HW2

Jiahao Tian

2023-01-28

Question 1

a) MH

```
lgbanana = function(x1, x2) {
  -x1 ^ 2.0 / 2.0 - (x2 - 2.0 * (x1 ^ 2 - 5.0)) ^ 2.0 / 2.0
}
```

```
mh = function(n_iter, burn, x1, x2, cand_sd) {

  ## step 1, initialize
  x1_out = numeric(n_iter)
  x2_out = numeric(n_iter)
  accpt = 0
  x1_now = x1
  x2_now = x2
  lgbanana_now = lgbanana(x1 = x1_now, x2 = x2_now)

  ## step 2, iterate
  for(i in 1:n_iter) {
    x1_cand = rnorm(1, x1_now, cand_sd) # draw a candidate
    x2_cand = rnorm(1, x2_now, cand_sd) # draw a candidate
    lgbanana_cand = lgbanana(x1 = x1_cand, x2 = x2_cand) # evaluate log banana with the candidate
    alpha = lgbanana_cand - lgbanana_now # log of acceptance ratio

    u = log(runif(1))
    if(u < alpha) {
      x1_now = x1_cand
      x2_now = x2_cand
      accpt = accpt + 1 # to keep track of acceptance
      lgbanana_now = lgbanana_cand
    }
    if(i > burn) {
      i1 = i - burn
      x1_out[i1] = x1_now
      x2_out[i1] = x2_now
      }

}
```

```
  list(x1 = x1_out, x2 = x2_out, accpt = accpt / n_iter)

}
```

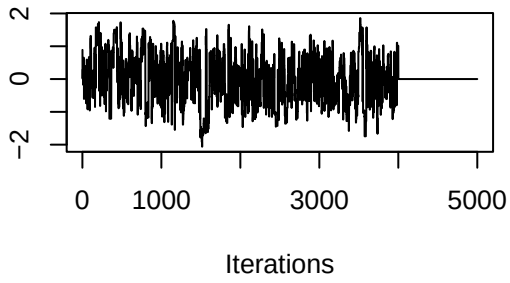
```
set.seed(43)
post1 = mh(n_iter = 5000, burn = 1000, x1 = 1, x2 = 5, cand_sd = 1)
post2 = mh(n_iter = 5000, burn = 1000, x1 = 1, x2 = 5, cand_sd = 5)
str(post1)
```

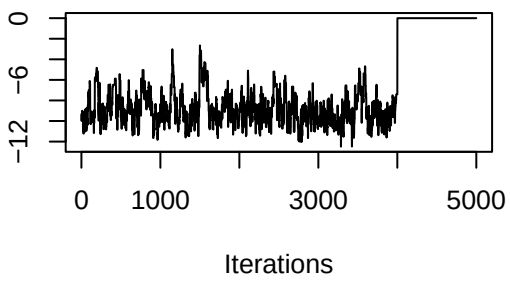
```
## List of 3
##  $ x1   : num [1:5000] 0.8895 0.0127 0.0127 0.0127 0.2919 ...
##  $ x2   : num [1:5000] -9.33 -9.99 -9.99 -9.99 -9.34 ...
##  $ accpt: num 0.377
```

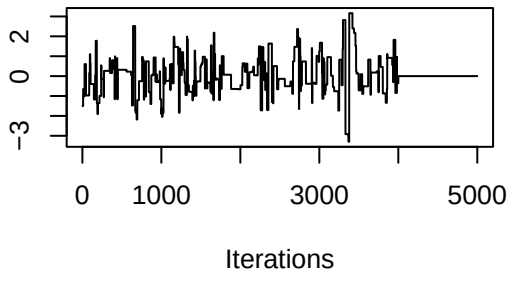
```
str(post2)
```

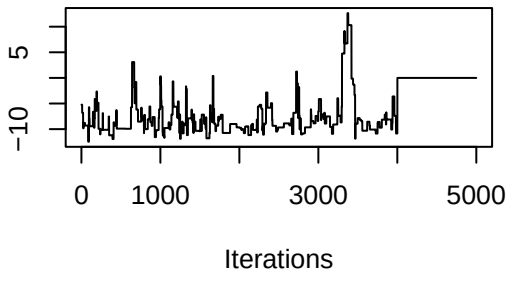
```
## List of 3
##  $ x1   : num [1:5000] -1.5 -1.5 -1.5 -1.5 -1.5 ...
##  $ x2   : num [1:5000] -5.23 -5.23 -5.23 -5.23 -5.23 ...
##  $ accpt: num 0.0576
```

```
library("coda")
par(mfrow=c(2,2))
traceplot(as.mcmc(post1$x1))
traceplot(as.mcmc(post1$x2))
traceplot(as.mcmc(post2$x1))
traceplot(as.mcmc(post2$x2))
```

