Lab3

Namita Sharma, Aman Kumar Nayak

5/17/2020

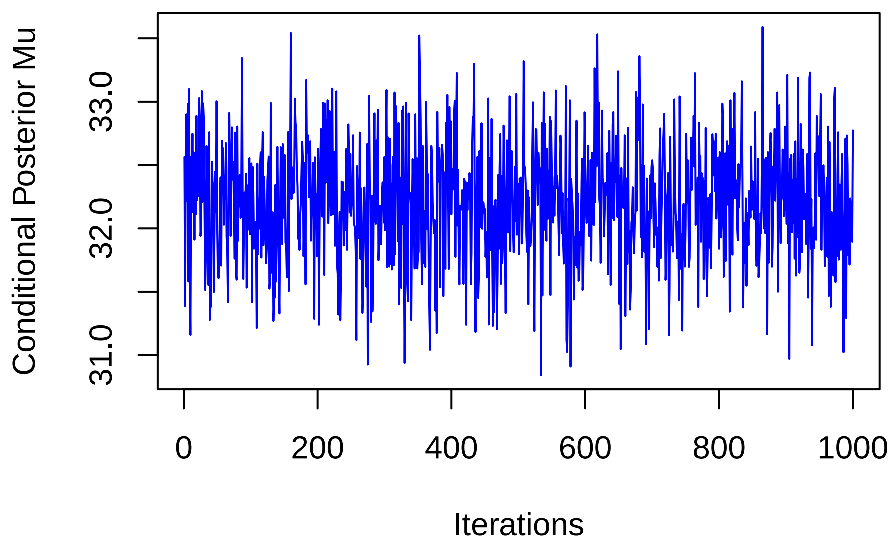
1. Normal model, mixture of normal model with semi-conjugate prior

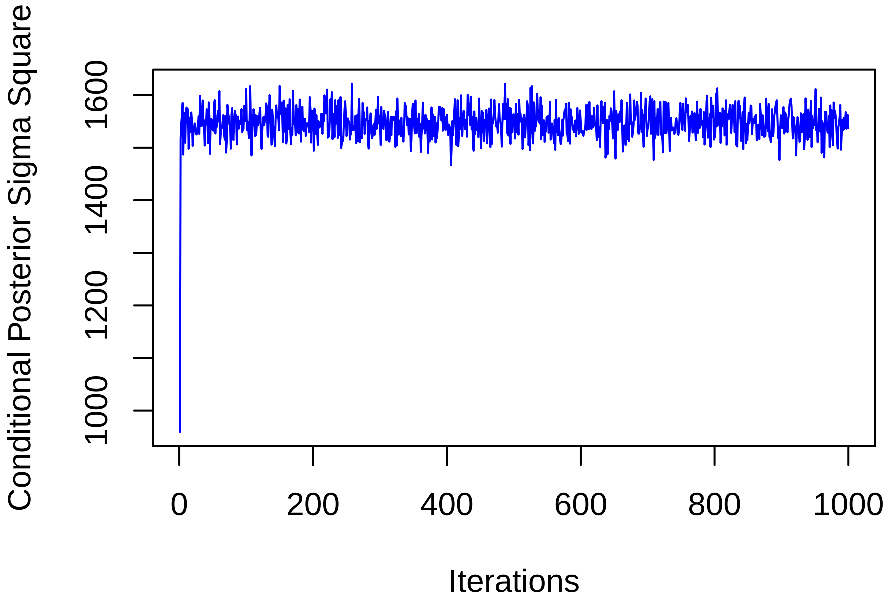
(a) Normal model

(i) Gibbs sampler

Code in appendix 1 (a)

