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. Two models were constructed using both `R` and `JAGS`. One of the `JAGS` models uses 3 features to predict sale prices while the final iteration uses 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 examined 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 ...
```

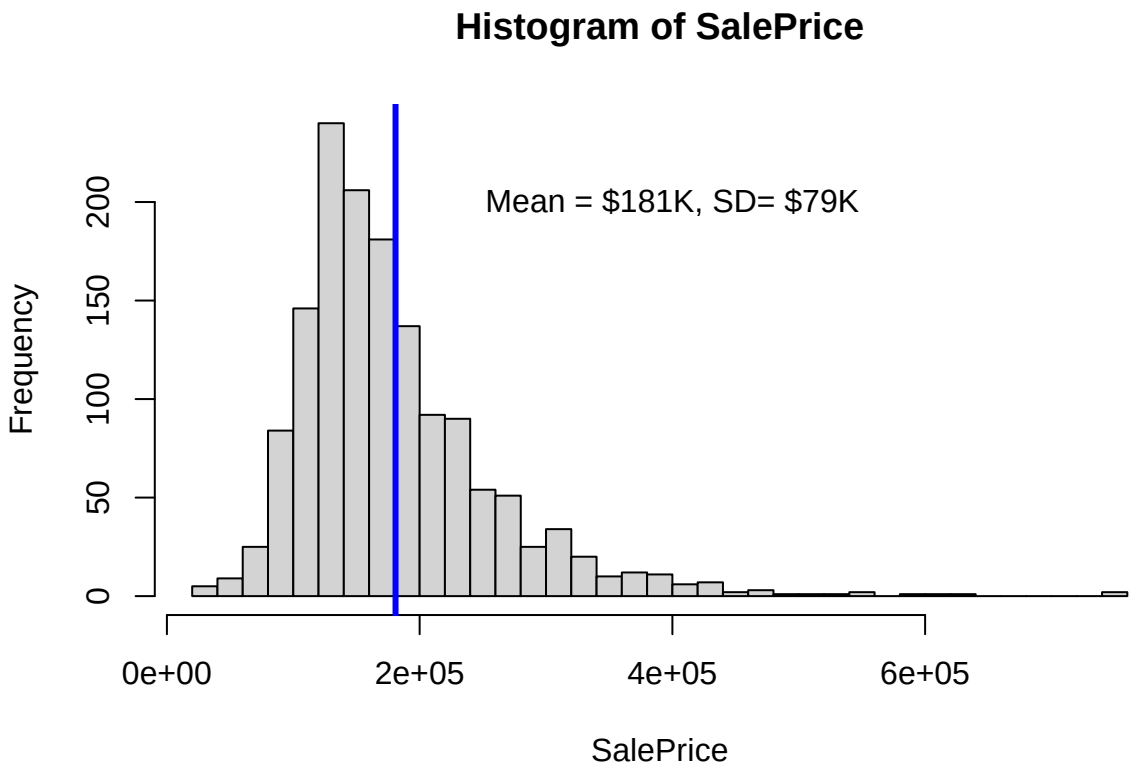
We will need to code the categorical variable `HouseStyle`.

```
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')
```



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
```

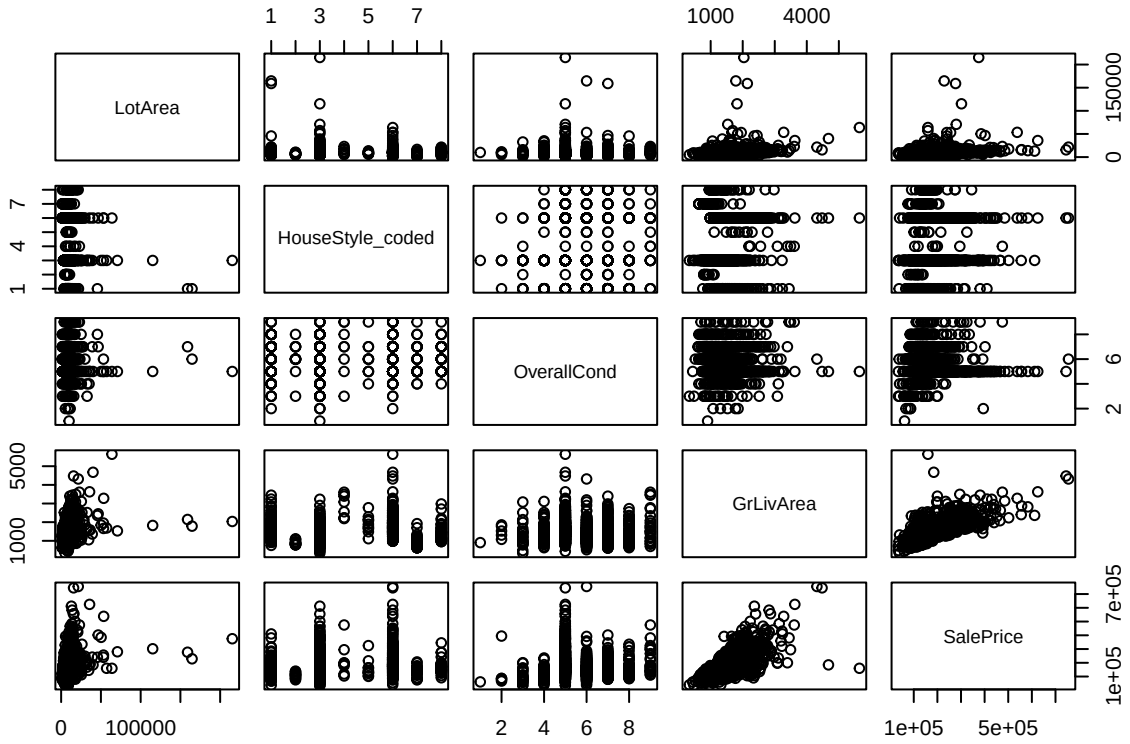
Now, let's look at the relationships among the features and their distributions:

```
library(Hmisc)
```

```
##
## Attaching package: 'Hmisc'
```

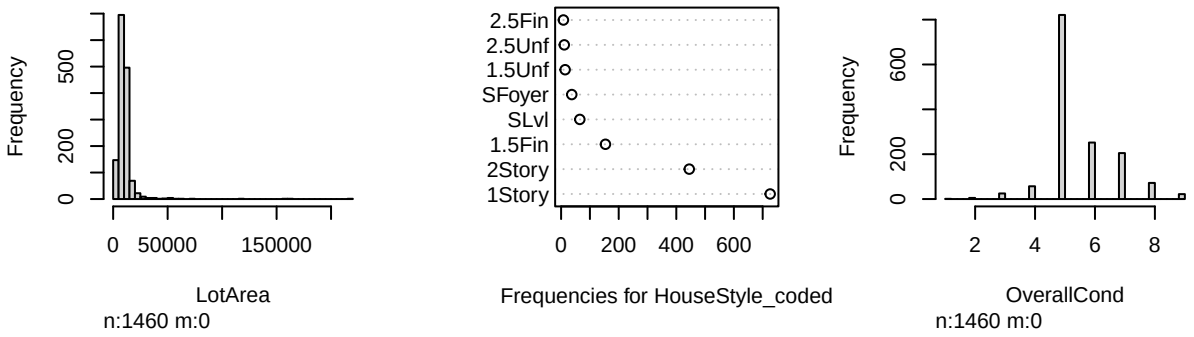
```
## The following objects are masked from 'package:base':
##
##     format.pval, units
```

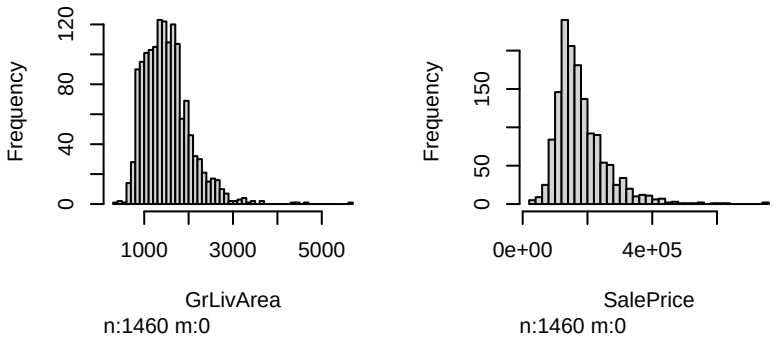
```
pairs(newdat1)
```



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:

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

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

Also,

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

where the noninformative prior for σ^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_codedSFoyer[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)
```

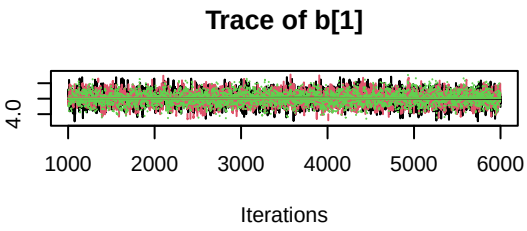
```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 1460
##    Unobserved stochastic nodes: 11
##    Total graph size: 17039
##
## Initializing model
```

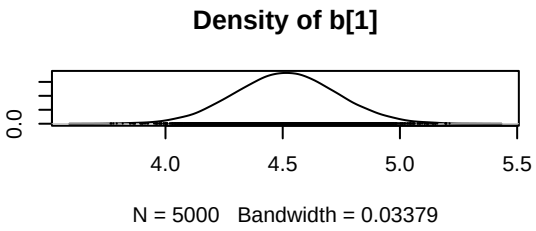
Fit the Model using the Monte Carlo-Markov Chain (MCMC) Sampler

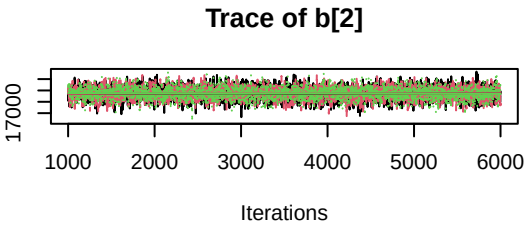
```
update(mod1_jags, 1000) # burn-in

