Project 5

Benjamin Draves

Exercise 1

(a)

Consider the hidden Markov model $(X_i, Y_i)_{i=1}^{n=200}$ where $X_i \in \{1, 2, 3\}$ and $Y_i | X_i = s \sim N(\mu_s, \sigma_s^2)$. First we calculate the forward probabilities which are given by the following.

$$\begin{aligned} f_1(x_1) &= \mathbb{P}(X_1, Y_1) \\ f_2(x_2) &= \mathbb{P}(X_2, Y_1, Y_2) \\ &\vdots \\ f_i(x_i) &= \mathbb{P}(X_i, Y_1, \ldots, Y_i) \end{aligned}$$

In practice we calculate the log of these probabilities.

$$\begin{aligned} \widetilde{f}_1(x_1) &= \log[\mathbb{P}(X_1, Y_1)] \\ \widetilde{f}_2(x_2) &= \log[\mathbb{P}(X_2, Y_1, Y_2)] \\ &\vdots \\ \widetilde{f}_i(x_i) &= \log[\mathbb{P}(X_i, Y_1, \ldots, Y_i)] \end{aligned}$$

```
# soft max (log-sum-exp) of vector `x`
LOGEPS <- log(.Machine$double.eps / 2)
lse <- function (x) {
  m <- max(x); x <- x - m
  m + log(sum(exp(x[x > LOGEPS])))
}

#Set variables for this algorithm

logI <- log(c(0, 1, 0)) # initial probabilities
logP <- matrix(log(c(0.50, 0.50, 0, 0.05, 0.90, 0.05, 0, 0.5, 0.5)),
               byrow = TRUE,
               nrow = 3) # Transition probs
Y <- cgh$log_ratio
logPYX <- matrix(c(dnorm(Y, mean = -1, sd = .7, log = TRUE),
                   dnorm(Y, mean = 0, sd = .5, log = TRUE),
                   dnorm(Y, mean = 1, sd = .7, log = TRUE)),
                 nrow = 200,
                 ncol = 3) # log[ P( Y_i | X_i = x) ]

forward <- function (s, logI, logP, logE) {
```

```
  n <- length(s); ns <- length(logI) #get dimensions
  f <- matrix(nrow = n, ncol = ns)
  f[1,] <- logI + logPYX[1,]
  for (i in 2:n) # for each position in sequence
    for (j in 1:ns) # for each X_i
      f[i, j] <- lse(f[i - 1,] + logP[,j]) + logPYX[i,j]
  f
}
f <- forward(Y, logI, logP, logPYX)
knitr::kable(as.data.frame(tail(f)))
```

	V1	V2	V3
[195,]	-196.6476	-191.5936	-193.8472
[196,]	-195.0963	-194.1988	-198.8496
[197,]	-196.8685	-194.3669	-199.0164
[198,]	-197.4882	-195.3724	-200.4511
[199,]	-200.1735	-195.9747	-199.2230
[200,]	-200.9410	-196.3838	-199.8068

Now that we have calculated these probabilities, we can find $\log \mathbb{P}(Y)$ through the following calculations.

$$\mathbb{P}(Y) = \sum_{x_{200}} \mathbb{P}(X_{200}, Y) = \sum_{x_{200}} f_{200}(x_{200})$$

$$\log \mathbb{P}(Y) = \log \sum_{x_{200}} \exp\{\log \mathbb{P}(X_{200}, Y)\} = \log \sum_{x_{200}=1}^{3} \exp\left\{\widetilde{f}_{200}(x_{200})\right\} = lse(\vec{\widetilde{f}}_{200}(x_{200}))$$

Hence we see that taking the *lse* of the final row of the forward probabilities we can arrive at our final solution.

```
logPY <- lse(f[200,])
logPY
```

```
## [1] -196.3416
```

(b)

```
# compute most likely assignment of states given data (MAP)
viterbi <- function (s, logI, logP, logE) {

  n <- length(s); ns <- length(logI) # = nrow(logE) = nrow(logP)
  m <- matrix(nrow = n, ncol = ns) # maxima
  b <- matrix(nrow = n, ncol = ns) # backtrack pointers

  # recurse
  m[1,] <- logI + logPYX[1,]
  for (i in 2:n) { # for each position in sequence
    for (j in 1:ns) { # for each X_i
      u <- m[i - 1,] + logP[,j]
      m[i, j] <- max(u) + logPYX[i,j]
      b[i - 1, j] <- which.max(u)
```

```
    }
  }
  # backtrack
  v <- numeric(n)
  v[n] <- which.max(m[n,]) # b[n]
  for (i in (n - 1):1)
    v[i] <- b[i, v[i + 1]]
  list(m = m, b = b, seq = v)
}

res <- viterbi(Y, logI, logP, logPYX)
xhat <- res$seq
```

Now that we have calculated $\widehat{X}$ we can calculate $\log \mathbb{P}(\widehat{X}|Y)$ as follows.

$$\begin{aligned}
\mathbb{P}(\widehat{X}|Y) &= \frac{\mathbb{P}(Y|\widehat{X})\mathbb{P}(\widehat{X})}{\mathbb{P}(Y)} \\
\log \mathbb{P}(\widehat{X}|Y) &= \log \mathbb{P}(Y|\widehat{X}) + \log \mathbb{P}(\widehat{X}) - \log \mathbb{P}(Y) \\
&= \sum_{i=1}^{n} \log \mathbb{P}(Y_i|\widehat{X}_i) + \log \mathbb{P}(\widehat{X}) - \log \mathbb{P}(Y)
\end{aligned}$$

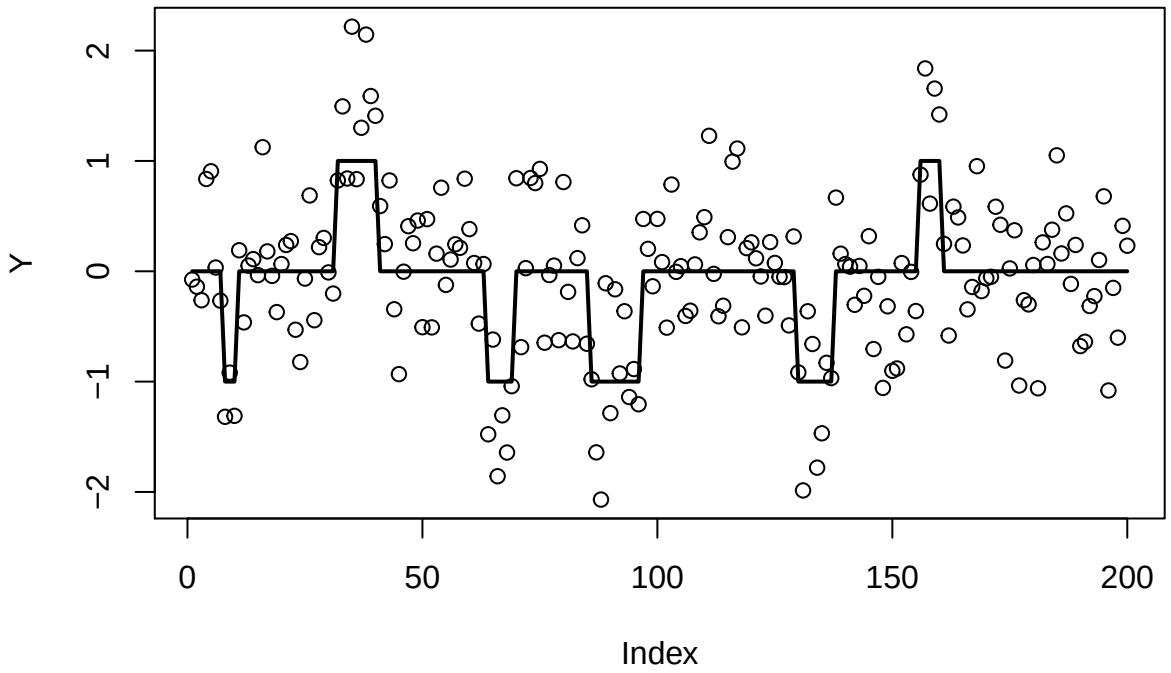
Notice here we used the fact that given $\widehat{X}_i$ the Y_i are independent. Moreover, we know the distribution of $Y_i|\widehat{X}_i$, can calculate $\log \mathbb{P}(\widehat{X}) = \sum_{i=1}^{n-1} \log \mathbb{P}(X_i + 1|X_i)$ from the Markov chain transition probabilities, and we've already calculated $\log \mathbb{P}(Y)$ in part a. Hence we can calculate the above as follows.

```
#calculate log P(Y|Xhat)
lPYX <- sum(logPYX[cbind(1:200, xhat)])

#calculate log P(Xhat)
lPX <- 0
for(i in 2:length(xhat)) lPX <- lPX + logP[xhat[i-1],xhat[i]]

#calculate solution
logPXY <- lPYX + lPX - logPY
logPXY
```

```
## [1] -16.5959
```

