Bayesian Lasso of Loading Data

Greg Johnson

We fit the following model to the heating load and cooling load dataset:

$$\mathbf{y} \sim N(\mathbf{1}_n\mu + X\boldsymbol{\beta}, \sigma^2 I_n)$$

$$\boldsymbol{\beta}|\Sigma_0 \sim N(\mathbf{0}, \sigma^2\Sigma_0 = \sigma^2 \cdot \text{diag}(\tau_1^2, ..., \tau_p^2))$$

$$\boldsymbol{\tau}^2|\lambda \sim \prod_{j=1}^{k} \text{Exp}(\lambda^2/2)$$

With priors:

$$p(\mu) \propto 1$$

$$p(\sigma^2) \propto (\sigma^2)^{-1}$$

$$\lambda^2 \sim \Gamma(0.01, 0.01)$$

Note that the use of $\boldsymbol{\tau^2}$ is the equivalent to specifying a Laplace prior on $\boldsymbol{\beta}$:

$$p(\boldsymbol{\beta}|\sigma^2|\lambda) = \prod_{j=1}^{k} \frac{\lambda}{2\sigma^2} \exp\Big(- \frac{\lambda|\beta_j|}{\sigma} \Big)$$

Derive Posteriors

We will take the following two-step approach. I have suppressed the iteration notation for readability.

1. Use Gibbs to approximately sample $(\boldsymbol{\beta}, \boldsymbol{\tau^2}, \lambda, \sigma^2)$.
2. Use the above Gibbs draw to conditionally sample μ from:

$$\mu|\boldsymbol{\beta}, \boldsymbol{\tau^2}, \lambda, \sigma^2, \mathbf{y} \sim \text{N}(\bar{y}, \sigma^2/n)$$

The Gibbs sampling itself will involve four steps, sampling from each of the conditional posteriors for $\boldsymbol{\beta}$, $\boldsymbol{\tau^2}$, λ, and σ^2. The posteriors come from *The Bayesian Lasso* (Park & Casella, 2008).

1. Sample $\boldsymbol{\beta}$ from:

$$\boldsymbol{\beta}|\boldsymbol{\tau^2}, \sigma^2, \lambda, \mathbf{y} \sim \text{N}\Big(\boldsymbol{\mu} = A^{-1}X^T\tilde{\boldsymbol{y}}, \Sigma = \sigma^2 A^{-1}\Big)$$
$$A = X^T X + \Sigma_0^{-1}$$

2. Sample $\boldsymbol{\tau^2}$ by independently sampling:

$$\frac{1}{\tau_j^2}|\boldsymbol{\beta}, \sigma^2, \lambda, \mathbf{y} \sim \text{Inv.N}\Big(\mu' = \frac{\lambda(\sigma^2)^{1/2}}{\beta_j}, \lambda' = \lambda^2\Big)$$

3. Sample σ^2 from:

$$\sigma^2|\boldsymbol{\tau^2}, \boldsymbol{\beta}, \lambda, \mathbf{y} \sim \Gamma^{-1}\left(\frac{n-1}{2} + \frac{k}{2}, \frac{1}{2}(\tilde{\boldsymbol{y}} - X\boldsymbol{\beta})^T(\tilde{\boldsymbol{y}} - X\boldsymbol{\beta}) + \frac{1}{2}\boldsymbol{\beta}^T\Sigma_0^{-1}\boldsymbol{\beta}\right)$$

4. Sample λ from:

$$\lambda^2|\sigma^2, \boldsymbol{\tau^2}, \boldsymbol{\beta}, \mathbf{y} \sim \Gamma\Big(k + 0.01, \frac{1}{2}\sum_{j=1}^{k}\tau_j^2 + 0.01\Big)$$

Bayesian Lasso of Heating Load

First we fit the Bayesian model using y_1 as the response.

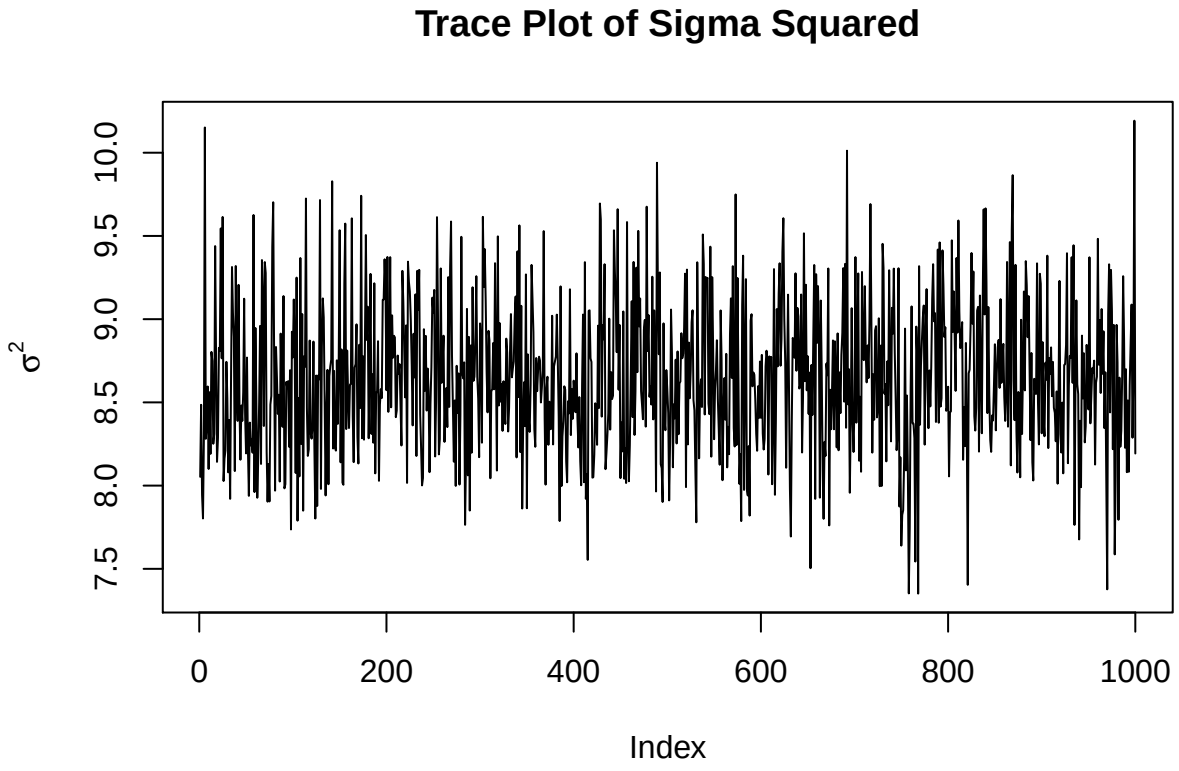
Implement MCMC

```
# read in data
Dat = read.table("data/Load.txt", header = TRUE)
Y1 = Dat[["Y1"]]
Y2 = Dat[["Y2"]]
X = scale(as.matrix(Dat[, paste("X", 1:8, sep = "")]))

BayesianLasso = function(Y, X, ndraws, start) {
    Ytilde = Y - mean(Y)
    n = nrow(X)
    k = ncol(X)

    # set-up list to collect draws
    post_draws = list(`Sigma Squared` = numeric(ndraws), Mu = numeric(ndraws),
        Lambda = numeric(ndraws), `Tau Squared` = matrix(0, ndraws, k, dimnames = list(NULL,
            paste("Tau Squared", 1:k))), Beta = matrix(0, ndraws, k, dimnames = list(NULL,
            paste("Beta", 1:k))))

    # starting points
    tau_sq = start[["Tau Squared"]]
    sig_sq = start[["Sigma Squared"]]
    lambda = start[["Lambda"]]

    for (g in 1:ndraws) {
        # draw beta vector
        A = t(X) %*% X + solve(diag(tau_sq))
        beta = t(rmvnorm(1, mean = solve(A) %*% t(X) %*% Ytilde, sigma = sig_sq *
            solve(A)))
        # draw tau squared vector
        for (j in 1:k) {
            tau_sq[j] = 1/rinvgauss(1, mean = sqrt((lambda^2 * sig_sq)/beta[j]^2),
                shape = lambda^2)
        }
```

