BayesianLearning - Lab3

Vinay Bengaluru(vinbe289), Tejashree R Mastamardi(tejma768)

May 19, 2019

```
library(geoR)
library(knitr)
library(mvtnorm)
```

Question 1

1(a)Normal Model

(i)Implement Gibbs sampler

```
set.seed(12345)

data <- read.table("rainfall.dat", header = T)
gibbsSampler <- function(iter, data, mu0, tau0sq, nu0, sigma0sq){
  n <- nrow(data)
  Xbar <- mean(data[,1])
  nun <- nu0 + n
  mu <- rnorm(n = 1,mean = mu0,sd = sqrt(tau0sq))
  sigmasq <- nu0*sigma0sq/rchisq(1,nu0)
  result <- matrix(ncol = 2,nrow = iter+1)
  result[1,1] <- mu
  result[1,2] <- sigmasq
  colnames(result) <- c("mu","sigmasq")
  for(i in 1:iter){
    W <- (n/result[i,2])/((n/result[i,2])+(1/tau0sq))
    mun <- W*Xbar + (1-W)*mu0
    taunsq <- 1/((n/result[i,2])+(1/tau0sq))
    mu <- rnorm(n = 1,mean = mun,sd = sqrt(taunsq))
    parN <- (nu0*sigma0sq+sum((data$X136-mu)^2))/nun
    sigmasq <- nun*parN/rchisq(n = 1,df = nun) #conditional posterior
    result[i+1,] <- c(mu, sigmasq)
  }
  result
}
```

(ii)Analyze the daily precipitation using Gibbs sampler in (a)(i). Evaluate the convergence of Gibbs sampler by suitable graphical methods, for example by plotting the trajectories of the sampled Markov chains.

```
#Setting prior values to parameters based on our knowledge
sigma0sq <- 20
tau0sq <- 50
nu0 <- 1
mu0 <- 20
iterMax <- 1000
burnIn <- 100
```

```
gibbsSample <- as.data.frame(gibbsSampler(iterMax,data,mu0,tau0sq,nu0,sigma0sq))
```

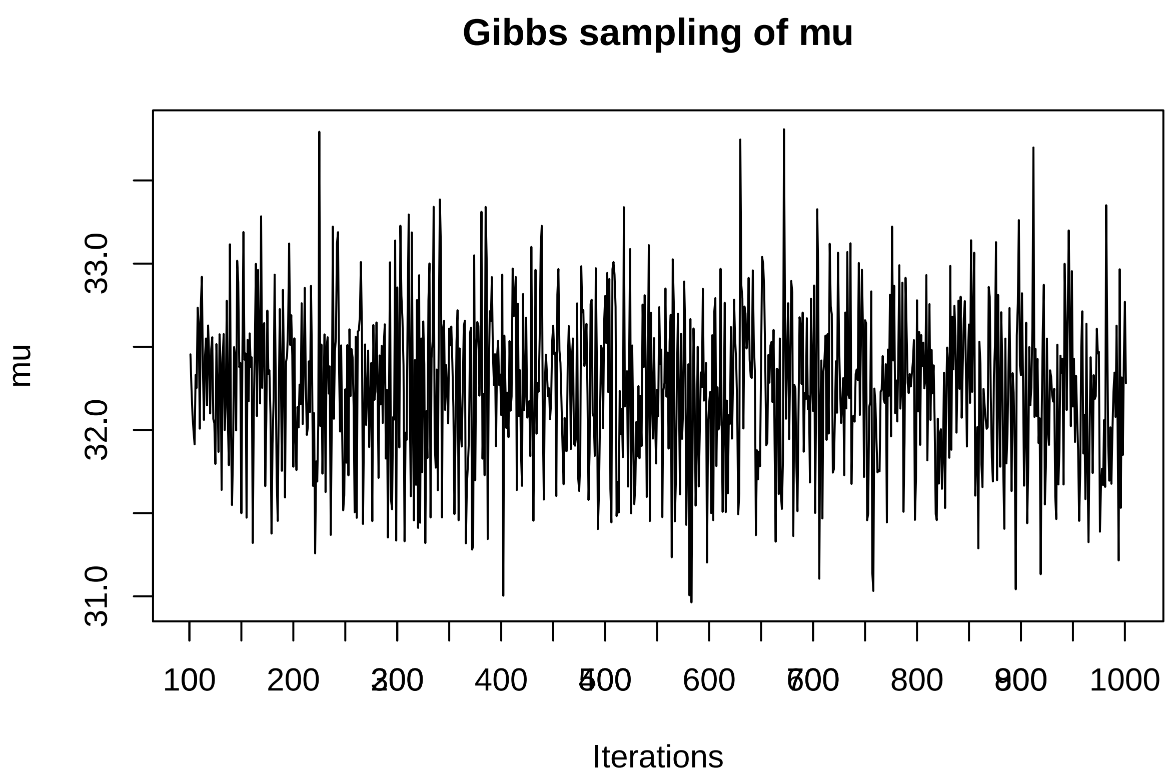
```
posteriorMu <- mean(gibbsSample$mu)
cat("Posterior Mean:",posteriorMu)
```

```
## Posterior Mean: 32.22681
```

