

# Data Analysis Project - Bayesian Statistics

Ken Wood

2023-08-03

## Executive Summary

The purpose of this project was to fit a series of regression models to a dataset containing housing features and a corresponding sale price as the response variable. Three models were constructed using both R and JAGS. One of the JAGS models used 3 features to predict sale prices while the final iteration used 4 features. A number of evaluation metrics (including the deviation information criterion (DIC)) were generated to gauge the accuracy of each model. It was determined that incorporating the 4th feature reduced the DIC and, hence, was a better for the data.

## Introduction

The Ames Housing dataset, which is available on Kaggle.com, was compiled by Dean De Cock for use in data science education. It's an incredible dataset resource for data scientists and statisticians looking for a modernized and expanded version of the often-cited Boston Housing dataset.

The subject dataset contains 79 explanatory variables describing (almost) every aspect of residential homes in Ames, Iowa, along with the sale price of each home. **For the purposes of this project, we will construct two models that leverage a subset (specifically, 3-4) of the most informative explanatory variables to predict sale prices for homes given their features.**

The features to be included are:

- **LotArea** - Lot size in square feet
- **OverallCond** - Rates the overall condition of the house (10=Excellent, 1=Very Poor)
- **GrLivArea** - Above grade (ground) living area in square feet

We will create an additional iteration of the model that incorporates the feature **TotRmsAbvGrd**, the total number of rooms above grade.

## Load and Explore the Dataset

```
dat = read.csv(file="data_files/housing-price-data.csv", header=TRUE)

# Subset the raw data to features of interest and the response variable, SalePrice
dat1=dat[,c("LotArea", "HouseStyle", "OverallCond", "GrLivArea", "SalePrice")]
head(dat1)

##   LotArea HouseStyle OverallCond GrLivArea SalePrice
## 1     8450      2Story         5     1710   208500
## 2     9600      1Story         8     1262   181500
## 3    11250      2Story         5     1786   223500
## 4     9550      2Story         5     1717   140000
## 5    14260      2Story         5     2198   250000
## 6    14115     1.5Fin         5     1362   143000
```

```

str(dat1)

## 'data.frame': 1460 obs. of 5 variables:
## $ LotArea    : int 8450 9600 11250 9550 14260 14115 10084 10382 6120 7420 ...
## $ HouseStyle : chr "2Story" "1Story" "2Story" "2Story" ...
## $ OverallCond: int 5 8 5 5 5 5 5 6 5 6 ...
## $ GrLivArea  : int 1710 1262 1786 1717 2198 1362 1694 2090 1774 1077 ...
## $ SalePrice   : int 208500 181500 223500 140000 250000 143000 307000 200000 129900 118000 ...

```

We will need to code the categorical variable `HouseStyle`.

```
dat1$HouseStyle_coded = factor(dat1$HouseStyle)
```

```
newdat1 = dat1[,c("LotArea", "HouseStyle_coded", "OverallCond", "GrLivArea", "SalePrice")]
```

```
head(newdat1)
```

```

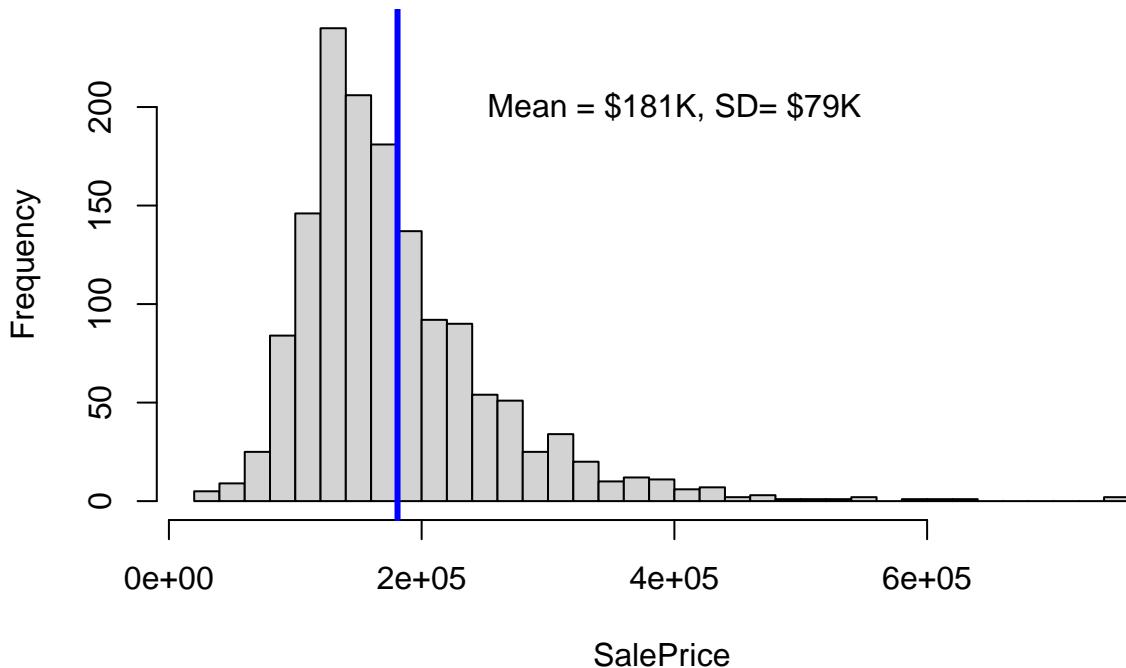
##   LotArea HouseStyle_coded OverallCond GrLivArea SalePrice
## 1    8450           2Story         5     1710  208500
## 2    9600           1Story         8     1262  181500
## 3   11250           2Story         5     1786  223500
## 4    9550           2Story         5     1717  140000
## 5   14260           2Story         5     2198  250000
## 6   14115          1.5Fin         5     1362  143000

```

Let's examine the distribution of the response variable, `SalePrice`

```
hist(newdat1$SalePrice, breaks=50, main="Histogram of SalePrice", xlab = "SalePrice")
abline(v = mean(newdat1$SalePrice), col='blue', lwd = 3)
#add label to mean vertical line
text(x=4e5, y=2e2, 'Mean = $181K, SD= $79K')
```

**Histogram of SalePrice**



It appears that the distribution of `SalePrice` is somewhat normal with a mean and standard deviation

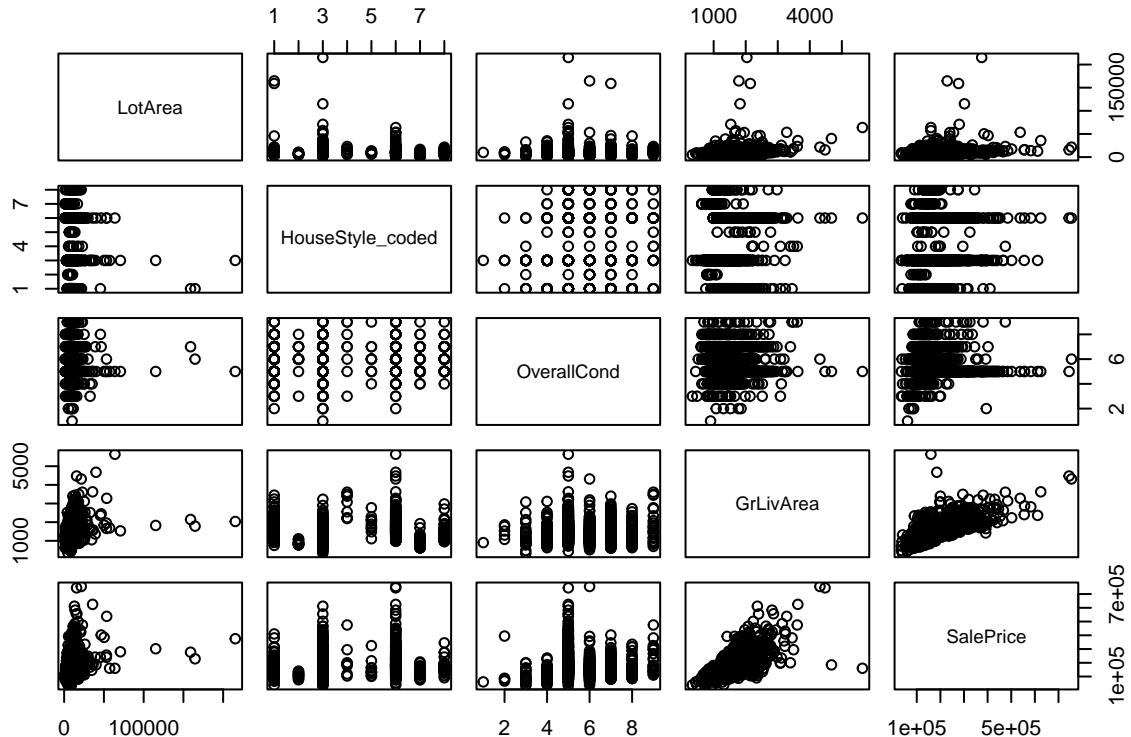
calculated as follows:

```
mean(newdat1$SalePrice)  
## [1] 180921.2  
sd(newdat1$SalePrice)  
## [1] 79442.5
```

Now, let's look at the relationships among the features and their distributions:

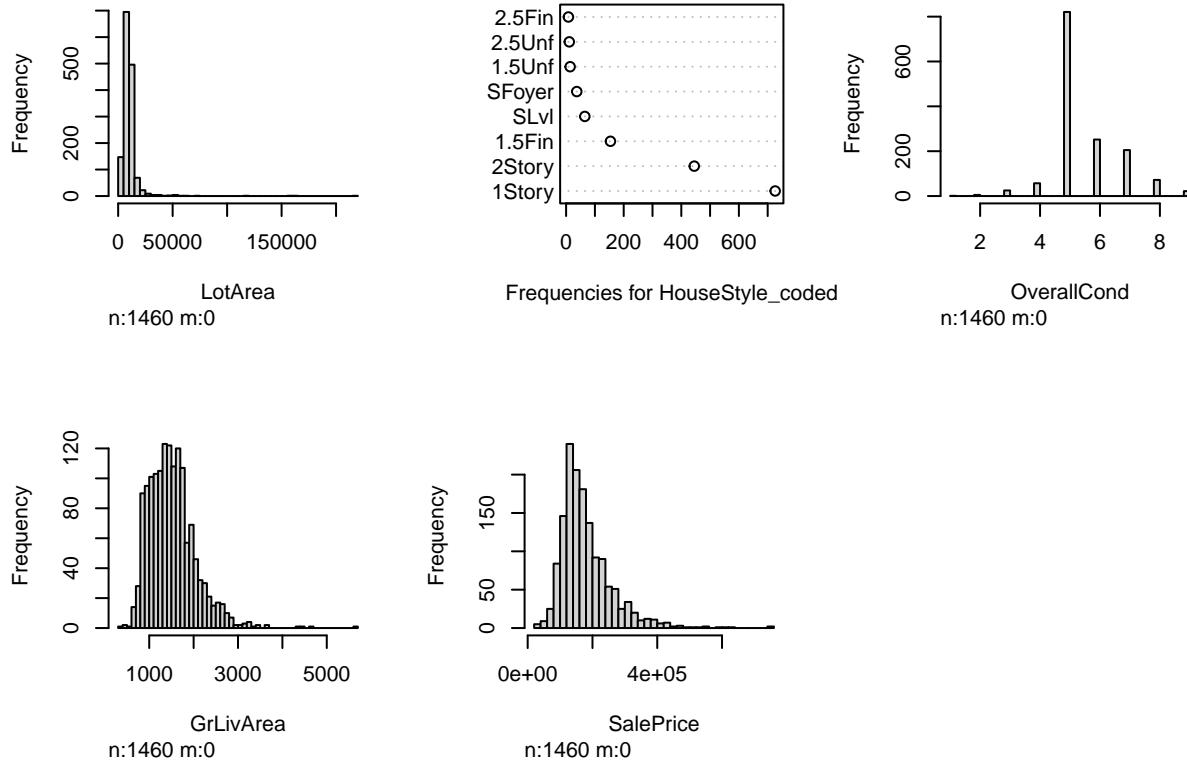
```
library(Hmisc)
```

```
##  
## Attaching package: 'Hmisc'  
## The following objects are masked from 'package:base':  
##  
##     format.pval, units  
pairs(newdat1)
```



It looks like most, if not all, of the numeric features have a somewhat linear relationship with SalePrice.

```
hist.data.frame(newdat1)
```



It also appears that all of our non-categorical variables (including the response variable, `SalePrice`) have a somewhat normal distribution.

