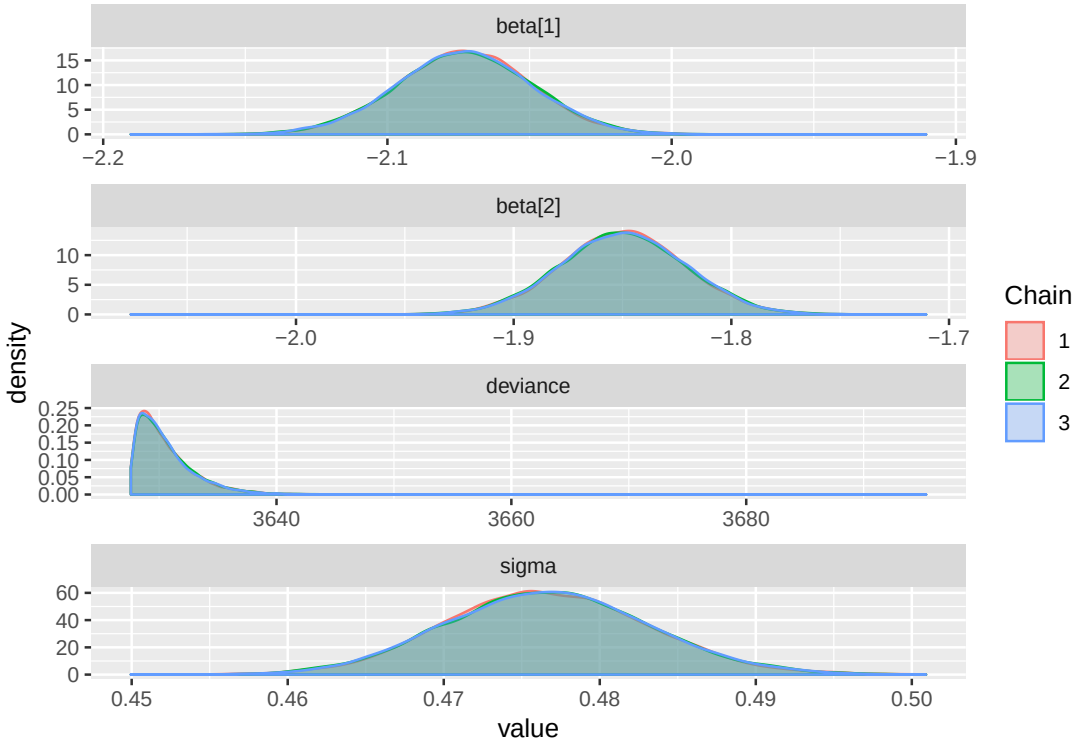
$$\rho_k = \frac{\sum_{i=1}^{n-k}(x_i - \bar{x})(x_{i+k} - \bar{x})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

We would expect the kth lag autocorrelation to be smaller as k increases

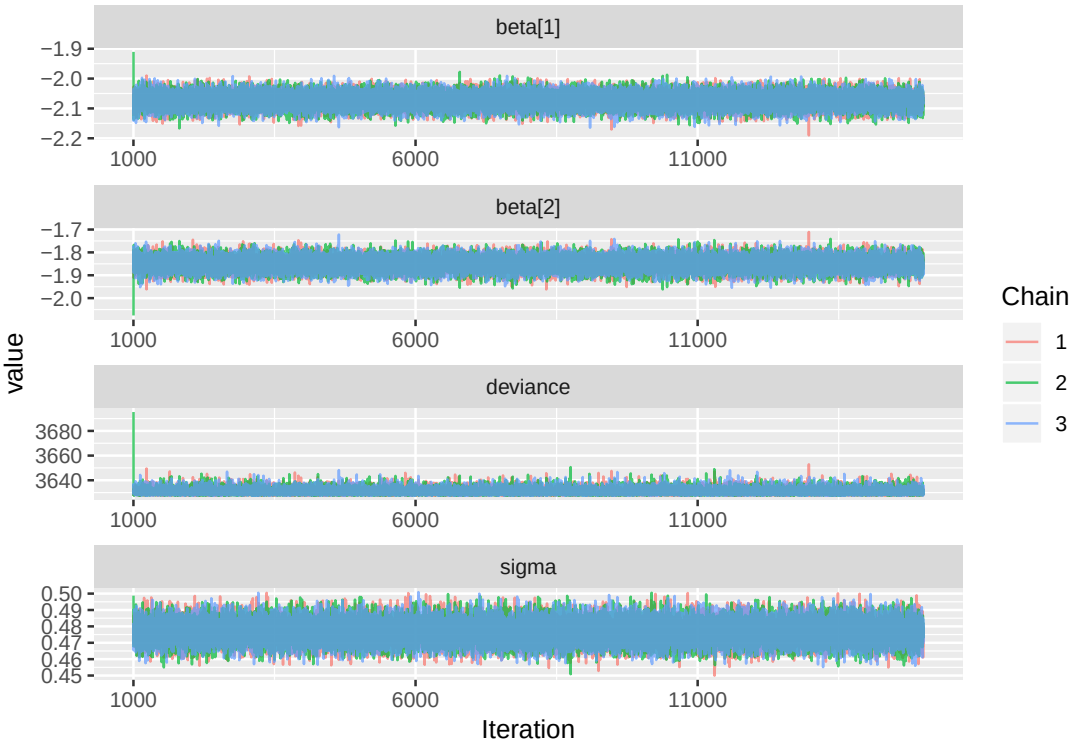
```
# use of the library ggmcmc that allows us to work with ggs object
mod.fit.ggs = ggs(as.mcmc(mod1))

ggs_histogram(mod.fit.ggs)
```



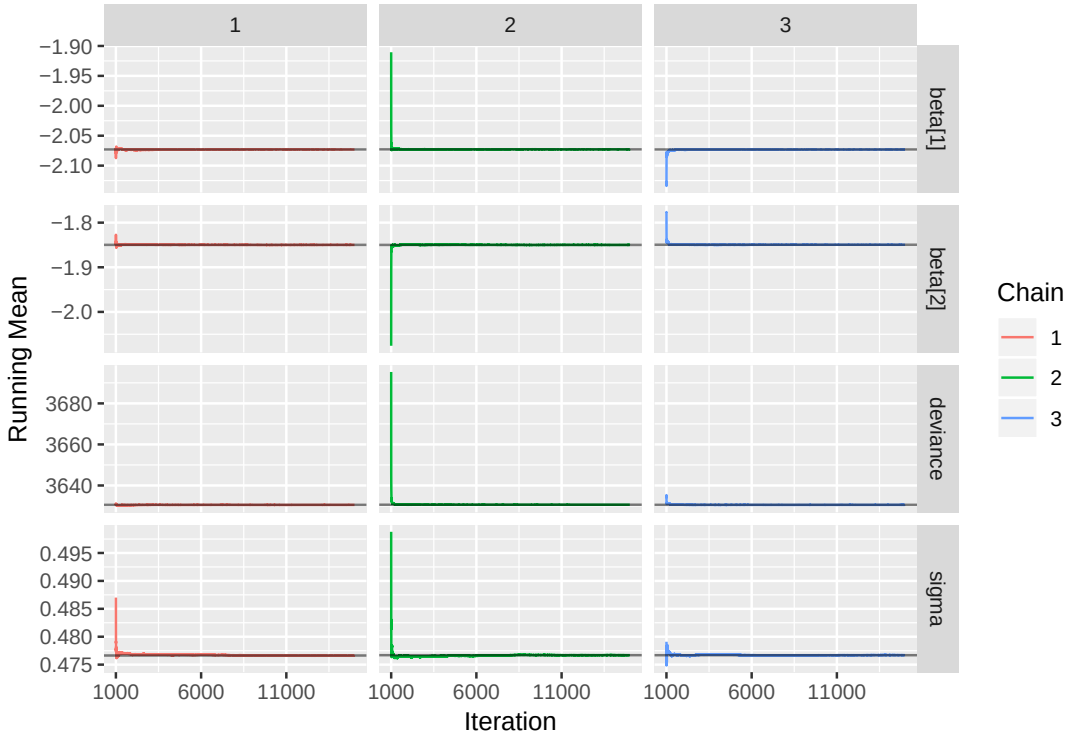

```
ggs_density(mod.fit.ggs)
```



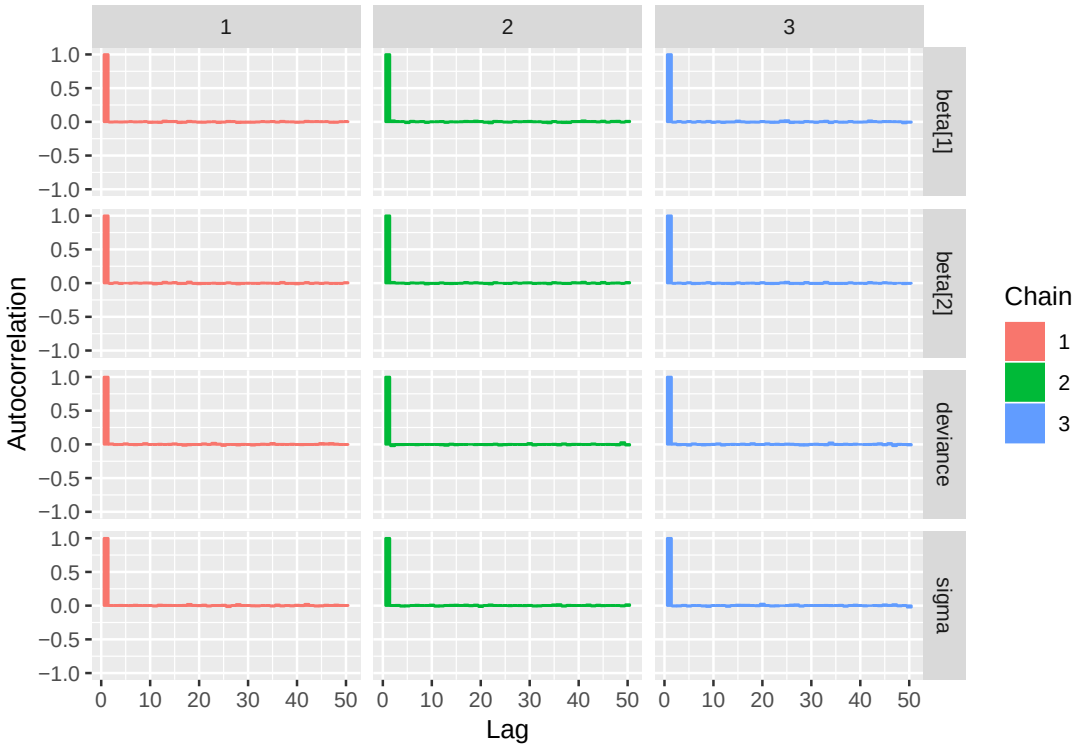

```
ggs_traceplot(mod.fit.ggs)
```




```
ggs_running(mod.fit.ggs)
```




```
ggs_autocorrelation(mod.fit.ggs)
```



We can use some more tools and *convergence diagnostics* and plots in order to infer the stationarity of our model like *gelman* or *geweke* plots, but this will make our analysis too fastidious.

Second model - Linear Model with uniform distributed variance

We are dealing with a linear model, so the *mean* of Y_i can be expressed as:

$$\mu_i = \beta_0 + \beta_1 x_i$$

where priors of parameters are:

$$\tau \sim Unif(0, 10)$$

and

$$\beta_i \sim Normal(0, 0.001)$$

for $i \in \{0, 1\}$

The likelihood and the each parameter conditional functions can be derived following the formula was shown in the previous model with different prior for the τ variance.

```
# second model -- linear mu -- with uniform tau prior

JAGS.data = list(
  Y = y,
  x = x,
  N = N)


model = function()
{
  for(i in 1:N){
        #Likelihood
        Y[i] ~ dnorm(mu[i], tau)
        mu[i] = beta[1] + beta[2]*x[i]
                }
  for(i in 1:2){
        #Prior for coefficients
        beta[i] ~ dnorm(0.0, 0.001)
    }
        #Prior for variance
        tau ~ dunif(0,10)
        sigma = sqrt(1.0 / tau) # precision
}


## Name the JAGS parameters
JAGS.params = c("beta", "sigma")

## Define the starting values for JAGS
JAGS.inits = function(){
  list("beta" = rnorm(2, mean = 0.0, sd = 10.0),
       "tau" = runif(1, min = 0.0, max = 10.0))
}

mod2 = jags(data=JAGS.data, inits=JAGS.inits, JAGS.params, n.chains=3, n.iter=15000,
             n.burnin=1000, n.thin = 1, model.file=model, DIC=TRUE)


## Compiling model graph
##    Resolving undeclared variables
```

```
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 2678
##    Unobserved stochastic nodes: 3
##    Total graph size: 7428
##
## Initializing model
```

```
mod2
```

```
## Inference for Bugs model at "/tmp/RtmpXBvZB9/model9ac3658239d.txt", fit using jags,
##  3 chains, each with 15000 iterations (first 1000 discarded)
##  n.sims = 42000 iterations saved
##           mu.vect sd.vect     2.5%      25%      50%      75%    97.5%
## beta[1]    -2.073   0.024   -2.120   -2.089   -2.073   -2.057   -2.026
## beta[2]    -1.850   0.029   -1.907   -1.869   -1.850   -1.830   -1.793
## sigma       0.476   0.006    0.464    0.472    0.476    0.481    0.489
## deviance 3630.580   2.449 3627.815 3628.793 3629.948 3631.691 3636.892
##           Rhat n.eff
## beta[1]  1.001 23000
## beta[2]  1.001 28000
## sigma    1.001 16000
## deviance 1.001 26000
##
## 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 = 3.0 and DIC = 3633.6
## DIC is an estimate of expected predictive error (lower deviance is better).
```

```
# model parameters
beta[1] = mod2$BUGSoutput$summary[,"mean"]["beta[1]"]
beta[2] = mod2$BUGSoutput$summary[,"mean"]["beta[2]"]

print(list(beta_0=round(beta[1],4),beta_1=round(beta[2],4)))
```

```
## $beta_0
## [1] -2.0728
##
## $beta_1
## [1] -1.8498
```

```
# predicted_data
y.fit = beta[1] + beta[2]*x

# DIC
DIC.mod2 = mod2$BUGSoutput$DIC

# rmse
rmse.mod2 = rmse(y, y.fit)


### marginal likelihood analysis


# model samples
```