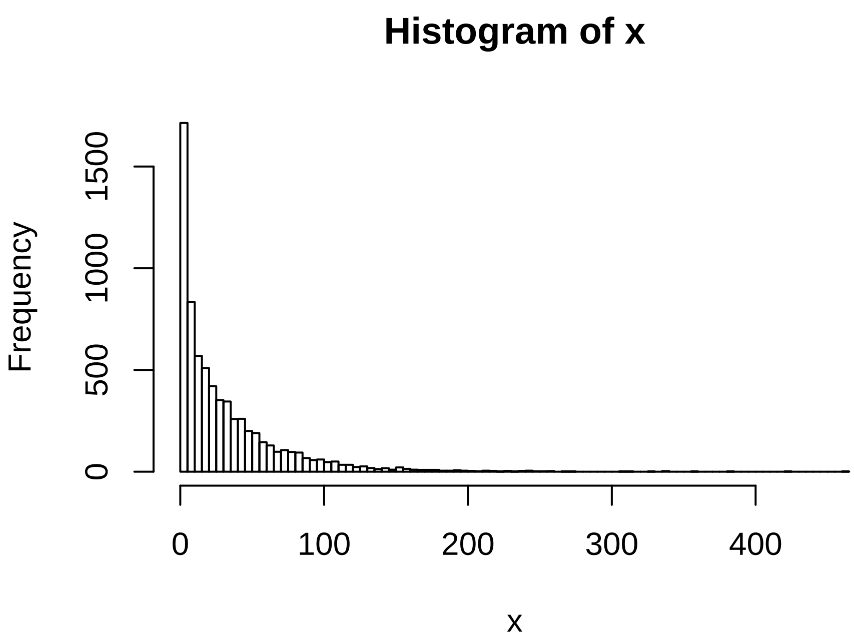
(ii) Convergence of gibbs sampler

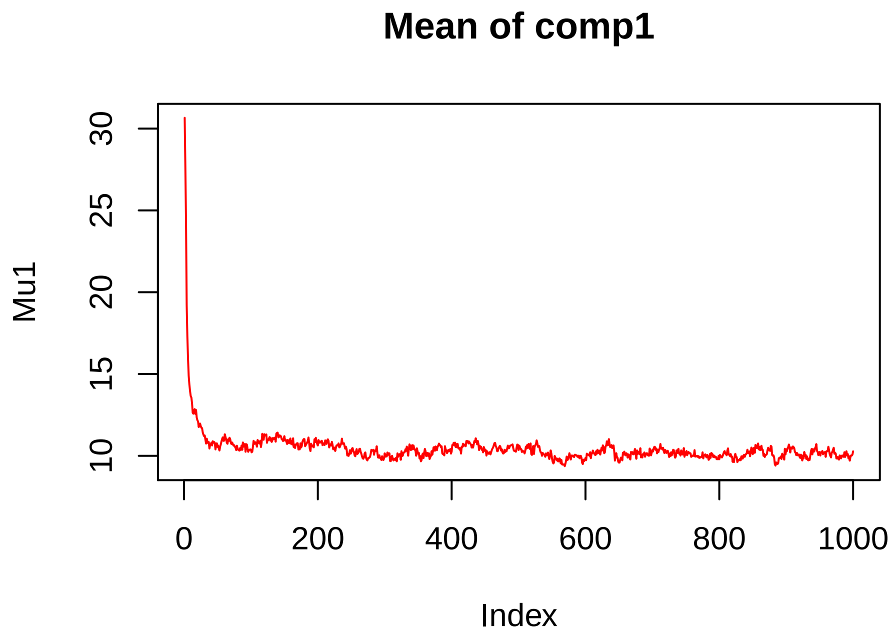


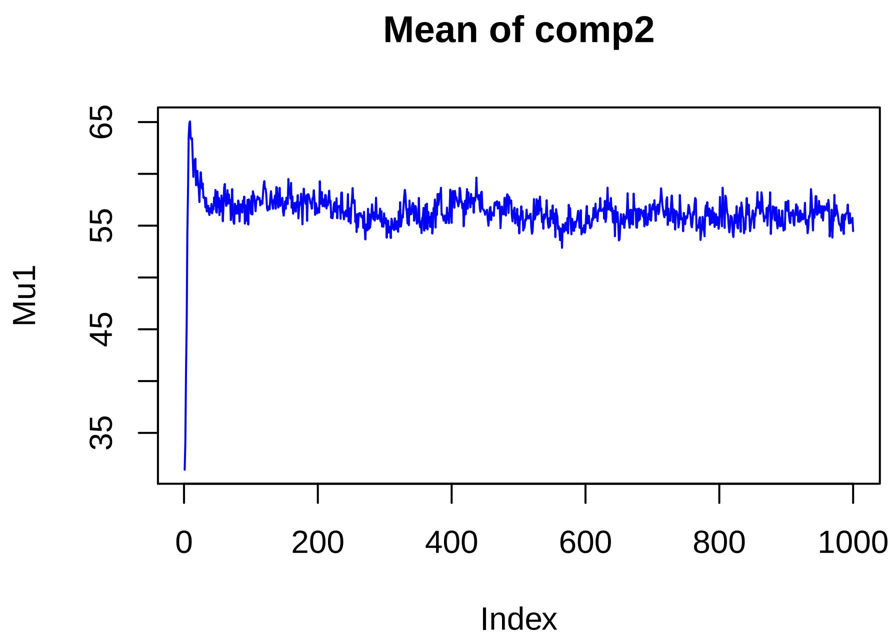


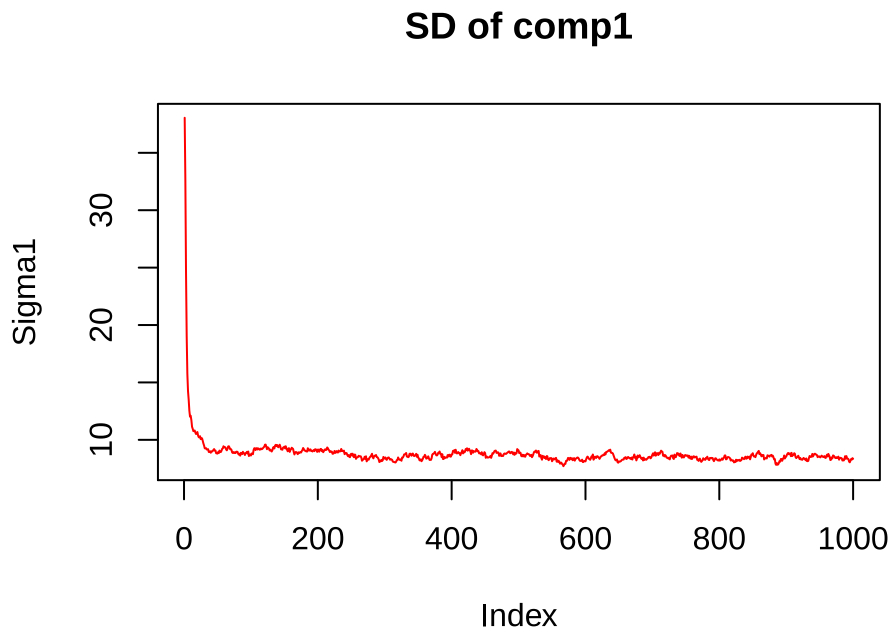
We can see from the traceplots that the gibbs sampler converges for both parameters μ and σ^2. We set the prior parameter values to $\mu_0 = 1$, τ_0^2=100, ν0=1, σ_0^2=100 as we assume an uninformative prior without seeing any data points (only 1 data point) and assume a random variance 100. It can be observed that the gibbs sampler converges approximately to a mean of 32.2 and a variance of 1550.