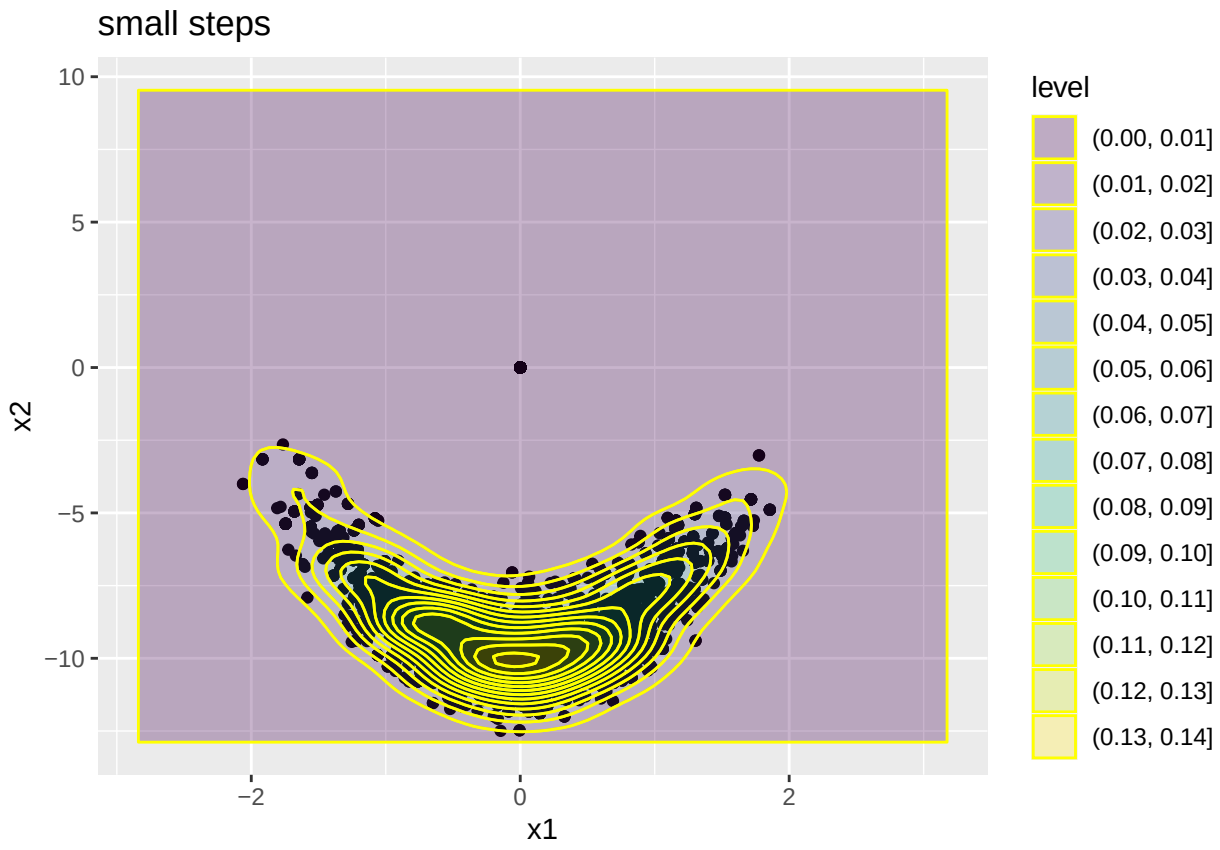




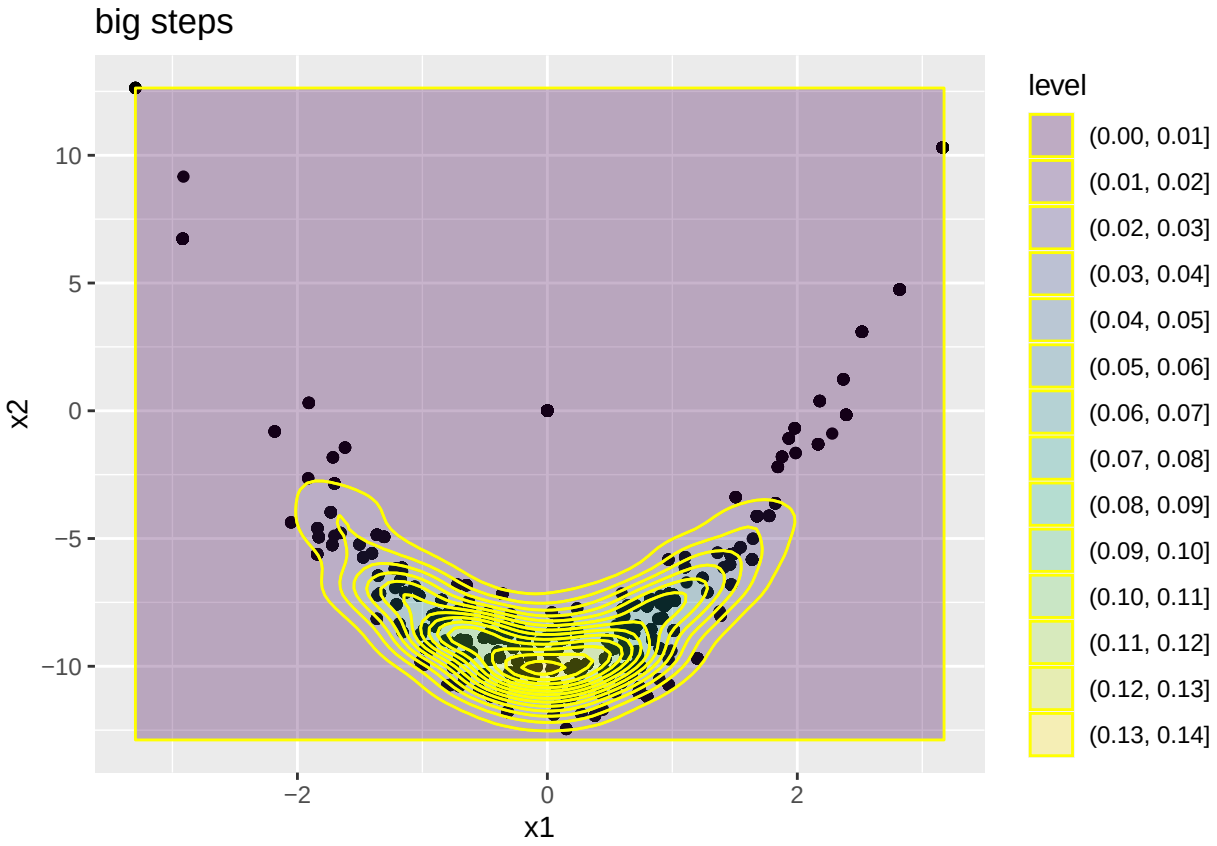




```
set.seed(4)
x11 = rnorm(1000, 0, 1)
x22 = rnorm(1000, 2 * (x11 ^ 2 - 5), 1)
banana = as.data.frame(exp(-x11^2 / 2) * exp(-(x22 - 2 * (x11^2 - 5.0))^2 / 2))
bananadd = as.data.frame(cbind(x11, x22))
post11 = as.data.frame(post1)
post22 = as.data.frame(post2)

library(ggplot2)

ggplot(post11) + geom_point(aes(x1, x2)) +
  labs(title = "small steps") +
  geom_density_2d_filled(mapping = aes(x = x11, y = x22),
                         data = bananadd, alpha = 0.3, color = "yellow")
```




```
ggplot(post22) + geom_point(aes(x1, x2)) +
  labs(title = "big steps") +
  geom_density_2d_filled(mapping = aes(x = x11, y = x22),
                         data = bananadd, alpha = 0.3, color = "yellow")
```



- When the step size is large, the acceptance rate is 5.76%. At low step size, the acceptance rate is increase to 37.7% and we see a good mix here. Because the acceptance rate is good between 23% -50%. For lower step sizes, with n_iter = 5000 and burn the first 1000 observations, we see a convergence to the target distribution.

b) MH continue, $x_1|x_2, \quad x_2|x_1$

```
mh2 = function(n_iter, burn, x1, x2, cand_sd) {

  ## step 1, initialize
  x1_out = numeric(n_iter)
  x2_out = numeric(n_iter)
  x1_now = x1
  x2_now = x2
  lgbanana_now = lgbanana(x1 = x1_now, x2 = x2_now)

  ## step 2, iterate
  for(i in 1:n_iter) {
    x1_cand = rnorm(1, x1_now, cand_sd) # draw a candidate
    x2_cand = rnorm(1, x2_now, cand_sd) # draw a candidate

    ##x1 | x2
    lgbanana_cand1 = lgbanana(x1 = x1_cand, x2 = x2_now) # evaluate log banana with the candidate
    alpha1 = lgbanana_cand1 - lgbanana_now # log of acceptance ratio
```

```
    u = log(runif(1))
    if(u < alpha1) {
      x1_now = x1_cand
      lgbanana_now = lgbanana_cand1
    }

     ##x2 | x1
    lgbanana_cand2 = lgbanana(x1 = x1_now, x2 = x2_cand) # evaluate log banana with the candidate
    alpha2 = lgbanana_cand2 - lgbanana_now # log of acceptance ratio

    u = log(runif(1))
    if(u < alpha2) {
      x2_now = x2_cand
      lgbanana_now = lgbanana_cand2
    }

    if(i > burn) {
      i1 = i - burn
      x1_out[i1] = x1_now # save both x1 x2
      x2_out[i1] = x2_now
      }

}
  list(x1 = x1_out, x2 = x2_out)

}
```

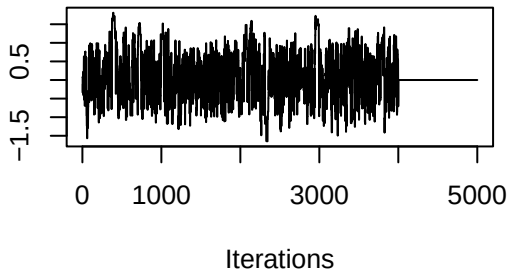
```
set.seed(43444)
post3 = mh2(n_iter = 5000, burn = 1000, x1 = 1, x2 = 5, cand_sd = 1)
post4 = mh2(n_iter = 5000, burn = 1000, x1 = 1, x2 = 5, cand_sd = 10)
str(post3)
```

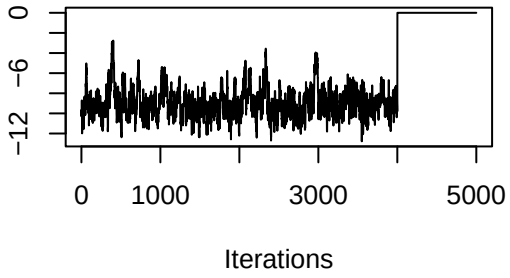
```
## List of 2
##  $ x1: num [1:5000] 0.08 0.08 0.08 -0.256 -0.343 ...
##  $ x2: num [1:5000] -10.3 -10.3 -9.63 -10.27 -8.71 ...
```

