Biostat 276 Project 1

Kelly Li

1/29/2022

All code for this homework project is available at https://github.com/k-m-li/biostat-276-2022-fall

Sampling from the Banana Distribution

a)

First we compute the marginal distribution of x_1 and the conditional distribution of $x_2|x_1$:

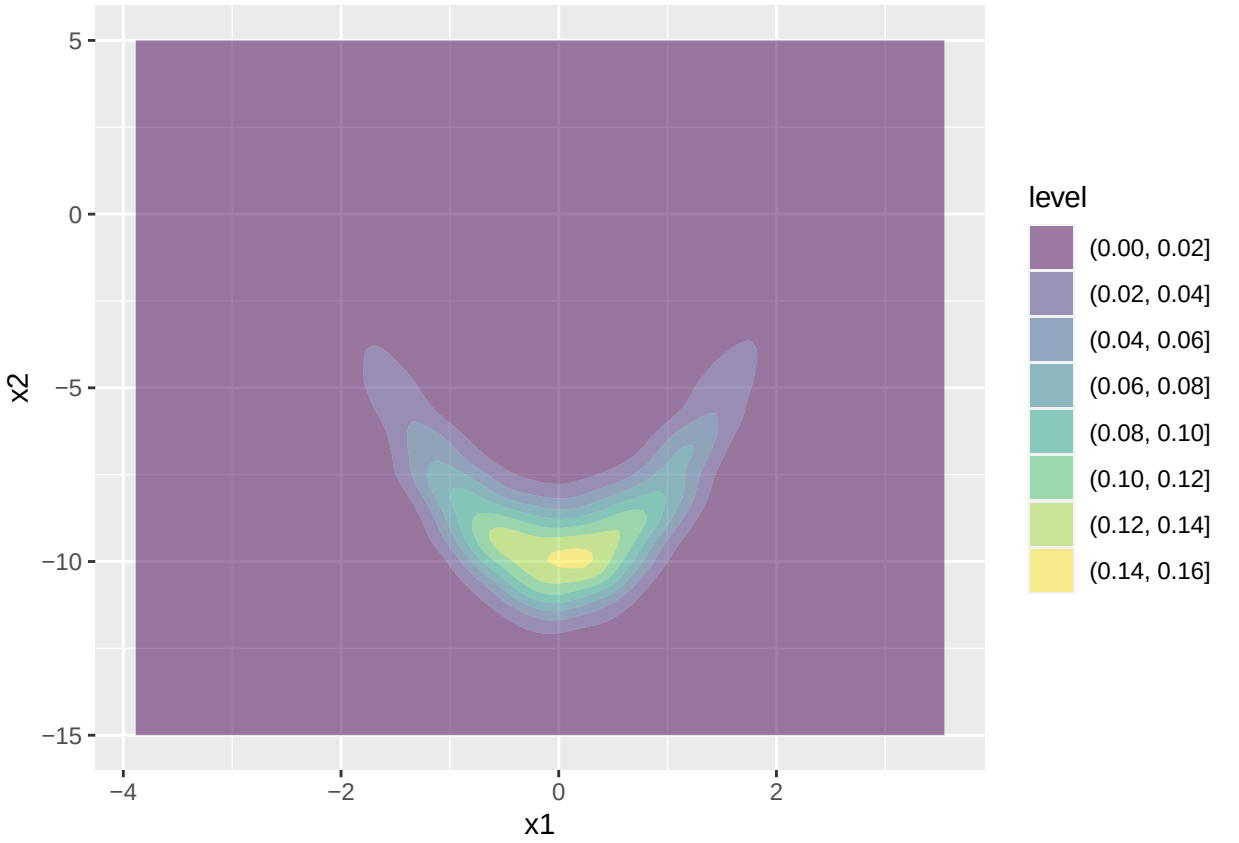
$$\begin{aligned} p(x_1) &\propto \int p(x1, x2)dx_2 \\ &\propto \exp[-\frac{x_1^2}{2}] \int \exp[-\frac{(x_2 - 2(x_1^2 - 5))^2}{2}]dx_2 \\ &\propto \exp[-\frac{x_1^2}{2}] \\ &= N(0, 1) \end{aligned}$$

$$\begin{aligned} p(x_2|x_1) &\propto exp[-\frac{(x_2 - 2(x_1^2 - 5))^2}{2}] \\ &= N(2(x_1^2 - 5), 1) \end{aligned}$$

We can next generate independent $u_1, u_2 \sim U(0, 1)$ and use the inverse of the cdf to generate our target distribution.

```
bananainv <- function(n){
  x1 <- rnorm(n, 0, 1)
  x2 <- sapply(x1, function(x){rnorm(1, 2*x^2-10, 1)})
  return(cbind(x1, x2))
}

df <- as.data.frame(bananainv(10000))

ggplot(df, aes(x = x1, y = x2)) +
  geom_density_2d_filled(alpha = 0.5) +
  ylim(-15, 5)
```

```
## Warning: Removed 58 rows containing non-finite values (stat_density2d_filled).
```

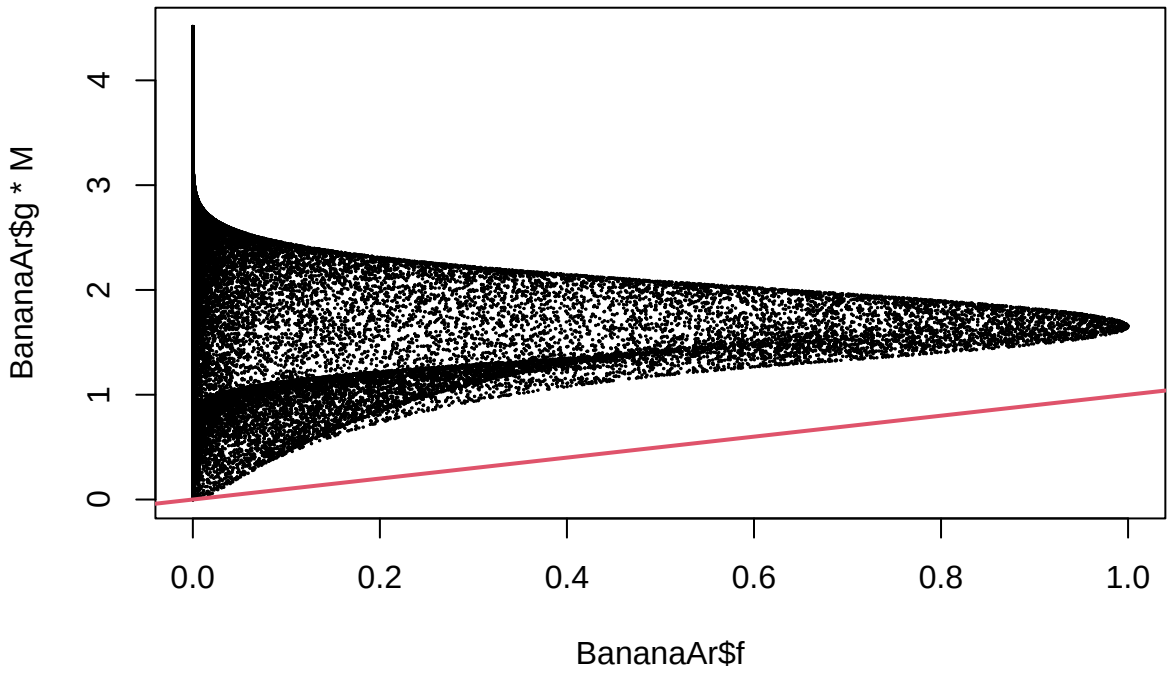


b) The pdf of the banana distribution is log-concave. We can guess that we can use two halves of a multivariate normal distribution centered about 0, which we can guess and form a heavier tail than the banana distribution to be used to implement accept-reject sampling.

AR sampling is implemented below:

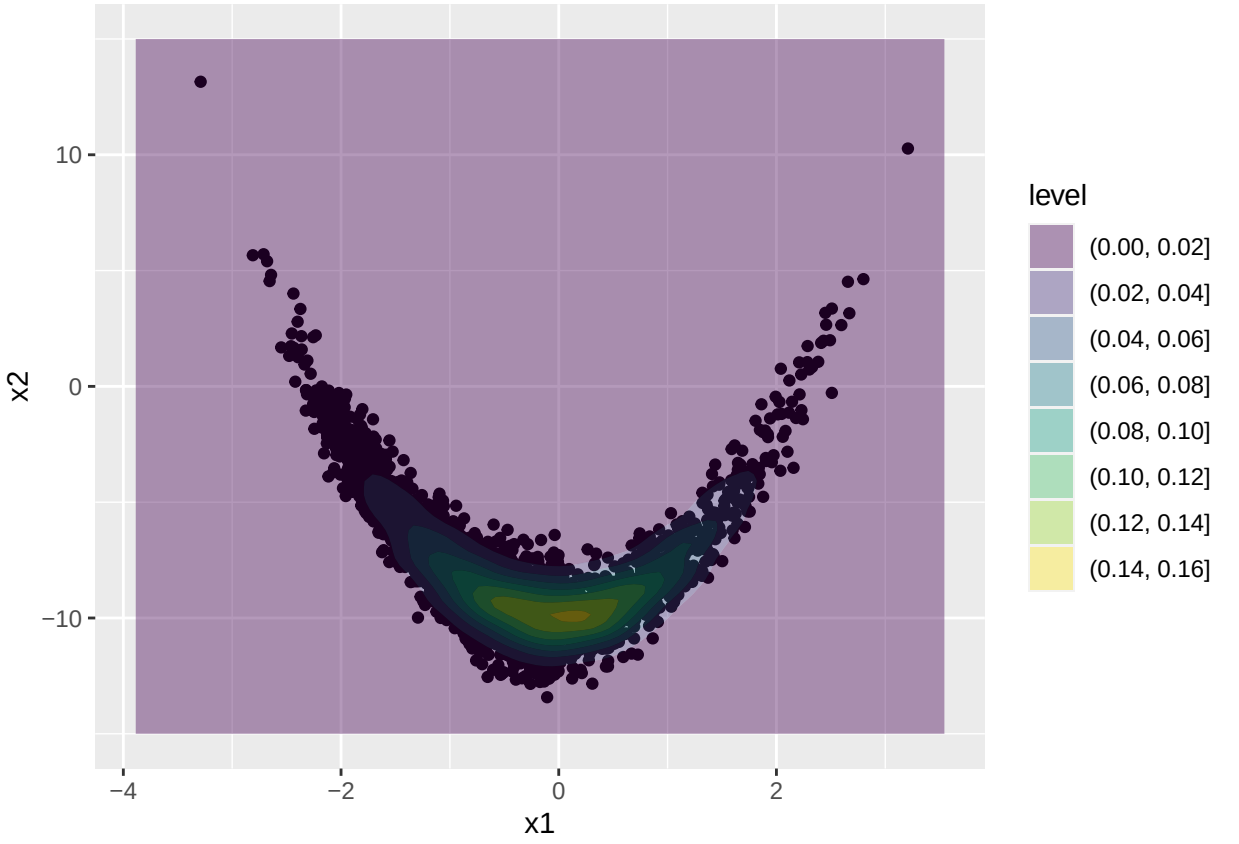
```
bananap <- function(x1, x2){
  return(exp(-x1^2/2)*exp(-(x2-2*(x1^2-5))^2/2))
}

getBananaAr <- function(nsim, sigma, M){
  x_est <- rmvnorm(n = nsim, sigma = sigma)
  x_est_neg <- -abs(x_est)
  x_est <- rbind(x_est, x_est_neg)
  nsim <- nrow(x_est)

  p <- dmvnorm(x_est, sigma = sigma)
  pban <- rep(NA, nsim)

  for(i in 1:nsim){
    x1 <- x_est[i, 1]
    x2 <- x_est[i, 2]
    pban[i] <- bananap(x1, x2)
  }
  u <- runif(nsim)
```

```
  c <- pban/p * 1/M

  AR <- x_est[c >= u,] %>% as.data.frame()
  colnames(AR) <- c("x1", "x2")

  isProductGood <- ifelse(length(which(pban / (p * M) > 1)) == 0, "yes", "no")
  eff <- nrow(AR)[1]/nsim

  return(list(AR = AR, isProductGood = isProductGood, efficiency = eff, f = pban, g = p, x_est = x_est))
}

nsim <- 50000
sigma <- matrix(c(1, 0.5, 0.5, 50), 2)
M <- 200

BananaAr <- getBananaAr(nsim, sigma, M)
AR <- BananaAr$AR

plot(BananaAr$f, BananaAr$g * M, cex = 0.1)
abline(a = 0, b = 1, col = 2, lwd = 2)
```




```
ggplot() +
  ylim(-15, 15) +
  geom_point(data = AR, mapping = aes(x = x1, y = x2)) +
  geom_density_2d_filled(mapping = aes(x = x1, y = x2), data = df, alpha = 0.4)
```

```
## Warning: Removed 3 rows containing non-finite values (stat_density2d_filled).
```




```
BananaAr$efficiency
```

```
## [1] 0.04365
```

```
BananaAr$isProductGood
```

```
## [1] "yes"
```

