Computer Lab 3

Christian von Koch, William Anzén

2020-05-13

Assignment 1

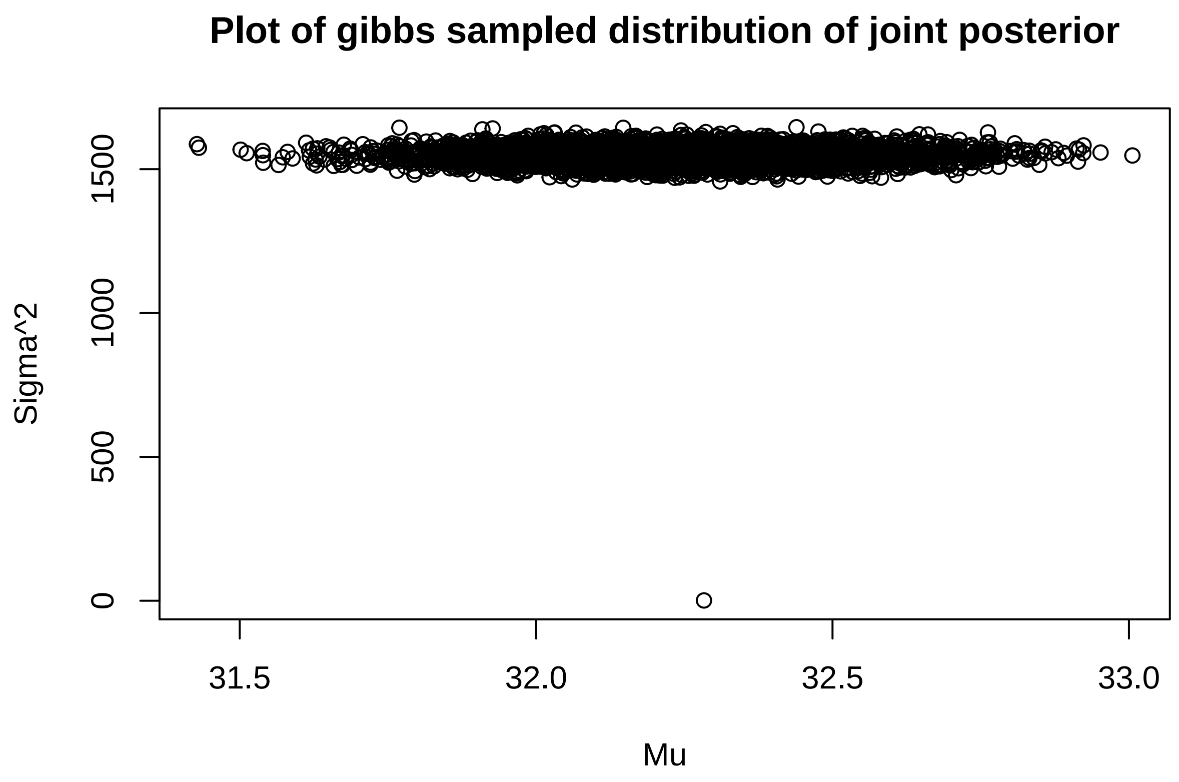
Task a)

```
## i)  Implement (code!) a Gibbs sampler that simulates from the joint posterior
## p(mu, sigma^2 given y1,...,yn). The full conditional posteriors are given on
## the slides from Lecture 7.

library(mvtnorm)
```

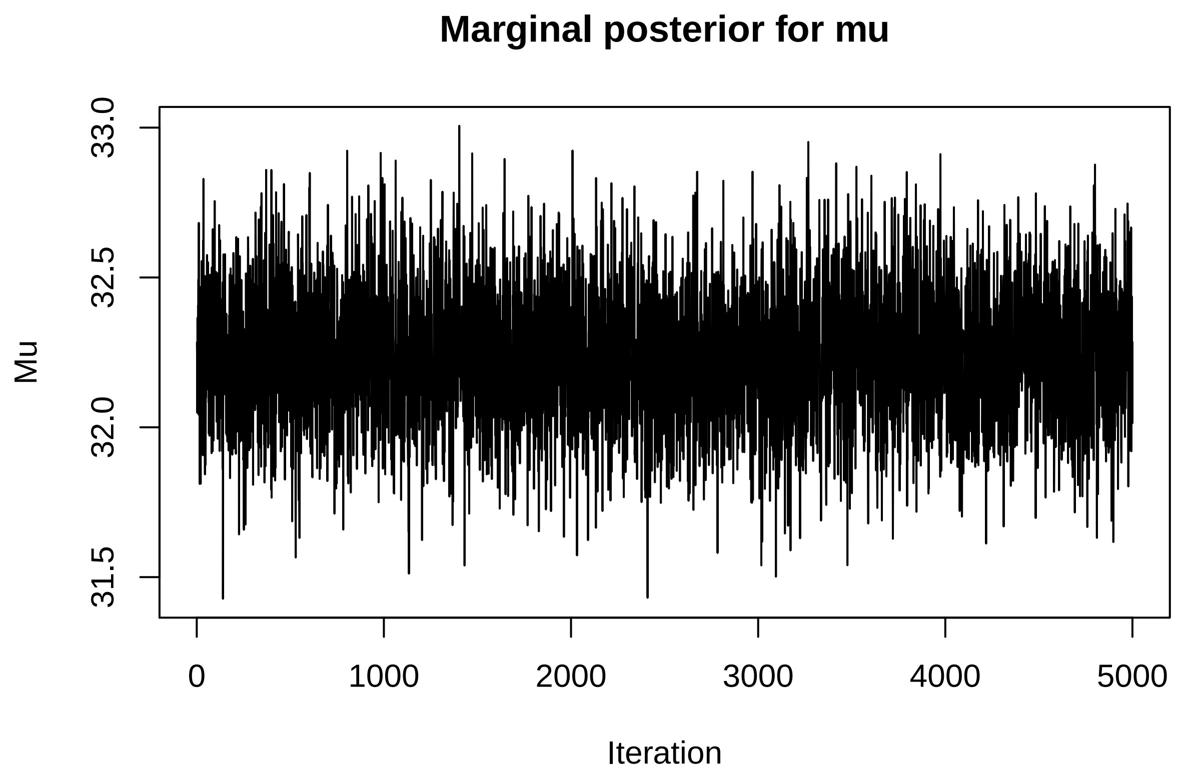
```
## Warning: package 'mvtnorm' was built under R version 3.6.2
```

```
# Read data
Rainfall = read.table("rainfall.dat")

# Setup
# Prior knowledge of mu0 taken from Google
mu0=14.79
mean_rainfall=mean(Rainfall[,1])
tao0sq=100
v0=1
sigma0sq=1
# Initial sigma value for Gibbs sampling
n=dim(Rainfall)[1]
vn=v0+n
nDraws=5000

# Function for calculating tao_n which is used as argument for the std dev for
## the normal distribution of mu
calcTaoN = function(sigmasq,tao0sq,n){
  return(1/(n/sigmasq+1/tao0sq))
}

calcMuN = function(sigmasq, tao0sq, mu0, mean, n) {
  w=(n/sigmasq)/(n/sigmasq+1/tao0sq)
  return(w*mean+(1-w)*mu0)
}

calcSigmaHat = function(v0, sigma0sq, data, mu, n) {
  return((v0*sigma0sq+sum((data-mu)^2))/(n+v0))
}
posteriorMatrix = matrix(0, nDraws, 2)
# Setting initial value of sigma^2 to 1
posteriorMatrix[1,2]=1
for (i in 1:nDraws) {
  posteriorMatrix[i,1] = rnorm(1,
                               calcMuN(posteriorMatrix[i,2],tao0sq, mu0, mean_rainfall, n),
```