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
```

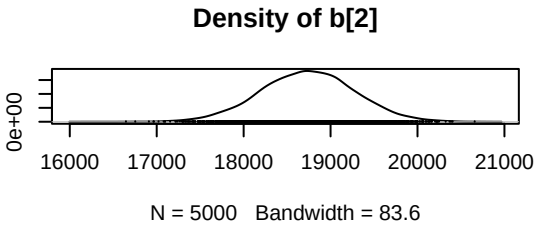
Check the Model by Examining Convergence Diagnostics

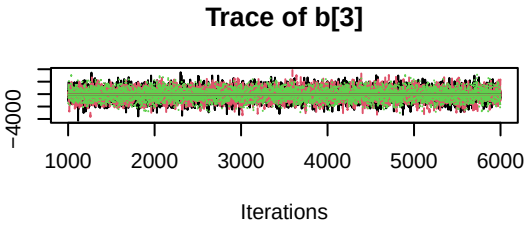
```
plot(mod1_jags_sim)
```

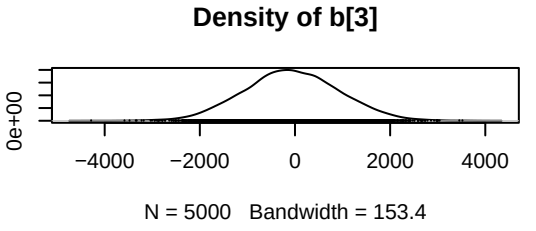

Trace of b[1]
4.0
1000
2000
3000
4000
5000
6000
Iterations

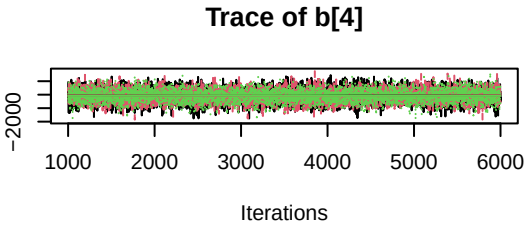

Density of b[1]
0.0
4.0
4.5
5.0
5.5
N = 5000 Bandwidth = 0.03379

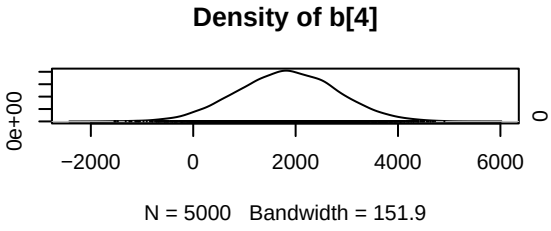

Trace of b[2]
17000
1000
2000
3000
4000
5000
6000
Iterations

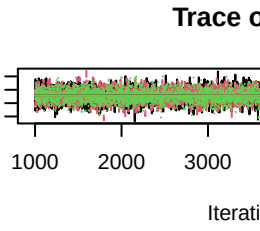

Density of b[2]
0e+00
16000
17000
18000
19000
20000
21000
N = 5000 Bandwidth = 83.6

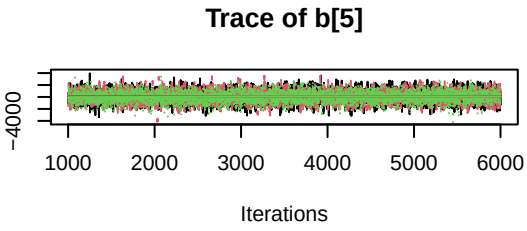

Trace of b[3]
−4000
1000
2000
3000
4000
5000
6000
Iterations

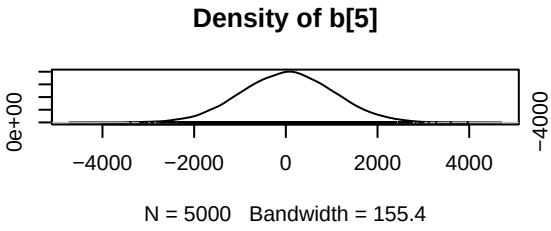

Density of b[3]
0e+00
−4000
−2000
0
2000
4000
N = 5000 Bandwidth = 153.4

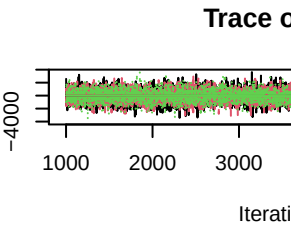

Trace of b[4]
−2000
1000
2000
3000
4000
5000
6000
Iterations

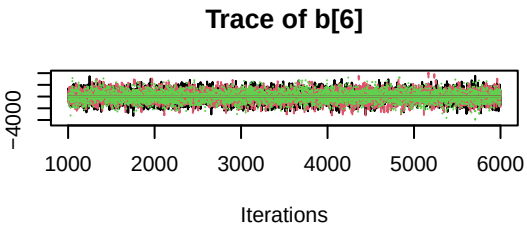

Density of b[4]
0e+00
−2000
0
2000
4000
6000
N = 5000 Bandwidth = 151.9

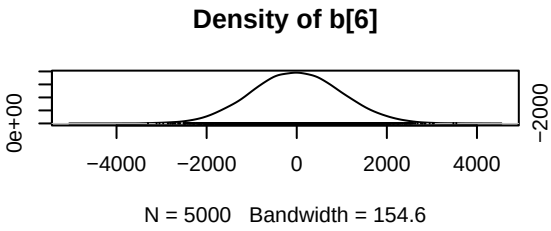

Trace o
0
1000
2000
3000
Iterati

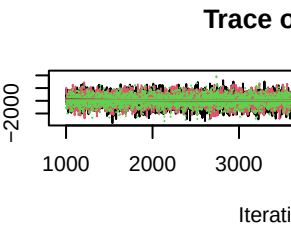

Trace of b[5]
−4000
1000
2000
3000
4000
5000
6000
Iterations

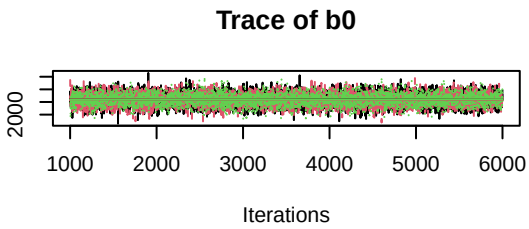

Density of b[5]
0e+00
−4000
−2000
0
2000
4000
N = 5000 Bandwidth = 155.4

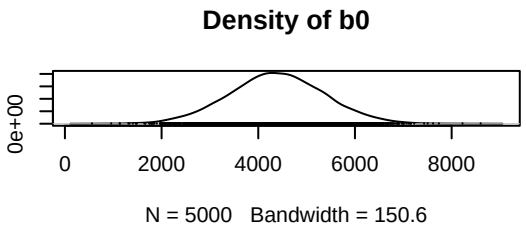

Trace o
−4000
1000
2000
3000
Iterati

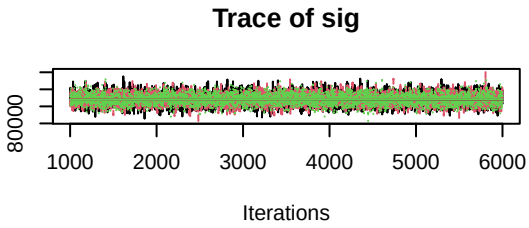

Trace of b[6]
−4000
1000
2000
3000
4000
5000
6000
Iterations

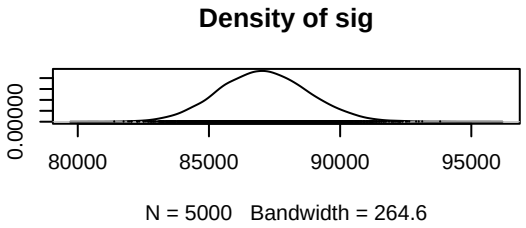

Density of b[6]
0e+00
−4000
−2000
0
2000
4000
N = 5000 Bandwidth = 154.6


Trace o
−2000
1000
2000
3000
Iterati










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

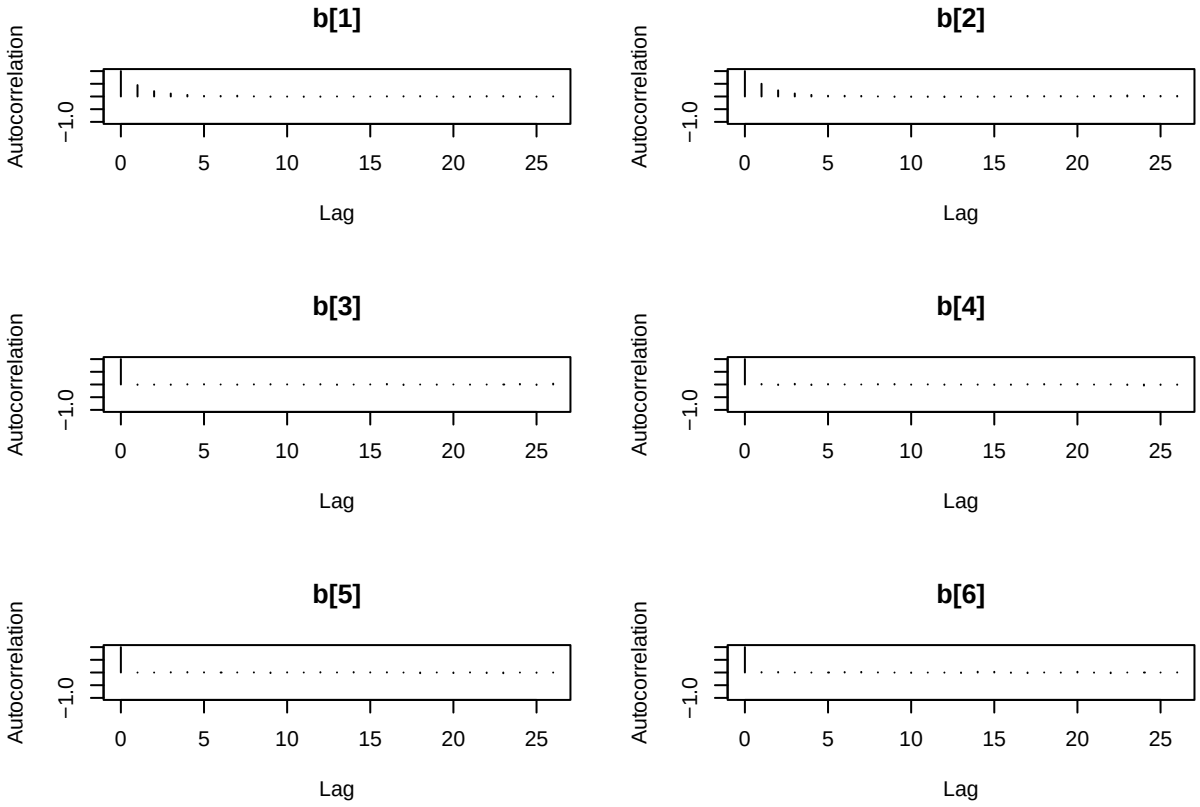
```
## Potential scale reduction factors:
##
##      Point est. Upper C.I.
## b[1]          1          1
## b[2]          1          1
## b[3]          1          1
## b[4]          1          1
## b[5]          1          1
## b[6]          1          1
## b[7]          1          1
## b[8]        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