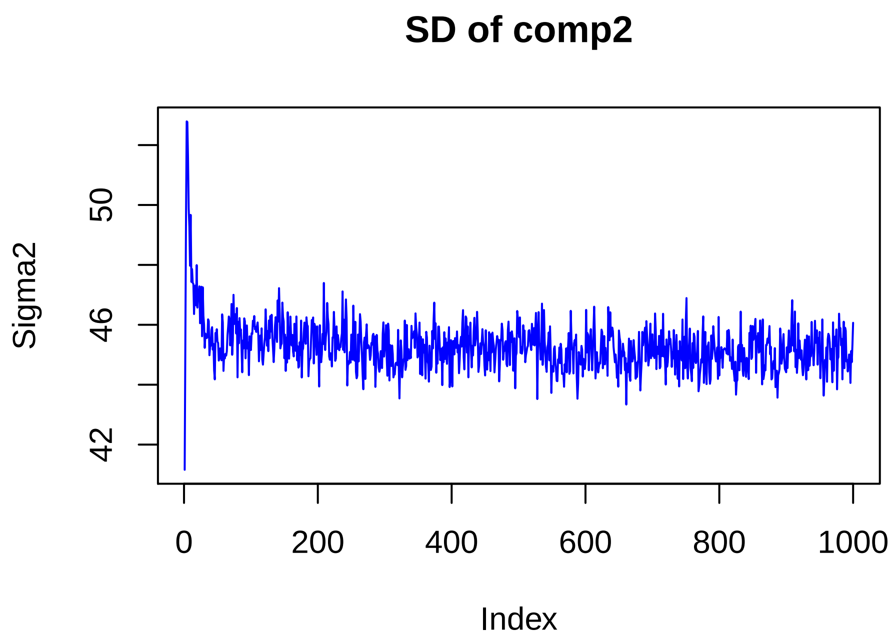
(b) Mixture normal model

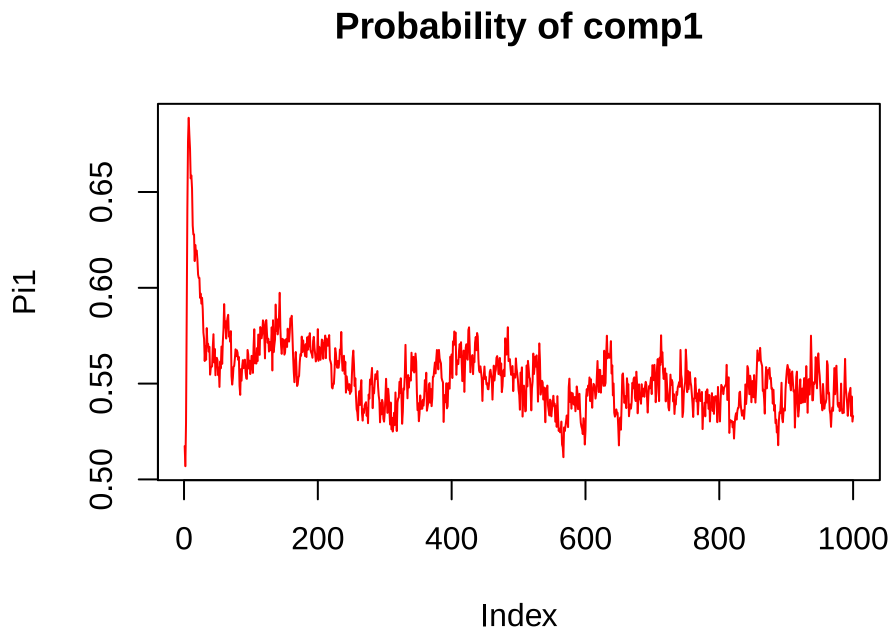


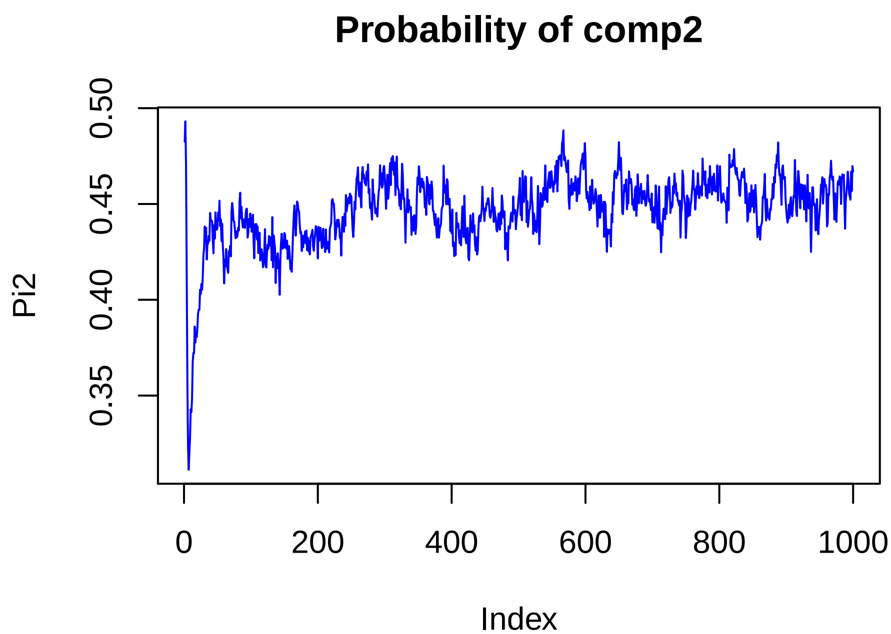

Mean of comp1
Mu1
Index
0
200
400
600
800
1000
10
15
20
25
30


Mean of comp2
Mu1
Index
0
200
400
600
800
1000
35
45
55
65


SD of comp1
Sigma1
10
20
30
0
200
400
600
800
1000
Index

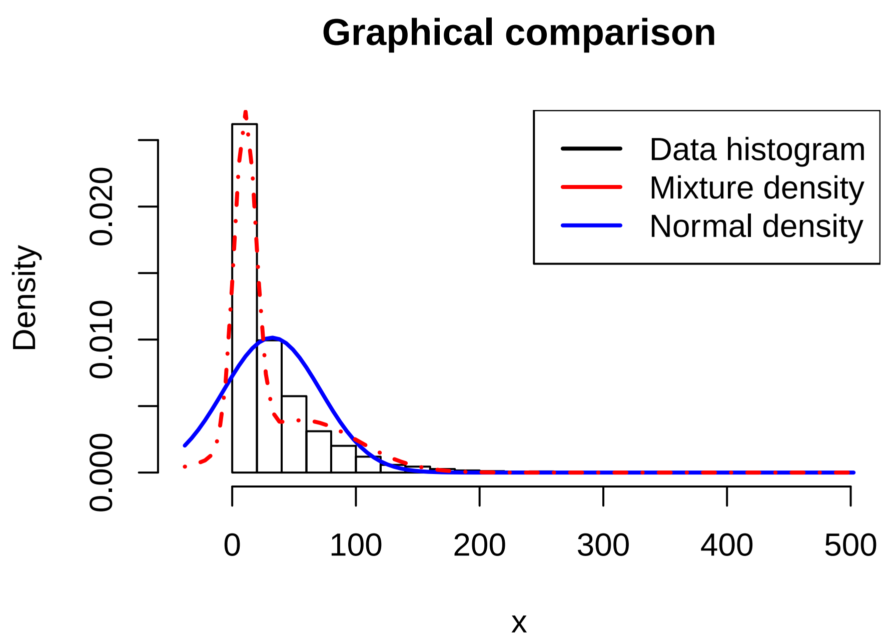

SD of comp2
Sigma2
42
46
50
0
200
400
600
800
1000
Index

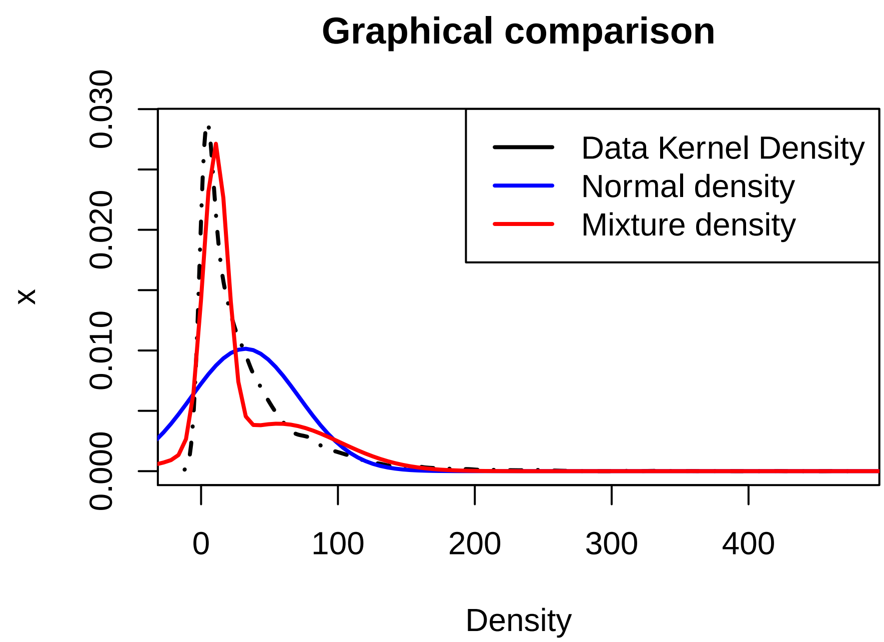




We can see from the traceplots, that after about 100 iterations (burn-in), all the parameters of the mixture densities seem to achieve convergence. We can see from the trajectories that the component means converge to values $\mu = [55, 10]$, the component standard deviations converge to approximately $\sigma = [5, 45]$ and the final component probabilities are approximately $\pi = [0.55, 0.45]$

(c) Graphical comparison





The mixture model does a much better job of capturing the data density than the joint normal posterior density evaluated using the gibbs sampler.

2. Metropolis Random Walk for Poisson regression

(a) MLE estimator of β

```
##
## Call:
## glm(formula = nBids ~ . - Const, family = "poisson", data = eBayData)
##
## 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|)
## (Intercept)  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: 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
```