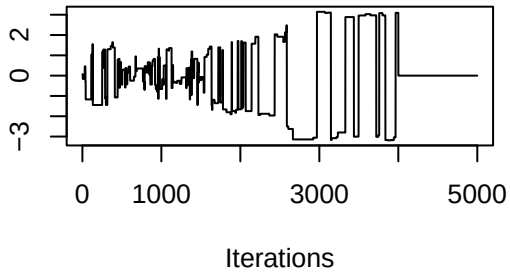
```
str(post4)
```

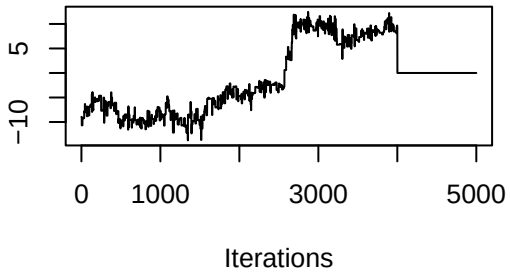
```
## List of 2
##  $ x1: num [1:5000] 0.0704 0.0704 0.0704 0.0704 -0.1595 ...
##  $ x2: num [1:5000] -8.95 -8.95 -8.95 -8.95 -10.68 ...
```

```
library("coda")
par(mfrow=c(2,2))
traceplot(as.mcmc(post3$x1))
traceplot(as.mcmc(post3$x2))
traceplot(as.mcmc(post4$x1))
traceplot(as.mcmc(post4$x2))
```

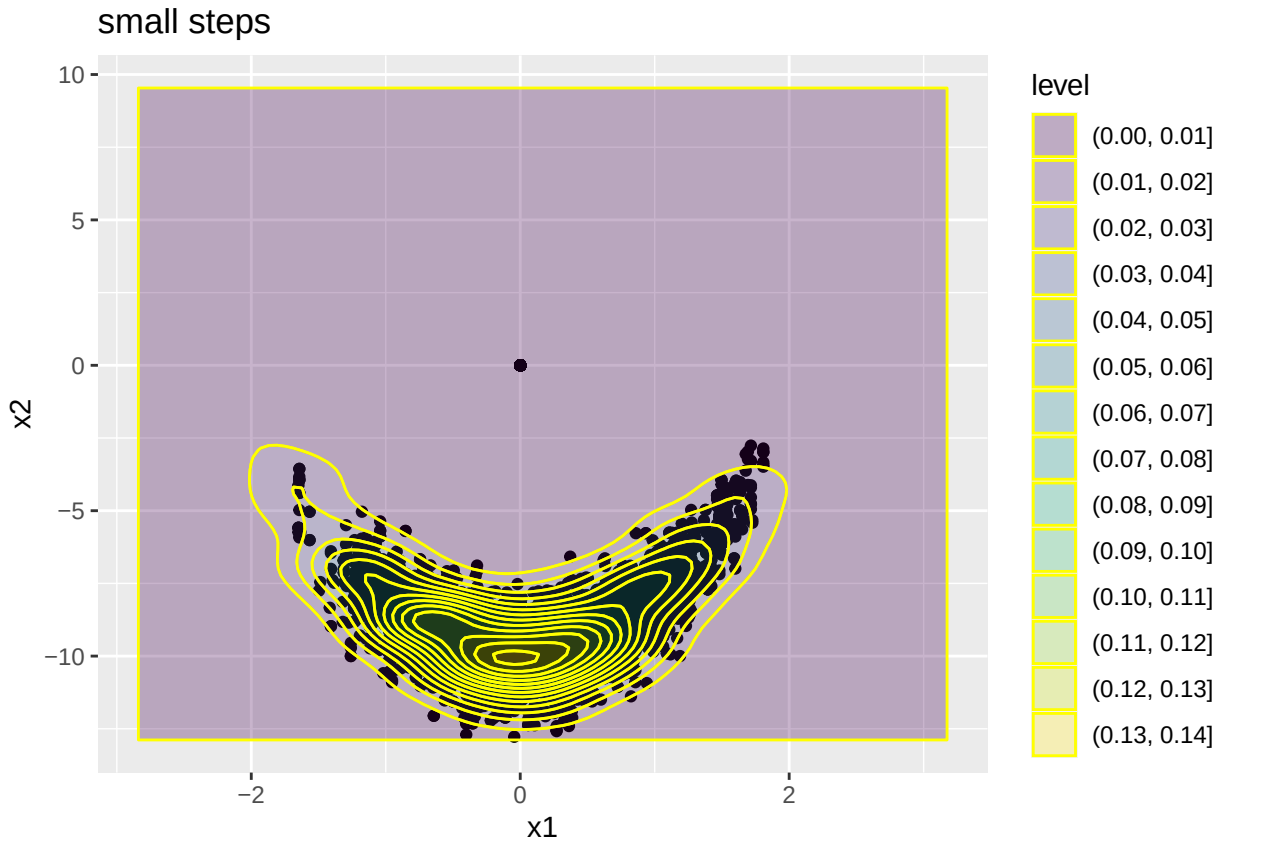




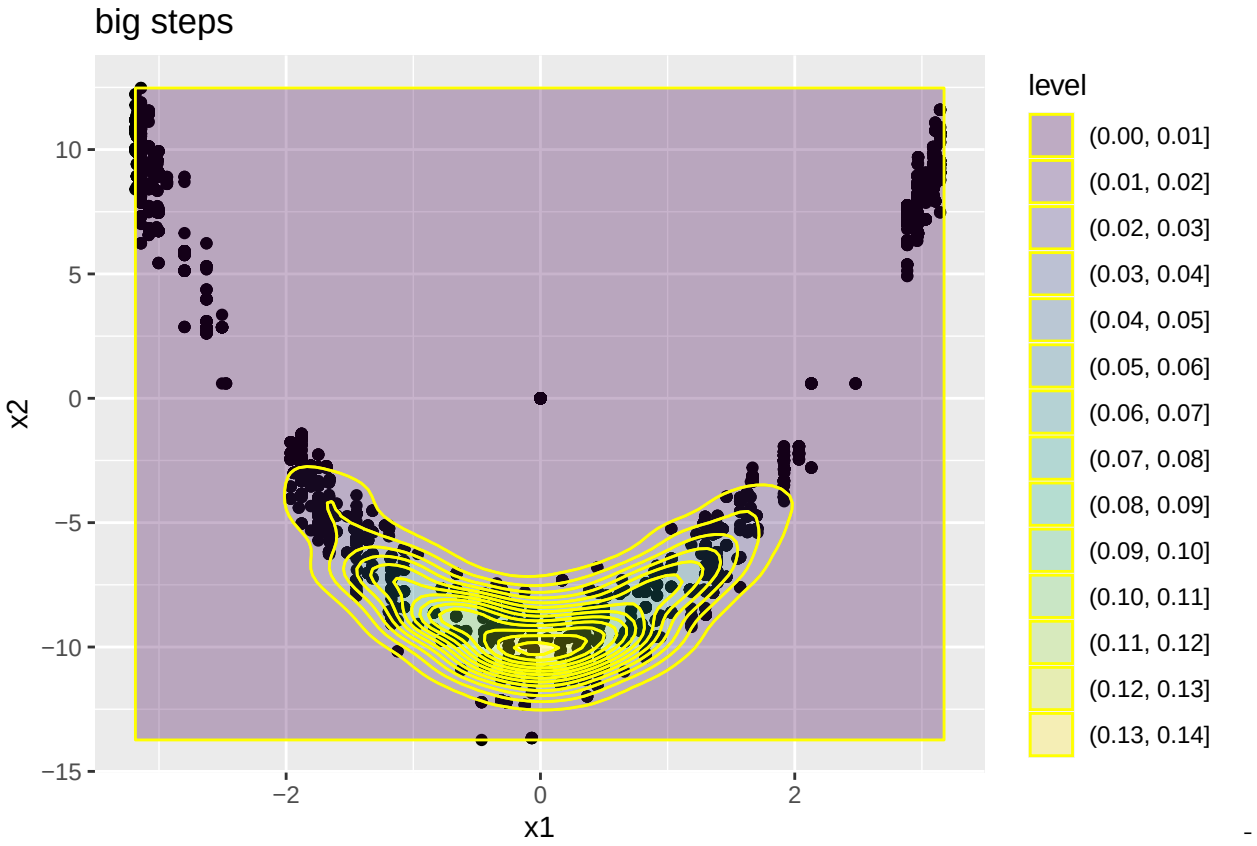





```
set.seed(4555)
post33 = as.data.frame(post3)
post44 = as.data.frame(post4)

library(ggplot2)

ggplot(post33) + geom_point(aes(x1, x2)) +
  labs(title = "small steps") +
  geom_density_2d_filled(mapping = aes(x = x11, y = x22),
                         data = bananadd, alpha = 0.3, color = "yellow")
```




```
ggplot(post44) + geom_point(aes(x1, x2)) +
  labs(title = "big steps") +
  geom_density_2d_filled(mapping = aes(x = x11, y = x22),
                         data = bananadd, alpha = 0.3, color = "yellow")
```



Same as part a), at large step size the acceptance rate is very low. When step size is small, the acceptance rate is increased and we see a good mix. For lower step sizes, with n_iter = 5000 and burn the first 1000 observations, we see a convergence to the target distribution.

c) Gibbs sampling

- Need to simulate from the full conditional distributions.

