STAT 460 - Assignment 6

Steve Hof

08/12/2020

Question

Using the same data you received for the final project and assuming $\sigma^2 = 1$, implement the Bayesian Lasso described in the article *The Bayesian Lasso* by Park and Casella to sample from the joint posterior of $(\beta, \tau_1^2, \dots, \tau_p^2)$. Carefully write down the model and the full conditionals. You don't need to provide a detailed derivation of the full conditionals.

Part 1

Consider the multiple linear regression model

$$\boldsymbol{y} = \mu \mathbf{1}_n + \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$$

where we incorporate the Bayesian Lasso to estimate the regression parameters, $\boldsymbol{\beta}$. The Lasso estimates achieve:

$$\min_{\beta} (\tilde{\boldsymbol{y}} - \boldsymbol{X}\boldsymbol{\beta})^T (\tilde{\boldsymbol{y}} - \boldsymbol{X}\boldsymbol{\beta}) + \lambda \sum_{j=1}^{p} |\beta_j|$$

The full conditional of $\boldsymbol{\beta}$ is given by

$$p(\boldsymbol{\beta} \mid \boldsymbol{y}, \boldsymbol{\tau}, \sigma^2) \sim \text{MVN}\left(\boldsymbol{A}^{-1}\boldsymbol{X}^T\tilde{\boldsymbol{y}}, \sigma^2 \boldsymbol{A}^{-1}\right)$$

with

- $\tilde{\boldsymbol{y}} = \boldsymbol{y} - \bar{y}\mathbf{1}_n$,
- $\boldsymbol{A} = \boldsymbol{X}^T\boldsymbol{X} + \boldsymbol{D}_\tau^{-1}$, where $\boldsymbol{D}_\tau = \text{diag}(\tau_1^2, \dots, \tau_p^2)$

$\tau_1^2, \dots, \tau_p^2$ are conditionally independent with full conditional given by

$$p(\boldsymbol{\tau} \mid \boldsymbol{\beta}, \lambda, \sigma^2) \sim \text{InverseGaussian}\left(\sqrt{\frac{\lambda^2\sigma^2}{\beta_j^2}}, \lambda^2\right)$$

To calculate our initial λ value we used

$$\lambda^{(0)} = \frac{p\sqrt{\hat{\sigma}^2_{\text{LS}}}}{\sum_{j=1}^{p} |\hat{\beta}_j^{\text{LS}}|}$$

We now read in the data and set up our global variables.

For convenience and readability of our code we write helper functions that perform the updates for $\boldsymbol{\beta}$ and $\boldsymbol{\tau}$, a function to execute removing a burn-in and thinning, and a function to calculate $\hat{R}$ which we will use to help determine if our posterior chain has converged.

```r
update.beta = function(tau.curr, bet) {
  D.inv = solve((1 / tau.curr) * diag(p))
  A = partial.A + D.inv
  A.inv = solve(A)
  mu = A.inv %*% t(X) %*% y.tilde
  Sig = sig2 * A.inv
  mvrnorm(n = 1, mu = mu, Sigma = Sig)
}

update.tau = function(bet, lam, tau) {
  for(j in 1:p) {
    mu.prime = sqrt(lam^2 * sig2 / bet[j]^2)
    tau[j] = (rinvgauss(n = 1, mean = mu.prime, shape = lam^2))^2
  }
  return(1 / tau)
}

burn.and.thin = function(post, bi, ti) {
  thin.indx = seq(from = bi, to = length(post[, 1]), by = ti)
  thin.post = post[thin.indx, ]
  colnames(thin.post) = beta.names
  return(thin.post)
}

calc.rhat = function(m, nchain, J, chain) {
  rhat = numeric(J)
  for (j in 1:J) {
    psi.mean = mean(chain[, j])
    psi.bar = numeric(m)
    aux.w = numeric(m)
    for (k in 1:m) {
      sub.chain = chain[seq((k - 1) * nchain + 1, k * nchain, 1), j]
      psi.bar[k] = mean(sub.chain)
      aux.w[k] = (1 / (nchain - 1)) * sum((sub.chain - mean(sub.chain))^2)
    }
    B = (nchain / (m - 1)) * (sum((psi.bar - psi.mean)^2))
    W = (1 / m) * sum(aux.w)
    VP = ((nchain - 1) / nchain) * W + (1 / nchain) * B
    rhat[j] = sqrt(VP / W)
    names(rhat) = beta.names
  }
  return(rhat)
}
```