## Postulate a Model

Let's start off by postulating a linear model of the descriptor features.

```
mod_linear = lm(SalePrice ~ LotArea + HouseStyle_coded + OverallCond + GrLivArea, data=newdat1)
summary(mod_linear)
```

```
##
## Call:
## lm(formula = SalePrice ~ LotArea + HouseStyle_coded + OverallCond +
##     GrLivArea, data = newdat1)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -549116  -22005   -1040   21462  300921
##
## Coefficients:
##                               Estimate Std. Error t value Pr(>|t|)    
## (Intercept)             -7.391e+04  1.004e+04 -7.364 2.97e-13 ***
## LotArea                  3.781e-01  1.402e-01   2.697 0.007069 **  
## HouseStyle_coded1.5Unf  5.377e+04  1.429e+04   3.762 0.000175 *** 
## HouseStyle_coded1Story  6.702e+04  4.621e+03  14.502 < 2e-16 ***
## HouseStyle_coded2.5Fin -8.912e+04  1.888e+04 -4.721 2.57e-06 ***
## HouseStyle_coded2.5Unf -2.873e+04  1.587e+04 -1.811 0.070345 .  
## HouseStyle_coded2Story  2.717e+04  4.885e+03   5.562 3.18e-08 ***
## HouseStyle_codedSFoyer 6.941e+04  9.463e+03   7.335 3.68e-13 ***
## HouseStyle_codedSLvl   4.864e+04  7.523e+03   6.465 1.38e-10 ***
```

```

## OverallCond      2.121e+03  1.221e+03   1.738  0.082451 .
## GrLivArea       1.278e+02  3.227e+00  39.612  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 50680 on 1449 degrees of freedom
## Multiple R-squared:  0.5958, Adjusted R-squared:  0.593
## F-statistic: 213.6 on 10 and 1449 DF,  p-value: < 2.2e-16

```

We now create a hierarchical model in JAGS with the following configuration:

$$y_i \sim N(\mu_i, \sigma^2), \mu_i = \beta_0 + \beta_1 x_{1i} + \dots + \beta_k x_{ki}, \beta_k \sim N(0, 1e6)$$

We note that  $k$  is the number of descriptor variables in the data set and  $i$  is the number of observations.

Also,

$$y_i | x_i, \beta, \sigma^2 \stackrel{ind}{\sim} N(\beta_0 + \beta_1 x_{1i} + \dots + \beta_k x_{ki}, \sigma^2),$$

where the noninformative prior for  $\sigma^2$  is modeled using an  $InverseGamma(\alpha, \beta)$  distribution.

```
library("rjags")
```

```

## Loading required package: coda
## Linked to JAGS 4.3.2
## Loaded modules: basemod, bugs
newdat1 = na.omit(newdat1)

mod1_jags_string = " model {
  for (i in 1:n) {
    y[i] ~ dnorm(mu[i], prec)
    mu[i] = b0
      + b[1]*LotArea[i]
      + b[2]*OverallCond[i]
      + b[3]*HouseStyle_coded1.5Unf[i]
      + b[4]*HouseStyle_coded1Story[i]
      + b[5]*HouseStyle_coded2.5Fin[i]
      + b[6]*HouseStyle_coded2.5Unf[i]
      + b[7]*HouseStyle_coded2Story[i]
      + b[8]*HouseStyle_codedSoyer[i]
      + b[9]*HouseStyle_codedSLvl[i]
  }
  b0 ~ dnorm(0.0, 1.0/1.0e6)

  for (i in 1:9) {
    b[i] ~ dnorm(0.0, 1.0/1.0e6)
  }

  prec ~ dgamma(5/2.0, 5*10.0/2.0)
  sig2 = 1.0 / prec
  sig = sqrt(sig2)
} "

set.seed(72)
data_jags = list(y=newdat1$SalePrice,

```

```

LotArea=newdat1$LotArea,
OverallCond=newdat1$OverallCond,
HouseStyle_coded1.5Unf=as.numeric(newdat1$HouseStyle_coded=="1.5Unf"),
HouseStyle_coded1Story=as.numeric(newdat1$HouseStyle_coded=="1Story"),
HouseStyle_coded2.5Fin=as.numeric(newdat1$HouseStyle_coded=="2.5Fin"),
HouseStyle_coded2.5Unf=as.numeric(newdat1$HouseStyle_coded=="2.5Unf"),
HouseStyle_coded2Story=as.numeric(newdat1$HouseStyle_coded=="2Story"),
HouseStyle_codedSFoyer=as.numeric(newdat1$HouseStyle_coded=="SFoyer"),
HouseStyle_codedSLvl=as.numeric(newdat1$HouseStyle_coded=="SLvl"),
n=nrow(newdat1))

params1 = c("b0", "b", "sig")

inits1 = function() {
  inits = list("b0"=rnorm(1,0.0,100.0), "b"=rnorm(9,0.0,100.0), "prec"=rgamma(1,1.0,1.0))
}

mod1_jags = jags.model(textConnection(mod1_jags_string), data=data_jags, inits=inits1, n.chains=3)

## Compiling model graph
## Resolving undeclared variables
## Allocating nodes
## Graph information:
##   Observed stochastic nodes: 1460
##   Unobserved stochastic nodes: 11
##   Total graph size: 17039
##
## Initializing model

```

Fit the Model using the Monte Carlo-Markov Chain (MCMC) Sampler

```

update(mod1_jags, 1000) # burn-in

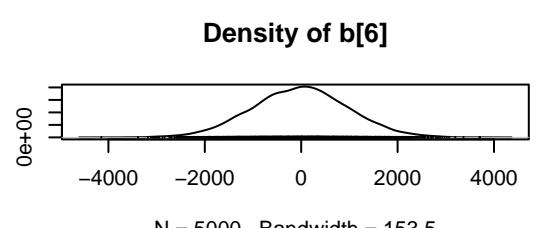
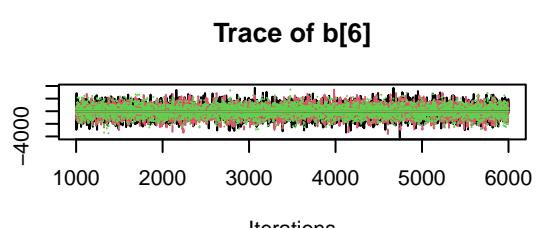
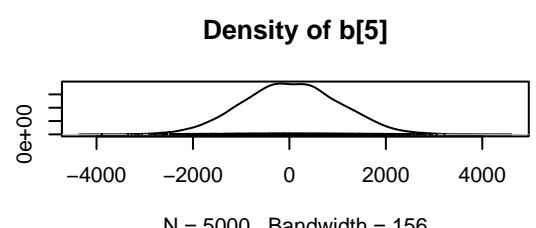
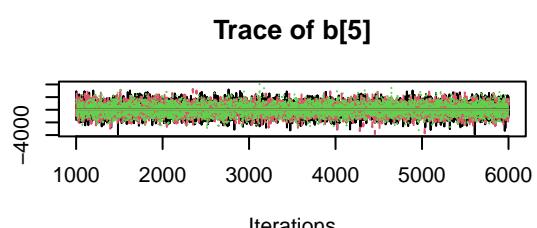
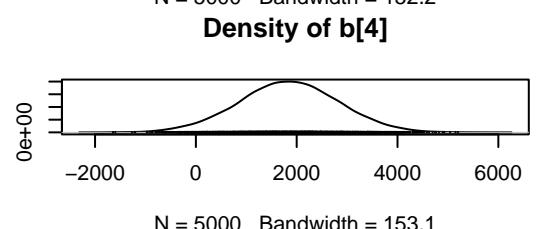
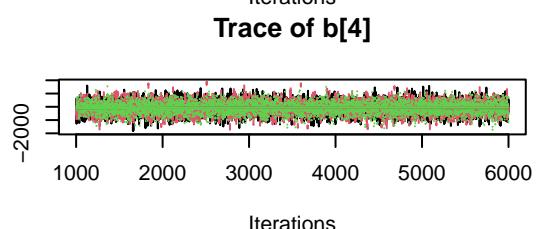
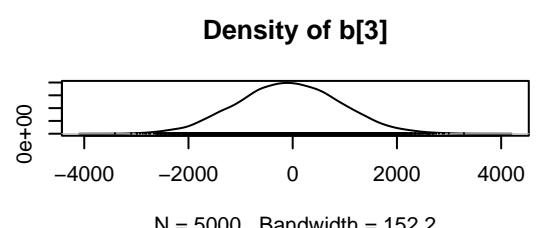
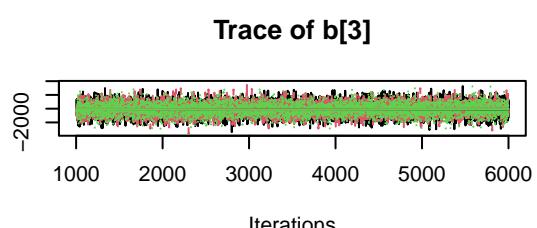
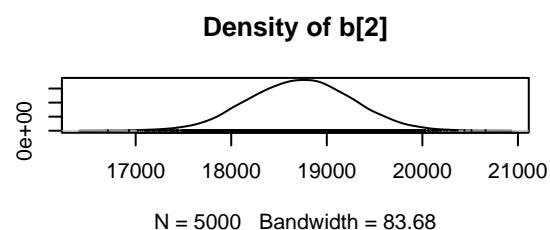
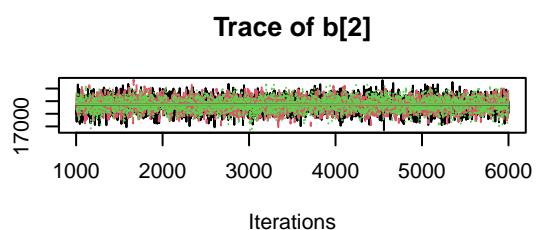
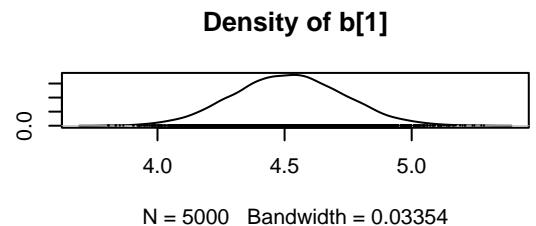
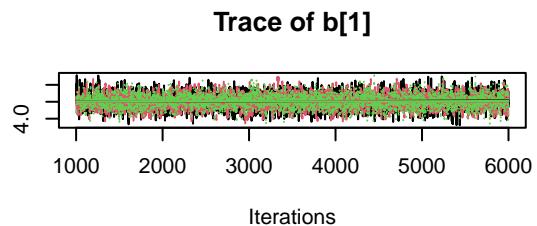
mod1_jags_sim = coda.samples(model=mod1_jags,
                             variable.names=params1,
                             n.iter=5000)

mod1_jags_csim = do.call(rbind, mod1_jags_sim) # combine multiple chains