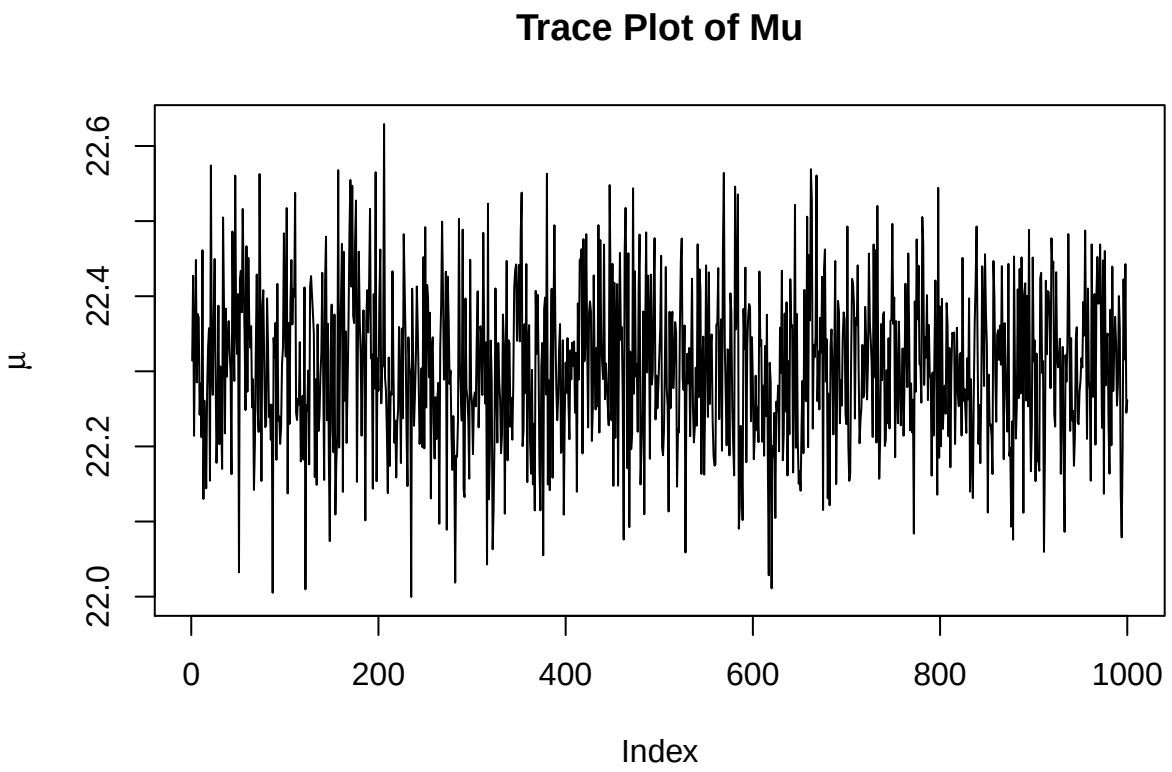
```
        # draw sigma squared

        sig_sq = rinvgamma(1, shape = (n - 1)/2 + k/2, rate = 1/2 * t(Ytilde -
            X %*% beta) %*% (Ytilde - X %*% beta) + 1/2 * t(beta) %*% solve(diag(tau_sq)) %*%
            beta)
        # draw lambda
        lambda = sqrt(rgamma(1, shape = k + 0.01, rate = 0.5 * sum(tau_sq) +
            0.01))
        # draw mu
        mu = rnorm(1, mean = mean(Y), sd = sqrt(sig_sq/n))
        # save parameters
        post_draws[["Beta"]][g, ] = beta
        post_draws[["Tau Squared"]][g, ] = tau_sq
        post_draws[["Sigma Squared"]][g] = sig_sq
        post_draws[["Lambda"]][g] = lambda
        post_draws[["Mu"]][g] = mu
    }
    return(post_draws)
}

start = list(`Tau Squared` = rep(1, ncol(X)), `Sigma Squared` = 1, Lambda = 1)

post_draws = BayesianLasso(Y1, X, 1000, start)
```

Convergence: Burn-in & Trace Plots

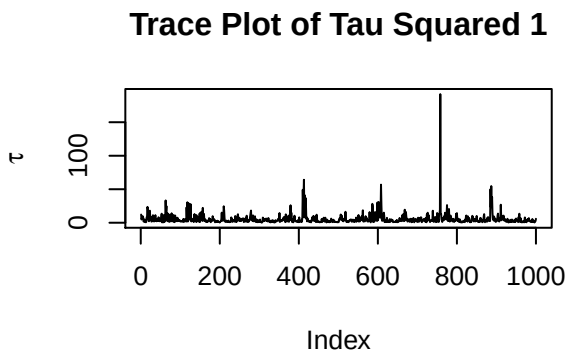
```
k = ncol(X)
plot(post_draws[["Sigma Squared"]], type = "l", xlab = "Index", ylab = expression(sigma^2),
    main = "Trace Plot of Sigma Squared")
```

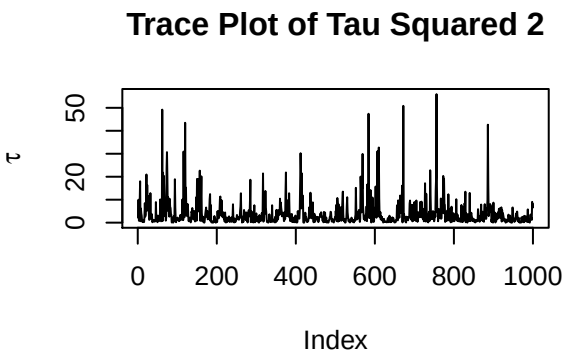


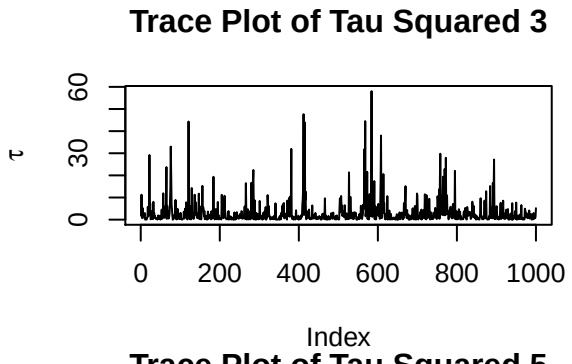
```r
plot(post_draws[["Mu"]], type = "l", xlab = "Index", ylab = expression(mu),
    main = "Trace Plot of Mu")
```

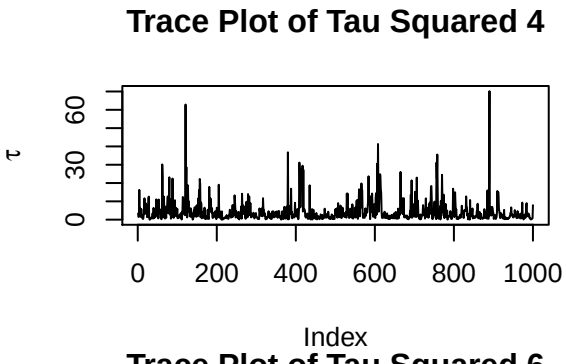


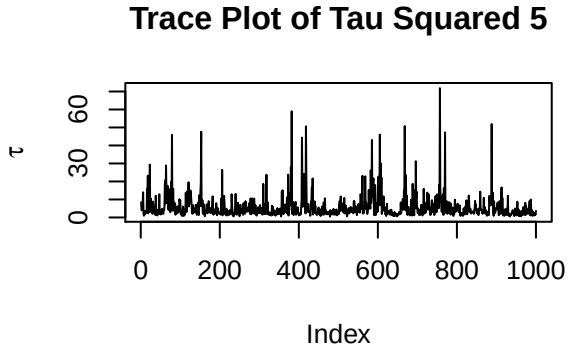

```r
par(mfrow = c(2, 2))
for (i in 1:k) {
    plot(post_draws[["Tau Squared"]][, i], type = "l", xlab = "Index", ylab = expression(tau),
        main = paste("Trace Plot of Tau Squared", i))
}
```