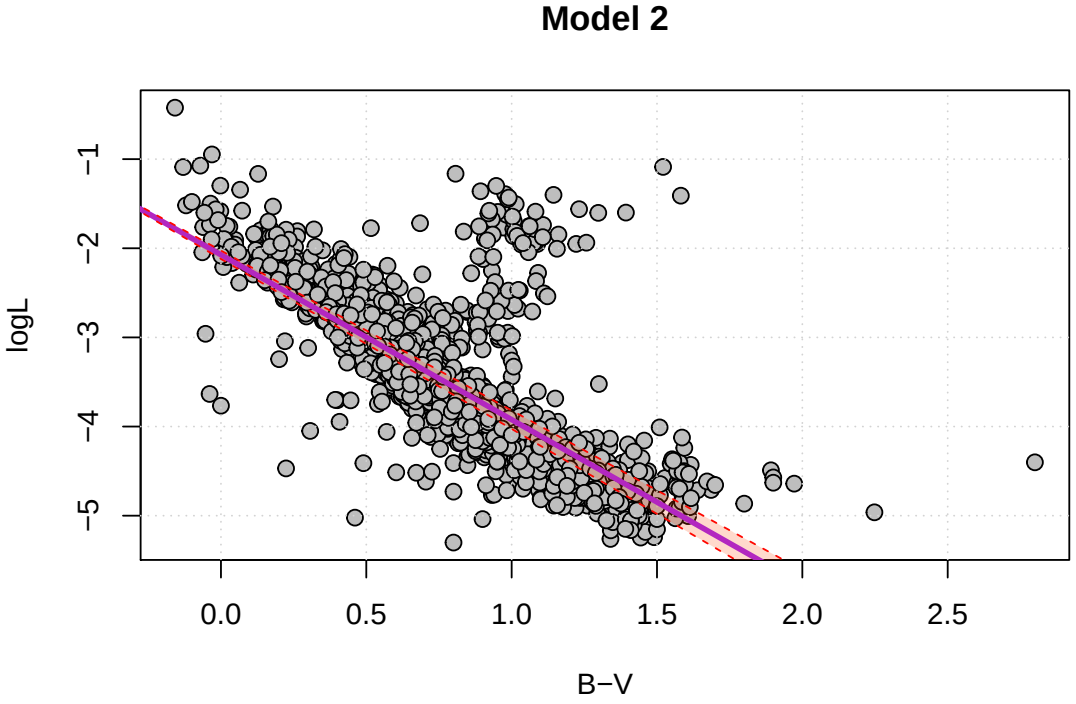
```r
samples_mod2 = mod2$BUGSoutput$sims.matrix


# evaluating the unnormalized posteriors on log scale
log_posterior_mod2 = function(data,samples) {

  sum(dnorm(data$Y, beta[1] + beta[2]*data$x,
            (1/sigma)^2, log = TRUE)) +
    dnorm(beta[1],0,0.001, log = TRUE) +
    dnorm(beta[2],0,0.001, log = TRUE) +
    dunif((1/sigma)^2,0, 10, log = TRUE)

}

# lower and upper bounds
cn = colnames(samples_mod2)
cn = cn[cn != "deviance"]
lb_mod2 = rep(-Inf, length(cn))
ub_mod2 = rep(Inf, length(cn))
names(lb_mod2) = names(ub_mod2) = cn
lb_mod2[[ "sigma" ]] = 0

# compute log marginal likelihood via bridge sampling
mod2.bridge = bridge_sampler(samples = mod2, data = JAGS.data,
                             log_posterior = log_posterior_mod2, lb = lb_mod2,
                             ub = ub_mod2, silent = TRUE)
print(mod2.bridge)
```

```
## Bridge sampling estimate of the log marginal likelihood: -3865695
## Estimate obtained in 15 iteration(s) via method "normal".
```

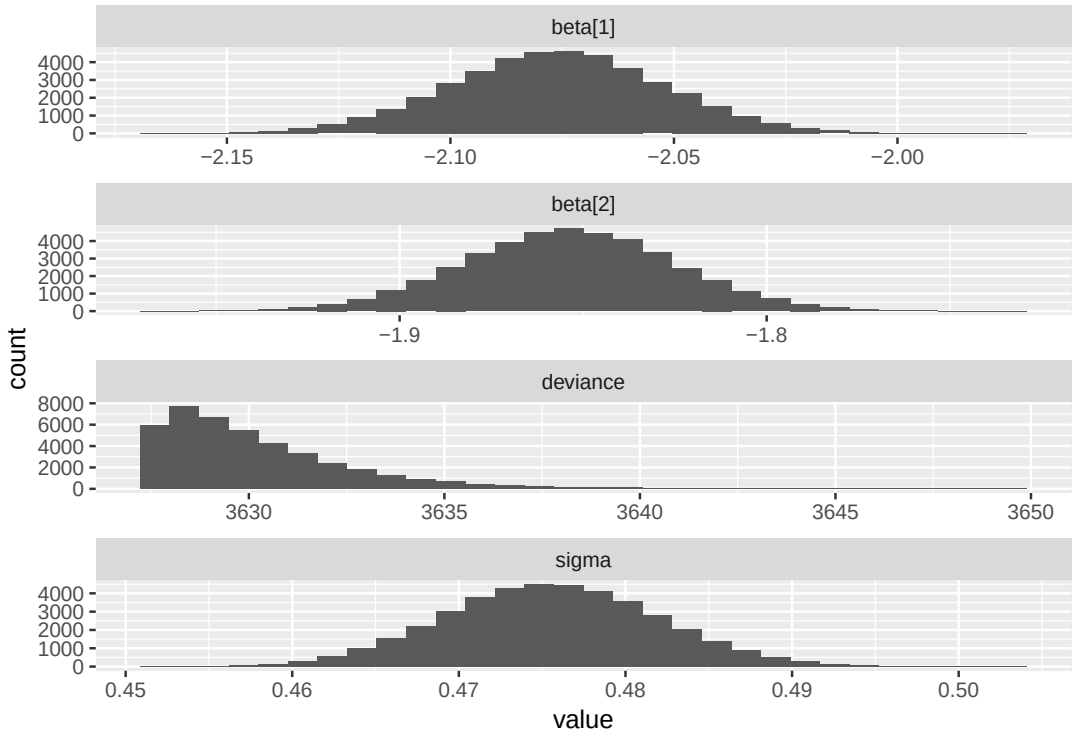
```r
# compute percentage errors
mod2.error = error_measures(mod2.bridge)$percentage
error.mod2 = error_measures(mod2.bridge)$cv
print(mod2.error)
```

```
## [1] "0.341%"
```

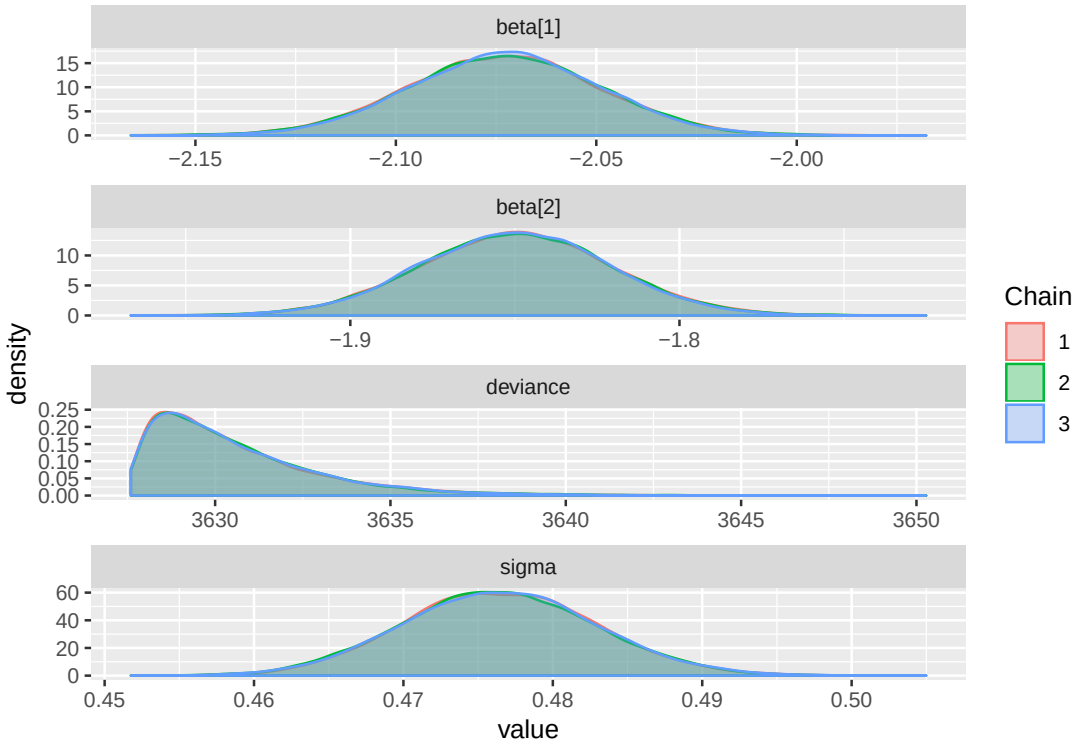
```r
# plot the fit model
x_vals = seq(min(x)-0.15,max(x)+0.15,0.01)
y_vals = beta[1] + beta[2]*x_vals
plot(x,y,xlab = "B-V", ylab = "logL", main = "Model 2",
     pch = 21,cex = 1.2,bg = 'gray')
grid(nx=NULL, ny=NULL, lty="dotted", equilogs=FALSE)
lines(x_vals,y_vals,col="purple", lwd=3)
low.ci=mod2$BUGSoutput$summary[1:2,3]
high.ci=mod2$BUGSoutput$summary[1:2,7]
lower = low.ci[1] + low.ci[2]*x_vals
upper = high.ci[1] + high.ci[2]*x_vals
lines(x_vals,lower,col="red", lty="dashed")
lines(x_vals,upper,col="red", lty="dashed")
polygon(c(x_vals,rev(x_vals)),c(lower,rev(upper)),
        border=NA,col=adjustcolor("orangered", alpha.f = 0.20))
```



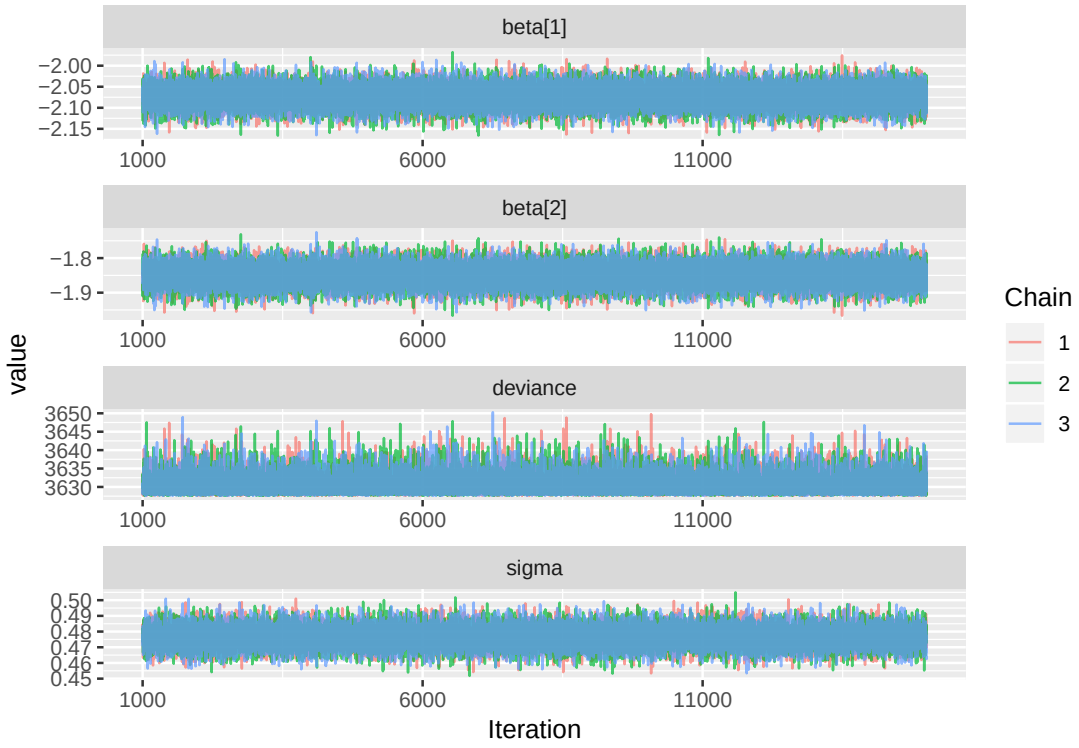

```
# use of the library ggmcmc that allows us to work with ggs object
mod.fit.ggs = ggs(as.mcmc(mod2))
ggs_histogram(mod.fit.ggs)
```



