Bayesian Lasso and SSVC - Predicting Ice Cream Consumption

Greg Johnson

Bayesian Lasso

Say we have the task of predicting ice cream consumption 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:

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

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:

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

3. Sample σ^2 from:

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

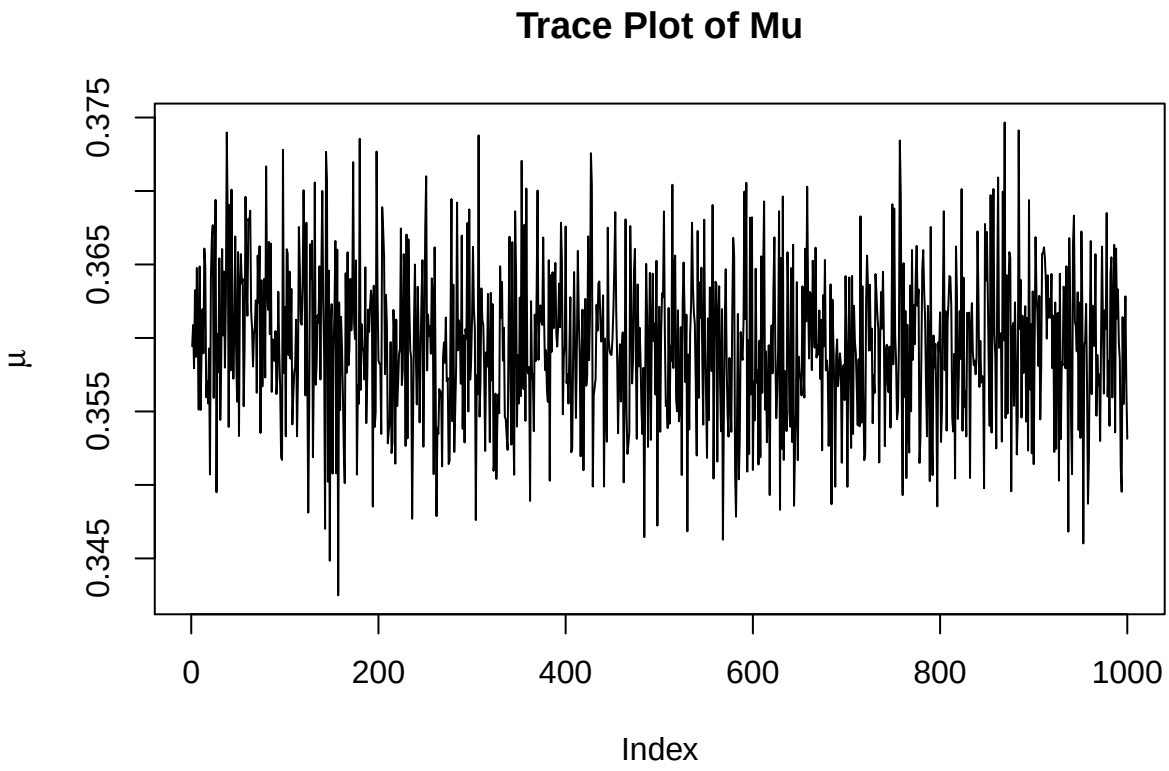
Implement MCMC

```
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)
```

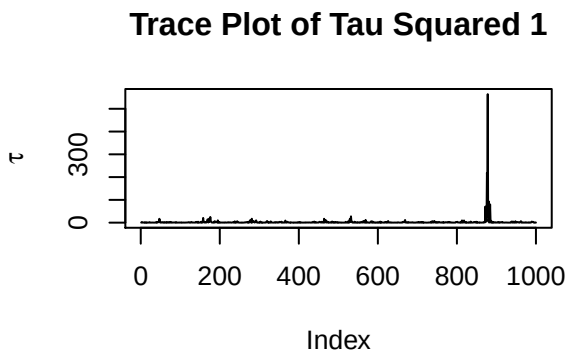
```
post_draws = BayesianLasso(Y, X, 0.03^2, 1000, start)
```

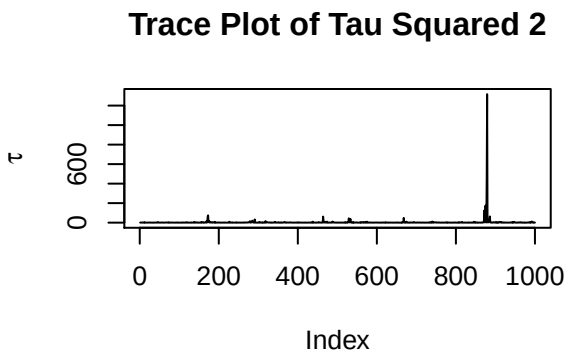
Convergence: Burn-in & Trace Plots

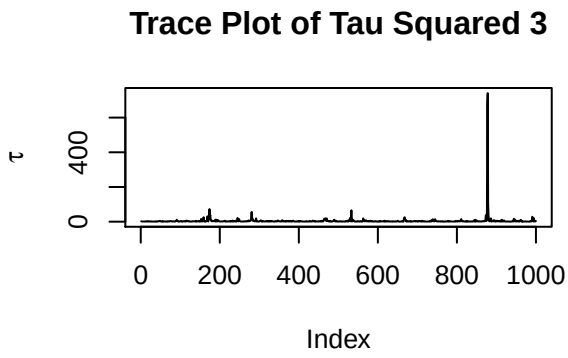
```
k = ncol(X)

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

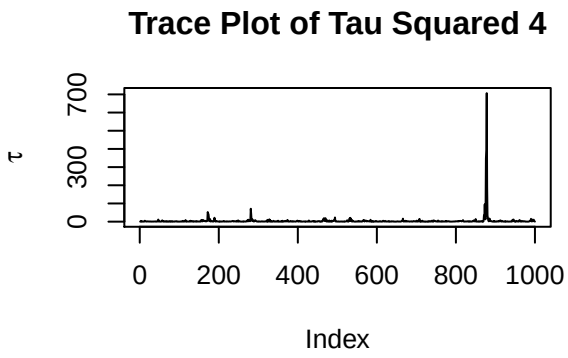



