Bayesian Lasso - Predicting Ice Cream Consumption

Greg Johnson

Consider icecream consumption predicted from price, income, temperature, and year. We will fit a Bayesian lasso so that we may drop any unnecessary explanatory variables. The model takes the form:

$$\mathbf{y} \sim \mathrm{N}(\mu \mathbf{1}_n + X\boldsymbol{\beta}, \sigma^2 I_n)$$
$$\boldsymbol{\beta} \overset{iid}{\sim} \mathrm{Laplace}(0, \lambda/\sigma^2)$$

The Laplace distribution is a pain to work with analytically so we choose an alternative, hierarchical representation that takes advantage of the fact that the Laplace distribution is scale-mixture of normals with an exponential density:

$$\boldsymbol{\beta}|\Sigma_0 \sim \mathrm{N}(\mathbf{0}, \sigma^2\Sigma_0)$$
$$\boldsymbol{\tau}^2|\lambda \sim \prod_{j=1}^{k} \mathrm{Exp}(\lambda^2/2)$$
$$\lambda^2 \sim \Gamma(0.01, 0.01)$$
$$p(\mu) \propto 1$$
$$\sigma^2 = 0.03^2$$

```
dat = read.table("data/icecream.txt", header = TRUE)
Y = dat[, 1]
X = scale(dat[, 2:5])  #standardize predictors!!
```

Posteriors

We will take a two-step approach:

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

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

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

1. Sample $\boldsymbol{\beta}$ from:
$$\boldsymbol{\beta}|\boldsymbol{\tau}^2, \lambda, \sigma^2, \mathbf{y} \sim \mathrm{N}(A^{-1}X^T\tilde{\boldsymbol{y}}, \sigma^2 A^{-1})$$
$$A = X^TX + \Sigma_0^{-1}$$

2. Sample $\boldsymbol{\tau}^2$ by independently sampling:
$$\frac{1}{\tau_j^2}|\boldsymbol{\beta}, \lambda, \sigma^2, \mathbf{y} \sim \mathrm{Inv.N}\left(\frac{\lambda(\sigma^2)^{1/2}}{\beta_j}, \lambda^2\right)$$

3. Sample σ^2 from:

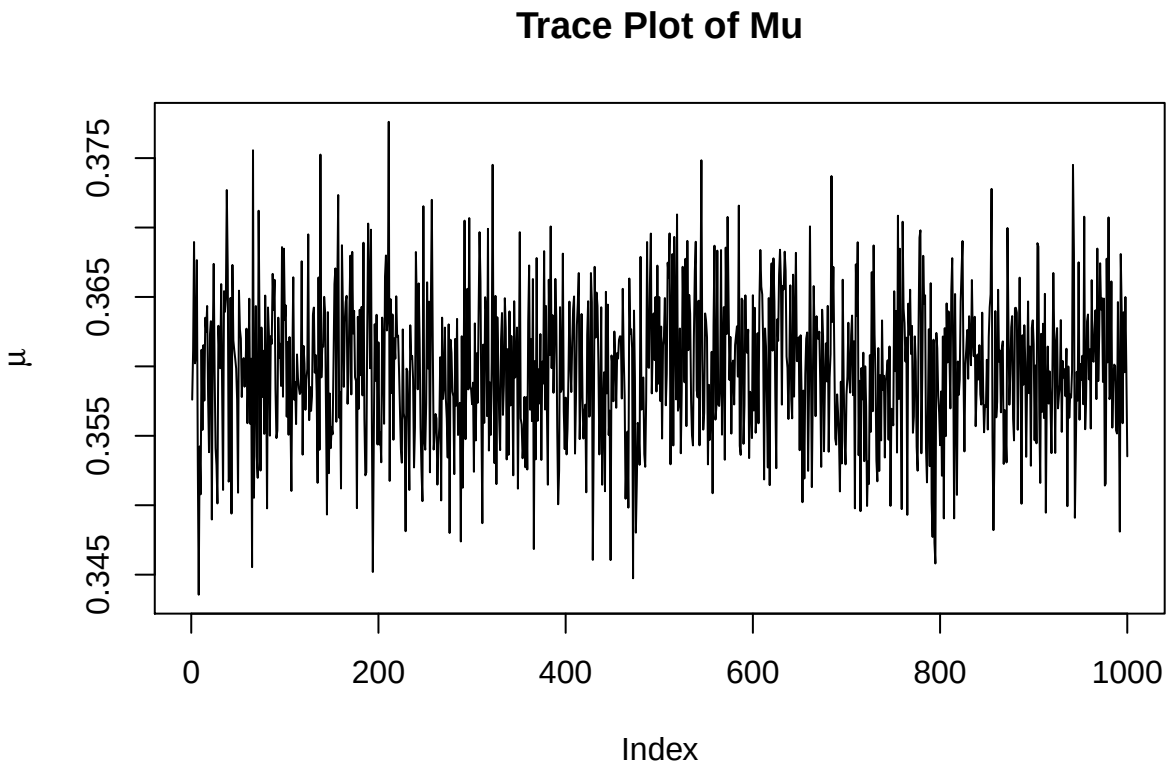
$$\lambda|\boldsymbol{\tau}^2, \boldsymbol{\beta}, \sigma^2, \mathbf{y} \sim \Gamma(k + 0.01, \frac{1}{2}\sum_{j=1}^{k} \tau_j^2 + 0.01)$$