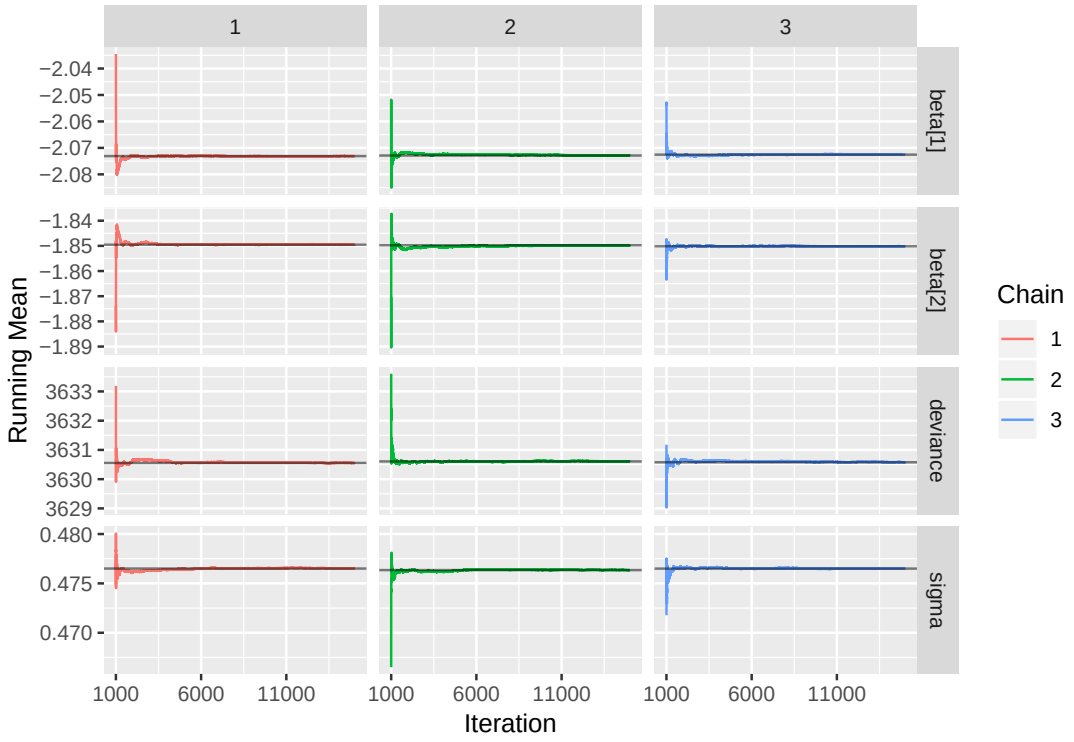

```
ggs_density(mod.fit.ggs)
```



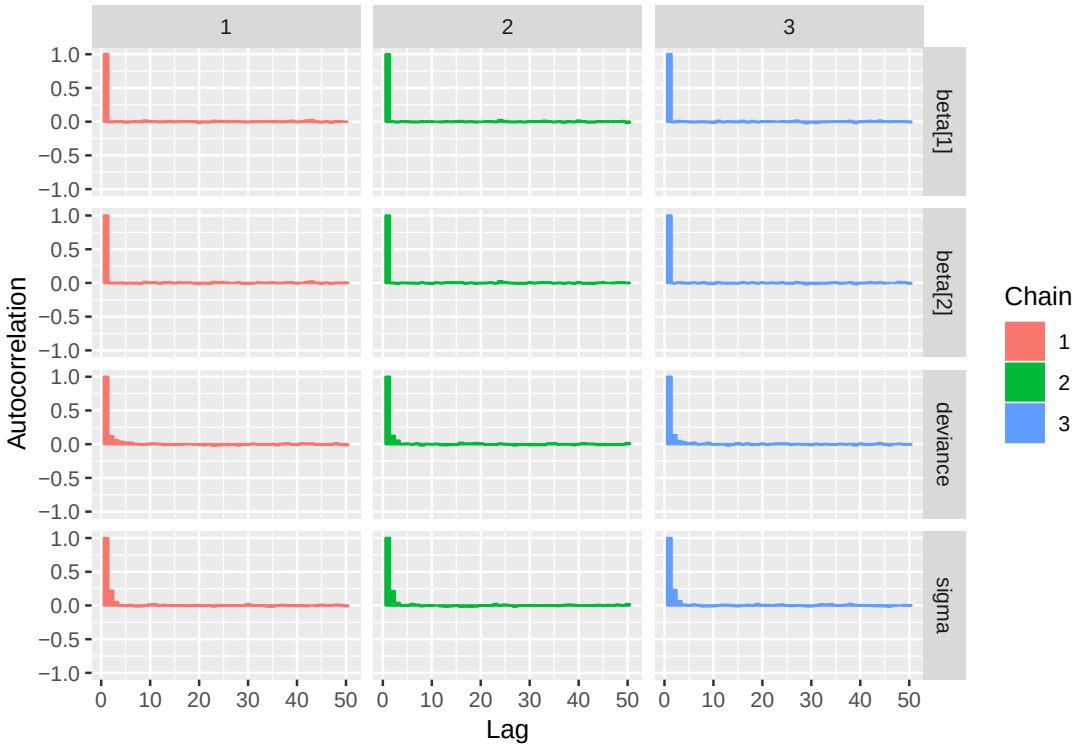

```
ggs_traceplot(mod.fit.ggs)
```




```
ggs_running(mod.fit.ggs)
```




```
ggs_autocorrelation(mod.fit.ggs)
```



Third model - Quadratic Model with gamma distributed variance

Now we will try to apply a polynomial model of degree 2. The *mean* of Y_i can be expressed as:

$$\mu_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2$$

where priors of parameters are:

$$\tau \sim Gamma(0.001, 0.001)$$

and

$$\beta_i \sim Normal(0, 0.001)$$

for every $i \in \{0, 2\}$

The likelihood function can be derived in this way:

$$L(y_i|\mu_i, \tau, x_i) = L(y_i|\beta_0, \beta_1, \beta_2, \tau, x_i) = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi\tau^2}} \exp\left\{\frac{-(y_i - \mu_i)^2}{2\tau^2}\right\} \propto \prod_{i=1}^{N} \exp\left\{\frac{-(y_i - \mu_i)^2}{2\tau^2}\right\}$$

$$\propto \exp\left\{-\frac{1}{2\tau^2}\sum_{i=1}^{N}(y_i - \mu_i)^2\right\} \propto \exp\left\{-\frac{1}{2\tau^2}\sum_{i=1}^{N}(y_i - (\beta_0 + \beta_1 x_i + \beta_2 x_i^2))^2\right\}$$

Full conditional for every β_i for $i \in \{0, 2\}$ can be derived as:

$$\pi(\beta_i|\beta_{j\neq i}, \tau, X, Y) = \frac{L(Y|\beta_0, \beta_1, \beta_2, \tau, X)\pi(\beta_i)}{f(Y)} \propto L(Y|\beta_0, \beta_1, \beta_2, \tau, X)\pi(\beta_i)$$

and for variance:

$$\pi(\tau|\beta_0, \beta_1, \beta_2, X, Y) = \frac{L(Y|\beta_0, \beta_1, \beta_2, \tau, X)\pi(\tau)}{f(Y)} \propto L(Y|\beta_0, \beta_1, \beta_2, \tau, X)\pi(\tau)$$

where

$$\pi(\beta_i) \sim Normal(0, 0.001)$$

and

$$\pi(\tau) \sim Gamma(0.001, 0.001)$$

Below is the code using RJAGS package in order to estimate the most appropriate factors for this model:

```
# third model -- polynomial mu -- with gamma tau prior

JAGS.data = list(
  Y = y,
  x = x,
  N = N)


model = function()
{
  for(i in 1:N){
        #Likelihood
        Y[i] ~ dnorm(mu[i], tau)
        mu[i] = beta[1] + beta[2]*x[i] + beta[3]*x[i]^2
```

```
                }
  for(i in 1:3){
        #Prior for coefficients
        beta[i] ~ dnorm(0.0, 0.001)
    }
        #Prior for  variance
        tau ~ dgamma(0.001,0.001)
        sigma = sqrt(1.0 / tau) # precision
}


## Name the JAGS parameters
JAGS.params = c("beta", "sigma")

## Define the starting values for JAGS
JAGS.inits = function(){
  list("beta" = rnorm(3, mean = 0.0, sd = 10.0),
       "tau" = rgamma(1, shape = 1.0, rate = 1.0))
}

mod3 = jags(data=JAGS.data, inits=JAGS.inits, JAGS.params, n.chains=3, n.iter=15000,
             n.burnin=1000, n.thin = 1, model.file=model, DIC=TRUE)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 2678
##    Unobserved stochastic nodes: 4
##    Total graph size: 9485
##
## Initializing model
```

```
mod3
```

```
## Inference for Bugs model at "/tmp/RtmpXBvZB9/model9ac3efed201.txt", fit using jags,
##  3 chains, each with 15000 iterations (first 1000 discarded)
##  n.sims = 42000 iterations saved
##           mu.vect sd.vect     2.5%      25%      50%      75%    97.5%
## beta[1]    -1.705   0.040   -1.783   -1.732   -1.705   -1.679   -1.627
## beta[2]    -2.883   0.095   -3.071   -2.947   -2.883   -2.820   -2.697
## beta[3]     0.616   0.054    0.510    0.580    0.616    0.652    0.722
## sigma       0.465   0.006    0.453    0.461    0.465    0.470    0.478
## deviance 3503.401   2.868 3499.882 3501.300 3502.740 3504.808 3510.561
##           Rhat n.eff
## beta[1]  1.001 16000
## beta[2]  1.001 17000
## beta[3]  1.001 18000
## sigma    1.001 15000
## deviance 1.001 42000
##
## 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 = 4.1 and DIC = 3507.5
## DIC is an estimate of expected predictive error (lower deviance is better).
```

```r
# model parameters
beta[1] = mod3$BUGSoutput$summary[,"mean"]["beta[1]"]
beta[2] = mod3$BUGSoutput$summary[,"mean"]["beta[2]"]
beta[3] = mod3$BUGSoutput$summary[,"mean"]["beta[3]"]


print(list(beta_0=round(beta[1],4),beta_1=round(beta[2],4),beta_2=round(beta[3],4)))
```

```
## $beta_0
## [1] -1.7053
##
## $beta_1
## [1] -2.8834
##
## $beta_2
## [1] 0.6158
```

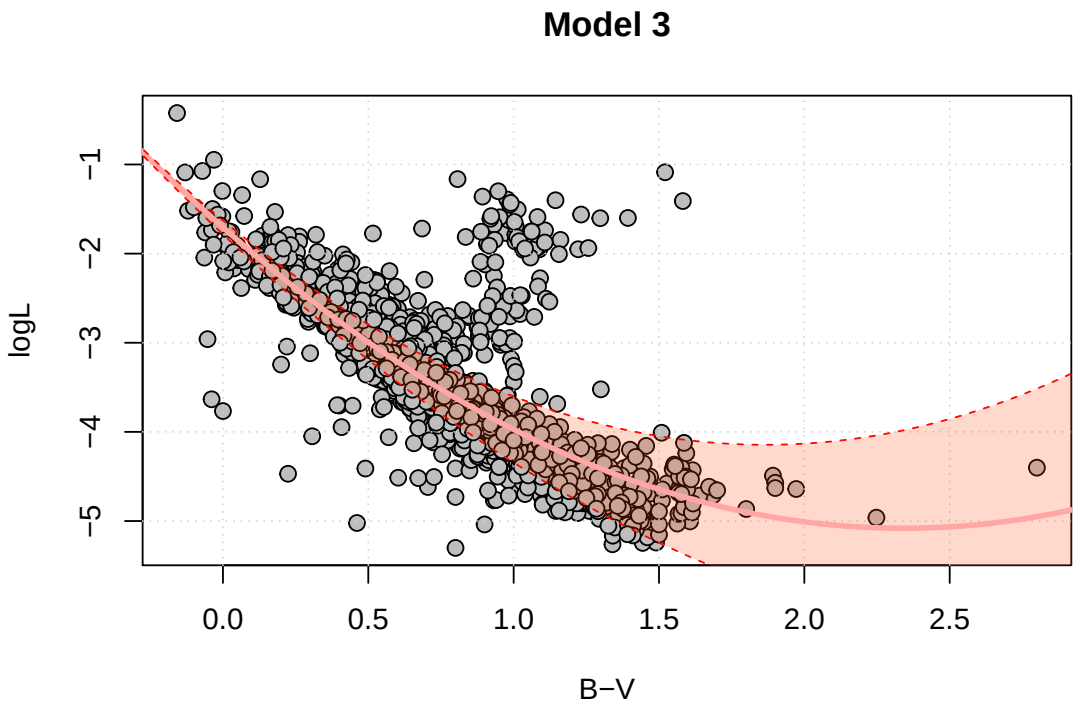
```r
# predicted_data
y.fit = beta[1] + beta[2] * x  + beta[3]*x^2

# DIC
DIC.mod3 = mod3$BUGSoutput$DIC
# rmse
rmse.mod3 = rmse(y, y.fit)


### marginal likelihood analysis


# model samples
samples_mod3 = mod3$BUGSoutput$sims.matrix


# evaluating the unnormalized posteriors on log scale
log_posterior_mod3 = function(data,samples) {

  sum(dnorm(data$Y, beta[1] + beta[2]*data$x  + beta[3]*data$x^2,
            (1/sigma)^2, log = TRUE)) +
    dnorm(beta[1],0,0.001, log = TRUE) +
    dnorm(beta[2],0,0.001, log = TRUE) +
    dnorm(beta[3],0,0.001, log = TRUE) +
    dgamma((1/sigma)^2,0.001, 0.001, log = TRUE)

}

# lower and upper bounds
cn = colnames(samples_mod3)
cn = cn[cn != "deviance"]
lb_mod3 = rep(-Inf, length(cn))
ub_mod3 = rep(Inf, length(cn))
names(lb_mod3) = names(ub_mod3) = cn
lb_mod3[[ "sigma" ]] = 0

# compute log marginal likelihood via bridge sampling
mod3.bridge = bridge_sampler(samples = mod3, data = JAGS.data,
                             log_posterior = log_posterior_mod3, lb = lb_mod3,
                             ub = ub_mod3, silent = TRUE)
print(mod3.bridge)
```

```
## Bridge sampling estimate of the log marginal likelihood: -5807056
## Estimate obtained in 17 iteration(s) via method "normal".
```

```
# compute percentage errors
mod3.error = error_measures(mod3.bridge)$percentage
error.mod3 = error_measures(mod3.bridge)$cv
print(mod3.error)
```