The simulated plot indicates the true banana distribution and the points generated from my AR algorithm. The model has around a 4.5% acceptance rate and satisfies the condition $\frac{f}{g*M} \leq 1$. This model in relation to the importance distribution can be improved by choosing a closer distribution - a more exact mixture of normals, or some different similarly-shaped exponential distribution, may generate a better envelope than the crude one discussed above. Since the banana distribution is very uniquely shaped, the basic AR sampling implemented above can exhibit quite low efficiency.

c)

For importance-rejection sampling, we can consider the same proposed distributions as in (b), and we implement standardized weights from (b) by which to accept or reject certain values to obtain the mixture.

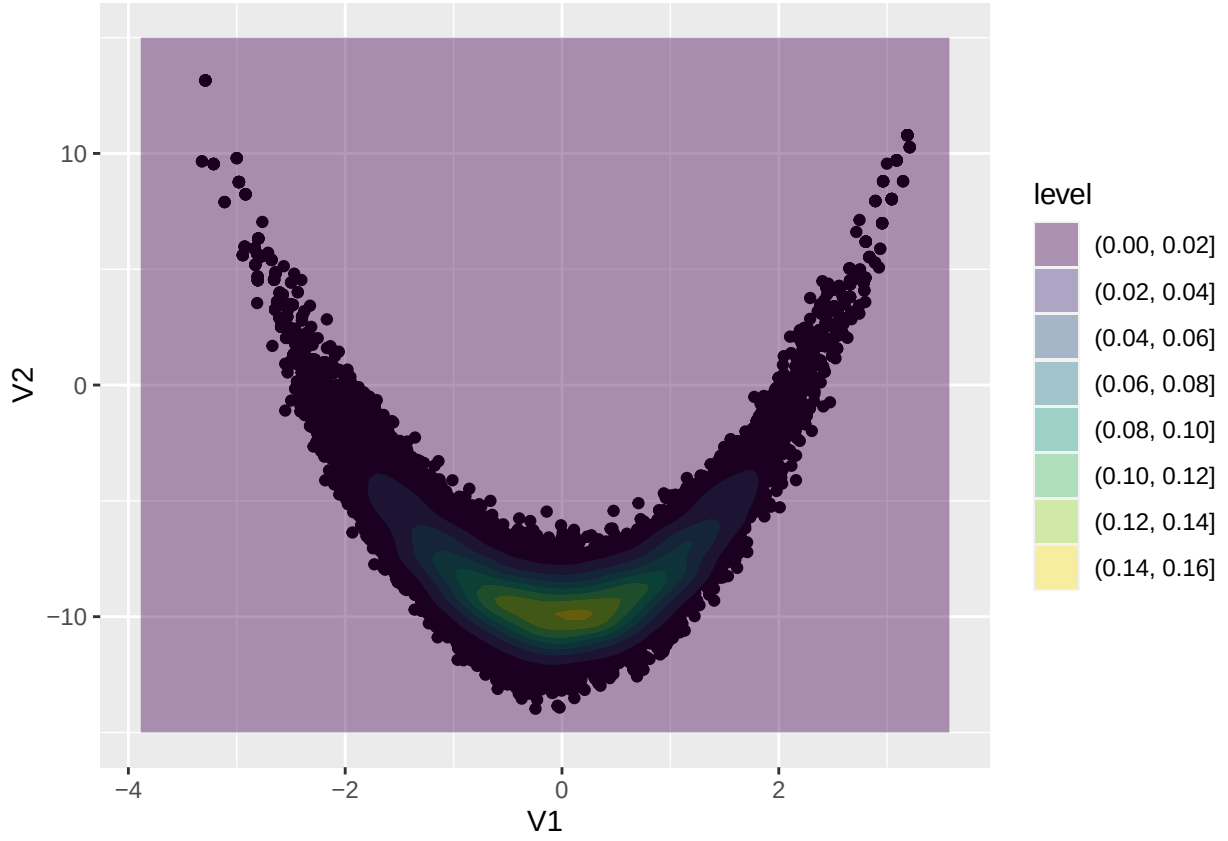
```r
w <- BananaAr$f/BananaAr$g
w <- w/sum(w)

x <- seq(1, 2*nsim, by = 1)
sample <- sample(x, prob = w, replace = TRUE)

BananaIS <- BananaAr$x_est[sample, ] %>% as.data.frame()
eff <- nrow(unique(BananaIS))/(nsim * 2)

eff
```

```
## [1] 0.16138
```

```r
ggplot() +
  ylim(-15, 15) +
  geom_point(data = BananaIS, mapping = aes(x = V1, y = V2)) +
  geom_density_2d_filled(mapping = aes(x = x1, y = x2), data = df, alpha = 0.4)
```

```
## Warning: Removed 3 rows containing non-finite values (stat_density2d_filled).
```

```
## Warning: Removed 2 rows containing missing values (geom_point).
```



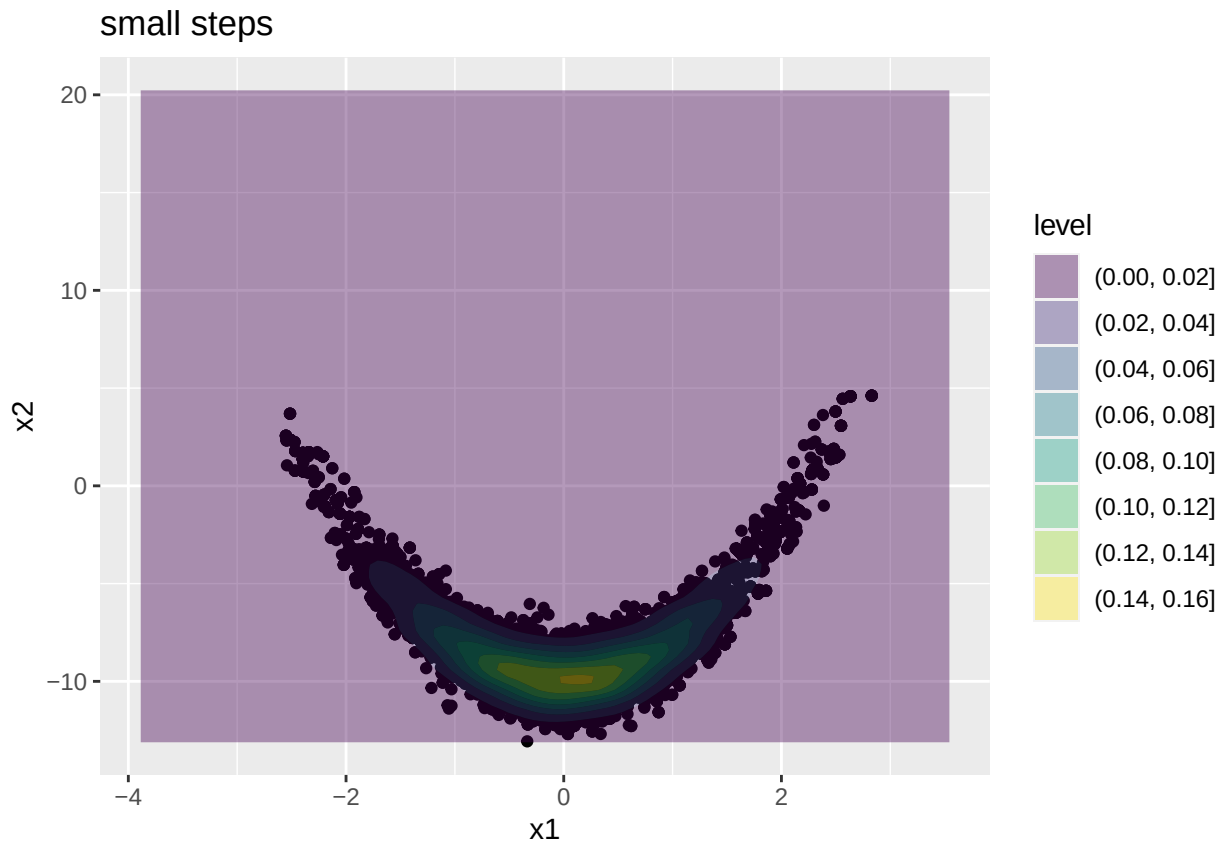
We implement the same importance distribution as from (b). For IS, we have approximately a 16% efficiency, which is already better than that from our original AR sampling. Much like in AR sampling, IS is very

dependent on the importance distribution being similarly shaped to the target distribution. In that case, the probability of the resampling step will yield a better mixture of values such that unique value count will be higher and efficiency improved.