```

Check the Model by Examining Convergence Diagnostics

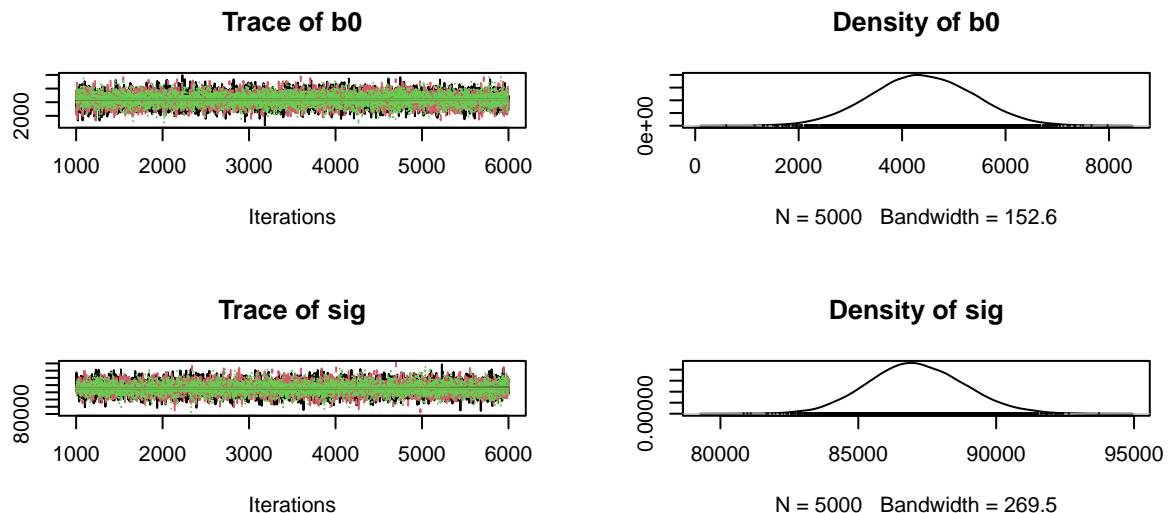
```
plot(mod1_jags_sim)
```



Trace o

Trace o

Trace o



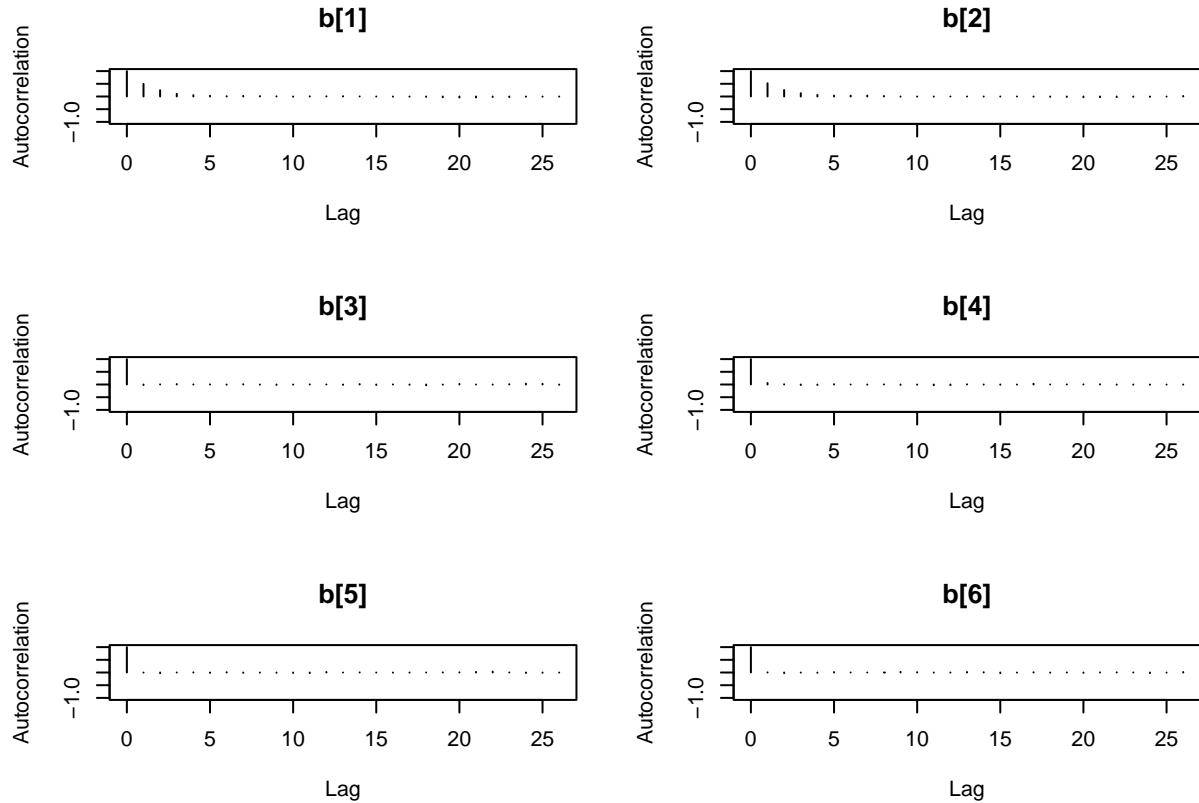
```
gelman.diag(mod1_jags_sim)
```

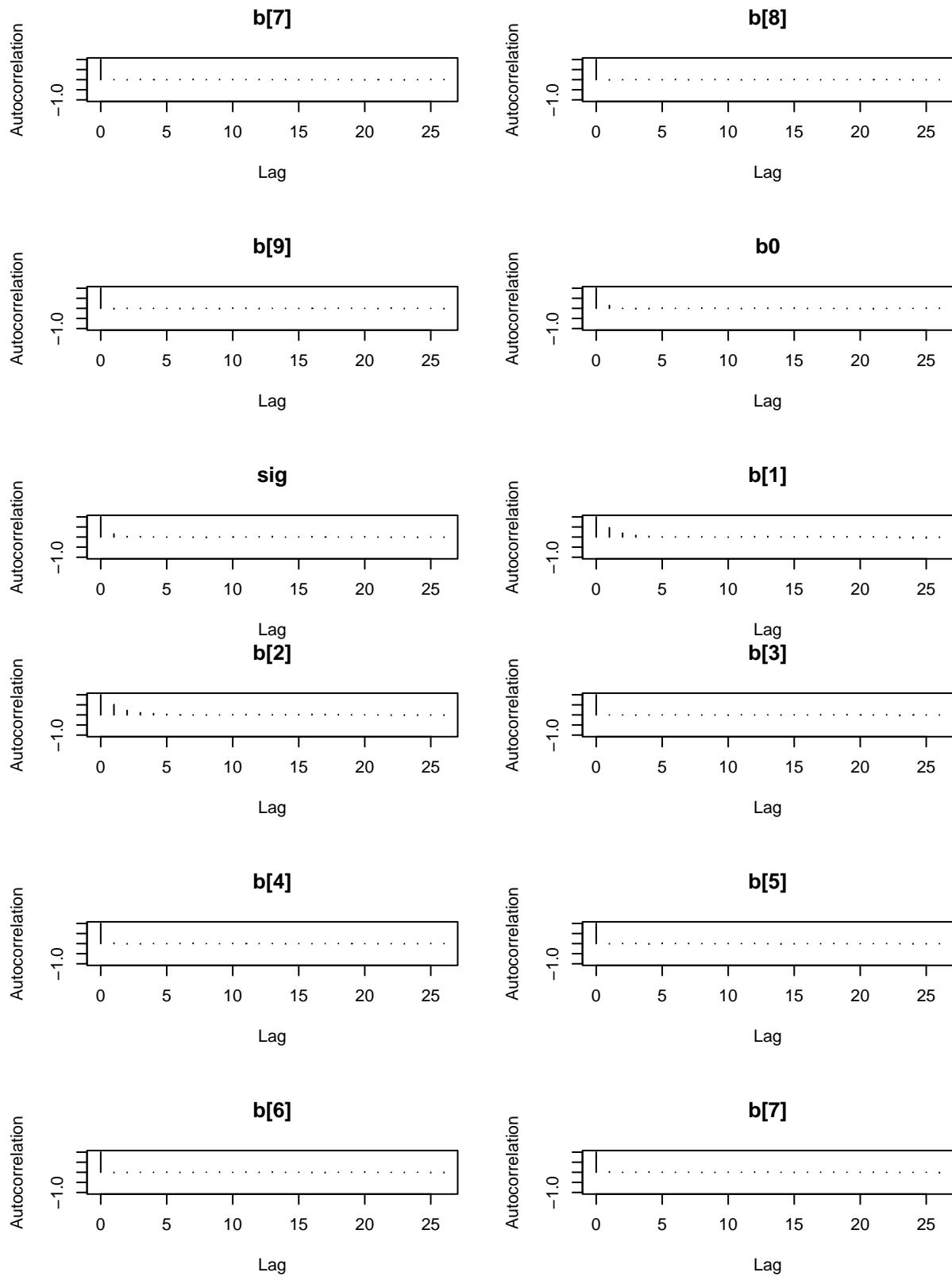
```
## Potential scale reduction factors:
##
##          Point est. Upper C.I.
## b[1]          1          1
## b[2]          1          1
## b[3]          1          1
## b[4]          1          1
## b[5]          1          1
## b[6]          1          1
## b[7]          1          1
## b[8]          1          1
## b[9]          1          1
## b0           1          1
## sig          1          1
##
## Multivariate psrf
##
## 1
```