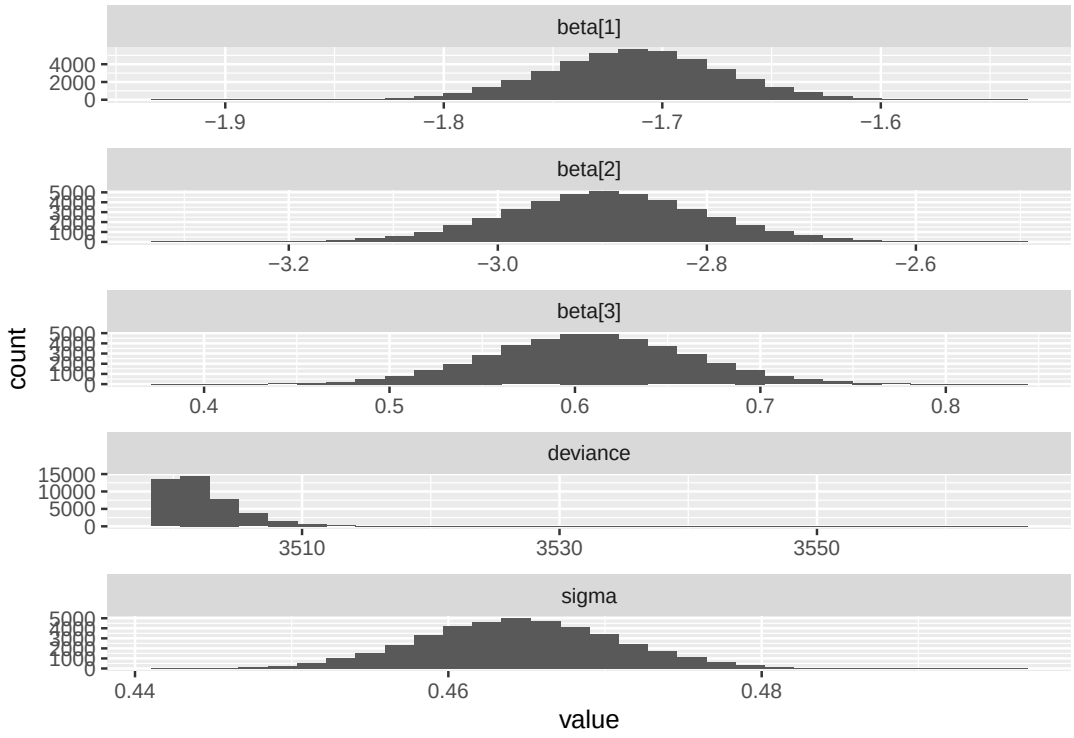
```
## [1] "0.381%"
```

```
# plot the fit model
x_vals = seq(min(x)-0.15,max(x)+0.15,0.01)
y_vals = beta[1] + beta[2]*x_vals + beta[3]*x_vals^2
plot(x,y,xlab = "B-V", ylab = "logL", main = "Model 3",
     pch = 21,cex = 1.2,bg = 'gray')
grid(nx=NULL, ny=NULL, lty="dotted", equilogs=FALSE)
lines(x_vals,y_vals,col="pink", lwd=3)
low.ci=mod3$BUGSoutput$summary[1:3,3]
high.ci=mod3$BUGSoutput$summary[1:3,7]
lower = low.ci[1] + low.ci[2]*x_vals + low.ci[3]*x_vals^2
upper = high.ci[1] + high.ci[2]*x_vals + high.ci[3]*x_vals^2
lines(x_vals,lower,col="red", lty="dashed")
lines(x_vals,upper,col="red", lty="dashed")
polygon(c(x_vals,rev(x_vals)),c(lower,rev(upper)),
        border=NA,col=adjustcolor("orangered", alpha.f = 0.20))
```

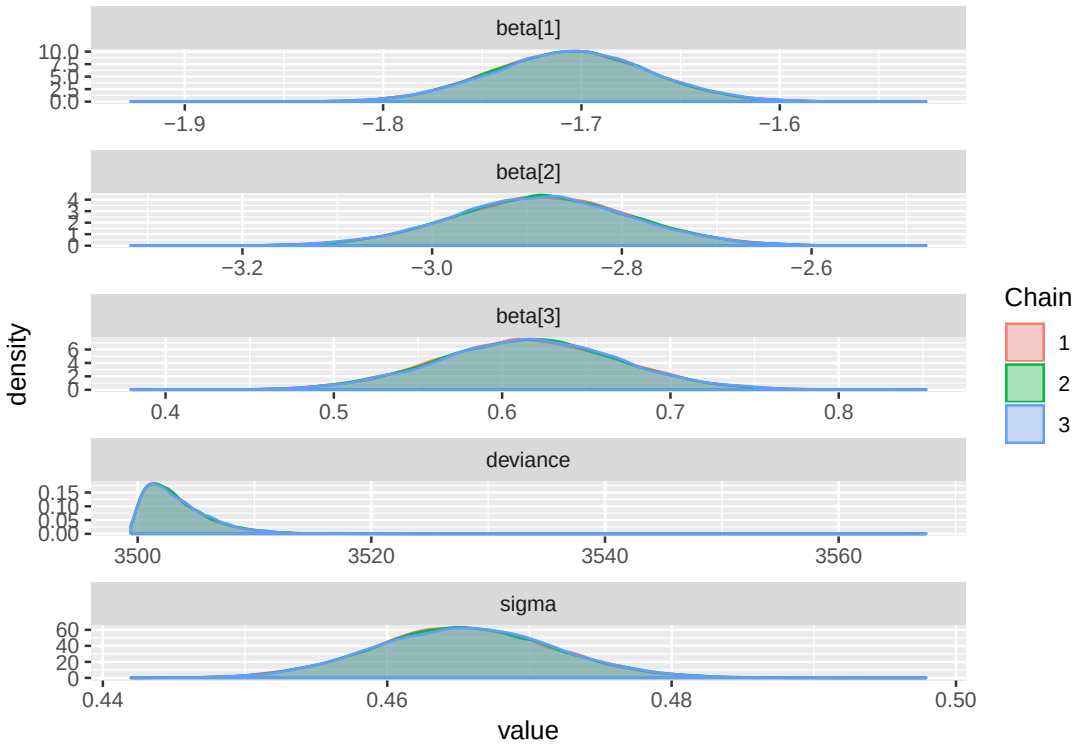


Below some more specific plots of the bayesian analysis:

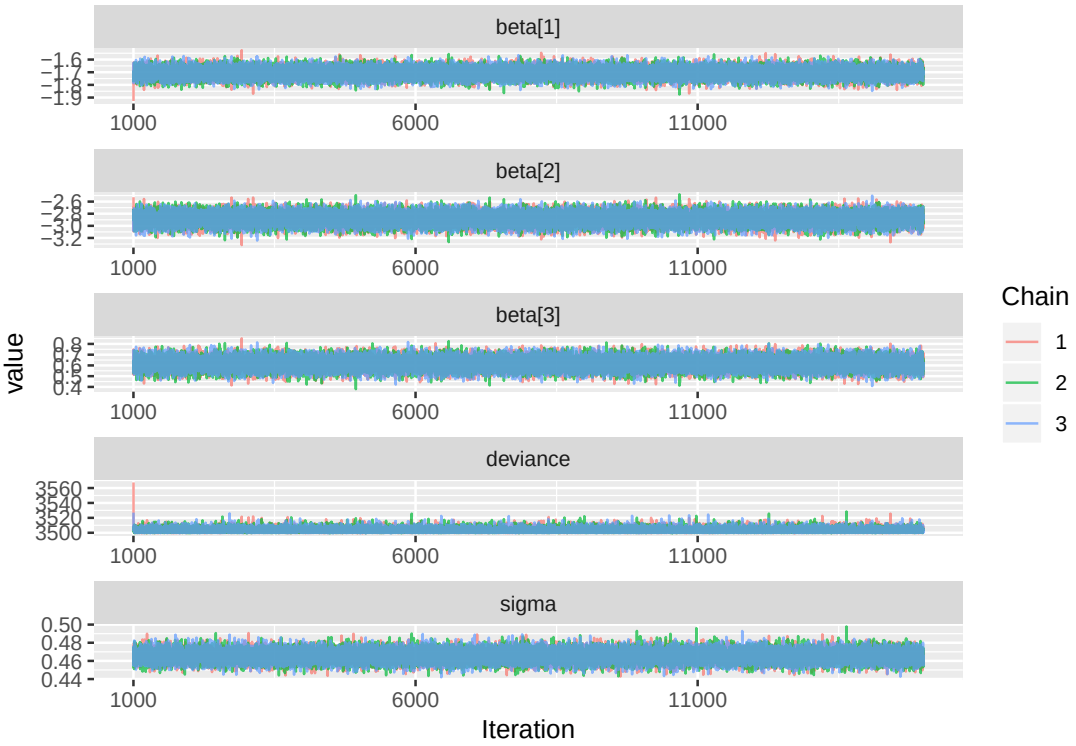
```
# use of the library ggmcmc that allows us to work with ggs object
mod.fit.ggs = ggs(as.mcmc(mod3))
ggs_histogram(mod.fit.ggs)
```



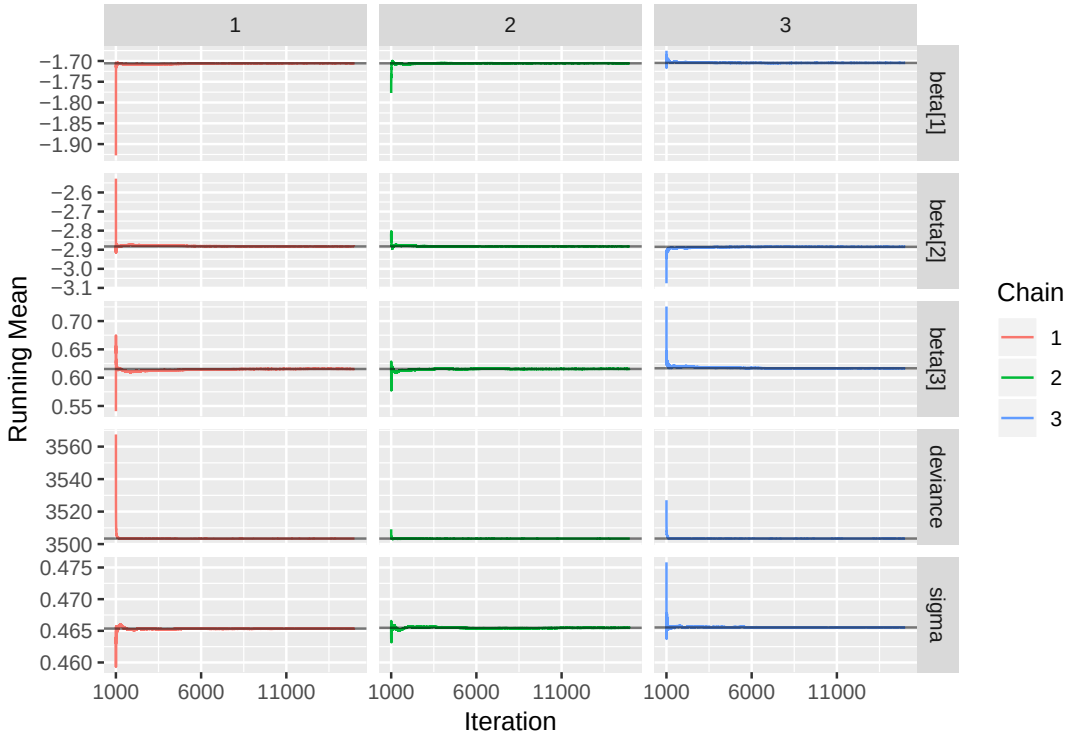

```
ggs_density(mod.fit.ggs)
```



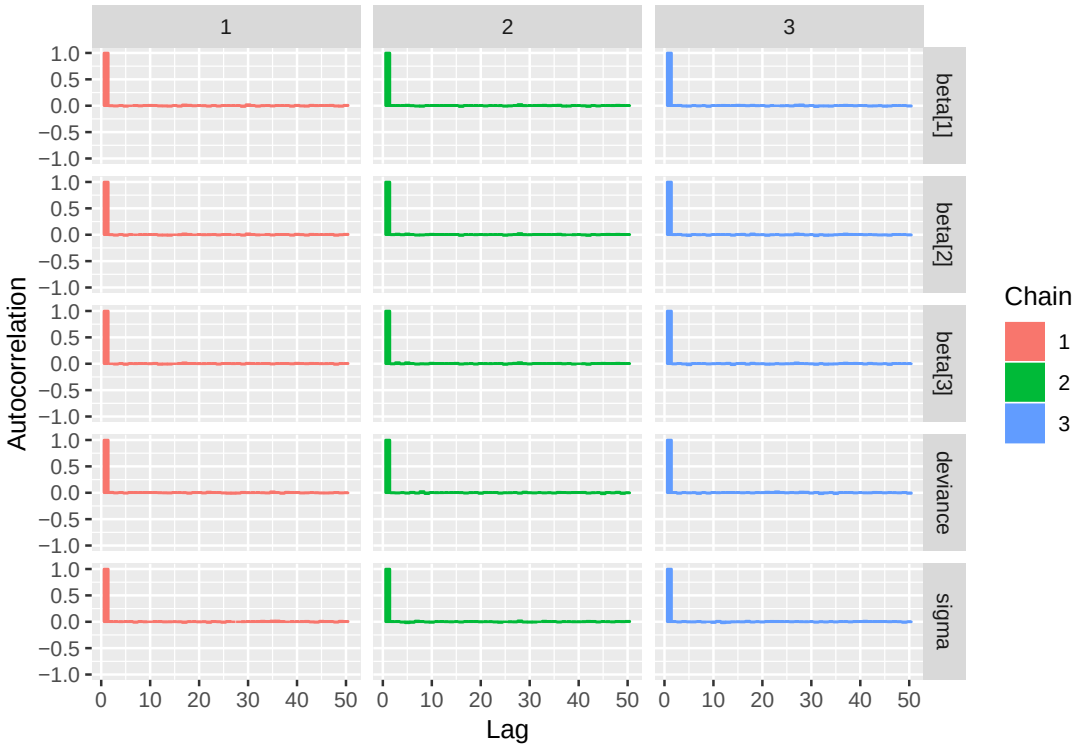

```
ggs_traceplot(mod.fit.ggs)
```




```
ggs_running(mod.fit.ggs)
```




```
ggs_autocorrelation(mod.fit.ggs)
```


1
2
3
beta[1]
beta[2]
beta[3]
deviance
sigma
Autocorrelation
1.0
0.5
0.0
−0.5
−1.0
0
10
20
30
40
50
Lag
Chain
1
2
3

Forth model - Exponential Model with gamma distributed variance

Now we will try to apply an exponential or super-polynomial model. The *mean* of Y_i can be expressed as:

$$\mu_i = \beta_0 + \beta_1 \gamma^{x_i}$$

where priors are:

$$\tau \sim Gamma(0.001, 0.001)$$

and

$$\beta_i \sim Normal(0, 0.001)$$

for every $i \in \{0, 1\}$

and

$$\gamma \sim Unif(0, 1)$$

The likelihood function can be derived in this way:

$$L(y_i|\mu_i, \tau, x_i) = L(y_i|\beta_0, \beta_1, \gamma, \tau, x_i) = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi\tau^2}} \exp\left\{\frac{-(y_i - \mu_i)^2}{2\tau^2}\right\} \propto \prod_{i=1}^{N} \exp\left\{\frac{-(y_i - \mu_i)^2}{2\tau^2}\right\}$$

$$\propto \exp\left\{-\frac{1}{2\tau^2}\sum_{i=1}^{N}(y_i - \mu_i)^2\right\} \propto \exp\left\{-\frac{1}{2\tau^2}\sum_{i=1}^{N}(y_i - (\beta_0 + \beta_1\gamma^{x_i}))^2\right\}$$