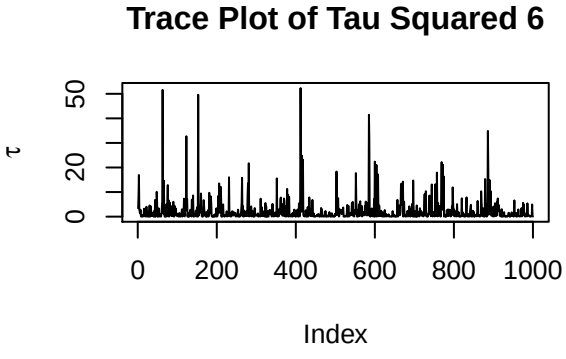


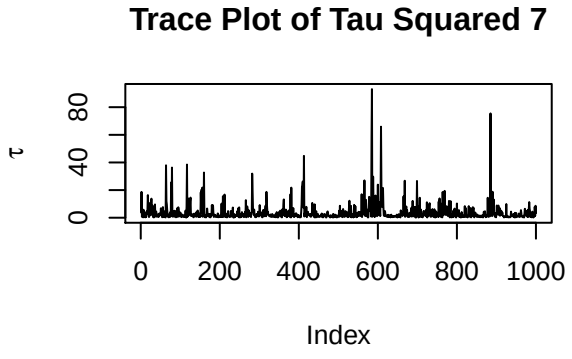


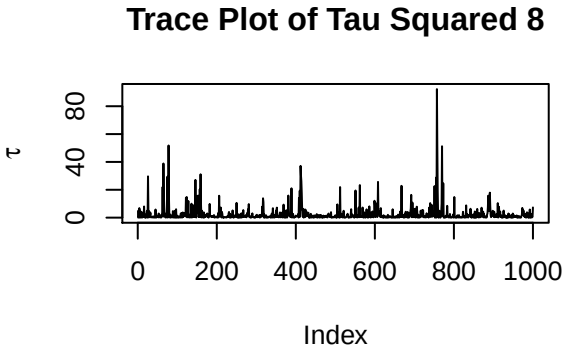




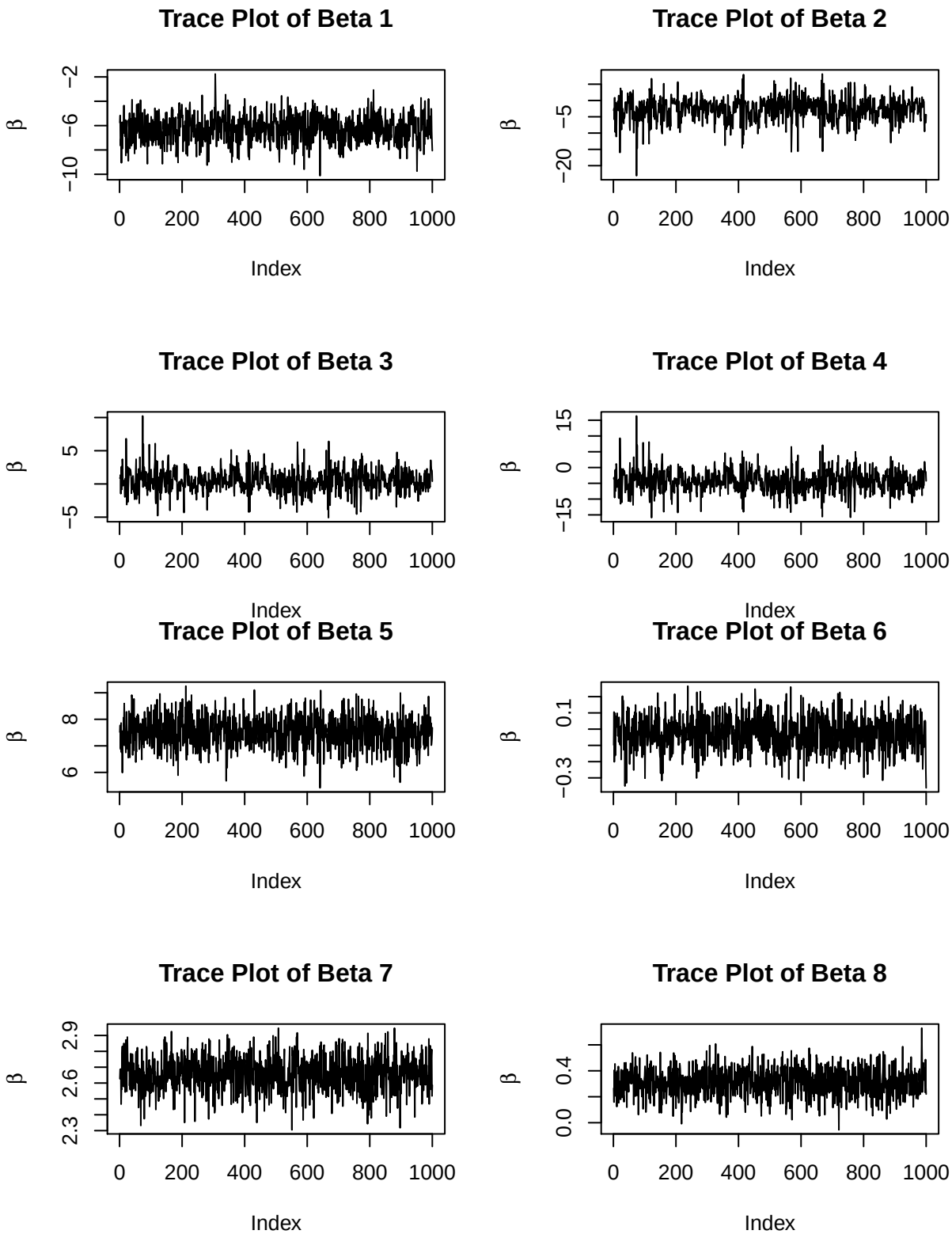




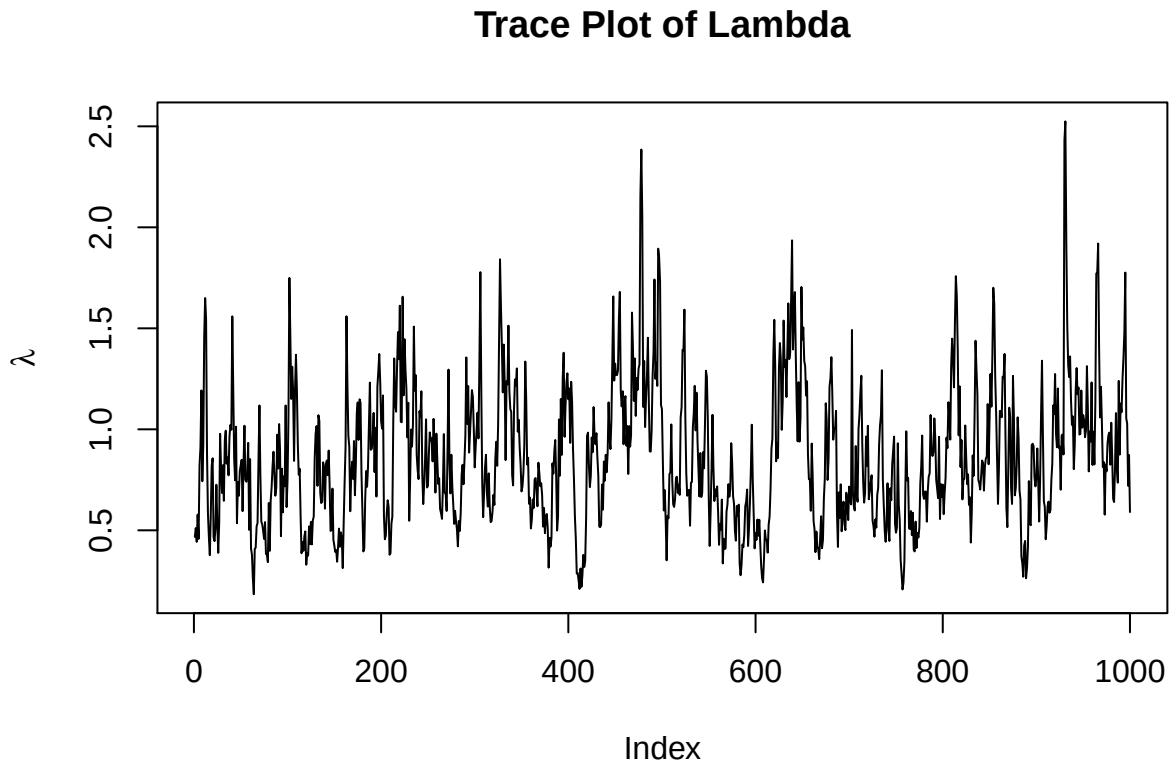





```
for (i in 1:k) {
    plot(post_draws[["Beta"]][, i], type = "l", xlab = "Index", ylab = expression(beta),
        main = paste("Trace Plot of Beta", i))
}
```