(c)

```
plot(Y)
mu <- c(-1,0,1)
points(mu[xhat],
       type = "l",
       lwd = 2)
```



Here we see that the HMM fits the data reasonably well. Due to the high variance of the Y_i, it is difficult for the model to detect when the underlying chain states are different from 2. Therefore, the majority of these states are estimates to be 2 corresponding with $\widehat{Y}_i = 0$ for the majority of the model. The model does however pick up clear trends in change especially for duplications in the (30,50) region and deletions in the $(80, 100)$ region. The model does fairly well throughout but could possibly be overfitting in some of the small regions. In general, this model fits the data well.

(d)

Recall that by definition $f_i(x) = \mathbb{P}(X_i, Y_1 = y_1, \ldots, Y_i = y_i)$. Hence $\mathbb{P}(x_{200}) = \mathbb{P}(X_{200}, Y = \vec{y})$. Therefore, we have

$$\mathbb{P}(X_{200} = 2|Y) = \frac{f_{200}(2)}{\mathbb{P}(Y)}$$
$$\log(\mathbb{P}(X_{200} = 2|Y)) = \log(f_{200}(2)) - \log(\mathbb{P}(Y))$$

Notice however this doesn't change based on the state. Hence, using the numbers we've already calculated we can find the probabilities of each state ending as follows.

$$\mathbb{P}(X_{200} = 2|Y) = \exp\left\{\widetilde{f}_{200}(x) - \log \mathbb{P}(Y)\right\}$$
$$\frac{\mathbb{P}(X_{200} = 1|Y)}{\mathbb{P}(X_{200} = 3|Y)} = \exp\left\{\widetilde{f}_{200}(1) - \widetilde{f}_{200}(3)\right\}$$

```
lp1 <- f[200, 1] - logPY
lp2 <- f[200, 2] - logPY
lp3 <- f[200, 3] - logPY

exp(lp2)
```

```
## [1] 0.9586747
```

```
exp(lp1 - lp3)
```

```
## [1] 0.3216633
```

Therefore we see that the probability of the last prob being normal is 0.9586747 and is 0.3216633 times more likely to be deleted than duplicated.

Exercise 2

(a)

$$\begin{aligned}
\mathbb{P}(\theta_j|\beta_0,\beta,\theta_{[-j]},Y) &\propto \mathbb{P}(\beta_0,\beta,\theta_{[-j]},Y|\theta_j)\mathbb{P}(\theta_j) \\
&\propto \mathbb{P}(\beta_0,\beta,Y|\theta_{[-j]},\theta_j)\mathbb{P}(\theta_{[-j]}|\theta_j)\mathbb{P}(\theta_j) \\
&\propto \mathbb{P}(\beta_0,\beta,Y|\theta)\mathbb{P}(\theta) \\
&\propto \mathbb{P}(Y|\beta_0,\beta)\mathbb{P}(\beta|\theta)\mathbb{P}(\beta_0)\mathbb{P}(\theta)
\end{aligned}$$

Now recall that we can drop any probability statement that does not include the θ_j of interest. Hence, with a little more work we see that

$$\begin{aligned}
\mathbb{P}(Y|\beta_0,\beta)\mathbb{P}(\beta|\theta)\mathbb{P}(\beta_0)\mathbb{P}(\theta) &\propto \Pi_{j=1}^{p}\mathbb{P}(\beta_j|\theta_j)\mathbb{P}(\theta) \\
&\propto \mathbb{P}(\beta_j|\theta_j)\mathbb{P}(\theta) \\
&\propto \sigma^2(\theta_j)^{-1/2}\exp\left\{-\frac{\beta_j^2}{2\sigma^2(\theta_j)}\right\}\exp\left\{\frac{h}{2}\sum_{u=1}^{p}(2\theta_u-1)+\frac{1}{T}\sum_{(u,v)\in G}(2\theta_u-1)(2\theta_v-1)\right\} \\
&\propto \sigma^2(\theta_j)^{-1/2}\exp\left\{-\frac{\beta_j^2}{2\sigma^2(\theta_j)}+\frac{h}{2}\sum_{u=1}^{p}(2\theta_u-1)+\frac{1}{T}\sum_{(u,v)\in G}(2\theta_u-1)(2\theta_v-1)\right\}
\end{aligned}$$

Now, again, we can drop all terms not including θ_j. Notice that for the sum over the imposed graph G, this means that we only sum over pairs $(j,v)\in G$. This set, however, is just the neighbors of j, N_j. Hence we can reduce this term further as follows.

$$\begin{aligned}
\mathbb{P}(\beta_j|\theta_j)\mathbb{P}(\theta) &\propto \sigma^2(\theta_j)^{-1/2}\exp\left\{-\frac{\beta_j^2}{2\sigma^2(\theta_j)} + \frac{h}{2}\sum_{u=1}^{p}(2\theta_u - 1) + \frac{1}{T}\sum_{(u,v)\in G}(2\theta_u - 1)(2\theta_v - 1)\right\} \\
&\propto \sigma^2(\theta_j)^{-1/2}\exp\left\{-\frac{\beta_j^2}{2\sigma^2(\theta_j)} + \frac{h}{2}(2\theta_j - 1) + \frac{1}{T}\sum_{v\in N_j}(2\theta_j - 1)(2\theta_v - 1)\right\} \\
&\propto \sigma^2(\theta_j)^{-1/2}\exp\left\{-\frac{\beta_j^2}{2\sigma^2(\theta_j)} + h\theta_j - \frac{h}{2} + \frac{2}{T}\theta_j\sum_{v\in N_j}(2\theta_v - 1) - \frac{1}{T}\sum_{v\in N_j}(2\theta_v - 1)\right\} \\
&\propto \sigma^2(\theta_j)^{-1/2}\exp\left\{-\frac{\beta_j^2}{2\sigma^2(\theta_j)} + h\theta_j + \frac{2}{T}\theta_j\sum_{v\in N_j}(2\theta_v - 1)\right\} \\
&\propto \sigma^2(\theta_j)^{-1/2}\exp\left\{-\frac{\beta_j^2}{2\sigma^2(\theta_j)} + \theta_j\left(h + \frac{2}{T}\sum_{v\in N_j}(2\theta_v - 1)\right)\right\}
\end{aligned}$$

(b)

Here we see that the function

$$\beta_0^{(t+1)} = \text{step-laplace-mh}(\beta_0^{(t)}; Y, 1_n, X\beta^{(t)}, 0, 0)$$

corresponds to the model

$$\begin{aligned}
Y_i|\beta &\overset{ind}{\sim} \text{Bern}[\text{logit}^{-1}((1_n)_i\beta + X\beta^{(t)})] \\
\beta &\sim N(0, \infty)
\end{aligned}$$

Recall that in the Gibbs framework, at this step in the algorithm we condition on $\theta^{(t+1)}$ and $\beta^{(t)}$ and try to estimate $\beta_0^{(t+1)}$. Well, taking $\beta = \beta_0$, we see that $\beta \sim N(0, \infty)$ is equivalent to $\mathbb{P}(\beta_0) \propto 1$. Moreover, we see that the Y_i are conditional distributed as follows

$$\begin{aligned}
Y_i|\beta_0 &\overset{ind}{\sim} \text{Bern}[\text{logit}^{-1}(\beta_0 + X\beta^{(t)})] \\
\mathbb{P}(\beta_0) &\propto 1
\end{aligned}$$