```
par(mfrow = c(2, 2))
for (i in 1:k) {
    plot(post_draws[["Tau Squared"]][, i], type = "l", xlab = "Index",
        ylab = expression(tau), main = paste("Trace Plot of Tau Squared",
            i))
}
```

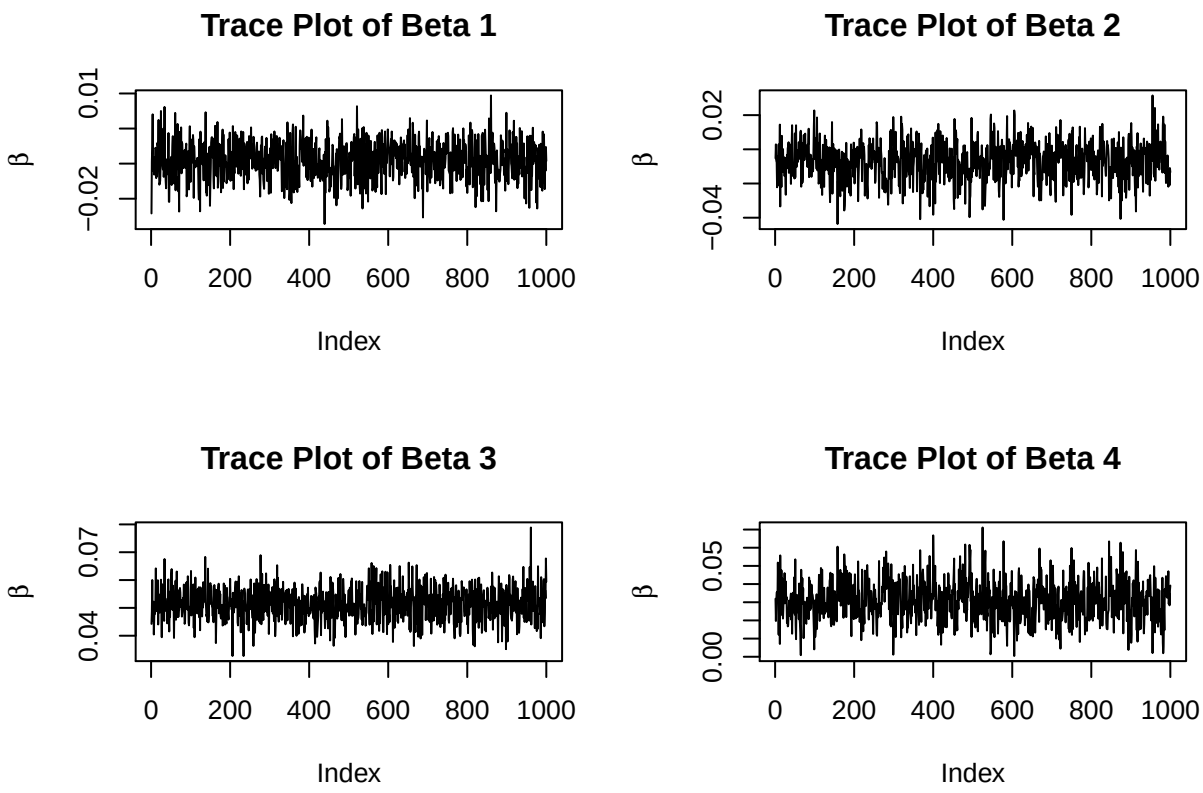




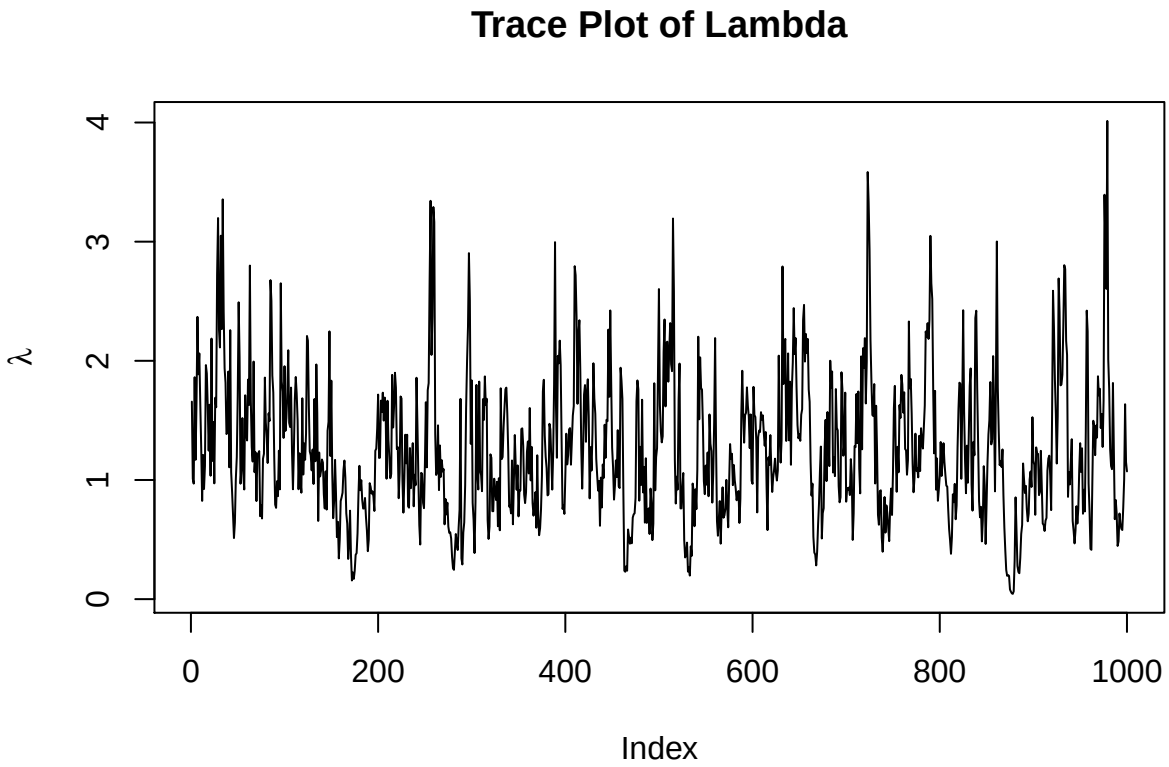





