Computer Lab 3

phiho267 & zijfe244

3-5-2019

Contents

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 $\frac{1}{100}$ inch, and records of zero precipitation are excluded) at Snoqualmie Falls, Washington. Analyze the data using the following two models.

(a) Normal model.

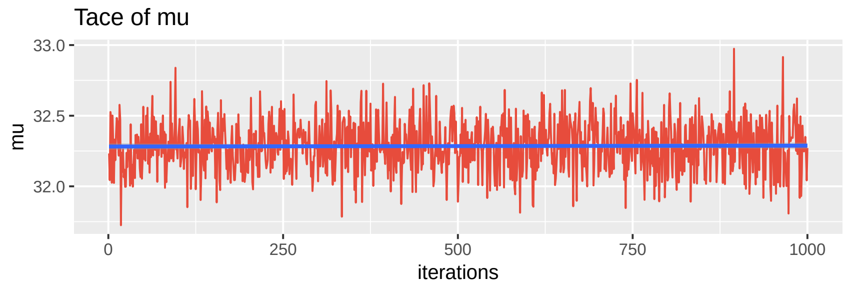
Assume the daily precipitation $\{y_1, ..., y_n\}$ are independent normally distributed, $y_1, ..., y_n \mid \mu, \sigma^2 \sim \mathcal{N}(\mu, \sigma^2)$ where both μ and σ^2 are unknown. Let $\mu \sim \mathcal{N}(\mu_0, \tau_0^2)$ independently of $\sigma^2 \sim (v_0, \sigma_0^2)$.

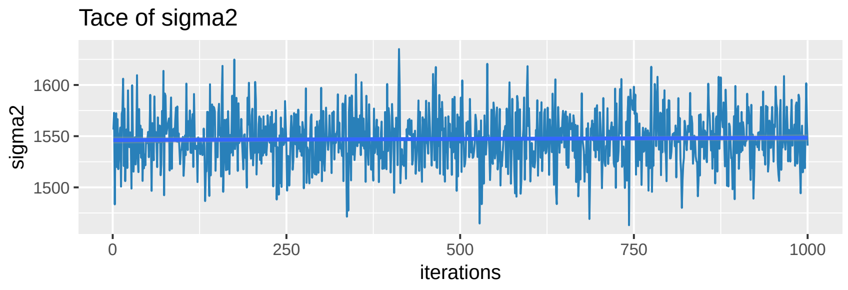
- i. Implement (code!) a Gibbs sampler that simulates from the joint posterior $p(\mu, \sigma^2 \mid y_1, ..., y_n)$ The full conditional posteriors are given on the slides from Lecture 7.

```r
# implement Gibbs sampling
# set seed
set.seed(123456)

# Gibbs sampling for normal model with non-conjugate prior - L7S15

# prior parameter for 'sigma2'
v_0 <- 1 # how much we trust the prior's var's mean is
sigma2_0 <- var(rainfall$precipitation) # the variacne of the data
# sigma2
sigma2 <- sigma2_0 # need this value for the first simulation

# prior parameter for 'mu'
tau2_0 <- 1 # how much we trust the prior's mu's var is
#tau2_0 <- sigma2_0
########
#Zijife: it's ok to define, since it will change automatically later
```

