Bayesian Learning (732A91) Lab3

Anubhav Dikshit(anudi287) and Lennart Schilling (lensc874)

09 May, 2019

Contents

Question 1: Normal model, mixture of normal model with semi-conjugate prior.

The data rainfall.dat consist of daily records, from the beginning of 1948 to the end of 1983, of precipitation (rain or snow in units of 0.01 inch, and records of zero precipitation are excluded) at Snoqualmie Falls, Washington. Analyze the data using the following two models.

a) Normal Model
Assume the daily precipitation $y_1, y_2, y_3 ...., y_n$ are independent normally distributed, $y_1, y_2, ..., y_n | \mu, \sigma^2 \sim N(\mu, \sigma^2)$ where both μ and σ^2 are unknown. Let $\mu \sim N(\mu_0, \tau_0^2)$ independently of $\sigma^2 \sim Inv - \chi^2(\nu_0, \sigma_0^2)$

i) Implement a Gibbs Sampler code that simulates from the joint posterior $p(\mu, \sigma^2 | y_1, y_2, ...y_n)$. The full conditional posteriors are given on the slides from Lecture 7.
ii) Analyze the daily precipitation using your Gibbs sampler in (a)-i. Evaluate the convergence of the Gibbs sampler by suitable graphical methods, for example by plotting the trajectories of the sampled Markov chains.

Full conditional posteriors are given by:

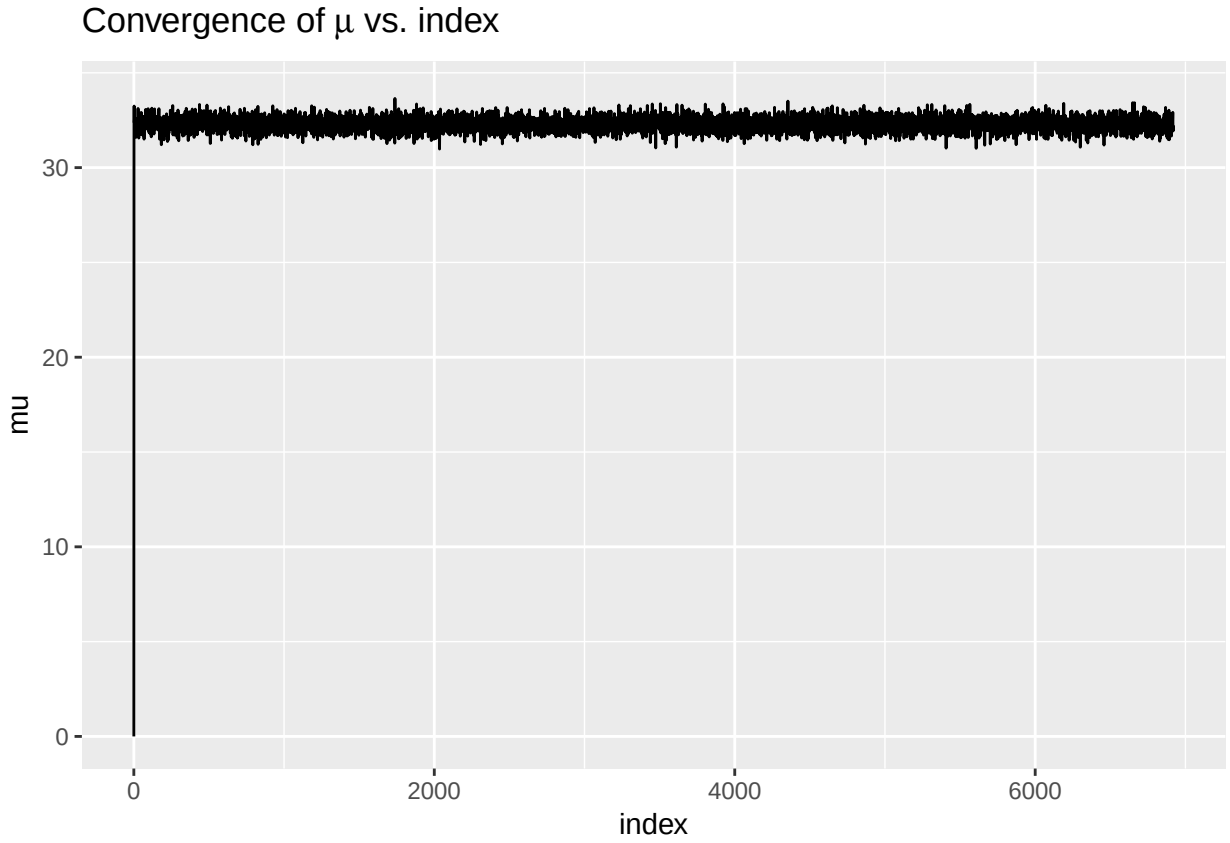
$$\mu | \sigma^2, x \sim N(\mu_n, \tau_n^2)$$
$$\sigma^2 | \mu, x \sim Inv - \chi^2(\nu_n, \frac{\nu_0 \sigma_0^2 + \sum_{i=1}^{n} (x_i - \mu)^2}{n + \nu_0})$$

Where with μ_n and τ_n^2 defined in the same way when σ^2 is unknown.

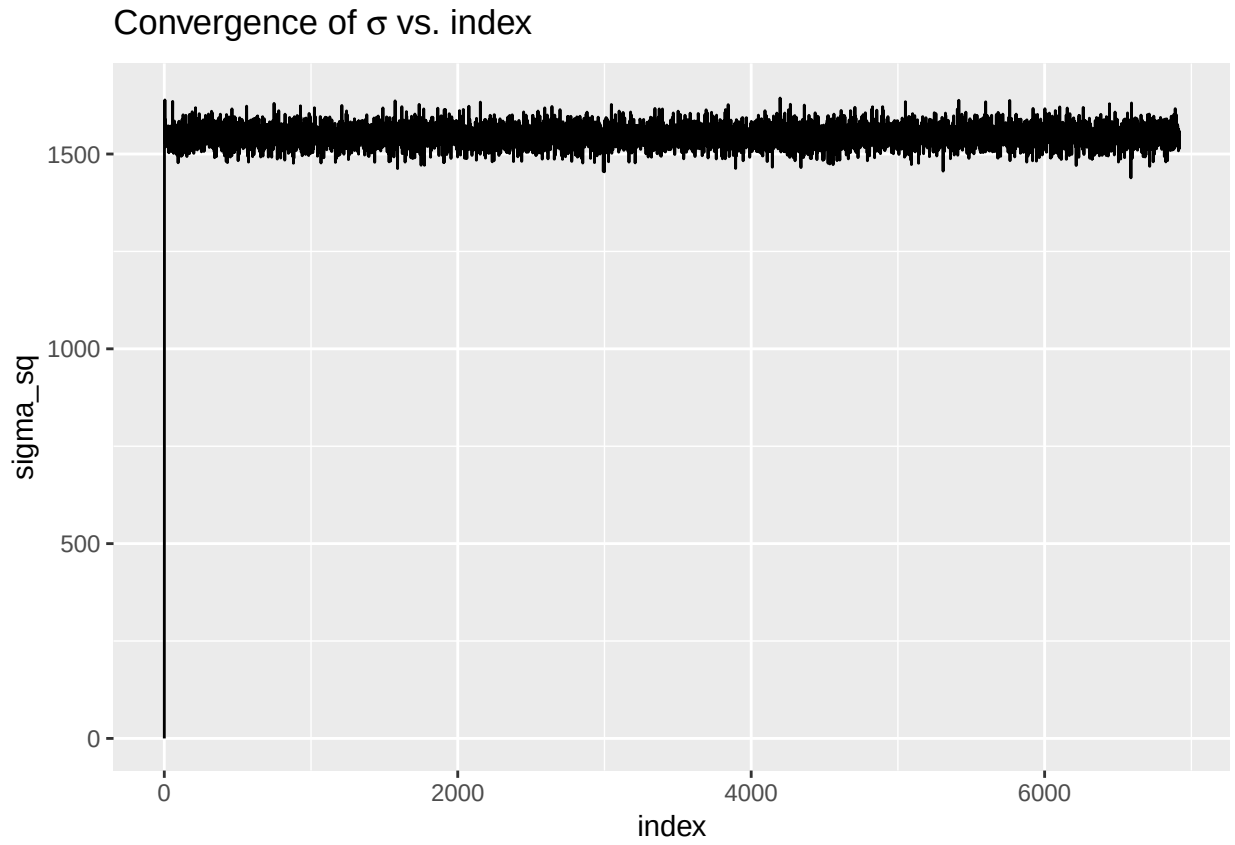
In the following, τ^2 can be used to simulate from the posterior $\sigma^2 | (y_1, ..., y_n); \mu \sim Inv - \chi^2(n, \tau^2)$. This will be done by first drawing $X \sim \chi^2(n)$. This drawed value will then be used within the formula $\sigma^2 = \frac{n\tau^2}{X}$ which is a draw from $Inv - \chi^2(n, \tau^2)$. This process will be repeated n times. The obtained values for σ^2 will be stored.

```
# Normal Model
set.seed(12345)
rain_data <-  read.table("rainfall.txt", quote="\"", comment.char="")
colnames(rain_data) <- c("rainfall")

n = length(rain_data$rainfall)
mu_0 = mean(rain_data$rainfall)
tau_sq = sum((log(rain_data$rainfall)-mu_0)^2)/n

# calculating tau_n_sq
tau_n_sq = c()
for (i in 1:n) {
  # Drawing x.
  x = rchisq(n = 1, df = n)
  # Calculating and storing tau_n_sq
  tau_n_sq[i] = (n*tau_sq)/x
}

# Using Full conditional posteriors

## using tau_n_sq and mu_0 we get estimate mu using \mu|\sigma^2,x \sim N(\mu_n, \tau_n^2)
mu <- c()
for (i in 1:n) {
```

```r
mu[i] <- rnorm(n=1, mean=mu_0, sd = sqrt(tau_n_sq[i]/n))
}

## Now sampling sigmas using \sigma^2|\mu,x \sim Inv-\chi^2(\nu_n,\frac{\nu_0\sigma^2_0+\sum_{i=1}^n (x

# Prior parameters for sigma (variance)
v0 = 1
sigma_sq0 = 1
v_n = v0 + n


sigma_sq <- c()
for (i in 1:n) {
sigma_sq[i] <- rinvchisq(n=1, v_n, (v0*sigma_sq0 + sum((rain_data$rainfall - mu[i])^2))/(n + v0))
}

## dataframe
gibbs_df <- cbind(mu,sigma_sq) %>% as.data.frame()
gibbs_df$index <- as.integer(row.names(gibbs_df))

## adding 0 values
temp <- c(mu=0,sigma_sq=0)
gibbs_df <- rbind(gibbs_df,temp)


## plots
ggplot(data=gibbs_df, aes(x=index, y=mu)) +
  geom_line() +
  ylim(0,33.9) +
  ggtitle(expression(paste("Convergence of ", mu,  " vs. index")))
```




```
ggplot(data=gibbs_df, aes(x=index, y=sigma_sq)) +
  geom_line() +
  ylim(0,1650) +
  ggtitle(expression(paste("Convergence of ", sigma,  " vs. index")))
```



b) Mixture normal model
Let us now instead assume that the daily precipitation $y_1, y_2, ..., y_n$ follow an iid two-component mixture of normals model:

$$p(y_i|\mu, \sigma^2, \pi) = \pi N(y_i|\mu_1, \sigma_1^2) + (1-\pi)N(y_i|\mu_2, \sigma_2^2)$$