```
for (i in 1:k) {
    plot(post_draws[["Beta"]][, i], type = "l", xlab = "Index",
        ylab = expression(beta), main = paste("Trace Plot of Beta",
            i))
}
```



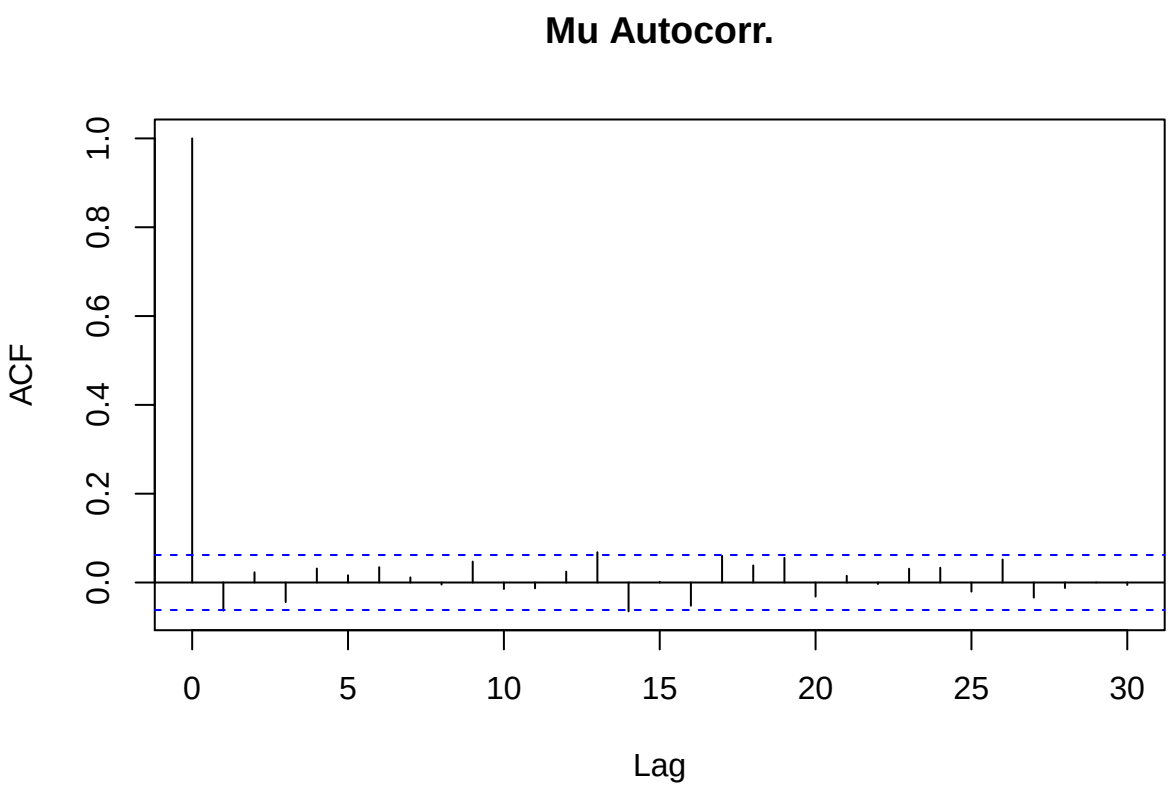

```
par(mfrow = c(1, 1))
plot(post_draws[["Lambda"]], type = "l", xlab = "Index", ylab = expression(lambda),
    main = "Trace Plot of Lambda")
```



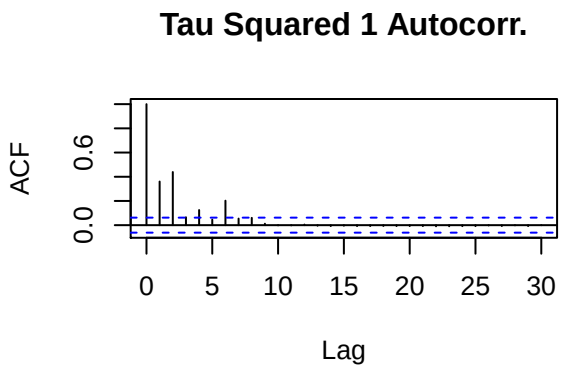
Convergence seems quick - we will burn in 50 just to air on the conservative side.

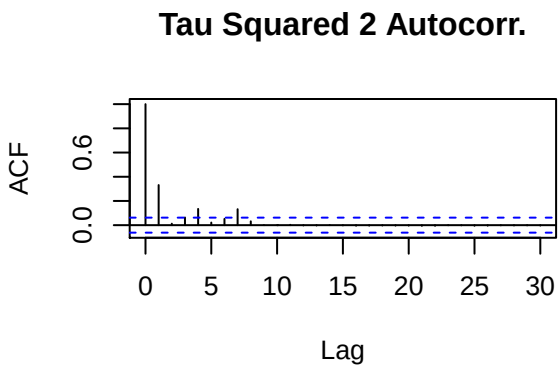
Effective Sample Size: Autocorrelation & Thinning

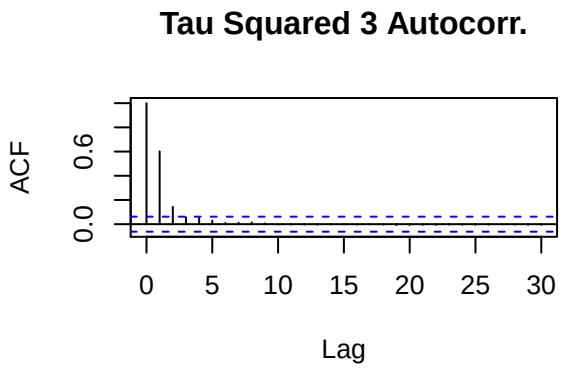
```
acf(post_draws[["Mu"]], main = "Mu Autocorr.")
```

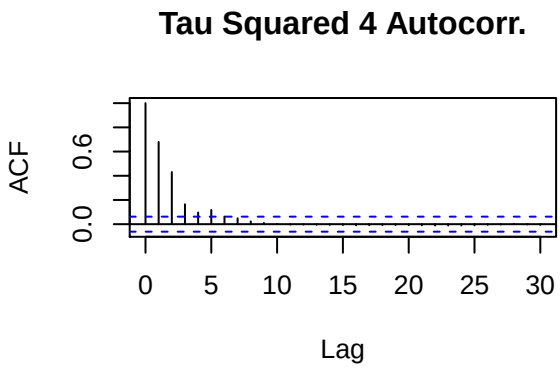