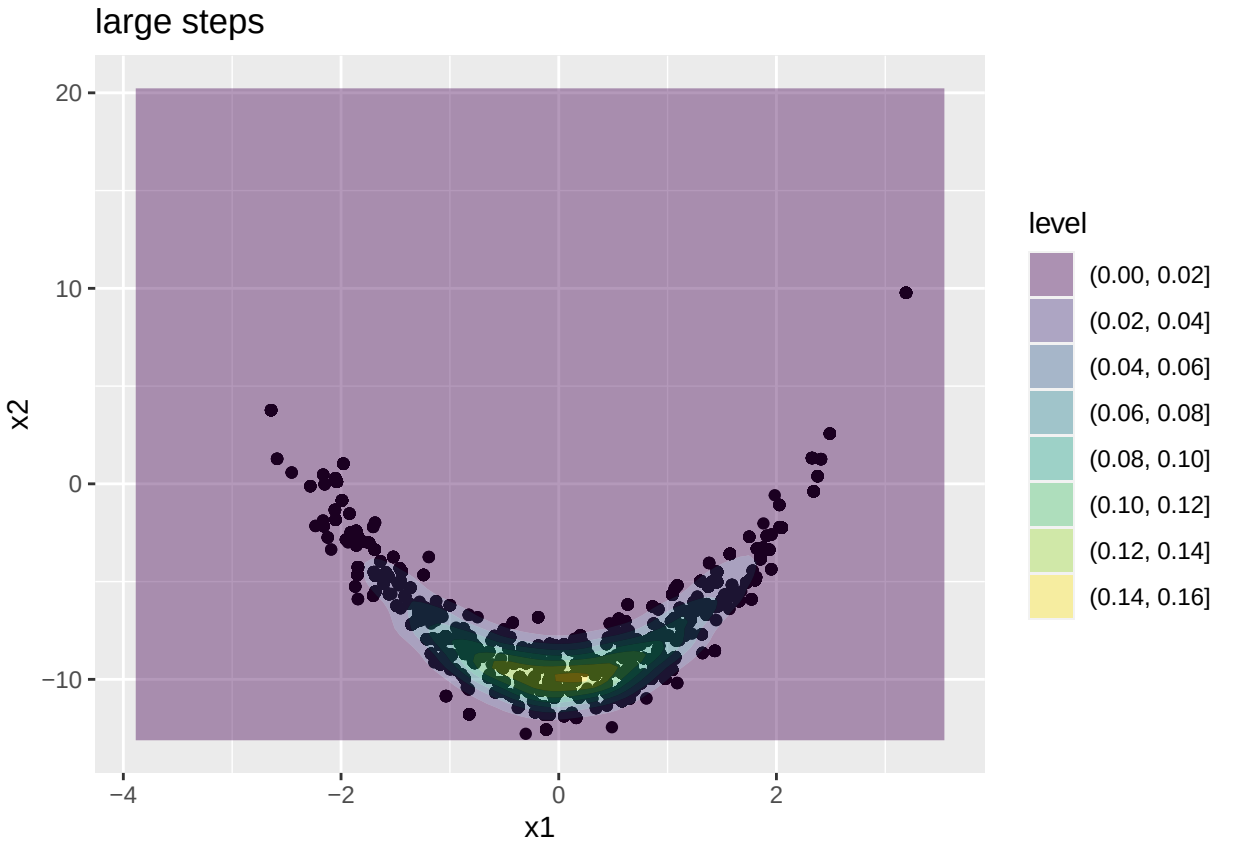
d)

We implement a basic one-dimensional random walk for our Markov Chain: $X_1^*|X_1 \sim N(X_1, \sigma)$ and $X_2^*|X_2 \sim N(X_2, \gamma)$, where X_i^* is the step after X_i and σ, γ our step sizes:

```
bananaMh <- function(n, burn, x1, x2, sigma, gamma){
  nburn <- n*burn
  nsim <- n + nburn

  x1.out <- c()
  x2.out <- c()

  for(i in 1:nsim){
    x1new <- rnorm(1, x1, sigma)
    x2new <- rnorm(1, x2, gamma)

    diff <- log(bananap(x1new, x2new)) - log(bananap(x1, x2))

    if(log(runif(1)) < diff){
      x1 <- x1new
      x2 <- x2new
    }

    if(i > nburn){
      i1 <- i-nburn
      x1.out[i1] <- x1
      x2.out[i1] <- x2
    }
  }
  return(data.frame(x1 = x1.out, x2 = x2.out))
}

Mhsmall <- bananaMh(10000, 0.1, 1, 5, 0.5, 1)
Mhlarge <- bananaMh(10000, 0.1, 1, 5, 5, 10)

ggplot(Mhsmall) +
   geom_point(aes(x1, x2)) +
   labs(title = "small steps") +
    geom_density_2d_filled(mapping = aes(x = x1, y = x2), data = df, alpha = 0.4)
```




```
ggplot(Mhlarge) +
   geom_point(aes(x1, x2)) +
   labs(title = "large steps") +
  geom_density_2d_filled(mapping = aes(x = x1, y = x2), data = df, alpha = 0.4)
```



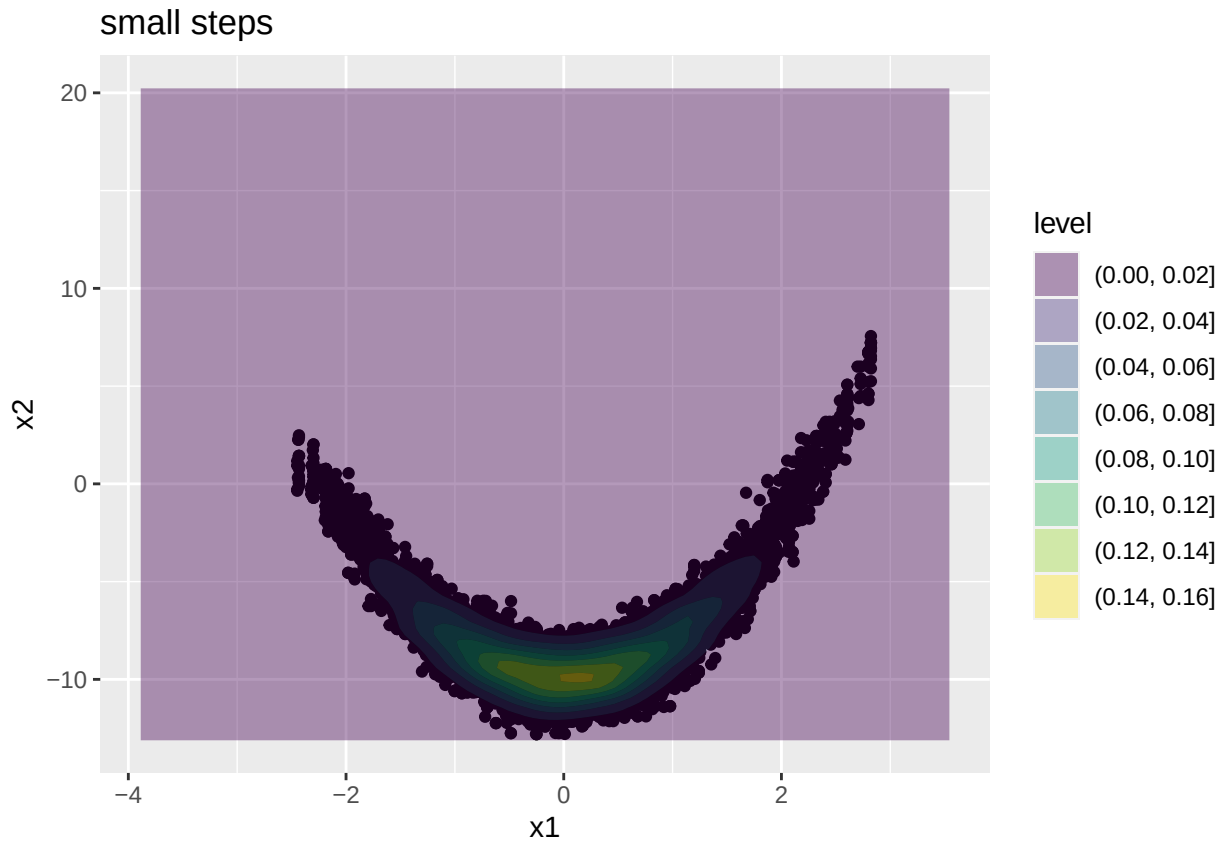
As step size increases in the MC, we see a potential poor mixture due to low acceptance rates. As step size decreases, computational time will increase, but the acceptance rate is higher and we see a good mix. For both of these step sizes, with $n = 10000$ and discarding the first $0.1 * 10000 = 1000$ observations, we see a convergence to the target distribution, highlighted by the density plot.

e)