We then code our trusty Gibbs Sampler.

```r
gibbs = function(n_iter, init, priors) {
  beta.out = matrix(data = NA, nrow = n_iter, ncol = p)
  beta.curr = init$beta
  beta.out[1, ] = beta.curr
  tau.curr = init$tau

  for(k in 2:n_iter) {
    tau.curr = update.tau(bet = beta.curr, lam = priors$lam, tau = tau.curr)
    beta.curr = update.beta(tau.curr = tau.curr, bet = beta.curr)
    beta.out[k, ] = beta.curr
  }
  colnames(beta.out) = beta.names
  return(beta.out)
}
```

The following code sets up our initial values and priors, where we use regular regression to supply us our initial values for $\boldsymbol{\beta}$ and initialize all $\tau_i, (i = 1, \ldots, p)$ to 1.

```r
priors = list()
init = list()
n_iter = 10000
model = lm(y ~ X - 1)
init$beta = model$coefficients
init$tau = rep(1, p)

priors$sig2 = 1
priors$lam = p / sum(abs(init$beta))
```

And finally, we run our Gibbs sampler, remove a $1,000$ iteration burn-in and view the final 4 values of our β_j's

```r
post = gibbs(n_iter, init, priors)
burnt.post = burn.and.thin(post = post, bi = 1000, ti = 1)
t(tail(post, 4))
```

```
##              [9997,]    [9998,]     [9999,]       [10000,]
## beta1   0.579853361 0.71137847  0.81777391  0.7228419767
## beta2   2.478345987 2.62290493  2.40793680  2.6199482721
## beta3   0.186943327 0.05274896 -0.03056096 -0.0580377943
## beta4   2.016742061 1.78218255  1.61392428  1.7632879934
## beta5   0.059675200 0.09215564  0.13299049 -0.0008582897
## beta6   1.159580278 1.23629830  1.29257166  1.3038573717
## beta7  -0.004649264 0.08644419 -0.03361989 -0.0478363248
## beta8   1.633178933 1.83995517  1.96186749  1.9393742488
## beta9   1.478739291 1.33380432  1.54713308  1.4540530731
## beta10  0.918374594 0.73670317  0.80267192  0.8132600833
```

The values for each β_j remain fairly constant through the final 4 iterations, which makes us hopeful that convergence has occurred.