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

```
update_x2 = function(x1) {
  mu_1 = 2 * x1 ^ 2 - 10
  rnorm(n = 1, mean = mu_1, sd = 1)
}
```

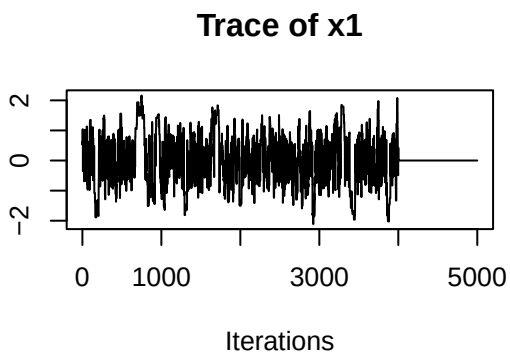
$$p(x_1|x_2) \propto \exp[-\frac{1}{2}(4x_1^4 - 4x_1^2x_2 - 41x_1^2)]$$

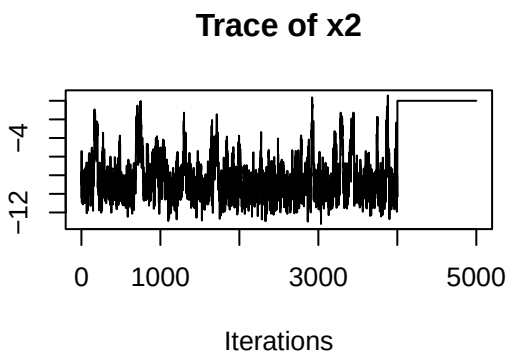
```
update_x1 = function(x1, sd) {
  #mu_1 = exp((-1 / 2) * 4 * x1_0^4 - 4 * x1_0^2 * x2 - 41 * x1_0^2)
  rnorm(n = 1, mean = x1, sd = sd)
}
```

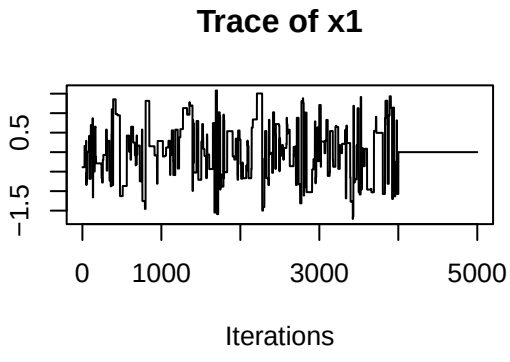
```r
gibbs = function(n_iter, burn, x1, x2, cand_sd) {

  ## initialize
  x1_out = numeric(n_iter)
  x2_out = numeric(n_iter)

  ## sampler
  for (i in 1:n_iter) {

     #lgbanana(x1 = x1_now, x2 = x2_cand)

    x1_now = update_x1(x1 = x1, sd = cand_sd)
    alpha = lgbanana(x1_now, x2) - lgbanana(x1, x2)

    u = log(runif(1))
    if(u < alpha) {
      x1 = x1_now
    }

    x2 = update_x2(x1 = x1)

    if(i > burn) {
      i1 = i - burn
      x2_out[i1] = x2 #save both x1 x2
      x1_out[i1] = x1
    }
  }
  cbind(x1 = x1_out, x2 = x2_out)
}
```

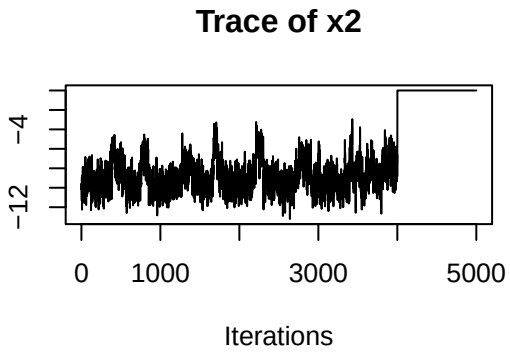
```r
set.seed(5366)

gpost1 = gibbs(n_iter = 5000, burn = 1000, x1 = 1, x2 = 5, cand_sd = 1)
gpost2 = gibbs(n_iter = 5000, burn = 1000, x1 = 1, x2 = 5, cand_sd = 10)
```

```r
par(mfrow=c(2,2))
traceplot(as.mcmc(gpost1))
traceplot(as.mcmc(gpost2))
```