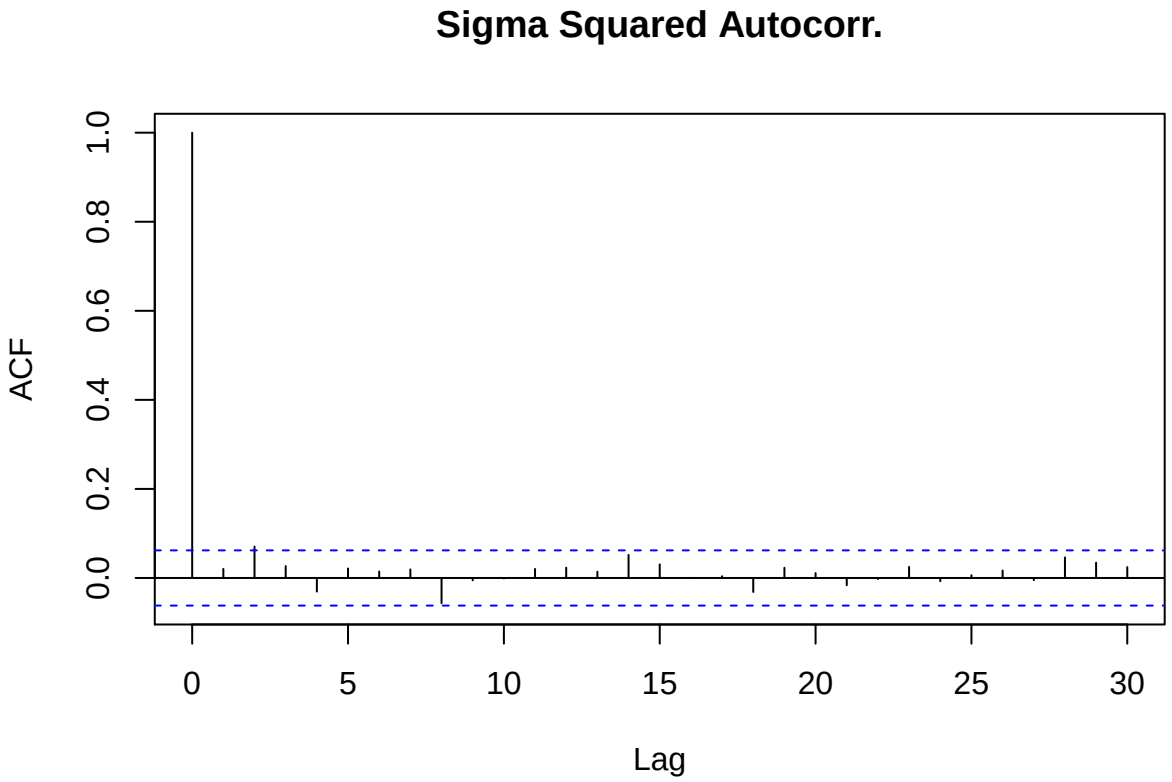

```
par(mfrow = c(1, 1))
plot(post_draws[["Lambda"]], type = "l", xlab = "Index", ylab = expression(lambda),
    main = "Trace Plot of Lambda")
```



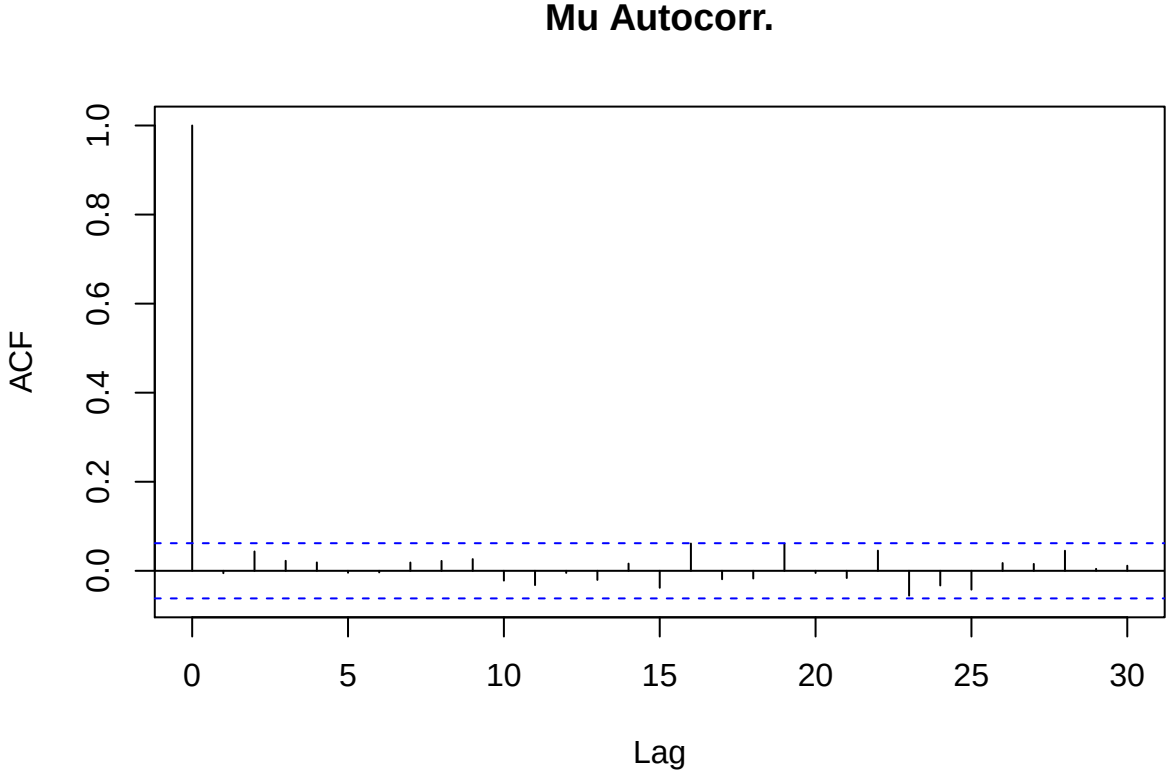
Convergence appears to be very fast. We will burn the first 50 iterations just to be sure.