We now calculate the $\hat{R}$ values

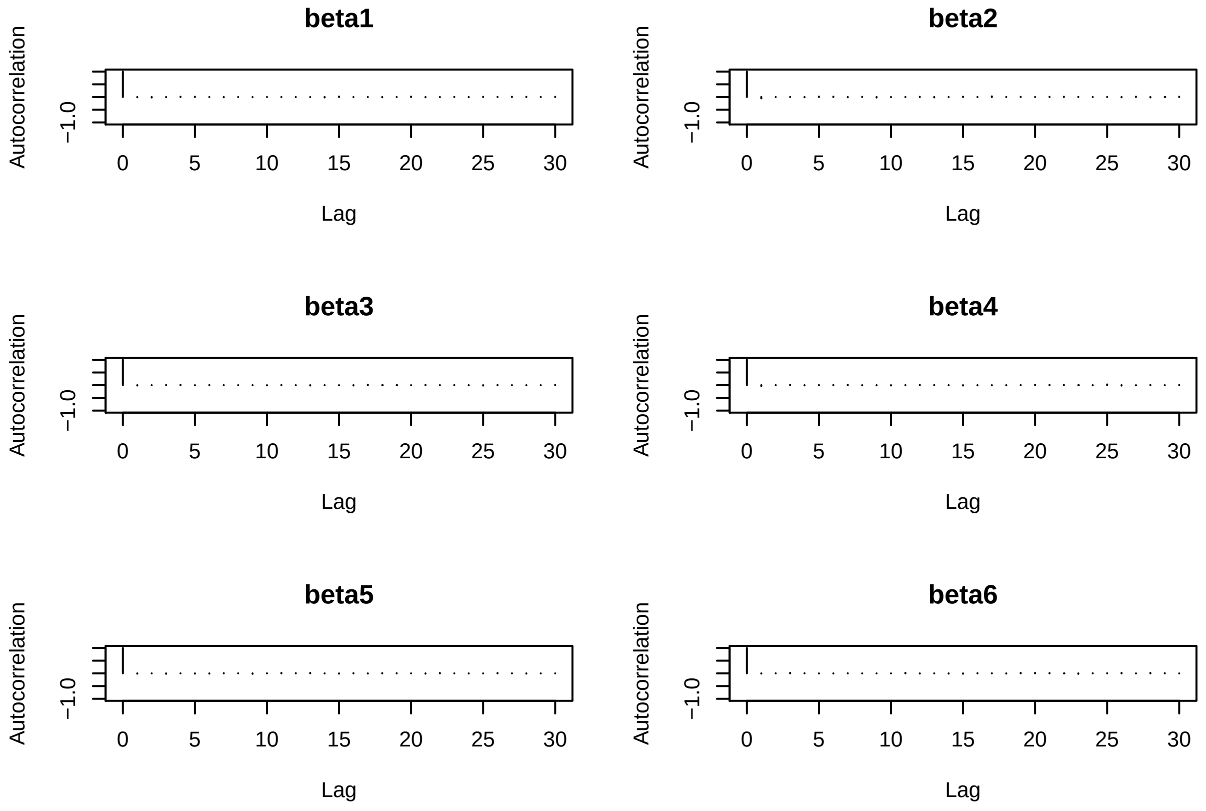
```
rhat.post = calc.rhat(m = 5, nchain = 100, J = 10, chain = post)
rhat.post
```

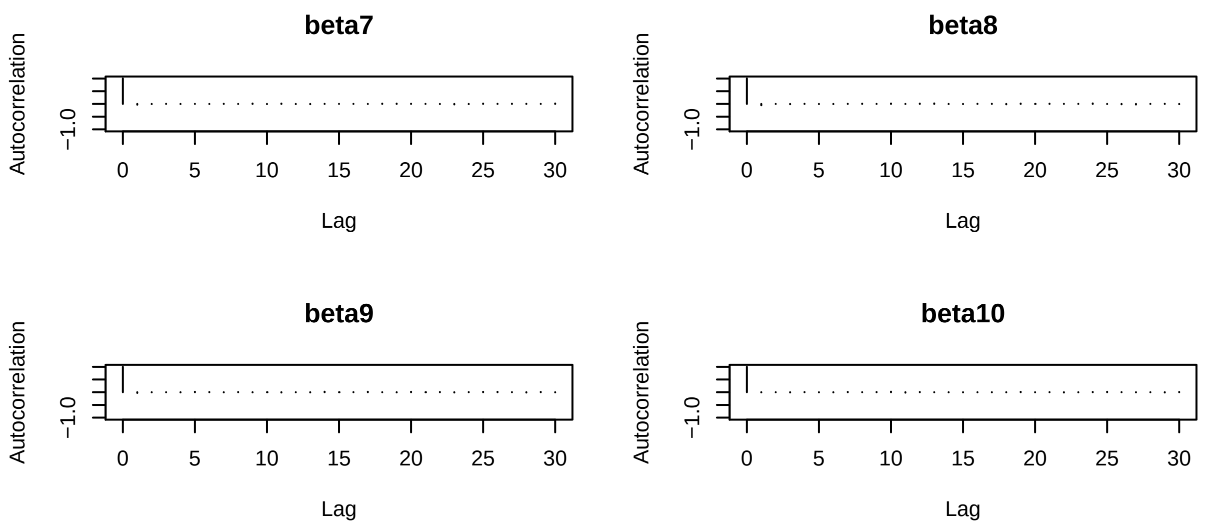
```
##     beta1     beta2     beta3     beta4     beta5     beta6     beta7     beta8
## 0.9996130 1.0033381 1.0030516 1.0108534 0.9979884 1.0074524 1.0000189 1.0021707
##     beta9    beta10
## 1.0017037 0.9997978
```