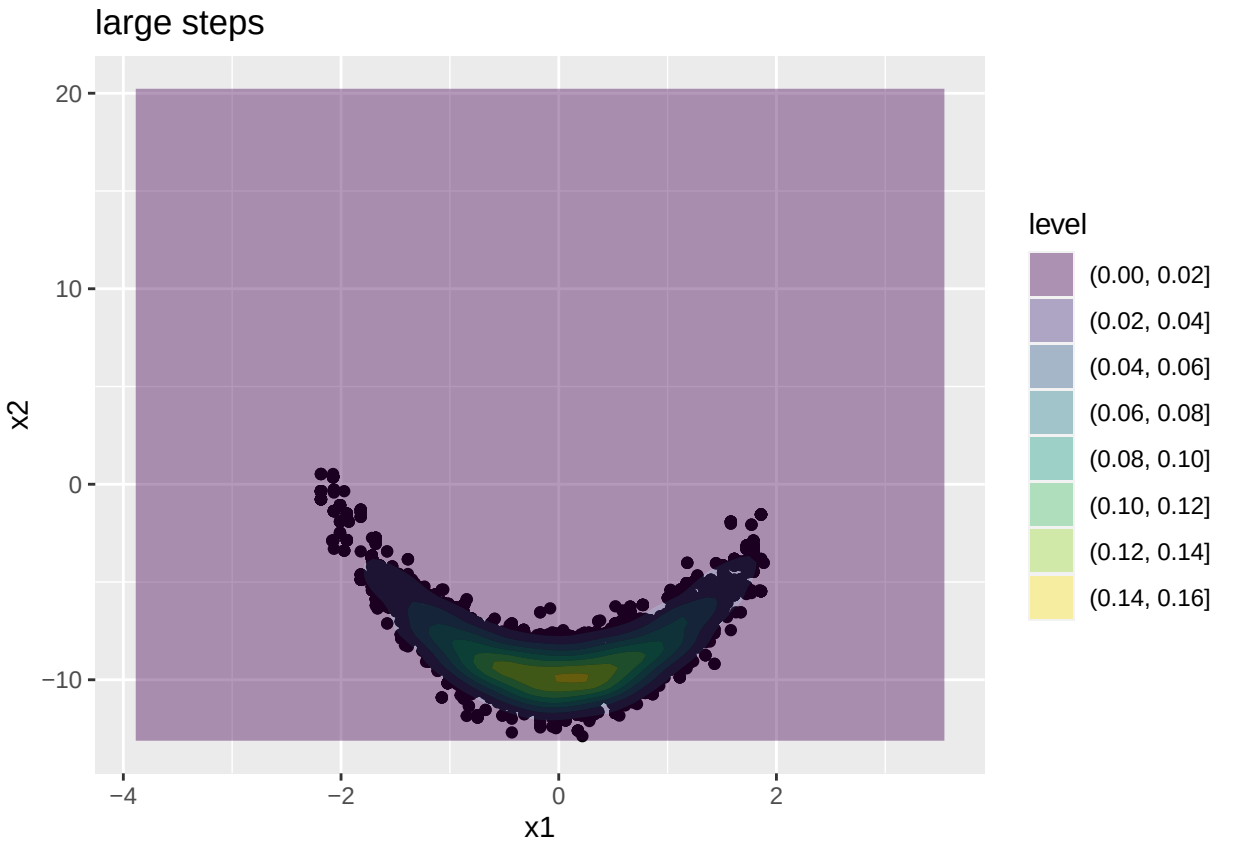
Using the same distributions specified in part (d) for our random walk, we update the conditional distributions in a component-wise fashion:

```
bananaMhCom <- function(n, burn, x1, x2, sigma, gamma){
  nburn <- n*burn
  nsim <- n + nburn

  x1.out <- c()
  x2.out <- c()

  for(i in 1:nsim){
    x1new <- rnorm(1, x1, sigma)
    x2new <- rnorm(1, x2, gamma)

    #x1 | x2
    diffx1 <- log(bananap(x1new, x2)) - log(bananap(x1, x2))
    if(log(runif(1)) < diffx1){
      x1 <- x1new
```

```
    }

    #x2 | x1
    diffx2 <- log(bananap(x1, x2new)) - log(bananap(x1, x2))
    if(log(runif(1)) < diffx2){
      x2 <- x2new
    }

 if(i > nburn){
      i1 <- i-nburn
      x1.out[i1] <- x1
      x2.out[i1] <- x2
    }
  }
  return(data.frame(x1 = x1.out, x2 = x2.out))
}

Mhsmall.com <- bananaMhCom(10000, 0.1, 1, 5, 0.5, 1)
Mhlarge.com <- bananaMhCom(10000, 0.1, 1, 5, 5, 10)

ggplot(Mhsmall.com) +
   geom_point(aes(x1, x2)) +
   labs(title = "small steps") +
    geom_density_2d_filled(mapping = aes(x = x1, y = x2), data = df, alpha = 0.4)
```



```r
ggplot(Mhlarge.com) +
   geom_point(aes(x1, x2)) +
   labs(title = "large steps") +
  geom_density_2d_filled(mapping = aes(x = x1, y = x2), data = df, alpha = 0.4)
```



The same observations from (d) apply regarding step size, mixture, and convergence for exercise (e): larger step sizes lead to a poor mix, even though convergence is still observed for 10000 samples while discarding the first 1000. It is apparent that updating conditional distributions iteratively yields a higher acceptance rate and mix, regardless of step size (small or large), compared to the algorithm implemented in (d).

f)

In order to use component-wise Gibbs sampling, we need to know the full conditional posteriors. From part (a):

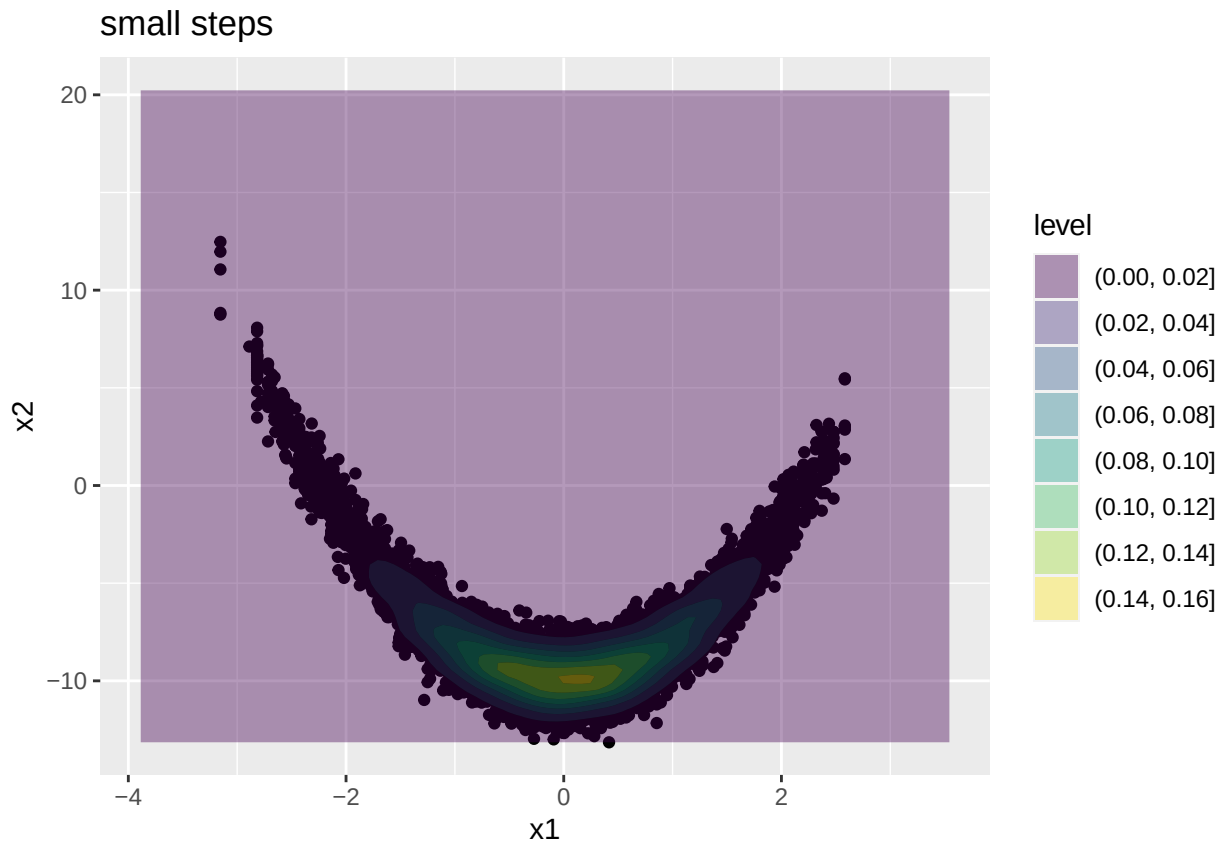
$$p(x_2|x_1) \sim N(2x_1^2 - 10, 1) \tag{1}$$

$$p(x_1|x_2) \propto \exp[-\frac{1}{2}(4(x_1)^2 - 4x_1^2(x_2 + 10) - x_1^2)] \tag{2}$$

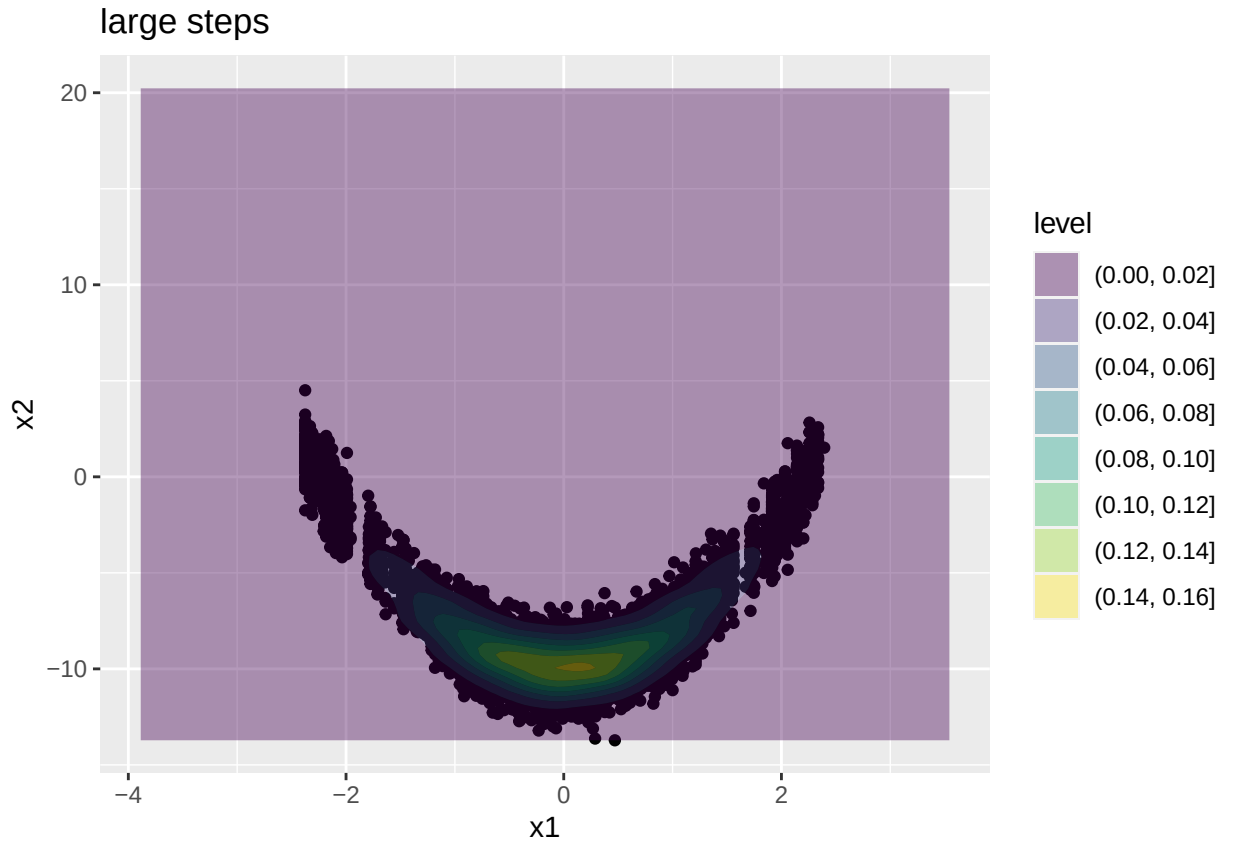
$$\propto \exp[-\frac{1}{2}(4x_1^4 - 4x_1^2x_2 - 41x_1^2)] \tag{3}$$