Looking at pValue we can see that below covariates are most significant:

1. VerifyID
2. Sealed
3. LogBook
4. MinBidShare
5. MajBlem

(b) Bayesian analysis of the Poisson regression

```
## The posterior mode is:
##  1.069841 -0.02051246 -0.393006 0.4435555 -0.05246627 -0.2212384 0.07069683 -0.1202177 -1.891985
```

```
## The posterior variance-covariance matrix is:
```

0.0009455	-0.0007139	-0.0002742	-0.0002709	-0.0004455	-0.0002772	-0.0005128	0.0000644	0.0011099
-0.0007139	0.0013531	0.0000402	-0.0002949	0.0001143	-0.0002083	0.0002802	0.0001182	-0.0005686
-0.0002742	0.0000402	0.0085154	-0.0007825	-0.0001014	0.0002283	0.0003314	-0.0003192	-0.0004293
-0.0002709	-0.0002949	-0.0007825	0.0025578	0.0003577	0.0004532	0.0003376	-0.0001311	-0.0000576
-0.0004455	0.0001143	-0.0001014	0.0003577	0.0036246	0.0003492	0.0000584	0.0000585	-0.0000644
-0.0002772	-0.0002083	0.0002283	0.0004532	0.0003492	0.0083651	0.0004049	-0.0000898	0.0002622
-0.0005128	0.0002802	0.0003314	0.0003376	0.0000584	0.0004049	0.0031751	-0.0002542	-0.0001063

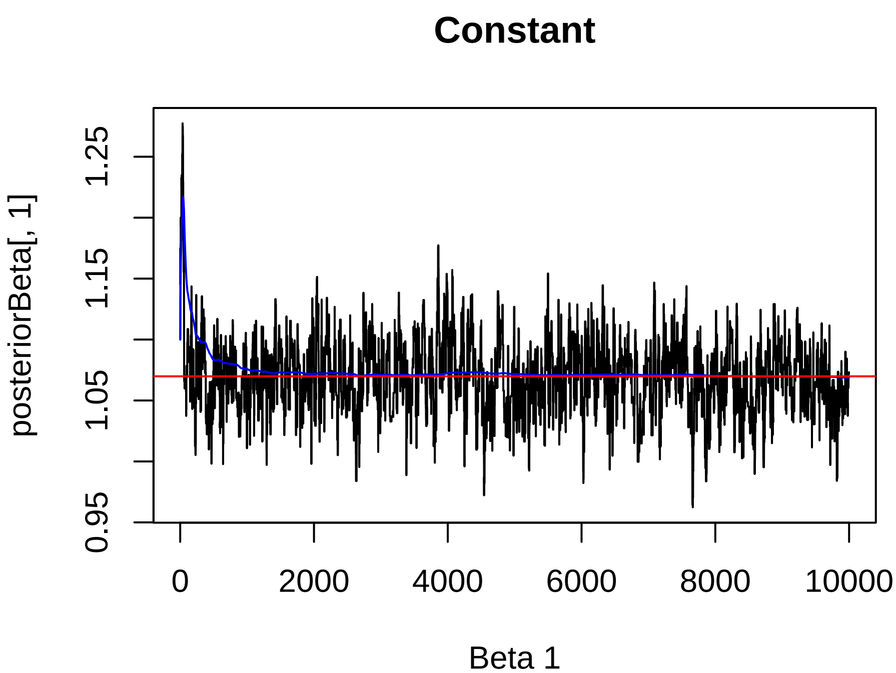
0.0000644	0.0001182	-0.0003192	-0.0001311	0.0000585	-0.0000898	-0.0002542	0.0008385	0.0010374
0.0011099	-0.0005686	-0.0004293	-0.0000576	-0.0000644	0.0002622	-0.0001063	0.0010374	0.0050548