Implement MCMC

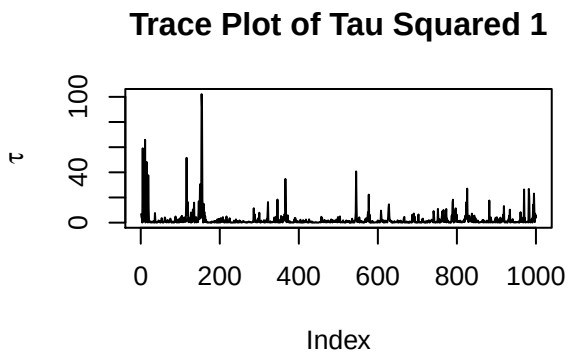
```
BayesianLasso = function(Y, X, sig_sq, ndraws, start) {
    Ytilde = Y - mean(Y)
    n = nrow(X)
    k = ncol(X)

    # set-up list to collect draws
    post_draws = list(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"]]
    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 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[["Lambda"]][g] = lambda
        post_draws[["Mu"]][g] = mu
    }
    return(post_draws)
}

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

post_draws = BayesianLasso(Y, X, 0.03^2, 1000, start)
```

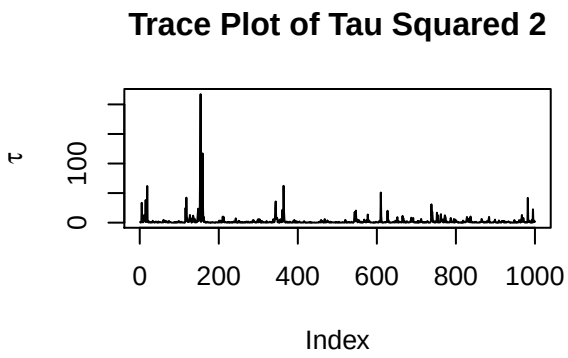
Convergence: Burn-in & Trace Plots

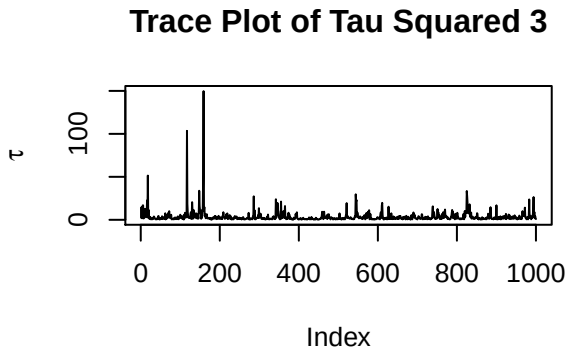
```r
k = ncol(X)

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

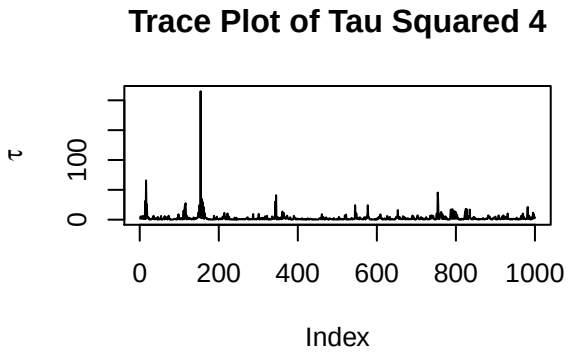