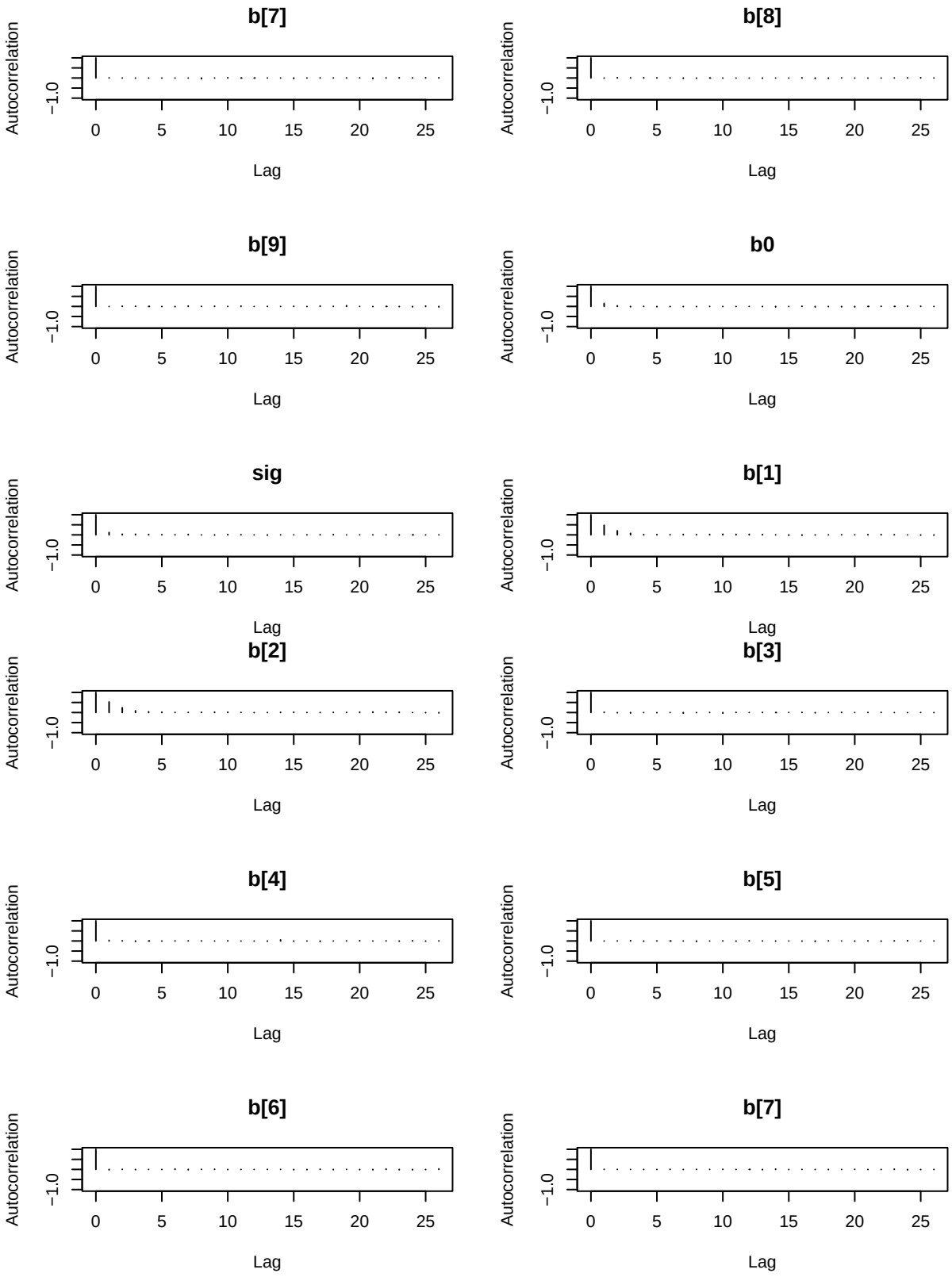
```
autocorr.diag(mod1_jags_sim)
```

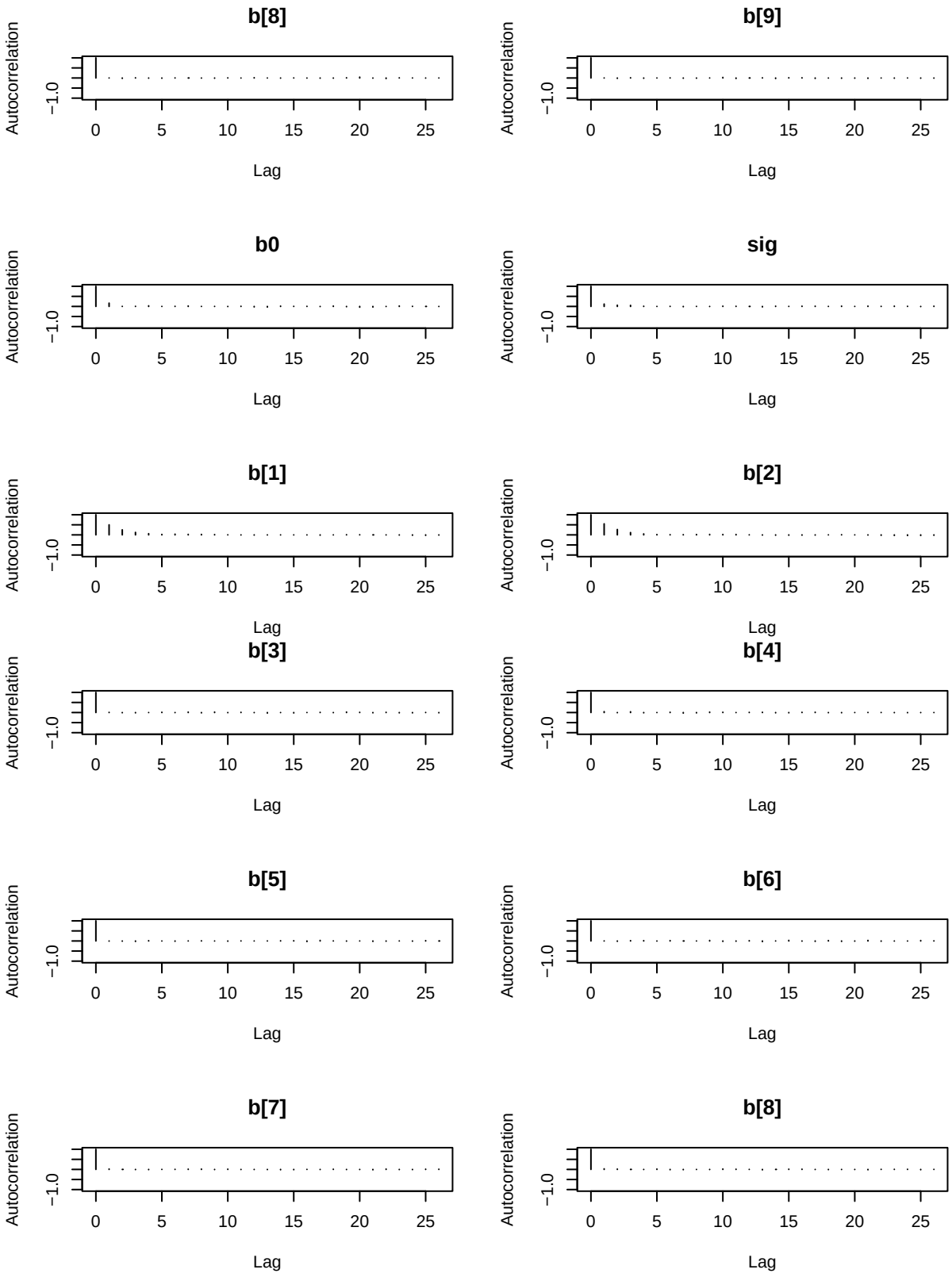
```
##               b[1]         b[2]         b[3]         b[4]          b[5]
## Lag 0   1.00000000  1.000000000  1.000000000  1.000000000  1.0000000000
## Lag 1   0.47460490  0.524571736  0.003371082  0.031754093 -0.0045776000
## Lag 5   0.02096458  0.022897461  0.004617954  0.002847906  0.0039815064
## Lag 10  0.01047594  0.008933128 -0.012267435  0.006600143  0.0007777233
## Lag 50 -0.01125489 -0.014328839 -0.007650862 -0.007958423  0.0084409505
##                b[6]         b[7]         b[8]         b[9]           b0
## Lag 0   1.000000000  1.000000000  1.000000000  1.000000000  1.000000000
## Lag 1  -0.002366874  0.011446359  0.008006565  0.005709090  0.148350360
## Lag 5  -0.003154483 -0.001745672  0.006210926  0.011225990 -0.009641888
## Lag 10 -0.015882997  0.012309154  0.007536047  0.017396400 -0.007493370
## Lag 50 -0.001941479 -0.017610101 -0.003082874 -0.008708285 -0.003344340
##                 sig
## Lag 0   1.000000000
## Lag 1   0.116794886
```