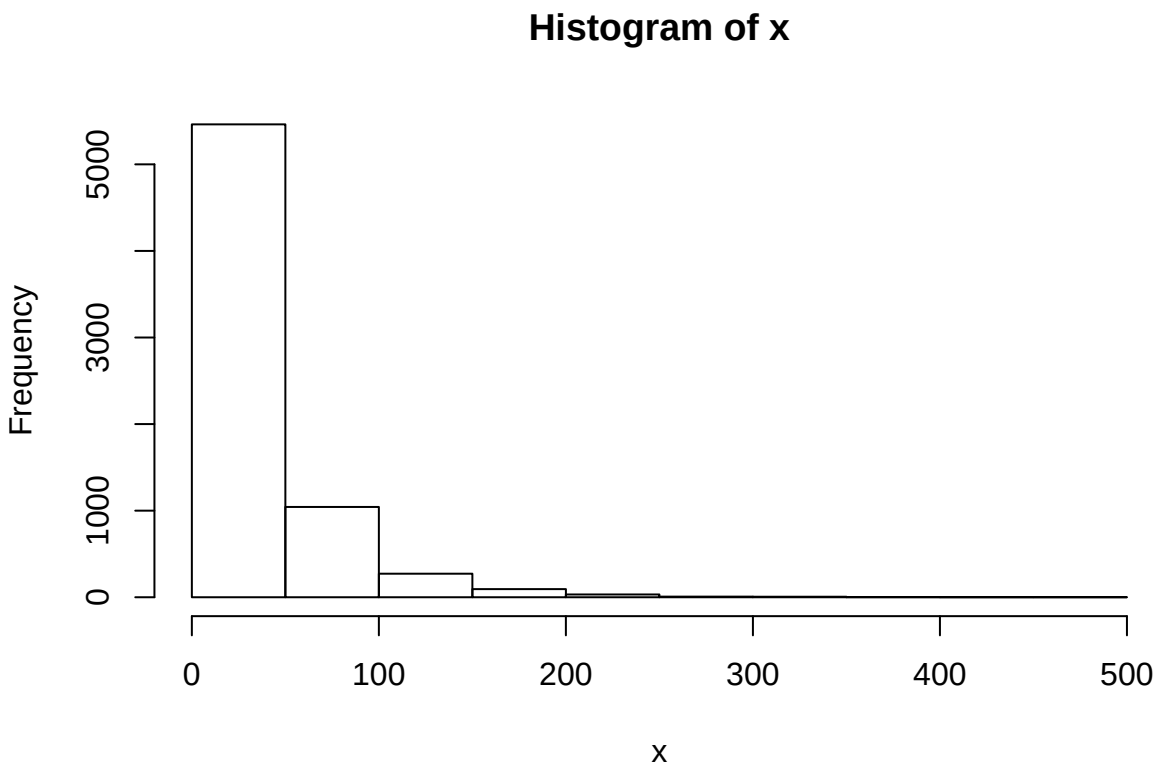
where

$$\mu = (\mu_1, \mu_2) \quad and \quad \sigma^2 = (\sigma_1^2, \sigma_2^2)$$

Use the Gibbs sampling data augmentation algorithm in NormalMixtureGibbs.R (available under Lecture 7 on the course page) to analyze the daily precipitation data. Set the prior hyperparameters suitably. Evaluate the convergence of the sampler.

```
set.seed(12345)
data = read.table("rainfall.txt", header=FALSE)[,1]
x = as.matrix(data)

# Model options
nComp <- 2 # Number of mixture components
# Prior options
alpha <- 10*rep(1,nComp) # Dirichlet(alpha)
muPrior <- rep(0,nComp) # Prior mean of theta
tau2Prior <- rep(10,nComp) # Prior std theta
sigma2_0 <- rep(var(x),nComp) # s20 (best guess of sigma2)
nu0 <- rep(4,nComp) # degrees of freedom for prior on sigma2
# MCMC options
nIter <- 100 # Number of Gibbs sampling draws
```

```
# Plotting options
plotFit <- FALSE
lineColors <- c("blue", "green", "magenta", 'yellow')
sleepTime <- 0.1 # Adding sleep time between iterations for plotting

################ END USER INPUT ###############
###### Defining a function that simulates from the
rScaledInvChi2 <- function(n, df, scale){
return((df*scale)/rchisq(n,df=df))
}

####### Defining a function that simulates from a Dirichlet distribution
rDirichlet <- function(param){
nCat <- length(param)
thetaDraws <- matrix(NA,nCat,1)
for (j in 1:nCat){
thetaDraws[j] <- rgamma(1,param[j],1)
}#Diving every column of ThetaDraws by the sum of the elements in that column.
thetaDraws = thetaDraws/sum(thetaDraws)
return(thetaDraws)
}

# Simple function that converts between two different
# representations of the mixture allocation
S2alloc <- function(S){
n <- dim(S)[1]
alloc <- rep(0,n)
for (i in 1:n){
alloc[i] <- which(S[i,] == 1)
}
return(alloc)
}

# Initial value for the MCMC
nObs <- length(x)
# nObs-by-nComp matrix with component allocations.
S <- t(rmultinom(nObs, size = 1 , prob = rep(1/nComp,nComp)))
theta <- quantile(x, probs = seq(0,1,length = nComp))
sigma2 <- rep(var(x),nComp)
probObsInComp <- rep(NA, nComp)
# Setting up the plot
xGrid <- seq(min(x)-1*apply(x,2,sd),max(x)+1*apply(x,2,sd),length = 100)
xGridMin <- min(xGrid)
xGridMax <- max(xGrid)
mixDensMean <- rep(0,length(xGrid))
effIterCount <- 0
ylim <- c(0,2*max(hist(x)$density))
```

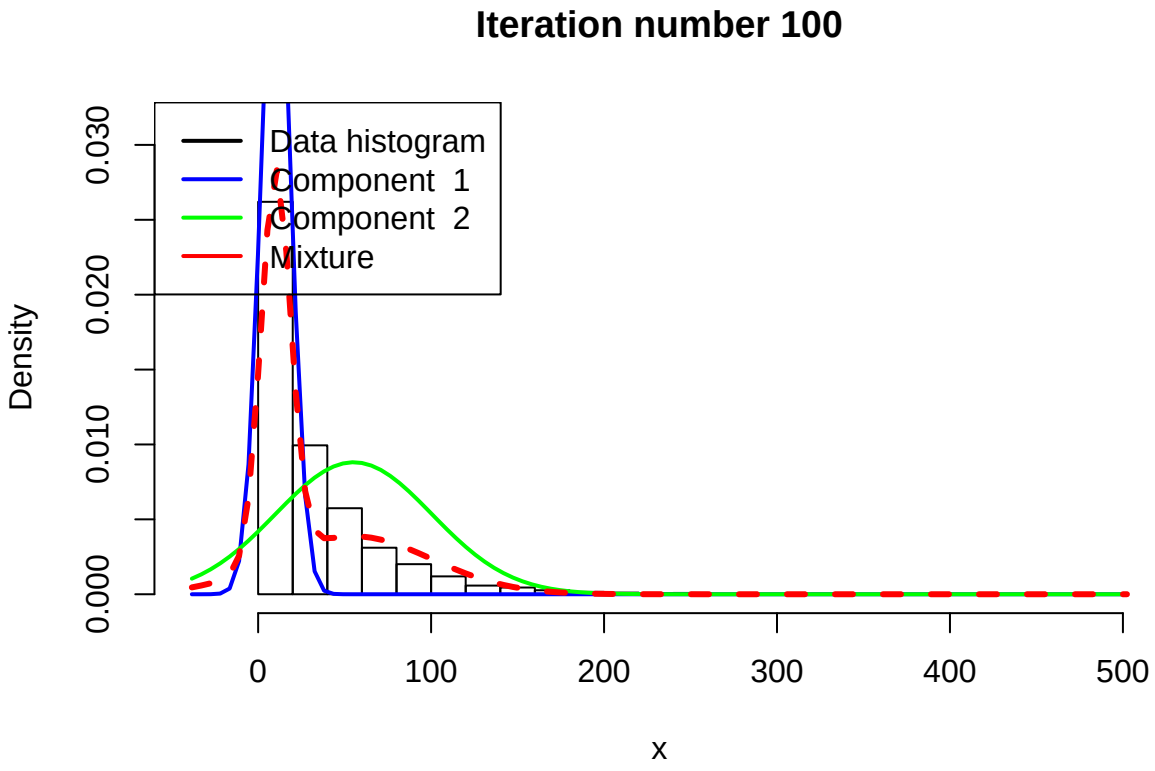



```
gibbs_thetas = matrix(0,nIter,2)
gibbs_sigmas = matrix(0,nIter,2)

for (k in 1:nIter){
# Just a function that converts between different representations
# of the group allocations
alloc <- S2alloc(S)
nAlloc <- colSums(S)
if(k == nIter){
message(paste('Iteration number:',k))
print(nAlloc)
}

# Update components probabilities
w <- rDirichlet(alpha + nAlloc)

# Update theta's
for (j in 1:nComp){
precPrior <- 1/tau2Prior[j]
precData <- nAlloc[j]/sigma2[j]
precPost <- precPrior + precData
wPrior <- precPrior/precPost
muPost <- wPrior*muPrior + (1-wPrior)*mean(x[alloc == j])
tau2Post <- 1/precPost
theta[j] <- rnorm(1, mean = muPost, sd = sqrt(tau2Post))
}
```

```
gibbs_thetas[k, ] = theta
# Update sigma2's
for (j in 1:nComp){
sigma2[j] <- rScaledInvChi2(1, df = nu0[j] + nAlloc[j],
scale = (nu0[j]*sigma2_0[j] +
sum((x[alloc == j] - theta[j])^2))/(nu0[j] + nAlloc[j]))
}

gibbs_sigmas[k,] = sigma2
# Update allocation
for (i in 1:nObs){
for (j in 1:nComp){
probObsInComp[j] <- w[j]*dnorm(x[i], mean = theta[j], sd = sqrt(sigma2[j]))
}
S[i,] <- t(rmultinom(1, size = 1 , prob = probObsInComp/sum(probObsInComp)))
}

# Printing the fitted density against data histogram
if ((k==nIter) && (k%%1 ==0)){
effIterCount <- effIterCount + 1
hist(x, breaks = 20, freq = FALSE, xlim = c(xGridMin,xGridMax),
main = paste("Iteration number",k), ylim = ylim)
mixDens <- rep(0,length(xGrid))
components <- c()

for (j in 1:nComp){
compDens <- dnorm(xGrid,theta[j],sd = sqrt(sigma2[j]))
mixDens <- mixDens + w[j]*compDens
lines(xGrid, compDens, type = "l", lwd = 2, col = lineColors[j])
components[j] <- paste("Component ",j)
}

mixDensMean <- ((effIterCount-1)*mixDensMean + mixDens)/effIterCount
lines(xGrid, mixDens, type = "l", lty = 2, lwd = 3, col = 'red')
legend("topleft", box.lty = 1, legend = c("Data histogram",components, 'Mixture'),
col = c("black",lineColors[1:nComp], 'red'), lwd = 2)
Sys.sleep(sleepTime)
}
}
```

```
## Iteration number: 100
```

```
## [1] 3791 3129
```




```
# Calculate mean of batches of 2 draws to visualize the
# auto correlation between sequential draws
t1 = c()
t2 = c()
s1 = c()
s2 = c()
for (i in 1:nIter){
if(i%%2 == 0){
t1 = c(t1, mean(gibbs_thetas[,1][i-1:i]))
t2 = c(t2, mean(gibbs_thetas[,2][i-1:i]))
s1 = c(s1, mean(gibbs_sigmas[,1][i-1:i]))
s2 = c(s2, mean(gibbs_sigmas[,2][i-1:i]))
}
}

# Plots displaying convergence of the Normal hyper
# parameters during sampling
pdf("3_1_2_gibbs_mixt.pdf")
par(mfrow=c(3,1))

# Plot comparison between kernel density, mixture of normals and a normal
# approximation
hist(x, breaks = 20,
cex=.1,
border="lightgray",
freq = FALSE,
```