```r
par(mfrow = c(2, 2))
for (i in 1:k) {
    plot(post_draws[["Tau Squared"]][, i], type = "l", xlab = "Index",
        ylab = expression(tau), main = paste("Trace Plot of Tau Squared",
            i))
}
```

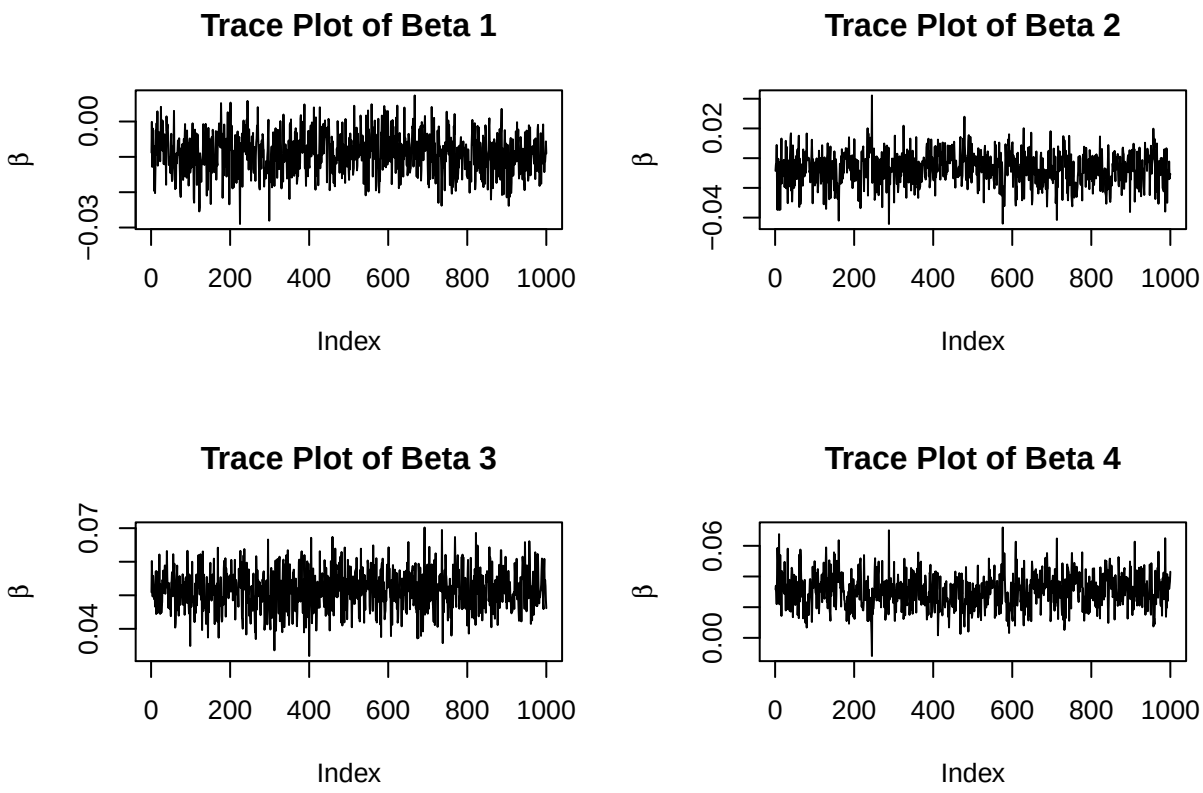




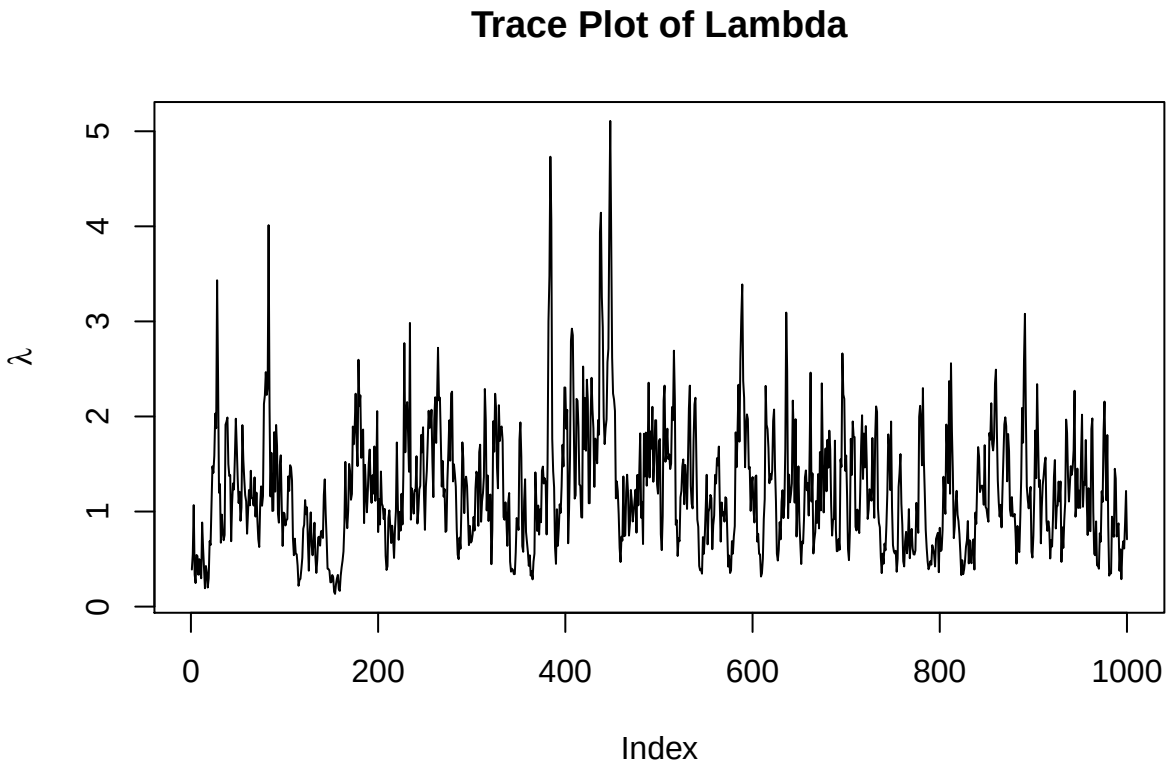