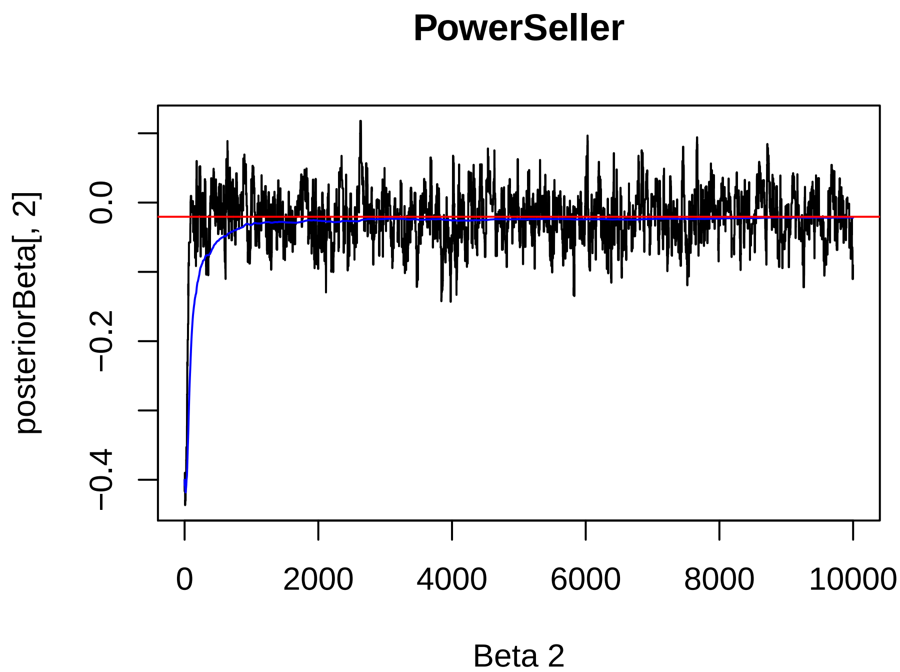
```
## The approximate posterior standard deviation is:
##  0.03074837 0.03678418 0.09227871 0.05057448 0.0602047 0.0914607 0.05634767 0.02895635 0.07109682
```

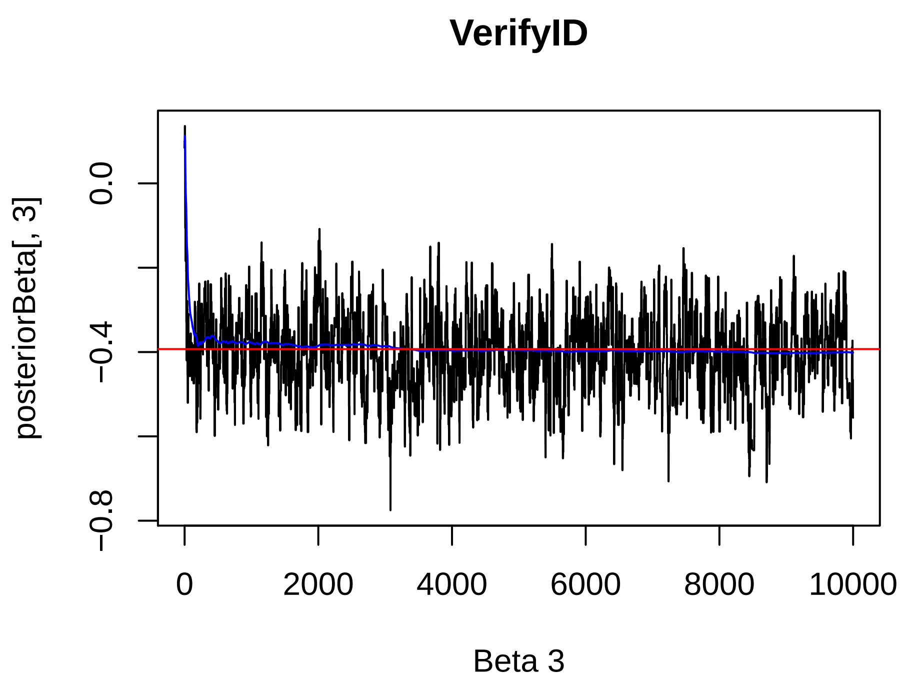
(c) Simulate from actual posterior of β using Metropolis Algorithm

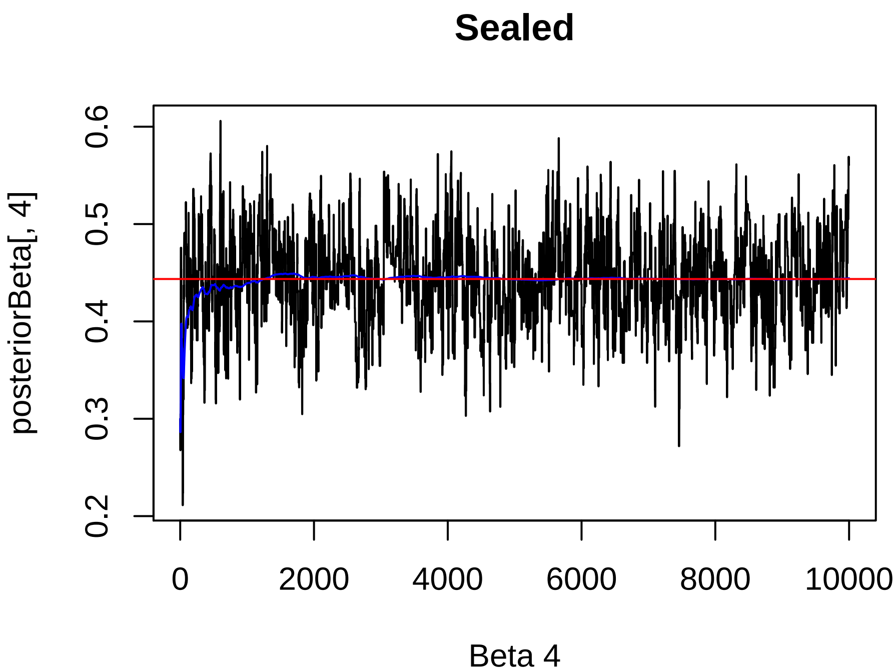
```
##
##  Acceptance Rate at c = 0.65 is : 0.2561
```

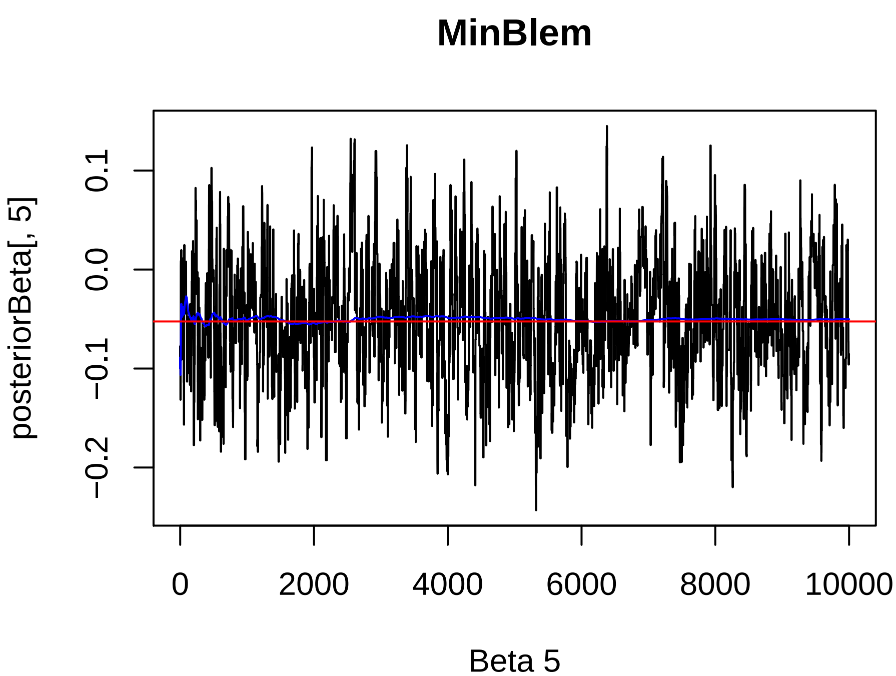
Since acceptance for c = 0.65 as tunning parameter is between 25% - 30% , taking it as tuning parameter.