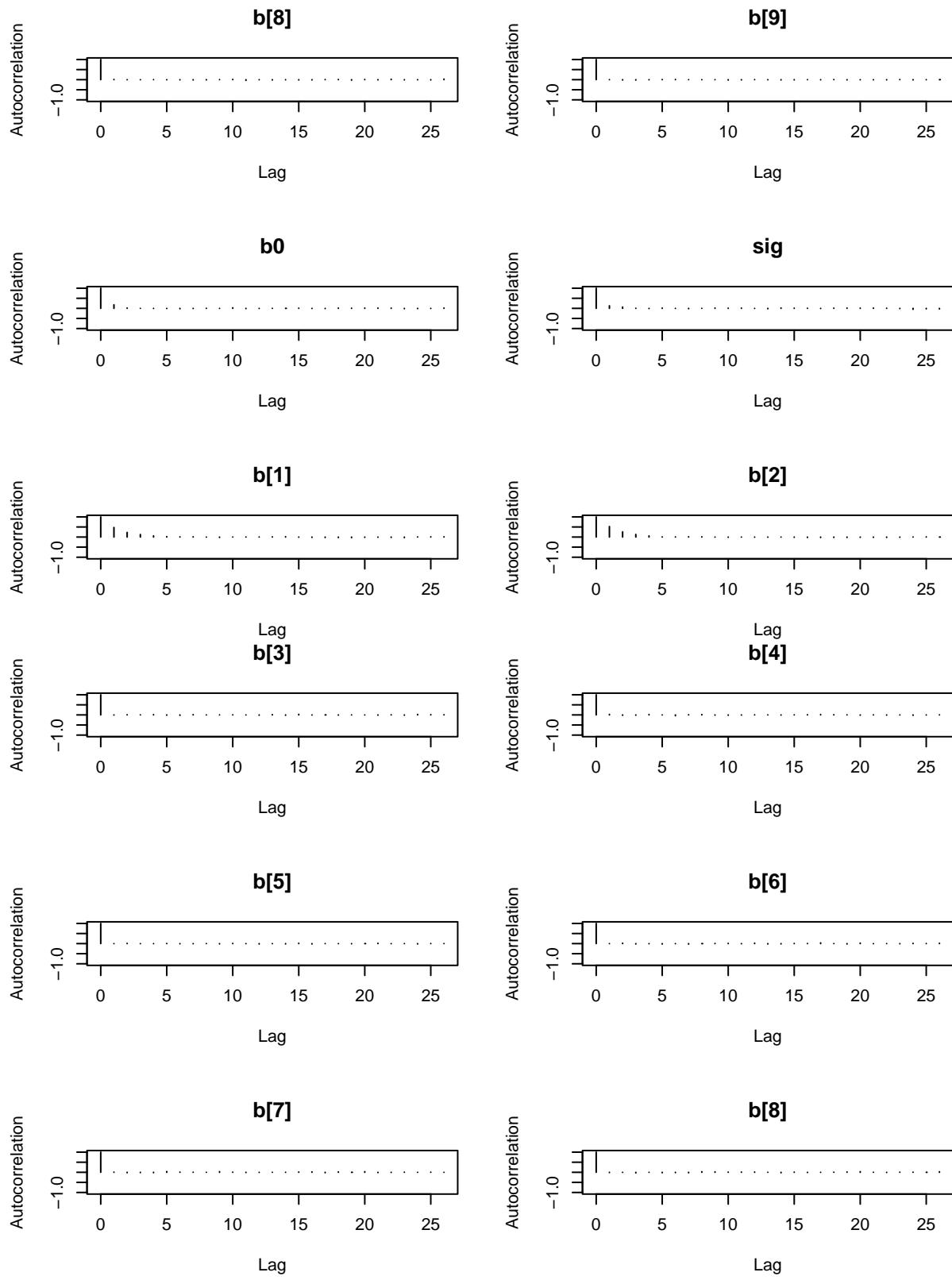
```
autocorr.diag(mod1_jags_sim)
```

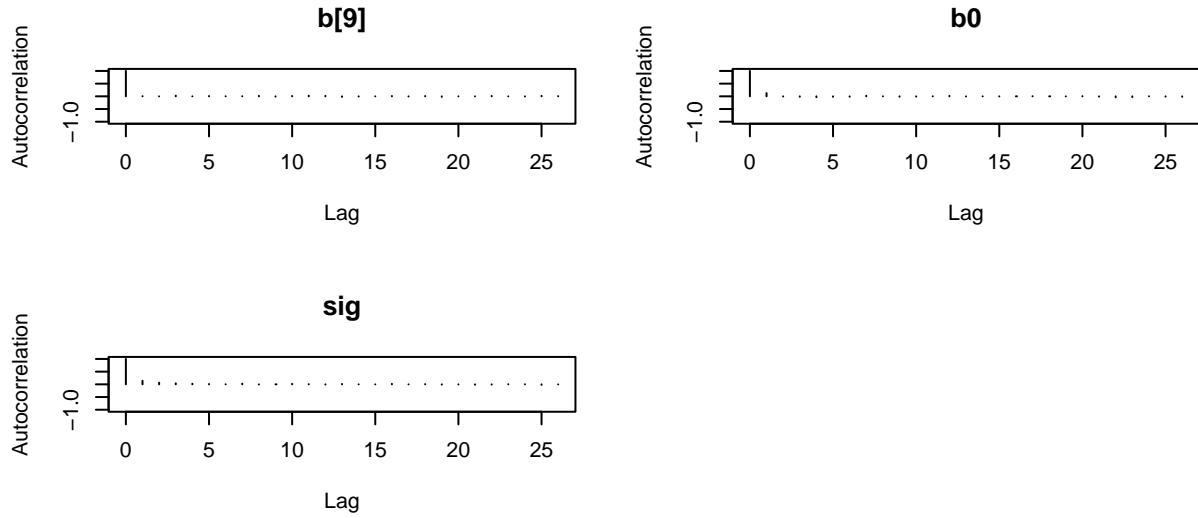
```
##          b[1]          b[2]          b[3]          b[4]          b[5]
## Lag 0  1.00000000  1.000000000  1.000000000  1.000000000  1.0000000000
## Lag 1  0.47826018  0.518759199 -0.010801669  0.039089278 -0.0068034536
## Lag 5   0.02008822  0.024911823 -0.003565759  0.001567526  0.0074322492
## Lag 10  -0.00619530 -0.001174090 -0.007490973 -0.004062278  0.0008611876
## Lag 50   0.01230692  0.005467157 -0.008553663 -0.010964634 -0.0027735407
##          b[6]          b[7]          b[8]          b[9]          b0
## Lag 0  1.000000000  1.000000000  1.000000000  1.000000000  1.0000000000
## Lag 1  -0.005223669  0.013639612 -0.007868987 -0.008675078  0.150787188
## Lag 5   0.002346409  0.011629115 -0.011165371  0.007259578 -0.003678037
## Lag 10  0.007735474  0.007592051  0.012607898  0.004881861  0.003866315
## Lag 50   0.002347924 -0.006582604 -0.002485365  0.005459084  0.004196902
##          sig
## Lag 0  1.000000000
## Lag 1  0.142126145
```

```
## Lag 5  0.005823401
## Lag 10 0.010156495
## Lag 50 -0.002808136
autocorr.plot(mod1_jags_sim)
```









```
effectiveSize(mod1_jags_sim)
```

```
##      b[1]      b[2]      b[3]      b[4]      b[5]      b[6]      b[7]      b[8]
## 5392.534 5055.092 15209.988 14824.688 14888.589 15000.000 14826.916 15900.408
##      b[9]      b0      sig
## 15355.498 12021.007 10410.004
```

```
dic.samples(mod1_jags, n.itер=1e3)
```

```
## Mean deviance: 37361
```

```
## penalty 3.29
```

```
## Penalized deviance: 37364
```

We can get a posterior summary of the parameters in our model.

```
summary(mod1_jags_sim)
```

```
##
## Iterations = 1001:6000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 5000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##           Mean        SD  Naive SE Time-series SE
## b[1]    4.517    0.2165  0.001768    0.002955
## b[2] 18724.352  540.1532  4.410332    7.606828
## b[3]   -74.113  982.2139  8.019743   7.966187
## b[4]   1840.495  989.3355  8.077890   8.142961
## b[5]    37.596 1006.9234  8.221495   8.252603
## b[6]   -20.642  996.1342  8.133401   8.133863
## b[7]  3111.494  977.8125  7.983806   8.034580
## b[8]   -57.550  992.2306  8.101529   7.877325
## b[9]    50.384 1007.3586  8.225049   8.134052
## b0     4389.804  984.8674  8.041409   9.030974
## sig   87098.308 1739.6541 14.204216  17.069935
##
```

```

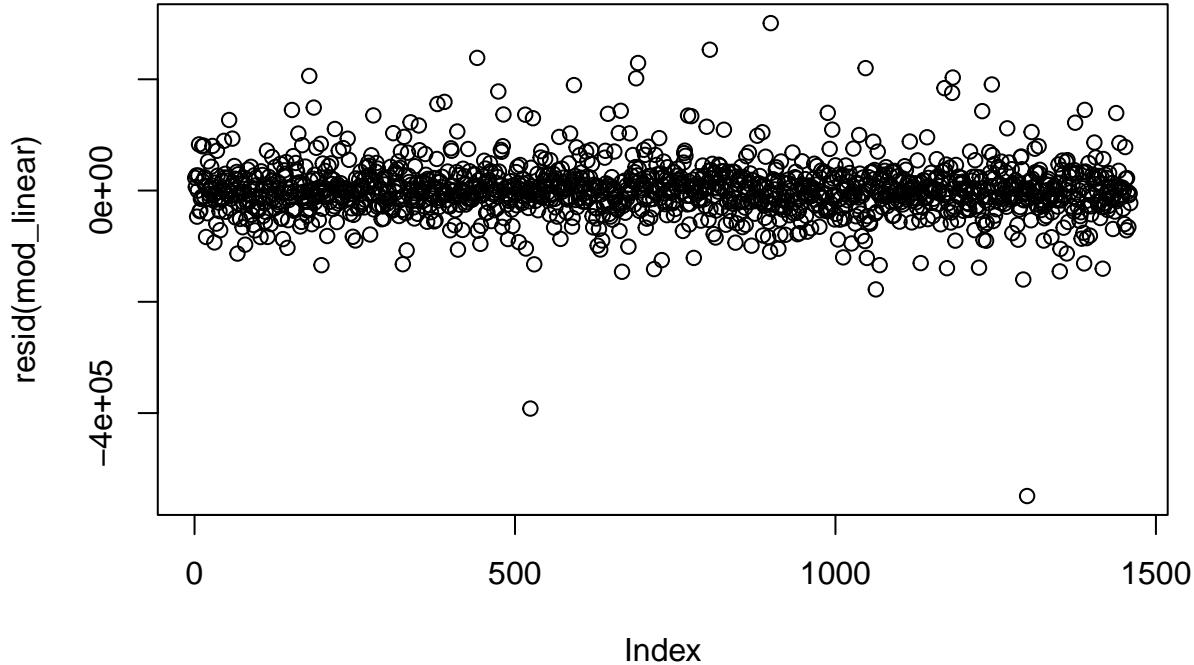
## 2. Quantiles for each variable:
##
##          2.5%      25%      50%      75%     97.5%
## b[1]    4.098     4.37     4.517     4.664     4.945
## b[2] 17669.028 18356.12 18729.101 19089.787 19776.182
## b[3] -1963.897 -735.23   -77.098   592.935  1853.362
## b[4] -135.084  1179.11  1843.873  2503.003  3772.284
## b[5] -1909.165 -645.71   27.399   714.305  1994.175
## b[6] -1985.879 -684.91  -12.853   642.927  1940.872
## b[7] 1190.231  2454.07  3119.179  3765.669  5024.593
## b[8] -2007.631 -720.52  -55.625   608.520  1882.332
## b[9] -1904.526 -620.71   44.402   730.279  2023.528
## b0     2473.553  3727.92  4378.989  5058.672  6311.579
## sig    83786.337 85920.71 87056.233 88254.830 90592.934