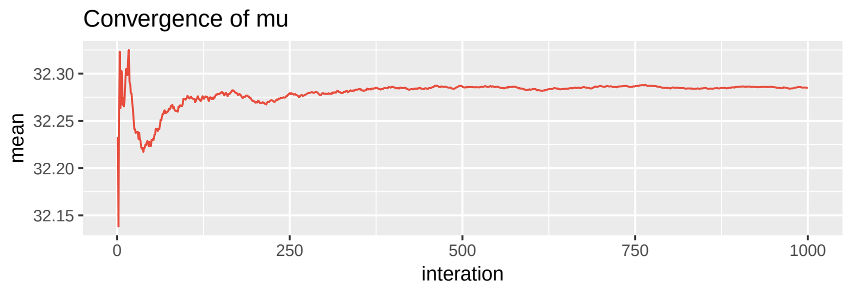
```r
########
mu_0 <- mean(rainfall$precipitation)
# mu
mu <- rnorm(n = 1, mean = mu_0, tau2_0) # need this value for the first simulation

# the sampling
nDraws <- 1000
Gibbs_sampling <- data.frame("mu" = numeric(), "sigma2" = numeric())
Gibbs_sampling2 <- data.frame("mu" = numeric(), "sigma2" = numeric())

# for exercise 1 a) ii
# create vairables for mean and varaicne
mean_vec <- vector(mode = "numeric", length = nDraws)
variance_vec <- vector(mode = "numeric", length = nDraws)

for (i in 1:nDraws) {
  # full conditional posterior
  # generate mu - need mu_n (need w to generate mu_n) & tau2_n
  # mu_n - L2S3 - Normal data, known variance - normal prior

  #-- generate mu -------------------------------------------------------- start
  # generate w
  w <- (n/sigma2)/((n/sigma2) + (1/tau2_0))
  # generate mu_n
  mu_n <- w * mean(rainfall$precipitation) + (1 - w) * mu_0

  # genearte tau2_n
  tau2_n <- 1 / (n/sigma2 + 1/tau2_0)

  #-- generate mu
  mu <- rnorm(n = 1, mean = mu_n, sd = tau2_n)
  #-- generate mu ---------------------------------------------------------- end

  old_sigma2 <- sigma2
  # generate sigma2 ------------------------------------------------------ start
  # generate v_n - L5s3 - Normal model with normal prior
  v_n <- v_0 + n

  # generate second term - to sample sigma2
  second_term <- (v_0 * sigma2_0 + sum((rainfall$precipitation - mu)^2))/ (n + v_0)

  #-- generate sigma2
  sigma2 <- rinvchisq(n = 1, df = v_n, scale = second_term)
  # generate sigma2 -------------------------------------------------------- end

  # save mu & sigma in df
  Gibbs_sampling2 <- rbind(Gibbs_sampling2, data.frame(mu, sigma2))
  if(i==1){
    sampling <- data.frame("mu" = mu, "sigma2" = sigma2)
  }else{
    sampling <- data.frame("mu" = c(mu, mu), "sigma2" = c(old_sigma2, sigma2))
  }
  # variables we want to save
  Gibbs_sampling <- rbind(Gibbs_sampling, sampling)
  #save the mean of mu and variance
  mean_vec[i] <- mean(Gibbs_sampling2$mu)
  variance_vec[i] <- var(Gibbs_sampling2$sigma2)

}
Gibbs_sampling2$iterations <- 1:nDraws
```

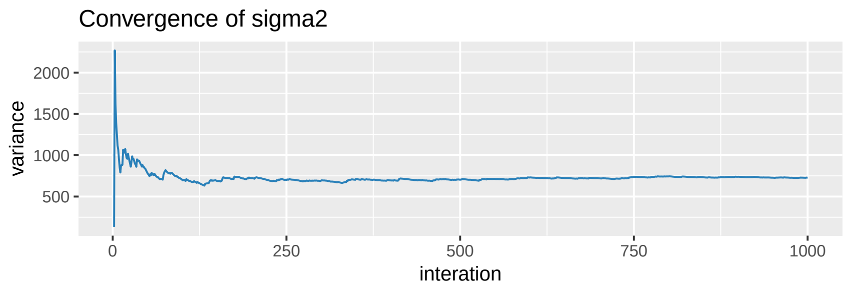
- ii. Analyze the daily precipitation using your Gibbs sampler in (a)-i. Evaluate the convergence of the Gibbs sampler by suitable graphical methods, for example by plotting the trajectories of the sampled Markov chains.