```
par(mfrow = c(2, 2))
for (i in 1:k) {
    acf(post_draws[["Tau Squared"]][, i], main = paste("Tau Squared",
        i, "Autocorr."))
}
```

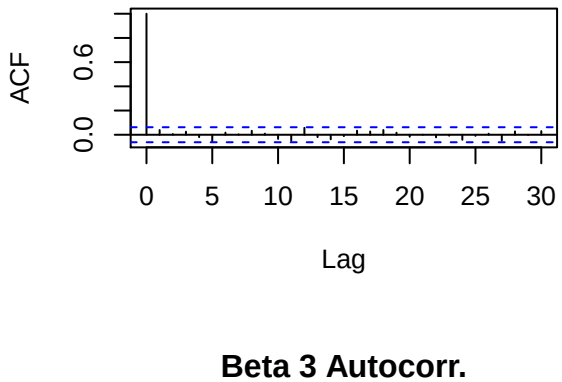


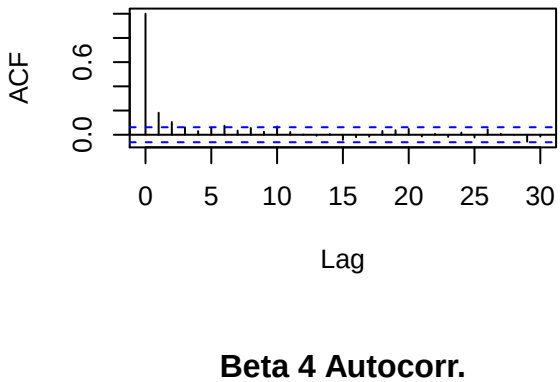


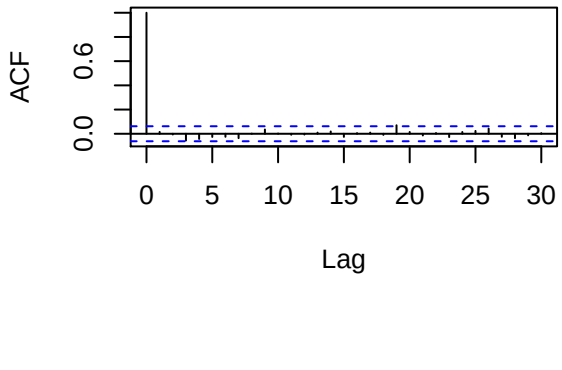


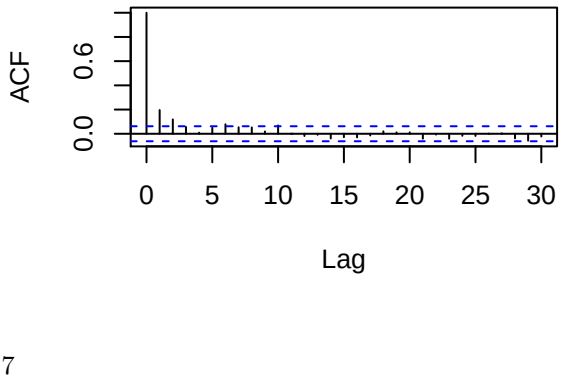



```
for (i in 1:k) {
    acf(post_draws[["Beta"]][, i], main = paste("Beta", i, "Autocorr."))
}
```

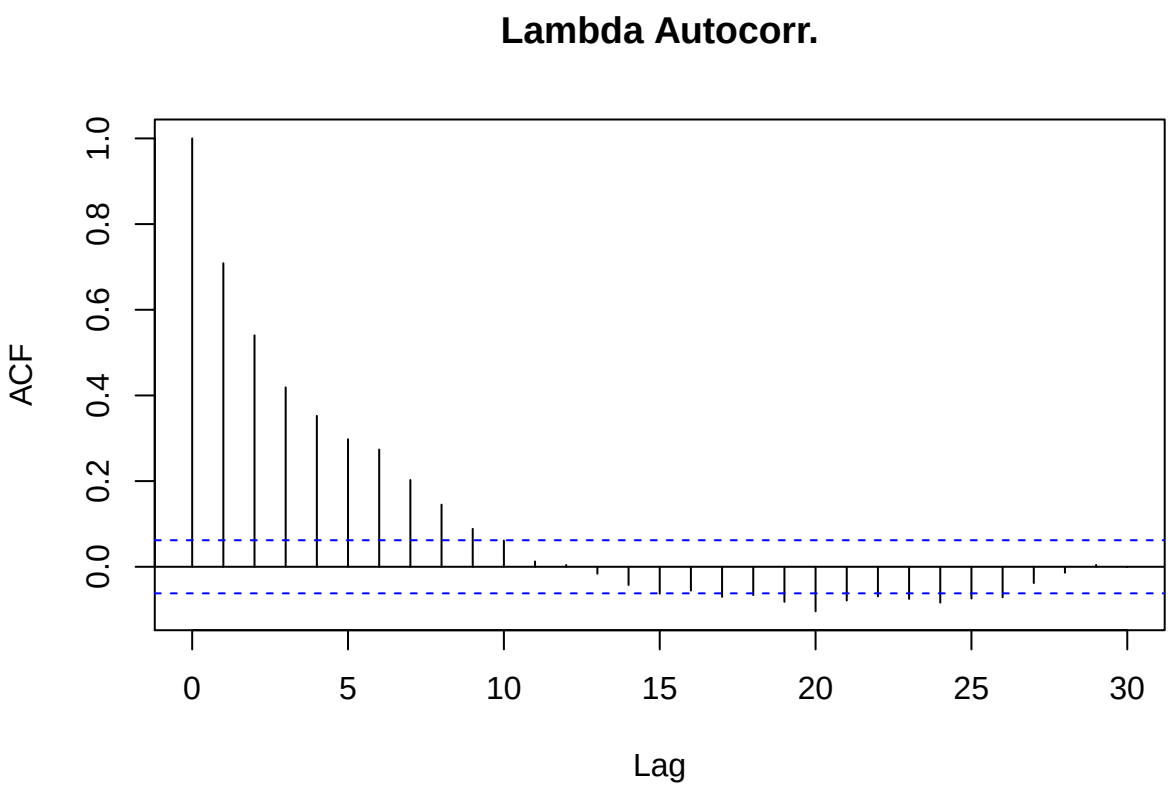








```
par(mfrow = c(1, 1))
acf(post_draws[["Lambda"]], main = "Lambda Autocorr.")
```



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)]
```