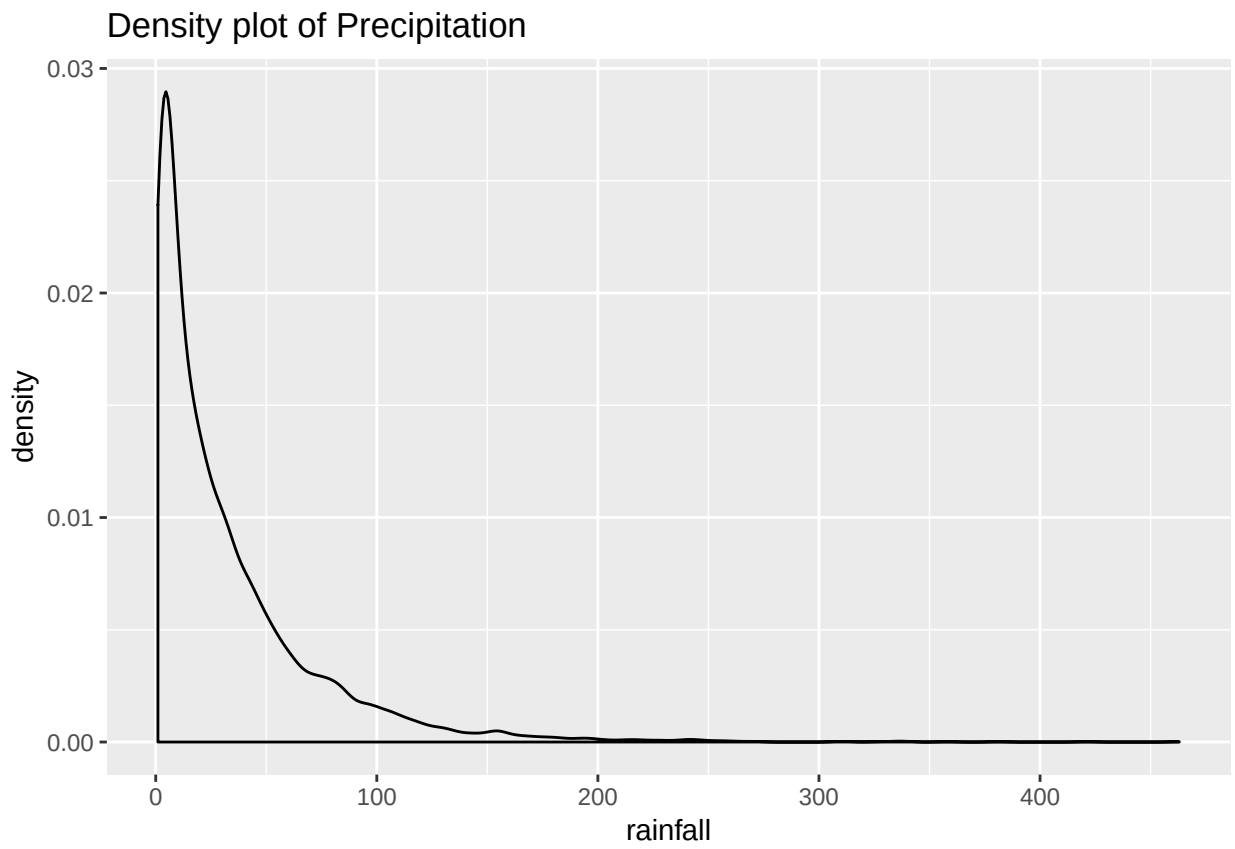
```
xlim = c(xGridMin,xGridMax),
xlab="Precipitation",
ylab="Density",
main = "Rainfall: Mixture of Normals")
lines(xGrid,
mixDensMean,
type = "l",
lwd = 2,
lty = 4,
col = "black")
lines(xGrid,dnorm(xGrid, mean = mean(x), sd = apply(x,2,sd)),type = "l",lwd = 2,col = "gray")
legend("topright",box.lty = 1,legend = c("Data histogram","Mixture density","Normal density"),col=c("li
# Plot the auto correlation (convergence) between draws of mu
min_t = min(c(min(t1), min(t2)))
max_t = max(c(max(t1), max(t2)))

plot(t1,type="l",ylim=c(min_t, max_t),cex=.1,lwd=2,
     main=expression(paste("Convergence of Gibbs Sampling ", "(", theta, ")", sep=" ")),
xlab="Batches of sequential draws",
ylab=expression(paste("Mean of seq. draws of ", theta, sep=" ")))
lines(t2, lwd=2, col="gray")
legend("topright",box.lty = 1,legend = c(expression(paste(theta, " (1)", sep=" ")),
                                         expression(paste(theta, " (2)", sep=" "))),col=c("black","gray
# Plot the auto correlation (convergence) between draws of sigma
min_s = min(c(min(s1), min(s2)))
max_s = max(c(max(s1), max(s2)))

plot(s1,type="l",ylim=c(min_s, max_s),cex=.1,lwd=2,main=expression(
paste("Convergence of Gibbs Sampling ", "(", sigma^2, ")", sep=" ")),
xlab="Batches of sequential draws",
ylab=expression(paste("Mean of seq. draws of ", sigma^2, sep=" ")))
lines(s2, lwd=2, col="gray")
legend("topright",box.lty = 1,legend = c(expression(paste(sigma^2, " (1)", sep=" ")),expression(paste(s

dev.off()
```

```
## pdf
##   2
```

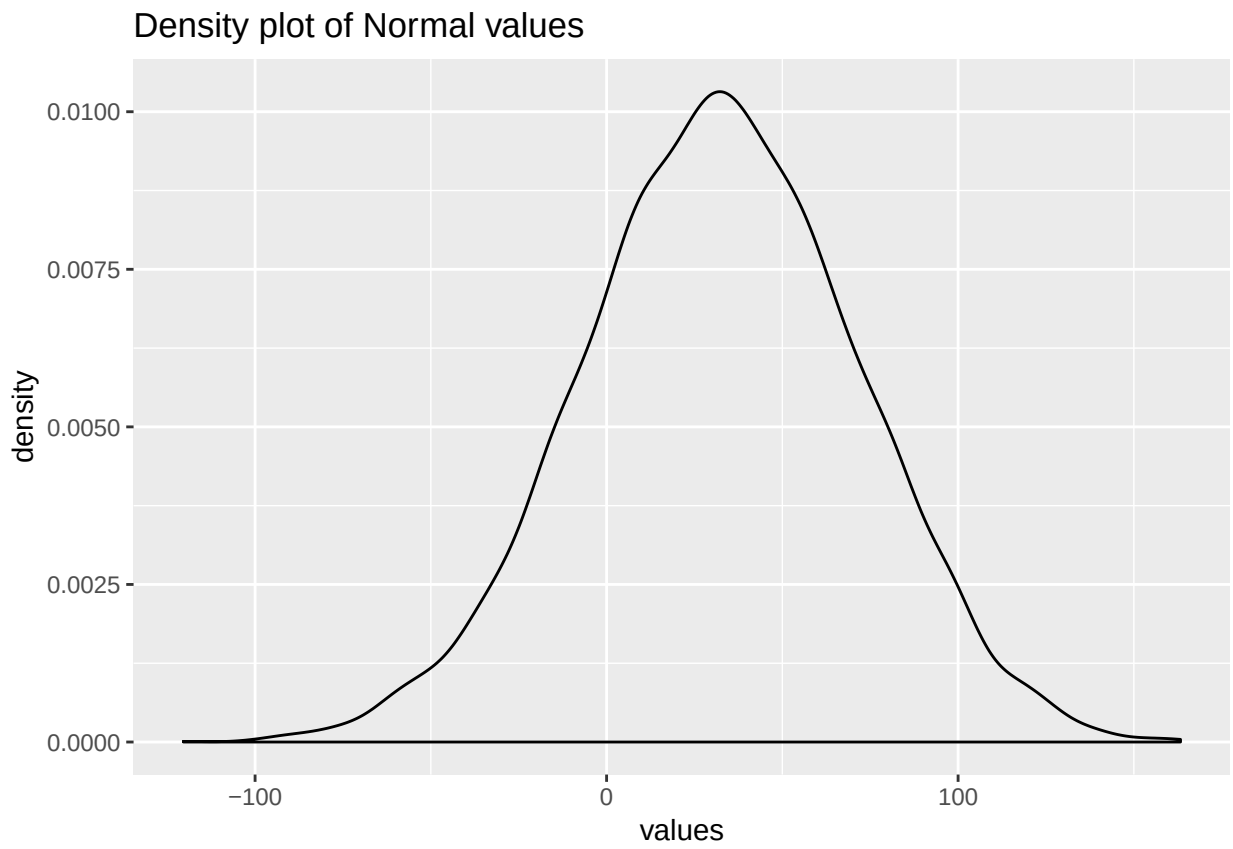
c) Graphical comparison
Let $\hat{\mu}$ denote the posterior mean of the parameter μ and correspondingly for the other parameters. Plot the following densities in one figure:

1) a histogram or kernel density estimate of the data.
2) Normal density $N(\hat{\mu}, \hat{\sigma^2})$ in a)
3) Mixture of normals density $p(y_i|\hat{\mu}, \hat{\sigma^2}, \hat{\pi})$ in b)

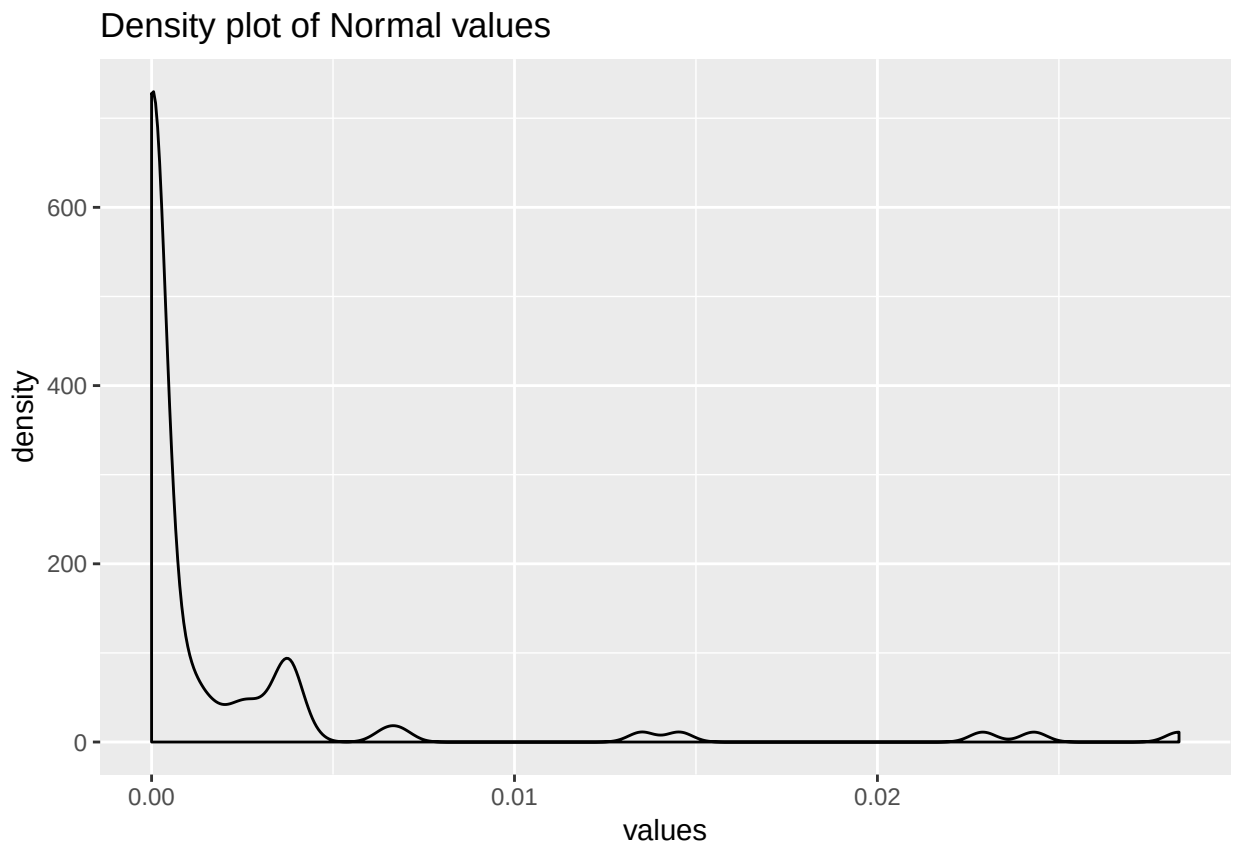
```
set.seed(12345)
ggplot(data = rain_data) +
  geom_density(aes(x=rainfall)) +
  ggtitle("Density plot of Precipitation")
```




```r
mean_mu_hat <- mean(gibbs_df$mu)
mean_sigma_hat <- mean(gibbs_df$sigma_sq)
normal_values <- rnorm(n=6920, mean = mean_mu_hat, sd = sqrt(mean_sigma_hat))
normal_values <- normal_values %>% as.data.frame()
colnames(normal_values) <- c("values")

ggplot(data=normal_values, aes(x=values)) +
  geom_density() +
  ggtitle("Density plot of Normal values")
```



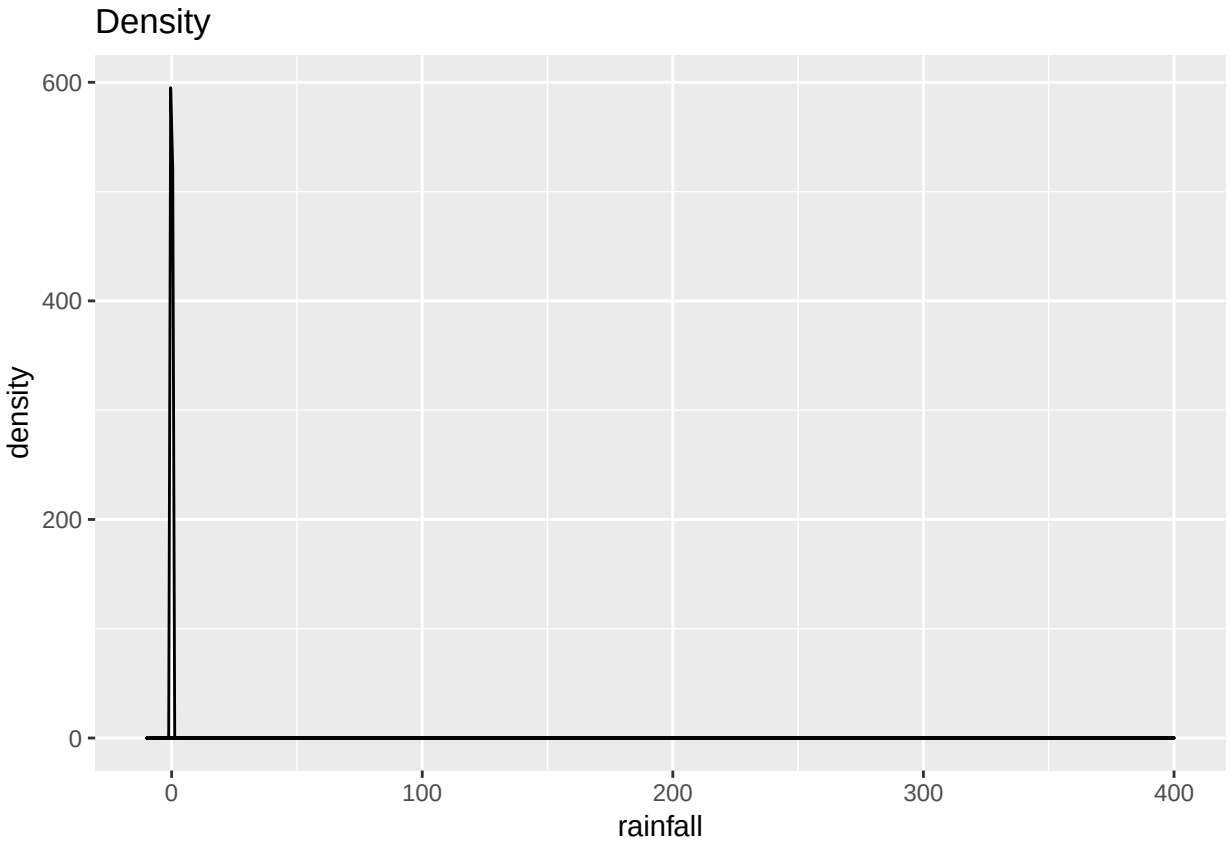

```
temp <- mixDens %>% as.data.frame()
colnames(temp) <- "values"
temp$values <- as.numeric(temp$values)

ggplot(data=temp, aes(x=values)) +
  geom_density() +
  ggtitle("Density plot of Normal values")
```




```
ggplot() +
  geom_density(data=rain_data, aes(x=rainfall))  +
  geom_density(data=normal_values, aes(x=values)) +
  geom_density(data=temp, aes(x=values)) +
  xlim(-10,400) +
  ggtitle("Density")
```

```
## Warning: Removed 2 rows containing non-finite values (stat_density).
```

```
## Warning: Removed 984 rows containing non-finite values (stat_density).
```


Density
600
400
200
0
density
0
100
200
300
400
rainfall

Question 2: Metropolis Random Walk for Poisson regression.

Consider the following Poisson regression model

$$y_i, \beta \sim Poission[\exp(x_i^T, \beta)], i = 1, 2, ..., n$$

where y_i is the count for the ith observation in the sample and x_i is the p-dimensional vector with covariate observations for the ith observation. Use the data set eBayNumberOfBidderData.dat. This dataset contains observations from 1000 eBay auctions of coins. The response variable is nBids and records the number of bids in each auction. The remaining variables are features/covariates (x):

Const (for the intercept)

PowerSeller (is the seller selling large volumes on eBay?)

VerifyID (is the seller verified by eBay?)

Sealed (was the coin sold sealed in never opened envelope?)

MinBlem (did the coin have a minor defect?)

MajBlem (a major defect?)

LargNeg (did the seller get a lot of negative feedback from customers?)