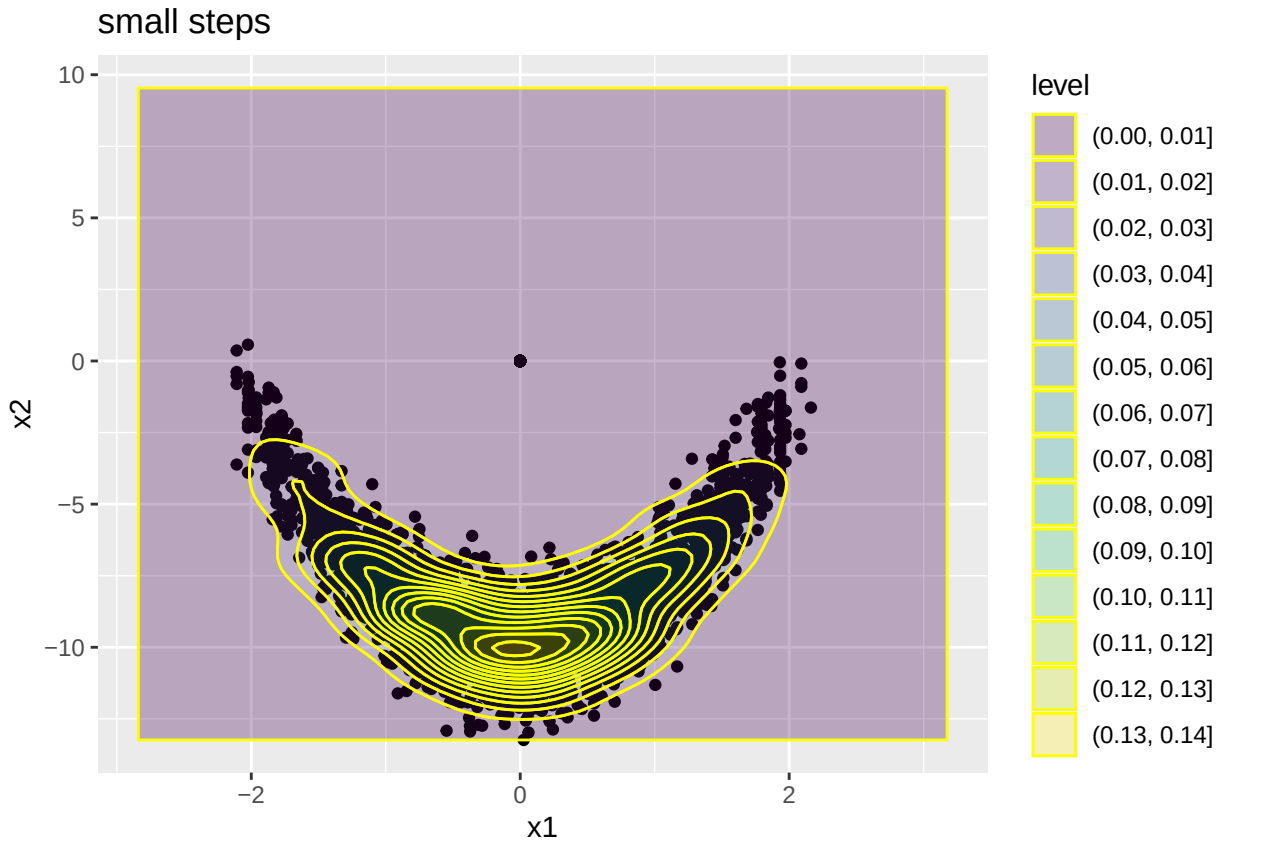




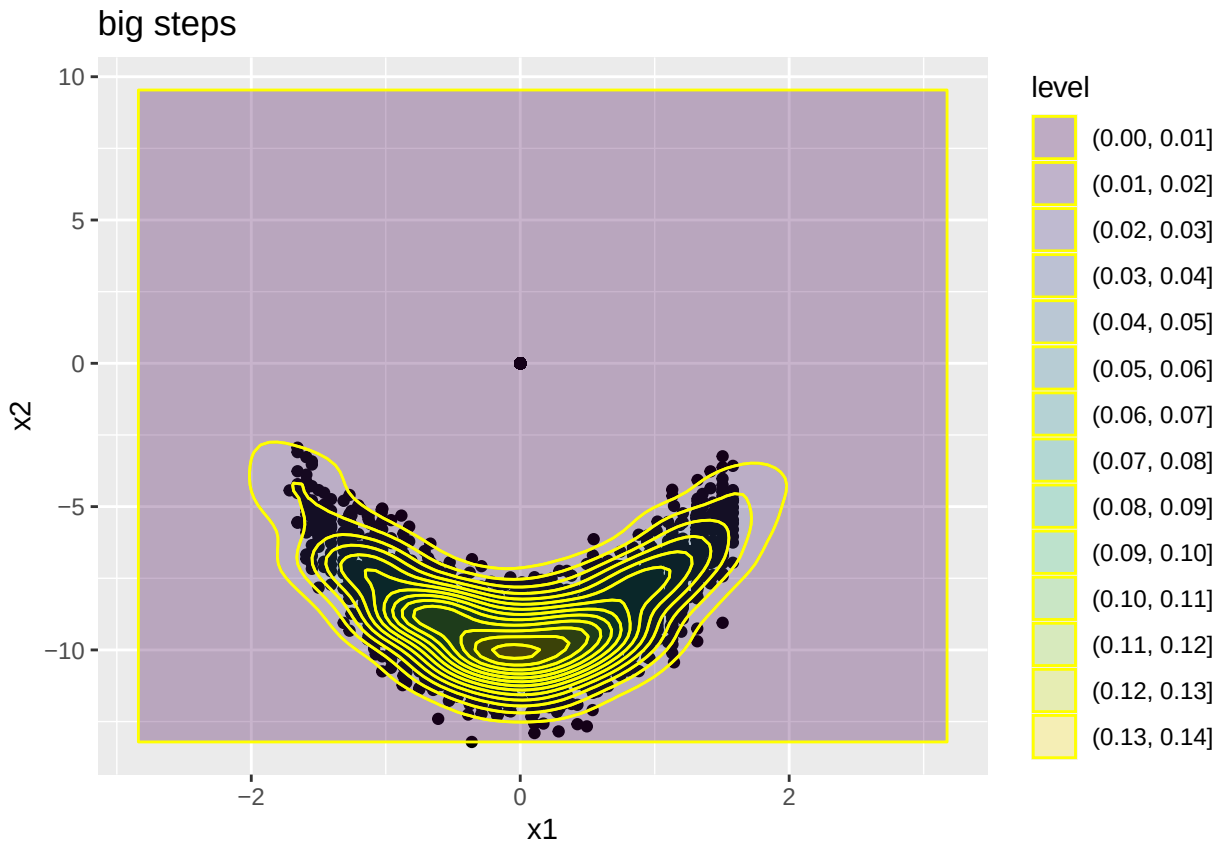





```
gpost11 = as.data.frame(gpost1)
gpost22 = as.data.frame(gpost2)

ggplot(gpost11) + geom_point(aes(x1, x2)) +
  labs(title = "small steps") +
  geom_density_2d_filled(mapping = aes(x = x11, y = x22),
                         data = bananadd, alpha = 0.3, color = "yellow")
```




```
ggplot(gpost22) + geom_point(aes(x1, x2)) +
  labs(title = "big steps") +
  geom_density_2d_filled(mapping = aes(x = x11, y = x22),
                         data = bananadd, alpha = 0.3, color = "yellow")
```



- From above results, for small step, when n_iter = 5000 and burn the first 1000 samples, the algorithm converges to desire distribution. And for larger step size in the Gibbs sampler the result also looks okay.

d) Use the algorithms in (a, b, c) to estimate the following.

- 1.$E(x_1^2)$

```
x1_2 = list(ba = bananadd$x1^2,
            mh = post11$x1^2,
            mhc = post33$x1^2,
            gib = gpost11$x1^2)

means = lapply(x1_2, mean)
vars = lapply(x1_2, var)
x1_2 = cbind(means, vars)
x1_2
```

```
##      means     vars
## ba  0.939795  1.67292
## mh  0.4775145 0.4408205
## mhc 0.4140598 0.3307716
## gib 0.6358012 0.8031922
```

- All methods shows similar estimates for the expected value of x_1^2. For component wise Metropolis Hastings algorithm shows the lowest variance value and it's variance value is very close to non-component wise Metropolis Hastings. The other two's variance values are not that good compare to Metropolis Hastings sampler.
- 2.$E(x_2)$.

```
x2 = list(ba = bananadd$x2,
          mh = post11$x2,
          mhc = post33$x2,
          gib = gpost11$x2)

means = lapply(x2, mean)
vars = lapply(x2, var)
x2 = cbind(means, vars)
x2
```

```
##     means     vars
## ba  -8.096034 7.613947
## mh  -7.081828 14.61644
## mhc -7.185003 14.63543
## gib -6.73792  15.02205
```

- All methods shows similar estimates for the expected value of x_2. For component wise Metropolis Hastings algorithm shows the highest variance value and it's variance value is very similar to non-component wise Metropolis Hastings and gibbs sampler.
- 3.$P(x_1 + x_2 > 0)$.

```
p = list(ba = as.numeric(bananadd$x1 + bananadd$x2 > 0),
         mh = as.numeric(post11$x1 + post11$x2 > 0),
         mhc = as.numeric(post33$x1 + post33$x2 > 0),
         gib = as.numeric(gpost11$x1 + gpost11$x2 > 0))
means = lapply(p, mean)
vars = lapply(p, var)
p = cbind(means, vars)
p
```