```
for (i in 1:k) {
    plot(post_draws[["Beta"]][, i], type = "l", xlab = "Index",
        ylab = expression(beta), main = paste("Trace Plot of Beta",
            i))
}
```



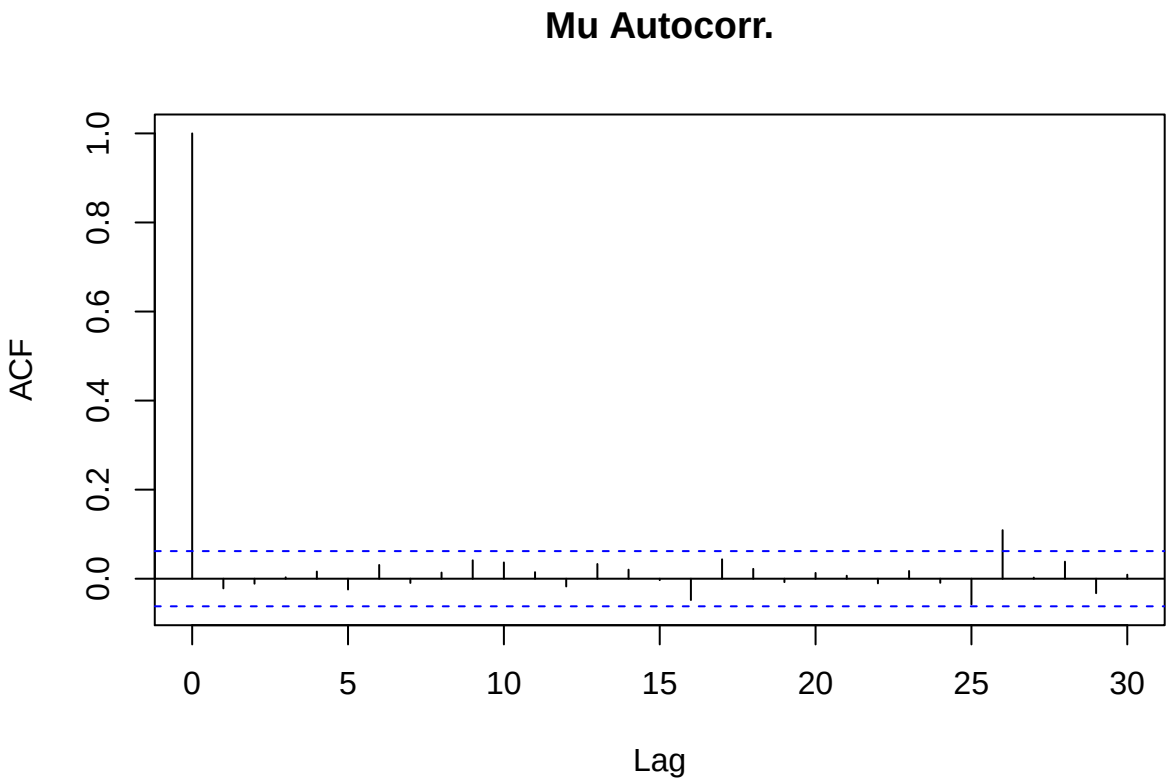