Full conditional for every β_i for $i \in \{0, 1\}$ can be derived as:

$$\pi(\beta_i|\beta_{j\neq i}, \tau, X, Y) = \frac{L(Y|\beta_0, \beta_1, \tau, X)\pi(\beta_i)}{f(Y)} \propto L(Y|\beta_0, \beta_1, \tau, X)\pi(\beta_i)$$

and

$$\pi(\gamma|\beta_0, \beta_1, \tau, X, Y) = \frac{L(Y|\beta_0, \beta_1, \gamma, \tau, X)\pi(\gamma)}{f(Y)} \propto L(Y|\beta_0, \beta_1, \gamma, \tau, X)\pi(\gamma)$$

and

$$\pi(\tau|\beta_0, \beta_1, \gamma, X, Y) = \frac{L(Y|\beta_0, \beta_1, \gamma, \tau, X)\pi(\tau)}{f(Y)} \propto L(Y|\beta_0, \beta_1, \gamma, \tau, X)\pi(\tau)$$

where

$$\pi(\beta_i) \sim Normal(0, 0.001)$$

and

$$\pi(\tau) \sim Gamma(0.001, 0.001)$$

and

$$\pi(\gamma) \sim Unif(0, 1)$$

Below is the code using RJAGS package in order to estimate the most appropriate factors for this model:

```r
#forth model -- exponential mu -- with gamma tau prior

JAGS.data = list(
  Y = y,
  x = x,
  N = N)


model = function()
{
  for(i in 1:N){
        #Likelihood
        Y[i] ~ dnorm(mu[i], tau)
        mu[i] = beta[1] + beta[2] *(gamma^x[i])
                }
  for(i in 1:2){
        #Prior for coefficients
        beta[i] ~ dnorm(0.0, 0.001)
  }
        #Prior for variance
        gamma ~ dunif(0,1)
        tau ~ dgamma(0.001,0.001)
        sigma = sqrt(1.0 / tau)  # precision

}


## Name the JAGS parameters
JAGS.params = c("beta","gamma","sigma")

## Define the starting values for JAGS
JAGS.inits = function(){
  list("beta" = rnorm(2, mean = 0.0, sd = 10.0),
       "tau" = rgamma(1, shape = 1.0, rate = 1.0),
       "gamma"= runif(1,min = 0,max=1))
}

mod4 = jags(data=JAGS.data, inits=JAGS.inits, JAGS.params, n.chains=3, n.iter=15000,
             n.burnin=1000, n.thin = 1, model.file=model, DIC=TRUE)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 2678
##    Unobserved stochastic nodes: 4
##    Total graph size: 8459
##
## Initializing model
```

```r
mod4
```

```
## Inference for Bugs model at "/tmp/RtmpXBvZB9/model9ac128d4741.txt", fit using jags,
##  3 chains, each with 15000 iterations (first 1000 discarded)
##  n.sims = 42000 iterations saved
##           mu.vect sd.vect     2.5%      25%      50%      75%    97.5%
## beta[1]    -6.686   0.334   -7.479   -6.854   -6.659   -6.470   -6.101
## beta[2]     4.988   0.303    4.468    4.791    4.962    5.142    5.714
```

```
## gamma       0.544   0.034    0.478    0.523    0.544    0.563    0.617
## sigma       0.466   0.006    0.454    0.462    0.466    0.471    0.479
## deviance 3515.387   3.003 3511.700 3513.156 3514.671 3516.883 3523.071
##           Rhat n.eff
## beta[1]  1.034    88
## beta[2]  1.033    89
## gamma    1.045    70
## sigma    1.001 42000
## deviance 1.006   480
##
## 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 = 4.5 and DIC = 3519.9
## DIC is an estimate of expected predictive error (lower deviance is better).
```

```
# model parameters
beta[1] = mod4$BUGSoutput$summary[,"mean"]["beta[1]"]
beta[2] = mod4$BUGSoutput$summary[,"mean"]["beta[2]"]
gamma = mod4$BUGSoutput$summary[,"mean"]["gamma"]

print(list(beta_0=round(beta[1],4),beta_1=round(beta[2],4),gamma=round(gamma,4)))
```

```
## $beta_0
## [1] -6.6859
##
## $beta_1
## [1] 4.988
##
## $gamma
##  gamma
## 0.5444
```

```
# predicted_data
y.fit = beta[1] + beta[2]*(gamma^x)

# DIC
DIC.mod4 = mod4$BUGSoutput$DIC
# rmse
rmse.mod4 = rmse(y, y.fit)


### marginal likelihood analysis


# model samples
samples_mod4 = mod4$BUGSoutput$sims.matrix


# evaluating the unnormalized posteriors on log scale
log_posterior_mod4 = function(data,samples) {

  sum(dnorm(data$Y, beta[1] + beta[2]*(gamma^data$x),(1/sigma)^2,log = TRUE)) +
    dnorm(beta[1],0,1e-3, log = TRUE) +
    dnorm(beta[2],0,1e-3, log = TRUE) +
    dunif(gamma,0,1,log = TRUE) +
    dgamma((1/sigma)^2,0.001, 0.001, log = TRUE)
```

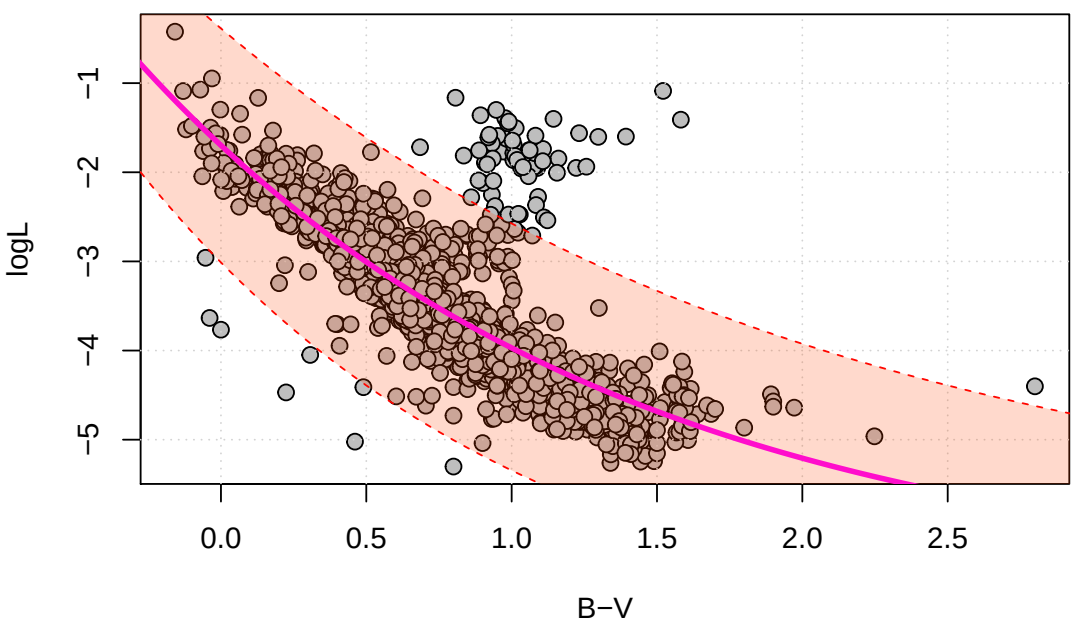
```
}

# lower and upper bounds
cn = colnames(samples_mod4)
cn = cn[cn != "deviance"]
lb_mod4 = rep(-Inf, length(cn))
ub_mod4 = rep(Inf, length(cn))
names(lb_mod4) = names(ub_mod4) = cn
lb_mod4[[ "sigma" ]] = 0

# compute log marginal likelihood via bridge sampling
mod4.bridge = bridge_sampler(samples = mod4, data = JAGS.data,
                             log_posterior = log_posterior_mod4, lb = lb_mod4,
                             ub = ub_mod4, silent = TRUE)
print(mod4.bridge)
```

```
## Bridge sampling estimate of the log marginal likelihood: -34797203
## Estimate obtained in 18 iteration(s) via method "normal".
```

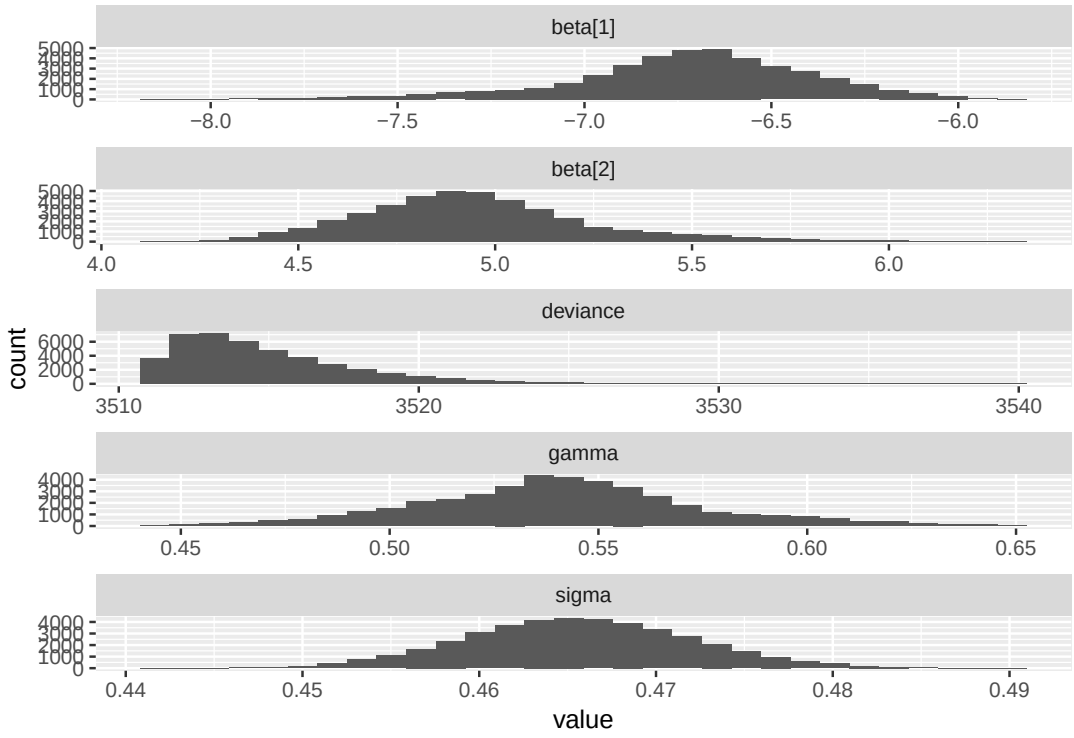
```
# compute percentage errors
mod4.error = error_measures(mod4.bridge)$percentage
error.mod4 = error_measures(mod4.bridge)$cv
print(mod4.error)
```

```
## [1] "1.95%"
```

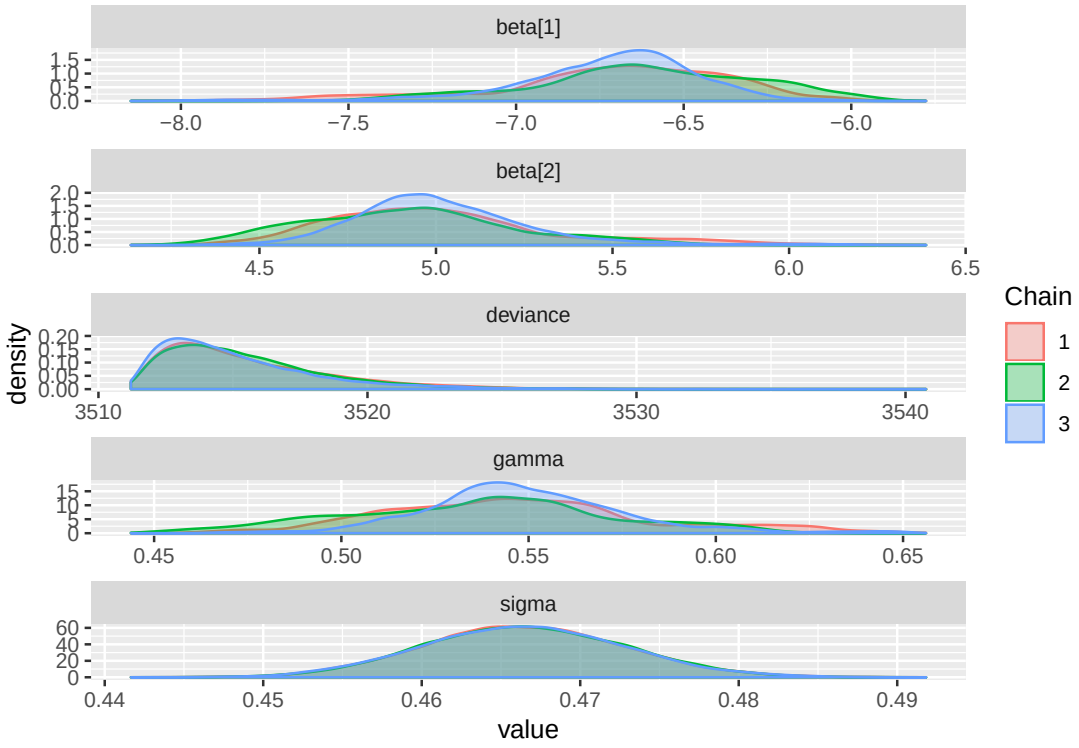
```
# plot fit model
x_vals = seq(min(x)-0.15,max(x)+0.15,0.01)
y_vals = beta[1] + beta[2]*(gamma^x_vals)
plot(x,y,xlab = "B-V", ylab = "logL",
     main = "Model 4",pch = 21,cex = 1.2,bg = 'gray')
grid(nx=NULL, ny=NULL, lty="dotted", equilogs=FALSE)
lines(x_vals,y_vals,col="magenta", lwd=3)
low.ci=mod4$BUGSoutput$summary[c(1:2,4),3]
high.ci=mod4$BUGSoutput$summary[c(1:2,4),7]
lower = low.ci[1] + low.ci[2]*(low.ci[3]^x_vals)
upper = high.ci[1] + high.ci[2]*(high.ci[3]^x_vals)
lines(x_vals,lower,col="red", lty="dashed")
lines(x_vals,upper,col="red", lty="dashed")
polygon(c(x_vals,rev(x_vals)),c(lower,rev(upper)),
        border=NA,col=adjustcolor("orangered", alpha.f = 0.20))
```