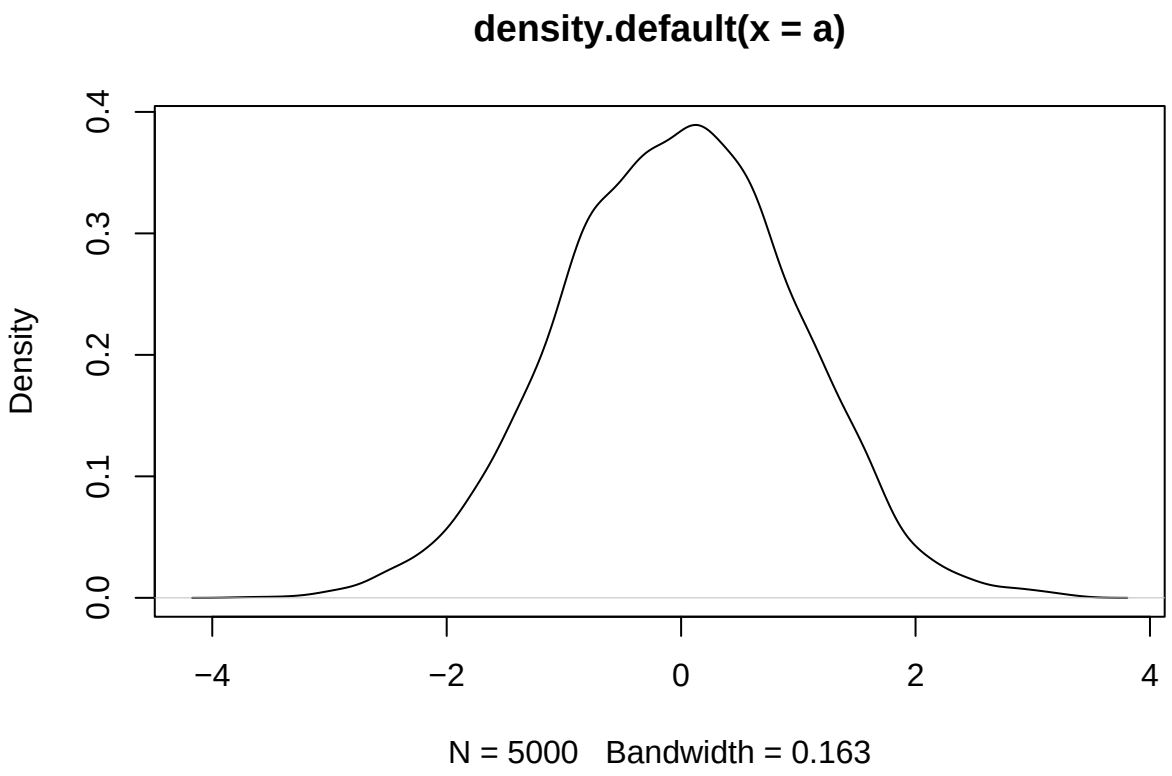
```
##     means  vars
## ba  0.022  0.02153754
## mh  0      0
## mhc 0      0
## gib 0.0042 0.004183197
```

- All methods shows similar estimates for the expected value of $P(x_1 + x_2 > 0)$. All the values are equal or approximately equal to 0.

Question 2

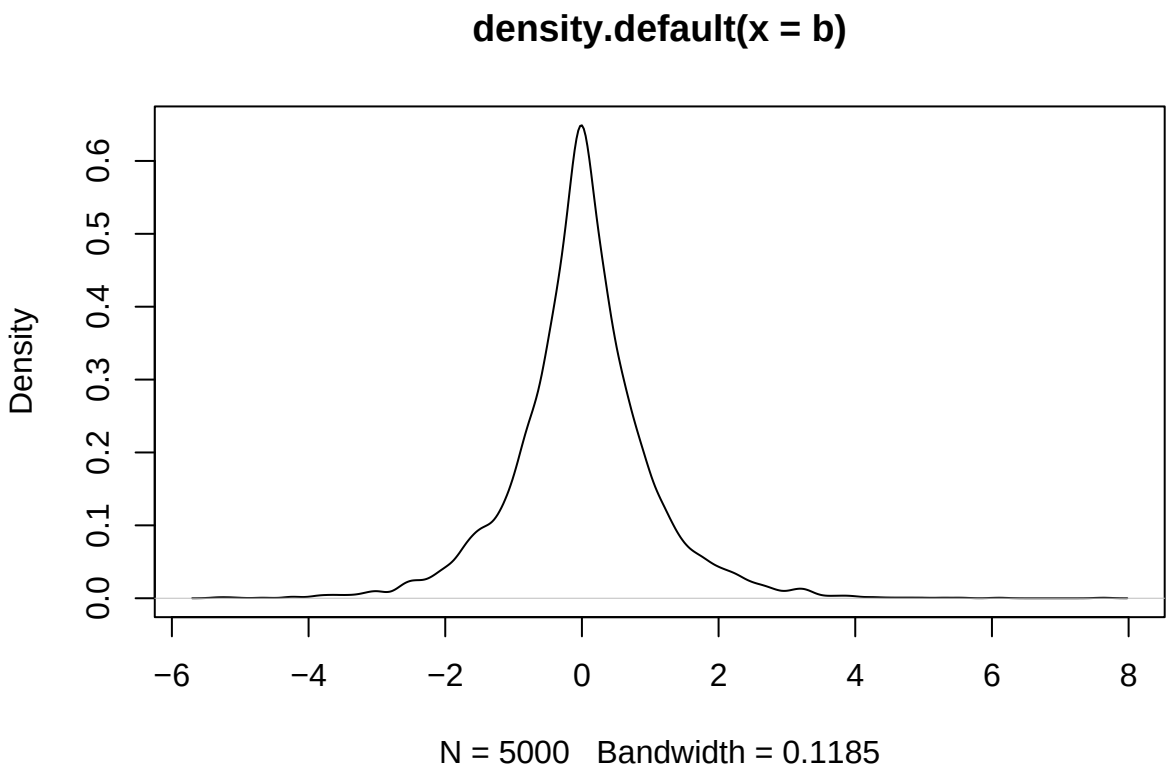
a) β marginal

```r
a = rnorm(5000, 0, 1)
plot(density(a))
```



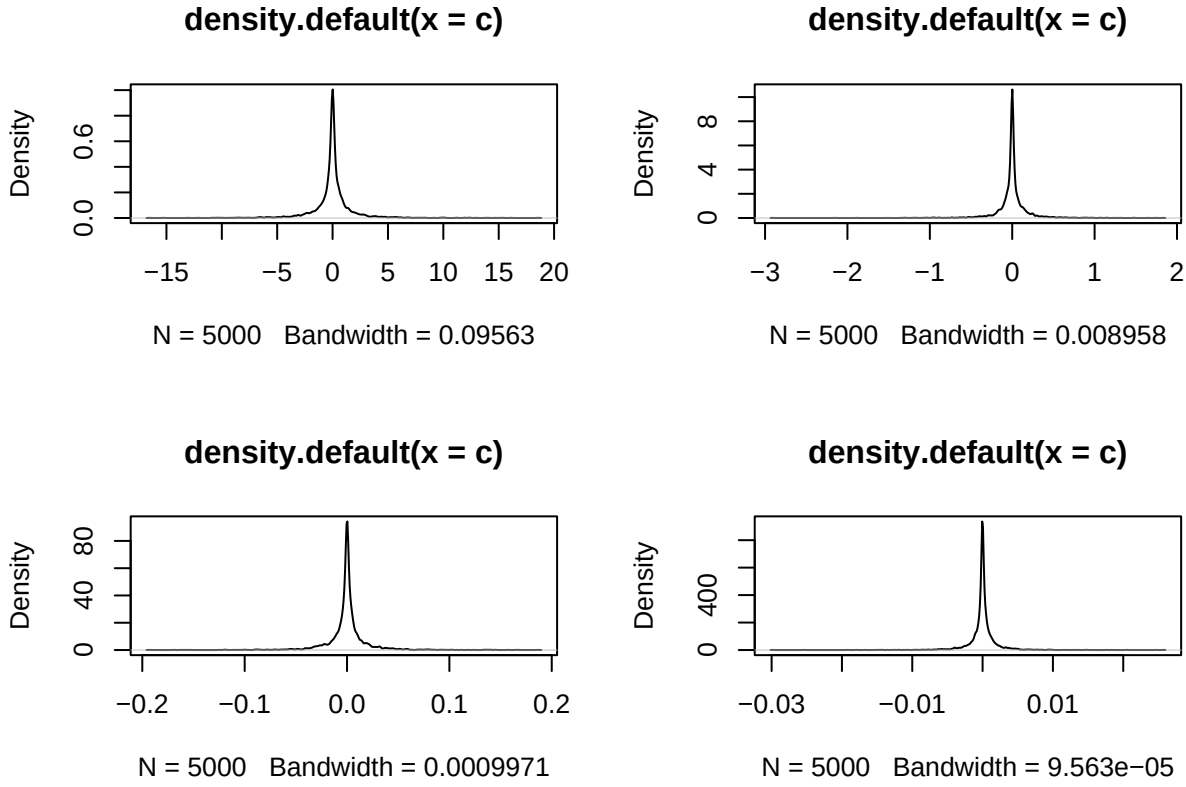
b) β marginal when $\lambda^2 = 2$

```r
lambda2 = 2

tau2 = rgamma(5000, 1, rate = lambda2 / 2)
b = rnorm(5000, 0, sd = sqrt(tau2))
plot(density(b))
```