This model is exactly the (t+1) Gibbs step for sampling $\beta_0^{(t+1)}$. Hence we can use the step_laplace_mh routine to sample $\beta_0^{(t)}$. Following a similar argument for $\beta_j^{(t+1)}$, we have

$$\begin{aligned}
\beta_j^{(t+1)} = \text{step-laplace-mh}(\beta_j^{(t)}; Y, X_j, 1_n\beta_0^{(t+1)} + X_{1:(j-1)}\beta_{1:(j-1)}^{t+1} \\
+ X_{(j+1):p}\beta_{(j+1):p}^{t}, 0, \frac{1-\theta_j^{(t+1)}}{\tau_0^2} + \frac{\theta_j^{(t+1)}}{\tau^2})
\end{aligned}$$

corresponds to the model

$$Y_i|\beta \overset{ind}{\sim} \text{Bern}\left[\text{logit}^{-1}\left((X_j)_i\beta + 1_n\beta_0^{(t+1)} + X_{1:(j-1)}\beta_{1:(j-1)}^{t+1} + X_{(j+1):p}\beta_{(j+1):p}^{t}\right)\right]$$
$$\beta \sim N\left(0, \left(\frac{1-\theta_j^{(t+1)}}{\tau_0^2} + \frac{\theta_j^{(t+1)}}{\tau^2}\right)^{-1}\right)$$

Recall that in the Gibbs framework, at this step in the algorithm we condition on $\theta^{(t+1)}$, $\beta_{1:(j-1)}^{(t+1)}$, $\beta_{(j+1):p}^{(t)}$ and try to estimate $\beta_j^{(t+1)}$. Well, taking $\beta = \beta_j$, we see that as $\theta_j^t = 0, 1$ that $\beta_j|\theta_j^{(t+1)} = 1 \sim N(0, \tau^2)$ and $\beta_j|\theta_j^{(t+1)} = 1 \sim N(0, \tau_0^2)$. Notice this is the exact prior we specified above for $\beta_j|\theta_j$. As we are in the Gibbs framework, we implicitly condition on θ_j and hence these are the same prior distributions. Moreover, we see that the Y_i are conditional distributed as follows

$$Y_i|\beta \overset{ind}{\sim} \text{Bern}[\text{logit}^{-1}((1_n^T)_i\beta_0^{(t+1)} + X_{1:(j-1)}\beta_{1:(j-1)}^T + X_j\beta_j + X_{(j+1):p}\beta_{(j+1):p}^{(t-1)})]$$
$$\beta_j \sim N\left(0, \left(\frac{1-\theta_j^{(t+1)}}{\tau_0^2} + \frac{\theta_j^{(t+1)}}{\tau^2}\right)^{-1}\right)$$

Notice here that the $1_n^T\beta_0^{(t+1)}$ is the common intercept from the previous estimation step and the $X_{1:(j-1)}\beta_{1:(j-1)}^{(t+1)}$ is the covariate term corresponding to the updated β terms. Lastly, $X_{(j+1):p}\beta_{(j+1):p}^{(t)}$ is the terms yet be updated, and $X_j\beta_j$ is the term currently being updated. Here, seeing that our model falls within the step_laplace_mh framework, we can use this function of draw our next $\beta_j^{(t+1)}$ sample.

(c)

Before we write out MCMC sampler, we first introduce some helpful functions used in our implementation.

```
logit <- function (p) log(p / (1 - p))
inv_logit <- function (x) 1 / (1 + exp(-x))
LOGEPS <- log(.Machine$double.eps / 2)
log1pe <- function (x) {
  l <- ifelse(x > 0, x, 0)
  x <- ifelse(x > 0, -x, x)
  ifelse(x < LOGEPS, l, l + log1p(exp(x)))
}
log_posterior <- function (beta, y, x, offset, beta0 = 0, omega = 0) {
  eta <- offset + x * beta
  sum(y * eta - log1pe(eta)) - .5 * omega * (beta - beta0) ^ 2
}
step_laplace_mh <- function (beta, y, x, offset, beta0 = 0, omega = 0) {
  mu <- inv_logit(offset + x * beta)
  s <- sqrt(1 / (sum(mu * (1 - mu) * x ^ 2) + omega))
  betac <- rnorm(1, beta, s)
  muc <- inv_logit(offset + x * betac)
  sc <- sqrt(1 / (sum(muc * (1 - muc) * x ^ 2) + omega))
  log_R <- log_posterior(betac, y, x, offset, beta0, omega) +
    dnorm(beta, betac, sc, log = TRUE) -
    (log_posterior(beta, y, x, offset, beta0, omega) +
       dnorm(betac, beta, s, log = TRUE))
  samp <- ifelse(log_R >= 0 || log(runif(1)) < log_R, betac, beta)
```

```
  return(list(samp, log_posterior(betac, y, x, offset, beta0, omega)))
}

A <- matrix(0, nrow = ncol(hla_study) - 1,
                      ncol = ncol(hla_study) - 1)
for (i in 1:11) {
  for (j in 1:11) {
    A[i,j] <- ifelse(abs(hla_snps[i, "position"] - hla_snps[j, "position"]) < 50000, 1, 0)
  }
  A[i, i] <- 0
}

X <- hla_study[, -1]
Y <- hla_study[, 1]
```

Next, we write our sampler.

```
fit_mcmc <- function(X, Y, tau0 = 1e-4, tau = 1, ns = 1000, h = -100, burnin = 0, thin = 1) {

  # Initialize
  theta <- matrix(nrow = ncol(X), ncol = ns); theta[, 1] <- 1 #rbinom(11, 1, .5)
  beta0 <- numeric(ns); beta0[1] <- 0
  beta <- matrix(nrow = ncol(X), ncol = ns); beta[, 1] <- 0
  logPost <- numeric(ns)

  for (t in 2:ns) {

    #set up local variables
    n <- length(Y)
    p <- nrow(beta)
    C <- 100


    #sample theta
    sigma2 <- function(theta) ((1 - theta) * tau0) + (theta * tau)
    theta[, t] <- theta[,t-1] # copy for better readabilitiy
    for(j in 1:p){

          #calculate bernoulli probs
          logp0 <- -.5 * log(sigma2(0)) - beta[j, t-1]^2/(2 * sigma2(0))
          logp1 <- -.5 * log(sigma2(1)) - beta[j, t-1]^2/(2 * sigma2(1)) + h + (2/C * sum(A[j,] * (2*th
          logp <- logp1 - lse(c(logp0, logp1))

          #update log Post
          logPost[t] <- logp


          #sample bernoulli prob
          theta[j, t] <- ifelse(log(runif(1)) < logp, 1, 0)
    }

        #sample beta0
    tmp <- step_laplace_mh(beta0[t-1], Y, rep(1, n), as.matrix(X) %*% as.matrix(beta[,t-1]), 0, 0)
    beta0[t] <- tmp[[1]]
```

```
    logPost[t] <- logPost[t] + tmp[[2]] #update log P

    #sample betas
    beta[,t] <- beta[,t-1] #copy over for better readability
    for(j in 1:p){
      tmp <- step_laplace_mh(beta[j,t-1], Y, X[,j], rep(beta0[t], n) + as.matrix(X[, -j]) %*% as.matrix
                                  0, (1 - theta[j,t])/tau0 + theta[j,t]/tau)
      beta[j,t] <- tmp[[1]]
      logPost[t] <- logPost[t] + tmp[[2]] #update log P
    }
  }

  return(list(theta = theta[,burnin:ncol(theta)][,seq(1, ncol(theta) - burnin , thin)],
              beta0 = beta0[burnin:length(beta0)][seq(1, length(beta0) - burnin, thin)],
              beta = beta[,burnin:ncol(beta)][,seq(1, ncol(beta) - burnin , thin)],
              lpost = logPost
              ))
}

get_beta0 <- function(X,Y){ fit <- fit_mcmc(X,Y); matrix(fit$beta0, ncol = 1)}
get_betas <- function(X,Y){ fit <- fit_mcmc(X,Y); t(fit$beta)}
get_thetas <- function(X,Y){ fit <- fit_mcmc(X,Y); t(fit$theta)}
```