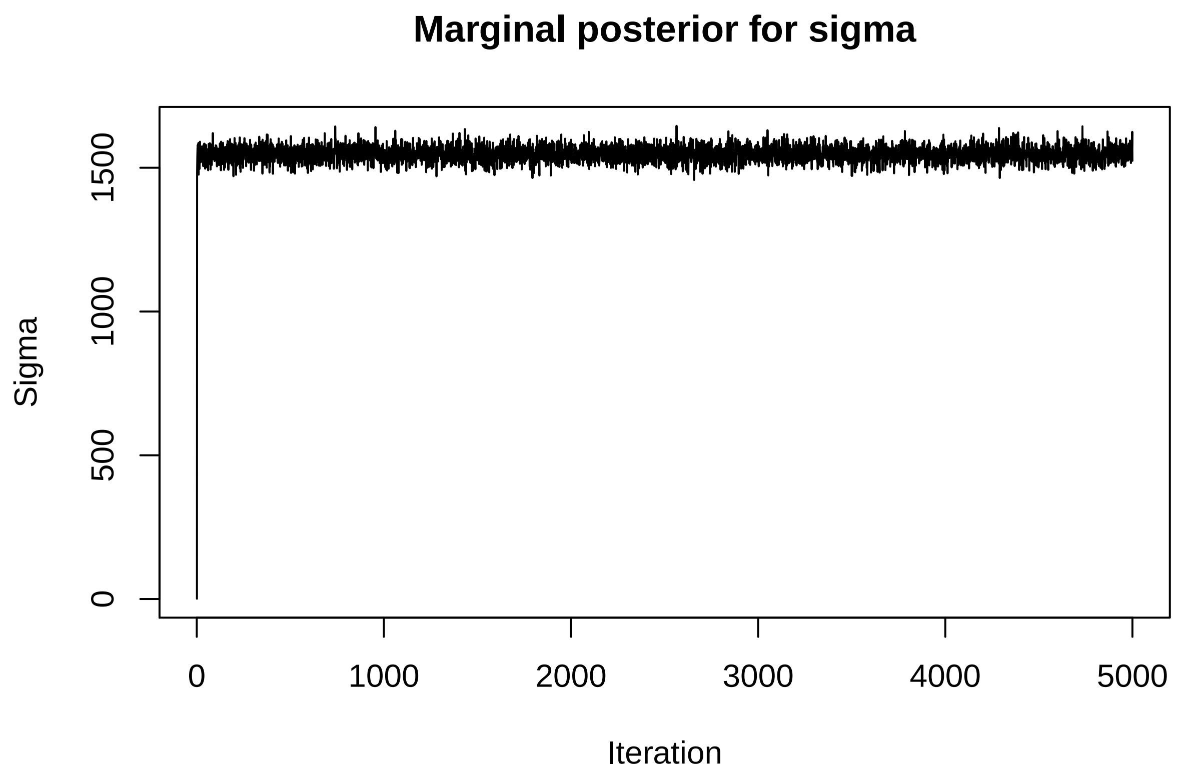
```r
                                calcTaoN(posteriorMatrix[i,2], tao0sq, n))
  if(i<nDraws) {
    drawX=rchisq(1,vn)
    posteriorMatrix[i+1,2]=vn*calcSigmaHat(v0, sigma0sq, Rainfall[,1],
                                            posteriorMatrix[i,1], n)/drawX
  }
}

# The posterior coverage
plot(posteriorMatrix[,1], posteriorMatrix[,2], xlab="Mu", ylab="Sigma^2",
     main="Plot of gibbs sampled distribution of joint posterior")
```



```r
## ii) AnalyzethedailyprecipitationusingyourGibbssamplerin(a)-i. Evaluate
## the convergence of the Gibbs sampler by suitable graphical methods, for
## example by plotting the trajectories of the sampled Markov chains.

iter=seq(1,nDraws,1)
plot(iter, posteriorMatrix[,1], type="l", xlab="Iteration", ylab="Mu",
     main="Marginal posterior for mu")
```



```r
plot(iter, posteriorMatrix[,2], type="l", xlab="Iteration", ylab="Sigma",
     main="Marginal posterior for sigma")
```

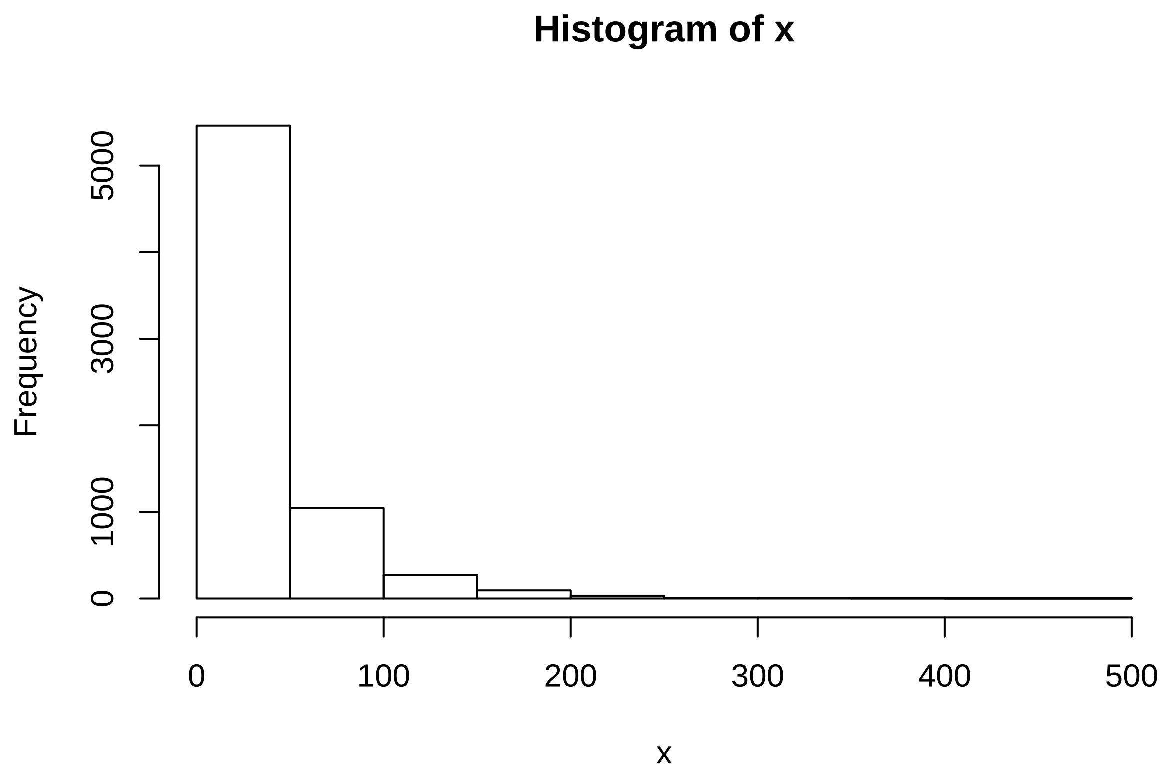


As seen in the plots above both the parameter μ and σ oscillates around a converged value. The converged values for the two parameters are approximately 32.2 and 1500 respectively.

Task b)

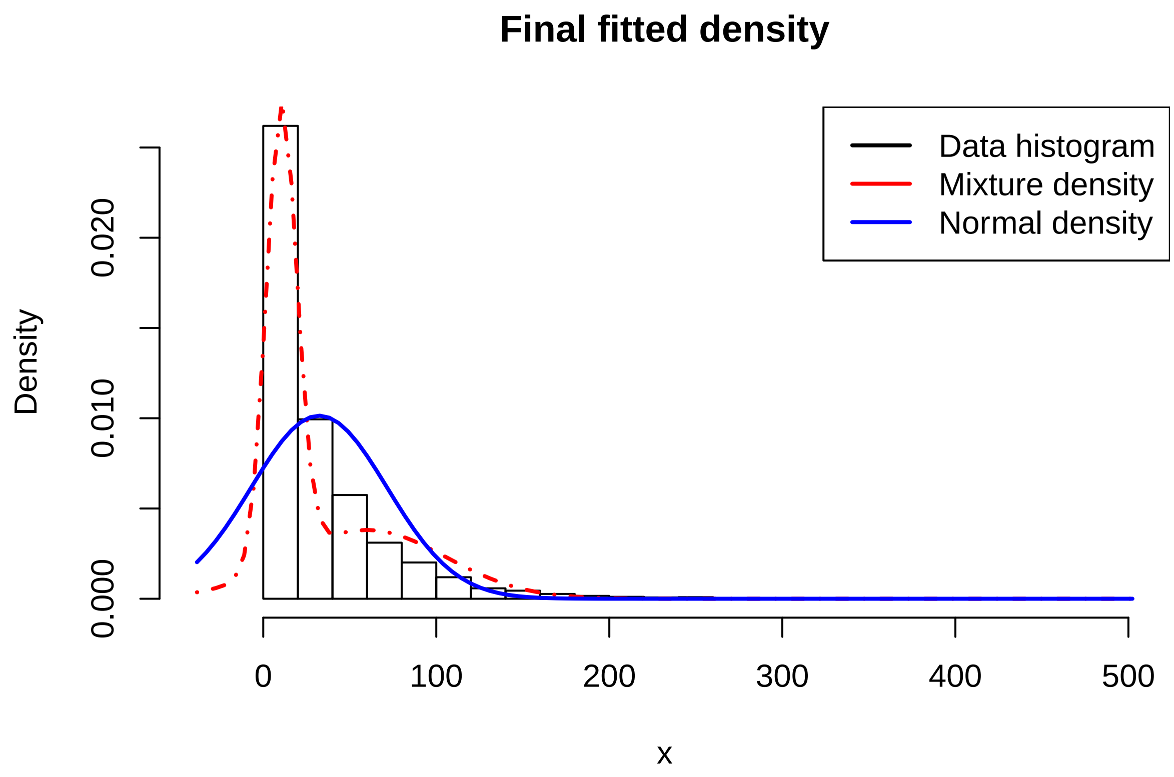
```
# NormalMixtureGibbs.R with modifications

##########    BEGIN USER INPUT #################
# Data options
x <- as.matrix(Rainfall[,1])

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

# Prior options
alpha <- rep(1,nComp) # Dirichlet(alpha)
# Obtained from Google, prior knowledge
muPrior <- c(14.79, 17.6) # Prior mean of mu
tau2Prior <- rep(100,nComp) # Prior std of mu
sigma2_0 <- rep(var(x),nComp) # s20 (best guess of sigma2)
nu0 <- rep(1,nComp) # degrees of freedom for prior on sigma2

# MCMC options
nIter <- 1000 # Number of Gibbs sampling draws

# Plotting options
```

```r
plotFit <- TRUE
lineColors <- c("blue", "green", "magenta", 'yellow')
sleepTime <- 0.01 # 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(Rainfall[,1])
S <- t(rmultinom(nObs, size = 1 , prob = rep(1/nComp,nComp))) # nObs-by-nComp matrix with component all
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){
  alloc <- S2alloc(S) # Just a function that converts between different representations of the group al
  nAlloc <- colSums(S)
  # 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){
```