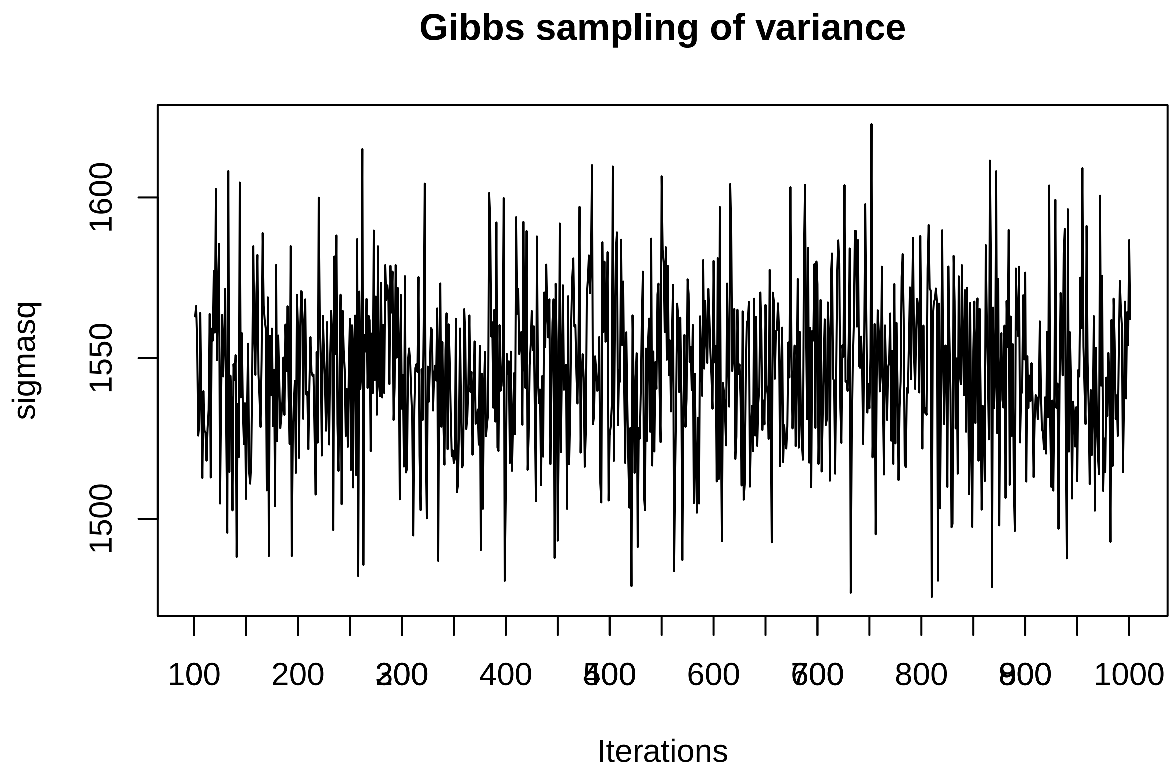
```
posteriorSigmasq <- mean(gibbsSample$sigmasq)
cat("\n Posterior Variance:",posteriorSigmasq)
```

```
##
##  Posterior Variance: 7061.133
```

```
plot(gibbsSample$mu[burnIn:iterMax],type="l",xlab="Iterations",ylab="mu",main="Gibbs sampling of mu")
axis(1,at=seq(0,(iterMax-burnIn),by=50),labels=seq(burnIn,iterMax,by=50))
```




```
plot(gibbsSample$sigmasq[burnIn:iterMax],type="l",xlab="Iterations",ylab="sigmasq",main="Gibbs sampling
axis(1,at=seq(0,(iterMax-burnIn),by=50),labels=seq(burnIn,iterMax,by=50))
```