The $\hat{R}$ values are all very close to one without even thinning the chain.

Let's now check the auto-correlation

```
coda::autocorr.plot(as.mcmc(post), lag.max = 30)
```

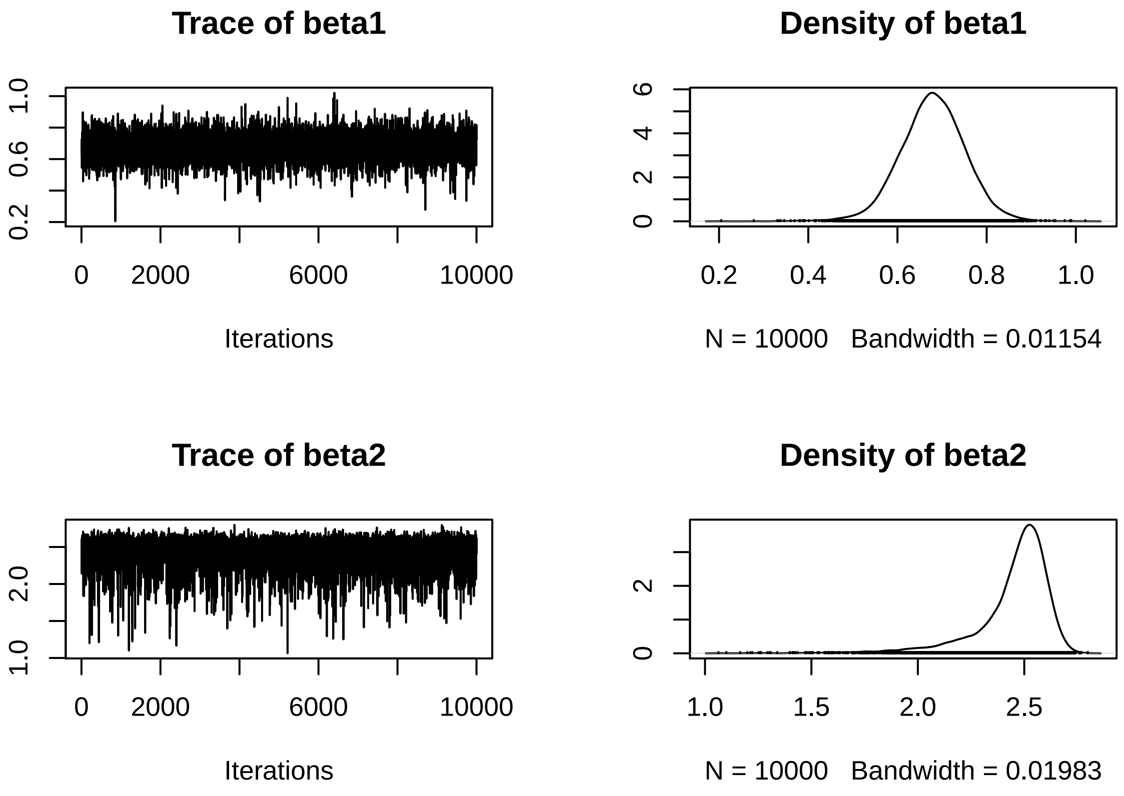




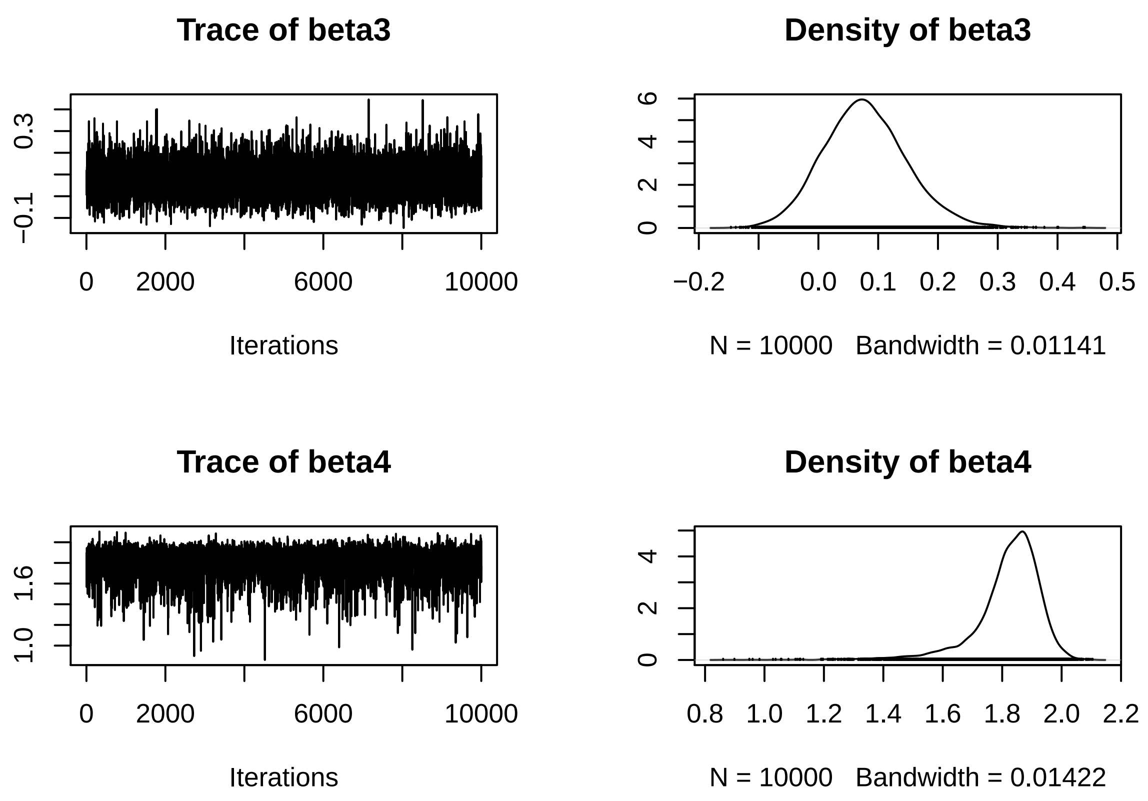
Notice we have almost no auto-correlation at even a lag of only 2!

Finally, we check the trace plots and densities of our β_j's.