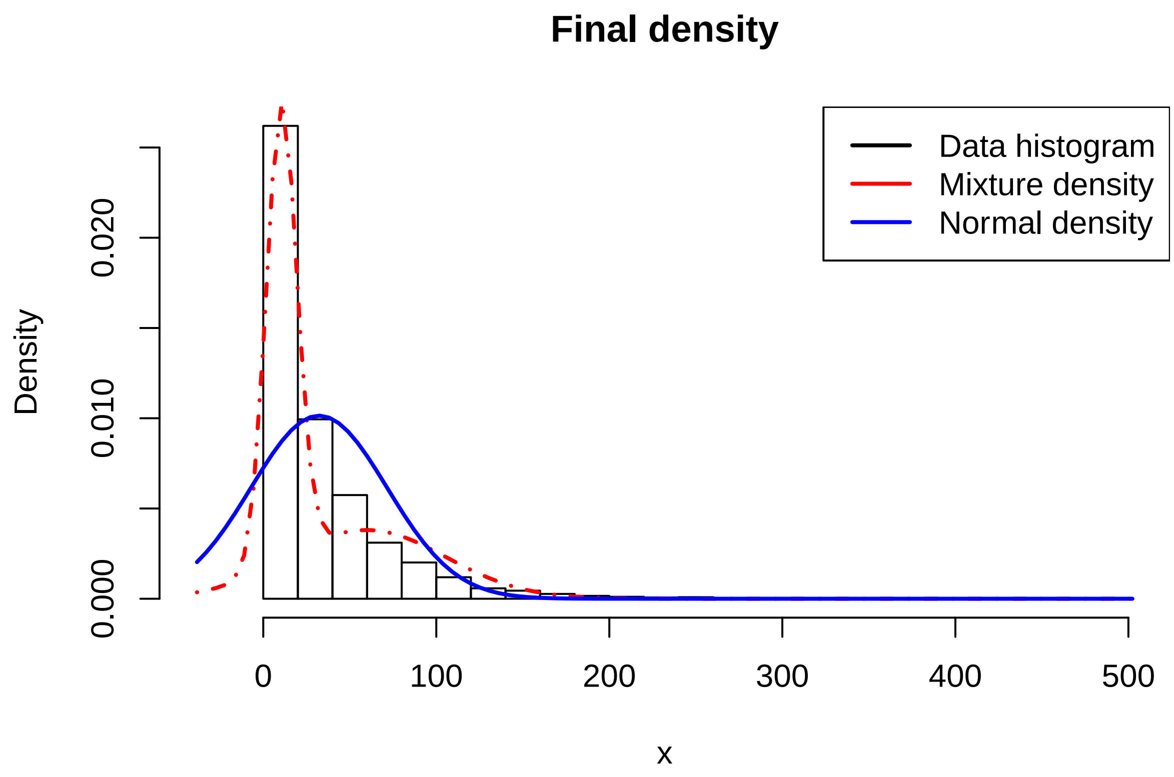
```r
    for (j in 1:nComp){
      probObsInComp[j] <- pi[j]*dnorm(x[i], mean = mu[j], sd = sqrt(sigma2[j]))
    }
    S[i,] <- t(rmultinom(1, size = 1 , prob = probObsInComp/sum(probObsInComp)))
  }

  # Printing the fitted density against data histogram
  if (plotFit && (k%%1 ==0)){
    effIterCount <- effIterCount + 1
    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
      components[j] <- paste("Component ",j)
    }
    mixDensMean <- ((effIterCount-1)*mixDensMean + mixDens)/effIterCount
  }

}


hist(x, breaks = 20, freq = FALSE, xlim = c(xGridMin,xGridMax), main = "Final fitted density")
lines(xGrid, mixDensMean, type = "l", lwd = 2, lty = 4, col = "red")
lines(xGrid, dnorm(xGrid, mean = mean(x), sd = apply(x,2,sd)), type = "l", lwd = 2, col = "blue")
legend("topright", box.lty = 1, legend = c("Data histogram","Mixture density","Normal density"),
       col=c("black","red","blue"), lwd = 2)
```



It seems like the sampler has converged towards a mixture distribution which resembles the histogram of the data. The mode of the distribution is approximately at $20 * \frac{1}{100}$ inches per day. The mixture density function seems to resemble the reality more accurately than the normal density function. It seems reasonable to apply a mixture distribution to this type of data since rain is not a constant occurance but can happen on some days, and on some days not. When going through the iterations it is apparent that the the mixture distribution converges quite quickly.

Task c)

```
hist(x, breaks = 20, freq = FALSE, xlim = c(xGridMin,xGridMax), main = "Final density")
lines(xGrid, mixDensMean, type = "l", lwd = 2, lty = 4, col = "red")
lines(xGrid, dnorm(xGrid, mean = mean(posteriorMatrix[,1]), sd = mean(sqrt(posteriorMatrix[,2]))),
      type = "l", lwd = 2, col = "blue")
legend("topright", box.lty = 1, legend = c("Data histogram","Mixture density","Normal density"),
       col=c("black","red","blue"), lwd = 2)
```



As seen in the new plot, where the only difference is the blue line, the resembles to the previous plot is obvious. The blue curve has not changed at all which is due to the fact that the mean of Gibbs sampled data, when iterations go towards infinity, converges to the true mean of the data.

Assignment 2

Task a)

```
# Read data
ebay = read.table("ebayNumberOfBidderData.dat", header=TRUE)
data = ebay[, -2]

# Create model
model = glm(nBids~., family="poisson", data=ebay)
print(model$coefficients)
```

```
## (Intercept)       Const PowerSeller    VerifyID      Sealed     Minblem
##  1.07244206          NA -0.02054076 -0.39451647  0.44384257 -0.05219829
##     MajBlem     LargNeg     LogBook MinBidShare
## -0.22087119  0.07067246 -0.12067761 -1.89409664
```

```
summary(model)
```

```
##
## Call:
## glm(formula = nBids ~ ., family = "poisson", data = ebay)
##
```

```
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.5800  -0.7222  -0.0441   0.5269   2.4605
##
## Coefficients: (1 not defined because of singularities)
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  1.07244    0.03077  34.848  < 2e-16 ***
## Const             NA         NA      NA       NA
## 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: 2151.28  on 999  degrees of freedom
## Residual deviance:  867.47  on 991  degrees of freedom
## AIC: 3610.3
##
## Number of Fisher Scoring iterations: 5
```

The covariates that are significant are $VerifyID$, *Sealed*, $MajBlem$, *LogBook*, $MinBidShare$.

Task b)

```r
library(mvtnorm)
# Defining constants
X = as.matrix(ebay[,2:ncol(ebay)])
Y = ebay[,1]
nFeatures = dim(X)[2]
covNames=names(ebay[,2:ncol(ebay)])

# Constructing prior
mu_prior = rep(0,nFeatures)
sigma_prior = 100*solve(t(X)%*%X)

# Defining function for returning the log posterior
logPostPoisson = function(beta, Y, X, mu, sigma) {
  n=length(Y)
  XBeta=beta%*%t(X)
  # Defining loglikelihood
  logLike <- sum(-log(factorial(Y))+XBeta*Y-exp(XBeta))
  # Defining prior
  prior=dmvnorm(beta, mean=mu, sigma=sigma, log=TRUE)
  # Adding loglikelihood and logprior together. Since it is log both of them are
  # added instead of multiplied
  return(logLike + prior)
}
```

```r
# Defining initial values to be passed on to the optimizer
set.seed(12345)
initVals = rnorm(dim(X)[2])

# Finding the optimized betavector
optimResult = optim(initVals, logPostPoisson, Y=Y, X=X, mu=mu_prior, sigma=sigma_prior,
                    method=c("BFGS"), control=list(fnscale=-1), hessian=TRUE)

# Defining the values of interest
postMode = optimResult$par
postCov = -solve(optimResult$hessian)
names(postMode) = covNames
approx_PostStd = sqrt(diag(postCov))
names(approx_PostStd) = covNames
print("The posterior mode is:")
```

```
## [1] "The posterior mode is:"
```

```r
print(postMode)
```

```
##       Const PowerSeller    VerifyID      Sealed     Minblem     MajBlem
##  1.06984176 -0.02051310 -0.39302504  0.44355992 -0.05246129 -0.22123414
##     LargNeg     LogBook MinBidShare
##  0.07070531 -0.12021984 -1.89197850
```

```r
print("The approximated standard deviations are:")
```

```
## [1] "The approximated standard deviations are:"
```

```r
print(approx_PostStd)
```