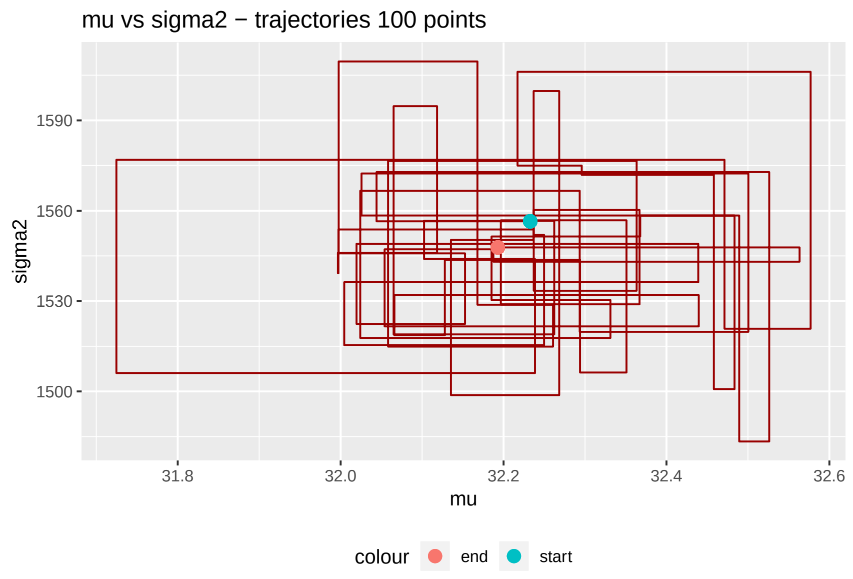


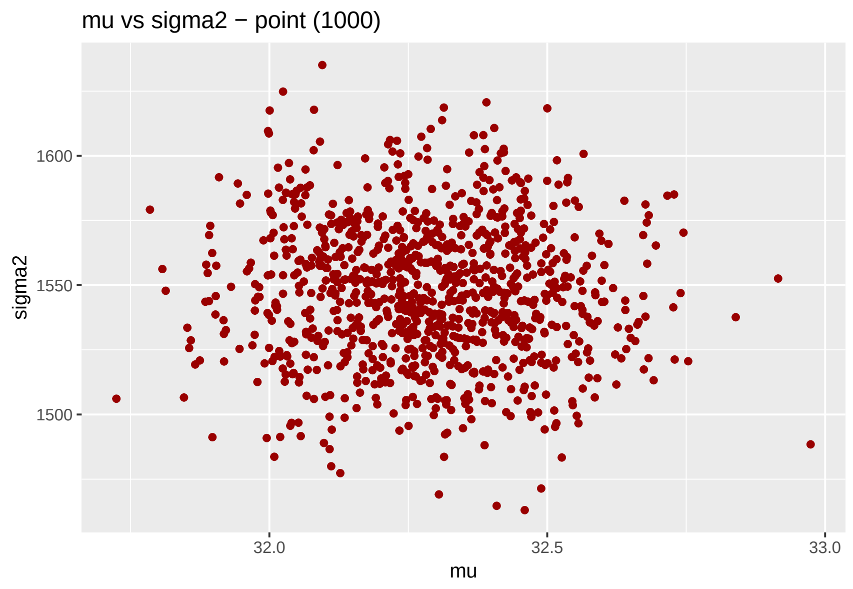
These two plots show the samplings of μ and σ^2. Based on the `geom_smooth(method="gam")`, it is obvious that we simulate thse two parameters well.





We have calculated the mean and the variance of the given iterations. This can be seen at the end of the code of the task 1a)i of the Gibbs sampler. The first picture confirms the assumption that mu converges after only a few iterations. The same is true for sigma, but here about 125 iterations are used. Anyway, our sampling simulations converge well.





(b) Mixture normal model.

Let us now instead assume that the daily precipitation $\{y_1, ..., y_n\}$ follow an iid two-component **mixture of normals** model:

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

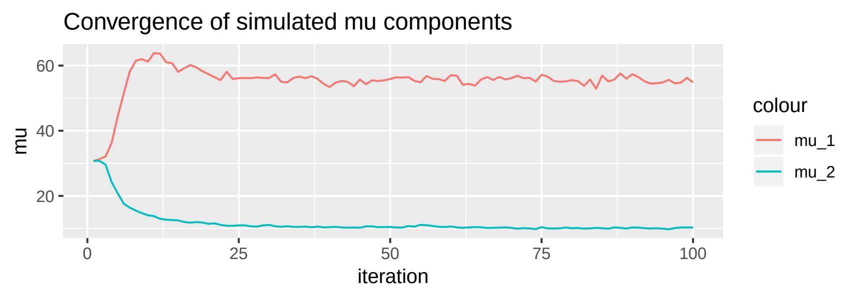
where

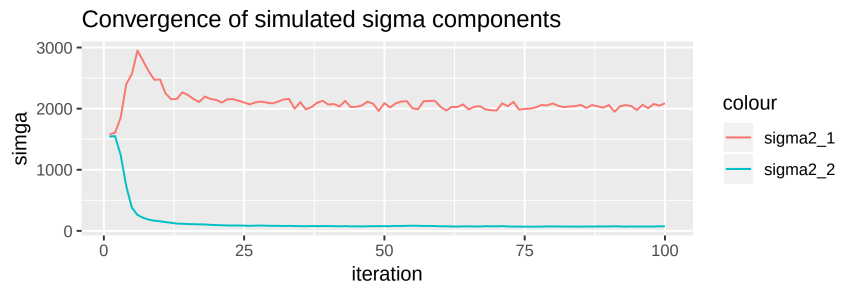
$$\mu = (\mu_1, \mu_2) \; and \; \sigma^2 = (\sigma_1^2, \sigma_2^2)$$

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.