Effective Sample Size: Autocorrelation & Thinning

```
acf(post_draws[["Sigma Squared"]],main="Sigma Squared Autocorr.")
```

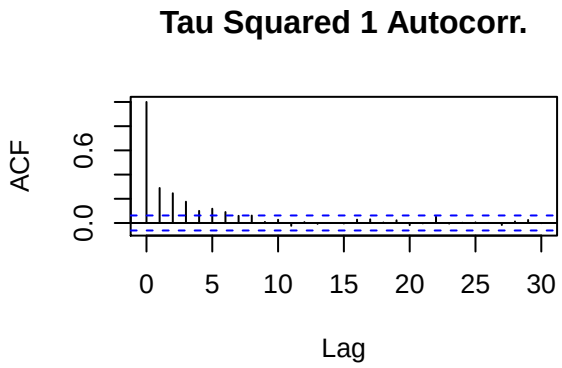


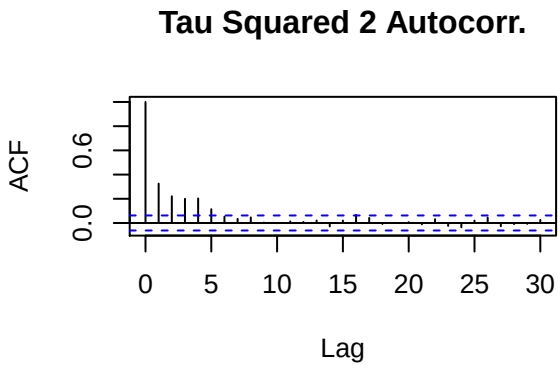

```
acf(post_draws[["Mu"]],main="Mu Autocorr.")
```

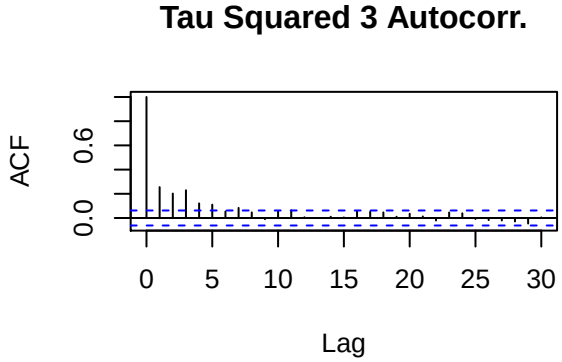


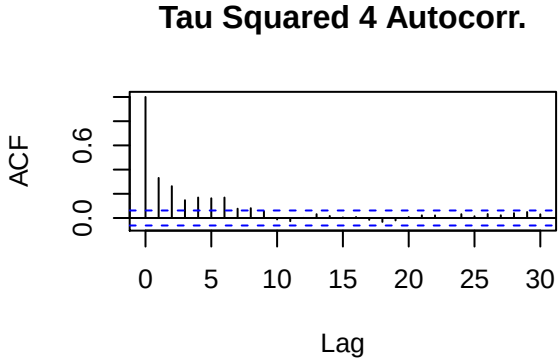

```
par(mfrow=c(2,2))
for(i in 1:k){
  acf(post_draws[["Tau Squared"]][,i],main=paste("Tau Squared",i,"Autocorr."))
```

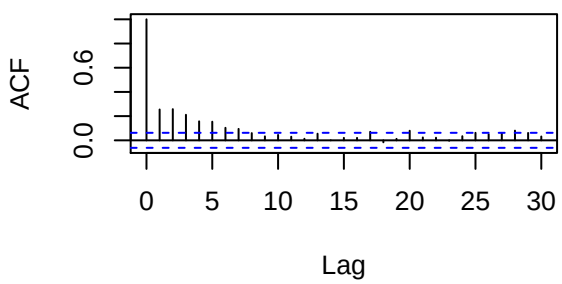
```
}
```

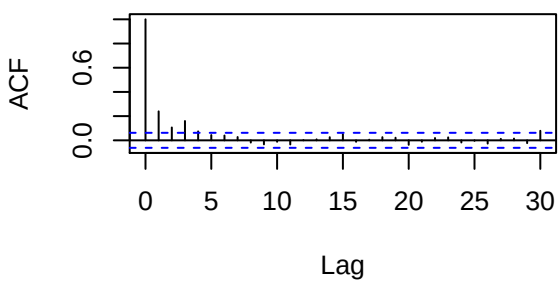


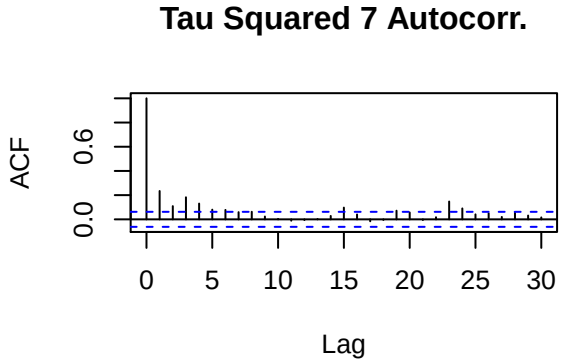


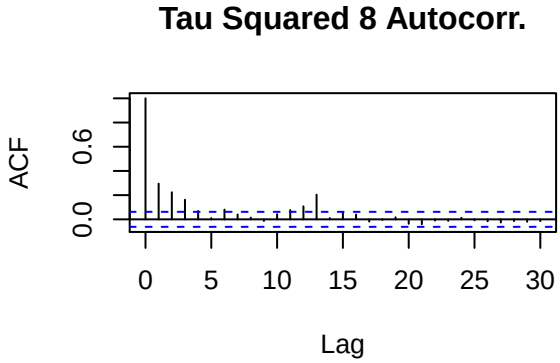






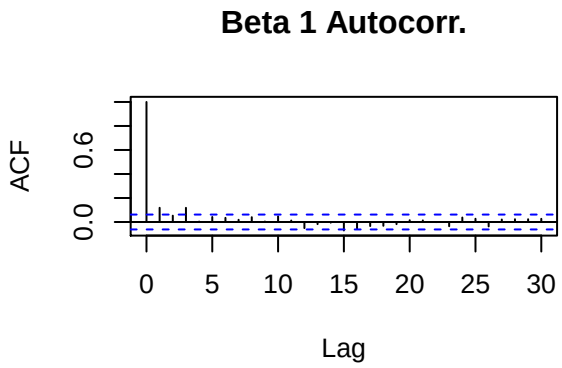


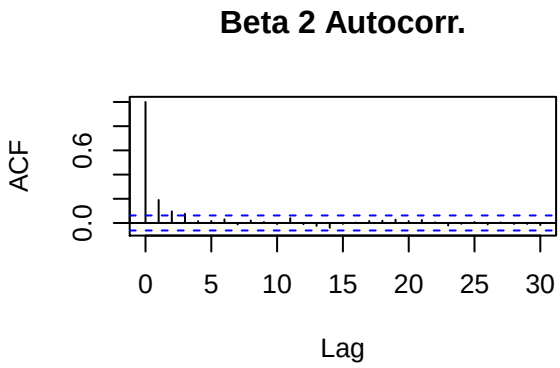


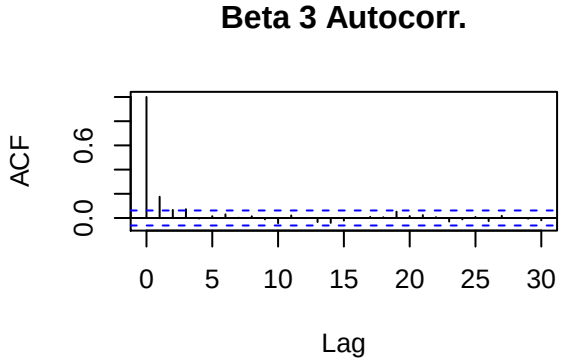


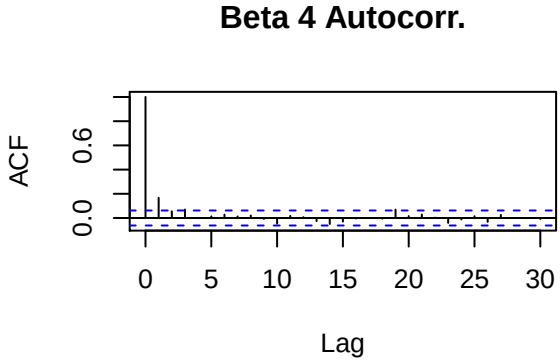

```
for(i in 1:k){
  acf(post_draws[["Beta"]][,i],main=paste("Beta",i,"Autocorr."))
```