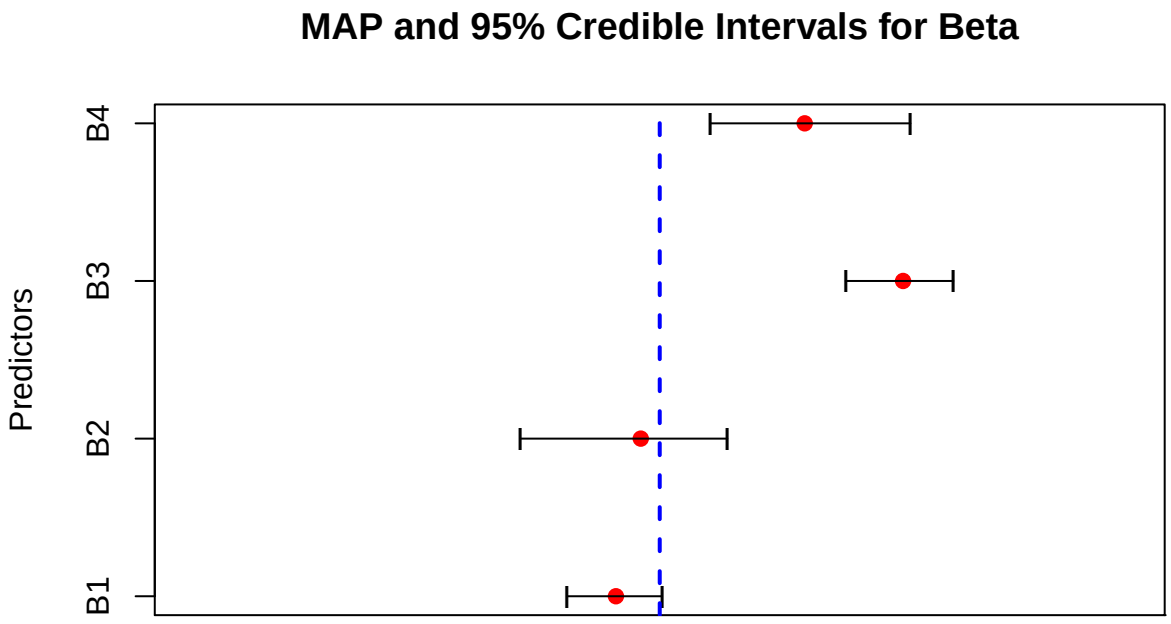
Interpreting the Model

```
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  0.001 -0.009
## Beta 2         -0.030  0.014 -0.004
## Beta 3          0.040  0.063  0.052
## Beta 4          0.011  0.054  0.031
## Tau Squared 1   0.024 10.267  0.443
## Tau Squared 2   0.023 12.919  1.905
## Tau Squared 3   0.393 19.378  0.565
## Tau Squared 4   0.142 15.039  0.502
## Lambda          0.283  2.678  1.029
## Mu              0.350  0.370  0.359
```

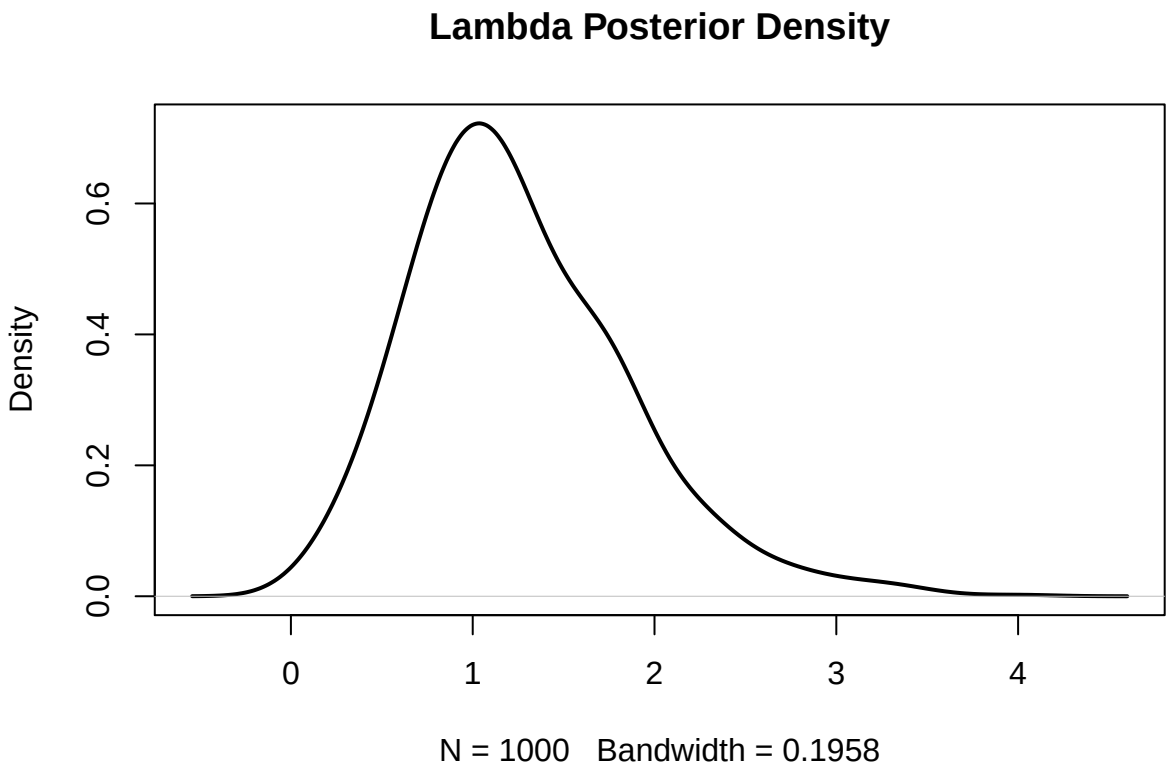
```
plot(CredInt95[1:4, "Mode"], 1:4, xlab = "Standardized Beta Coefficient",
    ylab = "Predictors", pch = 19, col = "red", xlim = c(-0.1,
        0.1), main = "MAP and 95% Credible Intervals for Beta",
    at = c(1, 2, 3, 4), labels = c("B1", "B2", "B3", "B4"))
for (i in 1:4) {
    x = CredInt95[i,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

Stepwise Selection

Bayesian lasso is just one of many methods of variable selection in modelling. We can contrast it with some other, more old-school methods like backwards and best subset selection:

```
x = as.matrix(icecream[, 2:5])  #predictors
y = icecream[, 1]  #outcome
n = nrow(x)
p = ncol(x)

I = diag(n)  #identity matrix
II = matrix(1, ncol = 1, nrow = n)  #vector of ones
X = (I - II %*% t(II)/n) %*% x
Y = (I - II %*% t(II)/n) %*% y


full = lm(Y ~ X - 1)
round(coef(summary(full)), 3)
```

```
##          Estimate Std. Error t value Pr(>|t|)
## Xprice     -1.287      0.715  -1.800    0.083
## Xincome    -0.002      0.002  -1.146    0.262
## Xtemp       0.003      0.000   8.038    0.000
## XYear       0.051      0.016   3.134    0.004
```

```
# remove income
bmod1 = lm(Y ~ X[, -2] - 1)
round(coef(summary(bmod1)), 3)
```

```
##              Estimate Std. Error t value Pr(>|t|)
## X[, -2]price   -1.150      0.709  -1.622    0.116
## X[, -2]temp     0.003      0.000   9.097    0.000
## X[, -2]Year     0.035      0.008   4.389    0.000
```

```
# remove price
bmod2 = lm(Y ~ X[, c(3, 4)] - 1)
round(coef(summary(bmod2)), 3)
```

```
##                  Estimate Std. Error t value Pr(>|t|)
## X[, c(3, 4)]temp    0.003      0.000   9.100        0
## X[, c(3, 4)]Year    0.036      0.008   4.409        0
```

Backwards elimination yields a 2-predictor model consisting of Temp and Year.