The following code from lecture 7, this one has been adapted and modified for the following tasks.

```r
# Set the prior hyperparameters suitably
# Evaluate the convergence of the sampler -> need to save mu's and sigma's

##########    BEGIN USER INPUT #################
# Data options
rawData <- rainfall
x <- as.matrix(rainfall$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", "magenta")
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)
  }
  # Diving every column of piDraws by the sum of the elements in that column.
  piDraws = piDraws/sum(piDraws)
  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)
# nObs-by-nComp matrix with component allocations.
S <- t(rmultinom(nObs, size = 1 , prob = rep(1/nComp,nComp)))
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))

# -------------------- Update - start -------------------
# we want to save mu's and sigma's for every interation
# we have two misxtured model - so two coloums & nIter (100) as rows
simulated_mu <- matrix(data = NA, nrow = nIter, ncol = 2)
simulated_sigma <- matrix(data = NA, nrow = nIter, ncol = 2)
# -------------------- Update - end -------------------

for (k in 1:nIter){
  #message(paste('Iteration number:',k))
  # Just a function that converts between different
  # representations of the group     allocations
  alloc <- S2alloc(S)
  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 - start -------------------
    simulated_mu[k,] <- mu
  # -------------------- Update - end ---------------------
  # 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 - start -------------------
    simulated_sigma[k,] <- sigma2
  # -------------------- Update - end ---------------------



  # 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("topleft", box.lty = 1, legend = c("Data histogram",components, 'Mixture'),
    # col = c("black",lineColors[1:nComp], 'red'), lwd = 2)
    # Sys.sleep(sleepTime)
  }

}

#########################    Helper functions    #############################
```



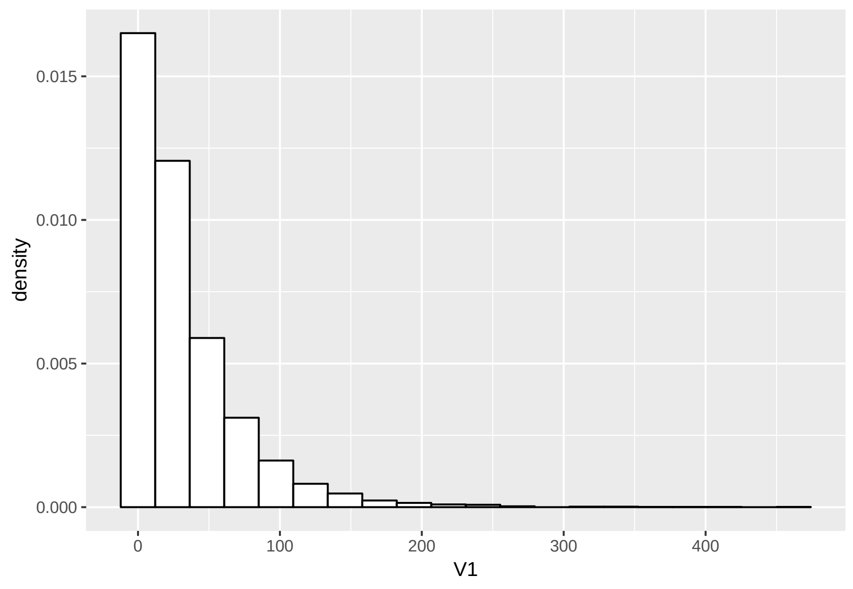


With the growth of iterations, μ's and σ^2's we assume converge respectively.