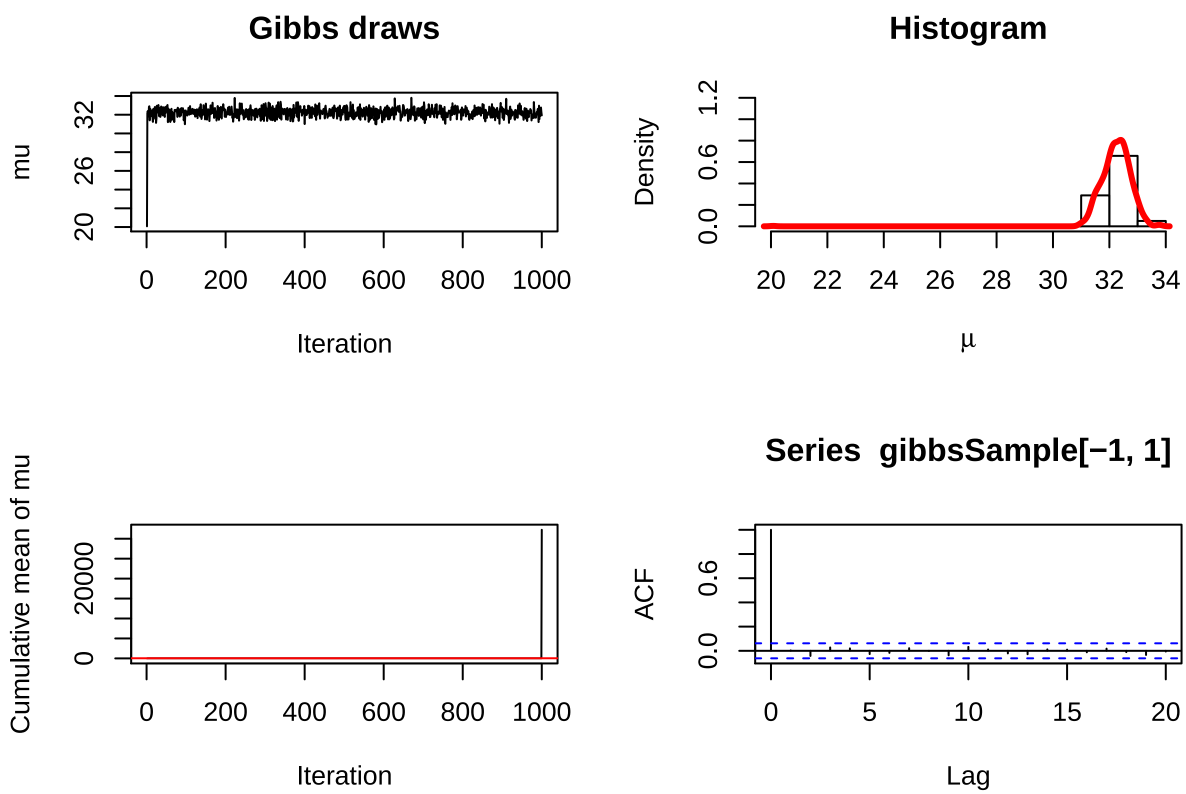


By looking at the plots it is evident that μ(mean) and σ^2(variance) values converges well through the iterations.

```
par(mfrow=c(2,2))
#convergence check for mu
plot(gibbsSample[-1,1],type="l",main='Gibbs draws',xlab='Iteration',ylab='mu')
hist(gibbsSample[-1,1],freq=FALSE,main='Histogram',xlab=expression(mu), breaks=15,ylim=c(0,1.2))
lines(density(gibbsSample[-1,1]),col="red",lwd=3)
plot(cumsum(gibbsSample[-1,1])/seq(1,iterMax-1),type="l",xlab='Iteration',ylab='Cumulative mean of mu')
```

```
## Warning in cumsum(gibbsSample[-1, 1])/seq(1, iterMax - 1): longer object
## length is not a multiple of shorter object length
```

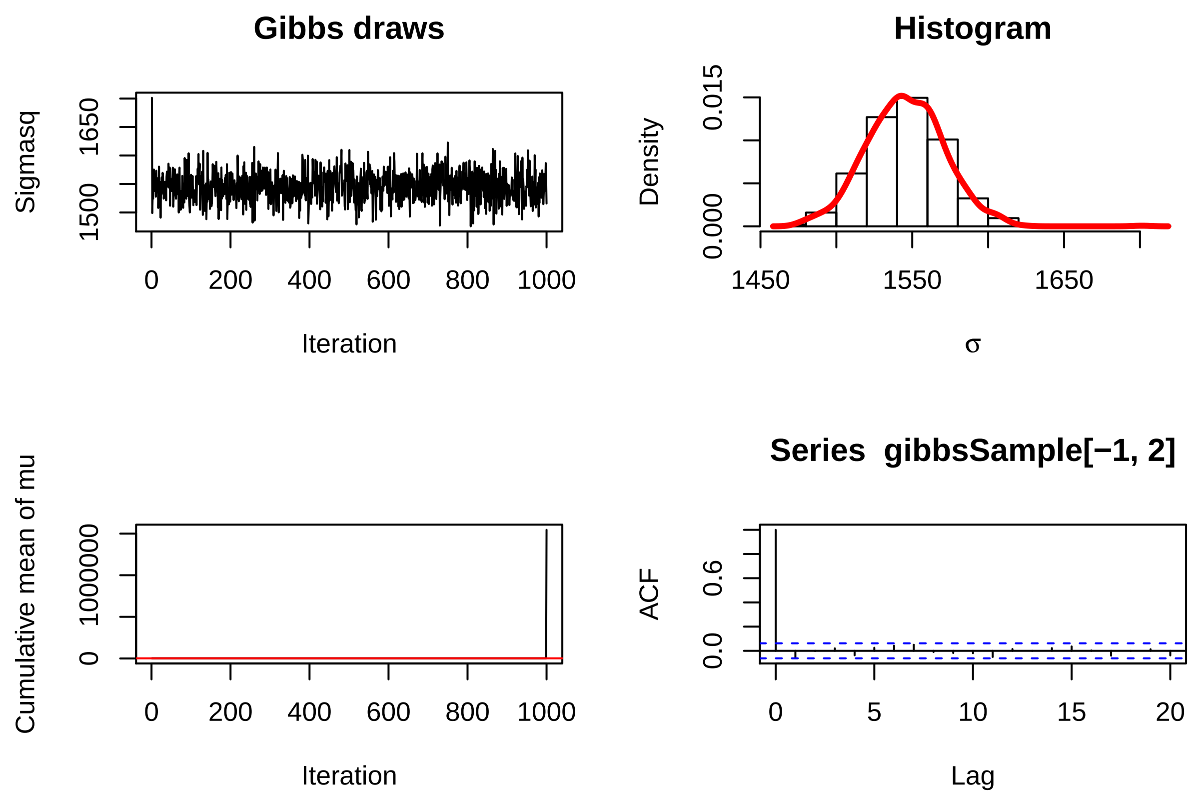
```
abline(h=cumsum(gibbsSample[-1,1])[iterMax-1]/(iterMax-1),col="red")
acf(gibbsSample[-1,1],lag.max=20)
```




```
par(mfrow=c(2,2))
#convergence check for mu
plot(gibbsSample[-1,2],type="l",main='Gibbs draws',xlab='Iteration',ylab='Sigmasq')
hist(gibbsSample[-1,2],freq=FALSE,main='Histogram',xlab=expression(sigma), breaks=15)
lines(density(gibbsSample[-1,2]),col="red",lwd=3)
plot(cumsum(gibbsSample[-1,2])/seq(1,iterMax-1),type="l",xlab='Iteration',ylab='Cumulative mean of mu')
```

```
## Warning in cumsum(gibbsSample[-1, 2])/seq(1, iterMax - 1): longer object
## length is not a multiple of shorter object length
```

```
abline(h=cumsum(gibbsSample[-1,2])[iterMax-1]/(iterMax-1),col="red")
acf(gibbsSample[-1,2],lag.max=20)
```



The μ and σ^2 converges to the data mean and variance as it is shown in the plots after the burn-in period and a certain number of iterations.