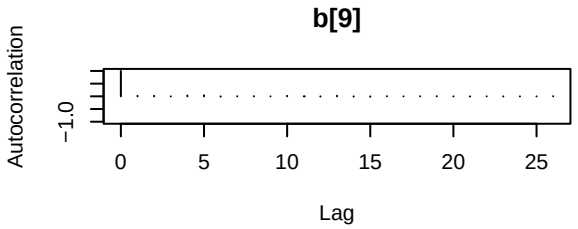
```
## Lag 5   0.006494652
## Lag 10  0.014014275
## Lag 50 -0.002976607
```

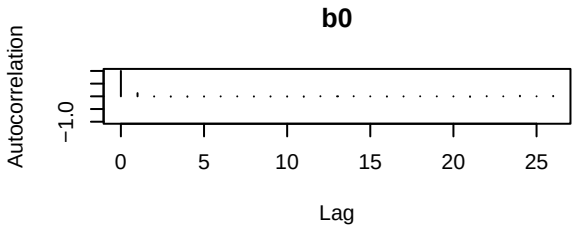
```
autocorr.plot(mod1_jags_sim)
```

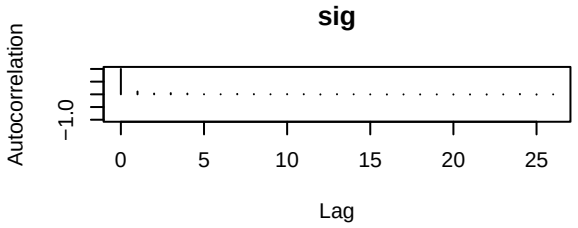



b[7]
b[8]
b[9]
b0
sig
b[1]
b[2]
b[3]
b[4]
b[5]
b[6]
b[7]
Autocorrelation
−1.0
0
5
10
15
20
25
Lag


b[8]
b[9]
b0
sig
b[1]
b[2]
b[3]
b[4]
b[5]
b[6]
b[7]
b[8]
Autocorrelation
−1.0
0
5
10
15
20
25
Lag








```
effectiveSize(mod1_jags_sim)
```

```
##      b[1]      b[2]      b[3]      b[4]      b[5]      b[6]      b[7]      b[8]
##  5453.508  5006.544 15000.000 13952.948 15000.000 15240.288 15000.000 14719.277
##      b[9]        b0       sig
## 14034.235 11213.777 10384.977
```

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

```
## Mean deviance:  37363
## penalty 3.142
## Penalized deviance: 37366
```

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

```
summary(mod1_jags_sim)
```

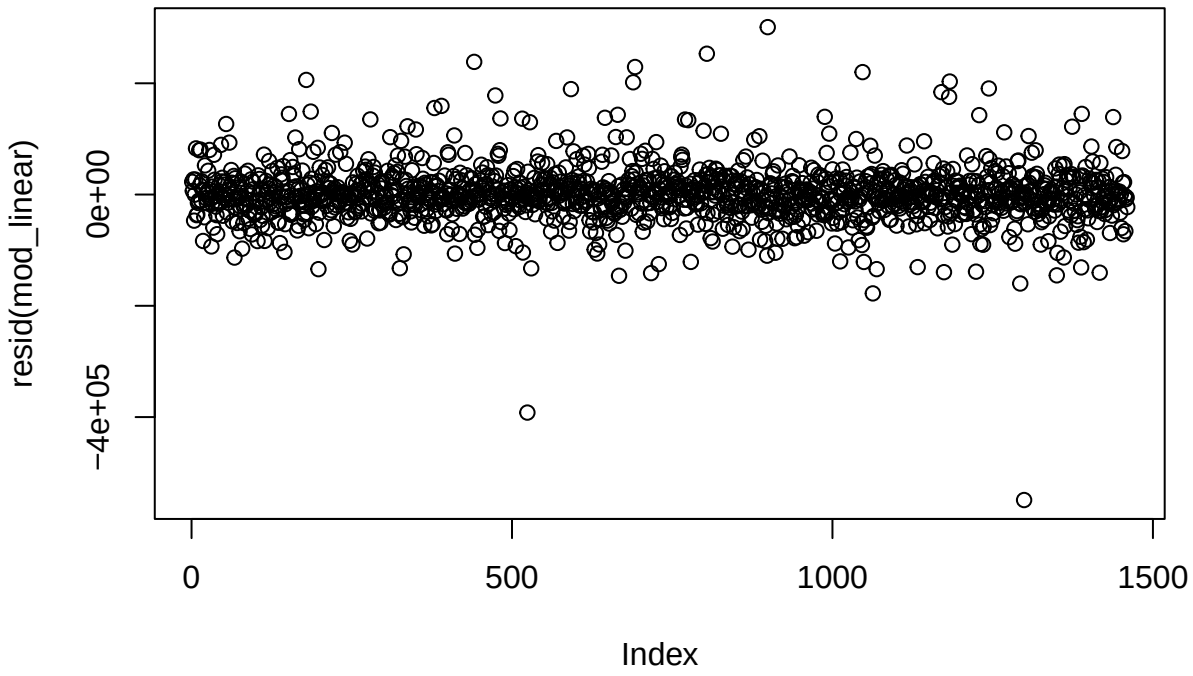
```
##
## Iterations = 1001:6000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 5000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##           Mean        SD  Naive SE Time-series SE
## b[1]     4.514    0.2184  0.001783       0.002965
## b[2] 18736.547  539.6437  4.406172       7.633348
## b[3]   -65.294  996.7326  8.138287       8.138778
## b[4]  1836.566  980.2418  8.003641       8.312726
## b[5]    41.764 1003.4062  8.192777       8.192966
## b[6]    -6.491  998.2810  8.150931       8.088538
## b[7]  3097.165  988.4776  8.070886       8.070527
## b[8]   -48.085  999.4872  8.160779       8.241835
## b[9]    47.018  987.2230  8.060642       8.372615
## b0    4383.798  983.3713  8.029193       9.311228
## sig  87093.361 1707.7680 13.943867      16.766095
##
```

```
## 2. Quantiles for each variable:
##
##           2.5%       25%       50%      75%     97.5%
## b[1]     4.084     4.368     4.514     4.66     4.949
## b[2] 17673.926 18370.876 18738.498 19095.85 19794.108
## b[3] -1976.837  -727.763   -77.439   599.39  1893.304
## b[4]   -61.981  1179.993  1841.378  2501.71  3753.215
## b[5] -1913.863  -636.420    44.459   720.42  2022.659
## b[6] -1935.947  -678.178   -12.071   667.07  1948.073
## b[7]  1108.984  2427.583  3103.949  3765.10  5041.550
## b[8] -1993.640  -712.342   -51.317   618.29  1920.212
## b[9] -1897.734  -612.347    45.446   709.07  1962.465
## b0    2437.751  3735.912  4375.494  5038.75  6334.057
## sig  83837.875 85918.898 87066.807 88223.76 90484.657
```

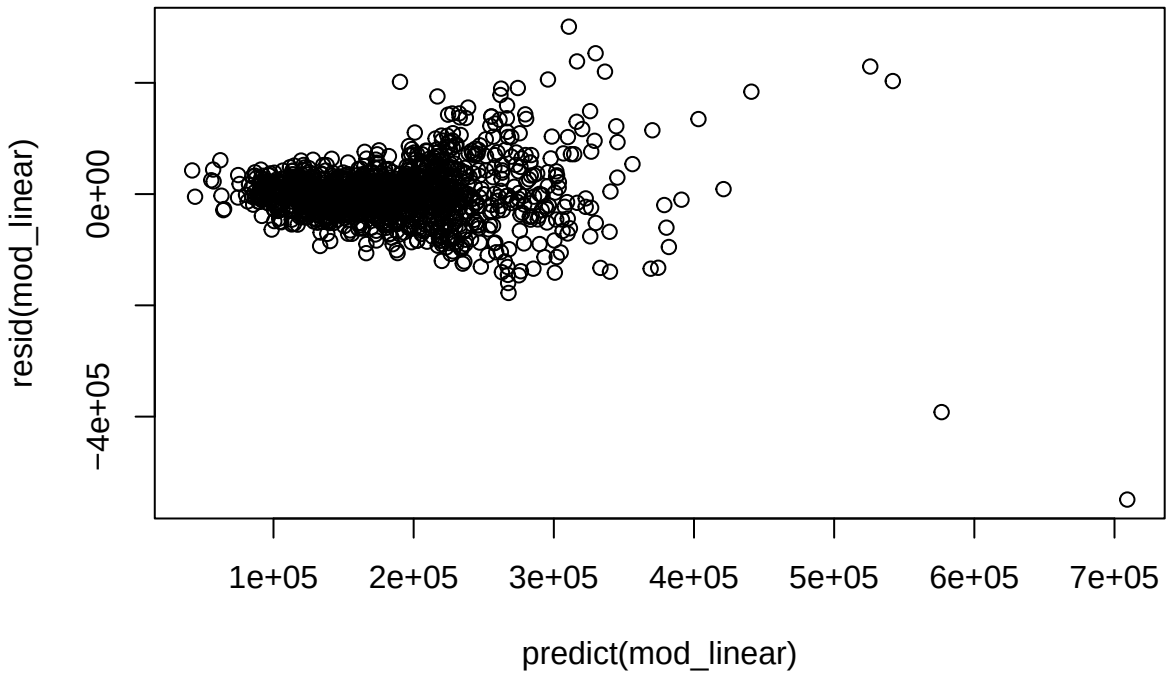
Residual checks

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:

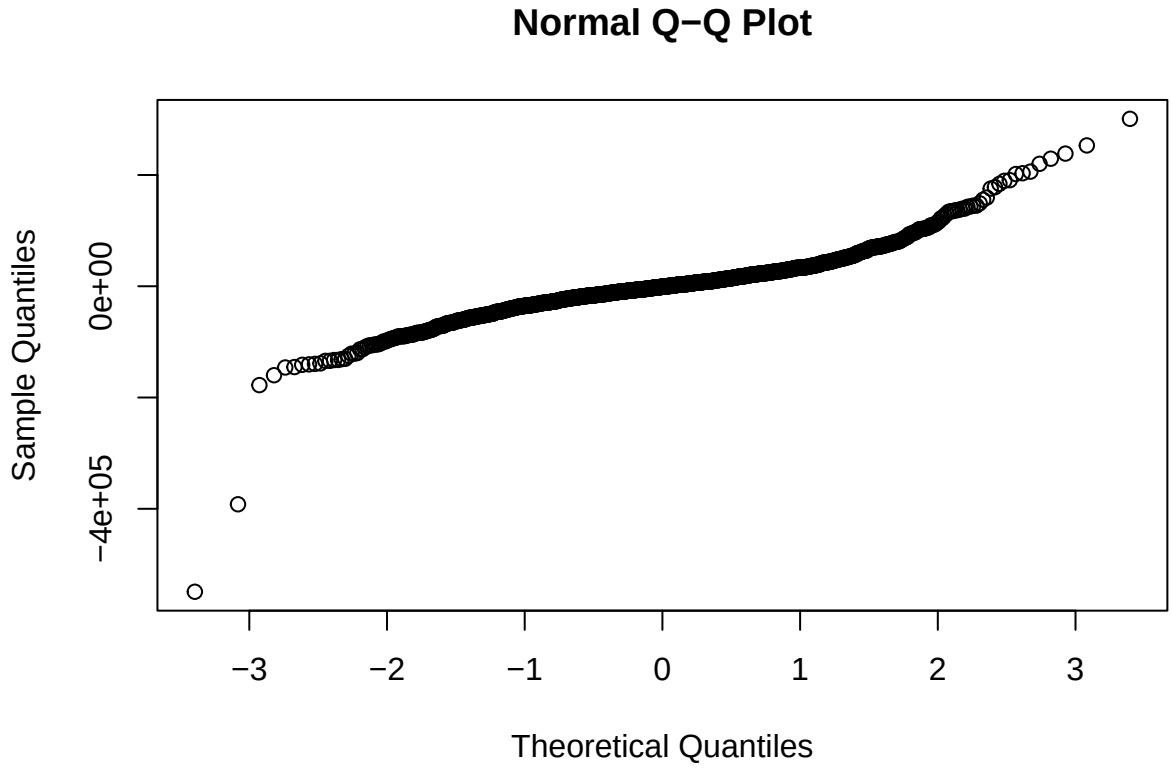
```
plot(resid(mod_linear)) # to check independence (looks okay)
```




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




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



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.

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

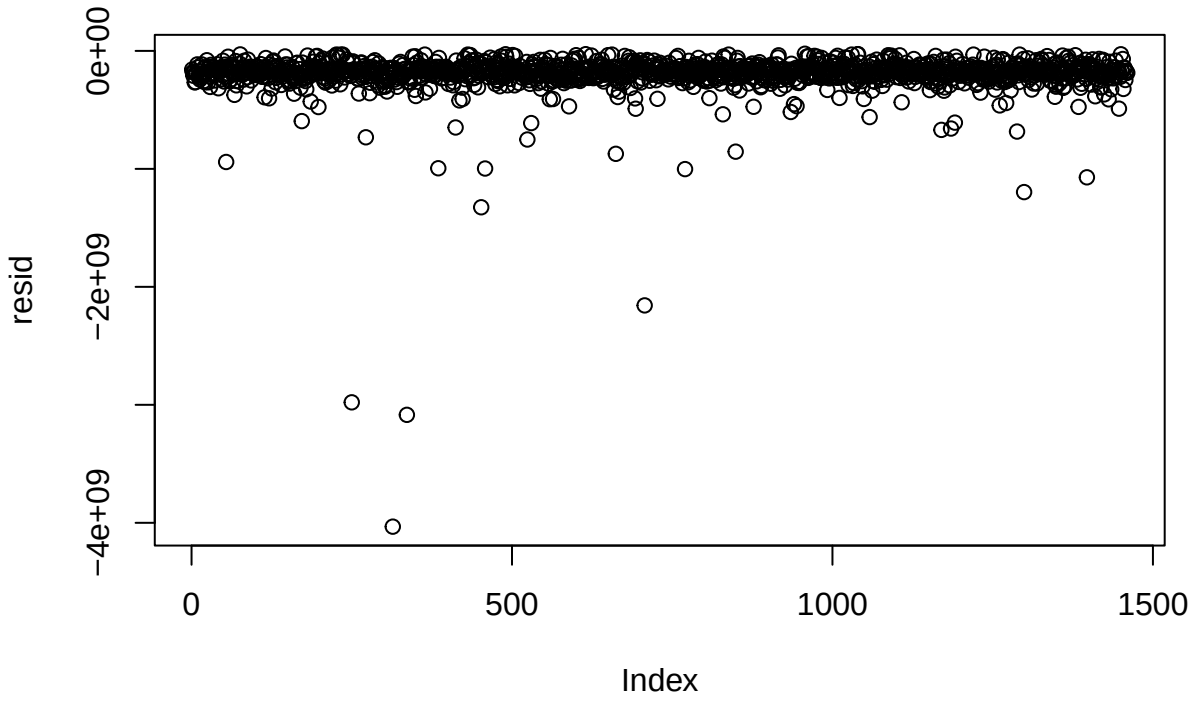
```
          data_jags$HouseStyle_coded1.5Unf,
          data_jags$HouseStyle_coded1Story,
          data_jags$HouseStyle_coded2.5Fin,
          data_jags$HouseStyle_coded2.5Unf,
          data_jags$HouseStyle_coded2Story,
          data_jags$HouseStyle_codedSFoyer,
          data_jags$HouseStyle_codedSLvl
          )
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    1    0     0
## [4,]    1  9550    5    0    0    0    0    1    0     0
## [5,]    1 14260    5    0    0    0    0    1    0     0
## [6,]    1 14115    5    0    0    0    0    0    0     0
```

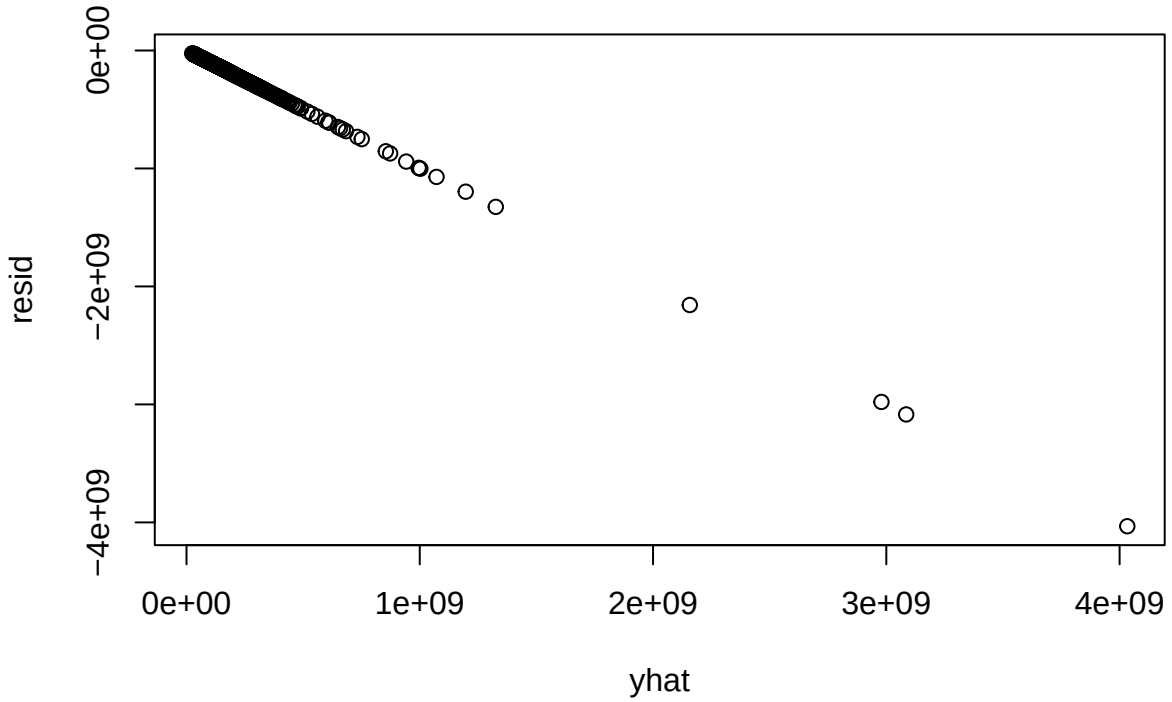
```
(pm_params = colMeans(mod1_jags_csim)) # posterior mean
```

```
##         b[1]         b[2]         b[3]         b[4]         b[5]         b[6]
##     4.514125 18736.547364   -65.294309  1836.566314    41.764420    -6.491105
##         b[7]         b[8]         b[9]           b0          sig
##  3097.164534   -48.085192    47.018280  4383.797873 87093.361381
```