LogBook (logarithm of the coins book value according to expert sellers. Standardized)

MinBidShare (a variable that measures ratio of the minimum selling price (starting price) to the book value. Standardized).

a) Obtain the maximum likelihood estimator of β in the Poisson regression model for the eBay data [Hint: glm.R, don't forget that glm() adds its own intercept so don't input the covariate Const]. Which covariates are significant?

```
set.seed(12345)
ebay_data <- read.csv("ebay_data.txt", sep="")
poisson_model <- glm(nBids~.+0, data = ebay_data, family = poisson)
summary(poisson_model)
```

```
##
## Call:
## glm(formula = nBids ~ . + 0, family = poisson, data = ebay_data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.5800  -0.7222  -0.0441   0.5269   2.4605
##
## Coefficients:
##             Estimate Std. Error z value             Pr(>|z|)
## Const        1.07244    0.03077  34.848 < 0.0000000000000002 ***
## PowerSeller -0.02054    0.03678  -0.558               0.5765
## VerifyID    -0.39452    0.09243  -4.268            0.0000197 ***
## Sealed       0.44384    0.05056   8.778 < 0.0000000000000002 ***
## Minblem     -0.05220    0.06020  -0.867               0.3859
## MajBlem     -0.22087    0.09144  -2.416               0.0157 *
## LargNeg      0.07067    0.05633   1.255               0.2096
## LogBook     -0.12068    0.02896  -4.166            0.0000309 ***
## MinBidShare -1.89410    0.07124 -26.588 < 0.0000000000000002 ***
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 6264.01  on 1000  degrees of freedom
## Residual deviance:  867.47  on  991  degrees of freedom
## AIC: 3610.3
##
## Number of Fisher Scoring iterations: 5
```

Analysis: The significant terms are Const,VerifyID,Sealed,LogBook and MinBidshare. MajBlem can be considered significant only if our thershold for significance is lowered.

b) Let's now do a Bayesian analysis of the Poisson regression. Let the prior be $\beta \sim N[0, 100 \cdot (X^T X)^{-1}]$ where X is the nxp covariate matrix. This is a commonly used prior which is called Zellner's g-prior. Assume first that the posterior density is approximately multivariate normal:
In the next step, the goal is to approximate the posterior distribution of β with a multivariate normal distribution

$$\beta|y, X \sim N\left(\widetilde{\beta}, J_y^{-1}(\widetilde{\beta})\right)$$

where $\widetilde{\beta}$ is the posterior mode and $J(\widetilde{\beta}) = -\frac{\partial^2 lnp(\beta|y)}{\partial\beta\partial\beta^T}|_{\beta=\widetilde{\beta}}$ is the observed Hessian evaluated at the posterior mode. Both $\widetilde{\beta}$ and $J(\widetilde{\beta})$ are computed by the 'optim'-function in R. We use the prior $\beta \sim N[0, 100 \cdot (X^T X)^{-1}]$

$$\beta|y \sim N(\widetilde{\beta}, J_y^{-1}(\widetilde{\beta}))$$

where $\widetilde{\beta}$ is the posterior mode and $J_y(\widetilde{\beta})$ is the negative Hessian at the posterior mode. $\widetilde{\beta}$ and $J_y(\widetilde{\beta})$ can be obtained by numerical optimization (optim.R)exactly like you already did for the logistic regression in Lab 2 (but with the log posterior function replaced by the corresponding one for the Poisson model, which you have to code up)

```
set.seed(12345)
X <- ebay_data[,c("Const","PowerSeller","VerifyID","Sealed","Minblem","MajBlem",
                  "LargNeg","LogBook","MinBidShare")] %>% as.matrix()
X_T <- t(X)
Y <- ebay_data[,c("nBids"), drop=FALSE]
inverse_X_T_X <- solve(X_T %*% X)
beta_sigma <- 100 * solve(inverse_X_T_X)
mu_0 = rep(0, ncol(X))
# intialize
beta_init = rep(0, ncol(X))

log_posterior <- function(betaVect,mu,sigma,Y,X){

  # log of the likelihood -> source wikipedia
  log.likelihood = sum(Y * X %*% beta_init - exp(X %*% beta_init))  #(sum(y*betas%*%t(X) - exp(betas%*%

  # if likelihood is very large or very small, stear optim away
  if (abs(log.likelihood) == Inf) log.likelihood = -20000;

# log of the prior
log.prior = dmvnorm(betaVect, mean = mu, sigma = sigma, log = TRUE)

return(log.likelihood + log.prior)
```

```
}

results_optim = optim(par = beta_init, fn = log_posterior,Y = Y, X = X,
                      mu = mu_0, sigma = beta_sigma, method=c("BFGS"),
                      gr = NULL,
                      # Multiplying objective function by -1 to find maximum instead of minimum.
                      control=list(fnscale=-1),
                      hessian=TRUE)

# evaluating the prior
Final_beta <- rmvnorm(n=1000, mean=results_optim$par, sigma= -solve(results_optim$hessian)) %>% as.data
colnames(Final_beta) <- c("Const","PowerSeller","VerifyID","Sealed","Minblem",
                          "MajBlem","LargNeg","LogBook","MinBidShare")

# comparing with glm model
beta_from_distribution <- colMeans(Final_beta) %>% as.data.frame()
kable(beta_from_distribution, caption = "Betas from sampling using log-likelihood")
```

Table 1: Betas from sampling using log-likelihood

	.
Const	1.6193189
PowerSeller	-5.7364092
VerifyID	-3.9891136
Sealed	-0.9687102
Minblem	2.8986607
MajBlem	-1.1769307
LargNeg	-0.7860519
LogBook	9.0555531
MinBidShare	-2.4905849

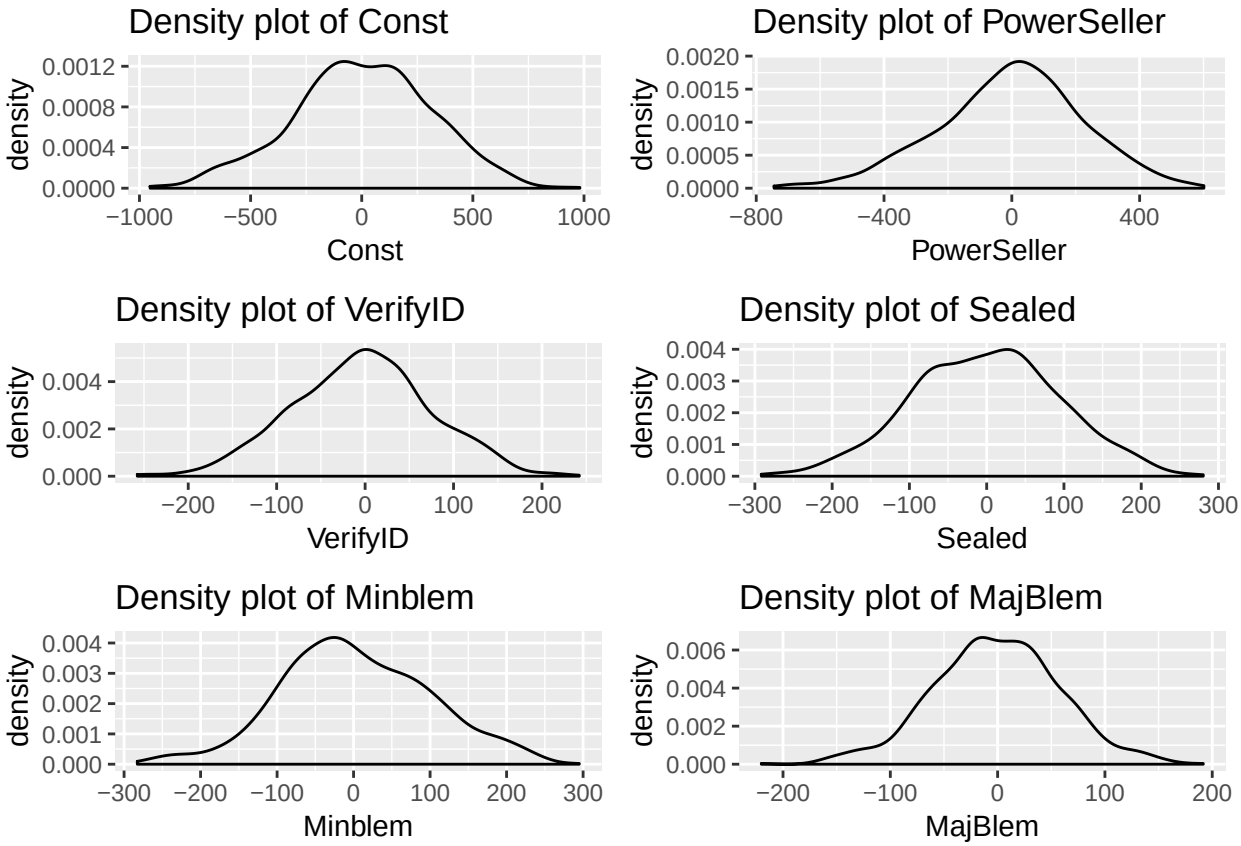
```
summary(poisson_model) %>%  xtable() %>% kable(caption = "Betas from GLM model")
```

Table 2: Betas from GLM model

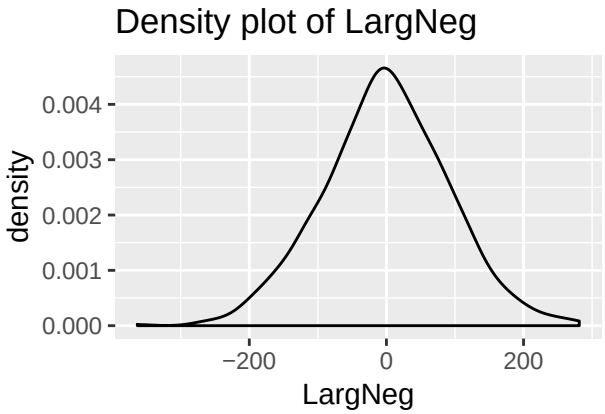
	Estimate	Std. Error	z value	Pr(>|z|)
Const	1.0724421	0.0307749	34.8479873	0.0000000
PowerSeller	-0.0205408	0.0367822	-0.5584429	0.5765420
VerifyID	-0.3945165	0.0924258	-4.2684678	0.0000197
Sealed	0.4438426	0.0505627	8.7780568	0.0000000
Minblem	-0.0521983	0.0602011	-0.8670661	0.3859058
MajBlem	-0.2208712	0.0914364	-2.4155711	0.0157106
LargNeg	0.0706725	0.0563318	1.2545743	0.2096334
LogBook	-0.1206776	0.0289646	-4.1663870	0.0000309
MinBidShare	-1.8940966	0.0712396	-26.5876846	0.0000000

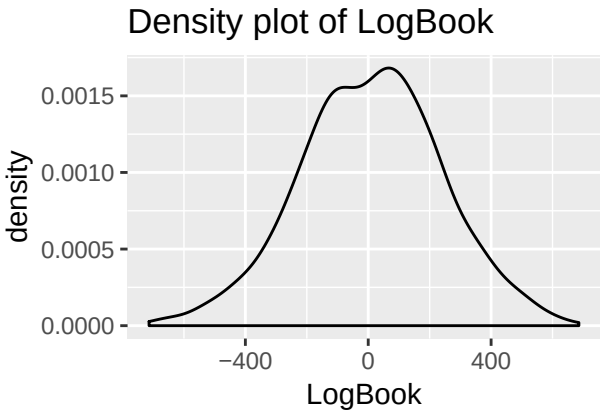
```
## Density plots for variable

p1 <- ggplot(data=Final_beta, aes(x=Const)) +
  geom_density() +
  ggtitle("Density plot of Const")
```