Therefore, $x_1|x_2$ is a unique distribution that we will sample using Metropolis Hasting.

```r
bananaGibbs <- function(n, burn, x1, x2, sigma){
  nburn <- n*burn
  nsim <- n + nburn

  x1.out <- c()
  x2.out <- c()

  for(i in 1:nsim){
  #x1 | x2
  x1new <- rnorm(1, x1, sigma)
  diffx1 <- log(bananap(x1new, x2)) - log(bananap(x1, x2))
  if(log(runif(1)) < diffx1){
    x1 <- x1new
  }

  #x2 | x1
  x2 <- rnorm(1, 2*(x1^2 -5), 1)

  if(i > nburn){
      i1 <- i-nburn
      x1.out[i1] <- x1
      x2.out[i1] <- x2
    }
  }
  return(data.frame(x1 = x1.out, x2 = x2.out))
}

gibbsSmall <- bananaGibbs(10000, 0.1, 1, 5, 0.5)
gibbsLarge <- bananaGibbs(10000, 0.1, 1, 5, 5)

ggplot(gibbsSmall) +
   geom_point(aes(x1, x2)) +
   labs(title = "small steps") +
    geom_density_2d_filled(mapping = aes(x = x1, y = x2), data = df, alpha = 0.4)
```




```
ggplot(gibbsLarge) +
   geom_point(aes(x1, x2)) +
   labs(title = "large steps") +
  geom_density_2d_filled(mapping = aes(x = x1, y = x2), data = df, alpha = 0.4)
```



Larger step size in the Mh step in the Gibbs sampler yields to a worse mix. Regardless of step size, at $n = 10000$ with discarding the first 1000 samples, the algorithm converges.

g)

```
x12 <- list(dir = df$x1^2,
            ar = AR$x1^2,
            ir = BananaIS$V1^2,
            mh = Mhsmall$x1^2,
            mhcom = Mhsmall.com$x1^2,
            gibbs = gibbsSmall$x1^2)

means <- lapply(x12, mean)
vars <- lapply(x12, var)
cbind(means, vars) %>% kable() %>% kable_styling()
```

All methods provide very similar estimates for mean of X_1. For controlling MC variance, the best algorithm was component-wise Metropolis Hastings, and the worst algorithm was Metropolis Hastings (non-componentwise) and direct sampling.

```
x2 <- list(dir = df$x2,
           ar = AR$x2,
           ir = BananaIS$V2,
```

	means	vars
dir	0.981849860081385	1.92199939903236
ar	0.892787733000681	1.40290573241422
ir	0.894534257694026	1.39687539866422
mh	0.954918102197342	1.50772780435109
mhcom	1.05575155748998	1.83187367702547
gibbs	1.00097697184452	1.92869965293806

	means	vars
dir	-8.02663087710074	8.73363524969792
ar	-8.20234119775215	6.5539820178964
ir	-8.20499062489531	6.52661424789193
mh	-8.10188610546506	6.88472315450761
mhcom	-7.88518104635248	8.31597151402244
gibbs	-7.99220040743578	8.69464234044486

```
            mh = Mhsmall$x2,
            mhcom = Mhsmall.com$x2,
            gibbs = gibbsSmall$x2)

means <- lapply(x2, mean)
vars <- lapply(x2, var)
cbind(means, vars) %>% kable() %>% kable_styling()
```

All methods provide similar estimates for mean of X_2.Regarding variance control, component-wise Metropolis Hastings was the superior algorithm, and non-component-wise Metropolis Hastings / direct sampling / Gibbs sampling performed the poorest.

```
probs <-list(dir = as.numeric(df$x2 + df$x1 > 0),
             ar = as.numeric(AR$x2 + AR$x1 > 0),
             ir = as.numeric(BananaIS$V2 + BananaIS$V1 > 0),
             mh = as.numeric(Mhsmall$x2 + Mhsmall$x1 > 0),
             mhcom = as.numeric(Mhsmall.com$x2 + Mhsmall.com$x1 > 0),
             gibbs = as.numeric(gibbsSmall$x2 + gibbsSmall$x1 > 0))
means <- lapply(probs, mean)
vars <- lapply(probs, var)
cbind(means, vars) %>% kable() %>% kable_styling()
```

All the algorithms seem to yield similar estimates for the mean of $P(X_1+X_2 > 0)$. As before, component-wise Metropolis Hastings seems to have the lowest variance, with regular MH yielding the highest.

	means	vars
dir	0.0298	0.0289148514851485
ar	0.0100801832760596	0.00998085974698748
ir	0.01045	0.0103409009090091
mh	0.021	0.0205610561056106
mhcom	0.0331	0.0320075907590759
gibbs	0.0306	0.0296666066606661

	mean	var
Rao	-7.998046	7.714799
Gibbs	-7.992200	8.694642

h)

Rao-Blackwellization is one method to control MCMC variance in our Gibbs sampler. Since we know the distribution of $x_2|x_1$, we know $E(x_2|x_1) = 2(x_1^2 - 5)$. Then:

```
gibbsSmall$rb <- 2*(gibbsSmall$x1^2 - 5)
rbStats <- data.frame(mean = mean(gibbsSmall$rb), var = var(gibbsSmall$rb))
gibbsStats <- data.frame(mean = mean(gibbsSmall$x2), var = var(gibbsSmall$x2))
rbind(Rao = rbStats, Gibbs = gibbsStats) %>% kable() %>% kable_styling()
```

This is an improvement in variance with an extremely similar mean estimate for X_2.

Antithetic variates is another method to control MCMC variance. In our AR sampler, we can take advantage of the fact that we implement the symmetric normal distributions about 0:

```
getBananaArBetter <- function(nsim, sigma, M){
  x_est <- rmvnorm(n = nsim/2, sigma = sigma)
  x_est1 <- x_est
  #Antithetic step
  x_est1[,1] <- -x_est1[,1]
  x_est <- rbind(x_est, x_est1)
  ##
  x_est_neg <- -abs(x_est)
  x_est <- rbind(x_est, x_est_neg)
  nsim <- nrow(x_est)

  p <- dmvnorm(x_est, sigma = sigma)
  pban <- rep(NA, nsim)

  for(i in 1:nsim){
    x1 <- x_est[i, 1]
    x2 <- x_est[i, 2]
    pban[i] <- bananap(x1, x2)
  }
  u <- runif(nsim)
  c <- pban/p * 1/M

  AR <- x_est[c >= u,] %>% as.data.frame()
  colnames(AR) <- c("x1", "x2")

  isProductGood <- ifelse(length(which(pban / (p * M) > 1)) == 0, "yes", "no")
  eff <- nrow(AR)[1]/nsim

  return(list(AR = AR, isProductGood = isProductGood, efficiency = eff, f = pban, g = p, x_est = x_est))
}

BananaArBetter <- getBananaArBetter(nsim, sigma, M)
ARBetter <- BananaArBetter$AR
```

```r
ARBetterstats <- data.frame(x1.mean = mean(ARBetter$x1), x1.var = var(ARBetter$x1),
                            x2.mean = mean(ARBetter$x2), x2.var = var(ARBetter$x2))

ARstats <- data.frame(x1.mean = mean(AR$x1), x1.var = var(AR$x1),
                      x2.mean = mean(AR$x2), x2.var = var(AR$x2))

rbind(AntitheticVariates = ARBetterstats, AR = ARstats)
```

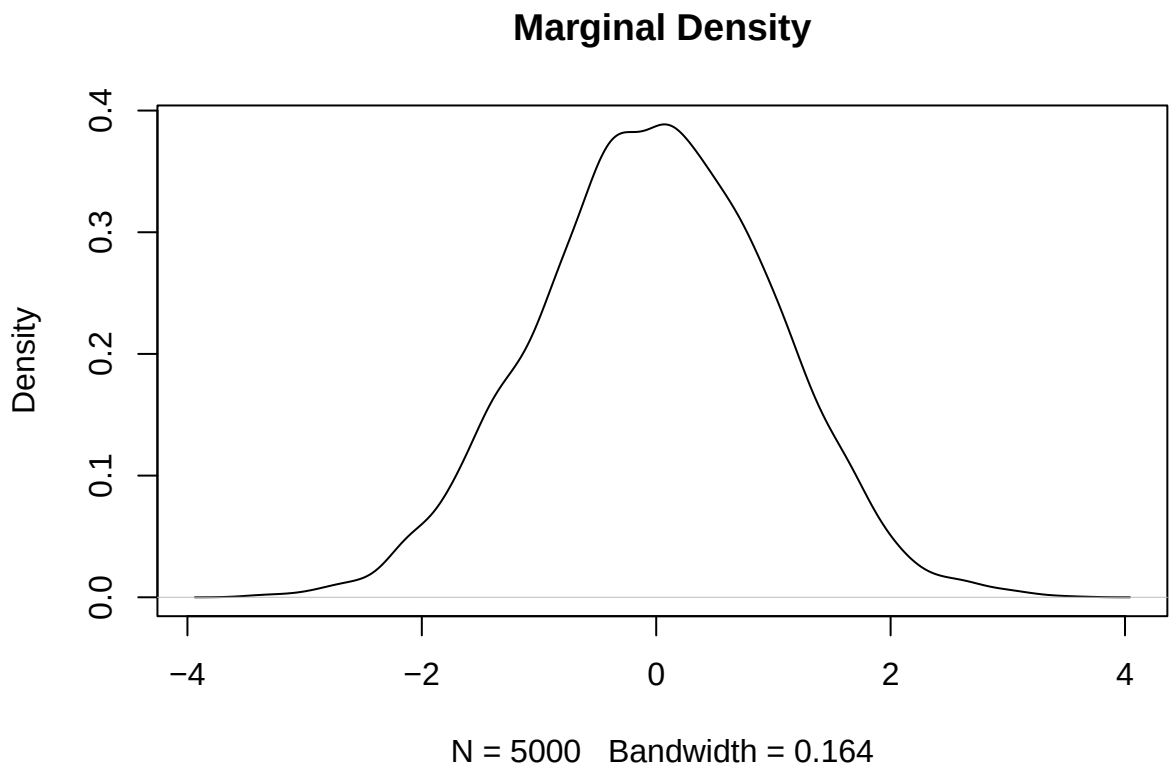
```
##                       x1.mean    x1.var   x2.mean   x2.var
## AntitheticVariates -0.4499130 0.7204882 -8.149132 6.601979
## AR                 -0.4867678 0.6559952 -8.202341 6.553982
```

Implementation of antithetic variates yields a very slight decrease in variance in estimating X_1 and X_2.