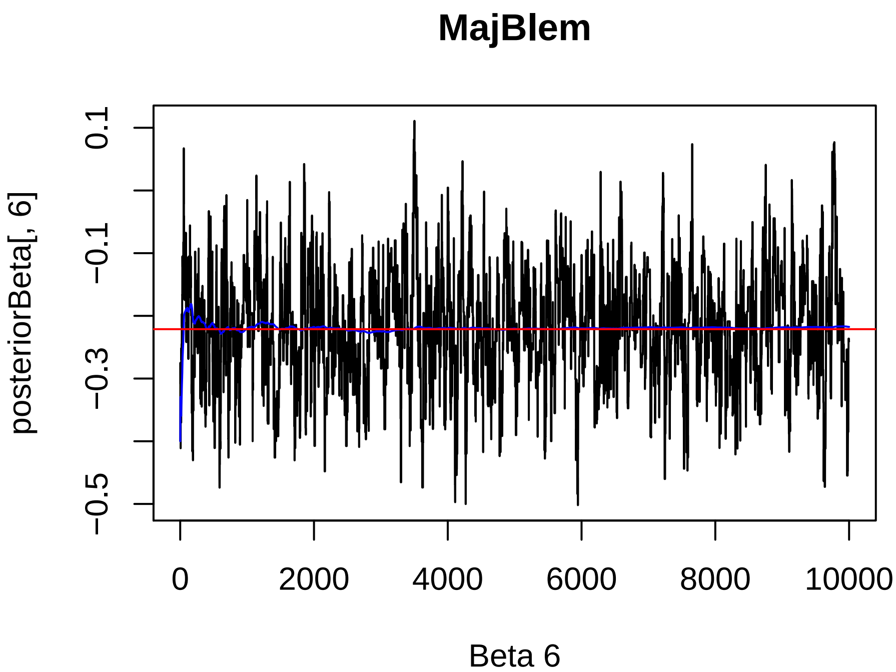


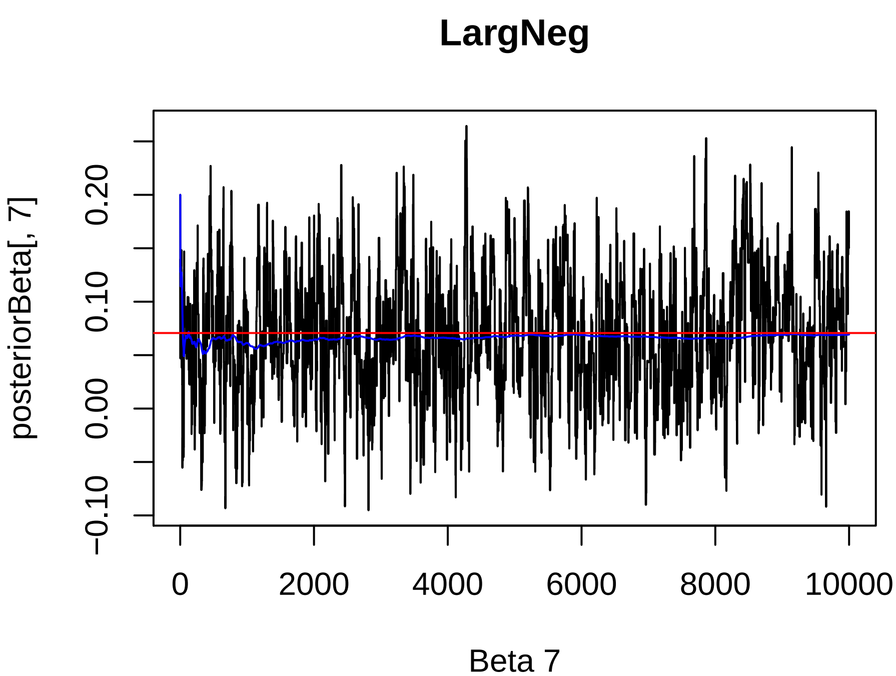

PowerSeller
posteriorBeta[, 2]
0.0
−0.2
−0.4
0
2000
4000
6000
8000
10000
Beta 2

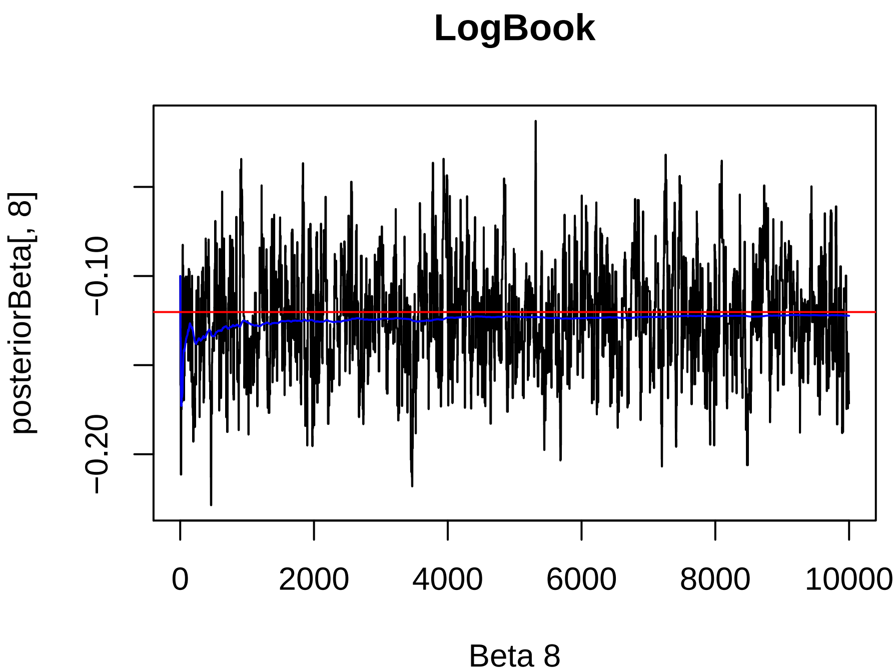

VerifyID
posteriorBeta[, 3]
0.0
−0.4
−0.8
0
2000
4000
6000
8000
10000
Beta 3

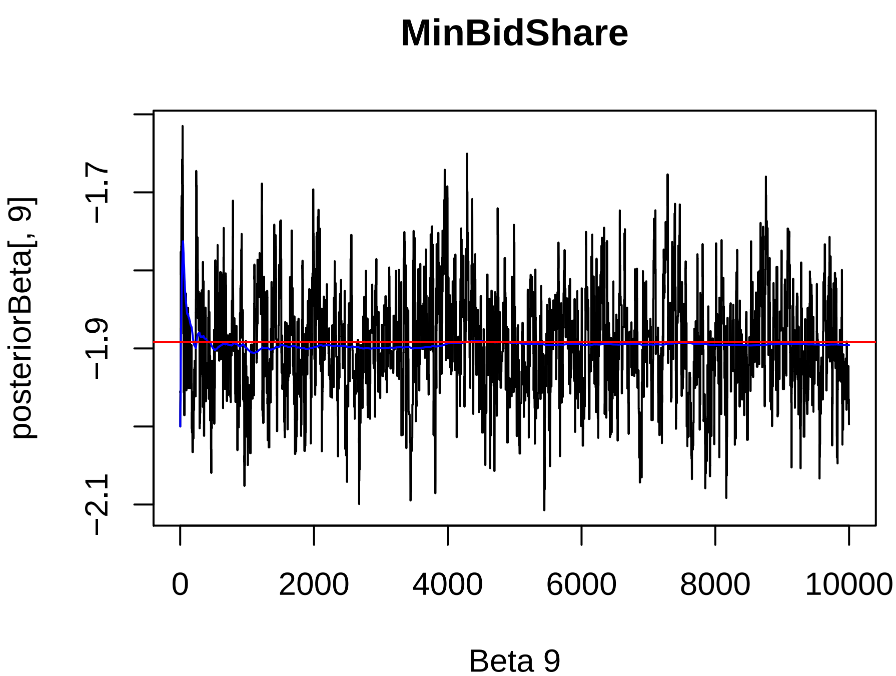

Sealed
posteriorBeta[, 4]
0.2
0.3
0.4
0.5
0.6
0
2000
4000
6000
8000
10000
Beta 4


MinBlem
posteriorBeta[, 5]
−0.2
−0.1
0.0
0.1
0
2000
4000
6000
8000
10000
Beta 5


MajBlem
posteriorBeta[, 6]
0.1
−0.1
−0.3
−0.5
0
2000
4000
6000
8000
10000
Beta 6


LargNeg
posteriorBeta[, 7]
0.20
0.10
0.00
−0.10
0
2000
4000
6000
8000
10000
Beta 7

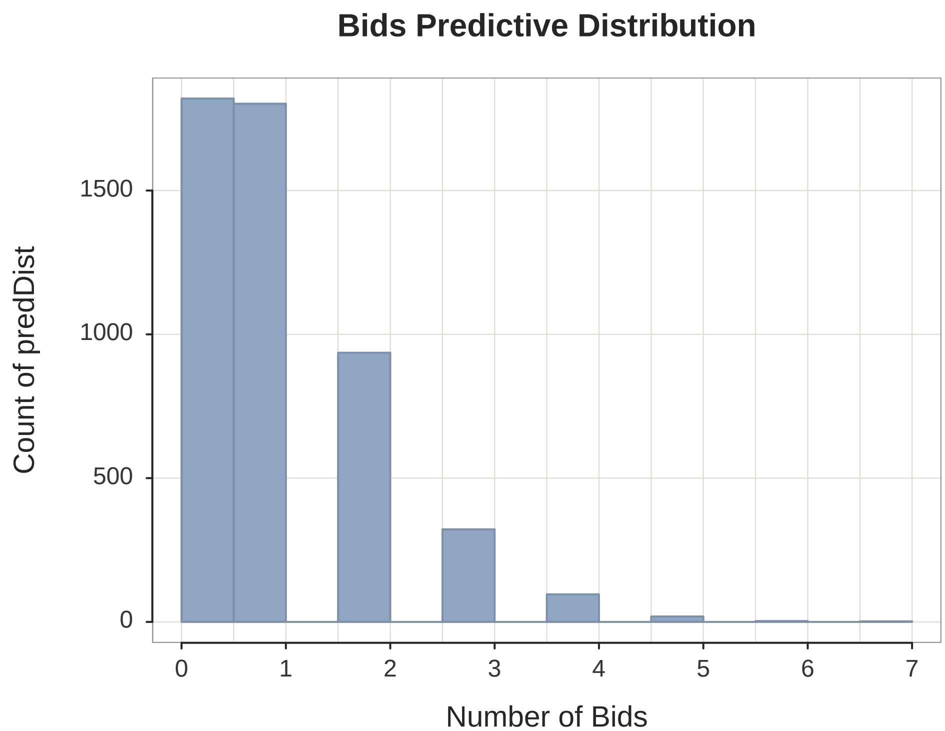




We can see that it is getting converged after 5000 Samples, so removing first 5000 samples as burn-in period.