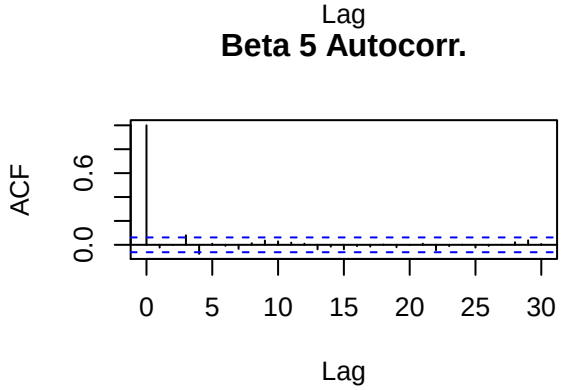
```
}
```

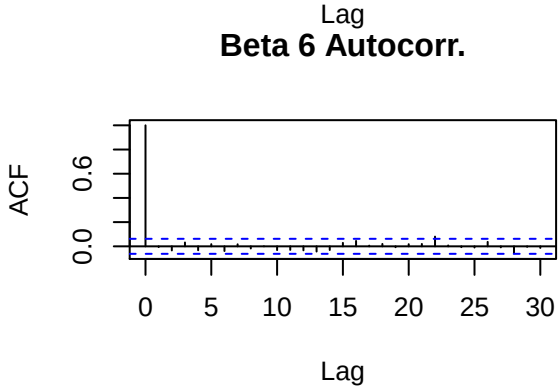


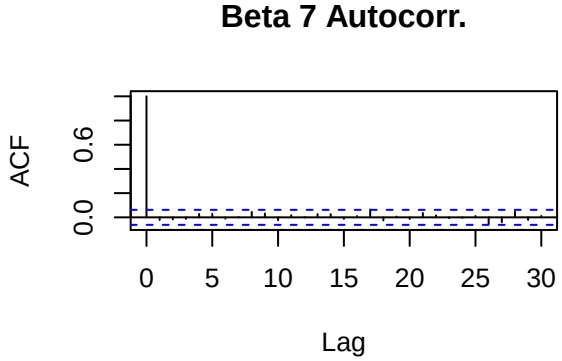


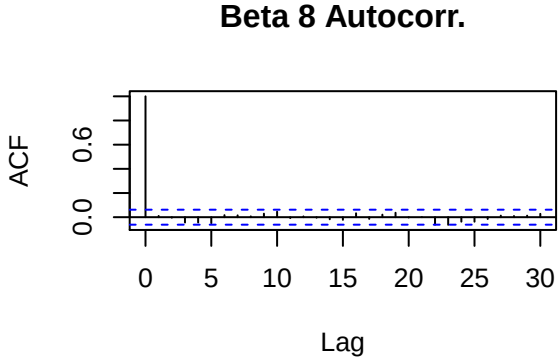




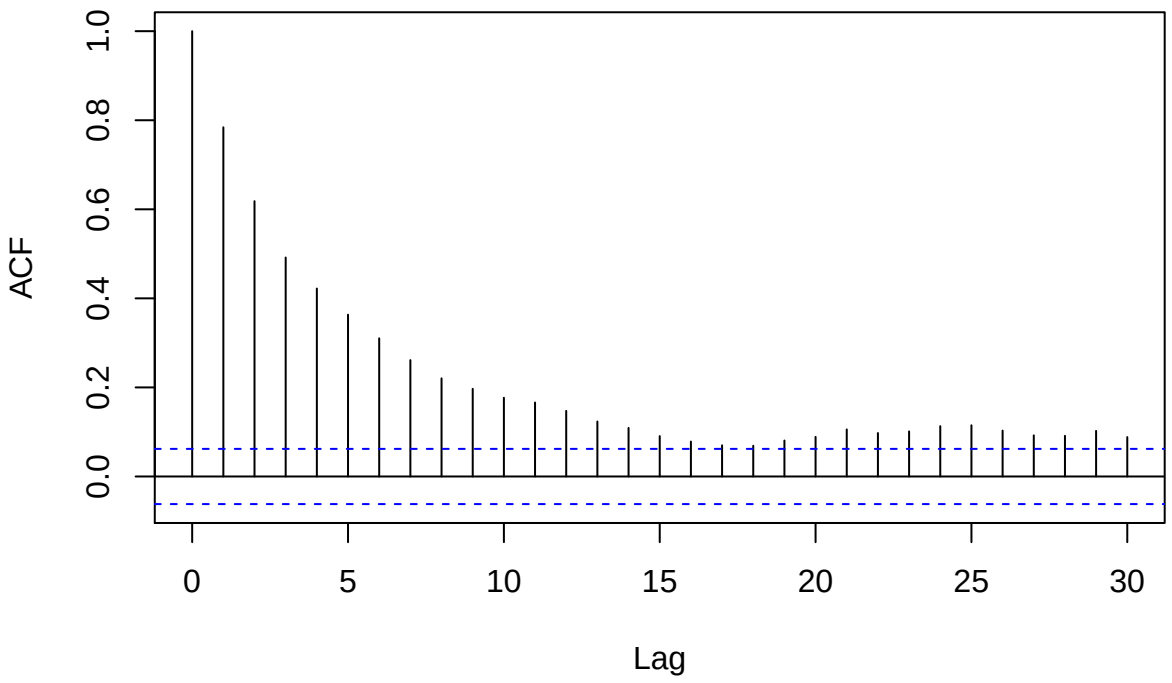









```
par(mfrow=c(1,1))
acf(post_draws[["Lambda"]],main="Lambda Autocorr.")
```



It appears that the max autocorrelation lag for all of the parameters in the model is 5. So to achieve an effective sample size of 2000, we must sample 10000 (not counting burn-in).

```
Y1post_draws = BayesianLasso(Y1,X,10050,start)
Y1post_draws[["Beta"]] = Y1post_draws[["Beta"]][seq(51,10050,5),]
Y1post_draws[["Tau Squared"]] = Y1post_draws[["Tau Squared"]][seq(51,10050,5),]
Y1post_draws[["Sigma Squared"]] = Y1post_draws[["Sigma Squared"]][seq(51,10050,5)]
Y1post_draws[["Lambda"]] = Y1post_draws[["Lambda"]][seq(51,10050,5)]
Y1post_draws[["Mu"]] = Y1post_draws[["Mu"]][seq(51,10050,5)]
```

Interpreting the Model

```
CredInt95 = matrix(NA,19,3,dimnames=list(
  c(paste("Beta",1:8),
    paste("Tau Squared",1:8),
    "Sigma Squared","Lambda","Mu"),
  c("LB","UB","Mode")))
#beta
for(i in 1:8){
  CredInt95[i,c(1,2)] = quantile(Y1post_draws[["Beta"]][,i],c(.025,.975))
  dpost = density(Y1post_draws[["Beta"]][,i])
  j = which.max(dpost$y)
  CredInt95[i,3] = dpost$x[j]
}
#tau squared
for(i in 1:8){
  CredInt95[i+8,c(1,2)] = quantile(Y1post_draws[["Tau Squared"]][,i],c(.025,.975))
  dpost = density(Y1post_draws[["Tau Squared"]][,i])
```

```
  j = which.max(dpost$y)
  CredInt95[i+8,3] = dpost$x[j]
}
#sigma squared
CredInt95[17,c(1,2)] = quantile(Y1post_draws[["Sigma Squared"]],c(.025,.975))
dpost = density(Y1post_draws[["Sigma Squared"]])
j = which.max(dpost$y)
CredInt95[17,3] = dpost$x[j]

CredInt95[18,c(1,2)] = quantile(Y1post_draws[["Lambda"]],c(.025,.975))
dpost = density(Y1post_draws[["Lambda"]])
j = which.max(dpost$y)
CredInt95[18,3] = dpost$x[j]

CredInt95[19,c(1,2)] = quantile(Y1post_draws[["Mu"]],c(.025,.975))
dpost = density(Y1post_draws[["Mu"]])
j = which.max(dpost$y)
CredInt95[19,3] = dpost$x[j]

round(CredInt95,2)
```

```
##                    LB    UB  Mode
## Beta 1          -8.34 -4.10 -6.20
## Beta 2         -10.22  2.87 -0.98
## Beta 3          -2.51  3.88 -0.01
## Beta 4         -10.79  2.75 -4.68
## Beta 5           6.34  8.66  7.52
## Beta 6          -0.23  0.19 -0.03
## Beta 7           2.44  2.86  2.66
## Beta 8           0.12  0.54  0.29
## Tau Squared 1    0.64 20.96  1.84
## Tau Squared 2    0.08 17.97  0.82
## Tau Squared 3    0.03 15.02  0.48
## Tau Squared 4    0.10 17.60  1.17
## Tau Squared 5    0.99 24.26  2.21
## Tau Squared 6    0.00 13.22  0.32
## Tau Squared 7    0.23 14.60  0.85
## Tau Squared 8    0.01 13.42  0.22
## Sigma Squared    7.78  9.64  8.63
## Lambda           0.36  1.67  0.74
## Mu              22.10 22.52 22.30
```