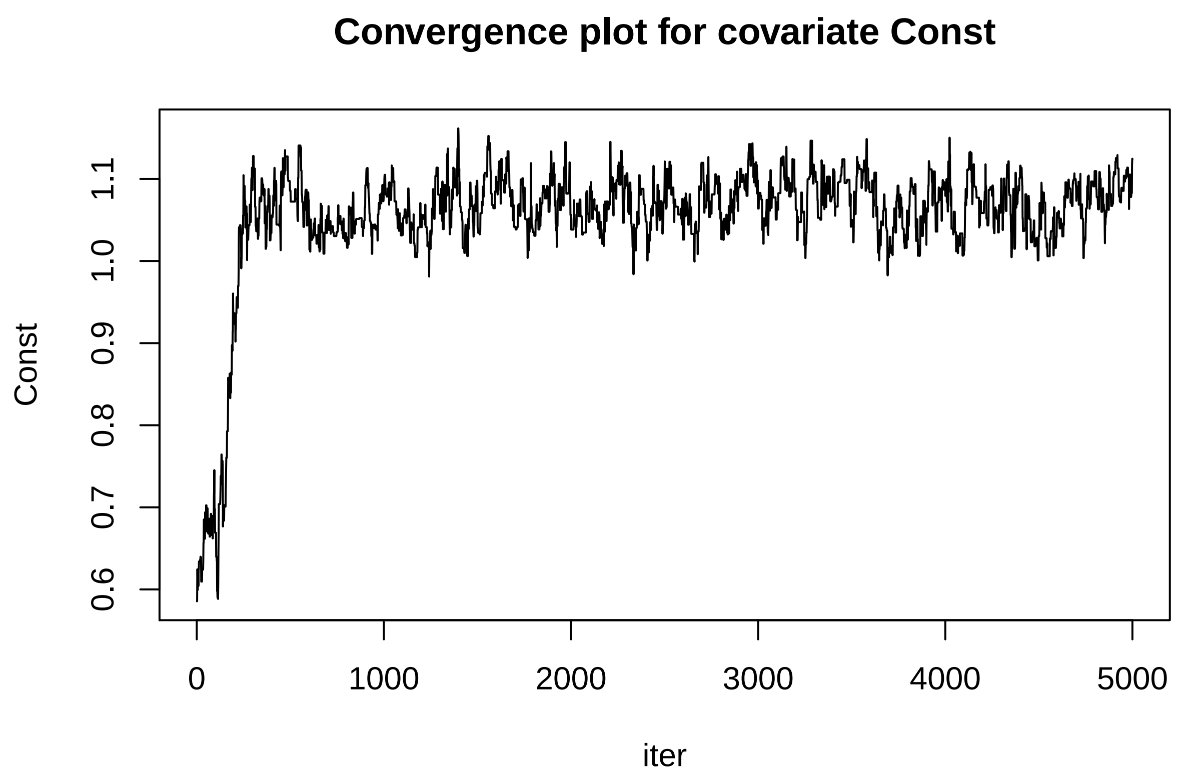
```
##       Const PowerSeller    VerifyID      Sealed     Minblem     MajBlem
##  0.03074836  0.03678419  0.09227949  0.05057447  0.06020459  0.09146061
##     LargNeg     LogBook MinBidShare
##  0.05634753  0.02895637  0.07109677
```

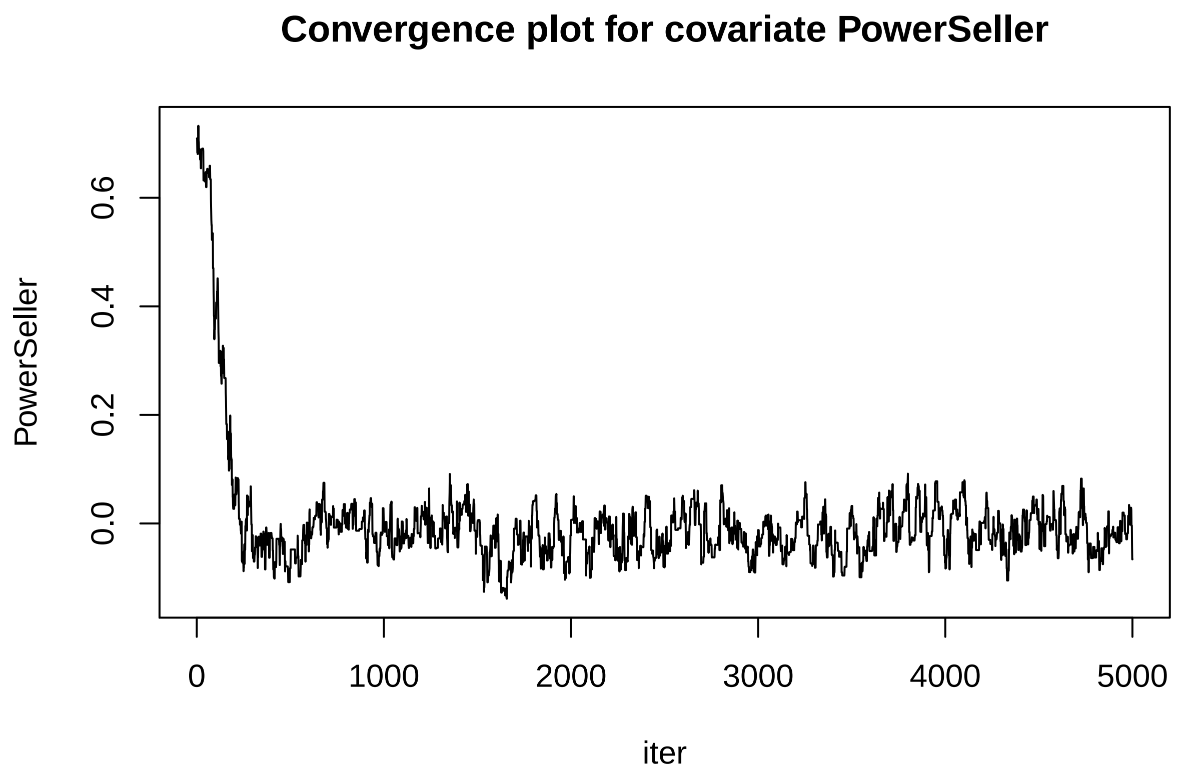
Through optimization we have obtained the optimal β as well as the hessian evaluated at the posterior mode.

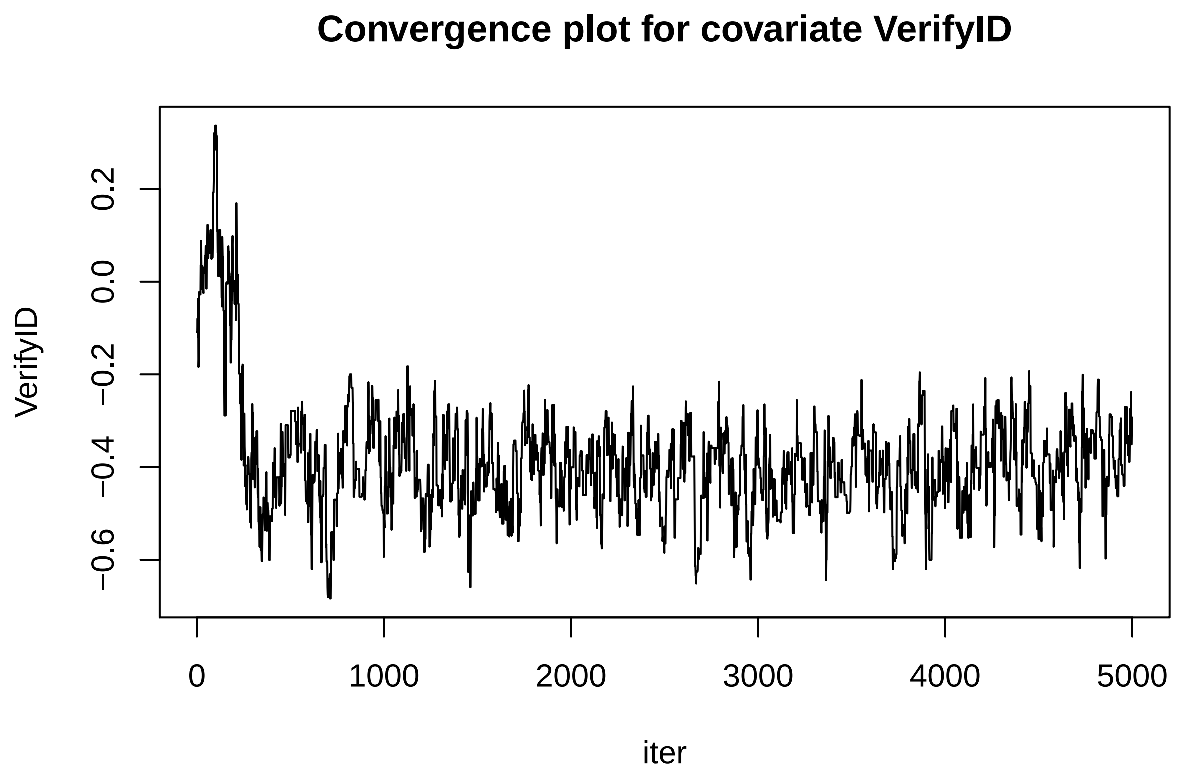
Task c)