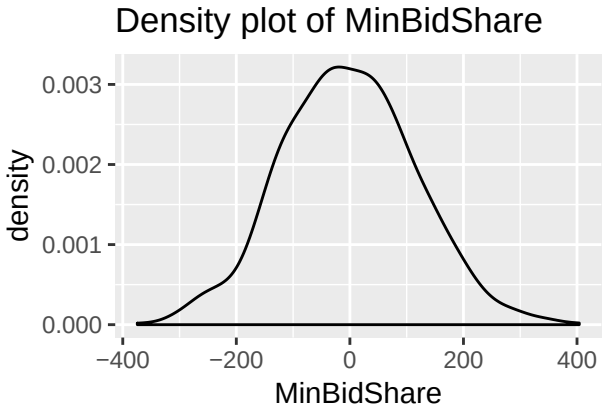
```r
p2 <- ggplot(data=Final_beta, aes(x=PowerSeller)) +
  geom_density() +
  ggtitle("Density plot of PowerSeller")

p3 <- ggplot(data=Final_beta, aes(x=VerifyID)) +
  geom_density() +
  ggtitle("Density plot of VerifyID")

p4 <- ggplot(data=Final_beta, aes(x=Sealed)) +
  geom_density() +
  ggtitle("Density plot of Sealed")

p5 <- ggplot(data=Final_beta, aes(x=Minblem)) +
  geom_density() +
  ggtitle("Density plot of Minblem")

p6 <- ggplot(data=Final_beta, aes(x=MajBlem)) +
  geom_density() +
  ggtitle("Density plot of MajBlem")

p7 <- ggplot(data=Final_beta, aes(x=LargNeg)) +
  geom_density() +
  ggtitle("Density plot of LargNeg")

p8 <- ggplot(data=Final_beta, aes(x=LogBook)) +
  geom_density() +
  ggtitle("Density plot of LogBook")

p9 <- ggplot(data=Final_beta, aes(x=MinBidShare)) +
  geom_density() +
  ggtitle("Density plot of MinBidShare")

grid.arrange(p1,p2,p3,p4,p5,p6, nrow=3, ncol=2)
```




```
grid.arrange(p7,p8,p9, nrow=2, ncol=2)
```







c) Now, let's simulate from the actual posterior of β using the Metropolis algorithm and compare with the approximate results in b). Program a general function that uses the Metropolis algorithm to generate random draws from an arbitrary posterior density. In order to show that it is a general function for any model, I will denote the vector of model parameters by θ. Let the proposal density be the multivariate normal density mentioned in Lecture 8 (random walk Metropolis):

$$\theta_p|\theta^{(i-1)} \sim N(\theta^{(i-1)}, c \cdot \sum)$$

where

$$\sum = J_y^{-1}(\widetilde{\beta})$$

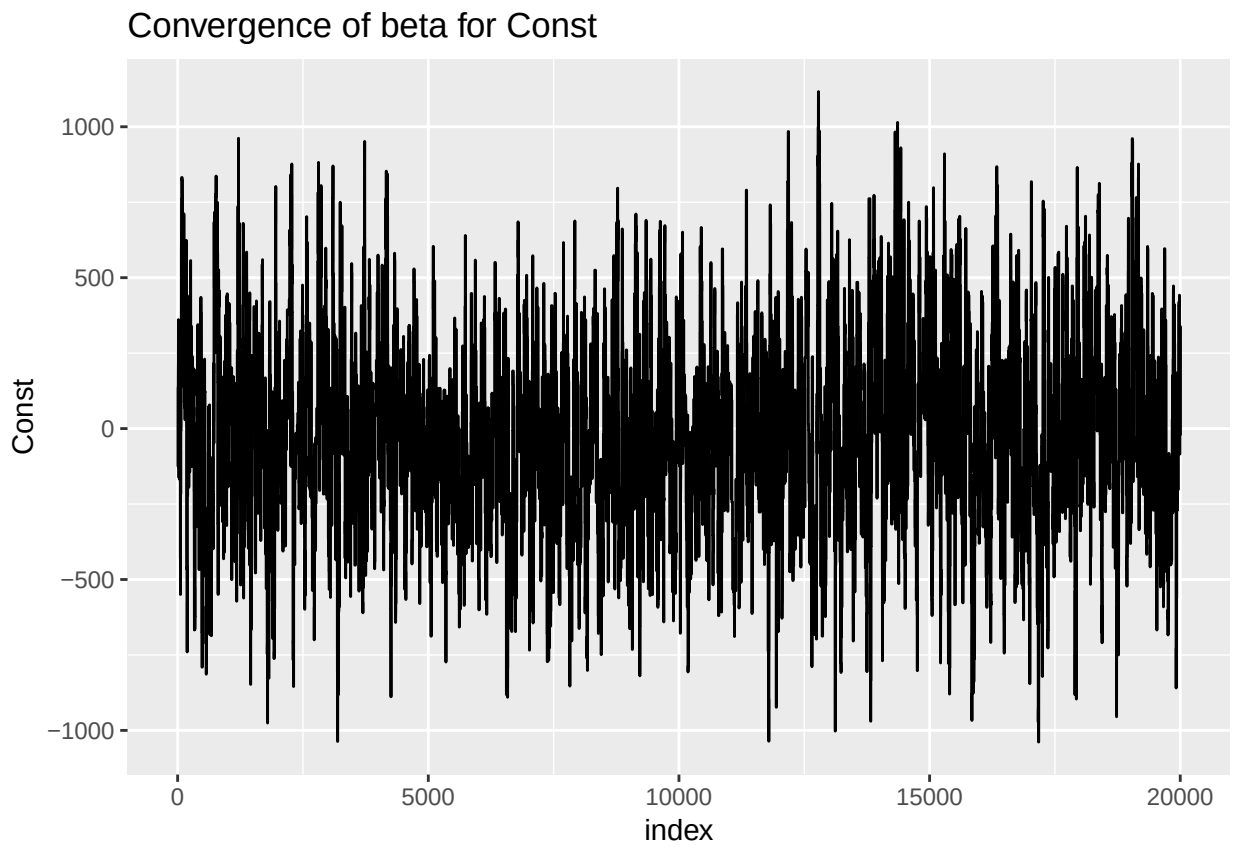
obtained in b). The value c is a tuning parameter and should be an input to your Metropolis function. The user of your Metropolis function should be able to supply her own posterior density function, not necessarily for the Poisson regression, and still be able to use your Metropolis function. This is not so straightforward, unless you have come across function objects in R and the triple dot (...) wildcard argument. I have posted a note (HowToCodeRWM.pdf) on the course web page that describes how to do this in R. Now, use your new Metropolis function to sample from the posterior of β in the Poisson regression for the eBay dataset. Assess MCMC convergence by graphical methods.

```
set.seed(12345)
Sigma = -solve(results_optim$hessian)
c = 0.6
n_draws = 20000

metropolisHastings = function(logPostFunc, theta, c, ...){
theta_draws = matrix(0, n_draws, length(theta))
# Set initial
```

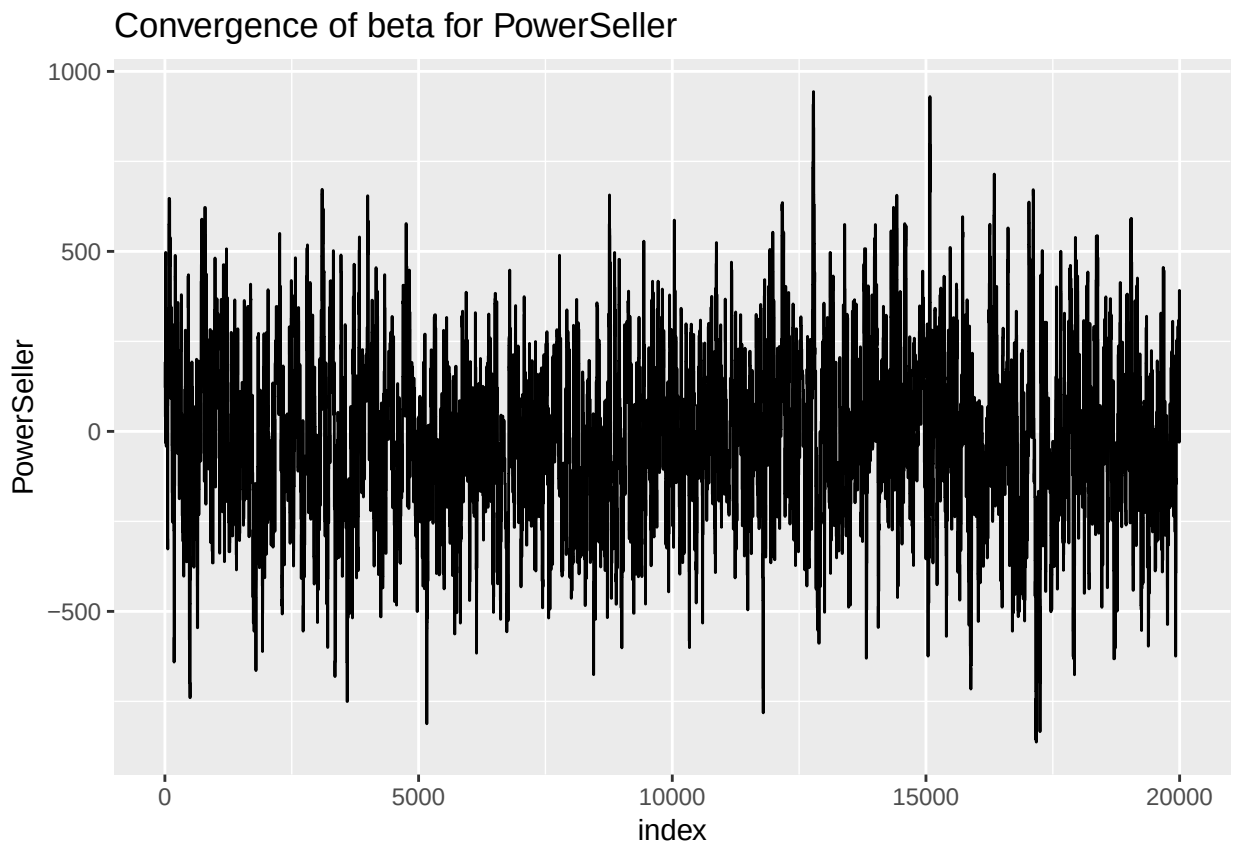
```
theta_c = mvrnorm(n=1, theta, c*Sigma)
prob_sum = 0

for(i in 1:n_draws){
# 1: Draw new proposal theta
  theta_p = mvrnorm(n=1, theta_c, c*Sigma)
# 2: Determine the acceptance probability
p_prev = logPostFunc(theta_c, ...)
p_new = logPostFunc(theta_p, ...)
acc_prob = min(c(1, exp(p_new - p_prev)))
prob_sum = prob_sum + acc_prob
# 3: Set new value with prob = acc_prob
if(rbern(n=1, p=acc_prob)==1){
theta_c = theta_p
}
theta_draws[i,] = theta_c
}
print(paste('Avg. acc. prob. = ', round(prob_sum/n_draws, 2)))
return (theta_draws)
}
init_beta = mvrnorm(n=1, mu_0, beta_sigma)

beta_draws = metropolisHastings(log_posterior, theta=init_beta, c=c, X=X, Y=Y, mu=mu_0, sigma=Sigma)
```

```
## [1] "Avg. acc. prob. =  0.27"
```