```
ICdf = as.data.frame(cbind(Y, X))
reg.model1 = regsubsets(V1 ~ ., ICdf, nvmax = 1)
summary(reg.model1)
```

```
## Subset selection object
## Call: regsubsets.formula(V1 ~ ., ICdf, nvmax = 1)
## 4 Variables  (and intercept)
##        Forced in Forced out
## price      FALSE      FALSE
## income     FALSE      FALSE
## temp       FALSE      FALSE
## Year       FALSE      FALSE
## 1 subsets of each size up to 1
## Selection Algorithm: exhaustive
##          price income temp Year
## 1  ( 1 ) " "   " "    "*"  " "
```

```
reg.model2 = regsubsets(V1 ~ ., ICdf, nvmax = 2)
summary(reg.model2)
```

```
## Subset selection object
## Call: regsubsets.formula(V1 ~ ., ICdf, nvmax = 2)
## 4 Variables  (and intercept)
##        Forced in Forced out
## price      FALSE      FALSE
## income     FALSE      FALSE
## temp       FALSE      FALSE
## Year       FALSE      FALSE
## 1 subsets of each size up to 2
## Selection Algorithm: exhaustive
##          price income temp Year
## 1  ( 1 ) " "   " "    "*"  " "
## 2  ( 1 ) " "   " "    "*"  "*"
```

```
reg.model3 = regsubsets(V1 ~ ., ICdf, nvmax = 3)
summary(reg.model3)
```

```
## Subset selection object
## Call: regsubsets.formula(V1 ~ ., ICdf, nvmax = 3)
## 4 Variables  (and intercept)
##        Forced in Forced out
## price      FALSE      FALSE
```

```
## income      FALSE      FALSE
## temp        FALSE      FALSE
## Year        FALSE      FALSE
## 1 subsets of each size up to 3
## Selection Algorithm: exhaustive
##          price income temp Year
## 1  ( 1 ) " "   " "    "*"  " "
## 2  ( 1 ) " "   " "    "*"  "*"
## 3  ( 1 ) "*"   " "    "*"  "*"
```

```
reg.model4 = regsubsets(V1 ~ ., ICdf, nvmax = 4)
```

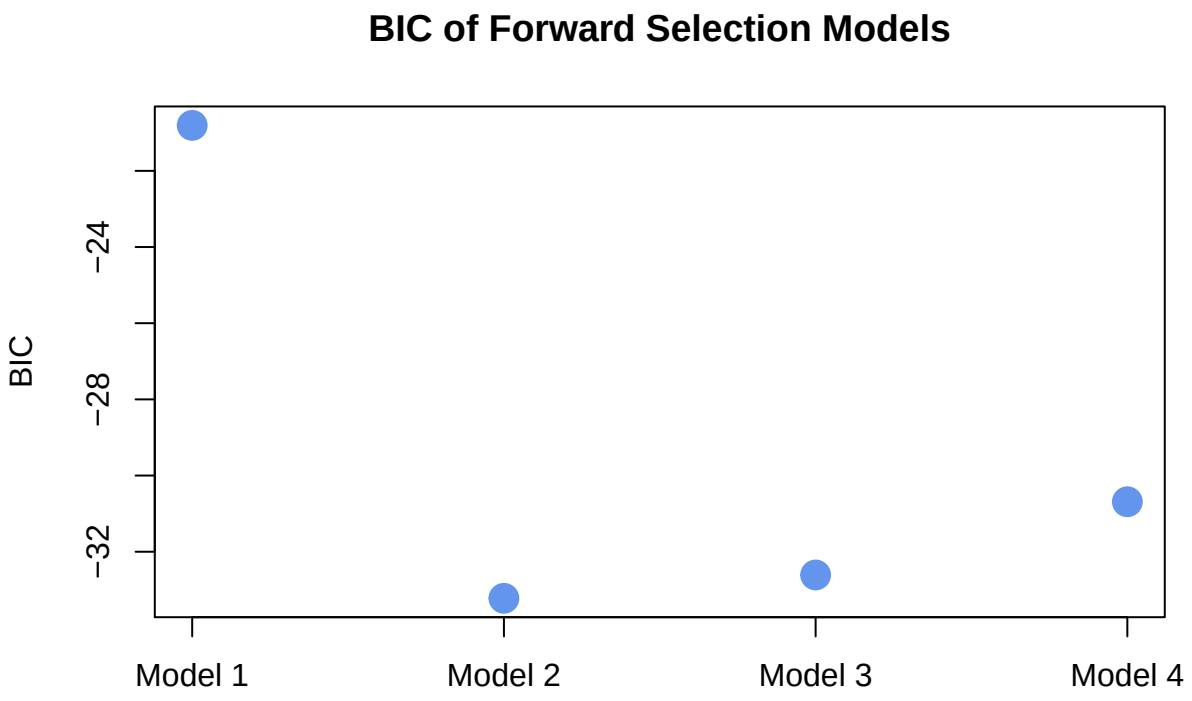
Performing subset selection, we find that the optimal lone predictor model includes Temp; best of two predictor models includes Temp and Year; for three predictors, Temp, Year, and Price.

Selection by Bayes Factor

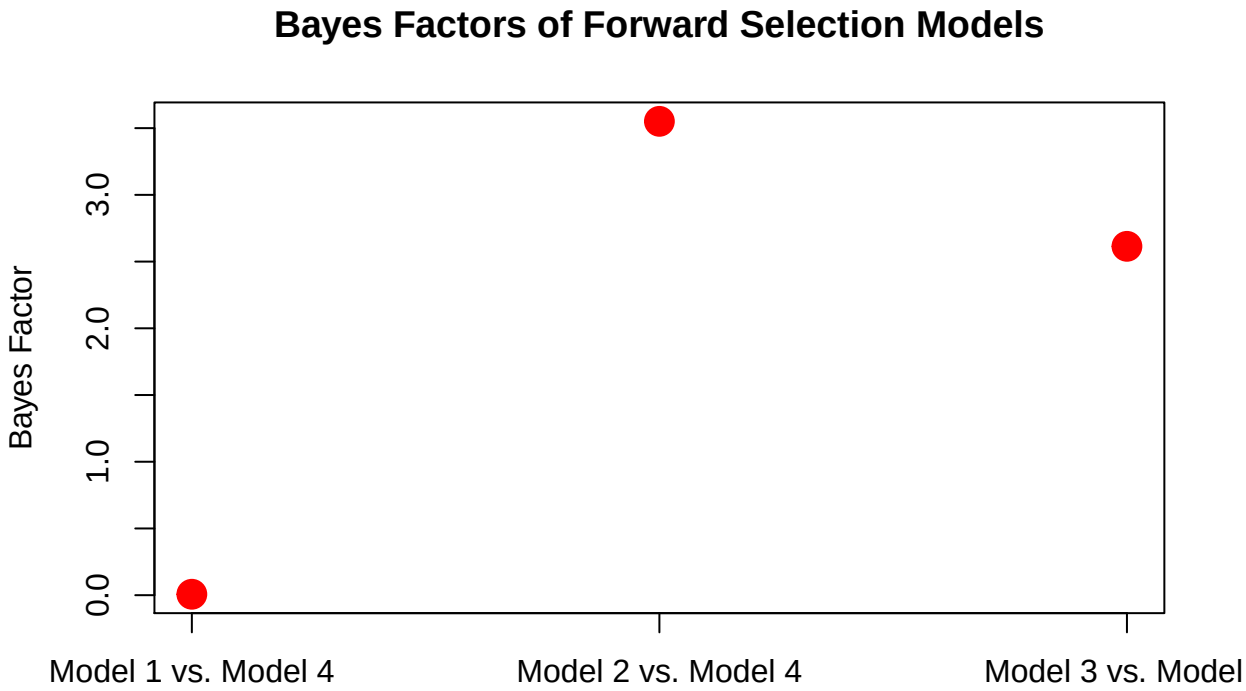
We may approximate the Bayes Factors using the BICs (under a uniform prior):

$$B_{ij} = \frac{p(M_i|\mathbf{y})}{p(M_j|\mathbf{y})} = \frac{p(\mathbf{y}|M_i)}{p(\mathbf{y}|M_j)} = \exp\Big\{-\frac{1}{2}(\text{BIC}_i - \text{BIC}_j)\Big\}$$