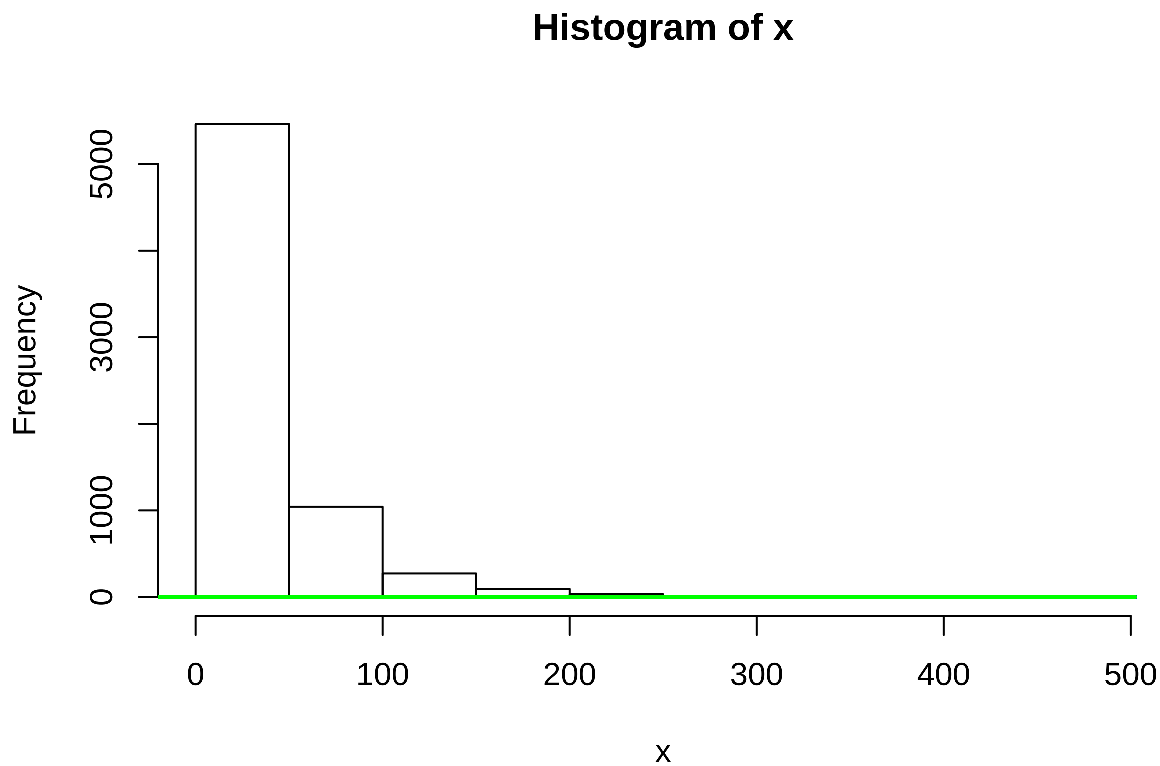
1(b) Mixture normal model

Use the Gibbs sampling data augmentation algorithm in NormalMixtureGibbs.R to analyze the daily precipitation data. Set the prior hyperparameters suitably. Evaluate the convergence of the sampler.

```
rainfall <- read.table("rainfall.dat")
names(rainfall) <- c("precipitation")

# Data options
rawData <- rainfall
x <- as.matrix(rawData['precipitation'])

# Model options
nComp <- 2         # Number of mixture components

# Prior options
alpha <- 10*rep(1,nComp)    # Dirichlet(alpha)
muPrior <- rep(0,nComp)     # Prior mean of mu
tau2Prior <- rep(10,nComp) # Prior std of mu
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 <- TRUE
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)
  piDraws <- matrix(NA,nCat,1)
  for (j in 1:nCat){
    piDraws[j] <- rgamma(1,param[j],1)
  }
  piDraws = piDraws/sum(piDraws) # Diving every column of piDraws by the sum of the elements in that co
  return(piDraws)
}
# 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)
S <- t(rmultinom(nObs, size = 1 , prob = rep(1/nComp,nComp)))
# nObs-by-nComp matrix with component allocations.

mu <- 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))
for (k in 1:nIter){
  #message(paste('Iteration number:',k))
  alloc <- S2alloc(S) # Just a function that converts between
```

```r
  # different representations of the group allocations
  nAlloc <- colSums(S)
  #print(nAlloc)
  # Update components probabilities
  pi <- rDirichlet(alpha + nAlloc)
  # Update mu'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
    mu[j] <- rnorm(1, mean = muPost, sd = sqrt(tau2Post))
  }
  # 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] - mu[j])^2))/(nu0[j] + nAlloc[j]))
  }
  # Update allocation
  for (i in 1:nObs){
    for (j in 1:nComp){
      probObsInComp[j] <- pi[j]*dnorm(x[i], mean = mu[j], sd = sqrt(sigma2[j]))
    }
    S[i,] <- t(rmultinom(1, size = 1 , prob = probObsInComp/sum(probObsInComp)))
  }


  # Printing the fitted density against data histogram
  if (plotFit && (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,mu[j],sd = sqrt(sigma2[j]))
      mixDens <- mixDens + pi[j]*compDens
      lines(xGrid, compDens, type = "l", lwd = 2, col = lineColors[j])
      components[j] <- 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)
    }
}
```