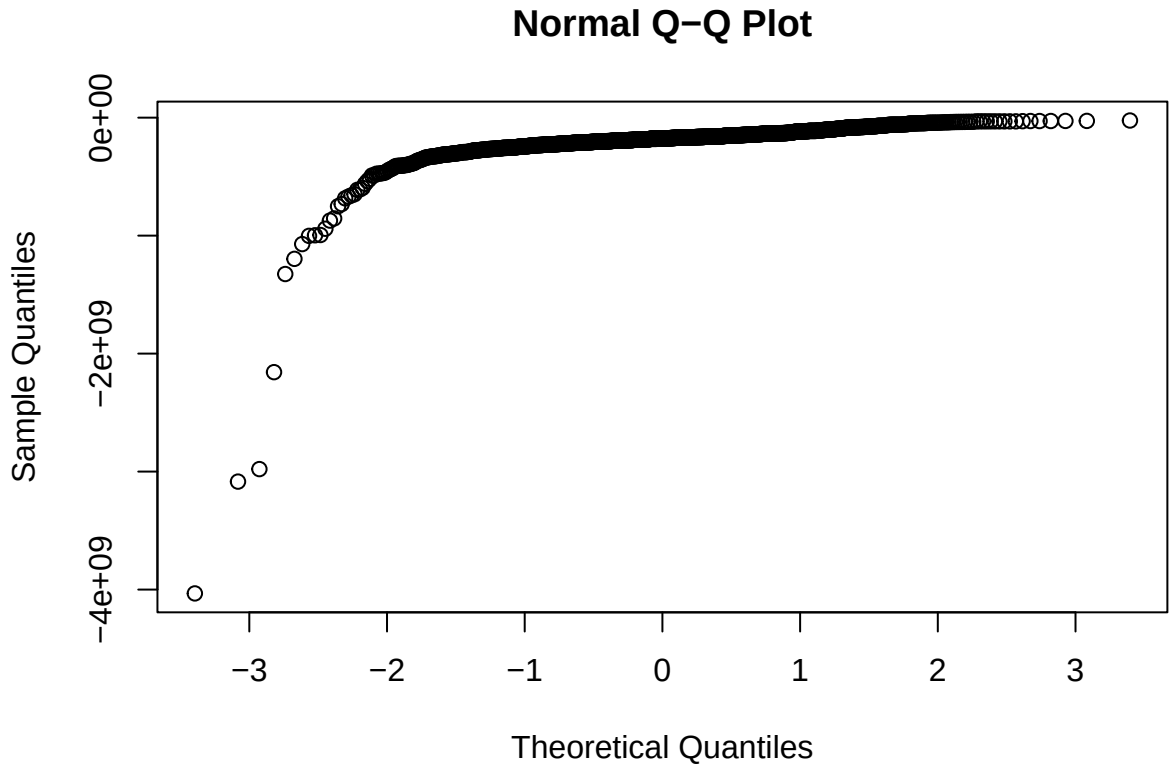
```
yhat = drop(X %*% pm_params[1:10])
resid = data_jags$y - yhat
plot(resid) # residuals against data index
```



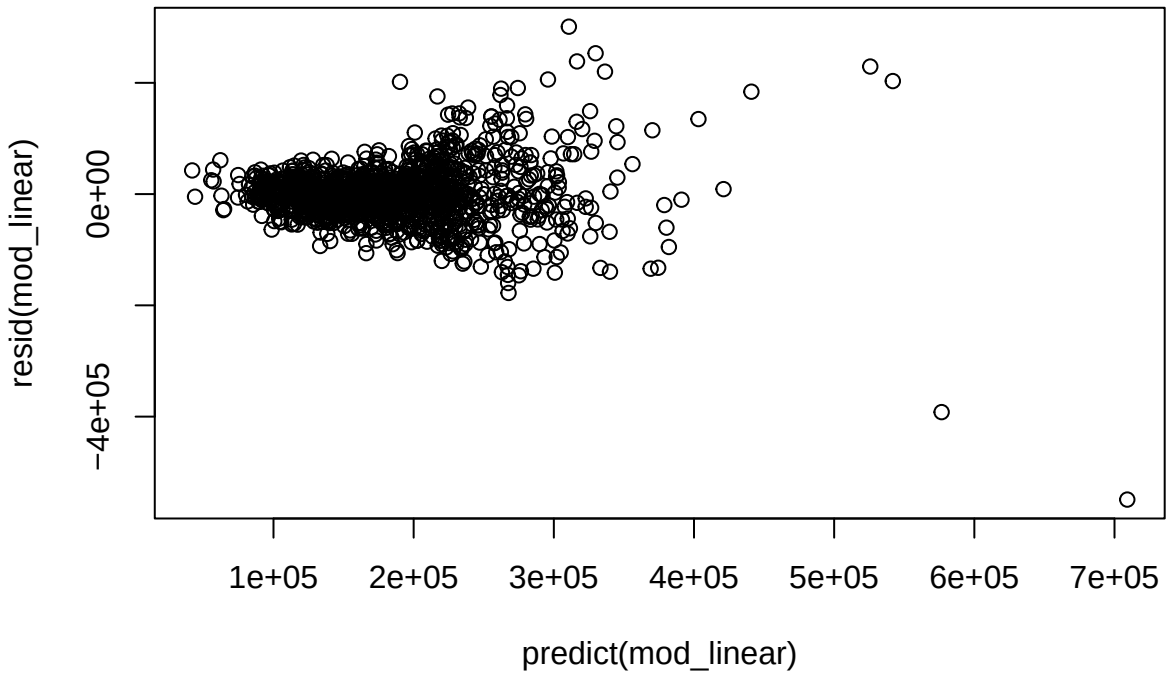

```
plot(yhat, resid) # residuals against predicted values
```




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




```
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]
```

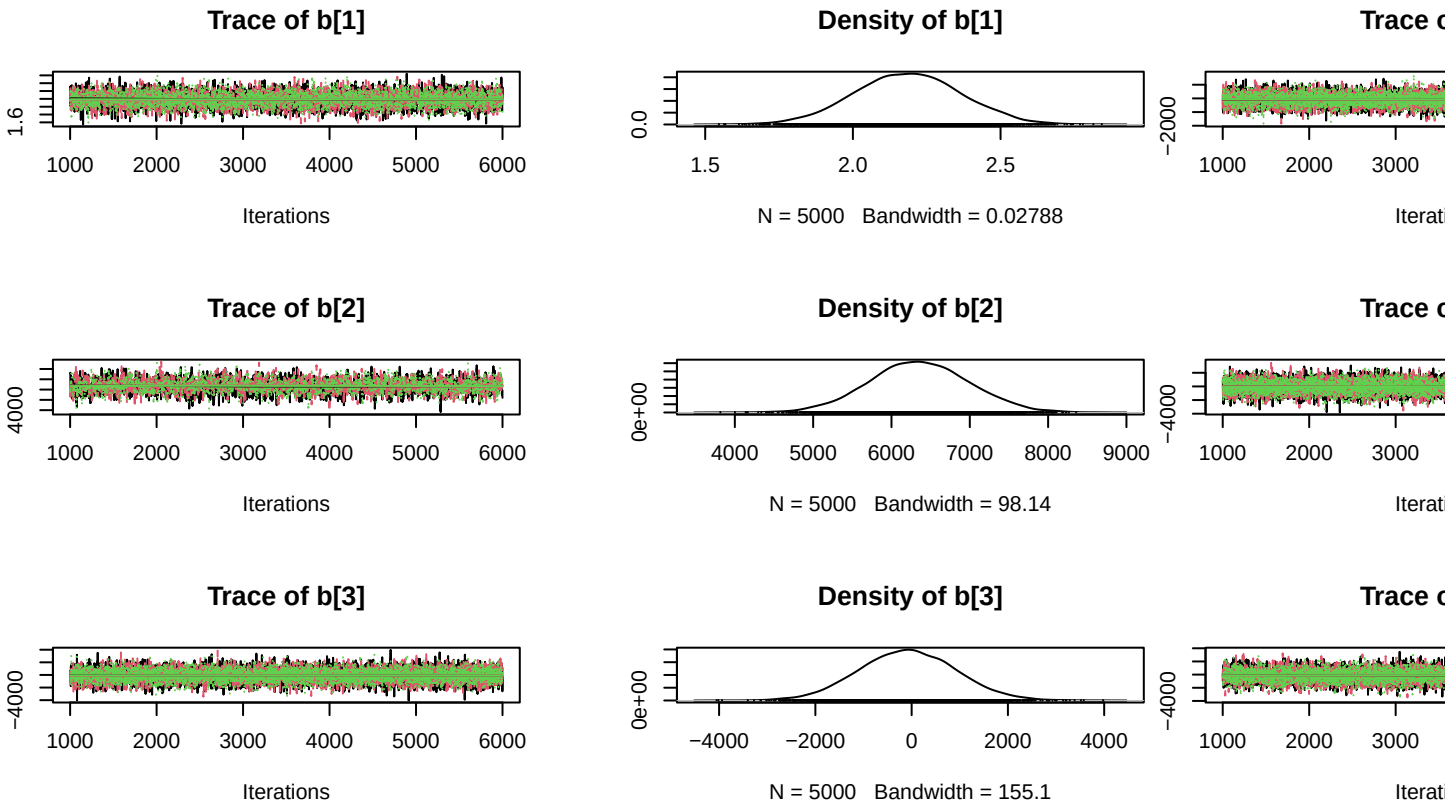
```
                + b[8]*HouseStyle_codedSFoyer[i]
                + b[9]*HouseStyle_codedSLvl[i]
                + b[10]*TotRmsAbvGrd[i]
    }

    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)
```