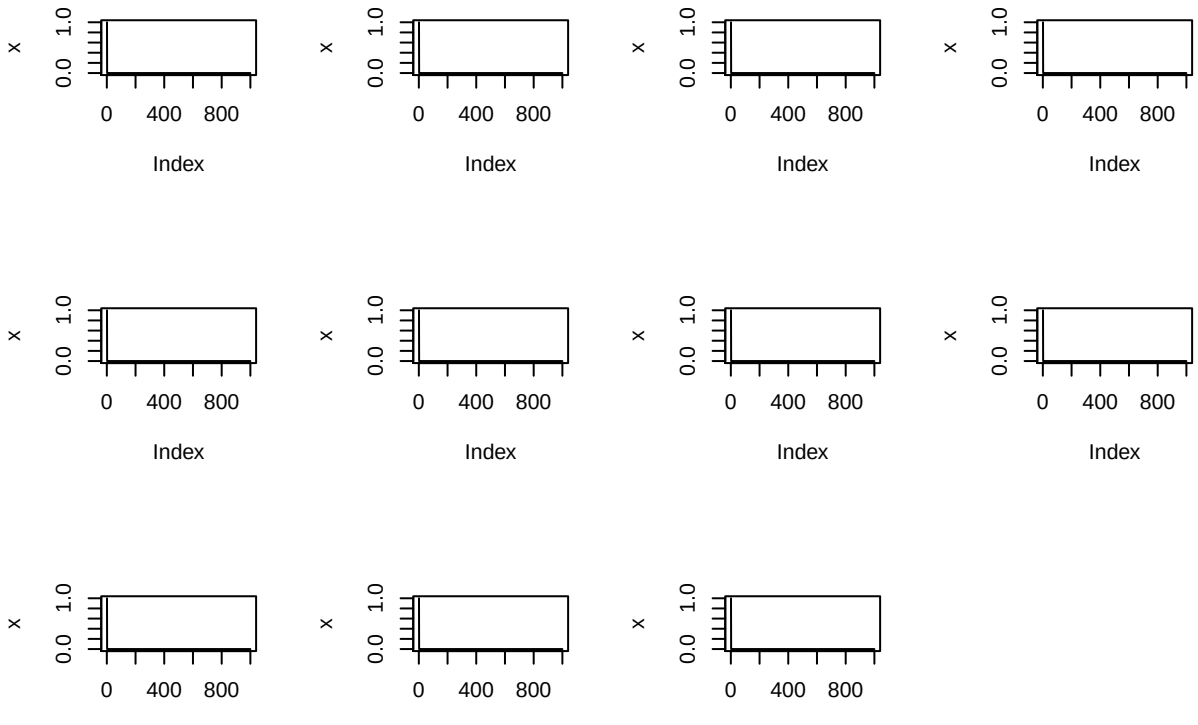
Having implemented our sampler, we now look to assess the convergence of our chain. First we look at which markers should be included in our model. That is we look at the θ samples.

```
set.seed(1985)
thetas <- get_thetas(X,Y)
probs <- apply(thetas, 2, function(x) sum(x)/length(x))
knitr::kable(data.frame(probs))
```

probs
0.001
0.001
0.001
0.001
0.001
0.001
0.001
0.001
0.001
0.001
0.001

```
#par(mfrow = c(3, 4)); apply(thetas, 2, hist)
par(mfrow = c(3, 4)); apply(thetas, 2, function(x) plot(x, type = "l"))
```

```
## NULL
```



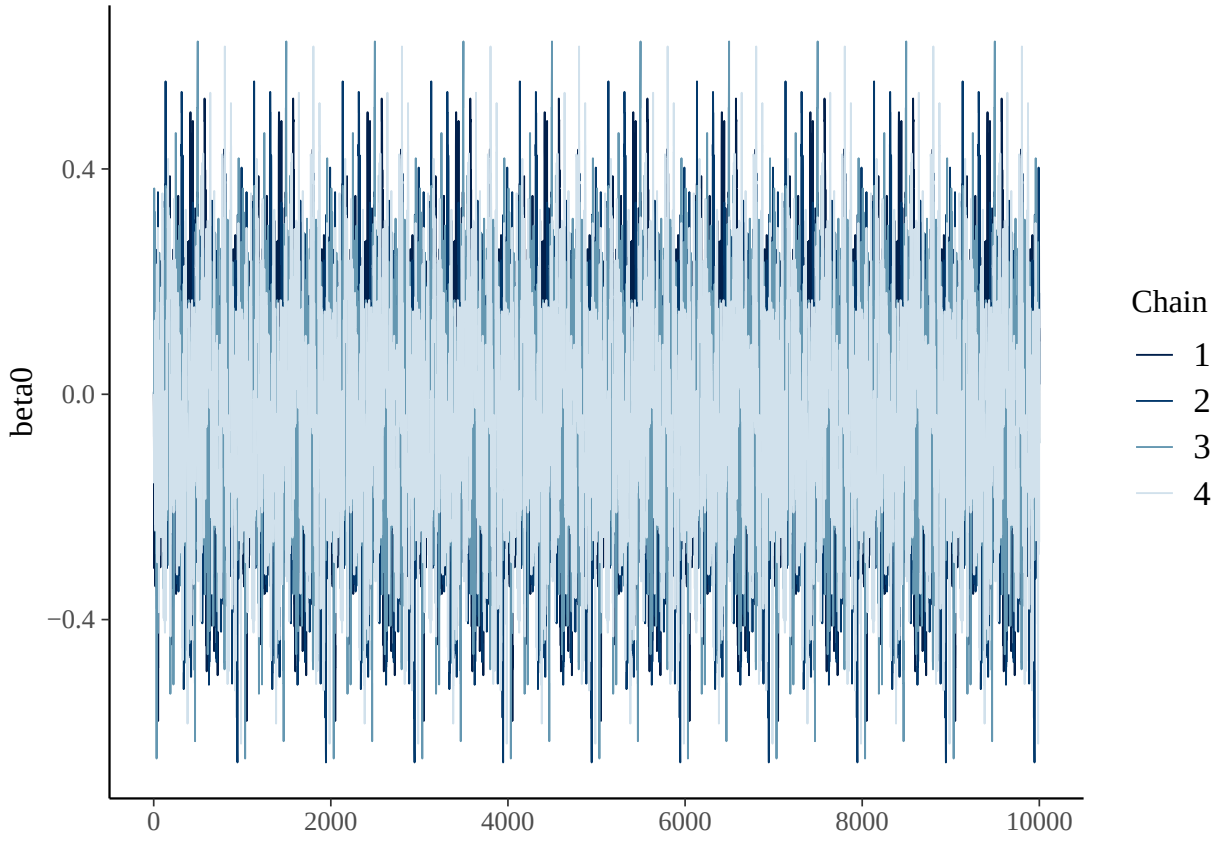
Here we see that the MCMC algorithm is simply setting all θ_j values to zero after a single gibbs iteration. This could be due to the $h = -100$ value. In either case, this MCMC algorithm is suggesting that *none* of the SNPs considered in this model effect the disease status Y_i. Regardless of this fact, we investigate the β and β_0 samples below. We begin with β_0.

```
#visual results using STAN
library(bayesplot)
```