(d) Simulate predictive distribution

```
## >>> Note: predDist is from the workspace, not in a data frame (table)
```




```
## >>> Suggestions
## bin_width: set the width of each bin
## bin_start: set the start of the first bin
## bin_end: set the end of the last bin
## Density(predDist)  # smoothed density curves plus histogram
## Plot(predDist)  # Violin/Box/Scatterplot (VBS) plot
##
##
## --- predDist ---
##
##        n   miss     mean       sd      min      mdn      max
##     5000      0     1.03     1.03     0.00     1.00     7.00
##
##
## (Box plot) Outliers: 5
##
## Small      Large
## -----      -----
##             7.0
##             7.0
##             6.0
##             6.0
##             6.0
##
##
## Bin Width: 0.5
## Number of Bins: 14
##
##        Bin  Midpnt  Count    Prop  Cumul.c  Cumul.p
```

```
## -----------------------------------------------------
##  0.0 > 0.5    0.25   1820    0.36     1820     0.36
##  0.5 > 1.0    0.75   1802    0.36     3622     0.72
##  1.0 > 1.5    1.25      0    0.00     3622     0.72
##  1.5 > 2.0    1.75    936    0.19     4558     0.91
##  2.0 > 2.5    2.25      0    0.00     4558     0.91
##  2.5 > 3.0    2.75    322    0.06     4880     0.98
##  3.0 > 3.5    3.25      0    0.00     4880     0.98
##  3.5 > 4.0    3.75     96    0.02     4976     1.00
##  4.0 > 4.5    4.25      0    0.00     4976     1.00
##  4.5 > 5.0    4.75     19    0.00     4995     1.00
##  5.0 > 5.5    5.25      0    0.00     4995     1.00
##  5.5 > 6.0    5.75      3    0.00     4998     1.00
##  6.0 > 6.5    6.25      0    0.00     4998     1.00
##  6.5 > 7.0    6.75      2    0.00     5000     1.00

##
##  Probability of no bids is:  0.364
```

```r
######################################################################
# 1. Normal model, mixture of normal model with semi-conjugate prior
######################################################################
library("geoR")

rainfall <- read.table(
  file="C:/Users/namit/Downloads/Bayesian Learning/R files/Lab3/rainfall.dat",
  header=FALSE)

#--------------------------------------------------------------------------
# (a) Normal model

## (i) Gibbs sampler
fullCondPostMu <- function(mu0, tausq0, sigsq, y) {
  n       <- length(y)
  ybar    <- mean(y)

  # Compute mu_n and tausq_n
  tausq_n <- sigsq*tausq0 / (n*tausq0 + sigsq)
  w       <- (n/sigsq) / (n/sigsq + 1/tausq0)
  mu_n    <- w*ybar + (1-w)*mu0

  # Sample from full conditional posterior of mu
  mu_post <- rnorm(1, mean=mu_n, sd=sqrt(tausq_n))

  return(mu_post)
}

fullCondPostSig <- function(nu0, sigsq0, mu_post, y) {
  n <- length(y)

  # Compute nu_n and sigsq_n
  nu_n       <- nu0 + n
  sigsq_n    <- (nu0*sigsq0 + sum((y-mu_post)**2)) / (n+nu0)

  # Sample from full conditional posterior of sigsq
```

```r
  sigsq_post <- geoR::rinvchisq(1, df=nu_n, scale=sigsq_n)

  return(sigsq_post)
}

gibbsSampler <- function(iter=100, mu0, tausq0, nu0, sigsq0, sigsq_init=1, data=rainfall$V1) {
  mu_post       <- numeric(iter)
  sigsq_post    <- numeric(iter)

  # First iteration of Gibbs sampler - Start with a random value for mu or sigsq
  sigsq_init    <- geoR::rinvchisq(1, df=nu0, scale=sigsq0)
  sigsq_post[1] <- sigsq_init
  mu_post[1]    <- fullCondPostMu(mu0=mu0, tausq0=tausq0, sigsq=sigsq_post[1], y=data)

  # Gibbs sampler for remaining iter-1 samples
  for (i in 2:iter) {
    mu_post[i]    <- fullCondPostMu(mu0=mu0, tausq0=tausq0, sigsq=sigsq_post[i-1], y=data)
    sigsq_post[i] <- fullCondPostSig(nu0=nu0, sigsq0=sigsq0, mu_post=mu_post[i], y=data)
  }

  return(list(mu=mu_post, sigsq=sigsq_post))
}

## (ii) Convergence of gibbs sampler

iter=1000
# mu0    :  Prior mean of Mu
# tausq0 :  Prior SD of Mu
# nu0    :  Degrees of freedom for prior of Sigsq
# sigsq0 :  Best guess of Sigsq
# sigsq  :  Initial value of Sigsq

# mu0=0, tausq0=1, nu0=1, sigsq0=1      : mu=26.5, sig2=1550
# mu0=10, tausq0=10, nu0=4, sigsq0=10   : mu=31.8, sig2=1550
# mu=1, tausq=100 nu0=1, sigsq=100      : mu=32.3, sig2=1550
gibbSample <- gibbsSampler(iter=iter, mu0=1, tausq0=100, nu0=1, sigsq0=100)

plot(1:iter, gibbSample$mu, type="l", col="blue",
     xlab="Iterations", ylab="Conditional Posterior Mu")
plot(1:iter, gibbSample$sigsq, type="l", col="blue",
     xlab="Iterations", ylab="Conditional Posterior Sigma Square")

#--------------------------------------------------------------------------
# (b) Mixture normal model

## Setting parameters
# Data parameters
x <- as.matrix(rainfall$V1)

# Model parameters
nComp <- 2

# Prior parameters
```

```
alpha  <- rep(10, nComp)     # Parameter for Dirichlet
mu0    <- rep(10, nComp)     # Prior mean of Mu
tau2_0 <- rep(10, nComp)     # Prior SD of Mu
nu0    <- rep(4, nComp)      # Degrees of freedom for prior of Sig2
sig2_0 <- rep(var(x), nComp) # Best guess of Sig2

# MCMC parameters
nIter <- 1000    # Number of gibbs sampling draws

# Plotting parameters
plotFit    <- TRUE               # Flag to set/unset plot display in each iteration
lcol       <- c("blue", "green") # Colours to plot
sleepTime  <- 0.1                # Time between iterations to plot graph

## Defining functions
# Function to simulate from Inv-Chisq distribution
rScaledInvChi2 <- function(n, df, scale){
  return((df*scale)/rchisq(n, df=df))
}

# Function to simulate from Dirichlet distribution
rDirichlet <- function(param){
  nCat    <- length(param)         # Number of categories
  piDraws <- matrix(NA, nCat, 1)   # Mixing coefficients of components in the model

  for (j in 1:nCat){
    piDraws[j] <- rgamma(1, param[j], 1)
  }
  piDraws = piDraws / sum(piDraws) # Dividing every column of piDraws by the sum of the elements in tha
  return(piDraws)
}

# Simple function that converts between two different representations of the mixture component allocati
S2alloc <- function(S){
  n     <- dim(S)[1]   # Number of data points
  alloc <- rep(0, n)   # Vector to hold the component number to which each data point belongs
  for (i in 1:n){
    alloc[i] <- which(S[i,] == 1) # The component number to which the data point is assigned
  }
  return(alloc)
}

## Initialize MCMC
nObs <- length(x)                                               # Number of observations
S    <- t(rmultinom(nObs, size=1 , prob=rep(1/nComp, nComp))) # nObs-by-nComp matrix with component all
probObsInComp <- rep(NA, nComp)                                 # Probability of a data point belonging t
mu     <- quantile(x, probs = seq(0, 1, length=nComp))
sig2   <- rep(var(x), nComp)

## Initialize plot
iterCount   <- 0
xGrid       <- seq(min(x)-1*sd(x), max(x)+1*sd(x), length=100) # x-values to plot the density
mixDensMean <- rep(0, length(xGrid))                            # Mean of mixture densities
```

```
xlim        <- c(min(xGrid), max(xGrid))
ylim        <- c(0, 2*max(hist(x, breaks=100)$density))

# Mixture component parameters
mu_collect   <- matrix(NA, nIter, nComp)
sig2_collect <- matrix(NA, nIter, nComp)
pi_collect   <- matrix(NA, nIter, nComp)

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

  # Update Pi's -components probabilities (Using full conditional posterior of pi)
  pi <- rDirichlet(alpha + nAlloc)

  # Collect the component probabilities in each iteration
  pi_collect[k, ] <- pi

  # Update mu's -components means (using full conditional posterior of mu)
  for (j in 1:nComp){
    tau2_n <- 1 / ((nAlloc[j]/sig2[j]) + (1/tau2_0[j])) # Posterior SD of Mu
    w      <- tau2_n * nAlloc[j]/sig2[j]
    mu_n   <- w*mean(x[alloc==j]) + (1-w)*mu0           # Posterior mean of Mu
    mu[j]  <- rnorm(1, mean=mu_n, sd=sqrt(tau2_n))      # Component means
  }

  # Collect the component means in each iteration
  mu_collect[k, ] <- mu

  # Update sigma2's -component variances (Using full conditional posterior of sigma)
  for (j in 1:nComp){
    nu_n    <- nu0[j] + nAlloc[j]
    sig2_n  <- (nu0[j]*sig2_0[j] + sum((x[alloc==j]-mu[j])^2)) / (nu0[j]+nAlloc[j])
    sig2[j] <- rScaledInvChi2(1, df=nu_n, scale=sig2_n) # Components variances
  }

  # Collect the component variances in each iteration
  sig2_collect[k, ] <- sig2

  # Update allocation using new component means and variances
  for (i in 1:nObs){
    for (j in 1:nComp){
      probObsInComp[j] <- pi[j]*dnorm(x[i], mean=mu[j], sd=sqrt(sig2[j]))
    }
    S[i, ] <- t(rmultinom(n=1, size=1 , prob=probObsInComp/sum(probObsInComp)))
  }

  # Printing the fitted density against data histogram
  if (plotFit && k%%1==0){
    iterCount <- iterCount + 1
```

```r
    #hist(x, breaks=20, freq=FALSE, xlim=xlim, 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(sig2[j]))    # Component density
      mixDens  <- mixDens + pi[j]*compDens                 # Mixture density
      #lines(xGrid, compDens, type="l", lwd=2, col=lcol[j])
      #components[j] <- paste("Component ", j)