Bayesian Adaptive Lasso

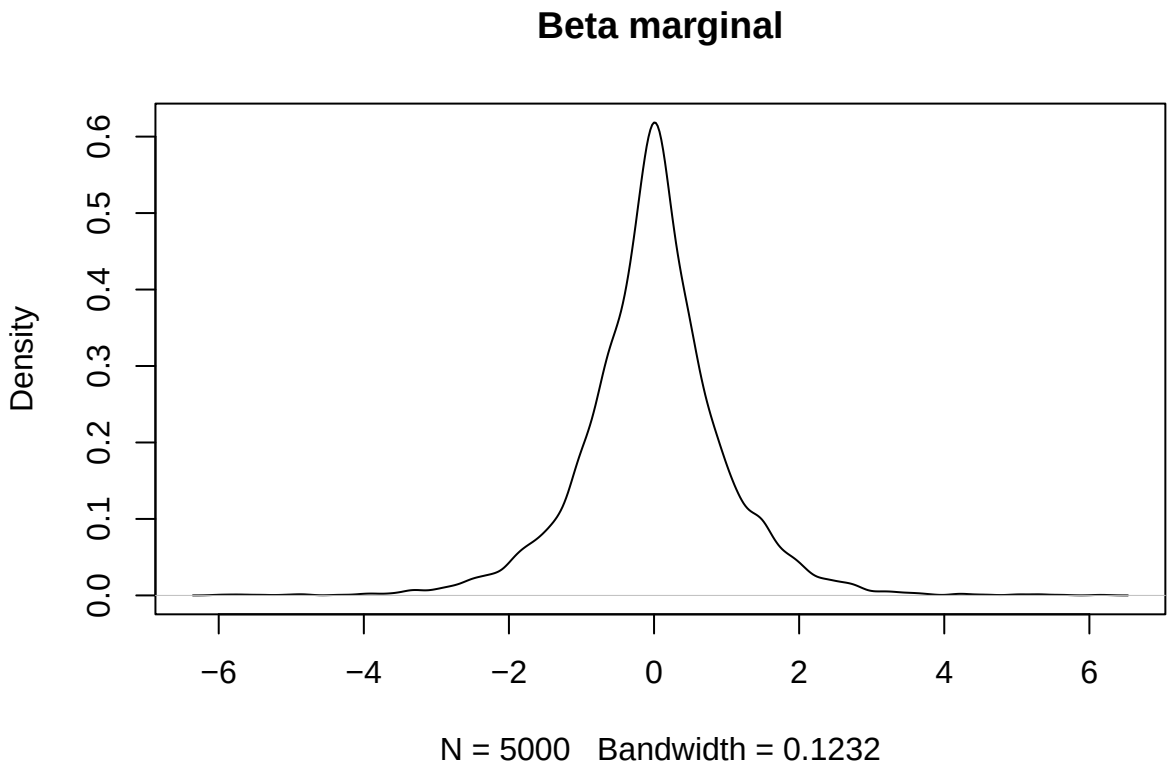
#a)

```r
sima <- rnorm(5000, 0, 1)
plot(density(sima), main = "Marginal Density")
```



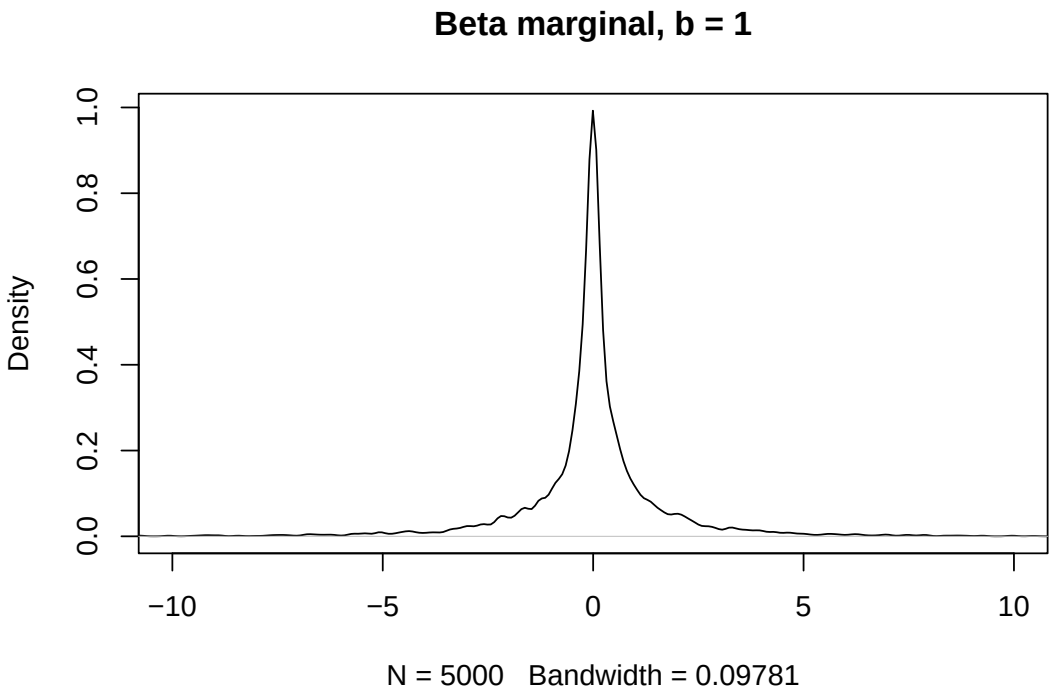
b)

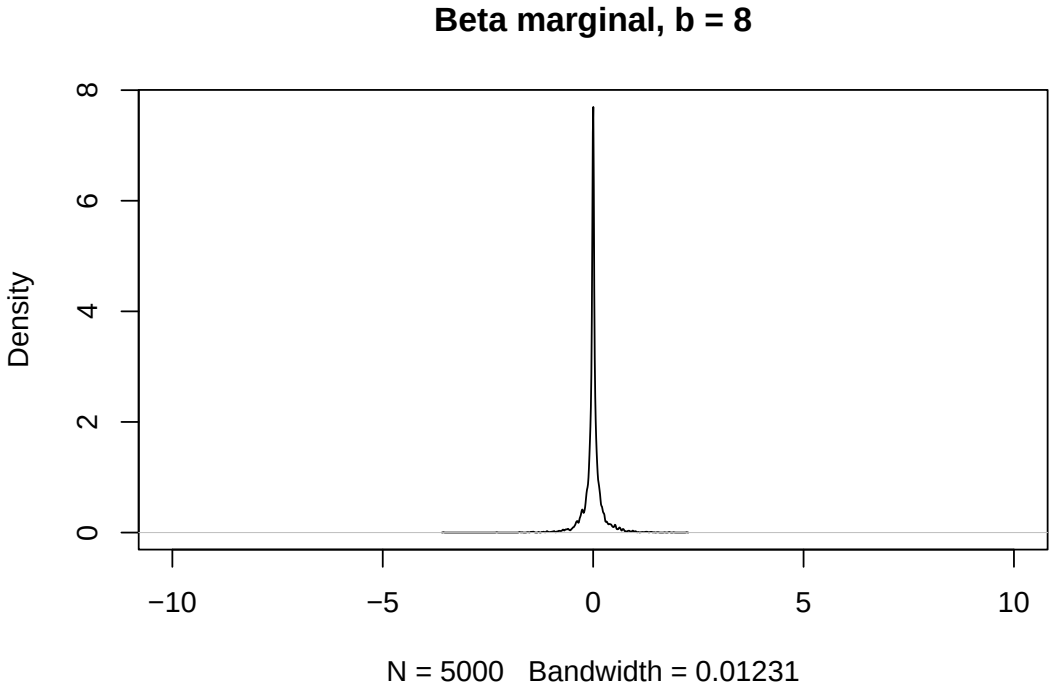
```
lambda2 <- 2
tau2 <- rgamma(5000, shape = 1, rate = lambda2/2)
simb <- rnorm(5000, 0, sqrt(tau2))
plot(density(simb), main = "Beta marginal")
```