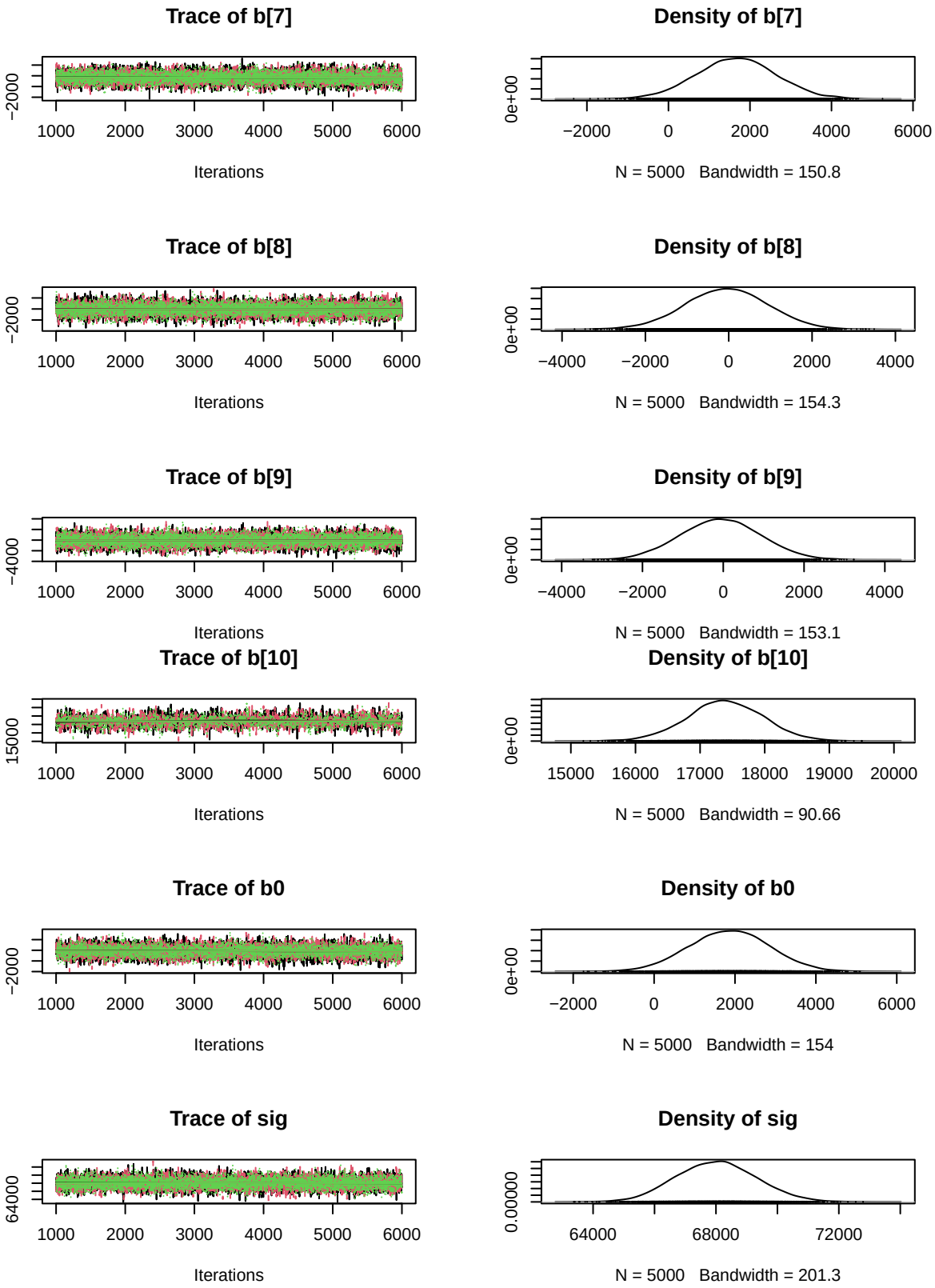
```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 1460
##    Unobserved stochastic nodes: 12
##    Total graph size: 18584
##
## Initializing model
```

```
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)
```

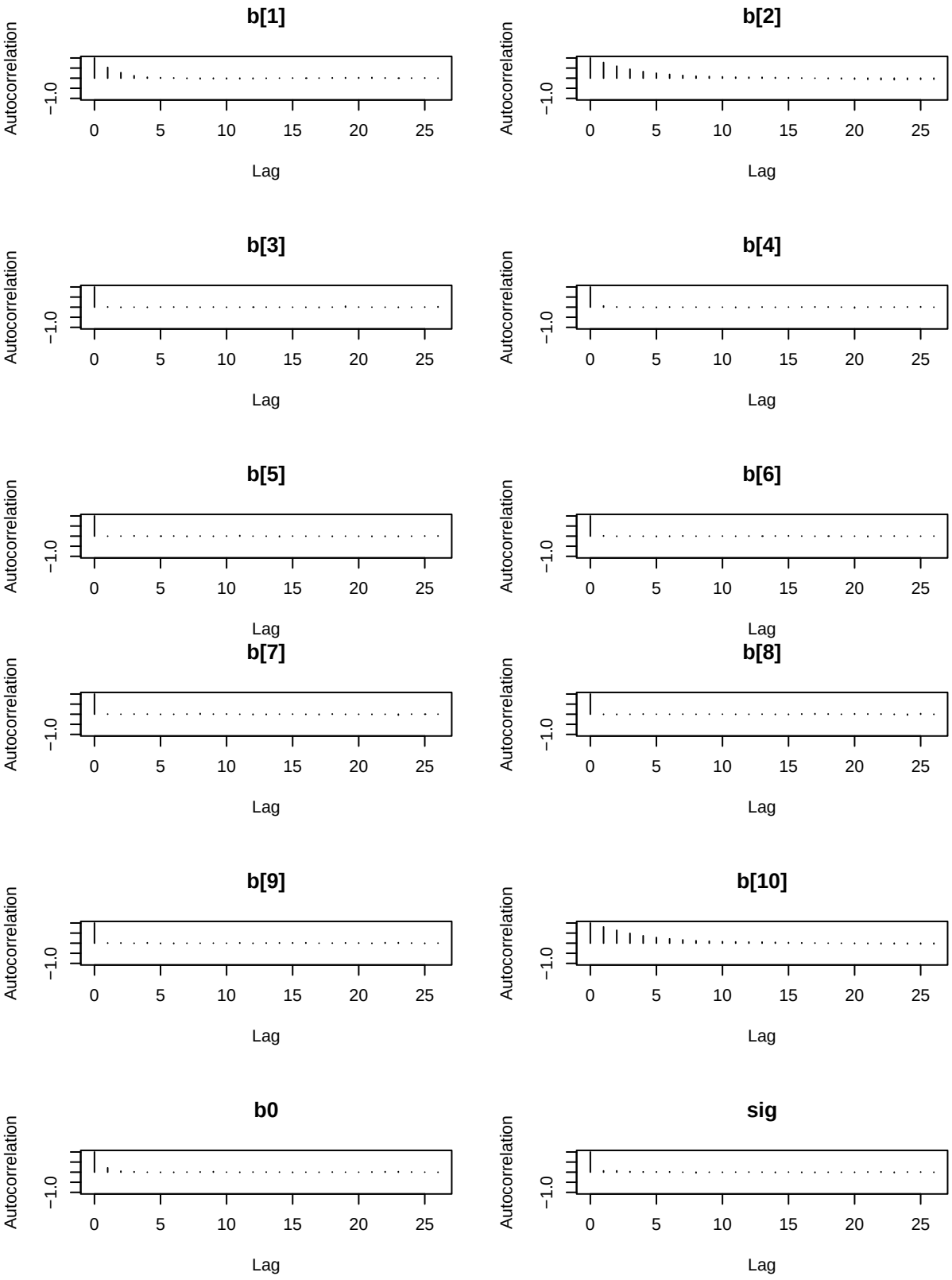
```
## Potential scale reduction factors:
```