```

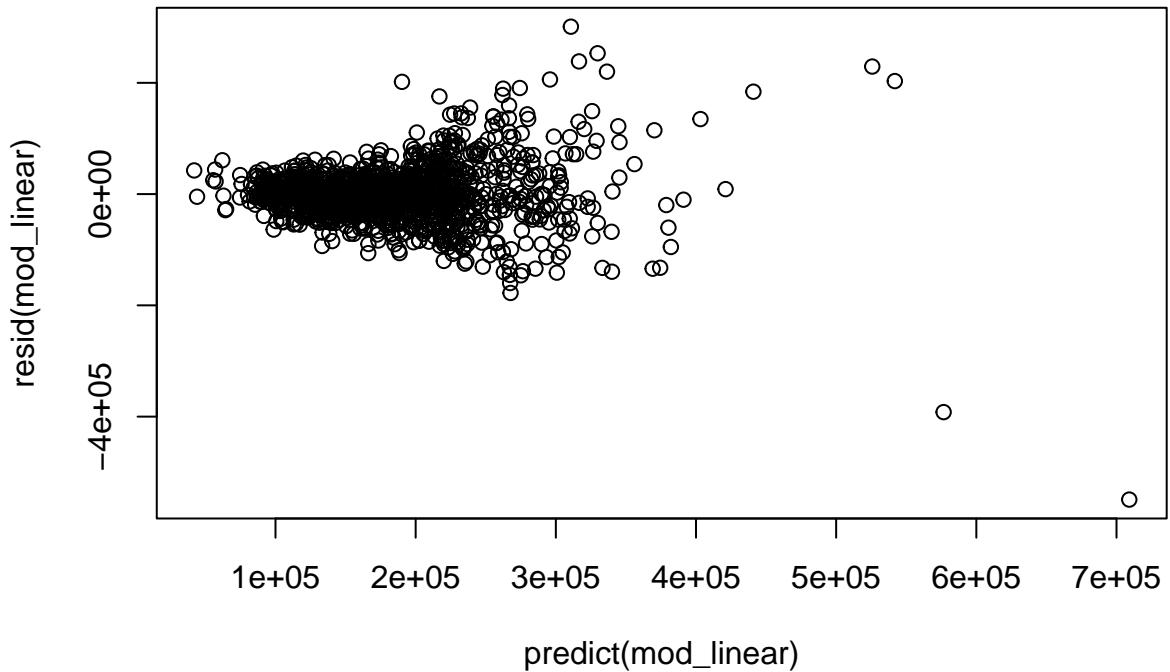
### Residual checks

Checking residuals (the difference between the response and the model's prediction for that value) is important with linear models since residuals can reveal violations of the assumptions we made to specify the model. In particular, we are looking for any sign that the model is not linear, normally distributed, or that the observations are not independent (conditional on covariates). We first evaluate the simple linear model proposed earlier:

```
plot(resid(mod_linear)) # to check independence (looks okay)
```

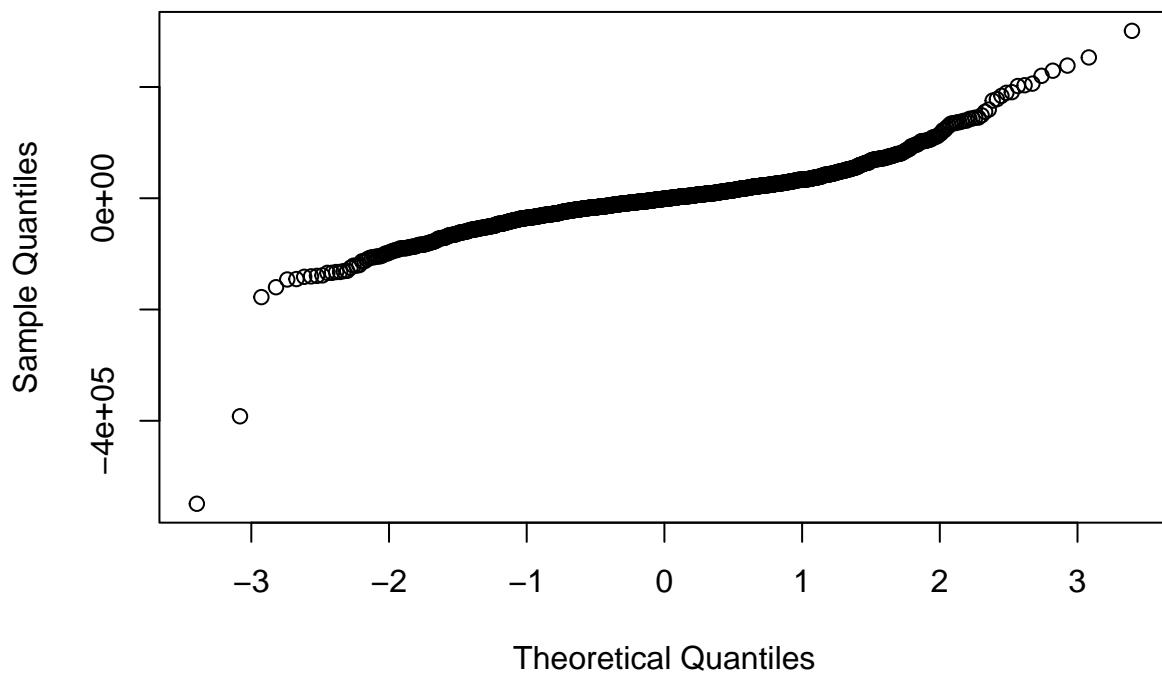


```
plot(predict(mod_linear), resid(mod_linear)) # to check for linearity, constant variance (looks reasonable)
```



```
qqnorm(resid(mod_linear)) # to check Normality assumption (we want this to be a straight line)
```

### Normal Q-Q Plot



Now let's return to our JAGS model. In a Bayesian model, we have distributions for residuals, but we'll simplify and look only at the residuals evaluated at the posterior mean of the parameters.

```
X = cbind(
  rep(1.0, data_jags$n),
  data_jags$LotArea,
  data_jags$OverallCond,
```

```

data_jags$HouseStyle_coded1.5Unf,
data_jags$HouseStyle_coded1Story,
data_jags$HouseStyle_coded2.5Fin,
data_jags$HouseStyle_coded2.5Unf,
data_jags$HouseStyle_coded2Story,
data_jags$HouseStyle_codedSoyer,
data_jags$HouseStyle_codedSLvl
)
head(X)

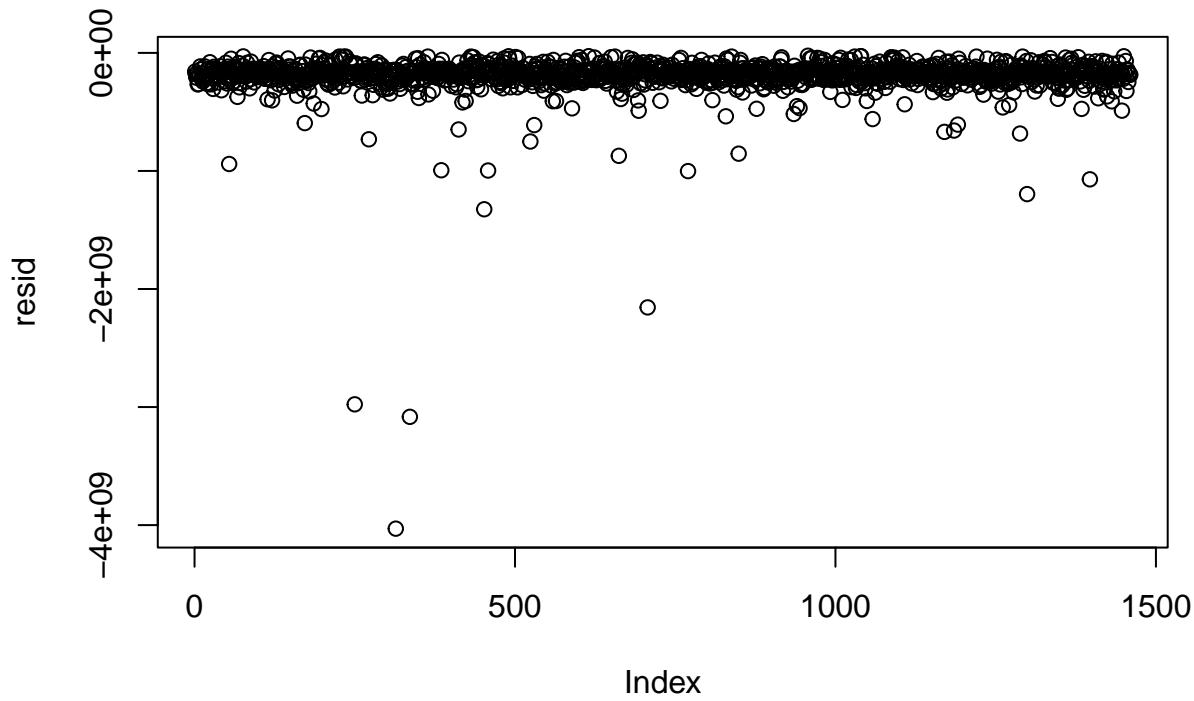
##      [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9]  [,10]
## [1,]    1  8450     5     0     0     0     0     1     0     0
## [2,]    1  9600     8     0     1     0     0     0     0     0
## [3,]    1 11250     5     0     0     0     0     0     1     0     0
## [4,]    1  9550     5     0     0     0     0     0     1     0     0
## [5,]    1 14260     5     0     0     0     0     0     1     0     0
## [6,]    1 14115     5     0     0     0     0     0     0     0     0

(pm_params = colMeans(mod1_jags_csim)) # posterior mean

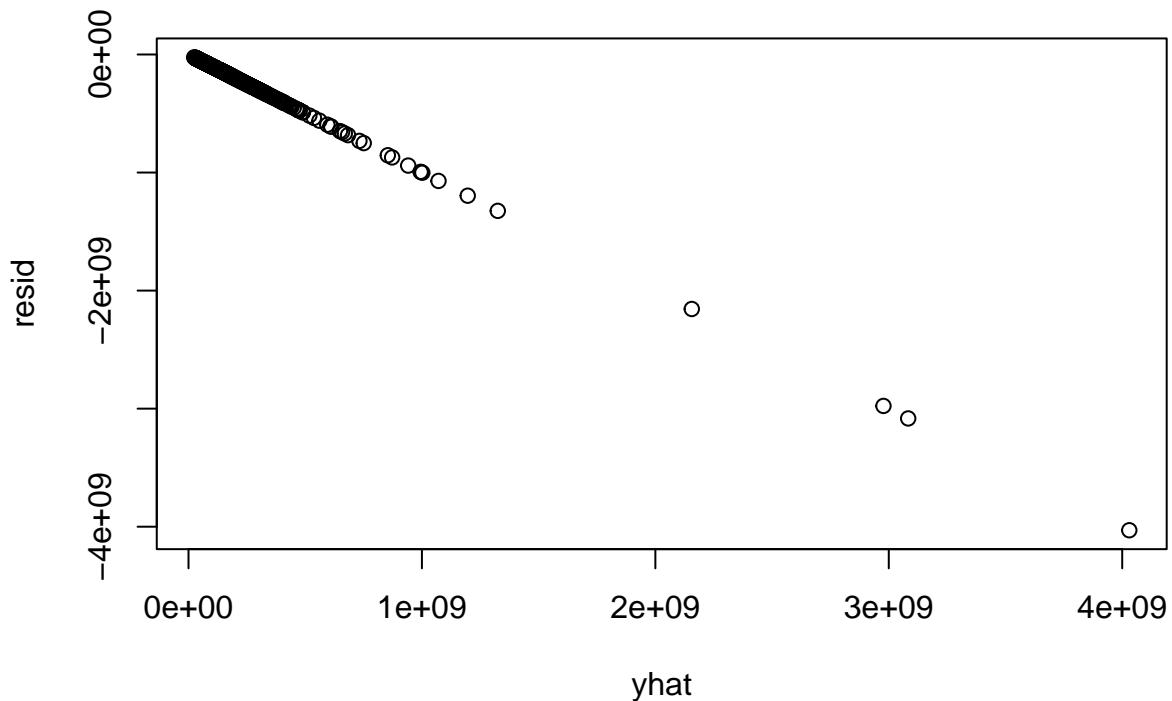
##          b[1]          b[2]          b[3]          b[4]          b[5]          b[6]
## 4.517282 18724.352303 -74.112644 1840.494543 37.596269 -20.642372
##          b[7]          b[8]          b[9]          b0          sig
## 3111.494483 -57.549617  50.384179 4389.804154 87098.307588

yhat = drop(X %*% pm_params[1:10])
resid = data_jags$y - yhat
plot(resid) # residuals against data index