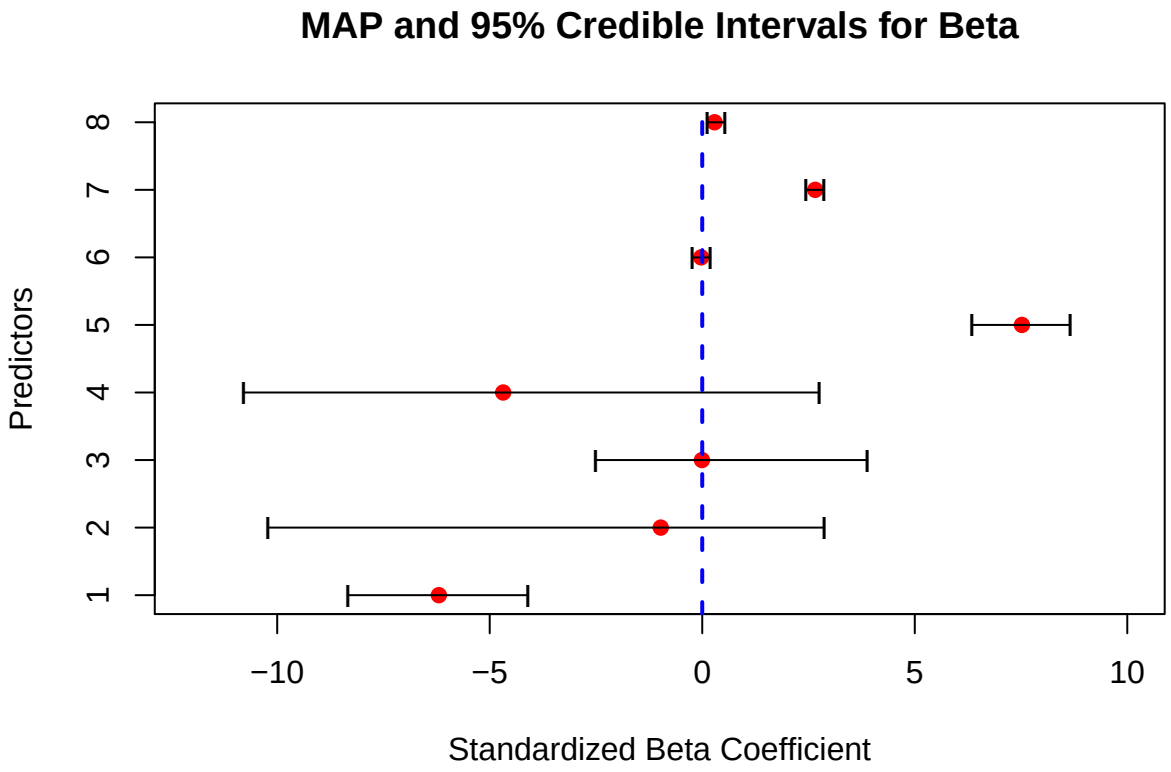
Looking at our 95% credible intervals, we see that four β's don't cross zero (and so have not been selected out by our Bayesian Lasso):

1. β_1: Relative Compactness
2. β_5: Overall Height
3. β_7: Glazing Area
4. β_8: Glazing Area Distribution

Our model has found that surface area, wall area, roof area, and orientation are unnecessary for predicting heating load requirements of buildings.

```
plot(CredInt95[1:8,"Mode"],1:8,xlab="Standardized Beta Coefficient",ylab="Predictors",
     pch=19,col="red",xlim=c(-12,10),main="MAP and 95% Credible Intervals for Beta")
for(i in 1:8){
```

```
  x = CredInt95[i,c(1,2)]
  y = c(i,i)
  lines(x,y)
  points(CredInt95[i,c(1,2)],c(i,i),pch="I")
}
abline(v=0,lty=2,lwd=2,col="blue")
```



Bayesian Lasso of Cooling Load

After a similar look at burn-in and thinning with the Cooling Load outcome, I found that a similar burn-in of 50 and a thinning of every 10th draw (determined by λ) was needed.

```
Y2post_draws = BayesianLasso(Y2,X,20050,start)

Y2post_draws[["Beta"]] = Y2post_draws[["Beta"]][seq(51,20050,10),]
Y2post_draws[["Tau Squared"]] = Y2post_draws[["Tau Squared"]][seq(51,20050,10),]
Y2post_draws[["Sigma Squared"]] = Y2post_draws[["Sigma Squared"]][seq(51,20050,10)]
Y2post_draws[["Lambda"]] = Y2post_draws[["Lambda"]][seq(51,20050,10)]
Y2post_draws[["Mu"]] = Y2post_draws[["Mu"]][seq(51,20050,10)]
```

Interpreting the Model

```
CredInt95Y2 = matrix(NA,19,3,dimnames=list(
  c(paste("Beta",1:8),
    paste("Tau Squared",1:8),
    "Sigma Squared","Lambda","Mu"),
  c("LB","UB","Mode")))
#beta
```

```
for(i in 1:8){
  CredInt95Y2[i,c(1,2)] = quantile(Y2post_draws[["Beta"]][,i],c(.025,.975))
  dpost = density(Y2post_draws[["Beta"]][,i])
  j = which.max(dpost$y)
  CredInt95Y2[i,3] = dpost$x[j]
}
#tau squared
for(i in 1:8){
  CredInt95Y2[i+8,c(1,2)] = quantile(Y2post_draws[["Tau Squared"]][,i],c(.025,.975))
  dpost = density(Y2post_draws[["Tau Squared"]][,i])
  j = which.max(dpost$y)
  CredInt95Y2[i+8,3] = dpost$x[j]
}
#sigma squared
CredInt95Y2[17,c(1,2)] = quantile(Y2post_draws[["Sigma Squared"]],c(.025,.975))
dpost = density(Y2post_draws[["Sigma Squared"]])
j = which.max(dpost$y)
CredInt95Y2[17,3] = dpost$x[j]

CredInt95Y2[18,c(1,2)] = quantile(Y2post_draws[["Lambda"]],c(.025,.975))
dpost = density(Y2post_draws[["Lambda"]])
j = which.max(dpost$y)
CredInt95Y2[18,3] = dpost$x[j]

CredInt95Y2[19,c(1,2)] = quantile(Y2post_draws[["Mu"]],c(.025,.975))
dpost = density(Y2post_draws[["Mu"]])
j = which.max(dpost$y)
CredInt95Y2[19,3] = dpost$x[j]

round(CredInt95Y2,2)
```