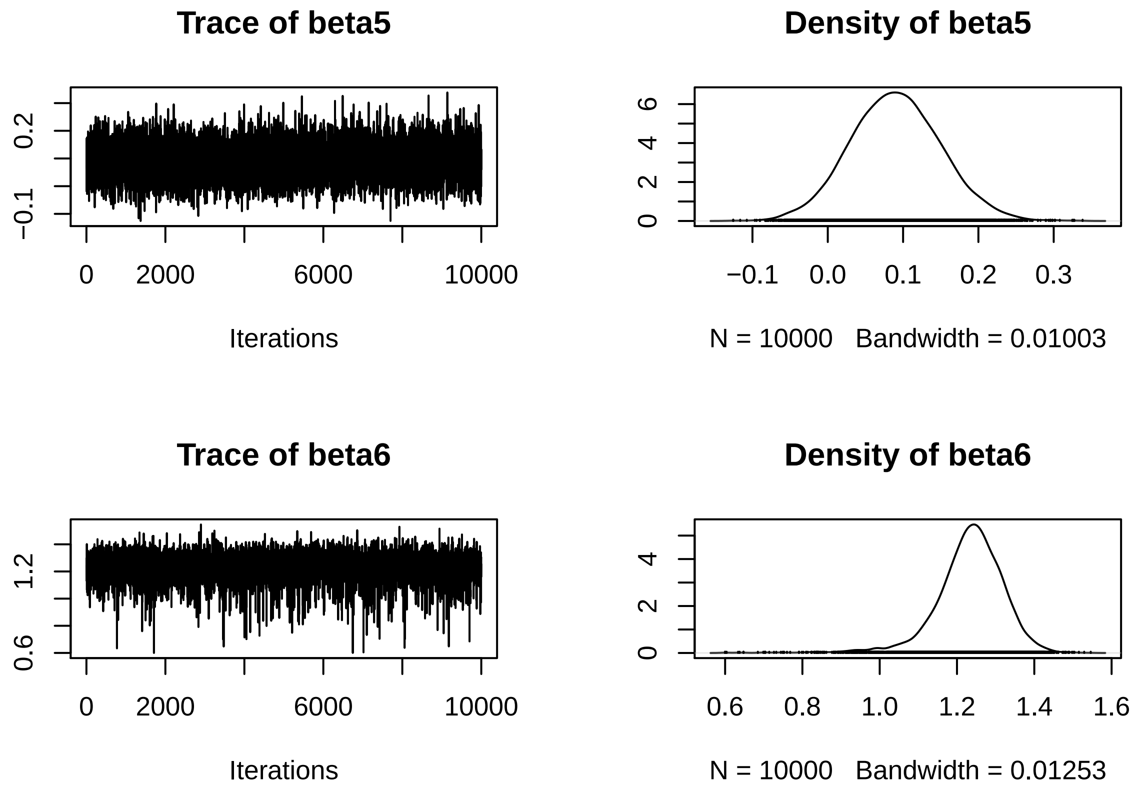
```r
plot(as.mcmc(post[, 1:2]))
```



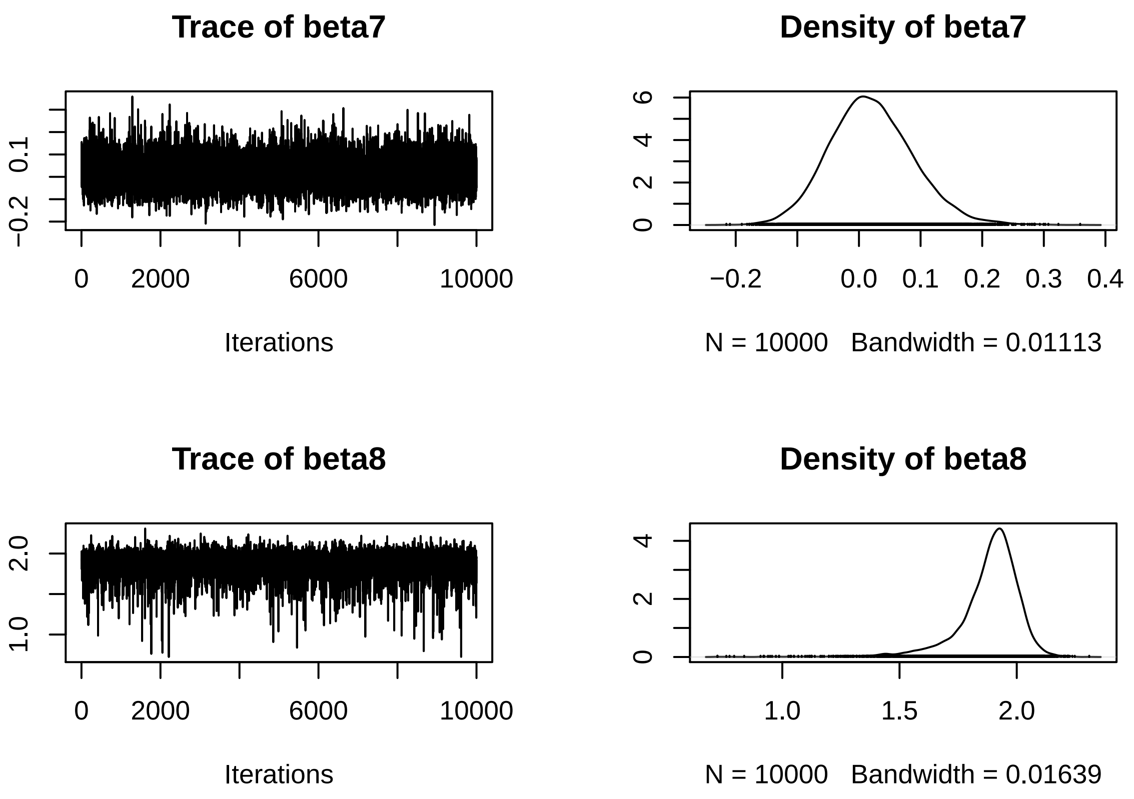
```r
plot(as.mcmc(post[, 3:4]))
```