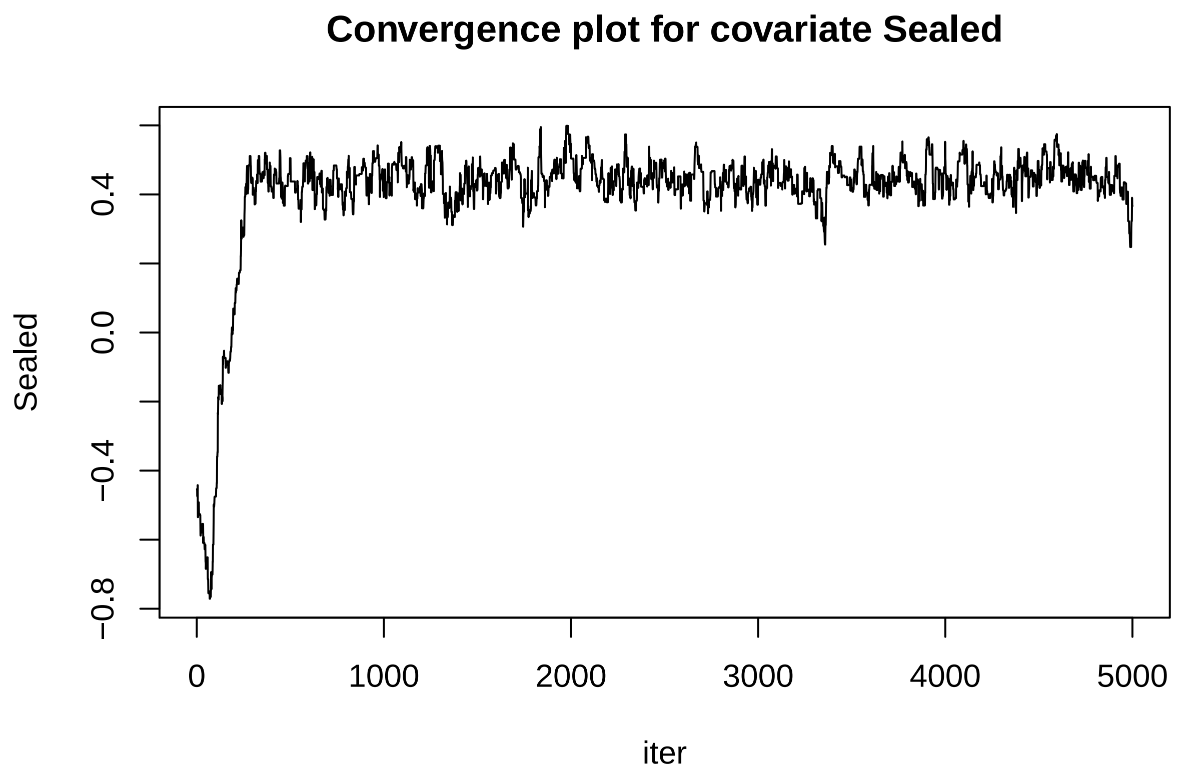
```r
# Defining function for sampling through metropolishastings
RVMSampler = function(previousVal, postCov, c, myFunction, ...) {
  proposalVal=rmvnorm(1, mean=previousVal, sigma=c*postCov)
  alpha=min(1, exp(myFunction(proposalVal,...)-myFunction(previousVal, ...)))
  u=runif(1)
  if(u < alpha) {
    return(proposalVal)
  } else {
    return(previousVal)
  }
}

nDraws=5000
beta_matrix = matrix(0, nDraws, ncol(X))
# Setting initial values of beta to same initVals as in the optimizer
# (taken randomly from normal distrib)
```

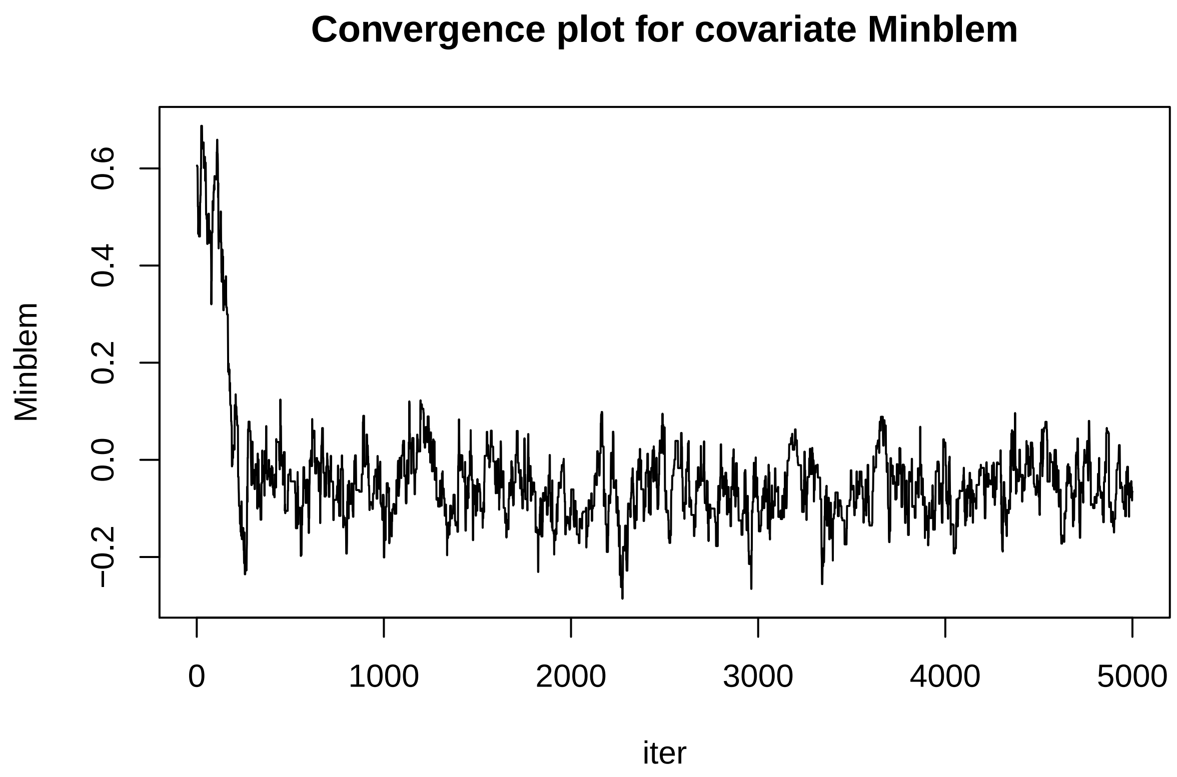
```
beta_matrix[1,]=initVals
c=0.5
set.seed(12345)

for(i in 1:nDraws) {
  if(i<nDraws) {
    beta_matrix[i+1,]=RVMSampler(beta_matrix[i,], postCov, c, logPostPoisson,
                                 Y, X, mu_prior, sigma_prior)
  }
}

iter=seq(1,nDraws,1)
for (i in 1:9) {
  plot(iter, beta_matrix[,i], type="l", main=paste("Convergence plot for covariate",
                                                   covNames[i]), ylab=covNames[i])
}
```