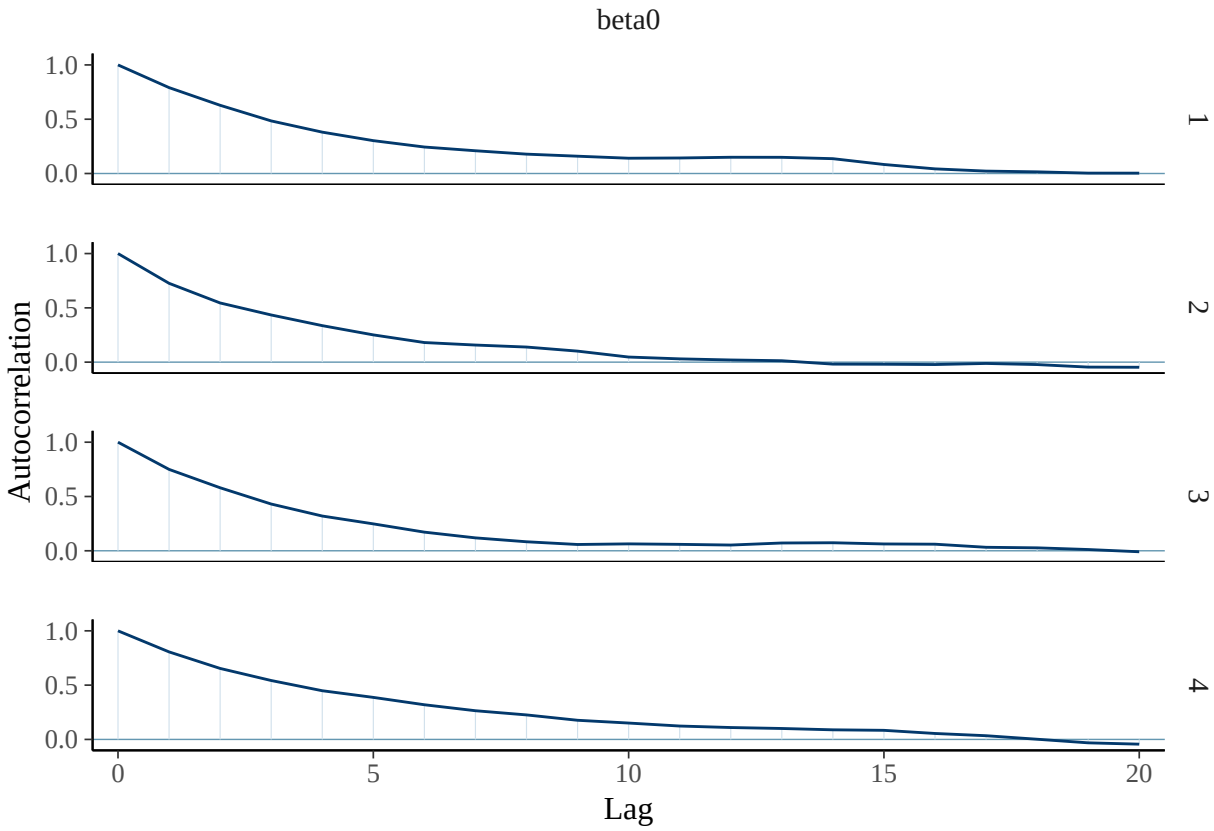
```
## This is bayesplot version 1.6.0
## - Online documentation and vignettes at mc-stan.org/bayesplot
## - bayesplot theme set to bayesplot::theme_default()
##    * Does _not_ affect other ggplot2 plots
##    * See ?bayesplot_theme_set for details on theme setting
##
## Attaching package: 'bayesplot'
## The following object is masked _by_ '.GlobalEnv':
##
##     log_posterior
```

```
mcmc_array <- function (ns, nchains, params) {
  nparams <- length(params)
  array(dim = c(ns, nchains, nparams),
        dimnames = list(iterations = NULL,
                        chains = paste0("chain", 1:nchains),
                        parameters = params))
}
set.seed(1985)
ns <- 10000
nchains <- 4
```

```
sims <- mcmc_array(ns, nchains, "beta0")
for (ic in 1:nchains)
  sims[, ic, ] <- get_beta0(X,Y)

rhat <- function (sims, ...)
  rstan::monitor(sims, print = FALSE, ...)[, "Rhat"]
neff_ratio <- function (sims, ...)
  rstan::monitor(sims, print = FALSE, ...)[, "n_eff"]

mcmc_trace(sims)
```




```
mcmc_acf(sims)
```



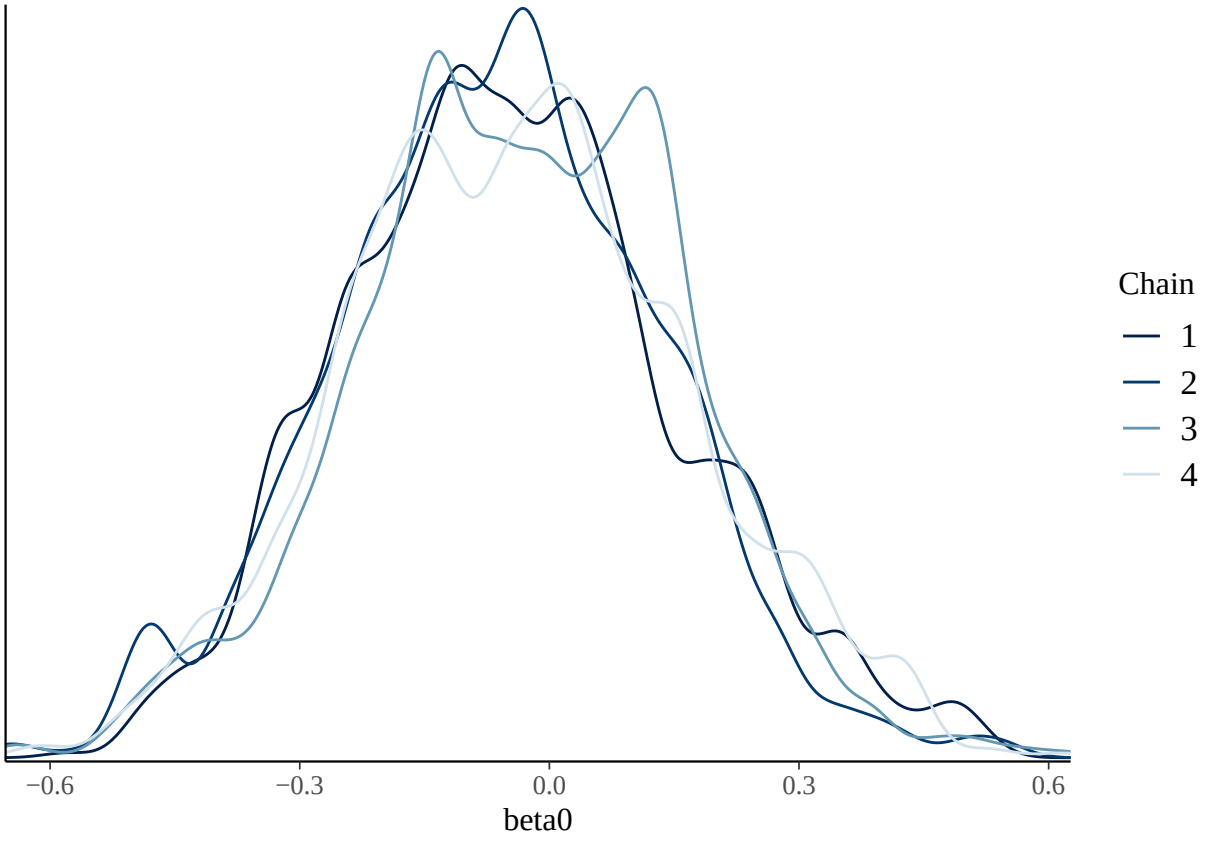

```
rhat(sims)
```

```
## [1] 1.002339
```