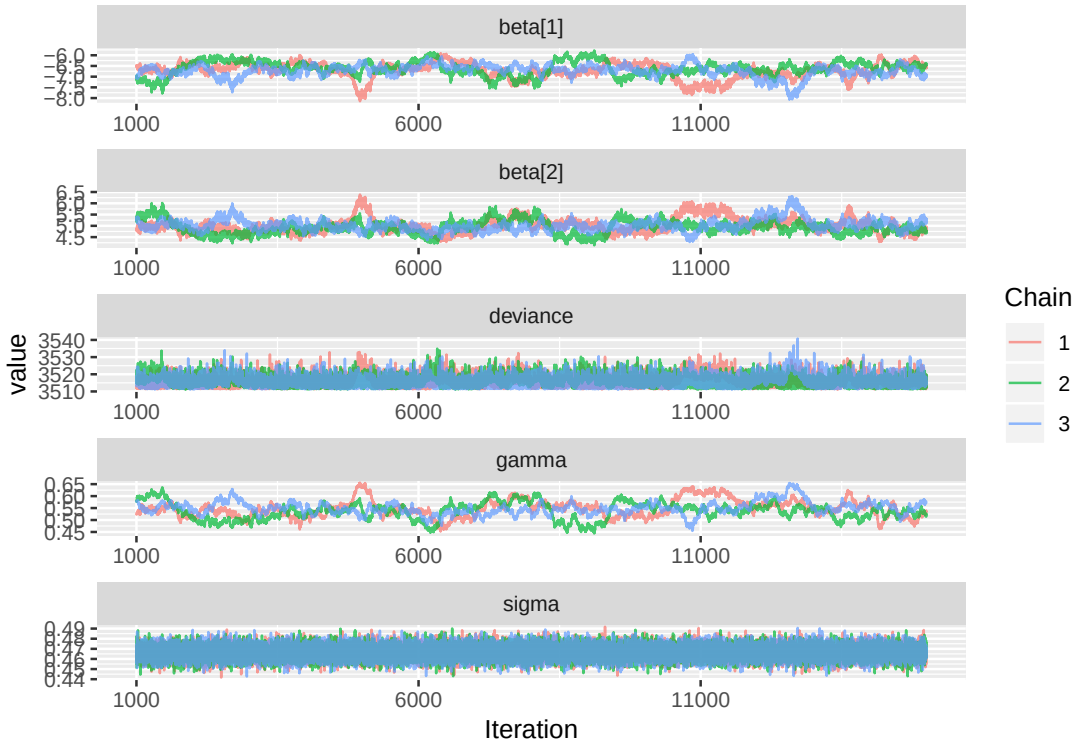

```
# use of the library ggmcmc that allows us to work with ggs object
mod.fit.ggs = ggs(as.mcmc(mod4))
ggs_histogram(mod.fit.ggs)
```



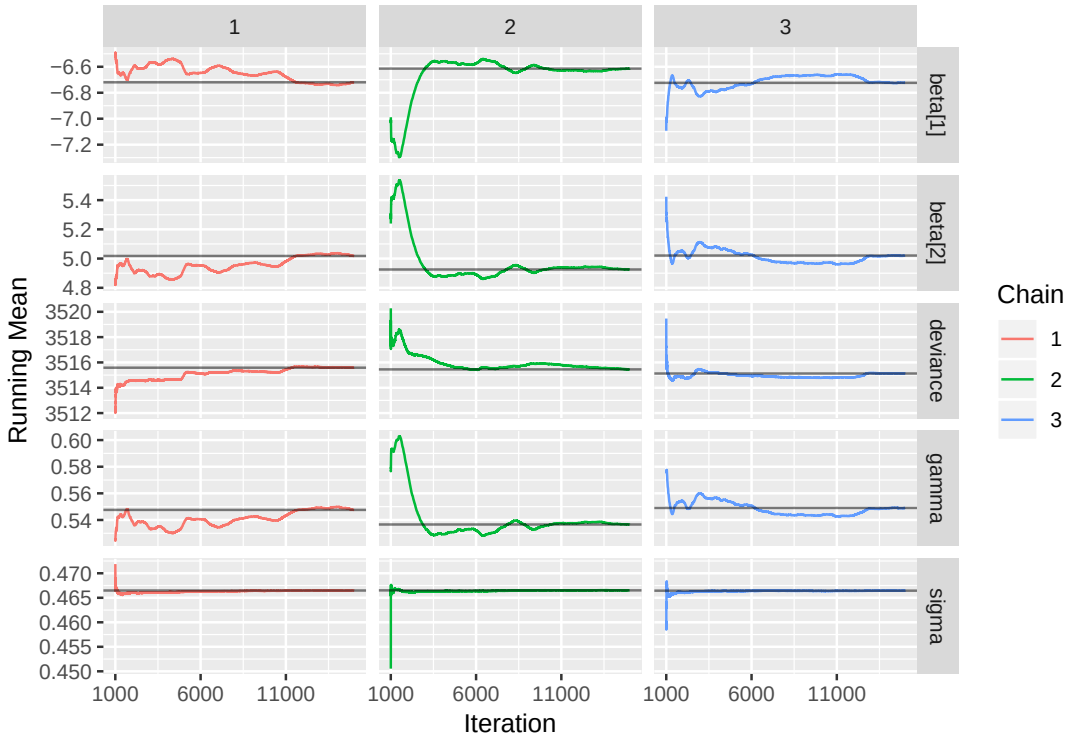

```
ggs_density(mod.fit.ggs)
```



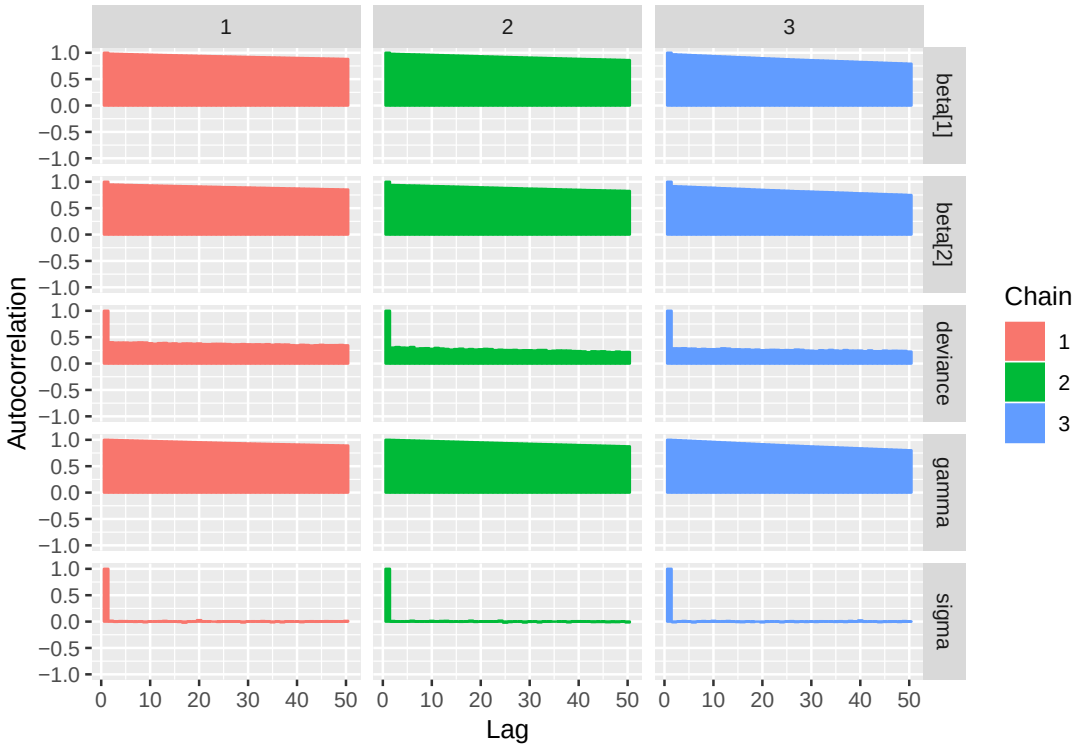

```
ggs_traceplot(mod.fit.ggs)
```




```
ggs_running(mod.fit.ggs)
```




```
ggs_autocorrelation(mod.fit.ggs)
```



Here we can see that convergence and stationarity is not that strong. Apparently, we need bigger amount of data in order to fit the model better and a greater number of k lags in order to show the independence in the autocorrelation plot.

Deviance Information Criterion, DIC

The Deviance Information Criterior (DIC) can be computed as:

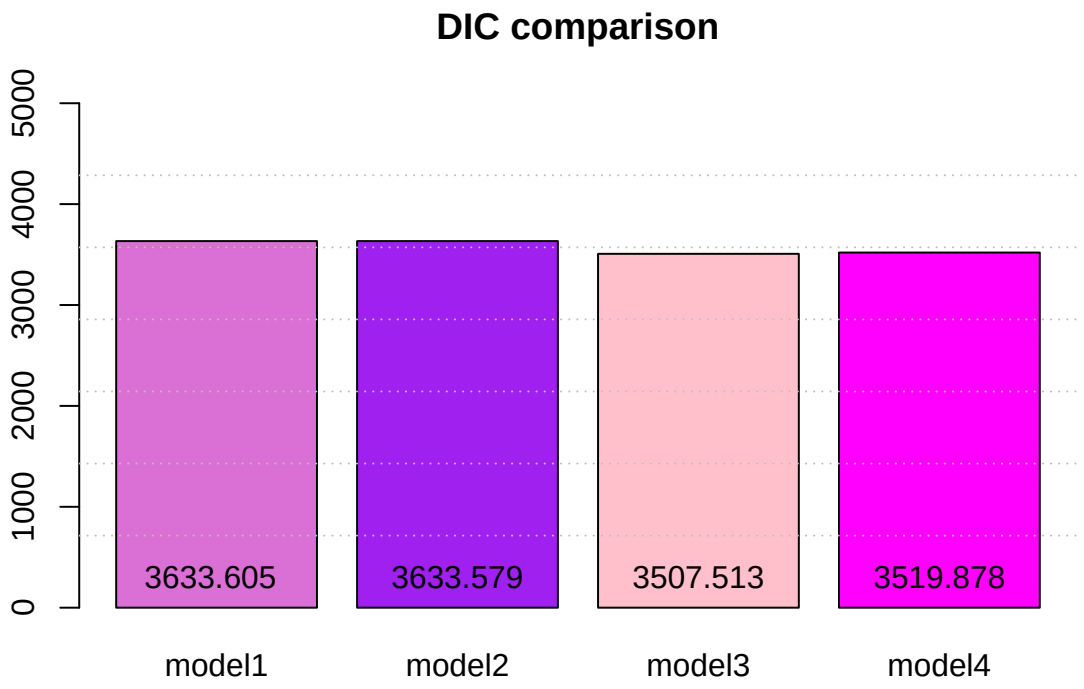
$$DIC = p_D + \bar{D} = \frac{1}{2}\widehat{var}(D(\theta)) - 2log(L(y|\theta)) + C$$

the smaller the DIC is for one model, the better...

```
DIC_array = cbind(DIC.mod1,DIC.mod2,DIC.mod3,DIC.mod4)
DIC_array
```

```
##      DIC.mod1 DIC.mod2 DIC.mod3 DIC.mod4
## [1,] 3633.605 3633.579 3507.513 3519.878
```

```
# comparison of DIC
DIC = c(DIC.mod1,DIC.mod2,DIC.mod3,DIC.mod4)
barplot(DIC, col=c("orchid", "purple", "pink", "magenta"), main="DIC comparison",
        names.arg = c("model1", "model2", "model3", "model4"), ylim=c(0,5000))
text(0.67, 300,round(DIC.mod1,3))
text(1.9, 300, round(DIC.mod2,3))
text(3.1, 300, round(DIC.mod3,3))
text(4.3, 300, round(DIC.mod4,3))
grid(nx = 0, ny = 7, col = "grey", lty = "dotted",
     lwd = par("lwd"), equilogs = TRUE)
```



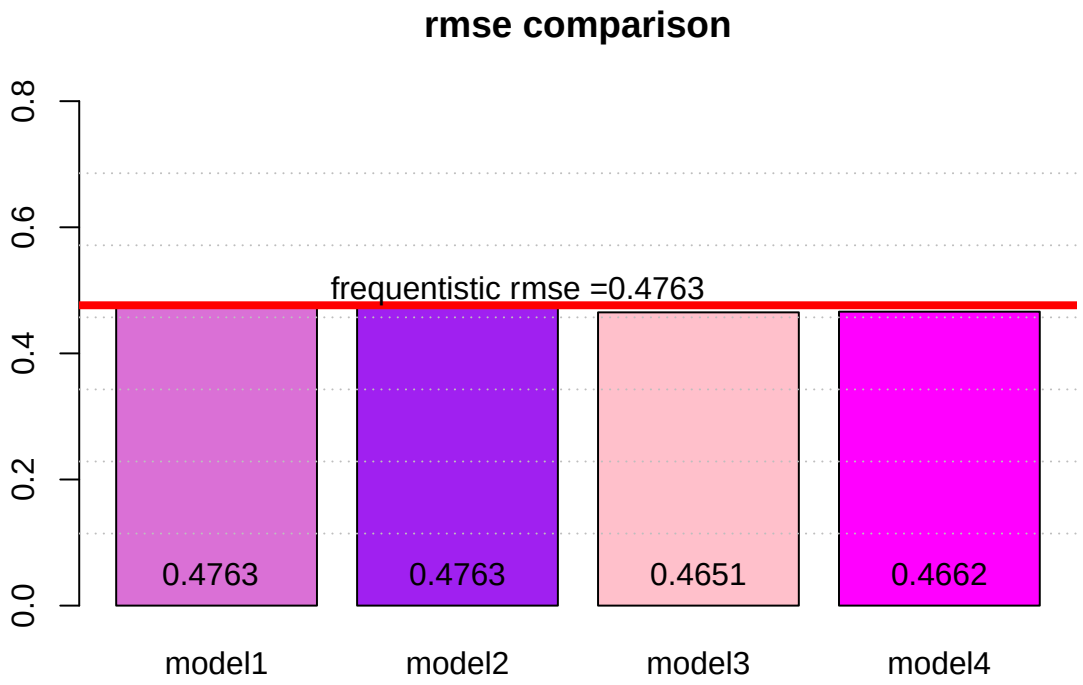
Generally all the models present similar DIC with the *third* model thought presents values of DIC lower respect to the rest of the models.