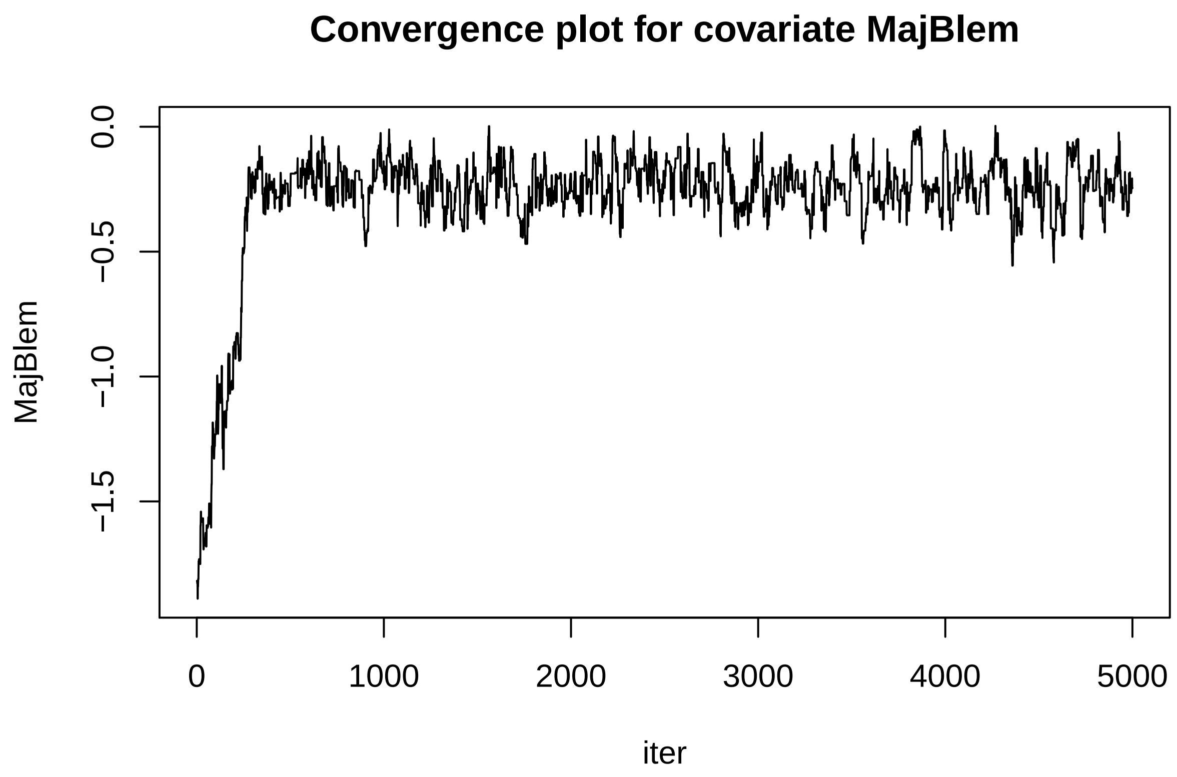


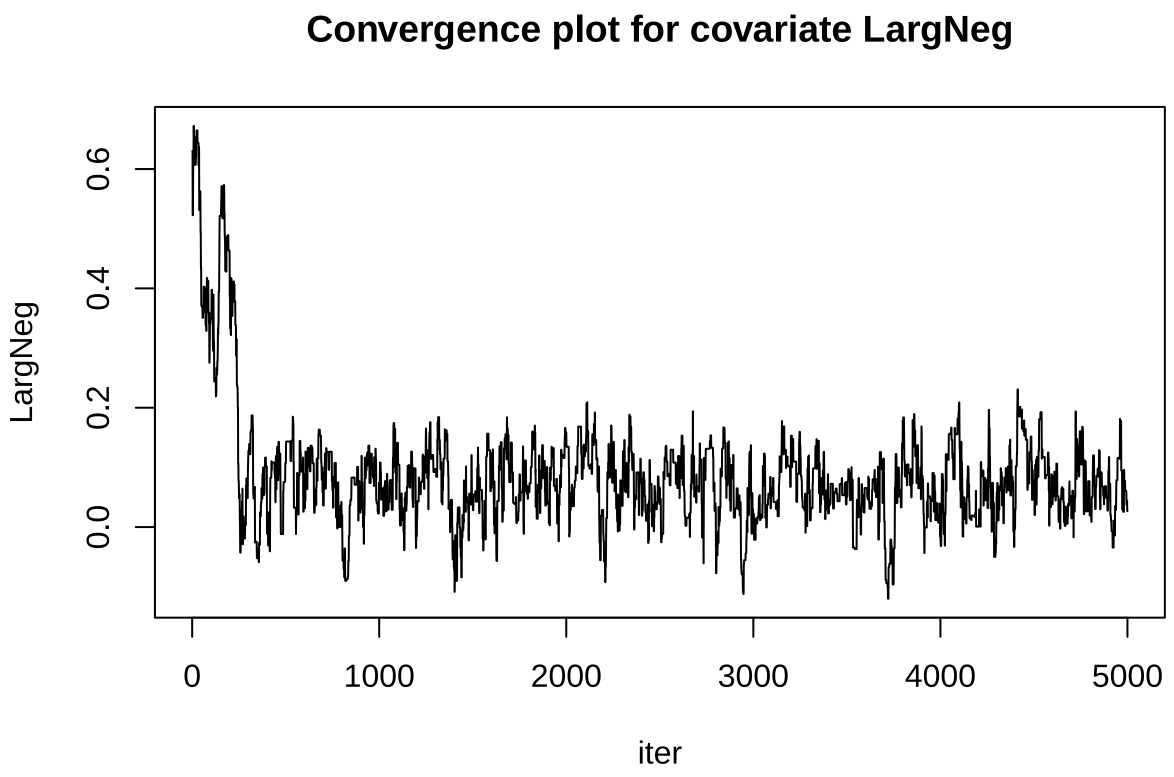

Convergence plot for covariate PowerSeller
PowerSeller
0.0
0.2
0.4
0.6
0
1000
2000
3000
4000
5000
iter

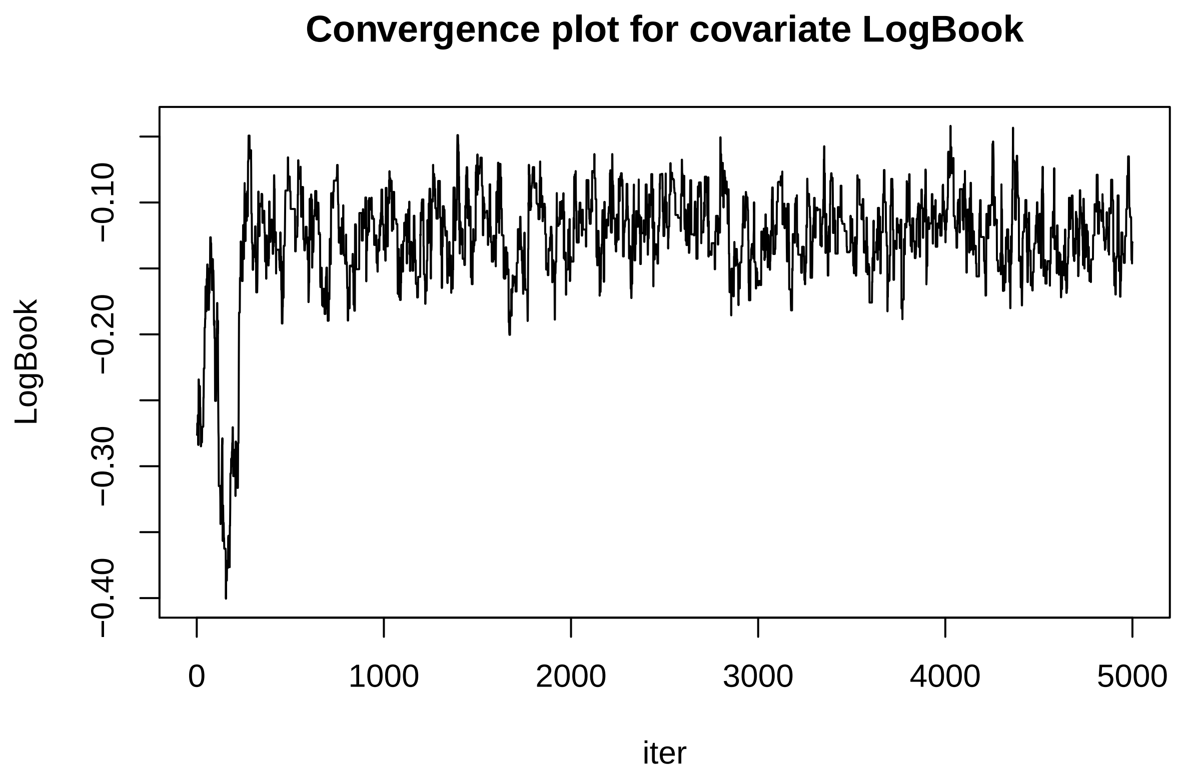

Convergence plot for covariate VerifyID
VerifyID
0.2
0.0
−0.2
−0.4
−0.6
0
1000
2000
3000
4000
5000
iter