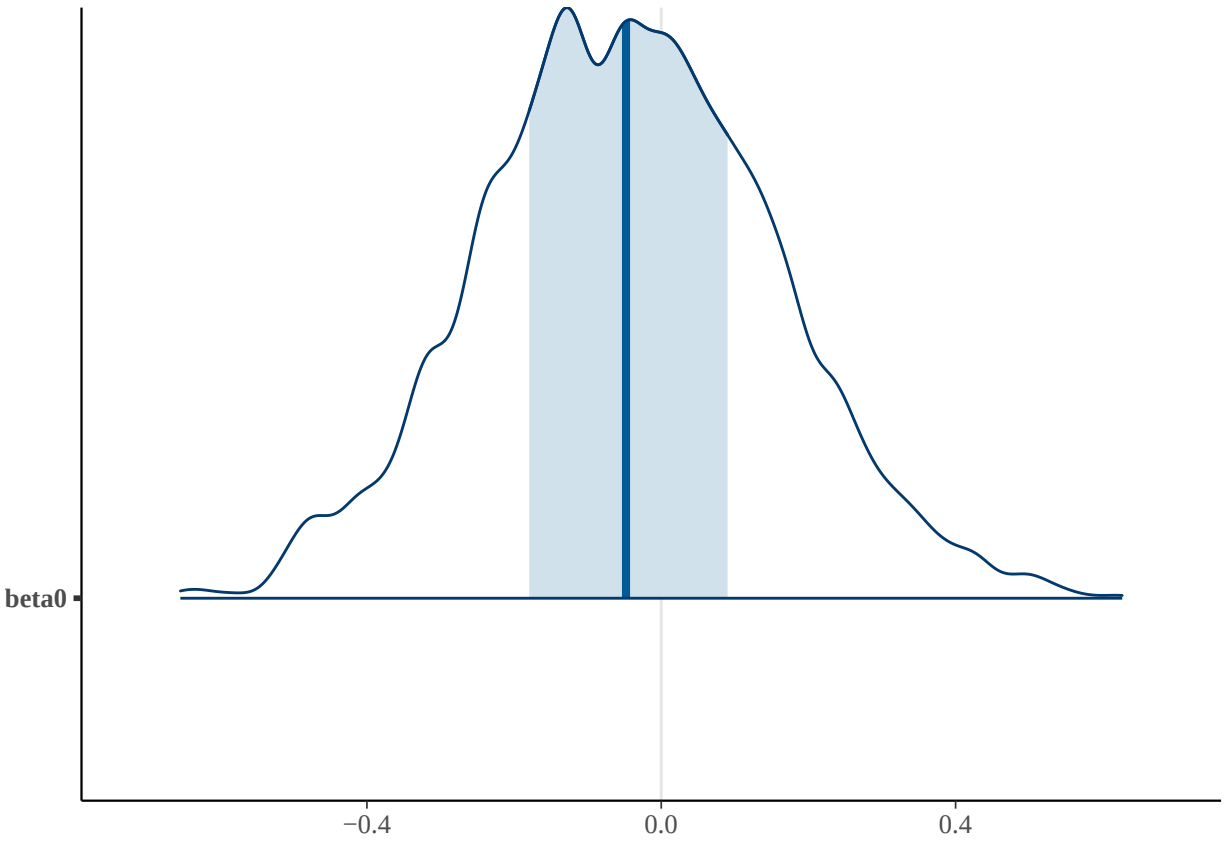
```
neff_ratio(sims)
```

```
## [1] 2301.923
```

```
mcmc_dens_overlay(sims)
```