```
par(mfrow = c(1, 1))
plot(post_draws[["Lambda"]], type = "l", xlab = "Index", ylab = expression(lambda),
    main = "Trace Plot of Lambda")
```



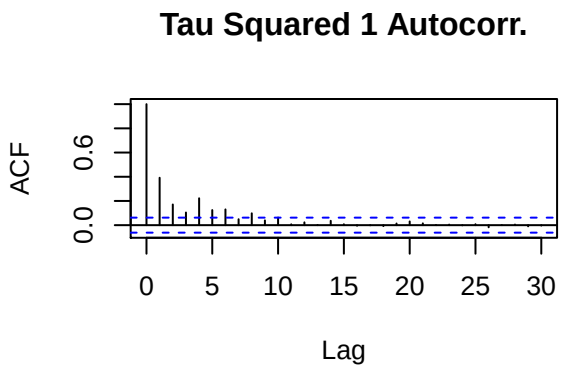
Convergence seems quick - we will burn in 50 just to air on the conservative side.

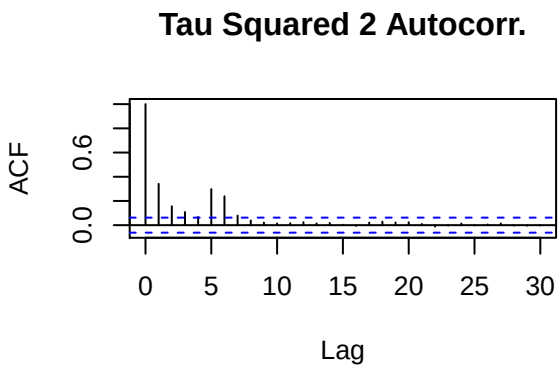
Effective Sample Size: Autocorrelation & Thinning

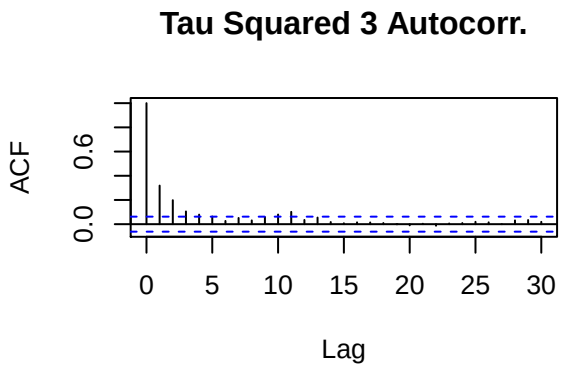
```
acf(post_draws[["Mu"]], main = "Mu Autocorr.")
```

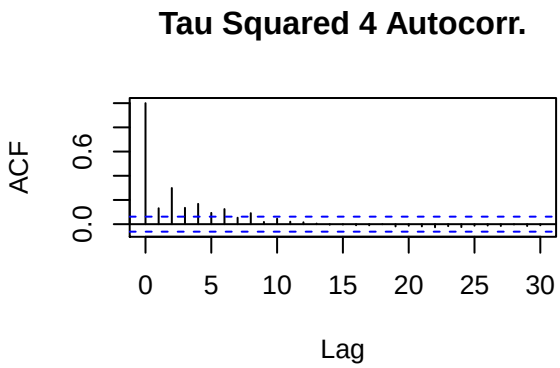



```
par(mfrow = c(2, 2))
for (i in 1:k) {
    acf(post_draws[["Tau Squared"]][, i], main = paste("Tau Squared",
        i, "Autocorr."))
}
```

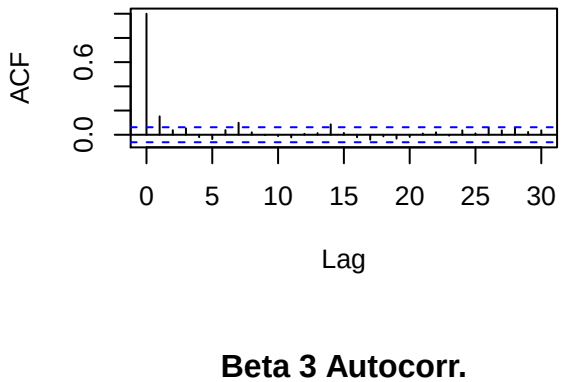


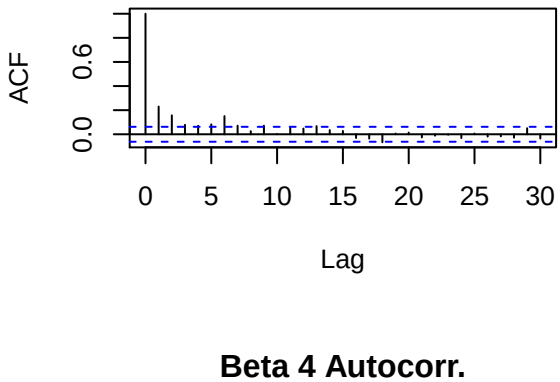


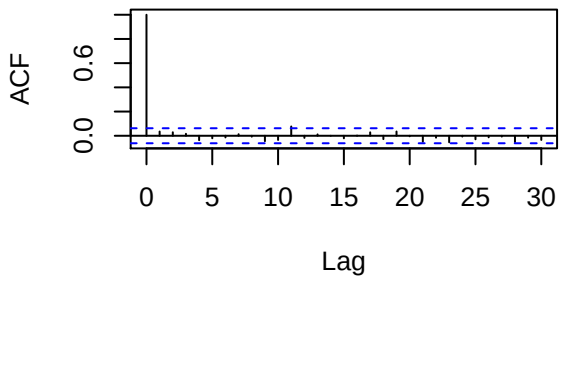


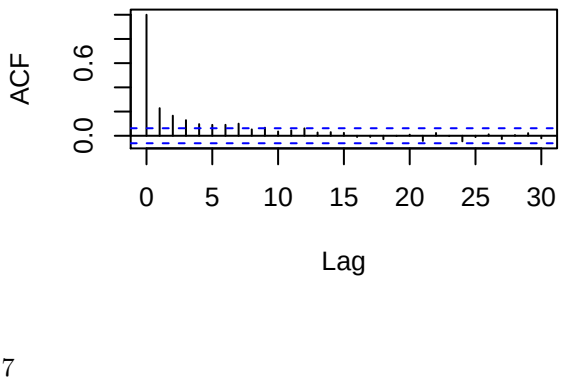