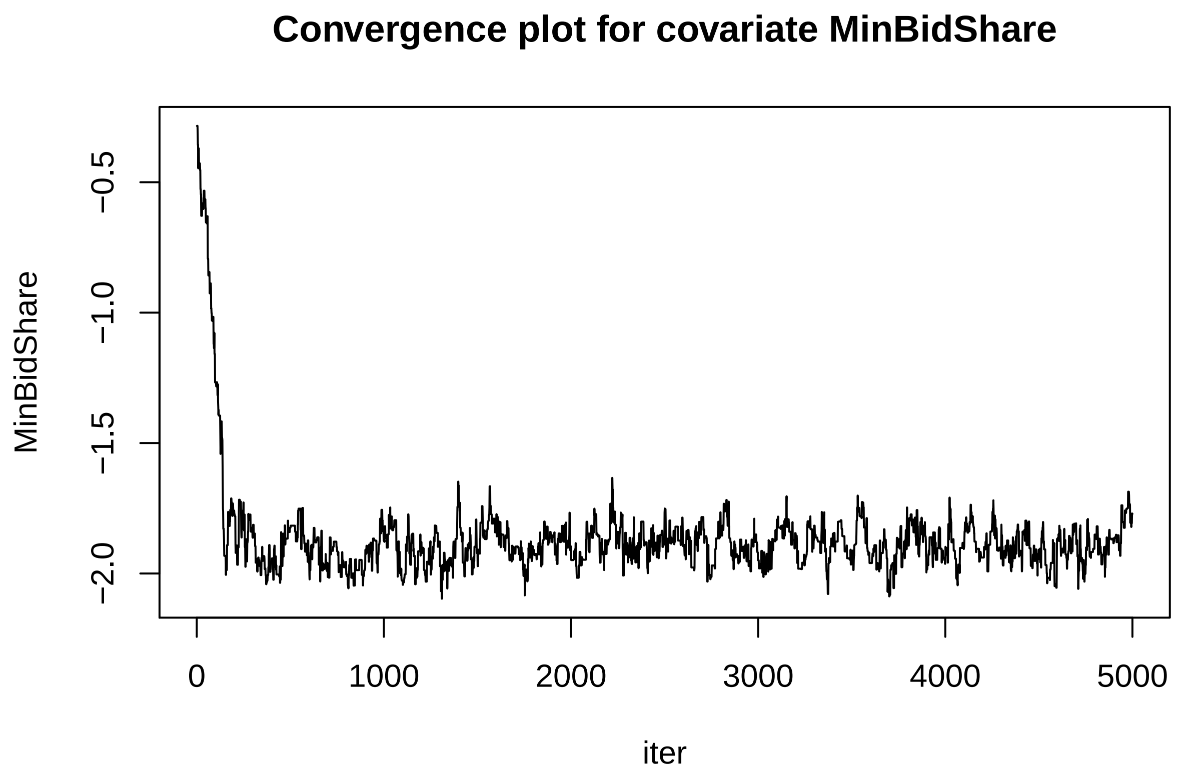

Convergence plot for covariate Sealed
Sealed
iter
0.4
0.0
−0.4
−0.8
0
1000
2000
3000
4000
5000


Convergence plot for covariate Minblem
Minblem
0.6
0.4
0.2
0.0
−0.2
0
1000
2000
3000
4000
5000
iter


Convergence plot for covariate MajBlem
MajBlem
iter
0.0
−0.5
−1.0
−1.5
0
1000
2000
3000
4000
5000


Convergence plot for covariate LargNeg
LargNeg
0.0
0.2
0.4
0.6
0
1000
2000
3000
4000
5000
iter


Convergence plot for covariate LogBook
LogBook
−0.40
−0.30
−0.20
−0.10
0
1000
2000
3000
4000
5000
iter

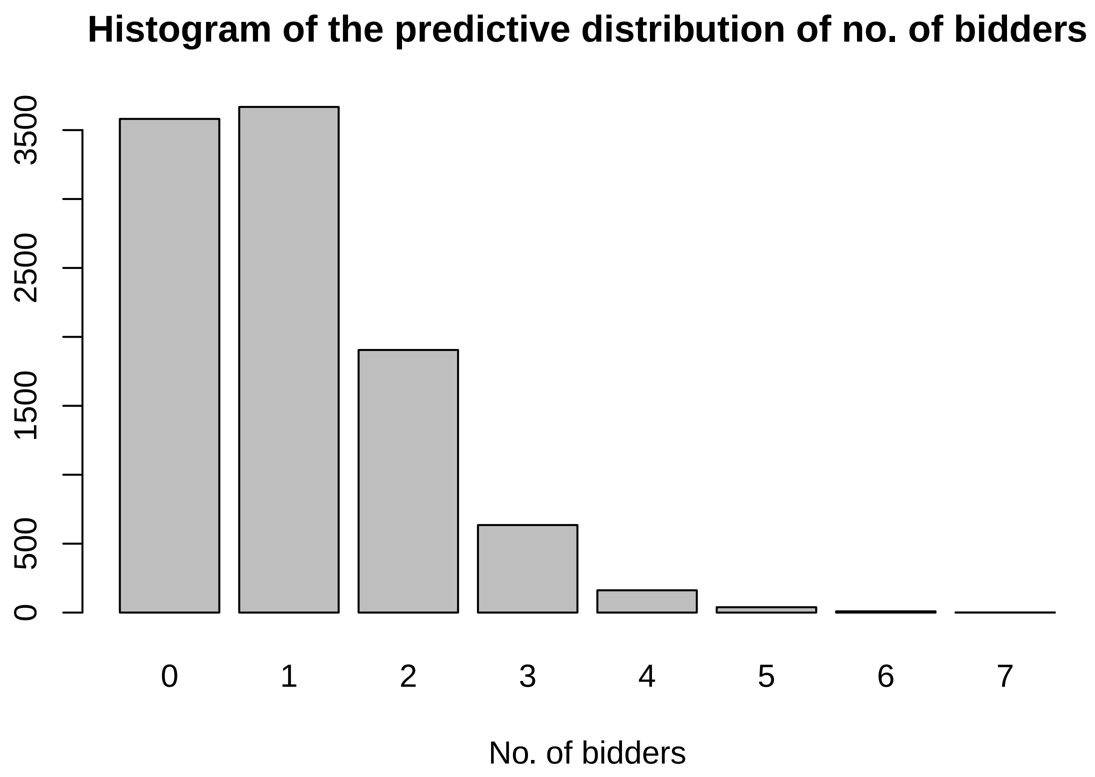


```
# Calculating distinct rows and dividing by total rows to get average
# acceptance probability
avg_alpha=dim(beta_matrix[!duplicated(beta_matrix),])[1]/dim(beta_matrix)[1]
```

As seen in the convergence plots the covariates oscillate around a similar value which was found in the previous problem where the optimal β was found through optimization. Since the variable c should be chosen in a way to acquire an average acceptance rate of approximately $25 - 30\%$, the average acceptance rate were calculated to approximately 33% which is deemed to be sufficiently satisfying.

Task d)

```
obs_X=c(1,1,1,1,0,0,0,1,0.5)
# Removing first 1000 rows since they are before the start of the convergence
approx_post_beta=beta_matrix[1001:nrow(beta_matrix),]
mean_vector=exp(approx_post_beta%*%obs_X)
set.seed(12345)
pred_distrib_bidder=rpois(10000, mean_vector)
barplot(table(pred_distrib_bidder),
        main="Histogram of the predictive distribution of no. of bidders",
        xlab="No. of bidders")
```



```
# Calculating the probability of no bidders with the given characteristics
prob_noBidders=sum(pred_distrib_bidder==0)/length(pred_distrib_bidder)
print(prob_noBidders)
```

```
## [1] 0.3581
```

As seen in the predictive distribution the majority of cases given the specified characteristics, will result in either 0 or 1 bidder with the probability decreasing for additional bidders. The calculated probability for no bidder is 0.3581.

```
## Assignment 1: The data rainfall.dat consist of daily records, from the beginning of 1948 to the end
## of precipitation (rain or snow in units of 1/100 inch, and records of zero precipitation are exluded
## Snoqualmie Falls  Washington. Analyze the data using the following two models.

## a) Assume the daily precipitation (y1,...,yn) are iid normally distributied,
## y1,...,yn given mu and sigma^2 ~ N(mu,sigma^2) where both mu and sigma^2 are unknown. Let mu ~ N(mu0
## independently of sigma^2 ~ Inv chisquare(v0, sigma0^2)
## i)  Implement (code!) a Gibbs sampler that simulates from the joint posterior p(mu, sigma^2 given y1
## The full conditional posteriors are given on the slides from Lecture 7.

library(mvtnorm)
# Read data
Rainfall = read.table("rainfall.dat")

# Setup
# Prior knowledge of mu0 taken from Google
mu0=14.79
```

```
mean_rainfall=mean(Rainfall[,1])
tao0sq=100
v0=1
sigma0sq=1
# Initial sigma value for Gibbs sampling
n=dim(Rainfall)[1]
vn=v0+n
nDraws=5000

# Function for calculating tao_n which is used as argument for the std dev for the normal distribution
calcTaoN = function(sigmasq,tao0sq,n){
  return(1/(n/sigmasq+1/tao0sq))
}

calcMuN = function(sigmasq, tao0sq, mu0, mean, n) {
  w=(n/sigmasq)/(n/sigmasq+1/tao0sq)
  return(w*mean+(1-w)*mu0)
}

calcSigmaHat = function(v0, sigma0sq, data, mu, n) {
  return((v0*sigma0sq+sum((data-mu)^2))/(n+v0))
}
posteriorMatrix = matrix(0, nDraws, 2)
# Setting initial value of sigma^2 to 1
posteriorMatrix[1,2]=1
for (i in 1:nDraws) {
  posteriorMatrix[i,1] = rnorm(1, calcMuN(posteriorMatrix[i,2],tao0sq, mu0, mean_rainfall, n),
                              calcTaoN(posteriorMatrix[i,2], tao0sq, n))
  if(i<nDraws) {
    drawX=rchisq(1,vn)
    posteriorMatrix[i+1,2]=vn*calcSigmaHat(v0, sigma0sq, Rainfall[,1], posteriorMatrix[i,1], n)/drawX
  }
}

# The posterior coverage
plot(posteriorMatrix[,1], posteriorMatrix[,2], xlab="Mu", ylab="Sigma^2")

## ii) AnalyzethedailyprecipitationusingyourGibbssamplerin(a)-i. Evaluate the convergence of the Gibbs
## by suitable graphical methods, for example by plotting the trajectories of the sampled Markov chains

iter=seq(1,nDraws,1)
plot(iter, posteriorMatrix[,1], type="l", xlab="Iteration", ylab="Mu", main="Marginal posterior for mu")
plot(iter, posteriorMatrix[,2], type="l", xlab="Iteration", ylab="Sigma", main="Marginal posterior for s

# NormalMixtureGibbs.R with modifications

##########    BEGIN USER INPUT #################
# Data options
x <- as.matrix(Rainfall[,1])

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