    }
    mixDensMean <- ((iterCount-1)*mixDensMean + mixDens)/iterCount # Mean mixture density

    #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", lcol[1:nComp], 'red'), lw=2)
    #Sys.sleep(sleepTime)
  }
}

# Plots of posterior trajectories and means to evaluate convergence
plot(mu_collect[, 1], type="l", ylab="Mu1", main="Mean of comp1", col="red")
plot(mu_collect[, 2], type="l", ylab="Mu1", main="Mean of comp2", col="blue")
plot(sqrt(sig2_collect[, 1]), type="l", ylab="Sigma1", main="SD of comp1", col="red")
plot(sqrt(sig2_collect[, 2]), type="l", ylab="Sigma2", main="SD of comp2", col="blue")
plot(pi_collect[, 1], type="l", ylab="Pi1", main="Probability of comp1", col="red")
plot(pi_collect[, 2], type="l", ylab="Pi2", main="Probability of comp2", col="blue")

#---------------------------------------------------------------------------
# (c) Graphical comparison

# Kernel density estimate of the data
kernelDensData = density(rainfall$V1)

# Mean of Gibbs full conditional posterior of Mu and Sigma2
meanPostMu   = mean(gibbSample$mu)
meanPostsig2 = mean(gibbSample$sigsq)

# Mean of posterior draws of mixture component parameters
meanPostMuMix   <- apply(mu_collect, 2, mean)
meanPostSig2Mix <- apply(sig2_collect, 2, mean)
meanPostPiMix   <- apply(pi_collect, 2, mean)

# Mean mixed density
meanMixDens <- rep(0, length(xGrid))
for (j in 1:nComp){
  compDens <- dnorm(xGrid, meanPostMuMix[j], sd=sqrt(meanPostSig2Mix[j])) # Component density
  meanMixDens <- meanMixDens + meanPostPiMix[j]*compDens                  # Mixture density
}

# Graphical comparison with data histogram
hist(x, breaks=20, freq=FALSE, xlim=xlim, main="Graphical comparison")
lines(xGrid, dnorm(xGrid, mean=mean(meanPostMu), sd=sqrt(meanPostsig2)), type="l", lwd=2, col="blue")
lines(xGrid, meanMixDens, type="l", lwd=2, lty=4, col="red") # (Same as mixDensMean)
```

```r
legend("topright", box.lty=1, legend=c("Data histogram","Mixture density","Normal density"), col=c("bla

# Graphical comparison with data kernel
plot(kernelDensData$x, kernelDensData$y, type="l", lwd=2, lty=4, col="black",
     main="Graphical comparison", xlab="Density", ylab="x")
lines(xGrid, dnorm(xGrid, mean=meanPostMu, sd=sqrt(meanPostsig2)), type="l", lwd=2, col="blue")
lines(xGrid, meanMixDens, type="l", lwd=2, col="red")
legend("topright", box.lty=1, legend=c("Data Kernel Density","Normal density", "Mixture density"), col=c

######################################################################
# 2. Metropolis Random Walk for Poisson regression
######################################################################
# (a) MLE estimator of beta
#library(glmnet)
eBayData = read.delim("C:/Users/namit/Downloads/Bayesian Learning/R files/Lab3/eBayNumberOfBidderData.da
                      header=TRUE, sep="")

glmModel = glm(formula = nBids ~ . - Const , data = eBayData , family = "poisson")
summary(glmModel)

#-------------------------------------------------------------------------
# (b) Bayesian analysis of the Poisson regression

# Log likelihood Estimation:
library(mvtnorm)

# Function that returns log posterior of beta
llk = function(beta, X , Y , mu, sigma){
  ncovariates = length(beta)
  x = X %*% beta

  logLikli = sum(Y * x - exp(x))
  prior = dmvnorm(beta, mu, sigma, log=TRUE)
  post = logLikli + prior

  return(post)
}

# Predictors and response variables
Y = eBayData[,1]
X = as.matrix(eBayData[,-1])

# Covariates
ncovariates = ncol(X)
covNames = names(eBayData)[-1]

# Set up prior parameters
mu_0 =as.vector(rep(0, ncovariates))
Sigma_0 = 100 * solve(t(X)%*% X)

# Find the optimum beta that maximizes the log posterior of beta
beta_init = as.vector(rep(0, ncovariates))
```

```r
optimBeta = optim(par = beta_init , fn = llk ,
                  X = X , Y = Y , mu = mu_0, sigma = Sigma_0 ,
                  method=c("BFGS"), control=list(fnscale=-1), hessian=TRUE)

postMode = optimBeta$par # Posterior mode=Optimum beta that maximizes the log posterior
postCov = -solve(optimBeta$hessian) # Posterior covariance matrix is -inv(Hessian)
PostStd = sqrt(diag(postCov)) # Computing approximate standard deviations.

cat("The posterior mode is: " , "\n" , postMode , "\n")
cat("\n")

cat("The posterior variance-covariance matrix is: " , "\n")
knitr::kable(postCov)
cat("\n")

#optimBeta
cat("The approximate posterior standard deviation is: " , "\n" , PostStd , "\n")
cat("\n")

#-------------------------------------------------------------------------
# (c) Simulate from actual posterior of beta using Metropolis Algorithm
fnMetropolish = function(nSample , theta , fnPoste , c , ...)
  {
    #Intialize
    theta_current = theta
    Sigma_current = c * postCov
    nAccepted = 0
    ntheta = matrix(nrow=  nSample , ncol = length(theta))
    ntheta[1,] = theta_current

    #j = 1
    for(i in 2:nSample)
    {
      #proposal
      thetaProp = as.vector(rmvnorm(1 , mean = theta_current , sigma =Sigma_current))

      #Posterior in log order
      poste_current = fnPoste(theta_current , ...)
      poste_propsal = fnPoste(thetaProp , ...)

      #acceptance propbability
      alpha = min(1 ,  exp(poste_propsal - poste_current))

      #check proposal acceptance
      u = runif(1 , 0 , 1)
      if(u < alpha){
        theta_current = thetaProp
        nAccepted = nAccepted + 1
      }

      #update theta matrix
      ntheta[i,] = theta_current
```

```r
      #j = j + 1
    }

    #Acceptance rate AR
    AR = 0
    if(nAccepted > 0) AR =  nAccepted / (nSample)

    return(list("AcceptanceRate" = AR , "Theta" = ntheta))
  }

#intialize Simulation parameter
c = 0.65
nSample = 10000
#initBurn = 500
theta_init = c(1.1, -0.4, 0.1, 0.3, -0.1, -0.4, 0.2, -0.1, -2)
#Generate Sample from above function
metroSample_0.65 = fnMetropolish(nSample = nSample,
                                 theta = theta_init ,
                                 fnPoste = llk, c = 0.65  ,
                                 X = X ,
                                 Y = Y , mu = mu_0, sigma = Sigma_0)
acceptRate_0.65 = metroSample_0.65$AcceptanceRate
cat("\n" , "Acceptance Rate at c = 0.65 is :" , acceptRate_0.65 , "\n" )


# MC beta
posteriorBeta = metroSample_0.65$Theta

#For MCMC Convergence
beta1CumMean = cumsum(posteriorBeta[,1])/seq(1 , 10000 , 1)
beta2CumMean = cumsum(posteriorBeta[,2])/seq(1 , 10000 , 1)
beta3CumMean = cumsum(posteriorBeta[,3])/seq(1 , 10000 , 1)
beta4CumMean = cumsum(posteriorBeta[,4])/seq(1 , 10000 , 1)
beta5CumMean = cumsum(posteriorBeta[,5])/seq(1 , 10000 , 1)
beta6CumMean = cumsum(posteriorBeta[,6])/seq(1 , 10000 , 1)
beta7CumMean = cumsum(posteriorBeta[,7])/seq(1 , 10000 , 1)
beta8CumMean = cumsum(posteriorBeta[,8])/seq(1 , 10000 , 1)
beta9CumMean = cumsum(posteriorBeta[,9])/seq(1 , 10000 , 1)


plot(posteriorBeta[,1] , xlab = "Beta 1" , main = "Constant" , type = "l")
points(beta1CumMean , type = "l" , col = "blue")
abline(h = postMode[1] , col = "red")


plot(posteriorBeta[,2] , xlab = "Beta 2" , main = "PowerSeller" , type = "l")
points(beta2CumMean , type = "l" , col = "blue")
abline(h = postMode[2] , col = "red")


plot(posteriorBeta[,3] , xlab = "Beta 3" , main = "VerifyID" , type = "l")
points(beta3CumMean , type = "l" , col = "blue")
abline(h = postMode[3] , col = "red")
```

```r
plot(posteriorBeta[,4] , xlab = "Beta 4" , main = "Sealed" , type = "l")
points(beta4CumMean , type = "l" , col = "blue")
abline(h = postMode[4] , col = "red")


plot(posteriorBeta[,5] , xlab = "Beta 5" , main = "MinBlem" , type = "l")
points(beta5CumMean , type = "l" , col = "blue")
abline(h = postMode[5] , col = "red")


plot(posteriorBeta[,6] , xlab = "Beta 6" , main = "MajBlem" , type = "l")
points(beta6CumMean , type = "l" , col = "blue")
abline(h = postMode[6] , col = "red")


plot(posteriorBeta[,7] , xlab = "Beta 7" , main = "LargNeg" , type = "l")
points(beta7CumMean , type = "l" , col = "blue")
abline(h = postMode[7] , col = "red")

plot(posteriorBeta[,8] , xlab = "Beta 8" , main = "LogBook" , type = "l")
points(beta8CumMean , type = "l" , col = "blue")
abline(h = postMode[8] , col = "red")

plot(posteriorBeta[,9] , xlab = "Beta 9" , main = "MinBidShare" , type = "l")
points(beta9CumMean , type = "l" , col = "blue")
abline(h = postMode[9] , col = "red")

#------------------------------------------------------------------------
# (d)  Simulate predictive distribution
Xpred = c(1,1,1,1,0,0,0,1,0.5)
posteriorBeta = posteriorBeta[5001:10000 , ]

n = NROW(posteriorBeta)

predDist = numeric(length = n)
for(i in 1:n){
  predDist[i] = rpois(1 , lambda = exp(Xpred %*% posteriorBeta[i,]))
}

library(lessR)
hs(predDist , xlab = "Number of Bids" , main = "Bids Predictive Distribution")

#probability of no bidders
probNoBids = length(which(predDist == 0)) / n
cat("\n" , "Probability of no bids is: " , probNoBids)
```