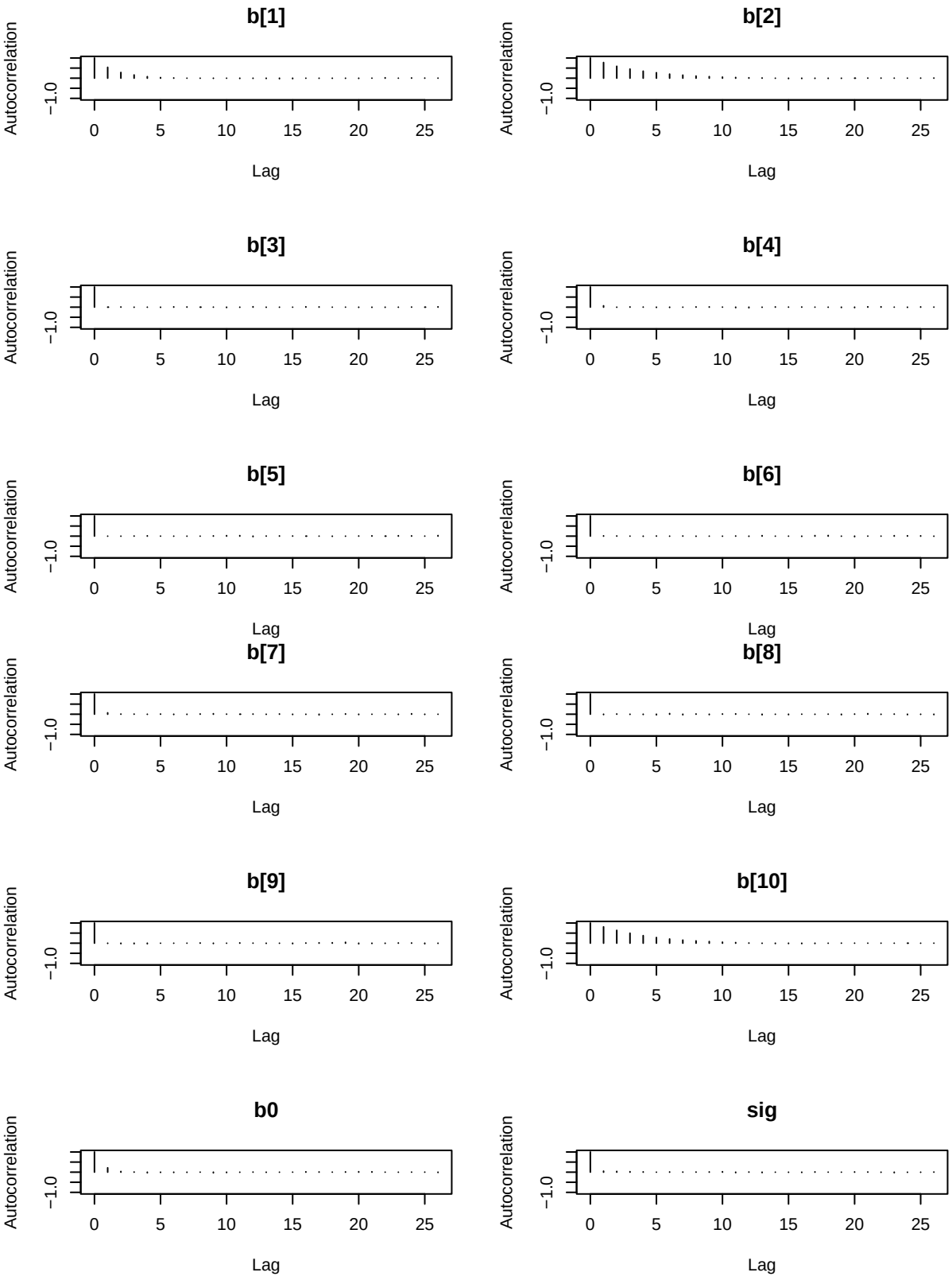
```
##
##       Point est. Upper C.I.
## b[1]           1       1.00
## b[2]           1       1.00
## b[3]           1       1.00
## b[4]           1       1.00
## b[5]           1       1.00
## b[6]           1       1.00
## b[7]           1       1.00
## b[8]           1       1.00
## b[9]           1       1.00
## b[10]          1       1.01
## b0             1       1.00
## sig            1       1.00
##
## Multivariate psrf
##
## 1
```

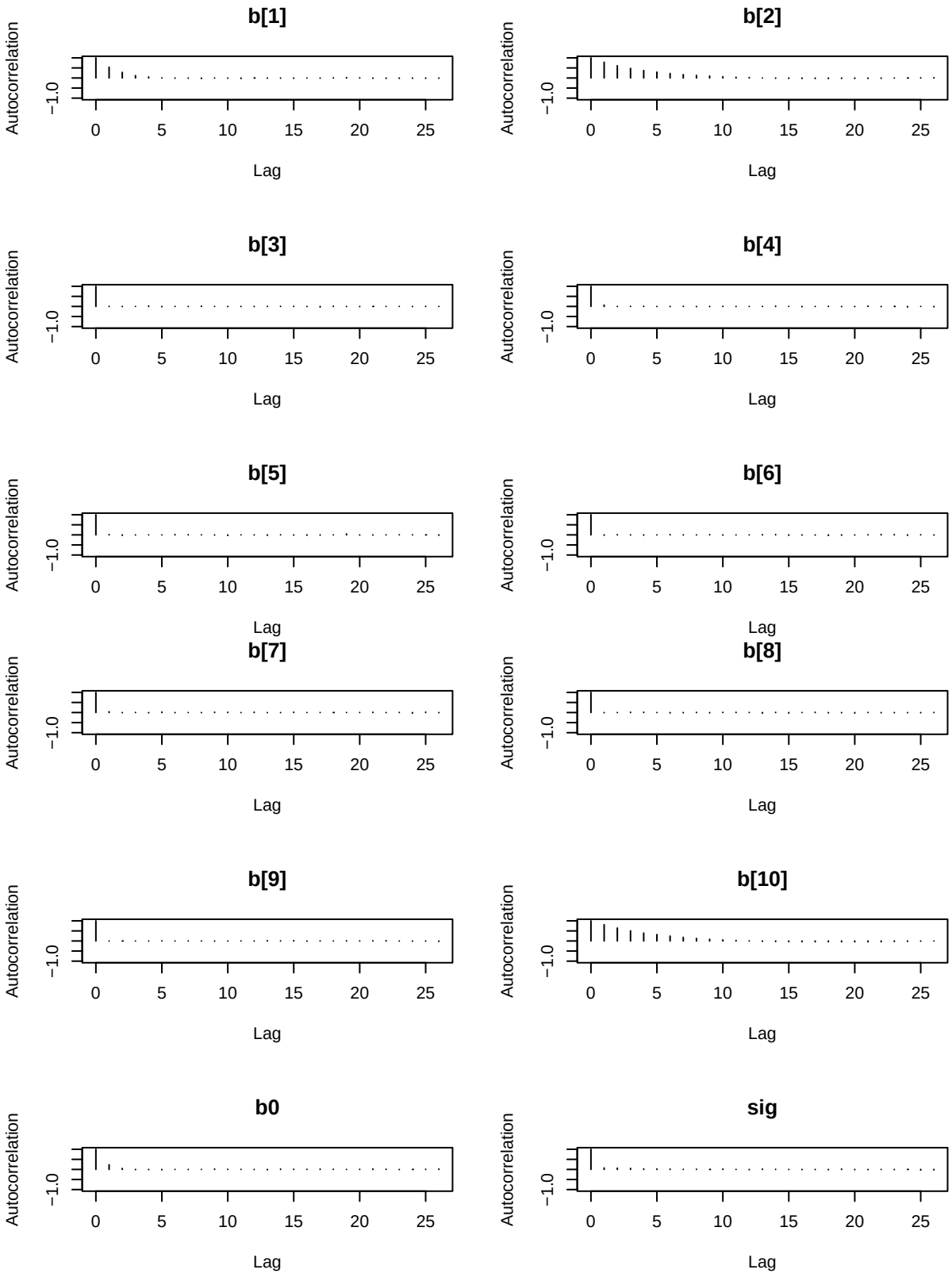
```
autocorr.diag(mod2_jags_sim)
```

```
##              b[1]        b[2]         b[3]         b[4]          b[5]
## Lag 0   1.00000000  1.00000000  1.000000000  1.000000000  1.0000000000
## Lag 1   0.53762634  0.77881787  0.006735773  0.063625122  0.0030100895
## Lag 5   0.02103064  0.27068907 -0.005648175 -0.013331055  0.0001905435
## Lag 10 -0.01414764  0.06153858 -0.011878209  0.003881659  0.0008160931
## Lag 50  0.01168580 -0.02183639 -0.013644420 -0.003337660 -0.0053508246
##               b[6]        b[7]          b[8]         b[9]       b[10]
## Lag 0   1.000000000 1.000000000  1.0000000000  1.000000000  1.00000000
## Lag 1   0.005029371 0.038929266 -0.0126826078 -0.001714391  0.80787761
## Lag 5  -0.006579041 0.008237473 -0.0093687156 -0.003710024  0.29357127
## Lag 10 -0.004109778 0.012049151  0.0078875323 -0.003345627  0.06028108
## Lag 50 -0.013155685 0.002606142 -0.0001150594  0.010121351 -0.01190270
##                 b0           sig
## Lag 0   1.000000000  1.0000000000
## Lag 1   0.219986915  0.0626872198
## Lag 5  -0.014007882  0.0127105106
## Lag 10 -0.004271829  0.0112395285
## Lag 50 -0.016096961 -0.0002685658
```

```
autocorr.plot(mod2_jags_sim)
```


b[1]
b[2]
b[3]
b[4]
b[5]
b[6]
b[7]
b[8]
b[9]
b[10]
b0
sig
Autocorrelation
−1.0
0
5
10
15
20
25
Lag


b[1]
b[2]
b[3]
b[4]
b[5]
b[6]
b[7]
b[8]
b[9]
b[10]
b0
sig
Autocorrelation
−1.0
0
5
10
15
20
25
Lag




```
effectiveSize(mod2_jags_sim)
```

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

```
##  4967.197  1963.064 15000.000 13203.054 15093.066 14802.933 13966.788 15558.708
##      b[9]     b[10]        b0       sig
## 15992.859  1861.602  9888.349 11329.088
```

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

```
## Mean deviance:  37362
## penalty 3.197
## Penalized deviance: 37365
```

```
dic.samples(mod2_jags, n.iter=1e3)
```

```
## Mean deviance:  36640
## penalty 3.785
## Penalized deviance: 36644
```

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.