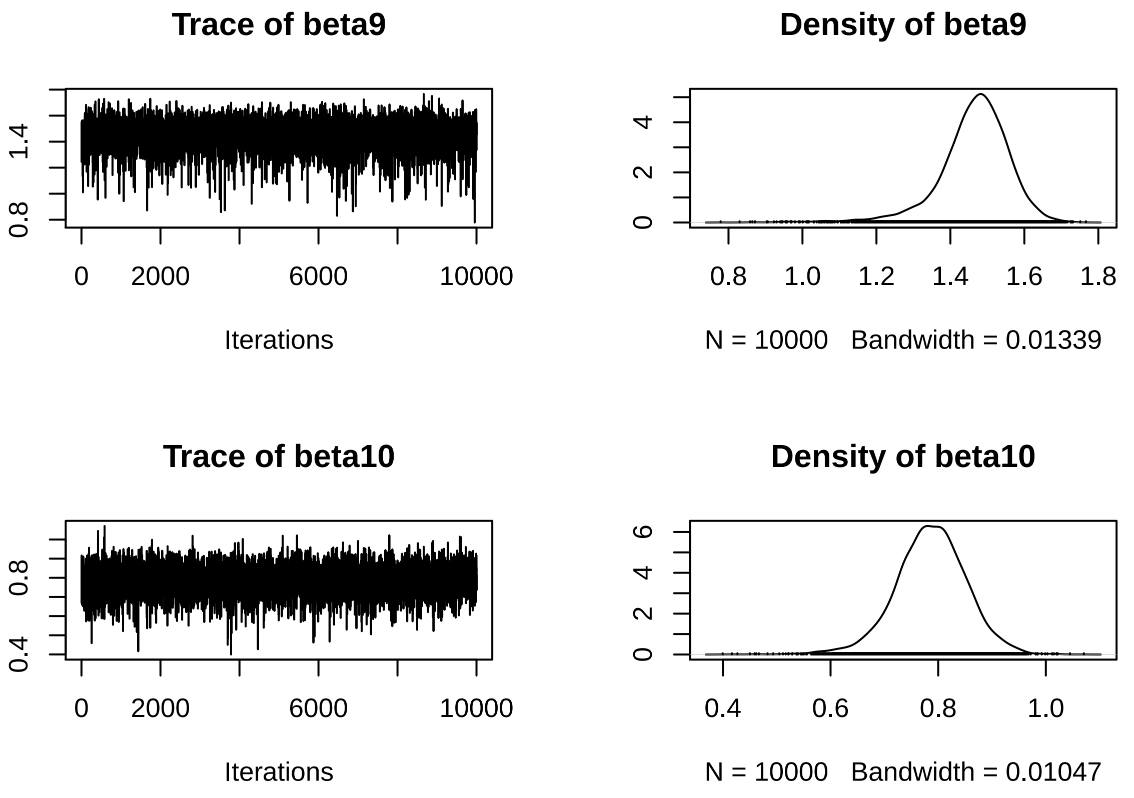
```r
plot(as.mcmc(post[, 5:6]))
```



```r
plot(as.mcmc(post[, 7:8]))
```



```r
plot(as.mcmc(post[, 9:10]))
```