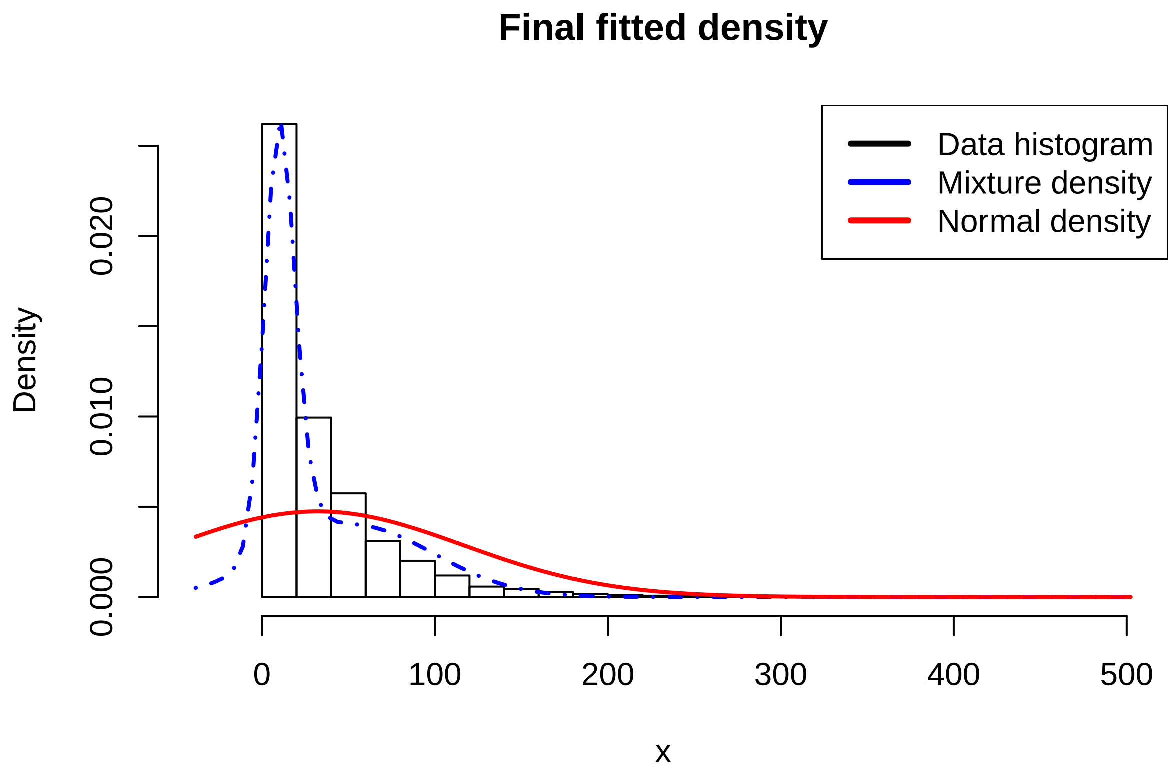
1(c) Plot the densities in one figure:

(i) A histogram or kernel density estimate of the data

(ii) Normal Density

(iii) Mixture of Normal Densities

```
# Histogram or kernel density estimate of data
hist(x, breaks = 20, freq = FALSE, xlim = c(xGridMin,xGridMax), main = "Final fitted density")

# Mixture of normal densities
lines(xGrid, mixDensMean, type = "l", lwd = 2, lty = 4, col = "blue")

# Normal density
lines(xGrid, dnorm(xGrid, mean = posteriorMu, sd = sqrt(posteriorSigmasq)),
                    type = "l", lwd = 2, col = "red")

legend("topright", box.lty = 1,
legend = c("Data histogram","Mixture density","Normal density"),
col=c("black","blue","red"), lwd = 3)
```



Question 2

2(a) Obtain the maximum likelihood estimator of beta in the Poisson regression model for the eBay data. Which covariates are significant?

```r
my_data <- read.table("eBayNumberOfBidderData.dat", header = TRUE)
glm_fit <- glm(nBids ~ . - 1, data = my_data, family = poisson(link = "log"))
summary(glm_fit)
```

```
##
## Call:
## glm(formula = nBids ~ . - 1, family = poisson(link = "log"),
##     data = my_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  < 2e-16 ***
## PowerSeller -0.02054    0.03678  -0.558   0.5765
## VerifyID    -0.39452    0.09243  -4.268 1.97e-05 ***
## Sealed       0.44384    0.05056   8.778  < 2e-16 ***
## 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 3.09e-05 ***
## MinBidShare -1.89410    0.07124 -26.588  < 2e-16 ***
## ---
## 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
```

From the summary of the model that we have fit using glm we can see that the significant covariates are the Intercept, VerifyId, MajBlem, LogBook and MinBidShare.

2(b) Bayesian Analysis of Piosson regression

```r
log_prior <- function(beta, Mu, Sigma){
dmvnorm(beta, mean = Mu, sigma = Sigma, log = TRUE)
}
log_likelihood <- function(beta, X, Y){
linear_pred <- t(X) %*% beta
prob <- Y * linear_pred - exp(linear_pred)
log_like <- sum(prob)
log_like
}
log_posterior <- function(beta, X, Y, Mu_prior, Sigma_prior){
log_likelihood(beta, X, Y) + log_prior(beta, Mu_prior, Sigma_prior)
}
X <- as.matrix(my_data[,-1])
Y <- as.matrix(my_data[,1])
Mu <- rep(0, ncol(X))
Sigma <- 100 * solve(t(X) %*% X)
result <- optim(par = matrix(rep(0, ncol(X)), ncol = 1),
fn = log_posterior, method = "BFGS", hessian = TRUE,
X = t(X), Y = Y,
Mu_prior= Mu, Sigma_prior = Sigma,
control=list(fnscale=-1))
Hessian <- result$hessian
```