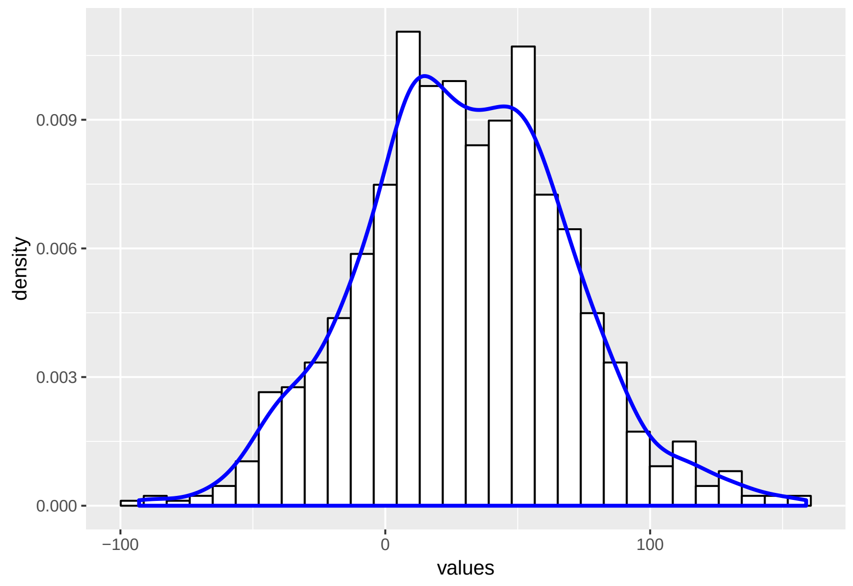
(c) Graphical comparison.

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

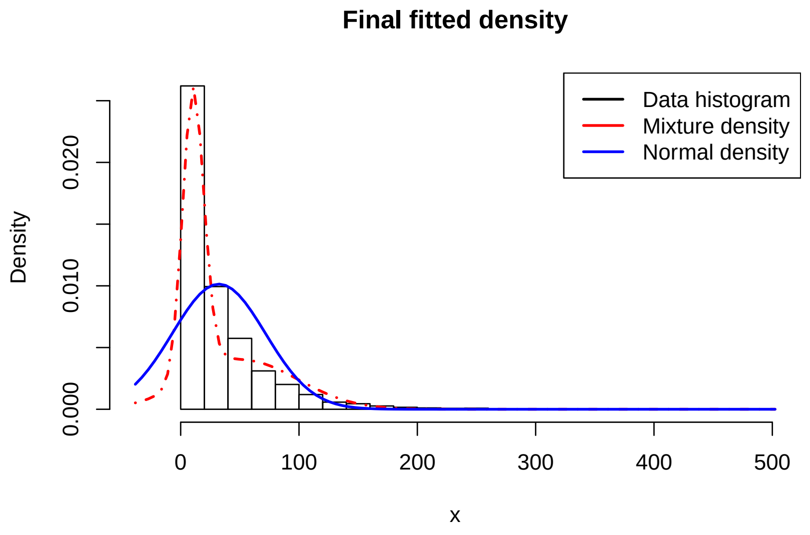
1) a histogram or kernel density estimate of the data.



2) Normal density $\mathcal{N}(\hat{\mu}, \hat{\sigma}^2)$ in (a).



3) Mixture of normals density $p(y_i \mid \hat{\mu}, \hat{\sigma}^2, \hat{\pi})$ in (b).



2. Metropolis Random Walk for Poisson regression.

Consider the following Poisson regression model

$$y_i \mid \beta \sim \ Poisson\ \left[\ exp(\mathbf{x}_i^T \beta)\right],\ i = 1, ..., n,$$

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 ($\mathbf{x}$):

- **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. Stan- dardized)
- **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?

```
rm(list = ls())
data <- read.table("eBayNumberOfBidderData.dat", head = TRUE)
mdl.2a <- glm(nBids ~ ., data = data[-2], family = poisson)
summary(mdl.2a)
```

```
##
## Call:
## glm(formula = nBids ~ ., family = poisson, data = data[-2])
##
## 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
```

Based on the estimate from `glm.R`, the covariate *MinBidShare* has the largest absolute value among all the covariates, which means it is the most significant.