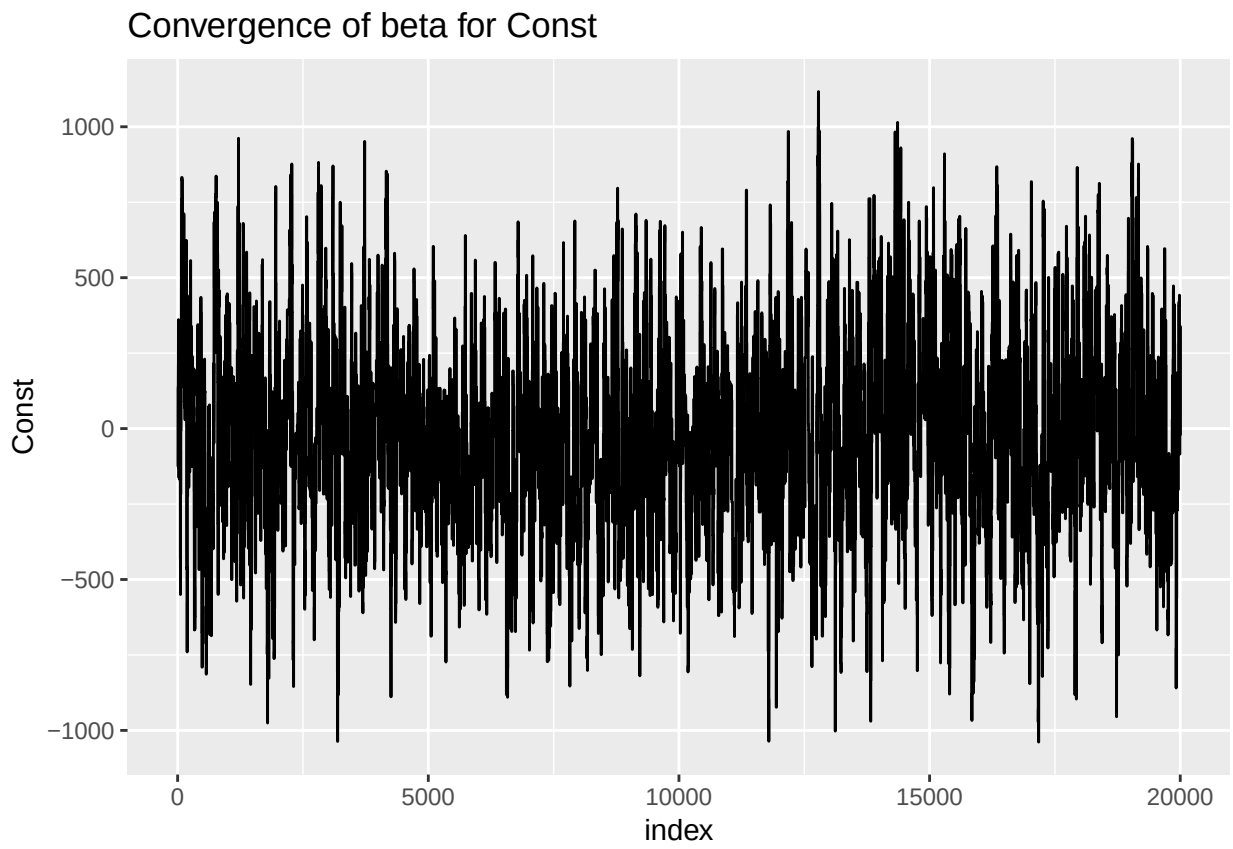
```
beta_draws <- beta_draws %>% as.data.frame()
colnames(beta_draws) <- c("Const","PowerSeller","VerifyID","Sealed","Minblem",
                          "MajBlem","LargNeg","LogBook","MinBidShare")

beta_draws$index <- as.integer(row.names(beta_draws))


ggplot(data=beta_draws, aes(x=index, y=Const)) +
  geom_line() +
  ggtitle("Convergence of beta for Const")
```




```
ggplot(data=beta_draws, aes(x=index, y=PowerSeller)) +
  geom_line() +
  ggtitle("Convergence of beta for PowerSeller")
```




```
ggplot(data=beta_draws, aes(x=index, y=Const)) +
  geom_line() +
  ggtitle("Convergence of beta for Const")
```