c) β marginal when $\lambda \sim \Gamma^{-1}(a, b)$

```
set.seed(666)
par(mfrow=c(2,2))
rate = c(1,10,100,1000)
for(b in rate) {
  lambda = 1 / rgamma(5000, 1, b)
  tau2 = rgamma(5000, 1, rate = lambda^2 / 2)
  c = rnorm(5000, 0, sd = sqrt(tau2))
  plot(density(c))
}
```



d)

$$j = 1, ..., p$$

$$p(\beta_j, \tau_j^2, \sigma^2, \lambda^2 | y_1, ..., y_p) \propto p(y_1, ..., y_p | \beta_j, \tau_j^2, \sigma^2, \lambda^2) p(\beta_j | \tau_j^2) p(\tau_j^2 | \lambda^2) p(\lambda^2) p(\sigma^2)$$

$$\begin{aligned} p(\sigma^2 | y_j) &\propto p(y_j | \beta_j, \tau_j^2, \beta, \lambda^2) \\ &\propto p(y_j | \beta_j, \tau_j^2, \sigma^2, \lambda^2) p(\sigma^2) \\ &\propto (\sigma^2)^{\frac{-n}{2}} \exp[-\frac{1}{2\sigma^2}(y_j - X\beta_j)^T (y_j - X\beta_j)] (\sigma^2)^{-1} \exp -(\frac{1}{\sigma^2}) \\ &= \sigma^{2^{(-\frac{n}{2})-1}} \exp[-\frac{1}{\sigma^2}(1 + \frac{1}{2}(y_j - X\beta_j)^T (y_j - X\beta_j)] \\ &which\ follows\ \sim \Gamma^{-1}(\frac{n}{2}, 1 + \frac{1}{2}(y_j - X\beta_j)^T (y_j - X\beta_j)) \end{aligned}$$

$$\begin{aligned} p(\beta_j | y_j) &\propto p(y_j | \beta_j, \tau_j^2, \sigma^2, \lambda^2)(\beta_j | \tau_j^2) \\ &\propto \exp[-\frac{1}{2\sigma^2}(y_j - X\beta_j)^T (y_j - X\beta_j)] \exp[-\frac{1}{2}\sigma^{-1}\beta_j^T \beta_j],\ \ for\ \ \sigma = diag(\tau_i^2) \\ &= \exp[-\frac{1}{2\sigma^2}(y_j^T y_j - 2y_j (X\beta_j)^T + (X\beta_j)^T (X\beta_j)) - \frac{1}{2}\beta_j^T \sigma^{-1} \beta_j) \\ &\propto \exp[-\frac{1}{2}(\beta_j^T (\frac{X^T X}{\sigma^2} + \sigma^{-1})\beta_j) - \frac{1}{2\sigma^2} 2(X\beta_j)^T y_j] \\ &which\ follows\ \sim N([\frac{X^T X}{\sigma^2} + \sigma^{-1}]^{-1} \frac{X^T y_j}{\sigma^2}, [\frac{X^T X}{\sigma^2} + \sigma^{-1}]^{-1}) \end{aligned}$$

e)

- After find out the full conditional distribution from part d, we can use gibbs sampling to draw sample from one for the other iteratively.

```
update_beta = function(X, y, sig2_0, tau2_0) {
  head = t(X) %*% X + tau2_0 / sig2_0
  var_1 = solve(head)
  mu_1 = solve(head, t(X) %*% y)
  rmvnorm(n = 1, mean = mu_1, sigma = var_1) %>%
    t()
}
```

```
update_sig2 = function(n, y, X, beta_0) {
  shape_1 = n / 2 + 0.1
  rate_1 = (t(y - X %*% beta_0) %*% (y - X %*% beta_0) * 0.5) + 10
  out_sig2 = rgamma(1, shape = shape_1, rate = rate_1)
  1 / out_sig2
}
```

```
update_lamb2 = function(a, b, tau2_0) {
 rgamma(1, shape = a, rate = sum(tau2_0) * 0.5 + b)
}
```

```
#update_tau2 = function(lambda2_0, beta_0, p) {
 #   1 / rinvgauss(1, mean = sqrt(lambda2_0) / sqrt(beta_0^2), shape = lambda2_0)
#}
```

```
gmh = function(y, X, n_iter, burn, lambda, a = 1, b = 1) {

  ## initialize 1
  xx = t(X) %*% X
  xy = t(X) %*% y
  n = length(y)
  p = ncol(X)

  betahat = solve(xx, xy)
  sig2hat = (n - 1) / sum((y - X %*% betahat)^2)
  lambda_now = lambda

  if(lambda_now < 0) {
    lambda_now = 0.1
  }
  tau2_0 = rep(0, 10)
  tau2 = diag(11)

  ## initialize 2
  beta_out = matrix(NA, nrow = n_iter, 11)
  sig2_out = numeric(n_iter)

    ## iterate
  for(i in 1:n_iter) {
```

```
    for(j in 1:p) {

      # draw for tau
      tau2_0 = 1 / rinvgauss(1, lambda_now / abs(betahat[j]),
                             shape = lambda_now^2)
      tau2[j, j] = tau2_0
    }

    # draw for beta
    beta_now = update_beta(X = X, y = y, sig2_0 = sig2hat, tau2_0 = tau2_0)
    # draw for sigma
    sig2_now = update_sig2(n = n, y = y, X = X, beta_0 = beta_now)

    # draw for lambda
    if(lambda_now < 0){
      lambda_now = update_lamb2(a, b, tau2_0 = tau2_0)
      #lambda_now = lambda_cand
    }

    if(i > burn){
      i1 = i - burn
      beta_out[i1, ] = beta_now # save this iteration's value of beta
      sig2_out[i1] = sig2_now # save this iteration's value of sigma
    }
}
  list(beta = beta_out, sigma = sqrt(sig2_out))
}
```

```
set.seed(888)
data("diabetes")

y = diabetes$y
X = cbind(rep(1, length(diabetes$x)), cbind(diabetes$x))

postgmh = gmh(y = y, X = X, n_iter = 5000, burn = 1000, lambda = 1 )
```

```
postgmh11 = apply(postgmh$beta, 2,
                  function(x){quantile(x, c(0,0.5,1), na.rm = T)})
postgmh22 = postgmh11[2,]

data = as.matrix(diabetes$x)
data_glm = glmnet(data, y)
data_coeff = coef(data_glm, s = min(data_glm$lambda))
both = tibble("Gibb with MH" = postgmh22, "Glmnet" = matrix(data_coeff))

tibble::tibble(both)
```

```
## # A tibble: 11 x 2
##    `Gibb with MH` Glmnet[,1]
##             <dbl>      <dbl>
##  1          152.      152.
##  2          -17.6      -9.22
```

```
##  3         -279.     -239.
##  4          492.      520.
##  5          308.      324.
##  6          728.     -716.
##  7         -625.      418.
##  8         -791.       65.4
##  9         -258.      165.
## 10          222.      724.
## 11           67.2      67.5
```