```
BIC = summary(reg.model4)[["bic"]]
plot(BIC, xaxt = "n", xlab = "", col = "cornflowerblue", pch = 19,
    cex = 2, main = "BIC of Forward Selection Models")
axis(1, at = 1:4, labels = paste("Model", 1:4))
```




```
BF = exp(-0.5 * (BIC[1:3] - BIC[4]))
plot(BF, xaxt = "n", xlab = "", col = "red", pch = 19, cex = 2,
    main = "Bayes Factors of Forward Selection Models", ylab = "Bayes Factor")
axis(1, at = 1:3, labels = c("Model 1 vs. Model 4", "Model 2 vs. Model 4",
    "Model 3 vs. Model 4"))
```



The 2-predictor model (Temp + Year) has the lowest Bayesian Information Criterion. The largest Bayes factor using the full model as the baseline, is also the 2-predictor model. In other words, compared to the posterior probability of the full model, the 2-predictor model has the highest posterior probability (compared to the 1- and 3-predictor models).

Stochastic Search Variable Selection

One last method we'll consider is SSVS, another Bayesian method akin to the Bayesian lasso.

Our priors are:

$$\beta|\gamma \sim \mathrm{N}_p(\mathbf{0}, D_\gamma R D_\gamma)$$

$$D_\gamma = \mathrm{diag}(a_1\tau_1, ..., a_p\tau_p)$$
$$\gamma \sim \mathrm{U}(2^p)$$

$$\sigma^2|\gamma \sim \mathrm{IG}(\upsilon_\gamma/2, \upsilon_\gamma\lambda_\gamma/2)$$

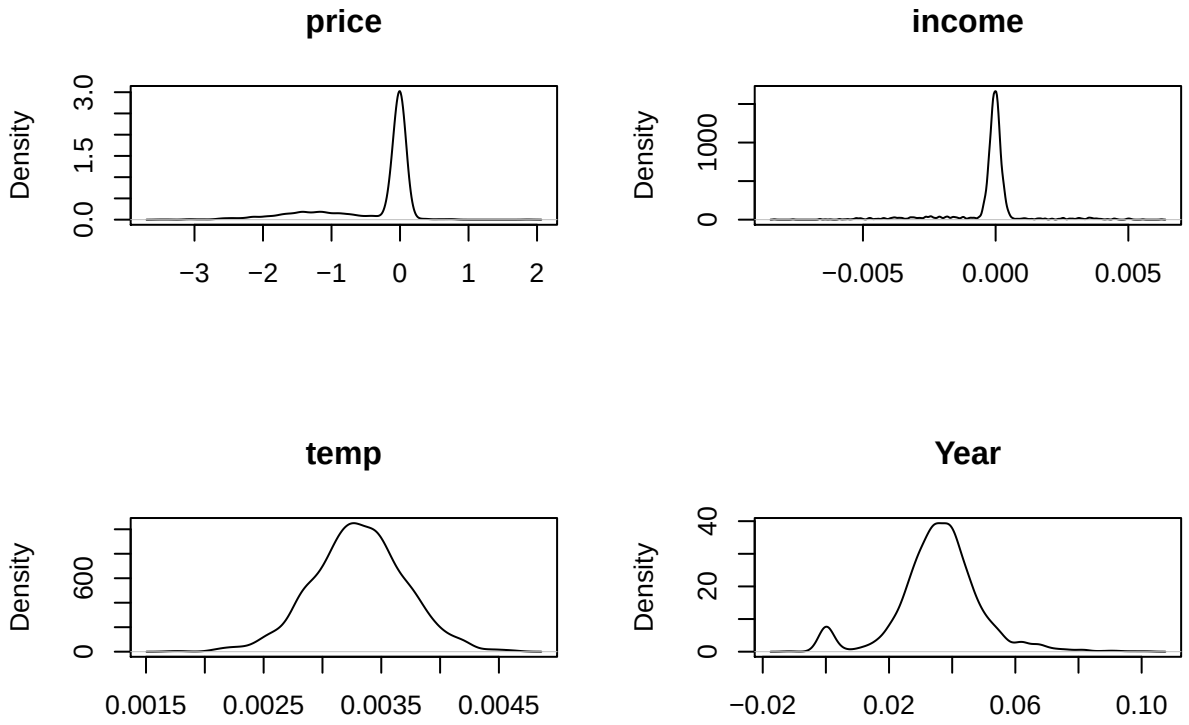
We will apply SSVS with the following parameters: $\boldsymbol{\tau} = \hat{\sigma}_{\boldsymbol{\beta}}/10$, $\mathbf{c} = 100 \cdot \mathbf{1}_4, R = I_p, \upsilon_{\boldsymbol{\gamma}} = \upsilon = 0$.