RMSE Comparison between models

```
rmse_array = cbind(rmse.mod1,rmse.mod2,rmse.mod3,rmse.mod4)
rmse_array
```

```
##      rmse.mod1 rmse.mod2 rmse.mod3 rmse.mod4
## [1,] 0.4763306 0.4763306 0.4650653 0.4661702
```

```r
# comparison of RMSE
rmse = c(rmse.mod1,rmse.mod2,rmse.mod3,rmse.mod4)
barplot(rmse, col=c("orchid", "purple", "pink", "magenta"), main="rmse comparison",
        names.arg = c("model1", "model2", "model3", "model4"), ylim=c(0,0.8))
abline(h=rmse.lm, col="red", lwd=4)
text(0.67, 0.05,round(rmse.mod1,4))
text(1.9, 0.05, round(rmse.mod2,4))
text(3.1, 0.05, round(rmse.mod3,4))
text(4.3, 0.05, round(rmse.mod4,4))
text(2.2, 0.5, paste0("frequentistic rmse =", round(rmse.lm, 4)))
grid(nx = 0, ny = 7, col = "grey", lty = "dotted",
     lwd = par("lwd"), equilogs = TRUE)
```



Similarly we can see that the rmse of all the models fluctuate in the same levels with the frequentistic aprroach we followed in the beginning, with that of the third polynomial model be the smallest by nominal difference. Nevertheless, taking into consideration the previous DIC analysis and the autocorrelation and traceplots of each model respectively we can definitely exclude the *forth* exponential model since it's the less stationary by far. Moreover, regardless that the *third* model shows the smallest rmse we can not be very confident on that model since it presents a high range credible interval (the longest) respect to the two first bayesian models, something that can be shown very clearly in the fit plot curve on the illustation of the model above. It is necessary that more criteria should be presented in order to have a more clear decision.

Bayesian Factor and percentage errors between models

For that reason we will resort to marginal likelihood and the *Bayesian Factor* comparison between the four models.

Let's assume that we have 2 models M_1 and M_2

The posterior odds between 2 alternative models for a given dataset D can be computed as:

$$K = \frac{\pi(D|M_1)}{\pi(D|M_2)} = \frac{\int \pi(\theta_1|M_1)\pi(D|\theta_1, M_1)d\theta_1}{\int \pi(\theta_2|M_2)\pi(D|\theta_2, M_2)d\theta_2} = \frac{\pi(M_1|D)\pi(M_2)}{\pi(M_2|D)\pi(M_1)}$$

A value of $K > 1$ means that M_1 is more strongly supported by the data under consideration than M_2 and that model can be better generalized on unseed data.

We can exclude the ratio of the posterior odds to the prior odds:

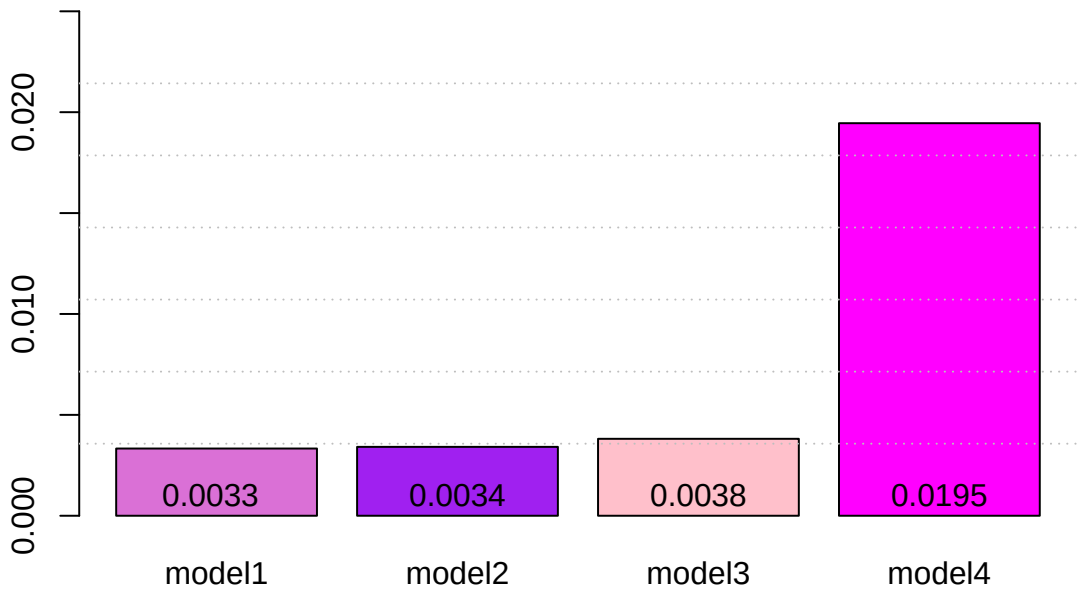
$$BF = \frac{\pi(M_1|D)}{\pi(M_2|D)}$$

Where $BF > 1$ supports for M_1 over M_2

```
error_array = cbind(error.mod1,error.mod2,error.mod3,error.mod4)
error_array
```

```
##       error.mod1  error.mod2  error.mod3 error.mod4
## [1,] 0.003331096 0.003414133 0.003813287 0.01945506
```

```
# comparison of percentage error
error = c(error.mod1,error.mod2,error.mod3,error.mod4)
barplot(error, col=c("orchid", "purple", "pink", "magenta"),
        main="error percentage comparison",
        names.arg = c("model1", "model2", "model3", "model4"),
        ylim=c(0,0.025))
text(0.67, 0.001,round(error.mod1,4))
text(1.9, 0.001, round(error.mod2,4))
text(3.1, 0.001, round(error.mod3,4))
text(4.3, 0.001, round(error.mod4,4))
grid(nx = 0, ny = 7, col = "grey", lty = "dotted",
     lwd = par("lwd"), equilogs = TRUE)
```




```
# compute Bayes factor


## between model 2 and model 1
BF01 <- bf(mod2.bridge, mod1.bridge,log=TRUE)
print(BF01)
```

```
## Estimated log Bayes factor in favor of mod2.bridge over mod1.bridge: 114.24851
```

```r
# compute posterior model probabilities (assuming equal prior model probabilities)
post1 <- post_prob(mod2.bridge, mod1.bridge,log=TRUE)
```

```
## Warning: Objects not of class 'bridge' or 'bridge_list' are ignored.
```

```r
print(post1)
```

```
##  mod2.bridge  mod1.bridge
## 1.000000e+00 2.412691e-50
```

```r
## between model 2 and model 3
BF02 <- bf(mod2.bridge, mod3.bridge,log=TRUE)
print(BF02)
```

```
## Estimated log Bayes factor in favor of mod2.bridge over mod3.bridge: 1941361.26233
```

```r
## between model 2 and model 4
BF03 <- bf(mod2.bridge, mod4.bridge,log=TRUE)
print(BF03)
```

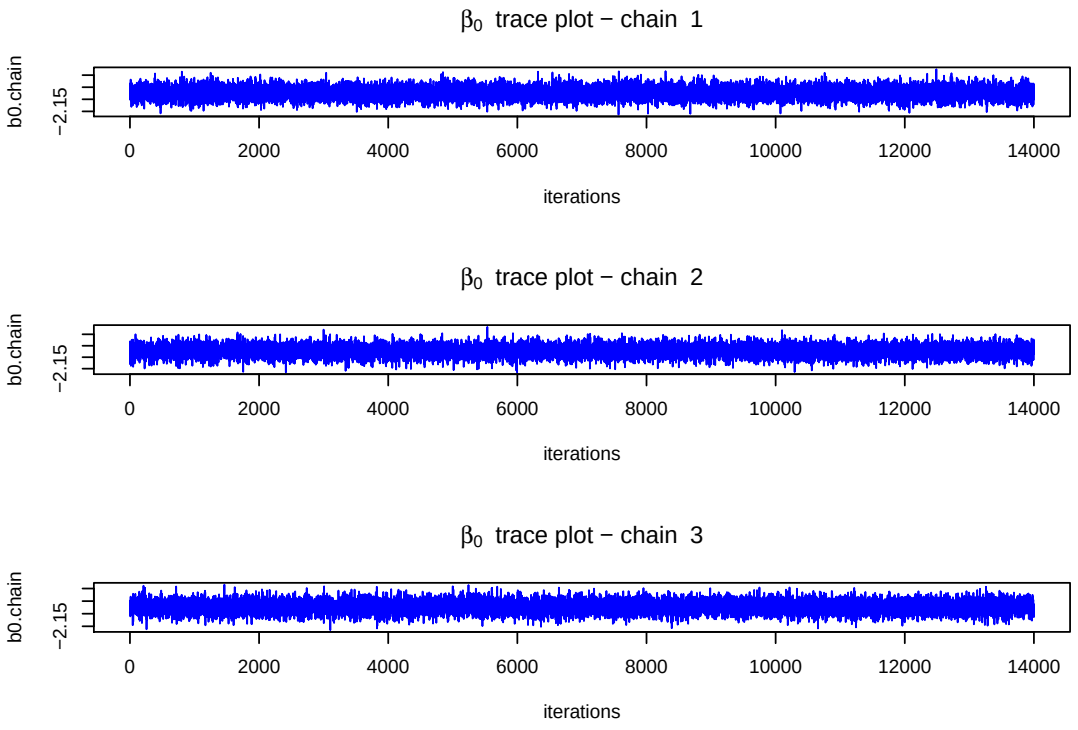
```
## Estimated log Bayes factor in favor of mod2.bridge over mod4.bridge: 30931508.57280
```

Taking everything into consideration we can choose **model_2** as the more suitable and consistent for our dataset. We should also mention some strong similarity between the models of that following the frequentistic linear approach and the final Bayesian model.

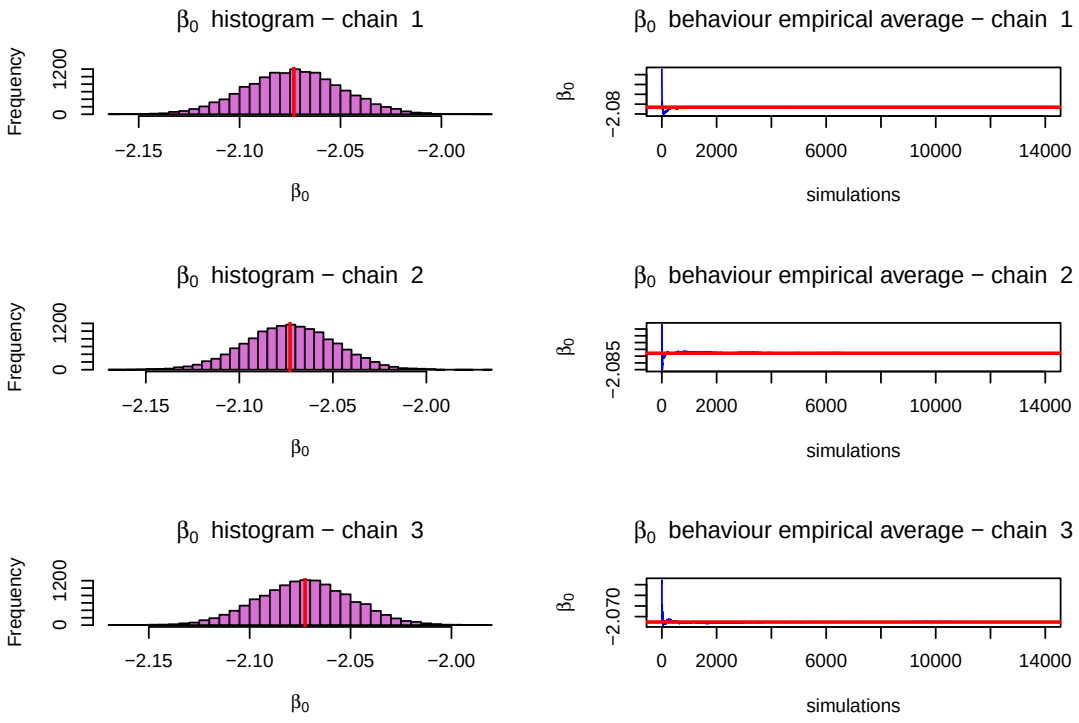
Analysis based on the second - linear model

Below we will see some more elaborated analysis of the final model we have chosen:

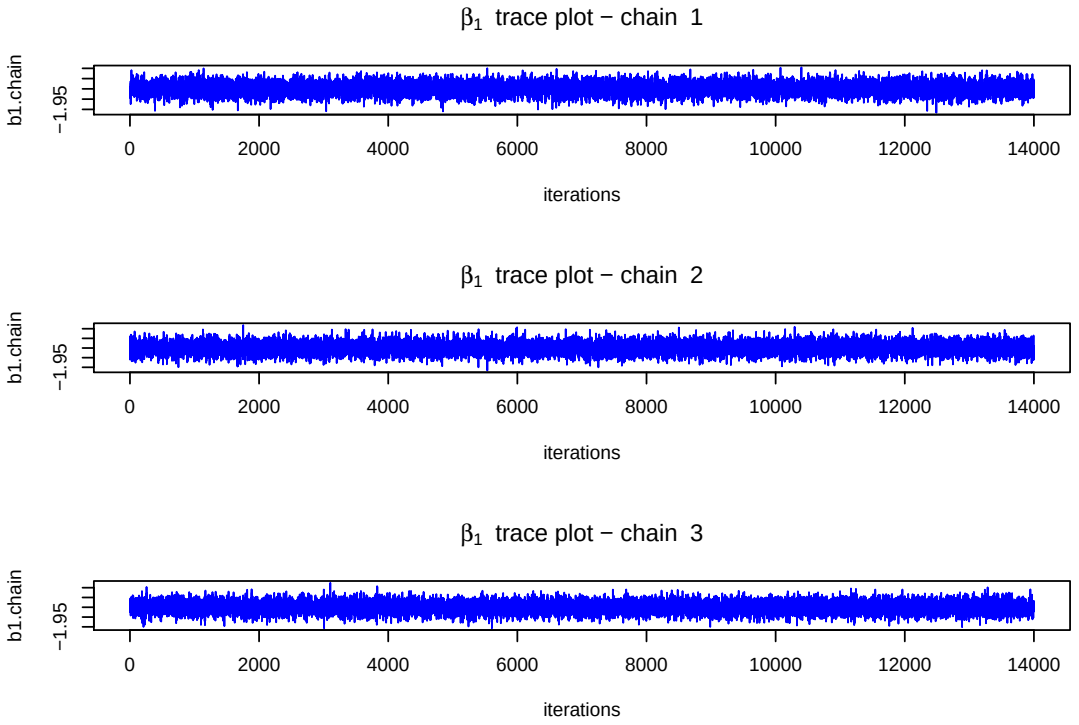
```
# let's see every parameter individually through traceplot, histogram, behaviour of
# the empirical mean and approximation error

# beta_0
par(mfrow=c(3,1))
for(i in 1:3){
  b0.chain = mod2$BUGSoutput$sims.array[,i,"beta[1]"]
  plot(b0.chain, xlab = "iterations",
       main=bquote(beta[0] ~ " trace plot - chain " ~ .(i)),type="l",col='blue')
}
```




```
par(mfrow=c(3,2))
for(i in 1:3){
  b0.chain = mod2$BUGSoutput$sims.array[,i,"beta[1]"]
  hist(b0.chain, main=bquote(beta[0] ~ " histogram - chain " ~ .(i)),
       xlab = expression(beta[0]),breaks=30,col='orchid')
  abline(v=mean(b0.chain), col="red", lwd=2)
  plot(cumsum(b0.chain)/(1:length(b0.chain)), type="l", ylab=expression(beta[0]),
     main=bquote(beta[0] ~ " behaviour empirical average - chain " ~ .(i)),
     xlab="simulations",col="blue")
  abline(h=mean(b0.chain), col="red", lwd=2)
}
```