```r
# Prior options
alpha <- rep(1,nComp) # Dirichlet(alpha)
# Obtained from Google, prior knowledge
muPrior <- c(14.79, 17.6) # Prior mean of mu
tau2Prior <- rep(100,nComp) # Prior std of mu
sigma2_0 <- rep(var(x),nComp) # s20 (best guess of sigma2)
nu0 <- rep(1,nComp) # degrees of freedom for prior on sigma2

# MCMC options
nIter <- 1000 # Number of Gibbs sampling draws

# Plotting options
plotFit <- TRUE
lineColors <- c("blue", "green", "magenta", 'yellow')
sleepTime <- 0.01 # 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(Rainfall[,1])
S <- t(rmultinom(nObs, size = 1 , prob = rep(1/nComp,nComp))) # nObs-by-nComp matrix with component all
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)
```

```r
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 different representations of the group al
  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] <- paste("Component ",j)
    }
    mixDensMean <- ((effIterCount-1)*mixDensMean + mixDens)/effIterCount
    lines(xGrid, mixDens, type = "l", lty = 2, lwd = 3, col = 'red')
    legend("topright", box.lty = 1, legend = c("Data histogram",components, 'Mixture'),
           col = c("black",lineColors[1:nComp], 'red'), lwd = 2)
  }

}

hist(x, breaks = 20, freq = FALSE, xlim = c(xGridMin,xGridMax), main = "Final fitted density")
lines(xGrid, mixDensMean, type = "l", lwd = 2, lty = 4, col = "red")
lines(xGrid, dnorm(xGrid, mean = mean(x), sd = apply(x,2,sd)), type = "l", lwd = 2, col = "blue")
legend("topright", box.lty = 1, legend = c("Data histogram","Mixture density","Normal density"),
       col=c("black","red","blue"), lwd = 2)

## c) Plot the following densities in one figure: 1) a histogram or kernel density estimate of the data
## 2) Normal density of N(yi given mu and sigma^2) in a); 3) Mixture of normal density
## p(yi given mu, sigma^2, pi) in b). Base your plots on the mean over all posterior draws.

hist(x, breaks = 20, freq = FALSE, xlim = c(xGridMin,xGridMax), main = "Final density")
lines(xGrid, mixDensMean, type = "l", lwd = 2, lty = 4, col = "red")
lines(xGrid, dnorm(xGrid, mean = mean(posteriorMatrix[,1]), sd = mean(sqrt(posteriorMatrix[,2]))),
      type = "l", lwd = 2, col = "blue")
legend("topright", box.lty = 1, legend = c("Data histogram","Mixture density","Normal density"),
       col=c("black","red","blue"), lwd = 2)

## Assignment 2: Consider the following Poisson regression model yi given Beta ~ Poisson(exp(xiT*Beta))
## where yi isthecountforthe ithobservationinthesampleand xi isthe p-dimensional vector with covariate
## for the ith observation. Use the data set eBayNumberOfBidderData.dat. This dataset contains observat
## eBay auctions of coins. The response variable is nBids and records the number of bids in each auctio
## remaining variables are features/covariates (x):

## a) Obtain the maximum likelihood estimator of Beta 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]. Wh
## covariates are significant?

# Read data
ebay = read.table("ebayNumberOfBidderData.dat", header=TRUE)
data = ebay[, -2]

# Create model
model = glm(nBids~., family="poisson", data=ebay)
print(model$coefficients)
summary(model)

## b) Let's now do a Bayesian analysis of the Poisson regression. Let the prior be Beta~N(0,100*(XTX)^(
## is the n x p covariate matrix. This is a commonly used prior which is called Zellner's g-prior. Assu
## the posterior density is approximately multivariate normal: Beta given y ~ N(Beta~, Jy(Beta~)^(-1)) 
## is the posterior mode and Jy(Beta~) is the negative Hessian at the posterior mode. Beta~ and J can b
## by numerical optimization (optim.R) exactly like you already did for the logistic regression in Lab2

library(mvtnorm)
# Defining constants
X = as.matrix(ebay[,2:ncol(ebay)])
Y = ebay[,1]
```

```
nFeatures = dim(X)[2]
covNames=names(ebay[,2:ncol(ebay)])

# Constructing prior
mu_prior = rep(0,nFeatures)
sigma_prior = 100*solve(t(X)%*%X)

# Defining function for returning the log posterior
logPostPoisson = function(beta, Y, X, mu, sigma) {
  n=length(Y)
  XBeta=beta%*%t(X)
  # Defining loglikelihood
  logLike <- sum(-log(factorial(Y))+XBeta*Y-exp(XBeta))
  # Defining prior
  prior=dmvnorm(beta, mean=mu, sigma=sigma, log=TRUE)
  # Adding loglikelihood and logprior together. Since it is log both of them are added instead of multi
  return(logLike + prior)
}

# Defining initial values to be passed on to the optimizer
set.seed(12345)
initVals = rnorm(dim(X)[2])

# Finding the optimized betavector
optimResult = optim(initVals, logPostPoisson, Y=Y, X=X, mu=mu_prior, sigma=sigma_prior, method=c("BFGS")
                    control=list(fnscale=-1), hessian=TRUE)

# Defining the values of interest
postMode = optimResult$par
postCov = -solve(optimResult$hessian)
names(postMode) = covNames
approx_PostStd = sqrt(diag(postCov))
names(approx_PostStd) = covNames
print("The posterior mode is:")
print(postMode)
print("The approximated standard deviations are:")
print(approx_PostStd)

## c) Now, let's simulate from the actual posterior of beta using the Metropolis algorithm and compare
## approximate results in b). Program a general function that uses the Metropolis algorithm to generate
## draws from an arbitrary posterior density. In order to show that it is a general function for any mo
## denote the vector of model parameters by theta. Let the proposal density be the multivariate normal
## mentioned in Lecture 8 (random walk Metropolis): Theta_p given Theta(i-1) ~ N(Theta(i-1), c*Cov) whe
## Cov = Jy(Beta~)^(-1) obtained in b). The value c is a tuning parameter and should be an input to you
## function. The user of your Metropolis function should be able to supply her own posterior density fu
## necessarily for the Poisson regression, and still be able to use your Metropolis function. This is n
## straightforward, unless you have come across function objects in R and the triple dot (...) wildcard
## I have posted a note (HowToCodeRWM.pdf) on the course web page that describes how to do this in R. N
## new Metropolis function to sample from the posterior of beta in the Poisson regression for the eBay
## Assess MCMC convergence by graphical methods.

# Defining function for sampling through metropolishastings
RVMSampler = function(previousVal, postCov, c, myFunction, ...) {
  proposalVal=rmvnorm(1, mean=previousVal, sigma=c*postCov)
  alpha=min(1, exp(myFunction(proposalVal,...)-myFunction(previousVal, ...)))
  u=runif(1)
  if(u < alpha) {
    return(proposalVal)
  } else {
    return(previousVal)
  }
}

nDraws=5000
beta_matrix = matrix(0, nDraws, ncol(X))
# Setting initial values of beta to same initVals as in the optimizer (taken randomly from normal distr
beta_matrix[1,]=initVals
c=0.5
set.seed(12345)

for(i in 1:nDraws) {
  if(i<nDraws) {
    beta_matrix[i+1,]=RVMSampler(beta_matrix[i,], postCov, c, logPostPoisson, Y, X, mu_prior, sigma_pric
  }
}

iter=seq(1,nDraws,1)
for (i in 1:9) {
  plot(iter, beta_matrix[,i], type="l", main=paste("Convergence plot for covariate", covNames[i]),
       ylab=covNames[i])
}

# Calculating distinct rows and dividing by total rows to get average acceptance probability
avg_alpha=dim(beta_matrix[!duplicated(beta_matrix),])[1]/dim(beta_matrix)[1]

## d) Use the MCMC draws from c) to simulate from the predictive distribution of the number of bidders
## auction with the characteristics below. Plot the predictive distribution. What is the probability of
## in this new auction? Use vector x=c(1,1,1,1,0,0,0,1,0.5)

obs_X=c(1,1,1,1,0,0,0,1,0.5)
# Removing first 1000 rows since they are before the start of the convergence
approx_post_beta=beta_matrix[1001:nrow(beta_matrix),]
mean_vector=exp(approx_post_beta%*%obs_X)
set.seed(12345)
pred_distrib_bidder=rpois(10000, mean_vector)
barplot(table(pred_distrib_bidder),
        main="Histogram of the predictive distribution of no. of bidders", xlab="No. of bidders")
# Calculating the probability of no bidders with the given characteristics
prob_noBidders=sum(pred_distrib_bidder==0)/length(pred_distrib_bidder)
print(prob_noBidders)
```