2(c)

```r
Target_Density <- function(theta, Mu_prior, Sigma_prior, X, Y, ...) {
Likelihood <- dpois(Y, lambda = exp(t(X) %*% t(theta)), log = TRUE)
Prior <- dmvnorm(theta, mean = Mu_prior, sigma = Sigma_prior, log = TRUE)
sum(Likelihood) + Prior
}
proposed_density <- function(theta, Mu, Prop_Sigma, ...){
dmvnorm(theta, mean = Mu, sigma = Prop_Sigma, log = TRUE)
}
proposed_sampler <- function(Mu, Prop_Sigma, ...){
matrix(rmvnorm(1, mean = Mu, sigma = Prop_Sigma), nrow = 1)
```

```r
}
Metropolis_Hastings <- function(log_post_targ, log_prop, prop_sample,
X0, iters, ...){
x <- X0
values <- matrix(0, ncol = length(X0), nrow = iters + 1)
values[1,] <- X0
alpha <- function(x, y, ...) {
Numerator <- log_post_targ(y, ...) + log_prop(x, y, ...)
Denominator <- log_post_targ(x, ...) + log_prop(y, x, ...)
exp(Numerator - Denominator)
}
for (i in 1:iters) {
y <- prop_sample(x, ...)
u <- runif(1)
if (u < alpha(x, y, ...)) {
x <- y
}
values[i+1,] <- x
}
values
}
iters <- 10000
X0 <- rep(0, times = ncol(X))
params <- list(
log_post_targ = Target_Density,
log_prop = proposed_density,
prop_sample = proposed_sampler,
X0 = matrix(rep(0, times = ncol(X)), nrow = 1),
iters = iters,
X = t(X),
Y = Y,
Mu_prior = rep(0, times = ncol(X)),
Sigma_prior = 100 * solve(t(X) %*% X),
Prop_Sigma = 0.6 * -solve(Hessian)
)
mh_res <- do.call(Metropolis_Hastings, params)
```

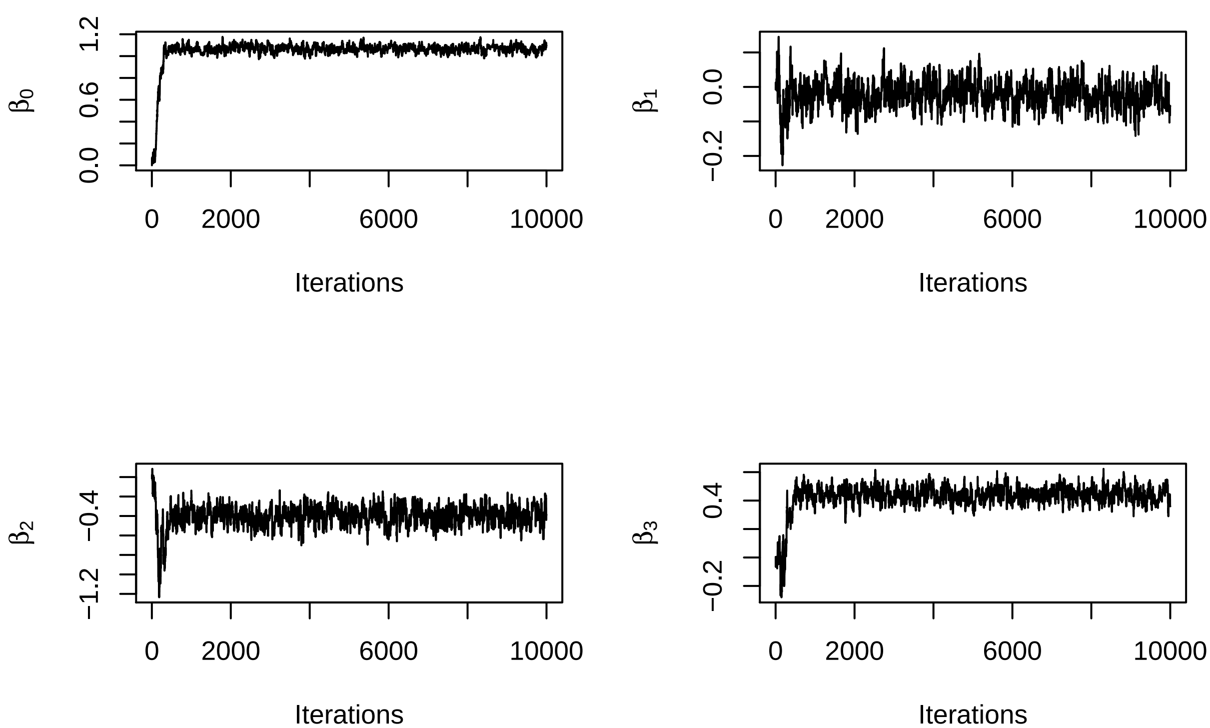
```r
Acc_prob <- as.vector(mh_res)
Avg_Acc <- mean(Acc_prob)
Beta_metro <- as.matrix(mh_res[,1:length(iters)])
phi <- exp(Beta_metro)
phi_means <- colMeans(phi)
```

```r
params <- matrix(c(coef(glm_fit), result$par, colMeans(mh_res)), nrow=3, byrow=TRUE)
colnames(params) <- c("Beta_0", "Beta_1", "Beta_2", "Beta_3", "Beta_4", "Beta_5", "Beta_6", "Beta_7", "
rownames(params) <- c("Glm", "Normal", "Metropolis")
kable(params, format="markdown", digits=5)
```

	Beta_0	Beta_1	Beta_2	Beta_3	Beta_4	Beta_5	Beta_6	Beta_7	Beta_8
Glm	1.07244	-0.02054	-0.39452	0.44384	-0.05220	-0.22087	0.07067	-0.12068	-1.89410
Normal	1.06984	-0.02051	-0.39301	0.44356	-0.05247	-0.22124	0.07070	-0.12022	-1.89199