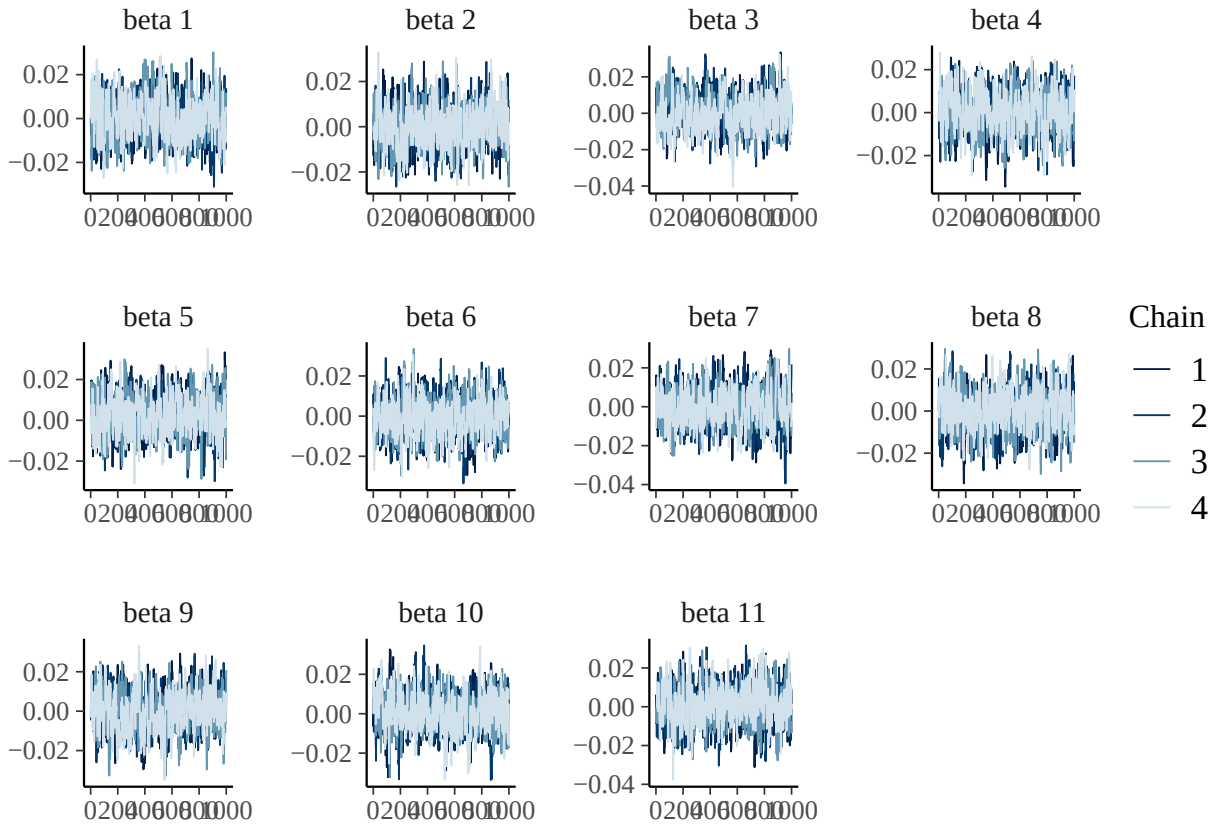



```
mcmc_areas(sims)
```

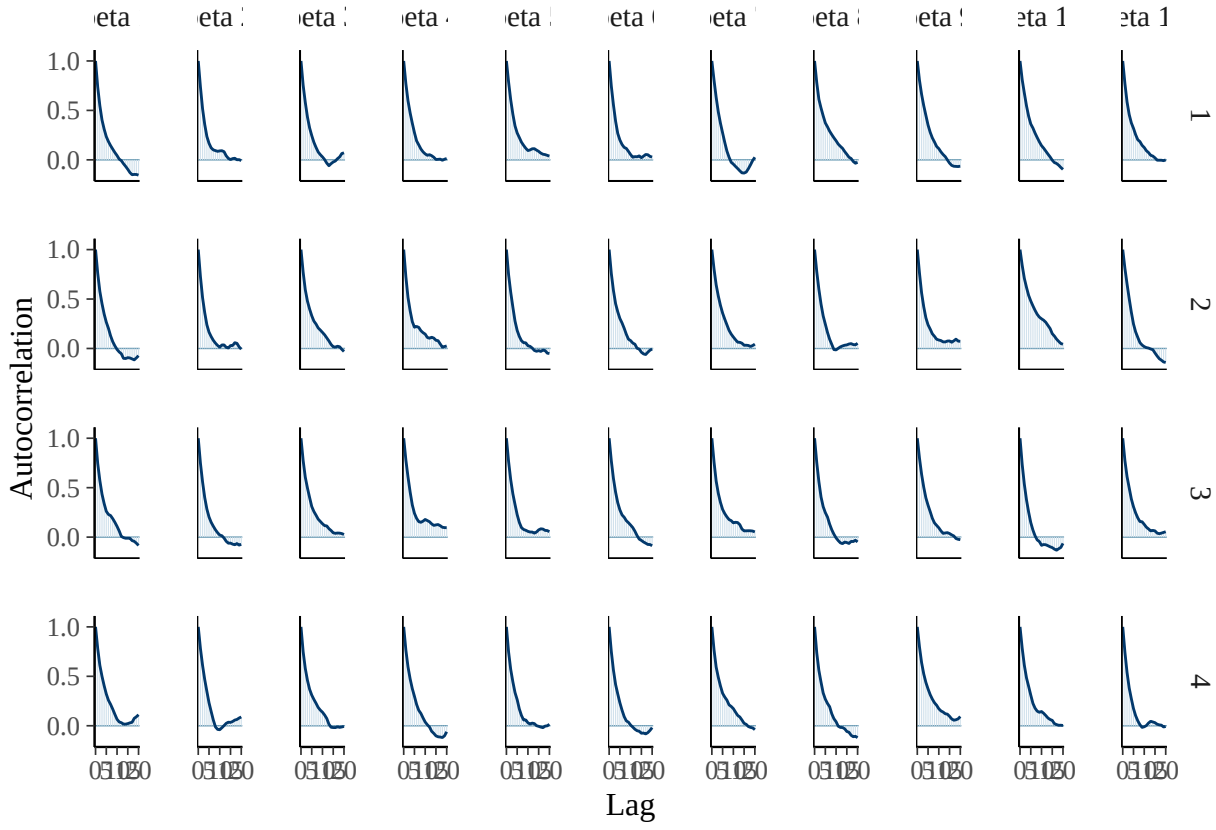


First we see that all four Markov chains are mixing quite well as apparent by the first visualization. There appears to be few rejections as all chains appear to be exploring the space relatively well. The next plot shows the ACF plots of the β_0 parameter. As evident by this plot, there is *significant* correlation in these estimates. That is, each ACF plot decays quite slowly suggesting a nontrivial correlative behavior in the estimates. To overcome these shortcomings, we could thin the chain estimates to reduce this variance. Next we calculate the scale reduction factor and the effective sample size. Here we see that $\widehat{R} = 1.013253$ which is less than the 1.1 rule of thumb. This suggests that there is little between chain variance which implies that these estimates are reliable. Moreover, the effective sample size is 228.8465. Note that $\frac{228.8465}{1000} = 0.2288465$ which is below the rule of thumb .5. This suggests again there is significant correlation in these estimates and should be thinned. Lastly, we plot the distribution of estimates across the four chains. Here we see that they all relatively agree and that the the distribution is centered around zero. Next we turn our attention to the beta coefficients.

```
#visual results using STAN
set.seed(1985)
ns <- 1000
nchains <- 4
sims <- mcmc_array(ns, nchains, c(sapply(1:11, function(x) paste("beta", x))))
for (ic in 1:nchains)
  sims[, ic, ] <- get_betas(X,Y)


rhat <- function (sims, ...)
  rstan::monitor(sims, print = FALSE, ...)[, "Rhat"]
neff_ratio <- function (sims, ...)
  rstan::monitor(sims, print = FALSE, ...)[, "n_eff"]

mcmc_trace(sims)
```




```
mcmc_acf(sims)
```



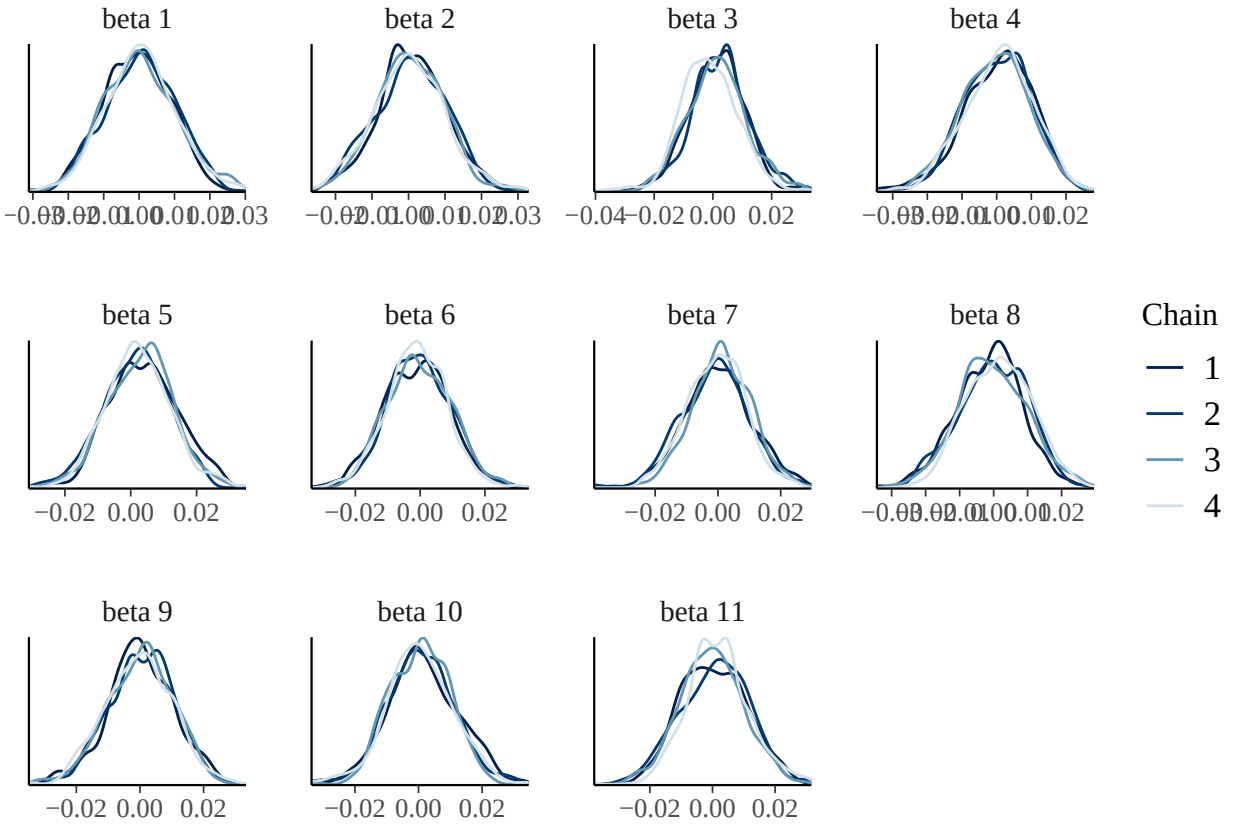
```
rhat(sims)
```

```
##   beta 1   beta 2   beta 3   beta 4   beta 5   beta 6   beta 7   beta 8
## 1.002399 1.015864 1.025306 1.013525 1.023976 1.019039 1.028973 1.011202
##   beta 9  beta 10  beta 11
## 1.011469 1.016922 1.012871
```

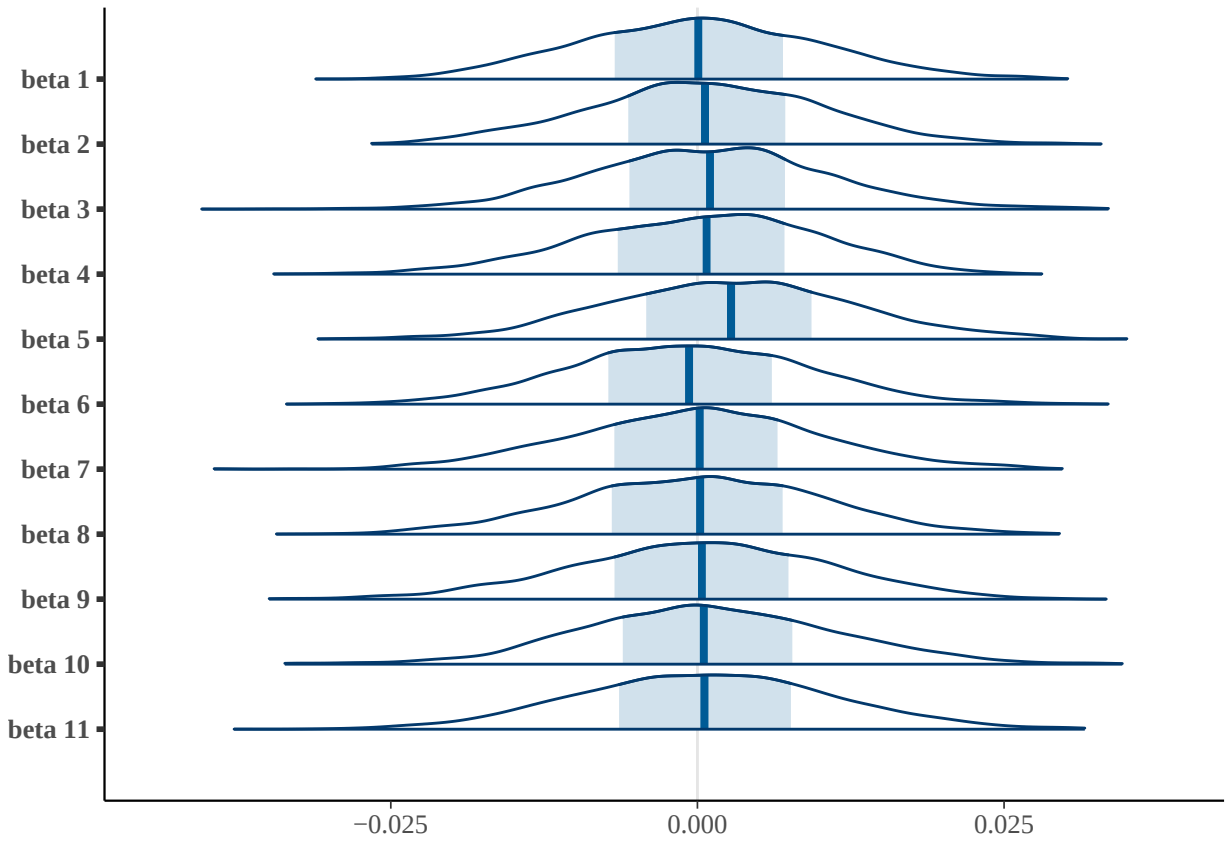
```
neff_ratio(sims)
```

```
##   beta 1   beta 2   beta 3   beta 4   beta 5   beta 6   beta 7   beta 8
## 277.0511 340.5130 225.2310 250.0605 236.5313 245.9466 202.9673 308.9607
##   beta 9  beta 10  beta 11
## 233.7993 229.9217 239.2705
```

```
mcmc_dens_overlay(sims)
```




```
mcmc_areas(sims)
```