```
for (i in 1:k) {
    acf(post_draws[["Beta"]][, i], main = paste("Beta", i, "Autocorr."))
}
```

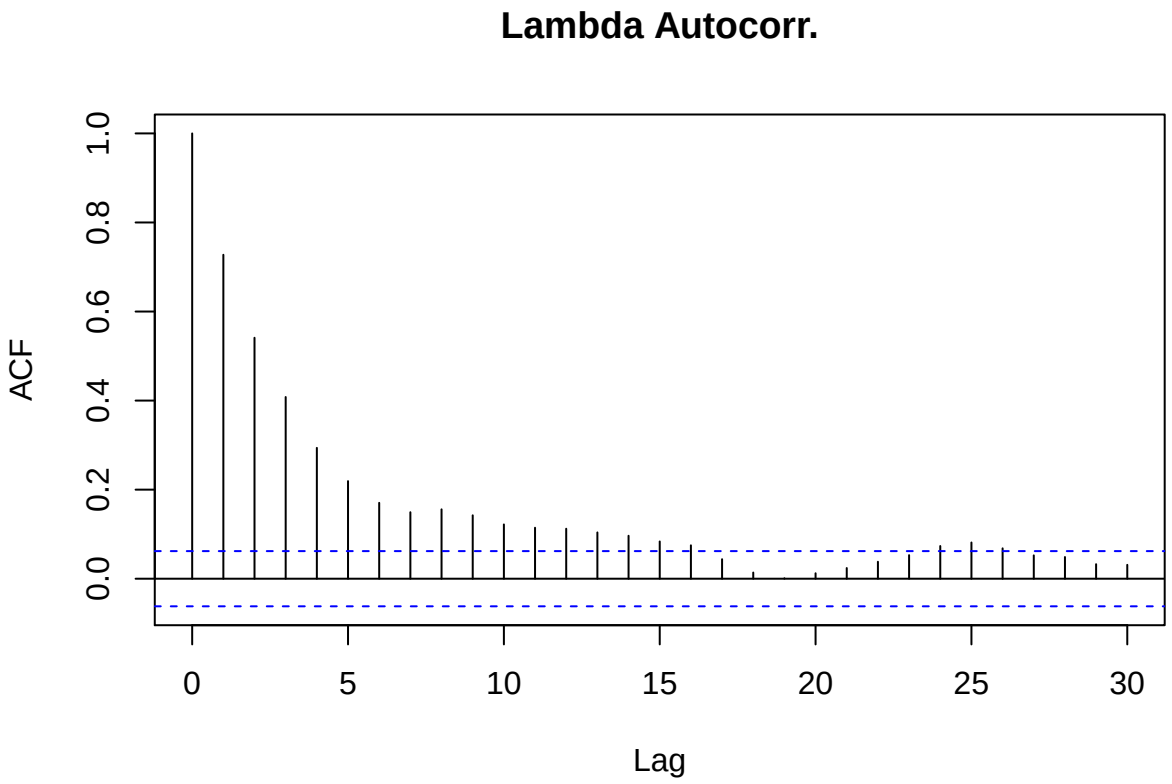








```
par(mfrow = c(1, 1))
acf(post_draws[["Lambda"]], main = "Lambda Autocorr.")
```



It appears that λ has the greatest autocorrelation lag, 8. So we thin out to every 8th draw and burn-in the first 50. To get an effective sample size of 2000, we have to run our chain for 16050 draws.