(b) Bayesian analysis of the Poisson regression

Let's now do a Bayesian analysis of the Poisson regression. Let the prior be $\beta \sim \mathcal{N}\left[\mathbf{0}, 100 \cdot (\mathbf{X}^T\mathbf{X})^{-1}\right]$ where $\mathbf{X}$ is the $n \times p$ 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:

$$\beta \mid y \sim \mathcal{N}\big(\tilde{\beta}, J_{\mathbf{y}}^{-1}(\tilde{\beta})\big),$$

where $\tilde{\beta}$ is the posterior mode and $J_{\mathbf{y}}(\tilde{\beta})$ is the negative Hessian at the posterior mode. $\tilde{\beta}$ and $J_{\mathbf{y}}(\tilde{\beta})$ 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.).

```
y <- data[,1]
X <- as.matrix(data[,2:10])

# rename X columns
covNames <- names(data)[2:10]
m <- dim(X)[2]

# setting up the prior
mu <- as.vector(rep(0,m))
Sigma <- 100 * solve(t(X)%*%X)

LogPostLogistic <- function(beta, y, X, mu, Sigma){

  M <- length(beta)

  # log the likelihood of Possion Distribution
  logLik <- -sum(exp(X%*%beta)) + sum(y%*%(X%*%beta))
  if (abs(logLik) == Inf){
    logLik <- -20000
    # Likelihood is not finite, stear the optimizer away from here!
    # by Teacher's idea
  }

  # prior follows multi-normal distribution with
  logPrior <- dmvnorm(beta, mu, Sigma, log = TRUE)

  # cuz we logarithmize the likelihood and prior,
  # posterior is the sum of them
  return(logLik + logPrior)
}

initVal <- as.vector(rnorm(dim(X)[2]))

OptimResults <- optim(initVal,
                   LogPostLogistic,
                   y = y,
                   X = X,
                   mu = mu,
                   Sigma = Sigma,
                   method = c("BFGS"),
                   control = list(fnscale=-1),
                   hessian = TRUE)  # output hessian matrix
```

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

##       Const PowerSeller    VerifyID      Sealed     Minblem     MajBlem
##  1.06984636 -0.02051537 -0.39303765  0.44355077 -0.05246085 -0.22125582
##     LargNeg     LogBook MinBidShare
##  0.07070030 -0.12020921 -1.89196643
```

```
## [1] "The approximate posterior standard deviation is:"
```

```
##       Const PowerSeller    VerifyID      Sealed     Minblem     MajBlem
##  0.03074832  0.03678420  0.09227990  0.05057459  0.06020452  0.09146140
##     LargNeg     LogBook MinBidShare
##  0.05634747  0.02895629  0.07109669
```

(c) Simulate from the actual posterior

Now, let's simulate from the actual posterior of β using the Metropolis algo- rithm 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):

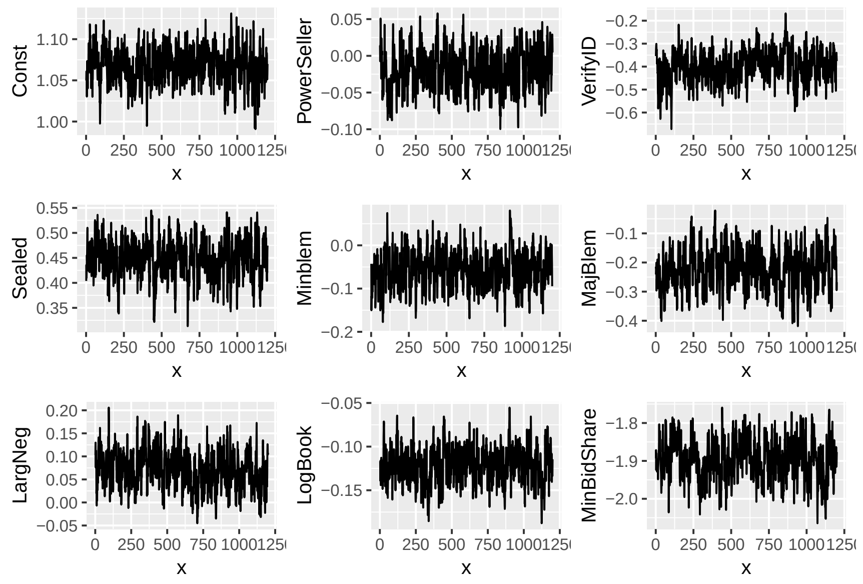
$$\theta_p \mid \theta^{i-1} \sim \mathcal{N}\big(\theta^{i-1}, c \cdot \sum\big),$$

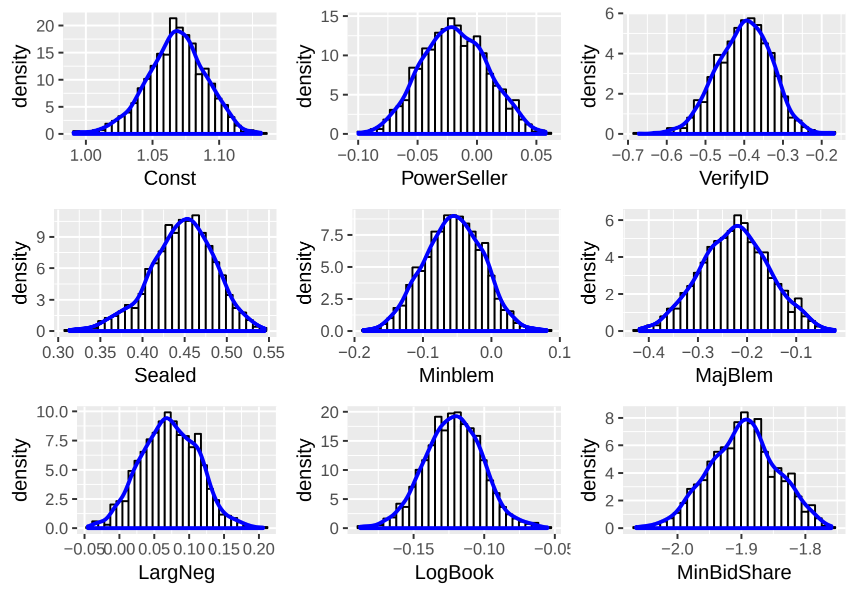
where $\sum = J_{\mathbf{y}}^{-1}(\tilde{\beta})$ obtained in b). The value c is a tuning parameter and y should be an input to your Metropolis function. The user of your Metropolis function should be able to supply her own posterior density function, not necessarily for the Poisson regression, and still 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 $(\ldots)$ 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.