Appendix

```
knitr::opts_chunk$set(echo = TRUE)
#options(tinytex.verbose = TRUE)
options(scipen=999)

library("ggplot2")
library("tidyverse")
library("LaplacesDemon")
library("mvtnorm") # multi variate Normal
library("MASS") # To access the mvrnorm() function
library("gridExtra") # combine plots
library("xtable") # model summary as table
library("knitr")


# The palette with black:
cbbPalette <- c("#000000", "#E69F00", "#56B4E9", "#009E73",
                "#F0E442", "#0072B2", "#D55E00", "#CC79A7")
set.seed(12345)

# Normal Model
set.seed(12345)
```

```
rain_data <-  read.table("rainfall.txt", quote="\"", comment.char="")
colnames(rain_data) <- c("rainfall")

n = length(rain_data$rainfall)
mu_0 = mean(rain_data$rainfall)
tau_sq = sum((log(rain_data$rainfall)-mu_0)^2)/n

# calculating tau_n_sq
tau_n_sq = c()
for (i in 1:n) {
  # Drawing x.
  x = rchisq(n = 1, df = n)
  # Calculating and storing tau_n_sq
  tau_n_sq[i] = (n*tau_sq)/x
}

# Using Full conditional posteriors

## using tau_n_sq and mu_0 we get estimate mu using \mu|\sigma^2,x \sim N(\mu_n, \tau_n^2)
mu <- c()
for (i in 1:n) {
mu[i] <- rnorm(n=1, mean=mu_0, sd = sqrt(tau_n_sq[i]/n))
}

## Now sampling sigmas using \sigma^2|\mu,x \sim Inv-\chi^2(\nu_n,\frac{\nu_0\sigma^2_0+\sum_{i=1}^n (x

# Prior parameters for sigma (variance)
v0 = 1
sigma_sq0 = 1
v_n = v0 + n


sigma_sq <- c()
for (i in 1:n) {
sigma_sq[i] <- rinvchisq(n=1, v_n, (v0*sigma_sq0 + sum((rain_data$rainfall - mu[i])^2))/(n + v0))
}

## dataframe
gibbs_df <- cbind(mu,sigma_sq) %>% as.data.frame()
gibbs_df$index <- as.integer(row.names(gibbs_df))

## adding 0 values
temp <- c(mu=0,sigma_sq=0)
gibbs_df <- rbind(gibbs_df,temp)


## plots
ggplot(data=gibbs_df, aes(x=index, y=mu)) +
  geom_line() +
  ylim(0,33.9) +
  ggtitle(expression(paste("Convergence of ", mu,  " vs. index")))

ggplot(data=gibbs_df, aes(x=index, y=sigma_sq)) +
```

```
  geom_line() +
  ylim(0,1650) +
  ggtitle(expression(paste("Convergence of ", sigma,  " vs. index")))

set.seed(12345)
data = read.table("rainfall.txt", header=FALSE)[,1]
x = as.matrix(data)

# Model options
nComp <- 2 # Number of mixture components
# Prior options
alpha <- 10*rep(1,nComp) # Dirichlet(alpha)
muPrior <- rep(0,nComp) # Prior mean of theta
tau2Prior <- rep(10,nComp) # Prior std theta
sigma2_0 <- rep(var(x),nComp) # s20 (best guess of sigma2)
nu0 <- rep(4,nComp) # degrees of freedom for prior on sigma2
# MCMC options
nIter <- 100 # Number of Gibbs sampling draws

# Plotting options
plotFit <- FALSE
lineColors <- c("blue", "green", "magenta", 'yellow')
sleepTime <- 0.1 # Adding sleep time between iterations for plotting

################ END USER INPUT ################
###### Defining a function that simulates from the
rScaledInvChi2 <- function(n, df, scale){
return((df*scale)/rchisq(n,df=df))
}

####### Defining a function that simulates from a Dirichlet distribution
rDirichlet <- function(param){
nCat <- length(param)
thetaDraws <- matrix(NA,nCat,1)
for (j in 1:nCat){
thetaDraws[j] <- rgamma(1,param[j],1)
}#Diving every column of ThetaDraws by the sum of the elements in that column.
thetaDraws = thetaDraws/sum(thetaDraws)
return(thetaDraws)
}

# Simple function that converts between two different
# representations of the mixture allocation
S2alloc <- function(S){
n <- dim(S)[1]
alloc <- rep(0,n)
for (i in 1:n){
alloc[i] <- which(S[i,] == 1)
}
return(alloc)
}

# Initial value for the MCMC
```

```
nObs <- length(x)
# nObs-by-nComp matrix with component allocations.
S <- t(rmultinom(nObs, size = 1 , prob = rep(1/nComp,nComp)))
theta <- quantile(x, probs = seq(0,1,length = nComp))
sigma2 <- rep(var(x),nComp)
probObsInComp <- rep(NA, nComp)
# Setting up the plot
xGrid <- seq(min(x)-1*apply(x,2,sd),max(x)+1*apply(x,2,sd),length = 100)
xGridMin <- min(xGrid)
xGridMax <- max(xGrid)
mixDensMean <- rep(0,length(xGrid))
effIterCount <- 0
ylim <- c(0,2*max(hist(x)$density))
gibbs_thetas = matrix(0,nIter,2)
gibbs_sigmas = matrix(0,nIter,2)

for (k in 1:nIter){
# Just a function that converts between different representations
# of the group allocations
alloc <- S2alloc(S)
nAlloc <- colSums(S)
if(k == nIter){
message(paste('Iteration number:',k))
print(nAlloc)
}

# Update components probabilities
w <- rDirichlet(alpha + nAlloc)

# Update theta's
for (j in 1:nComp){
precPrior <- 1/tau2Prior[j]
precData <- nAlloc[j]/sigma2[j]
precPost <- precPrior + precData
wPrior <- precPrior/precPost
muPost <- wPrior*muPrior + (1-wPrior)*mean(x[alloc == j])
tau2Post <- 1/precPost
theta[j] <- rnorm(1, mean = muPost, sd = sqrt(tau2Post))
}

gibbs_thetas[k, ] = theta
# Update sigma2's
for (j in 1:nComp){
sigma2[j] <- rScaledInvChi2(1, df = nu0[j] + nAlloc[j],
scale = (nu0[j]*sigma2_0[j] +
sum((x[alloc == j] - theta[j])^2))/(nu0[j] + nAlloc[j]))
}

gibbs_sigmas[k,] = sigma2
# Update allocation
for (i in 1:nObs){
for (j in 1:nComp){
probObsInComp[j] <- w[j]*dnorm(x[i], mean = theta[j], sd = sqrt(sigma2[j]))
```

```
}
S[i,] <- t(rmultinom(1, size = 1 , prob = probObsInComp/sum(probObsInComp)))
}

# Printing the fitted density against data histogram
if ((k==nIter) && (k%%1 ==0)){
effIterCount <- effIterCount + 1
hist(x, breaks = 20, freq = FALSE, xlim = c(xGridMin,xGridMax),
main = paste("Iteration number",k), ylim = ylim)
mixDens <- rep(0,length(xGrid))
components <- c()

for (j in 1:nComp){
compDens <- dnorm(xGrid,theta[j],sd = sqrt(sigma2[j]))
mixDens <- mixDens + w[j]*compDens
lines(xGrid, compDens, type = "l", lwd = 2, col = lineColors[j])
components[j] <- paste("Component ",j)
}

mixDensMean <- ((effIterCount-1)*mixDensMean + mixDens)/effIterCount
lines(xGrid, mixDens, type = "l", lty = 2, lwd = 3, col = 'red')
legend("topleft", box.lty = 1, legend = c("Data histogram",components, 'Mixture'),
col = c("black",lineColors[1:nComp], 'red'), lwd = 2)
Sys.sleep(sleepTime)
}
}

# Calculate mean of batches of 2 draws to visualize the
# auto correlation between sequential draws
t1 = c()
t2 = c()
s1 = c()
s2 = c()
for (i in 1:nIter){
if(i%%2 == 0){
t1 = c(t1, mean(gibbs_thetas[,1][i-1:i]))
t2 = c(t2, mean(gibbs_thetas[,2][i-1:i]))
s1 = c(s1, mean(gibbs_sigmas[,1][i-1:i]))
s2 = c(s2, mean(gibbs_sigmas[,2][i-1:i]))
}
}

# Plots displaying convergence of the Normal hyper
# parameters during sampling
pdf("3_1_2_gibbs_mixt.pdf")
par(mfrow=c(3,1))

# Plot comparison between kernel density, mixture of normals and a normal
# approximation
hist(x, breaks = 20,
cex=.1,
border="lightgray",
freq = FALSE,
```

```
xlim = c(xGridMin,xGridMax),
xlab="Precipitation",
ylab="Density",
main = "Rainfall: Mixture of Normals")
lines(xGrid,
mixDensMean,
type = "l",
lwd = 2,
lty = 4,
col = "black")
lines(xGrid,dnorm(xGrid, mean = mean(x), sd = apply(x,2,sd)),type = "l",lwd = 2,col = "gray")
legend("topright",box.lty = 1,legend = c("Data histogram","Mixture density","Normal density"),col=c("li
# Plot the auto correlation (convergence) between draws of mu
min_t = min(c(min(t1), min(t2)))
max_t = max(c(max(t1), max(t2)))

plot(t1,type="l",ylim=c(min_t, max_t),cex=.1,lwd=2,
     main=expression(paste("Convergence of Gibbs Sampling ", "(", theta, ")", sep=" ")),
xlab="Batches of sequential draws",
ylab=expression(paste("Mean of seq. draws of ", theta, sep=" ")))
lines(t2, lwd=2, col="gray")
legend("topright",box.lty = 1,legend = c(expression(paste(theta, " (1)", sep=" ")),
                                         expression(paste(theta, " (2)", sep=" "))),col=c("black","gray
# Plot the auto correlation (convergence) between draws of sigma
min_s = min(c(min(s1), min(s2)))
max_s = max(c(max(s1), max(s2)))

plot(s1,type="l",ylim=c(min_s, max_s),cex=.1,lwd=2,main=expression(
paste("Convergence of Gibbs Sampling ", "(", sigma^2, ")", sep=" ")),
xlab="Batches of sequential draws",
ylab=expression(paste("Mean of seq. draws of ", sigma^2, sep=" ")))
lines(s2, lwd=2, col="gray")
legend("topright",box.lty = 1,legend = c(expression(paste(sigma^2, " (1)", sep=" ")),expression(paste(s

dev.off()

set.seed(12345)
ggplot(data = rain_data) +
  geom_density(aes(x=rainfall)) +
  ggtitle("Density plot of Precipitation")

mean_mu_hat <- mean(gibbs_df$mu)
mean_sigma_hat <- mean(gibbs_df$sigma_sq)
normal_values <- rnorm(n=6920, mean = mean_mu_hat, sd = sqrt(mean_sigma_hat))
normal_values <- normal_values %>% as.data.frame()
colnames(normal_values) <- c("values")

ggplot(data=normal_values, aes(x=values)) +
  geom_density() +
  ggtitle("Density plot of Normal values")

temp <- mixDens %>% as.data.frame()
colnames(temp) <- "values"
```

```r
temp$values <- as.numeric(temp$values)

ggplot(data=temp, aes(x=values)) +
  geom_density() +
  ggtitle("Density plot of Normal values")


ggplot() +
  geom_density(data=rain_data, aes(x=rainfall))  +
  geom_density(data=normal_values, aes(x=values)) +
  geom_density(data=temp, aes(x=values)) +
  xlim(-10,400) +
  ggtitle("Density")



set.seed(12345)
ebay_data <- read.csv("ebay_data.txt", sep="")
poisson_model <- glm(nBids~.+0, data = ebay_data, family = poisson)
summary(poisson_model)



set.seed(12345)
X <- ebay_data[,c("Const","PowerSeller","VerifyID","Sealed","Minblem","MajBlem",
                  "LargNeg","LogBook","MinBidShare")] %>% as.matrix()
X_T <- t(X)
Y <- ebay_data[,c("nBids"), drop=FALSE]
inverse_X_T_X <- solve(X_T %*% X)
beta_sigma <- 100 * solve(inverse_X_T_X)
mu_0 = rep(0, ncol(X))
# intialize
beta_init = rep(0, ncol(X))

log_posterior <- function(betaVect,mu,sigma,Y,X){

  # log of the likelihood -> source wikipedia
  log.likelihood = sum(Y * X %*% beta_init - exp(X %*% beta_init))  #(sum(y*betas%*%t(X) - exp(betas%*%

  # if likelihood is very large or very small, stear optim away
  if (abs(log.likelihood) == Inf) log.likelihood = -20000;

# log of the prior
log.prior = dmvnorm(betaVect, mean = mu, sigma = sigma, log = TRUE)

return(log.likelihood + log.prior)
}

results_optim = optim(par = beta_init, fn = log_posterior,Y = Y, X = X,
                      mu = mu_0, sigma = beta_sigma, method=c("BFGS"),
                      gr = NULL,
                      # Multiplying objective function by -1 to find maximum instead of minimum.
                      control=list(fnscale=-1),
```

```
                      hessian=TRUE)

# evaluating the prior
Final_beta <- rmvnorm(n=1000, mean=results_optim$par, sigma= -solve(results_optim$hessian)) %>% as.data
colnames(Final_beta) <- c("Const","PowerSeller","VerifyID","Sealed","Minblem",
                          "MajBlem","LargNeg","LogBook","MinBidShare")

# comparing with glm model
beta_from_distribution <- colMeans(Final_beta) %>% as.data.frame()
kable(beta_from_distribution, caption = "Betas from sampling using log-likelihood")
summary(poisson_model) %>%  xtable() %>% kable(caption = "Betas from GLM model")

## Density plots for variable

p1 <- ggplot(data=Final_beta, aes(x=Const)) +
  geom_density() +
  ggtitle("Density plot of Const")

p2 <- ggplot(data=Final_beta, aes(x=PowerSeller)) +
  geom_density() +
  ggtitle("Density plot of PowerSeller")

p3 <- ggplot(data=Final_beta, aes(x=VerifyID)) +
  geom_density() +
  ggtitle("Density plot of VerifyID")

p4 <- ggplot(data=Final_beta, aes(x=Sealed)) +
  geom_density() +
  ggtitle("Density plot of Sealed")

p5 <- ggplot(data=Final_beta, aes(x=Minblem)) +
  geom_density() +
  ggtitle("Density plot of Minblem")

p6 <- ggplot(data=Final_beta, aes(x=MajBlem)) +
  geom_density() +
  ggtitle("Density plot of MajBlem")

p7 <- ggplot(data=Final_beta, aes(x=LargNeg)) +
  geom_density() +
  ggtitle("Density plot of LargNeg")

p8 <- ggplot(data=Final_beta, aes(x=LogBook)) +
  geom_density() +
  ggtitle("Density plot of LogBook")

p9 <- ggplot(data=Final_beta, aes(x=MinBidShare)) +
  geom_density() +
  ggtitle("Density plot of MinBidShare")

grid.arrange(p1,p2,p3,p4,p5,p6, nrow=3, ncol=2)
grid.arrange(p7,p8,p9, nrow=2, ncol=2)
```

```
set.seed(12345)
Sigma = -solve(results_optim$hessian)
c = 0.6
n_draws = 20000

metropolisHastings = function(logPostFunc, theta, c, ...){
theta_draws = matrix(0, n_draws, length(theta))
# Set initial
theta_c = mvrnorm(n=1, theta, c*Sigma)
prob_sum = 0

for(i in 1:n_draws){
# 1: Draw new proposal theta
  theta_p = mvrnorm(n=1, theta_c, c*Sigma)
# 2: Determine the acceptance probability
p_prev = logPostFunc(theta_c, ...)
p_new = logPostFunc(theta_p, ...)
acc_prob = min(c(1, exp(p_new - p_prev)))
prob_sum = prob_sum + acc_prob
# 3: Set new value with prob = acc_prob
if(rbern(n=1, p=acc_prob)==1){
theta_c = theta_p
}
theta_draws[i,] = theta_c
}
print(paste('Avg. acc. prob. = ', round(prob_sum/n_draws, 2)))
return (theta_draws)
}
init_beta = mvrnorm(n=1, mu_0, beta_sigma)

beta_draws = metropolisHastings(log_posterior, theta=init_beta, c=c, X=X, Y=Y, mu=mu_0, sigma=Sigma)

beta_draws <- beta_draws %>% as.data.frame()
colnames(beta_draws) <- c("Const","PowerSeller","VerifyID","Sealed","Minblem",
                          "MajBlem","LargNeg","LogBook","MinBidShare")

beta_draws$index <- as.integer(row.names(beta_draws))


ggplot(data=beta_draws, aes(x=index, y=Const)) +
  geom_line() +
  ggtitle("Convergence of beta for Const")


ggplot(data=beta_draws, aes(x=index, y=PowerSeller)) +
  geom_line() +
  ggtitle("Convergence of beta for PowerSeller")



ggplot(data=beta_draws, aes(x=index, y=Const)) +
  geom_line() +
  ggtitle("Convergence of beta for Const")
```