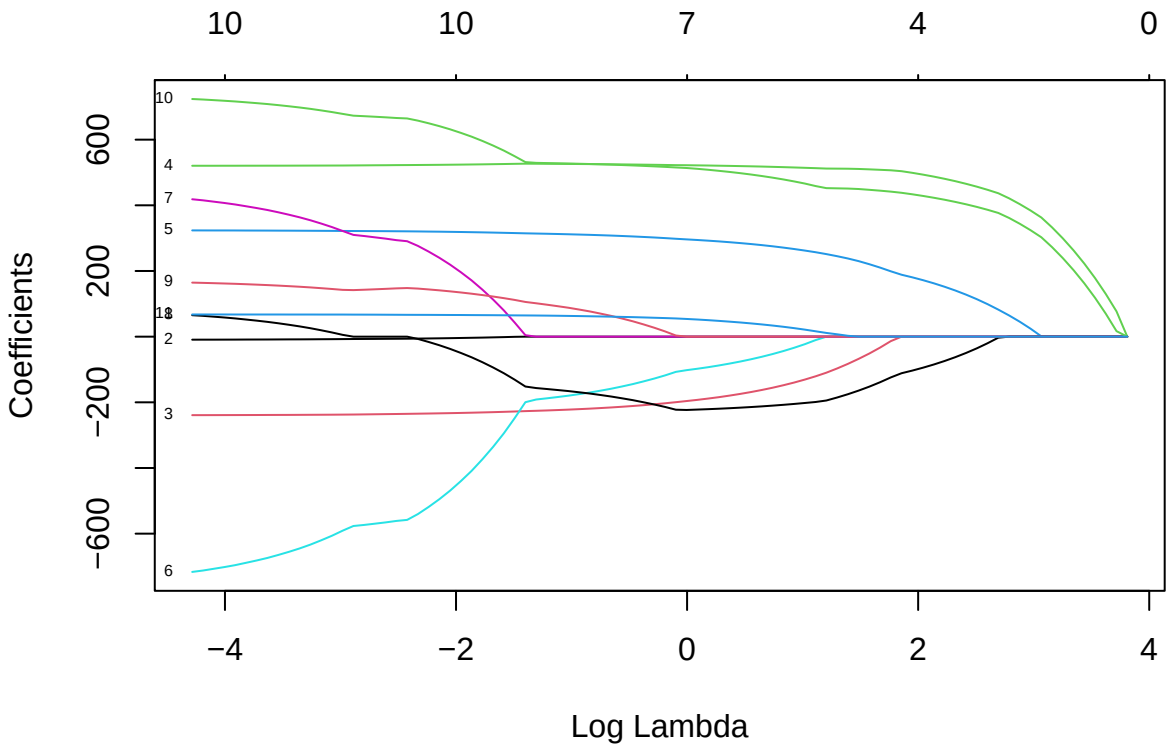
- From the results above, there still got a few differences between glmnet and bayesian lasso for $\lambda = 1$.

f)

```
set.seed(7888)
lambdas = seq(from = -5, to = 5, length.out = 11)

for(i in 1:length(lambdas)) {
  outpts = gmh(y = y, X = X, n_iter = 5000,
               burn = 1000, lambda = exp(lambdas[i]))
  }
```

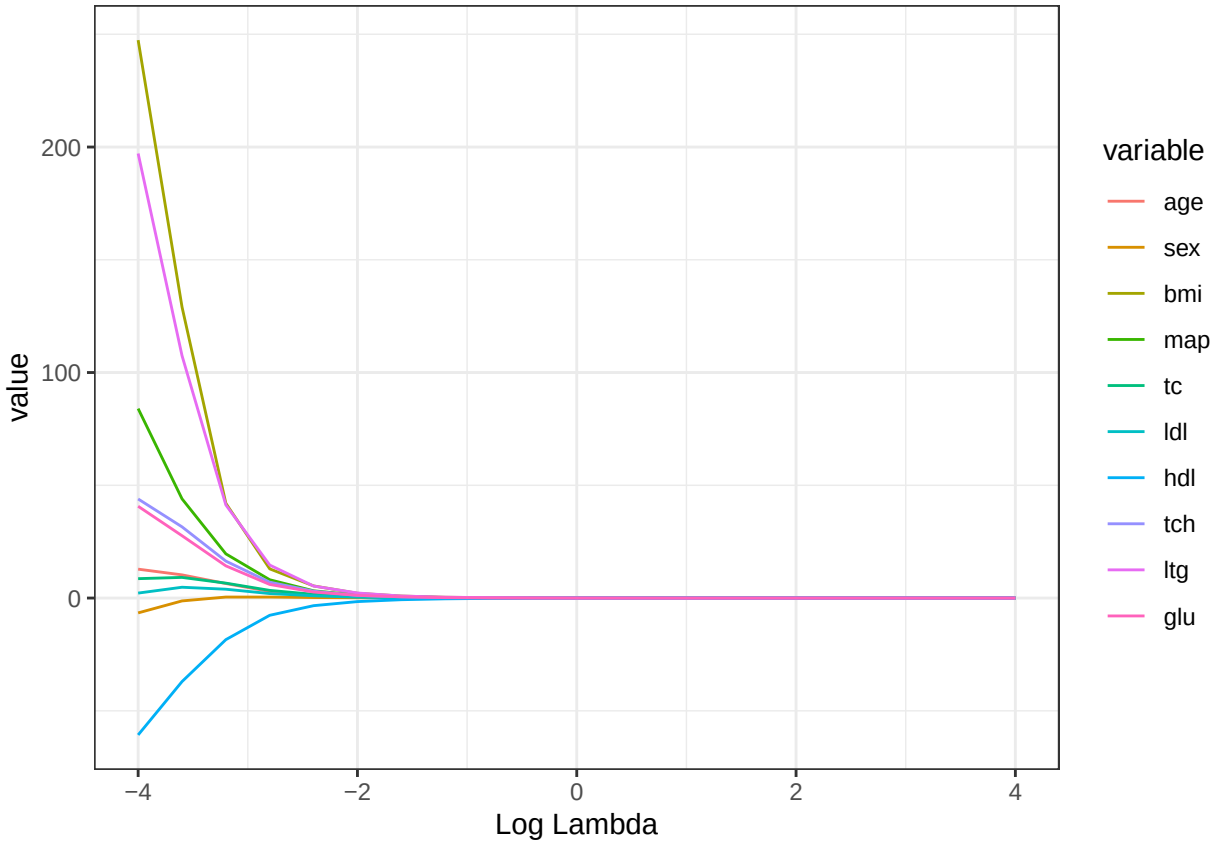
- Glmnet plot

```
glmnet1 = glmnet(X, y, alpha = 1)
plot(glmnet1, xvar = "lambda", label = TRUE)
```



- Bayesian Lasso plot

```
beta = lapply(outputs, function(x){x[[1]]})
beta = lapply(beta, function(x){apply(x, 2, median)})
df = data.frame(Reduce("rbind", beta))
colnames(df) = colnames(XX)
df$lambda = rep(seq(-4, 4, 0.4))
df = melt(df, id.vars = names(df)[11])
ggplot(df, aes(lambda, value, color = variable)) +
  geom_line()+ labs(x = "Log Lambda") + theme_bw()
```



- From above plots, for the diabetes data, fix λ and produce a regularization path for adaptive Bayesian Lasso is more smoother than Glmnet lasso.

g)

```
set.seed(667788)

newab = data.frame(a = rep(1, 9), b = seq(1.5, 5.5, 0.5))

output = apply(newab, 1, function(newab){
  gmh(n_iter = 5000, burn = 1000, a = newab[1],
      b = newab[2], X = X, y = y, lambda = 1)
```

```
  })

ab = lapply(outputs, function(x){x[[1]]})
ab = lapply(ab, function(x){apply(x,2,median)})

data = do.call("cbind", ab)
```

```
tibble::tibble(data)
```

```
## # A tibble: 10 x 9
##        V1     V2      V3     V4     V5     V6      V7      V8       V9
##     <dbl>  <dbl>   <dbl>  <dbl>  <dbl>  <dbl>   <dbl>   <dbl>    <dbl>
##  1  12.8   10.3    6.44   2.93   1.47   0.643  0.322   0.158   0.0731
##  2  -6.59  -1.27   0.438  0.445  0.232  0.128  0.0583  0.0272  0.0189
##  3 247.   129.    42.0   12.9    5.38   2.11   0.893   0.361   0.154
##  4  84.0   44.0   19.6    8.11   3.22   1.30   0.550   0.230   0.0798
##  5   8.60   9.16   6.61   3.39   1.63   0.816  0.394   0.200   0.113
##  6   2.20   4.79   3.91   1.99   1.01   0.369  0.114   0.0383 -0.00507
##  7 -60.7  -37.1  -18.4   -7.64  -3.38  -1.57  -0.736  -0.343  -0.155
##  8  43.9   31.6   16.4    7.03   3.04   1.18   0.541   0.217   0.0939
##  9 197.   108.    41.3   14.6    5.33   2.23   0.905   0.380   0.165
## 10  40.7   27.7   14.3    6.04   2.66   1.20   0.529   0.185   0.0744
```