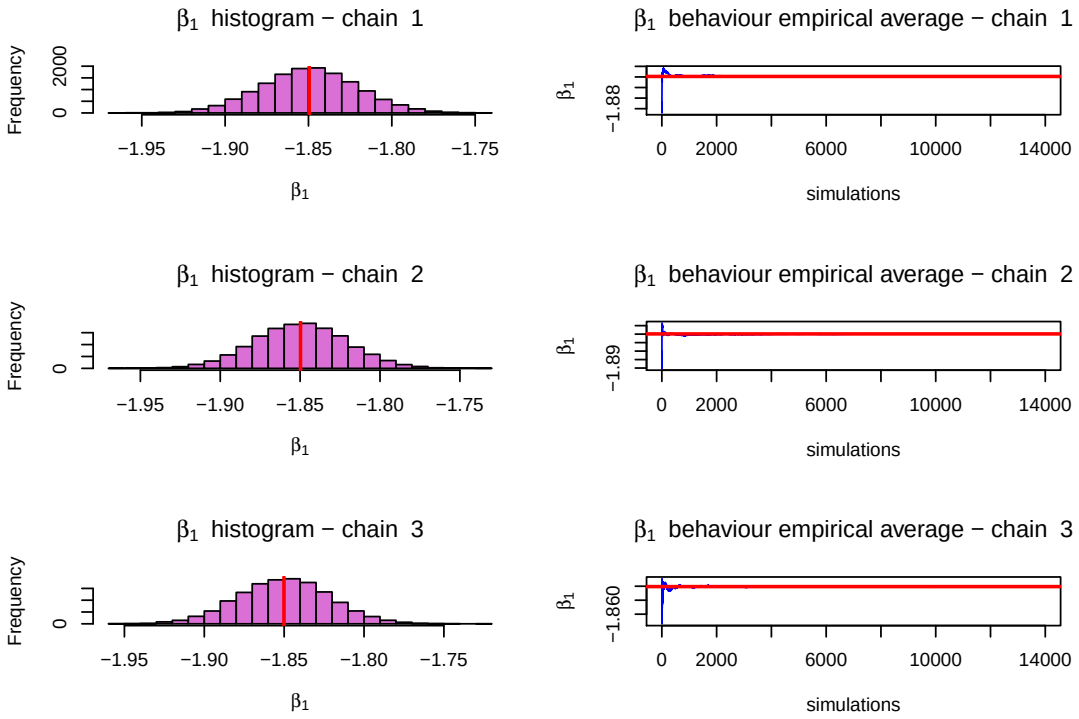



```
# beta_1
par(mfrow=c(3,1))
for(i in 1:3){
  b1.chain = mod2$BUGSoutput$sims.array[,i,"beta[2]"]
  plot(b1.chain, xlab = "iterations",
       main=bquote(beta[1] ~ " trace plot - chain " ~ .(i)),type="l",col='blue')
}
```

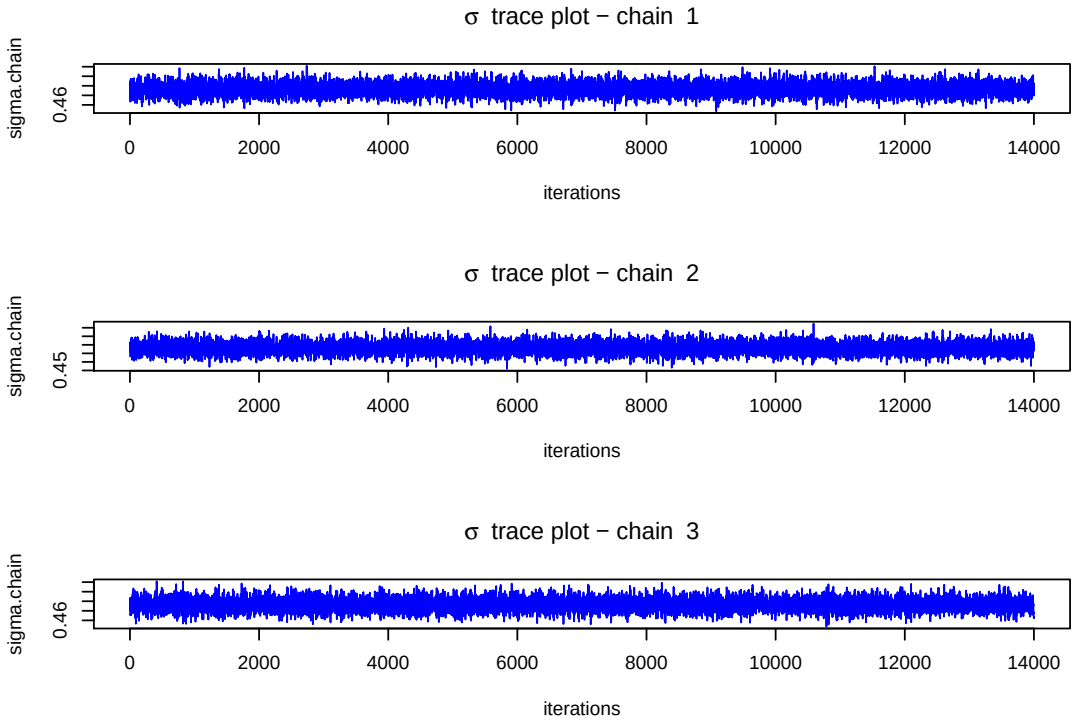



```
par(mfrow=c(3,2))
for(i in 1:3){
  b1.chain = mod2$BUGSoutput$sims.array[,i,"beta[2]"]
  hist(b1.chain, main=bquote(beta[1] ~ " histogram - chain " ~ .(i)),
```

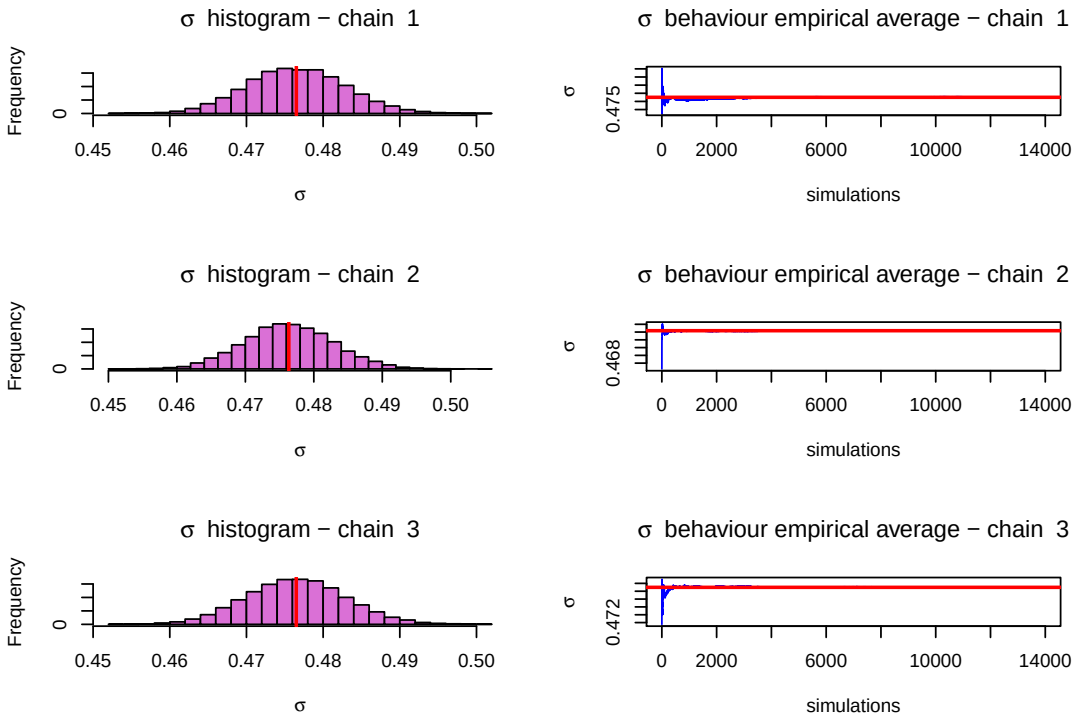
```
      xlab = expression(beta[1]),breaks=30,col='orchid')
  abline(v=mean(b1.chain), col="red", lwd=2)
  plot(cumsum(b1.chain)/(1:length(b1.chain)), type="l", ylab=expression(beta[1]),
     main=bquote(beta[1] ~ " behaviour empirical average - chain " ~ .(i)),
     xlab="simulations",col="blue")
  abline(h=mean(b1.chain), col="red", lwd=2)
}
```




```
# sigma
par(mfrow=c(3,1))
for(i in 1:3){
  sigma.chain = mod2$BUGSoutput$sims.array[,i,"sigma"]
  plot(sigma.chain, xlab = "iterations",
       main=bquote(sigma ~ " trace plot - chain " ~ .(i)),type="l",col='blue')
}
```




```
par(mfrow=c(3,2))
for(i in 1:3){
  sigma.chain = mod2$BUGSoutput$sims.array[,i,"sigma"]
  hist(sigma.chain, main=bquote(sigma ~ " histogram - chain " ~ .(i)),
       xlab = expression(sigma),breaks=30,col='orchid')
  abline(v=mean(sigma.chain), col="red", lwd=2)
  plot(cumsum(sigma.chain)/(1:length(sigma.chain)), type="l", ylab=expression(sigma),
     main=bquote(sigma ~ " behaviour empirical average - chain " ~ .(i)),
     xlab="simulations",col="blue")
  abline(h=mean(sigma.chain), col="red", lwd=2)
}
```

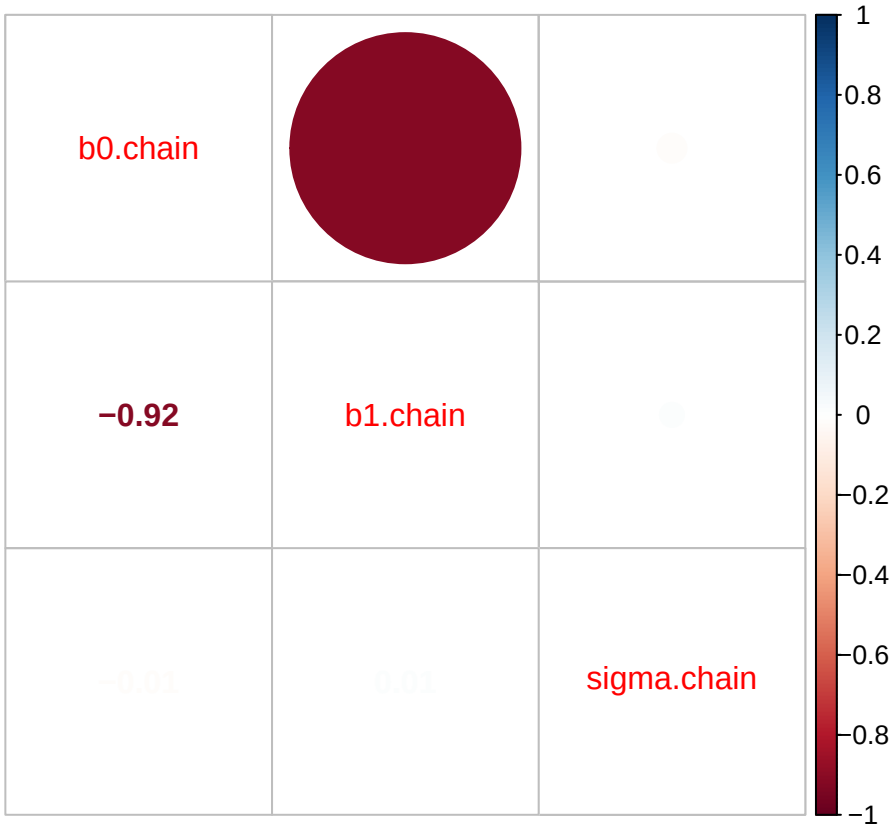


Also some correlation plot between the parameters of the model could be useful:

```
# correlation
mod.parameters = cbind(b0.chain, b1.chain, sigma.chain)
correlation=cor(mod.parameters)
correlation
```

```
##                b0.chain    b1.chain sigma.chain
## b0.chain     1.00000000 -0.91941150 -0.01454019
## b1.chain    -0.91941150  1.00000000  0.01003279
## sigma.chain -0.01454019  0.01003279  1.00000000
```

```
corrplot.mixed(correlation)
```



From the correlation plot above we can see some strong (negative-inversely) correlation between the β_0 and the β_1 chains.

Some Predictions on the final Bayesian model

Below we can see some predictions based on the choosen final linear model.

```
prediction = function(x){
  y.pred = rep(NA, 14000)
  for(i in 1:14000){
    y.pred[i] = b0.chain[i] + b1.chain[i]*x
  }
  return(mean(y.pred))
}


# mean prediction of logL with B-V equals to -0.3
prediction(-0.3)
```

```
## [1] -1.517513
```

```
# mean prediction of logL with B-V equals to 0.8
prediction(0.8)
```

```
## [1] -3.552666
```

```
# mean prediction of logL with B-V equals to 1.2
prediction(1.2)
```

```
## [1] -4.292721
```

```
# mean prediction of logL with B-V equals to 1.8
prediction(1.8)
```

```
## [1] -5.402804
```

```
# mean prediction of logL with B-V equals to 2.4
prediction(2.4)
```

```
## [1] -6.512888
```

References

- Hipparcos star dataset (https://astrostatistics.psu.edu/datasets/HIP_star.html)
- Hertzsprung–Russell diagram (https://en.wikipedia.org/wiki/Hertzsprung%E2%80%93Russell_diagram)
- (Chapters 10-11) Bayesian Modeling Using WinBUGS - Ioannis Ntzoufras
- A First Course in Bayesian Statistical Methods - Peter D. Hoff


```
## Last update by JT: Sat Jul 20 19:14:04 2019
```