Here we see a similar behavior as above for β_0. First we see that all four Markov chains are mixing quite well as apparent by the first visualization. There appears to be few rejections as all chains appear to be exploring the space relatively well. The next plot shows the ACF plots of the β parameters. While difficult to see initially, As evident by this plot, there is *significant* correlation in these estimates. That is, each ACF plot decays quite slowly suggesting a nontrivial correlative behavior in the estimates. To overcome these shortcomings, we could thin the chain estimates to reduce this variance. Next we calculate the scale reduction factor and the effective sample size. Here we see that $\max \widehat{R} = 1.028973$ which is less than the 1.1 rule of thumb. This suggests that there is little between chain variance which implies that these estimates are reliable. Moreover, the effective sample size is in the range $(225.2310, 340.5130)$, Note that even for the largest coefficient $\frac{340.5130}{1000} = 0.340513$ which is below the rule of thumb .5. This suggest again there is significant correlation in these estimates and should be thinned. Lastly, we plot the distribution of estimates across the four chains. Here we see that they all relatively agree and that the the distribution is centered around zero with a very small variance.

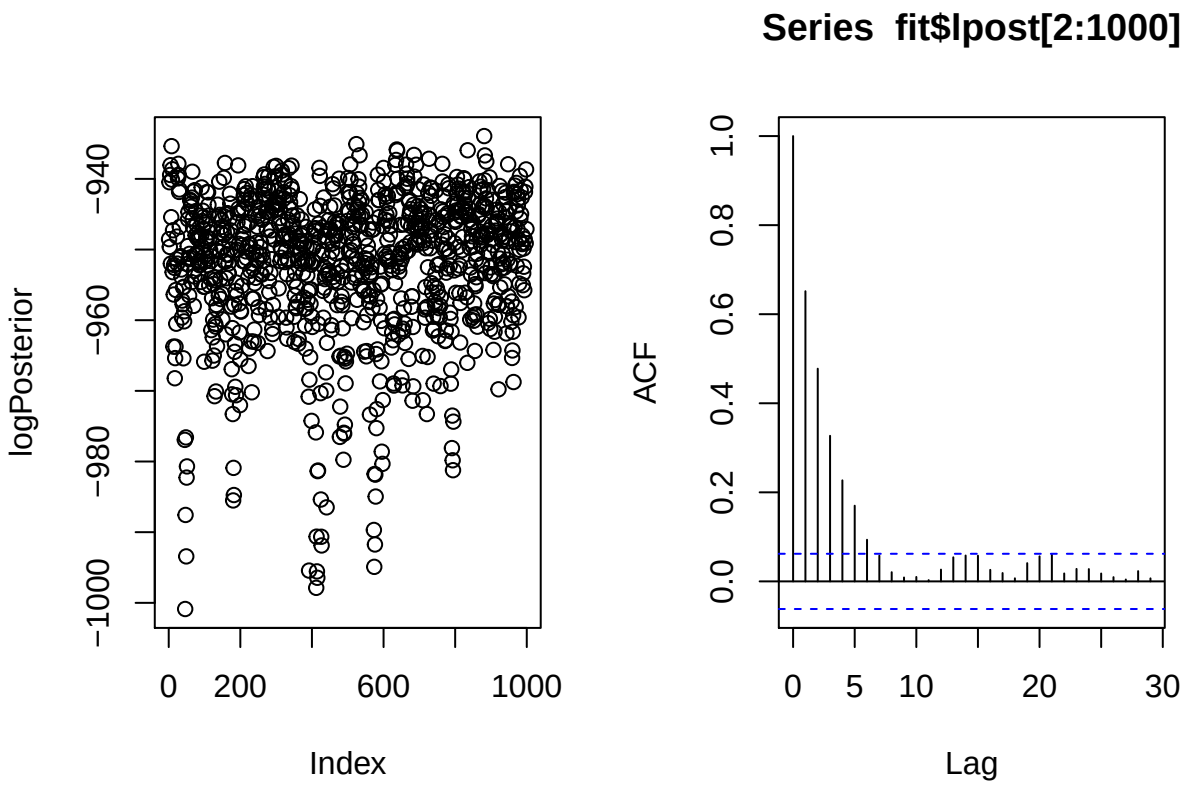
Lastly we look to understand the convergence of the log posterior $\log \mathbb{P}(\theta, \beta, \beta_0|Y)$. First consider the following decomposition

$$\begin{aligned}\log \mathbb{P}(\theta, \beta, \beta_0|Y) &\propto \log \mathbb{P}(\beta_0|Y) + \log \mathbb{P}(\beta, \theta|Y) \\ &\propto \log \mathbb{P}(\beta_0|Y) + \log \mathbb{P}(\beta|\theta, Y) + \log \mathbb{P}(\theta|Y)\end{aligned}$$

Here we see that this is approximately equivalent to summing the posterior probabilities of each proposal step in the Gibbs model. We plot the trace plot and acf plot of these probabilities below.

```
set.seed(1985)
fit <- fit_mcmc(X,Y)
par(mfrow = c(1,2))
```

```
plot(fit$lpost[2:1000], ylab = "logPosterior")
acf(fit$lpost[2:1000])
```



Here again we see that sampler converges rather quickly to a solution as evident by the $\log \mathbb{P}(\beta_0, \beta, \theta|Y)$. Again we see a long-term correlation of the estimates. Therefore thinning will be appropriate.

Additional remarks

In this model, we see that all parameters get pushed to zero almost immediately. We believe that this is due to the $h = -100$ value. Here, we complete a sensitivity analysis in terms of h. Below, we plot the average θ estimate as a function of h. Here we see that for $h < -5$, all estimates are 0. For $h > -5$, we see that some values are zeroed while others are not. This is simply an additional remark to suggest that a more appropriate choice of h would lead to more insightful inference.

```
library(ggplot2)

set.seed(1985)
h <- -1:-11
theta_mean <- matrix(NA, ncol = 11, nrow = length(h))
for(i in 1:11){
  fit <- fit_mcmc(X,Y, h = h[i])
  theta_mean[i,] <- apply(fit$theta[, 200:1000], 1, mean)
}

colnames(theta_mean) <- sapply(1:11, function(x) paste0("theta",x))
df <- data.frame(cbind(h, theta_mean))
df2 <- tidyr::gather(df, theta, P, 2:12)
df2$theta <- factor(df2$theta, levels = colnames(theta_mean))
```

```
p1 <- ggplot(df2, aes(x = h, y = P, group = theta))+
  geom_line(aes(col = theta))+
  theme_bw()
p1
```