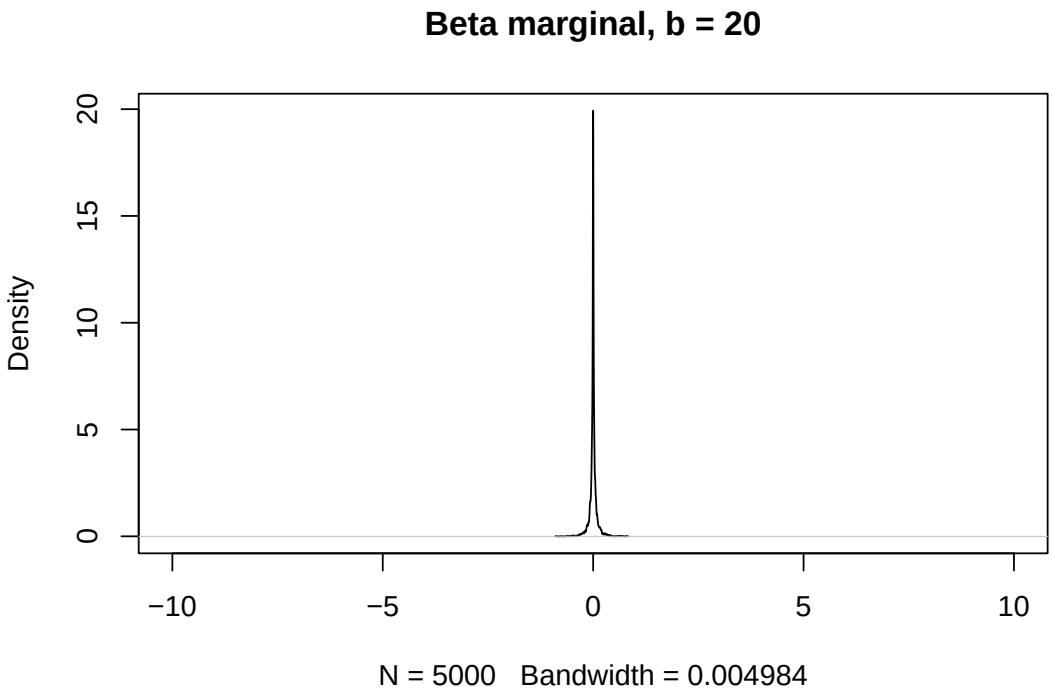


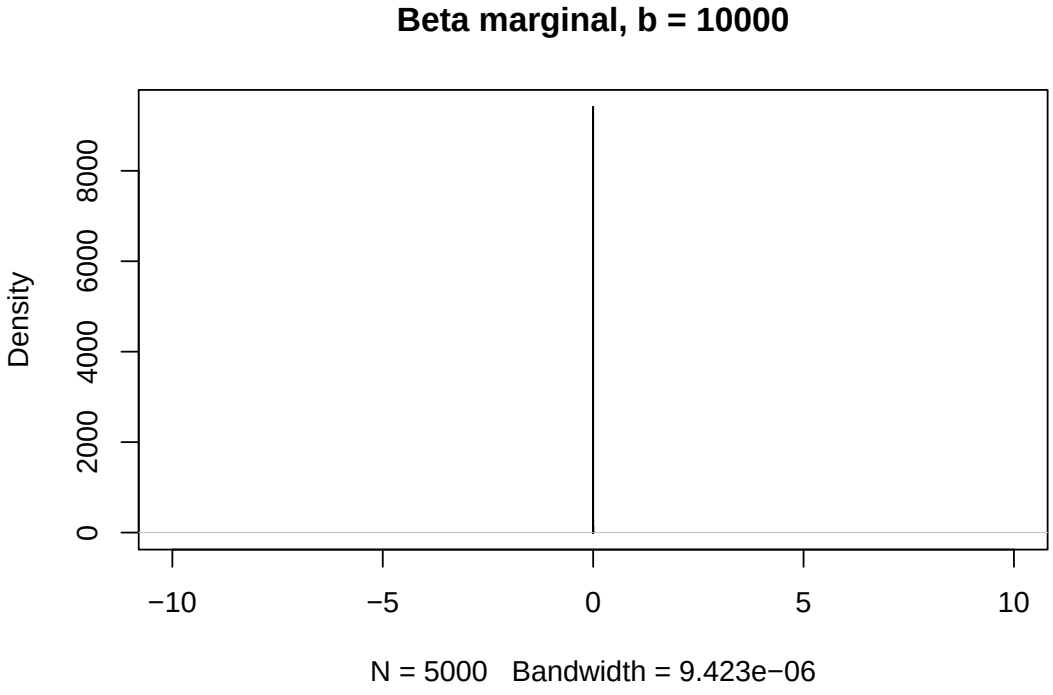
c)

```
bvec <- c(1, 8, 20, 10000)
marginalplot <- function(n, b){
  lambda <- 1/rgamma(n, 1, b)
  tau2 <- rgamma(n, shape = 1, rate = lambda^2/2)
  sim <- rnorm(n, 0, sqrt(tau2))
  plot <- plot(density(sim),
               main = paste0("Beta marginal, b = ", b),
               xlim = c(-10, 10))
  save_plot <- recordPlot()
  return(save_plot)
}

plots <- lapply(bvec, marginalplot, n = 5000)
```


Beta marginal, b = 1
Density
0.0
0.2
0.4
0.6
0.8
1.0
−10
−5
0
5
10
N = 5000 Bandwidth = 0.09781


Beta marginal, b = 8
Density
0
2
4
6
8
−10
−5
0
5
10
N = 5000 Bandwidth = 0.01231


Beta marginal, b = 20
Density
0
5
10
15
20
−10
−5
0
5
10
N = 5000 Bandwidth = 0.004984


Beta marginal, b = 10000
Density
0
2000
4000
6000
8000
−10
−5
0
5
10
N = 5000 Bandwidth = 9.423e−06

d)

We compute the full conditionals below for use in MCMC implementation:

$$p(\vec{\beta}, \tau_1^2, \ldots, \tau_p^2, \sigma^2, \lambda^2 | \vec{y}) \propto p(\vec{y}|\vec{\beta}, \tau_1^2, \ldots, \tau_p^2, \sigma^2, \lambda^2) p(\vec{\beta}|\tau_1^2, \ldots, \tau_p^2,) p(\tau_1^2, \ldots, \tau_p^2, |\lambda^2) p(\lambda^2) p(\sigma^2)$$

$$\begin{aligned}
p(\sigma^2|\vec{y}) &\propto p(\vec{y}|\vec{\beta}, \tau_1^2, \ldots, \tau_p^2, \beta, \lambda^2) \\
&\propto p(\vec{y}|\vec{\beta}, \tau_1^2, \ldots, \tau_p^2, \sigma^2, \lambda^2) p(\sigma^2) \\
&\propto (\sigma^2)^{\frac{-n}{2}} \exp[\frac{-1}{2\sigma^2}(\vec{y} - X\vec{\beta})^T(\vec{y} - X\vec{\beta})](\sigma^2)^{-0.1-1} \exp(\frac{-10}{\sigma^2}) \\
&= \sigma^{2(-\frac{n}{2}+0.1)-1} \exp[-\frac{1}{\sigma^2}(10 + \frac{1}{2}(\vec{y} - X\vec{\beta})^T(\vec{y} - X\vec{\beta})] \\
&= \text{Inverse Gamma}(\frac{n}{2} + 0.1, 10 + \frac{1}{2}(\vec{y} - X\vec{\beta})^T(\vec{y} - X\vec{\beta}))
\end{aligned}$$

$$\begin{aligned}
p(\vec{\beta}|\vec{y}) &\propto p(\vec{y}|\vec{\beta}, \tau_1^2, \ldots, \tau_p^2, \sigma^2, \lambda^2)(\vec{\beta}|\tau_1^2, \ldots, \tau_p^2,) \\
&\propto \exp[-\frac{1}{2\sigma^2}(\vec{y} - X\vec{\beta})^T(\vec{y} - X\vec{\beta})] \exp[-\frac{1}{2}\Sigma^{-1}\vec{\beta}^T\vec{\beta}] \text{ for } \Sigma = \text{diag}(\tau_i^2) \\
&= \exp[-\frac{1}{2\sigma^2}(\vec{y}^T\vec{y} - 2\vec{y}^y(X\vec{(\beta)})^T + (X\vec{\beta})^T(X\vec{\beta})) - \frac{1}{2}\vec{\beta}^T\Sigma^{-1}\vec{\beta})] \\
&\propto \exp[-\frac{1}{2}(\vec{\beta}^T(\frac{X^TX}{\sigma^2} + \Sigma^{-1})\vec{\beta}) - \frac{1}{2\sigma^2}2(X\vec{\beta})^T\vec{y}] \\
&= N_p([\frac{X^TX}{\sigma^2} + \Sigma^{-1}]^{-1}\frac{X^T\vec{y}}{\sigma^2}, [\frac{X^TX}{\sigma^2} + \Sigma^{-1}]^{-1}) \\
p(\tau_i^2|\beta_i, \lambda^2, \sigma^2, \vec{y}) &\propto p(\beta_i|\tau_i^2)p(\tau_i^2|\lambda^2) \\
&\propto \frac{1}{\tau_i}\exp[-\frac{1}{2}(\frac{\beta_i}{\tau_i})^2 - \frac{\lambda^2}{2}\tau_i^2] \\
&= (\frac{1}{\tau_i^2})^{\frac{1}{2}}\exp[-\frac{1}{2}(\beta^2(\frac{1}{\tau_i^2}) + \lambda^2(\frac{1}{\frac{1}{\tau_i^2}})]
\end{aligned}$$

Let $u = \frac{1}{\tau_i^2}$ such that the Jacobian: $\frac{d}{du}(\frac{1}{u}) = -\frac{1}{u^2}$

$$\begin{aligned}
p(e|\beta_i, \lambda) &\propto u^{\frac{1}{2}}\exp[-\frac{\lambda^2}{2}(\frac{\beta_i^2}{\lambda^2}u + \frac{1}{u})]u^{-2} \\
&\propto e^{-\frac{3}{2}}\exp[-\frac{\lambda^2}{2}(\frac{\beta_i^2}{\lambda^2}u + \frac{1}{u})] \\
&= IG([\frac{\lambda^2}{\beta_i^2}]^{\frac{1}{2}}, \lambda^2)
\end{aligned}$$

$$\begin{aligned}
p(\lambda_i^2|\tau_i^2, \vec{\beta}, \sigma^2, \vec{y}) &\propto p(\vec{\tau}|\lambda^2)p(\lambda^2) \\
&\propto \Pi_{i=1}^p \exp[-\frac{\lambda^2}{2}\tau_i^2](\lambda^2)^{a-1}\exp[-b\lambda^2] \\
&\propto \exp[-(\frac{\sum_{i=1}^p \tau_i^2}{2} + b)\lambda^2](\lambda^2)^{a-1} \\
&= \text{Gamma}(a, \frac{\sum_{i=1}^p \tau_i^2}{2} + b)
\end{aligned}$$

e)

I will implement a Gibbs Sampler algorithm to sample from the posterior distributions.

```
gibbs <- function(y = y, X = Xdat,
                  lambda2, tau2,
                  n.sim = 1000, burn = 0.1,
                  a = 1, b = 1, beta, sigma2, fixlambda = FALSE){
  #Chain information
  n.total <- n.sim*(1.0 + burn)
  n.burn <- n.sim*burn

  #Initializing matrices
  betamu.out <- matrix(NA, n.sim, 10)
  beta.out <- matrix(NA, n.sim, 10)
  sigma2.out <- c()

  #Data and Parameters
  n <- length(y)
  p <- ncol(X)
  XtX <- t(X) %*% X

  for(i in 1:n.total){
    #beta
    betavar <- XtX * (1/sigma2) + solve(diag(tau2))
    betamu <- solve(betavar) %*% t(X) %*% y * (1/sigma2)
    beta <- rmvnorm(n=1, mean = betamu, sigma = solve(betavar)) %>% t()
    #sigma2
    shape <- n/2 + 0.1
    rate <- (t(y - X %*% beta)%*%(y - X %*% beta) * 0.5) + 10
    sigma2 <- 1/rgamma(1, shape = shape, rate = rate)
    #lambda2
    if(fixlambda == TRUE){
        lambda2 <- lambda2
    } else {
        lambda2 <- rgamma(1, shape = a, rate = sum(tau2) * 0.5 + b)
    }
    #tau2
    for(a in 1:p){
      tau2[a] <- 1/rinvgauss(1, mean = sqrt(lambda2)/sqrt(beta[a]^2), shape = lambda2)
    }
    #store values
    if(i > n.burn){
      i1 <- i - n.burn
      betamu.out[i1, ] <- betamu
      beta.out[i1, ] <- beta
      sigma2.out[i1] <- sigma2
    }
  }
 return(list(beta = beta.out, sigma2 = sigma2.out))
}

data("diabetes")
Xdat <- as.matrix(diabetes$x)
```

beta	My MCMC	Glmnet
age	-4.9193975	-9.220679
sex	-214.8012166	-239.063410
bmi	519.5842341	520.458638
map	312.1303358	323.616100
tc	-176.1894509	-716.483385
ldl	0.2016306	418.448411
hdl	-159.1730213	65.386847
tch	86.5854530	164.562823
ltg	525.2089143	723.735577
glu	61.7190240	67.540420

```
y <- diabetes$y
Xdat.c <- as.matrix(scale(diabetes$x, scale = FALSE))
y.c <- scale(diabetes$y, scale = FALSE)

betahat <- solve(t(Xdat) %*% Xdat) %*% t(Xdat) %*% y
out <- gibbs(y = y.c, X = Xdat, n = 1000, burn = 0.1,
             a = 1, b = 1.78, beta = betahat, lambda2 = 0.25, sigma2 = 1, tau2 = rep(1, 10))

coeffun <- apply(out$beta, 2, function(x){quantile(x,c(0.16,0.5,0.84))})
coefme <- coeffun[2,]

fit.glm <- glmnet(Xdat, y)
coefglm <- coef(fit.glm, s = min(fit.glm$lambda))
compare <- tibble("beta" = colnames(diabetes$x), "My MCMC" = coefme, "Glmnet" = matrix(coefglm[-1]))

kable(compare) %>% kable_styling(full_width = F)
```