```
Ypost_draws = BayesianLasso(Y, X, 0.03^2, 16050, start)
Ypost_draws[["Beta"]] = Ypost_draws[["Beta"]][seq(51, 16050,
    8), ]
Ypost_draws[["Tau Squared"]] = Ypost_draws[["Tau Squared"]][seq(51,
    16050, 8), ]
Ypost_draws[["Lambda"]] = Ypost_draws[["Lambda"]][seq(51, 16050,
    8)]
Ypost_draws[["Mu"]] = Ypost_draws[["Mu"]][seq(51, 16050, 8)]
```

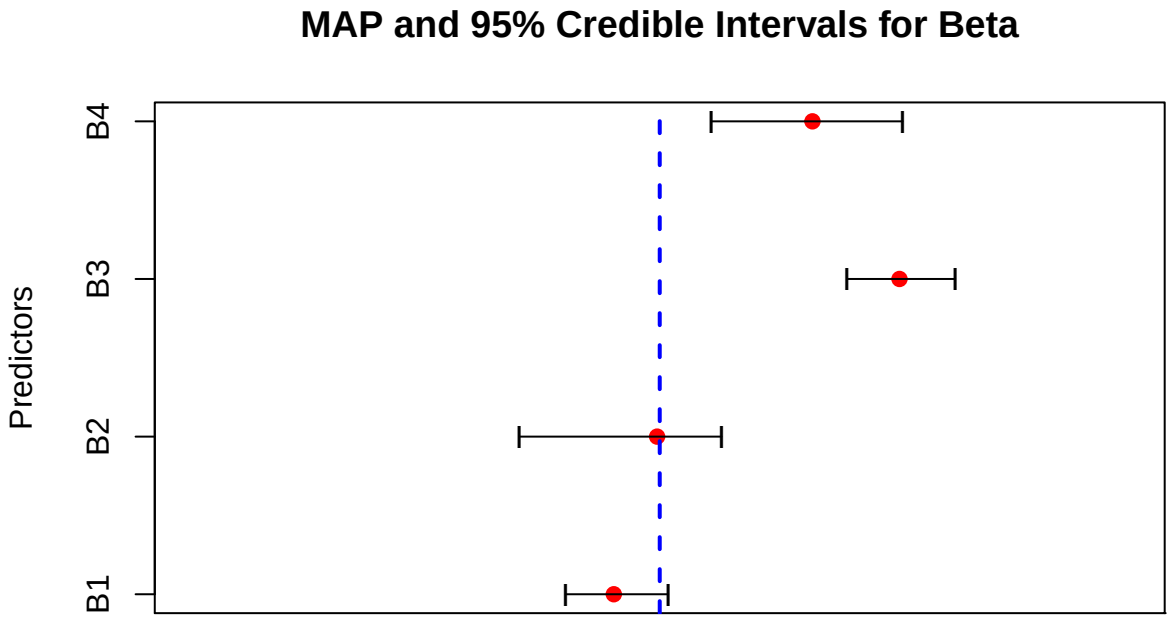
Interpreting the Model

```
CredInt95 = matrix(NA, 10, 3, dimnames = list(c(paste("Beta",
    1:4), paste("Tau Squared", 1:4), "Lambda", "Mu"), c("LB",
    "UB", "Mode")))
# beta
for (i in 1:4) {
    CredInt95[i, c(1, 2)] = quantile(post_draws[["Beta"]][, i],
        c(0.025, 0.975))
    dpost = density(post_draws[["Beta"]][, i])
    j = which.max(dpost$y)
    CredInt95[i, 3] = dpost$x[j]
}
# tau squared
```

```
for (i in 1:4) {
    CredInt95[i + 4, c(1, 2)] = quantile(post_draws[["Tau Squared"]][,
        i], c(0.025, 0.975))
    dpost = density(post_draws[["Tau Squared"]][, i])
    j = which.max(dpost$y)
    CredInt95[i + 4, 3] = dpost$x[j]
}

CredInt95[9, c(1, 2)] = quantile(post_draws[["Lambda"]], c(0.025,
    0.975))
dpost = density(post_draws[["Lambda"]])
j = which.max(dpost$y)
CredInt95[9, 3] = dpost$x[j]