```

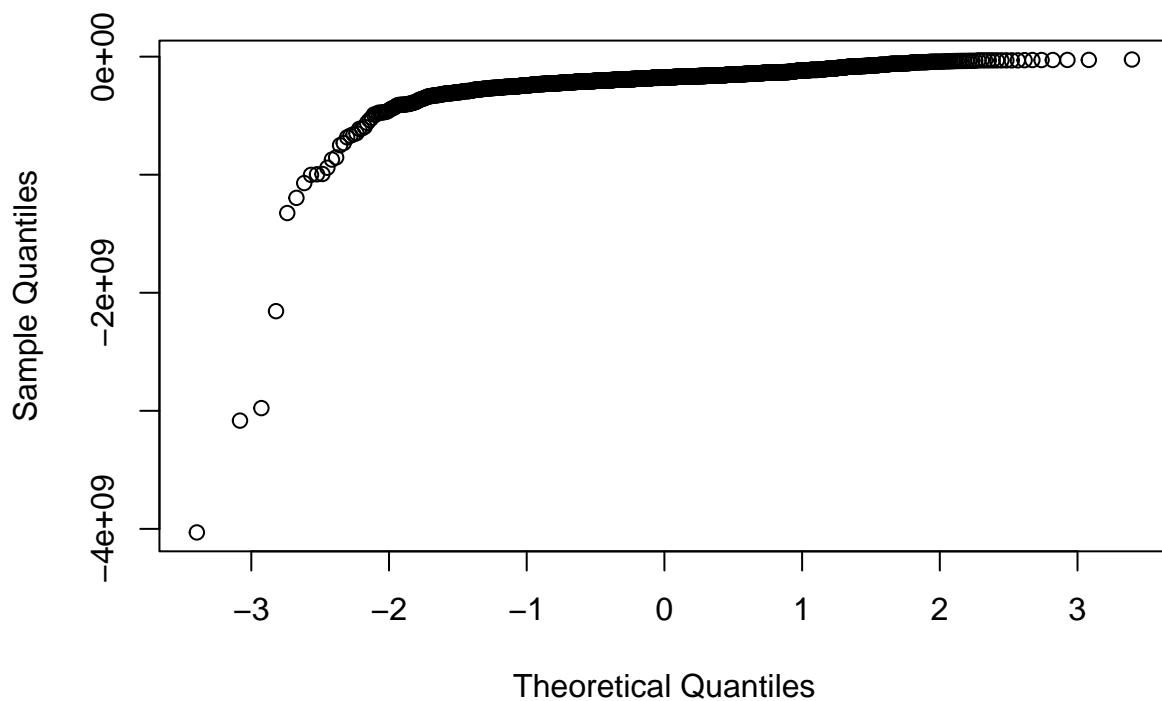


```
plot(yhat, resid) # residuals against predicted values
```

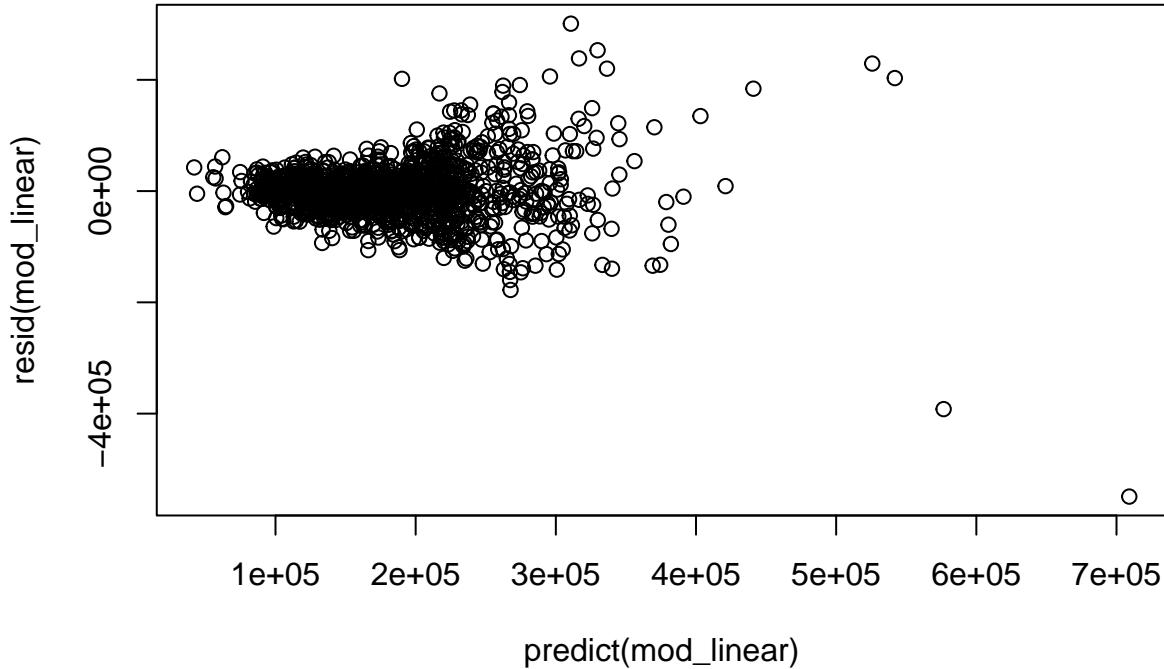


```
qqnorm(resid) # checking normality of residuals
```

Normal Q-Q Plot



```
plot(predict(mod_linear), resid(mod_linear)) # to compare with reference linear model
```



```
# rownames(dat1)[order(resid1, decreasing=TRUE)[1:5]]
```

### Iterate with another model

As mentioned previously, we will build another linear model that incorporates the feature `TotRmsAbvGrd`, which is the total rooms above grade (does not include bathrooms).