Although β_2, β_4 and β_8 make occasional dips below their mean, when we calculate the posterior mean, these tiny anomalies will vanish.

Since all of our diagnostics check out, we will now present our results including the posterior mean, posterior standard deviation, quantiles and a 95% credible interval for each coefficient

Summary statistics for the mean and standard deviation are given in the following output.

```
summary(as.mcmc(post))$statistics
```

```
##              Mean         SD     Naive SE Time-series SE
## beta1  0.67913009 0.07236135 0.0007236135   0.0007009469
## beta2  2.45075499 0.17785090 0.0017785090   0.0016845042
## beta3  0.07831903 0.07022003 0.0007022003   0.0006922170
## beta4  1.82011022 0.11518791 0.0011518791   0.0011166363
## beta5  0.09037119 0.05970851 0.0005970851   0.0005970851
## beta6  1.23329296 0.08887299 0.0008887299   0.0008887299
## beta7  0.02047402 0.06779701 0.0006779701   0.0006582710
## beta8  1.88340883 0.13596369 0.0013596369   0.0012895230
## beta9  1.46404623 0.09700896 0.0009700896   0.0009436228
## beta10 0.78615421 0.06613301 0.0006613301   0.0006613301
```

The quantiles of the distribution are given in the following output.

```
summary(as.mcmc(post))$quantiles
```

```
##               2.5%         25%        50%        75%     97.5%
## beta1   0.53545936  0.63389874 0.67977003 0.72597534 0.8190535
## beta2   1.95556434  2.40139043 2.49370041 2.55954800 2.6628574
## beta3  -0.05184926  0.03083006 0.07532472 0.12182325 0.2260091
## beta4   1.51460327  1.77808094 1.84037711 1.89149110 1.9769253
## beta5  -0.02483364  0.04924948 0.08952367 0.13001356 0.2099375
## beta6   1.02913832  1.18910178 1.24008999 1.28902605 1.3833965
## beta7  -0.10503301 -0.02556904 0.01719352 0.06320908 0.1612061
## beta8   1.53798677  1.83354039 1.90672137 1.96428976 2.0735952
## beta9   1.23081839  1.41834286 1.47432242 1.52516239 1.6238184
## beta10  0.64772711  0.74544311 0.78777505 0.82895207 0.9123132
```

The 95% Credible Intervals for each of the β_j's are given in the following output.

```
t(hdi(as.mcmc(post)))
```

```
##              lower     upper
## beta1   0.53756827 0.8199366
## beta2   2.08178554 2.7102462
## beta3  -0.05725742 0.2175293
## beta4   1.58531410 2.0105576
## beta5  -0.02490568 0.2097714
## beta6   1.06146172 1.4009462
## beta7  -0.10622001 0.1582412
## beta8   1.61272129 2.1188095
## beta9   1.27205089 1.6465190
## beta10  0.66137700 0.9229210
## attr(,"credMass")
## [1] 0.95
```

Overall, using the Bayesian Lasso in Bayesian Regression provided an extremely fast sampling algorithm that converged quickly. So fast that we did not even need to apply thinning.

The end. :)