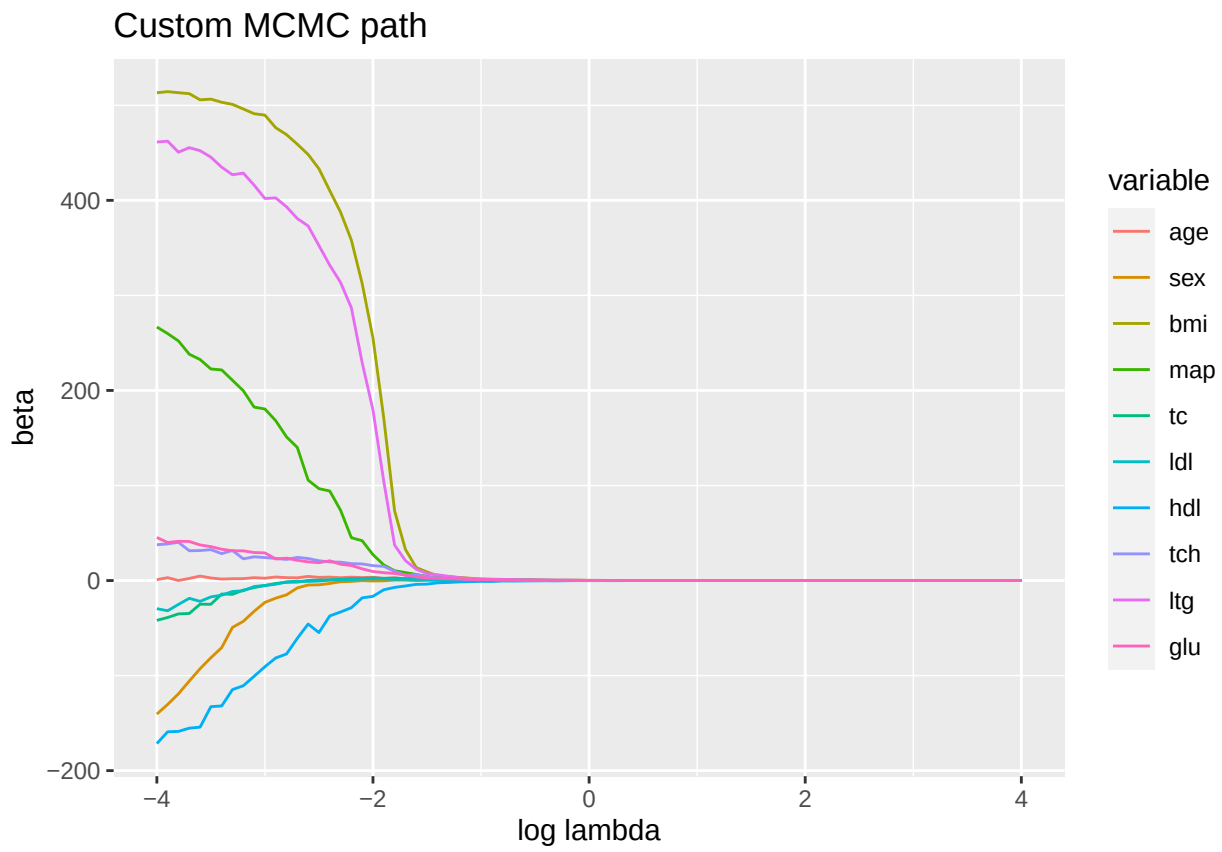
In comparison to results from glmnet, many coefficients are quite similar, with only coefficients for covariates tc, ldl, and hdl differing between the two models.

f

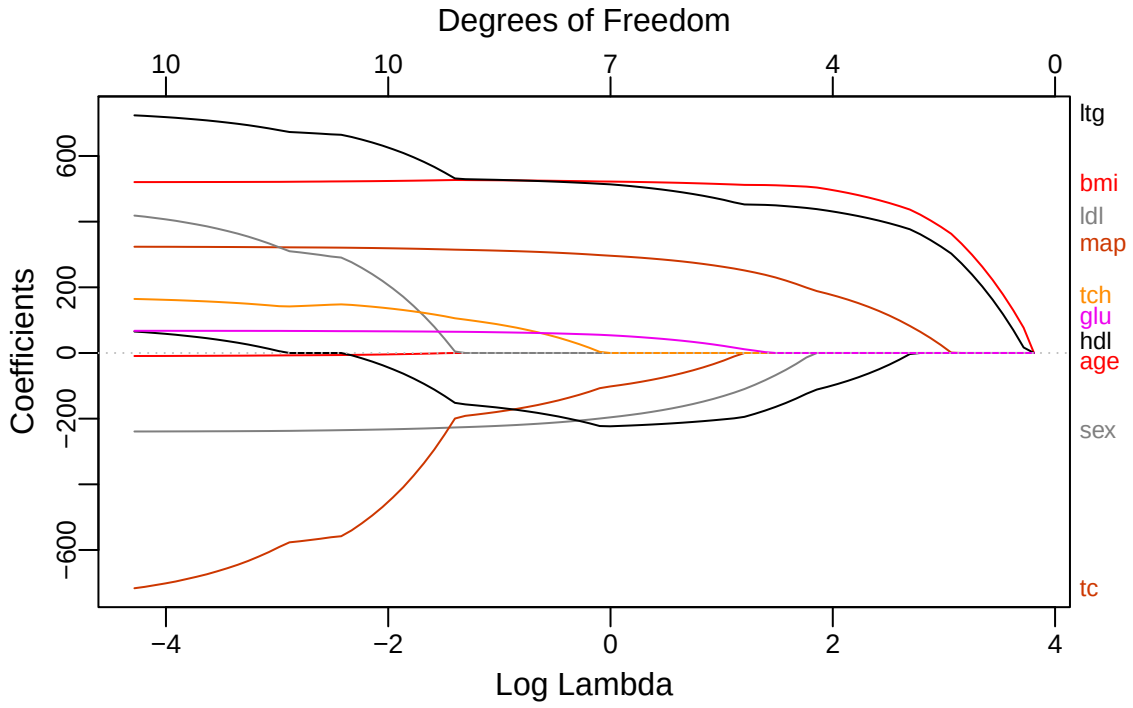
```
lambdas <- exp(seq(-4, 4, 0.1))^2
out <- lapply(lambdas, function(var)gibbs(y = y.c, X = Xdat, n = 1000, burn = 0.1,
             a = 1, b = 1.78, beta = betahat, lambda2 = var, tau2 = rep(1, 10), sigma2 = 1, fixlambda =

betas <- lapply(out, `[`, c("beta"))
betas <- lapply(betas, sapply, colMedians)

df <- do.call("cbind", betas)
rownames(df) <- colnames(Xdat)
df <- t(df) %>% as.data.frame() %>% melt()
```

```
## No id variables; using all as measure variables
```

```
df$lambda <- rep(seq(-4, 4, 0.1), 10)

ggplot(data = df, aes(lambda, value, color = variable)) +
  geom_line() +
  ggtitle("Custom MCMC path") + xlab("log lambda") + ylab("beta")
```




```
plot_glmnet(fit.glm, xvar = "lambda")
```



λ was fixed across a range of $[-4, 4]$, and coefficients were computed for values in this range spaced 0.1 apart. It appears that the regularization path converges a lot earlier with the adaptive Bayesian Lasso than with glmnet, even though converging structures of the paths seem to be similar.

g

```
df.ab <- data.frame(a = c(1, 5, 10, 1, 5, 10, 1, 5, 10),
                    b = c(1, 1, 1, 5, 5, 5, 10, 10, 10))
out <- apply(df.ab, 1, function(df.ab)gibbs(y = y.c, X = Xdat,
                     n.sim = 1000, burn = 0.1,
                     a = df.ab[1], b = df.ab[2], beta = betahat,
                     lambda2 = 1, sigma2 = 1, tau2 = rep(1, 10)))
betas <- lapply(out, '[', c("beta"))
betas <- lapply(betas, sapply, colMedians)
df <- do.call("cbind", betas)
colnames(df) <- paste0("(",df.ab$a,", ", df.ab$b, ")")
rownames(df) <- colnames(Xdat)
df %>% kable(digits = 1) %>% kable_styling(full_width = F)
```

Lambda was generated initially with a gamma(a, b) distribution, and then updated using its conditional posterior described in part (d). It appears that as b increases for $b > 1$, even with varying as, the coefficients are not as sensitive to change. This might indicate that for the given Bayesian Lasso algorithm, it is robust to changes in its hyper prior parameters.

	(1, 1)	(5, 1)	(10, 1)	(1, 5)	(5, 5)	(10, 5)	(1, 10)	(5, 10)	(10, 10)
age	-0.5	-4.7	-2.8	-0.8	-5.7	3.4	0.6	-3.6	-2.5
sex	-212.5	-213.9	-211.2	-208.6	-215.5	-210.6	-214.2	-214.1	-213.1
bmi	528.0	524.0	524.4	518.7	517.5	520.2	523.3	519.9	518.5
map	304.9	311.8	302.7	303.0	312.7	306.9	307.9	304.4	305.9
tc	-169.4	-156.5	-172.2	-161.3	-183.2	-160.1	-178.4	-182.4	-162.7
ldl	-2.3	-1.6	-3.1	-10.3	3.8	0.7	-3.2	7.2	-0.4
hdl	-149.5	-152.6	-154.3	-145.8	-148.7	-153.9	-149.9	-144.8	-144.0
tch	95.2	87.3	87.9	92.6	94.5	82.4	95.6	95.9	95.7
ltg	520.9	513.9	526.1	522.4	526.8	520.1	527.9	528.3	523.2
glu	64.7	65.6	62.2	66.1	66.4	62.8	61.4	58.4	62.0