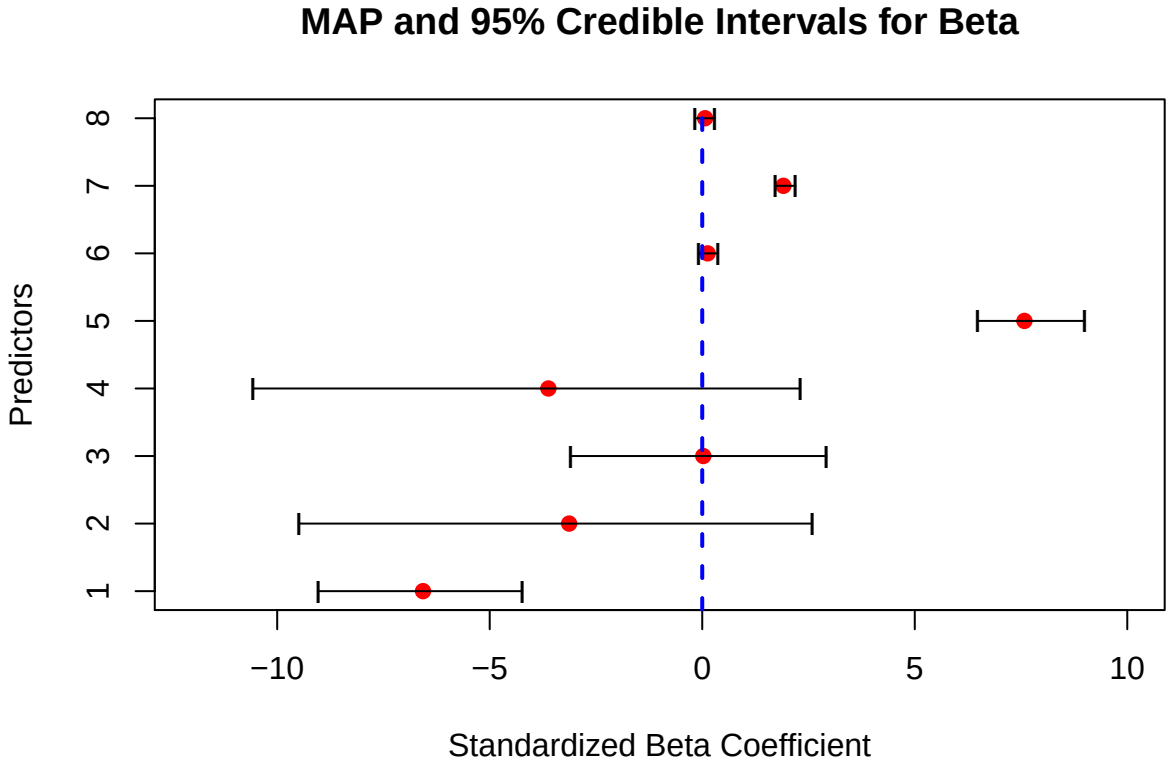
```
##                    LB    UB  Mode
## Beta 1          -9.03 -4.24 -6.57
## Beta 2          -9.49  2.59 -3.13
## Beta 3          -3.10  2.92  0.02
## Beta 4         -10.57  2.31 -3.62
## Beta 5           6.48  9.00  7.58
## Beta 6          -0.09  0.37  0.13
## Beta 7           1.72  2.19  1.91
## Beta 8          -0.17  0.29  0.07
## Tau Squared 1    0.56 14.22  1.61
## Tau Squared 2    0.07 14.81  0.59
## Tau Squared 3    0.02 11.17  0.34
## Tau Squared 4    0.08 15.07  0.77
## Tau Squared 5    0.80 16.01  1.90
## Tau Squared 6    0.00 10.18  0.11
## Tau Squared 7    0.12 11.61  0.60
## Tau Squared 8    0.00  8.95  0.18
## Sigma Squared    9.28 11.35 10.18
## Lambda           0.42  1.95  0.86
## Mu              24.35 24.81 24.59
```

Looking at our 95% credible intervals, we see that three β's don't cross zero (and so have not been selected out by our Bayesian Lasso):

1. β_1: Relative Compactness
2. β_5: Overall Height
3. β_7: Glazing Area

Our model has found that surface area, wall area, roof area, orientation, and glazing area distribution are unnecessary for predicting cooling load requirements of buildings.

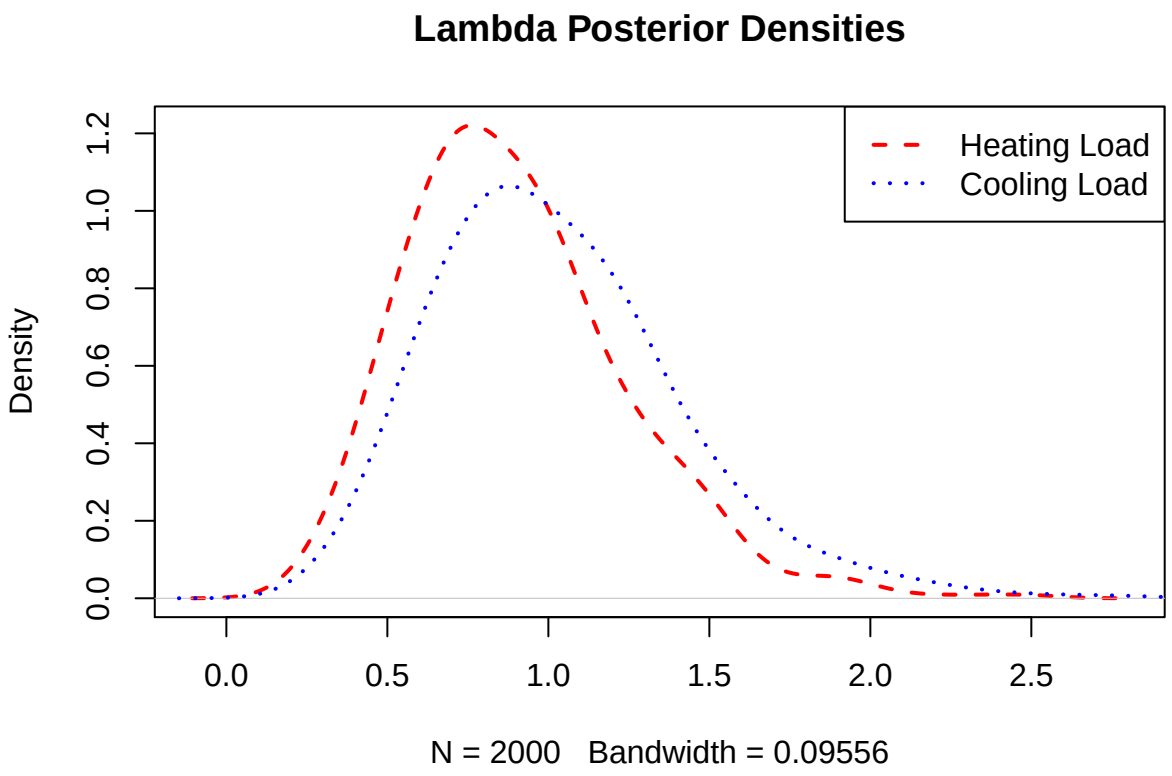
```
plot(CredInt95Y2[1:8,"Mode"],1:8,xlab="Standardized Beta Coefficient",ylab="Predictors",
     pch=19,col="red",xlim=c(-12,10),main="MAP and 95% Credible Intervals for Beta")
for(i in 1:8){
  x = CredInt95Y2[i,c(1,2)]
  y = c(i,i)
  lines(x,y)
  points(CredInt95Y2[i,c(1,2)],c(i,i),pch="I")
}
abline(v=0,lty=2,lwd=2,col="blue")
```



Comparing Lasso's

Results are extremely similar. The key difference is that the model selection performed by the Bayesian lasso selected out Glazing Area Distribution for Cooling Load but not for Heating Load. This is because the posterior distribution for lambda of the Cooling Load is shifted right (larger values of lambda are more probable) of that for the Heating Load - the higher value of lambda imposes more penalty and has the effect of growing the credible interval for the Glazing Area Distribution effect such that it crosses zero.

```
plot(density(Y1post_draws[["Lambda"]],adjust=1.5),col="red",lty=2,lwd=2,main="Lambda Posterior Densities
lines(density(Y2post_draws[["Lambda"]],adjust=1.5),col="blue",lty=3,lwd=2)
legend("topright",c("Heating Load","Cooling Load"),col=c("red","blue"),lty=c(2,3),lwd=2)
```


Lambda Posterior Densities
Heating Load
Cooling Load
Density
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0.0
0.5
1.0
1.5
2.0
2.5
N = 2000 Bandwidth = 0.09556