```
x = as.matrix(icecream[, 2:5])  #predictors
y = icecream[, 1]  #outcome
p = dim(x)[2]
n = dim(x)[1]
I = diag(n)  #identity matrix
II = matrix(1, ncol = 1, nrow = n)  #vector of ones
# SSVC assumes you don't include any predictors that would be
# included in every model (this means the intercept has to be
# integrated out)
```

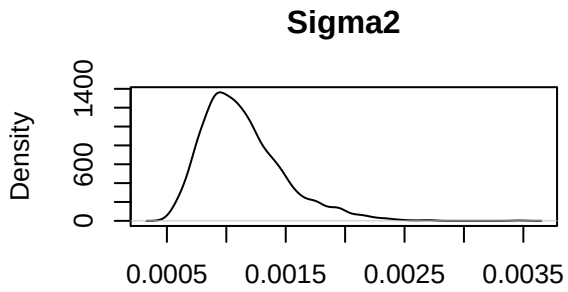
```
X = (I - II %*% t(II)/n) %*% x
Y = (I - II %*% t(II)/n) %*% y


# set prior parameters

R = diag(p)  #prior correlation matrix for beta (conditional on gamma) - taken to be identity i.e. unde
gamma.c = numeric()
beta.hat = solve(t(X) %*% X) %*% t(X) %*% Y  #OLS/MLE of beta
prob = rep(0.5, p)
c = 100  #set c to 100
tau = sqrt(diag(solve(t(X) %*% X)) %*% t(Y - X %*% beta.hat) %*%
    (Y - X %*% beta.hat)/n)/10  #set tau equal to SE of MLE divided by 10
a.c = rep(c, p)  #put c into vector
sig2.c = c(t(Y - X %*% beta.hat) %*% (Y - X %*% beta.hat)/n)

I = 130000  #number of MCMC iterations
sig2 = numeric()  #to collect sig2 draws
beta = gamma = matrix(0, nrow = p, ncol = I)  # to collect beta and gamma draws

# begin MCMC
for (b in 1:I) {

    ### Beta's
    D = diag(c(a.c * tau))
    D.inv = solve(D)
    A1 = t(X) %*% X/sig2.c + D.inv %*% solve(R) %*% D.inv
    A = solve(A1)
    beta.mean = A %*% t(X) %*% X %*% beta.hat/sig2.c
    beta.c = rmvnorm(1, beta.mean, A)
    beta.c = cbind(c(beta.c))


    #### Sigma
    sh = n/2
    sc = t(abs(Y - X %*% beta.c)) %*% abs(Y - X %*% beta.c)/2
    sig2.c = rinvgamma(1, sh, sc)


    ### gamma's
    for (i in 1:p) {
        D.temp.a = D
        D.temp.a[i, i] = c * tau[i]
        D.temp.b = D
        D.temp.b[i, i] = tau[i]
        a.1 = dmvnorm(c(beta.c), rep(0, p), D.temp.a %*% R %*%
            D.temp.a)
        b.1 = dmvnorm(c(beta.c), rep(0, p), D.temp.b %*% R %*%
            D.temp.b)
        gamma.c[i] = rbinom(1, 1, a.1/(a.1 + b.1))
        if (gamma.c[i] == 0) {
            a.c[i] = 1
        } else {
```

```
            a.c[i] = c
        }
    }

    sig2[b] = c(sig2.c)
    gamma[, b] = c(gamma.c)
    beta[, b] = c(beta.c)
}  #end MCMC


######### Thinning/Burn-in and obtaining Distinct Models and their
######### frequencies
EF = 2000  #Effective Sample Size
th = 60  #thinning
B = 10000  #burn-in
betath = gammath = matrix(0, nrow = p, ncol = EF)
sig2th = numeric()
model.distinct = matrix(0, ncol = p, nrow = 8)  #you may need to mess around with the nrows - it is the
M = numeric()  #Frequencies of distinct models


for (i in 1:EF) {
    betath[, i] = beta[, th * (i) + B]
    gammath[, i] = gamma[, th * (i) + B]
    sig2th[i] = sig2[th * (i) + B]

    temp = 0
    if (i == 1) {
        M[1] = 1
        model.distinct[1, ] = gammath[, i]
    }

    if (i > 1) {
        for (m in 1:length(M)) {
            if (all(gammath[, i] == model.distinct[m, ])) {
                M[m] = M[m] + 1
            } else {
                temp = temp + 1
            }

            if (sum(temp) == length(M)) {
                model.distinct[(m + 1), ] = gammath[, i]
                M[(m + 1)] = 1
            }

        }
    }

}



### Posterior Plots
```

```r
par(mfrow = c(2, 2))

for (i in 1:p) {
    plot(density(betath[i, ]), main = names(icecream)[(i + 1)],
        xlab = "")
}
```




```r
plot(density(sig2th), main = "Sigma2", xlab = "")
```



Based on the posterior plots of the beta vector, it appears that Temp and Year are not zero-centered, indicating that they should be included the final model; the other two predictors on the other hand, should be selected out.

Comparisons

In summary, we have performed 5 different methods of model selection:

1. Bayesian Lasso
2. Backwards Selection
3. Best Subset Selection: BIC
4. Best Subset Selection: Bayes Factor

5. SSVC

All methods agree on the same 2-predictor model of Temp and Year. I think in this particular dataset it is very clear that the Temp and Year model is superior to all other models so it's no surprise that all of thes model selection methods converge on the same model. Consequently it is difficult to contrast them given that they give the same results. I personally like the Bayesian Lasso and SSVC - they have sound statistical theory underlying them versus stepwise and subset selection methods. Unfortunately they are also computationally expensive and I worry about how they'll scale up to datasets with thousands of predictors. The non-Bayesian lasso would probably be a good method for big-data or wide-data (p>N) situations.