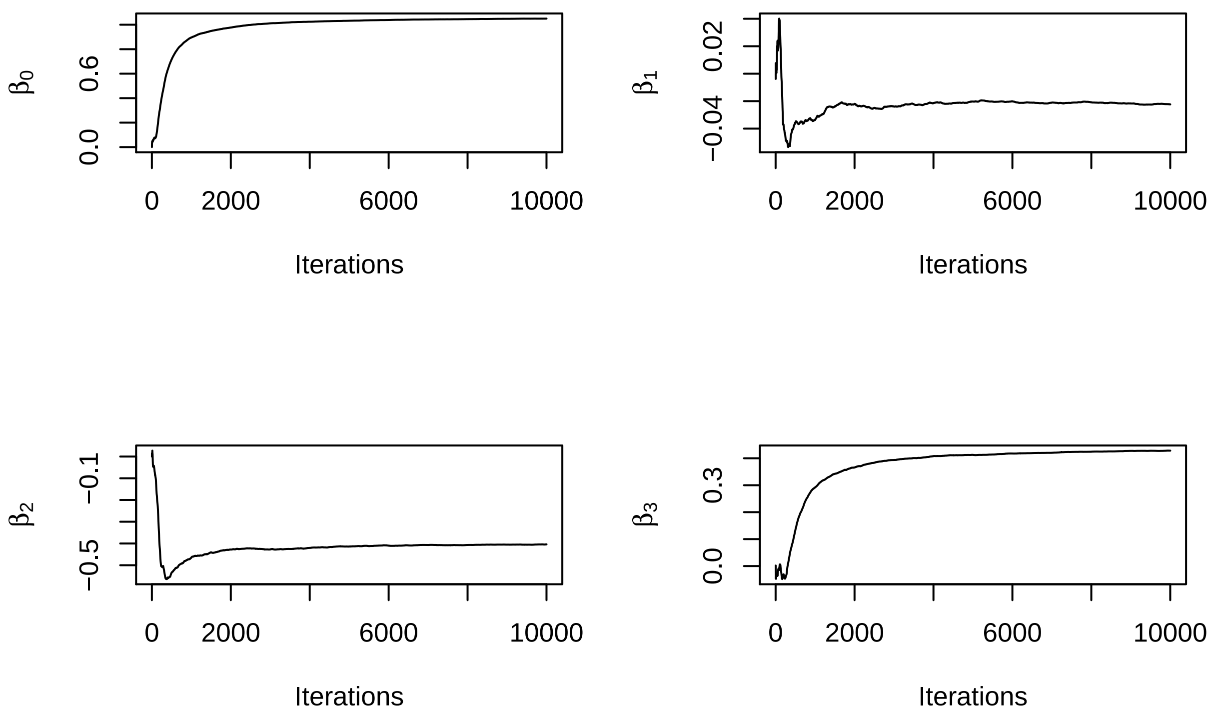
	Beta_0	Beta_1	Beta_2	Beta_3	Beta_4	Beta_5	Beta_6	Beta_7	Beta_8
Metropolis	1.04987	- 0.02235	- 0.40399	0.42811	- 0.04453	- 0.20451	0.06396	- 0.12258	- 1.86398

```r
par(mfrow=c(2,2))

plot(x = 1:nrow(mh_res), y = mh_res[,1], type = "l",
xlab="Iterations", ylab=bquote(beta[.(0)]))

plot(x = 1:nrow(mh_res), y = mh_res[,2], type = "l",
xlab="Iterations", ylab=bquote(beta[.(1)]))

plot(x = 1:nrow(mh_res), y = mh_res[,3], type = "l",
xlab="Iterations", ylab=bquote(beta[.(2)]))

plot(x = 1:nrow(mh_res), y = mh_res[,4], type = "l",
xlab="Iterations", ylab=bquote(beta[.(3)]))
```

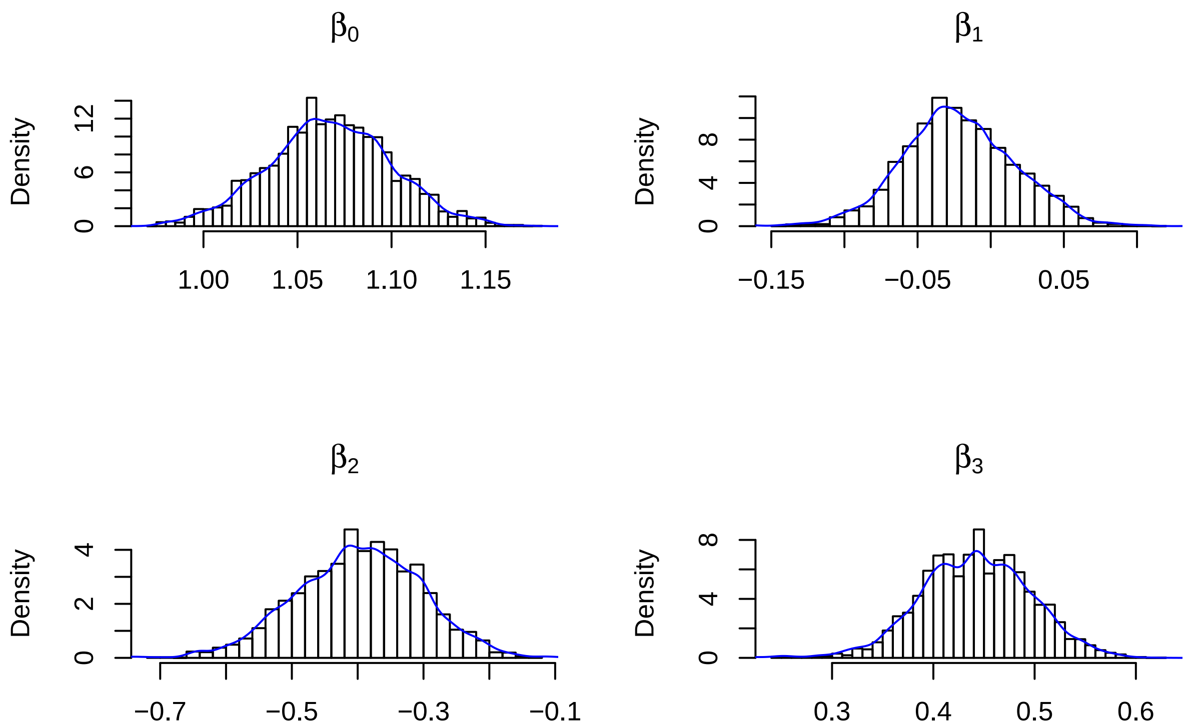



```r
Metropolis_func <- apply(mh_res, MARGIN = 2, FUN = function(x) cumsum(x) / (1:length(x)))
par(mfrow=c(2,2))

plot(x = 1:nrow(Metropolis_func), y = Metropolis_func[,1], type = "l",
xlab="Iterations", ylab=bquote(beta[.(0)]))

plot(x = 1:nrow(Metropolis_func), y = Metropolis_func[,2], type = "l",
```