We first adjust the dataset to include `TotRmsAbvGrd`.

```
dat2=dat[,c("LotArea", "HouseStyle", "OverallCond", "GrLivArea", "TotRmsAbvGrd", "SalePrice")]
dat2$HouseStyle_coded = factor(dat2$HouseStyle)
newdat2 = dat2[,c("LotArea", "HouseStyle_coded", "OverallCond", "GrLivArea", "TotRmsAbvGrd", "SalePrice")]

head(newdat2)

##   LotArea HouseStyle_coded OverallCond GrLivArea TotRmsAbvGrd SalePrice
## 1    8450           2Story         5      1710          8     208500
## 2    9600           1Story         8      1262          6     181500
## 3   11250           2Story         5      1786          6     223500
## 4    9550           2Story         5      1717          7     140000
## 5   14260           2Story         5      2198          9     250000
## 6   14115          1.5Fin         5      1362          5     143000

mod2_jags_string = " model {
  for (i in 1:n) {
    y[i] ~ dnorm(mu[i], prec)
    mu[i] = b0
    + b[1]*LotArea[i]
    + b[2]*OverallCond[i]
    + b[3]*HouseStyle_coded1.5Unf[i]
    + b[4]*HouseStyle_coded1Story[i]
    + b[5]*HouseStyle_coded2.5Fin[i]
    + b[6]*HouseStyle_coded2.5Unf[i]
    + b[7]*HouseStyle_coded2Story[i]
```

```

        + b[8]*HouseStyle_codedSFoyer[i]
        + b[9]*HouseStyle_codedSLvl[i]
        + b[10]*TotRmsAbvGrd[i]
    }

b0 ~ dnorm(0.0, 1.0/1.0e6)

for (i in 1:10) {
    b[i] ~ dnorm(0.0, 1.0/1.0e6)
}

prec ~ dgamma(5/2.0, 5*10.0/2.0)
sig2 = 1.0 / prec
sig = sqrt(sig2)
} "

set.seed(72)
data_jags = list(y=newdat2$SalePrice,
                  LotArea=newdat2$LotArea,
                  OverallCond=newdat2$OverallCond,
                  HouseStyle_coded1.5Unf=as.numeric(newdat2$HouseStyle_coded=="1.5Unf"),
                  HouseStyle_coded1Story=as.numeric(newdat2$HouseStyle_coded=="1Story"),
                  HouseStyle_coded2.5Fin=as.numeric(newdat2$HouseStyle_coded=="2.5Fin"),
                  HouseStyle_coded2.5Unf=as.numeric(newdat2$HouseStyle_coded=="2.5Unf"),
                  HouseStyle_coded2Story=as.numeric(newdat2$HouseStyle_coded=="2Story"),
                  HouseStyle_codedSFoyer=as.numeric(newdat2$HouseStyle_coded=="SFoyer"),
                  HouseStyle_codedSLvl=as.numeric(newdat2$HouseStyle_coded=="SLvl"),
                  TotRmsAbvGrd=dat2$TotRmsAbvGrd,
                  n=nrow(newdat2))

params1 = c("b0", "b", "sig")

inits1 = function() {
    inits = list("b0"=rnorm(1,0.0,100.0), "b"=rnorm(10,0.0,100.0), "prec"=rgamma(1,1.0,1.0))
}

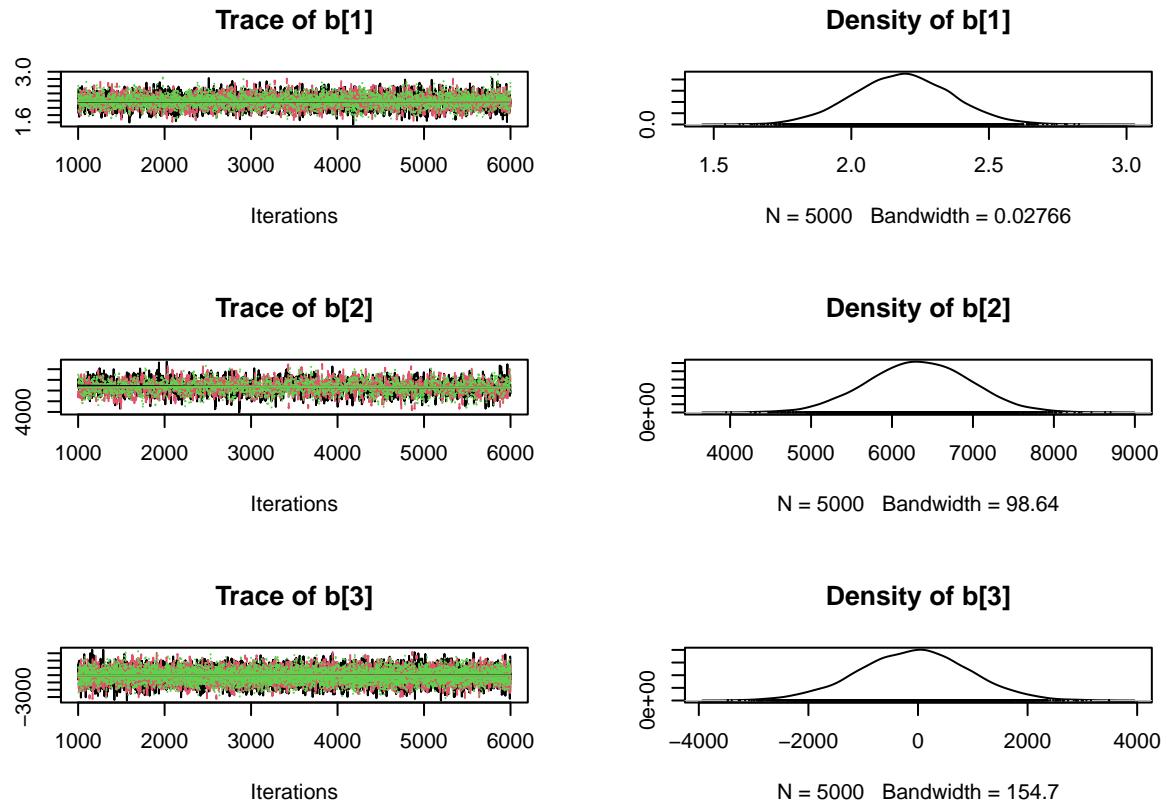
mod2_jags = jags.model(textConnection(mod2_jags_string), data=data_jags, inits=inits1, n.chains=3)

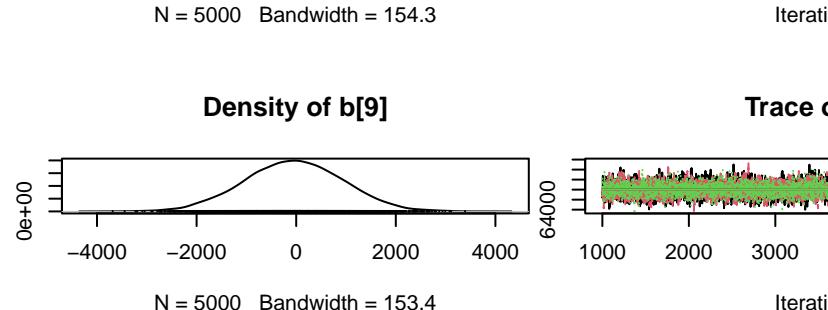
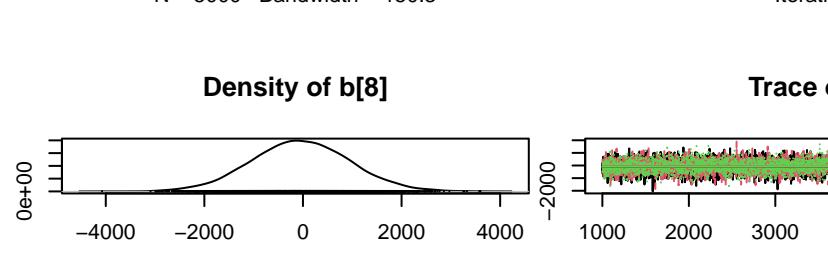
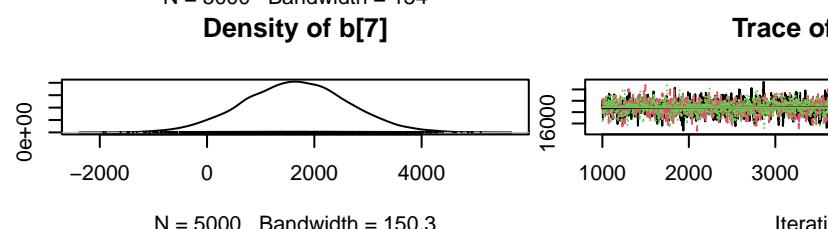
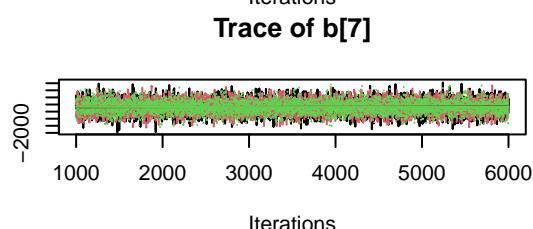
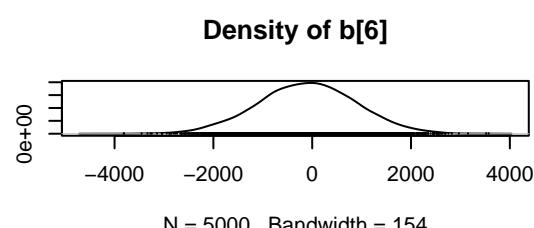
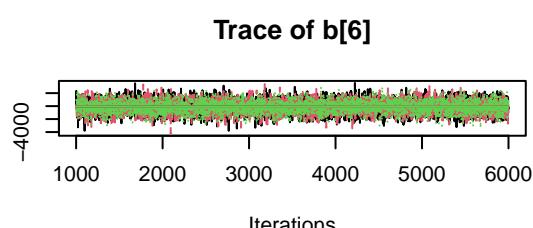
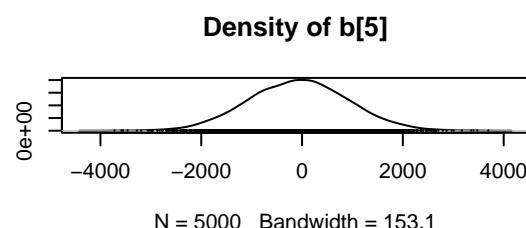
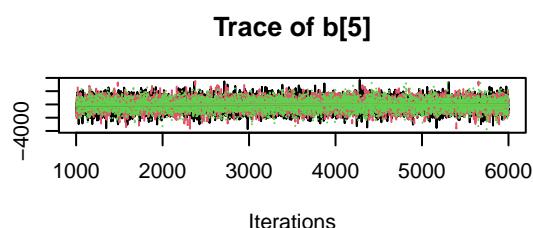
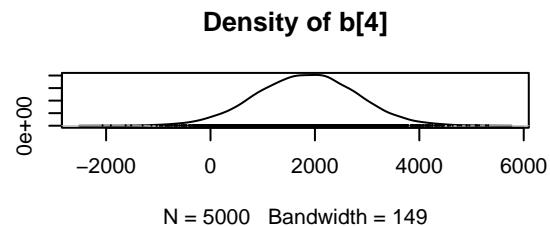
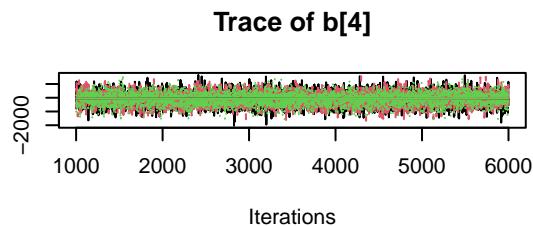
## Compiling model graph
## Resolving undeclared variables
## Allocating nodes
## Graph information:
##   Observed stochastic nodes: 1460
##   Unobserved stochastic nodes: 12
##   Total graph size: 18584
##
## Initializing model
update(mod2_jags, 1000) # burn-in

mod2_jags_sim = coda.samples(model=mod2_jags,
                             variable.names=params1,
                             n.iter=5000)

```

```
mod2_jags_csim = do.call(rbind, mod2_jags_sim) # combine multiple chains
plot(mod2_jags_sim)
```





gelman.diag(mod2\_jags\_sim)