CredInt95[10, c(1, 2)] = quantile(post_draws[["Mu"]], c(0.025,
    0.975))
dpost = density(post_draws[["Mu"]])
j = which.max(dpost$y)
CredInt95[10, 3] = dpost$x[j]

round(CredInt95, 3)
```

```
##                    LB     UB   Mode
## Beta 1         -0.020  0.002 -0.010
## Beta 2         -0.030  0.013 -0.001
## Beta 3          0.040  0.063  0.051
## Beta 4          0.011  0.052  0.033
## Tau Squared 1   0.021 16.394  0.230
## Tau Squared 2   0.017 16.866  0.505
## Tau Squared 3   0.401 16.786  0.994
## Tau Squared 4   0.121 16.975  0.641
## Lambda          0.334  2.664  1.000
## Mu              0.350  0.370  0.358
```

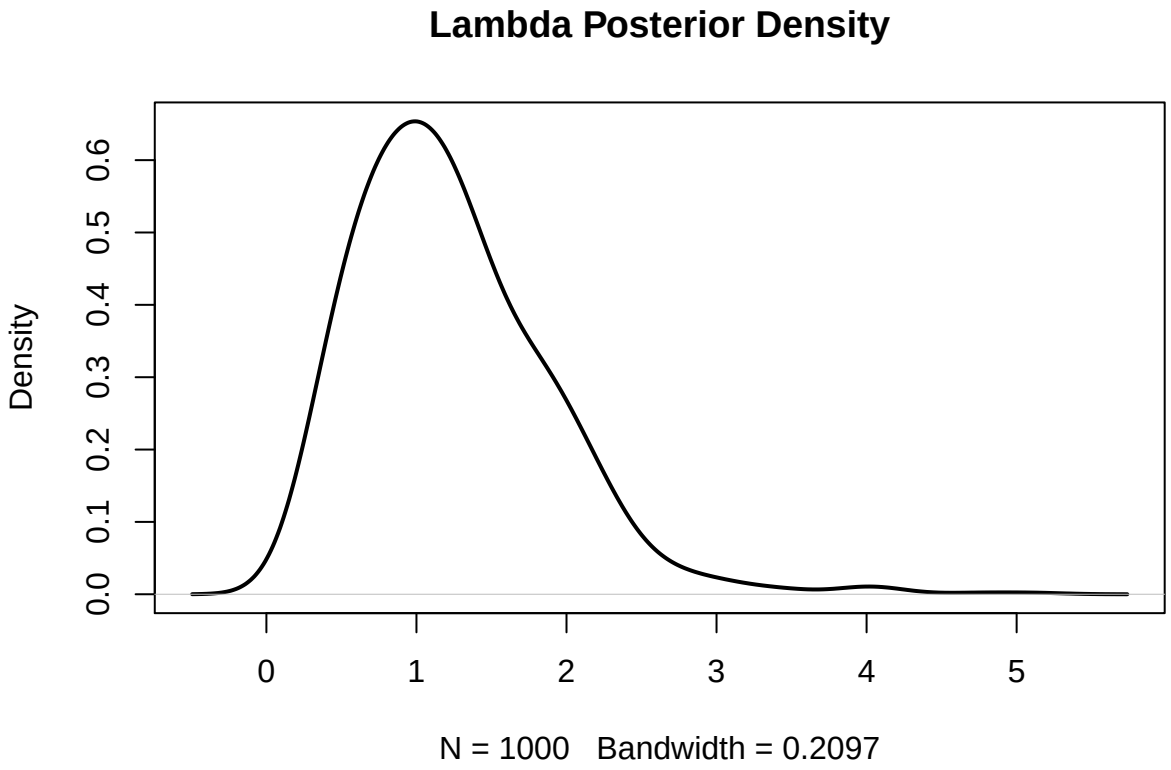
```
plot(CredInt95[1:4, "Mode"], 1:4, xlab = "Standardized Beta Coefficient",
    ylab = "Predictors", pch = 19, col = "red", xlim = c(-0.1,
        0.1), main = "MAP and 95% Credible Intervals for Beta",
    at = c(1, 2, 3, 4), labels = c("B1", "B2", "B3", "B4"))
for (i in 1:4) {
    x = CredInt95[i, 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")
```



Our Bayesian lasso has selected out price and income since the credible intervals for their effects cross 0 - there is a greater than 5% chance that they have no effect! So according to our model, temperature and year have an effect on ice cream consumption. Using MAP estimates, for every 0.82 increase in temperature, Ice cream consumption increases by one. And for every quarter (~.27) of a year, ice cream consumption increases by one.

To get a feel for the L_1 penalizations strength that our model employed we can look at the posterior for λ :

```
plot(density(post_draws[["Lambda"]], adjust = 1.5), lwd = 2,
    main = "Lambda Posterior Density")
```



The MAP estimate for λ indicates that a value of 1 is most likely for regularization.