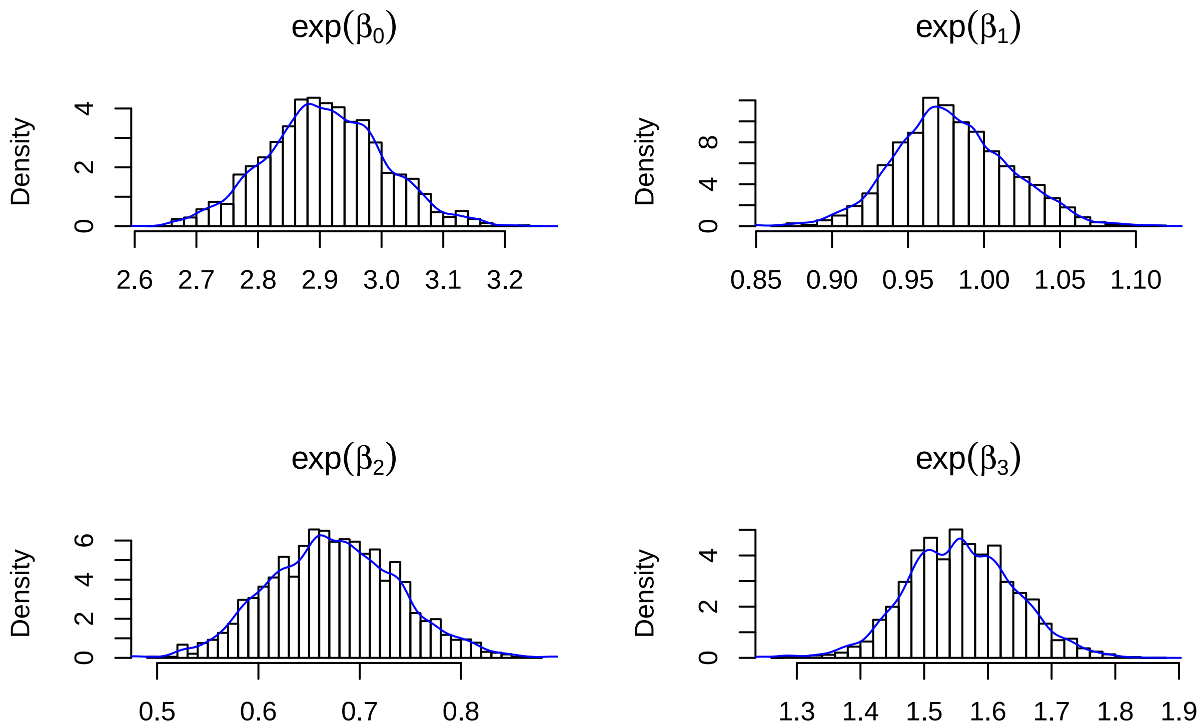
```
xlab="Iterations", ylab=bquote(beta[.(1)]))

plot(x = 1:nrow(Metropolis_func), y = Metropolis_func[,3], type = "l",
xlab="Iterations", ylab=bquote(beta[.(2)]))

plot(x = 1:nrow(Metropolis_func), y = Metropolis_func[,4], type = "l",
xlab="Iterations", ylab=bquote(beta[.(3)]))
```




```
par(mfrow=c(2,2))
for (i in 1:4){
hist(mh_res[,i][1000:iters], freq = FALSE, breaks = 30,
main =bquote(beta[.(i-1)]), xlab="", axes=TRUE)
lines(density(mh_res[,i]), col = "blue")
}
```




```
par(mfrow=c(2,2))
for (i in 1:4){
hist(exp(mh_res[,i][1000:iters]), freq = FALSE, breaks = 30,
main = bquote(exp(beta[.(i-1)])), xlab="")
lines(density(exp(mh_res[,i])), col = "blue")
}
```


exp(β_0)
Density
0
2
4
2.6
2.7
2.8
2.9
3.0
3.1
3.2
exp(β_1)
Density
0
4
8
0.85
0.90
0.95
1.00
1.05
1.10
exp(β_2)
Density
0
2
4
6
0.5
0.6
0.7
0.8
exp(β_3)
Density
0
2
4
1.3
1.4
1.5
1.6
1.7
1.8
1.9