## Potential scale reduction factors:

```

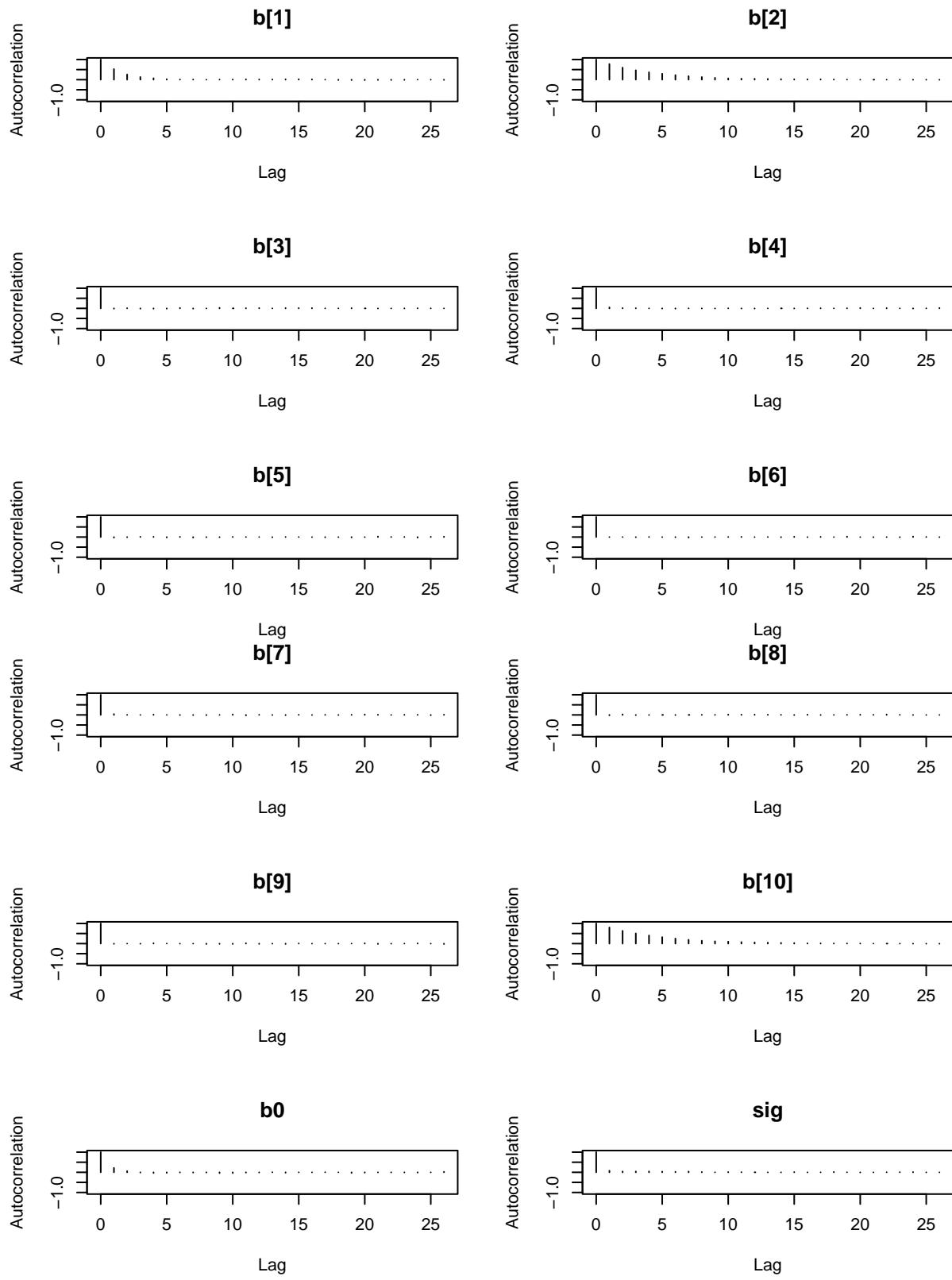
## Point est. Upper C.I.
## b[1] 1 1
## b[2] 1 1
## b[3] 1 1
## b[4] 1 1
## b[5] 1 1
## b[6] 1 1
## b[7] 1 1
## b[8] 1 1
## b[9] 1 1
## b[10] 1 1
## b0 1 1
## sig 1 1
##
## Multivariate psrf
##
## 1

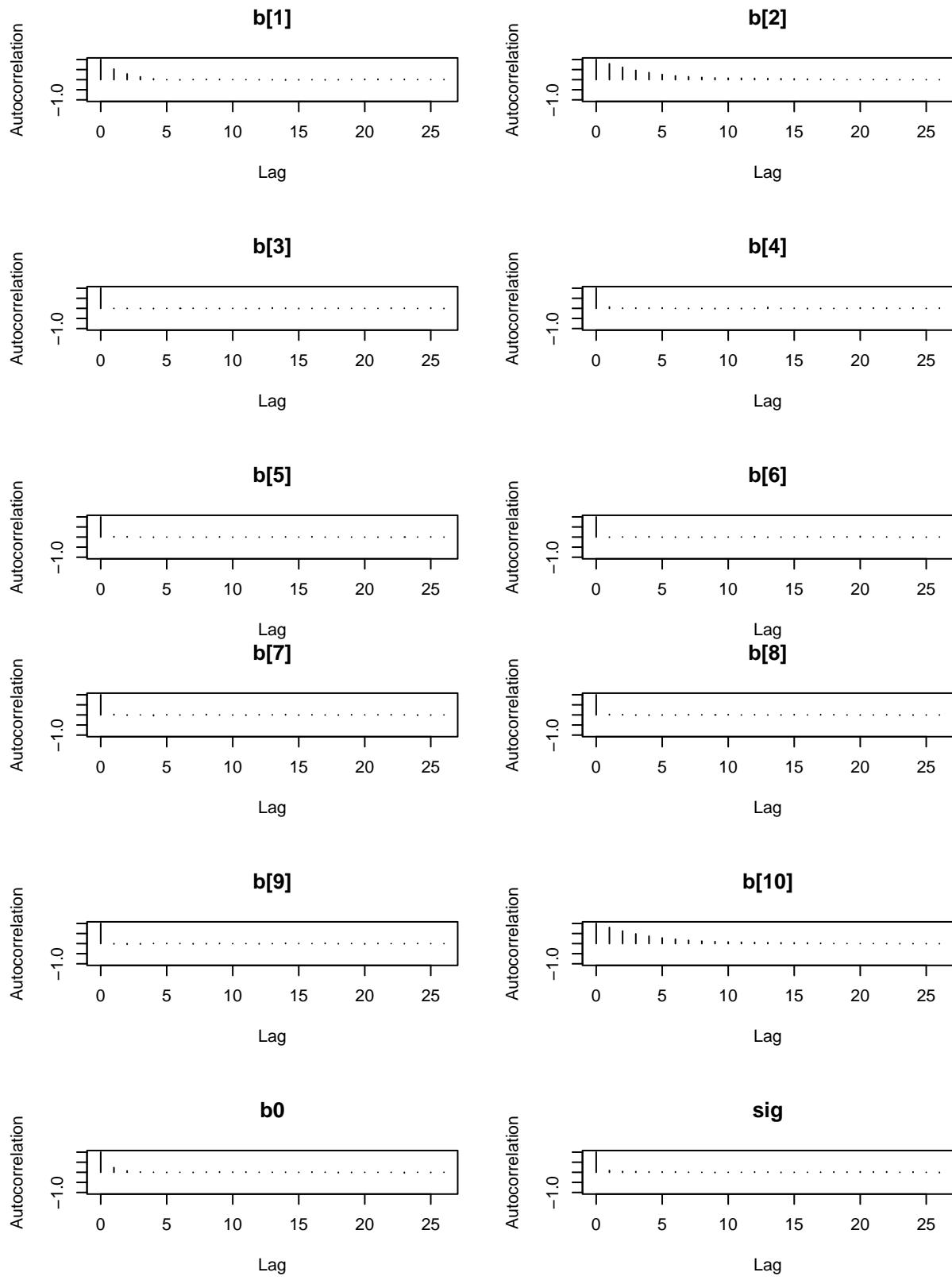
autocorr.diag(mod2_jags_sim)

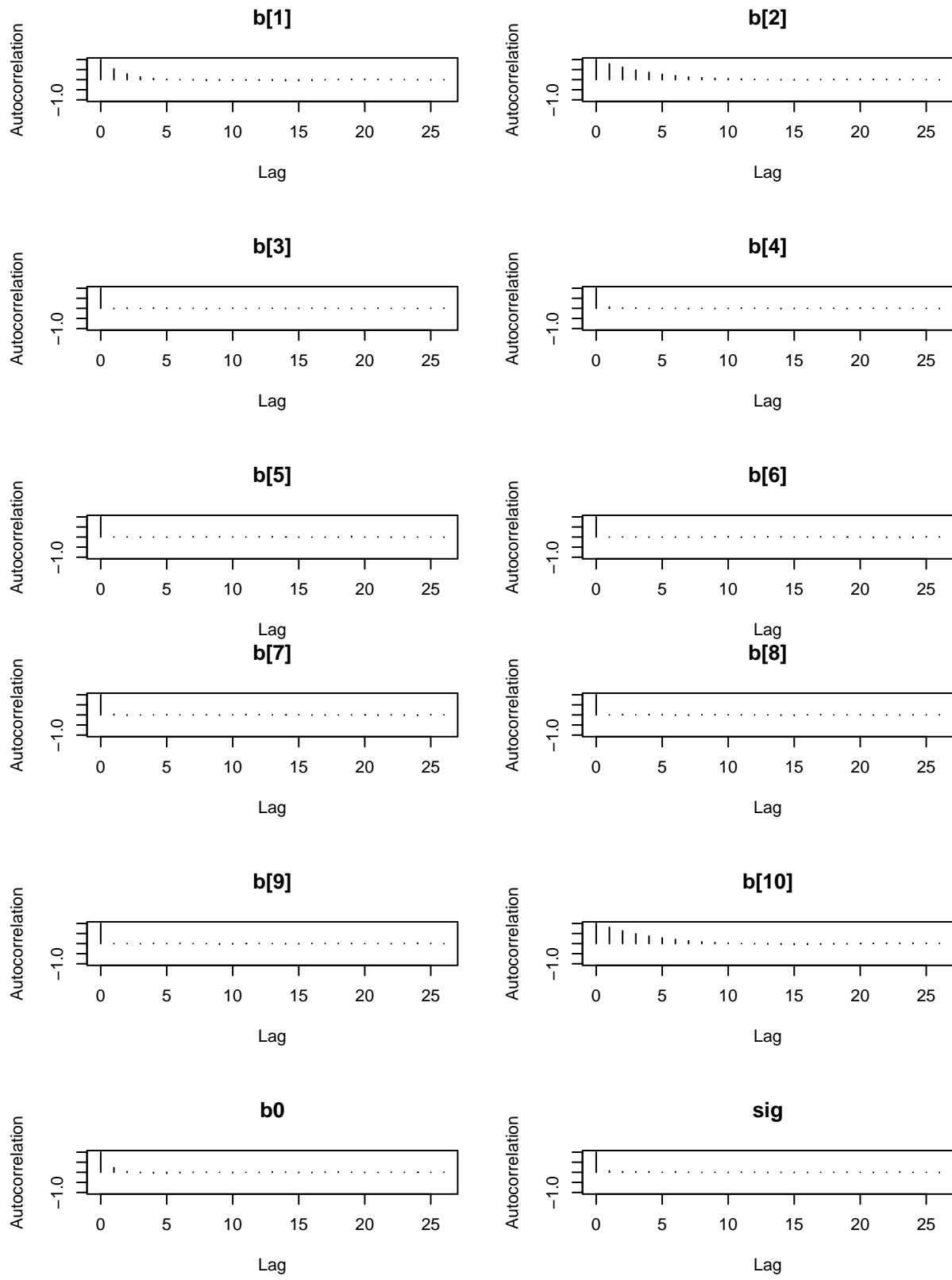
##          b[1]          b[2]          b[3]          b[4]          b[5]
## Lag 0  1.000000000 1.000000000 1.000000000 1.000000000 1.0000000000
## Lag 1  0.530327065 0.783623190 -0.009566133 0.062132500 -0.0026804625
## Lag 5  0.012230041 0.272910684 -0.000365819 0.002425897 -0.0096201569
## Lag 10 -0.004325473 0.062551404  0.003023818 -0.003242516 0.0005466591
## Lag 50  0.016072772 0.009343129  0.006428225 0.005614751 0.0163806874
##          b[6]          b[7]          b[8]          b[9]          b[10]
## Lag 0  1.000000000 1.000000000 1.000000000 1.000000000 1.0000000000
## Lag 1  -0.006581827 0.034640676 0.002383727 -0.008703826 0.802631346
## Lag 5  -0.007328268 0.008350276 0.001154129 0.006476519 0.295955435
## Lag 10  0.009321226 0.006512792 0.013976469 -0.014821413 0.067630765
## Lag 50  -0.008772307 0.004030820 0.002223556 -0.004089579 0.001473009
##          b0          sig
## Lag 0  1.0000000 1.0000000000
## Lag 1  0.22741756 0.0800211066
## Lag 5  -0.03050485 0.0193602017
## Lag 10 -0.01631905 0.0003858429
## Lag 50  0.02394910 0.0013681581

autocorr.plot(mod2_jags_sim)

```







```
effectiveSize(mod2_jags_sim)
```

```
##   b[1]   b[2]   b[3]   b[4]   b[5]   b[6]   b[7]   b[8]
```

```

## 4751.148 1886.631 15000.000 13244.917 15120.047 15000.000 14789.705 14688.633
##      b[9]      b[10]      b0      sig
## 15794.928 1777.969 11241.093 10091.241
dic.samples(mod1_jags, n.ITER=1e3)

## Mean deviance: 37362
## penalty 3.082
## Penalized deviance: 37365
dic.samples(mod2_jags, n.ITER=1e3)

## Mean deviance: 36638
## penalty 3.508
## Penalized deviance: 36642

```

Upon examination of the deviance information criterion (DIC), the penalized deviance of the second model (which incorporates `TotRmsAbvGrd`) is lower than the first model. We conclude that the second model is a better fit (and predictor) for the data.