```r
set.seed(12345)
RWMSampler <- function(LogPostFunc, num, c,...){

  # In random walk, all draws are from sample proposal
  draws <- data.frame(matrix(nrow = num, ncol = 9))
  colnames(draws) <- colnames(X)
  draws[1,] <- mvrnorm(n = 1, postMode, c*postCov)
  count <- 1
  i <- 0
  while(count<num) {
    i <- i+1
    theta.old <- as.numeric(draws[count,])
    theta.new <- mvrnorm(n = 1, theta.old, c*postCov)
    u <- runif(1,0,1)
    # LogPostFunc is the distribution of theta/posterior we assume
    left  <- LogPostFunc(theta.new,...)
    right <- LogPostFunc(theta.old,...)
    a <- min(1, exp(left-right))
    if(u<a){
      count <- count + 1
      draws[count,] <- theta.new
    }
  }
  cat("acceptance probability: ",num/i,"\n")
  return(draws)
}
```

```r
c <- 0.35
num <- 1200
RW <- RWMSampler(LogPostFunc = LogPostLogistic,
            num = num,
            c = c,
            y = y, X = X, mu = postMode, Sigma = postCov)
```

```
## acceptance probability:  0.2502607
```





step	Const	PowerSeller	VerifyID	Sealed	Minblem	MajBlem	LargNeg	LogBook	MinBidShare
Q2a	1.072	-0.021	-0.395	0.444	-0.052	-0.221	0.071	-0.121	-1.894
Q2b	1.07	-0.021	-0.393	0.444	-0.052	-0.221	0.071	-0.12	-1.892
Q2c	1.068	-0.019	-0.401	0.447	-0.056	-0.224	0.072	-0.122	-1.897

Based on the table, the results from all the steps are close to each other.

Lecture code

```r
# the given R file - NormalMixtureGibbs.R

# Estimating a simple mixture of normals
# Author: Mattias Villani, IDA, Linkoping University. http://mattiasvillani.com

##########    BEGIN USER INPUT #################
# Data options
data(faithful)
rawData <- faithful
x <- as.matrix(rawData['eruptions'])

# Model options
nComp <- 4    # 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 <- 1000 # 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)
  }
  # Diving every column of piDraws by the sum of the elements in that column.
  piDraws = piDraws/sum(piDraws)
  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 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){
  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 =
  }

  # 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),
    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("topleft", box.lty = 1, legend = c("Data histogram",components, 'Mixture'),
           col = c("black",lineColors[1:nComp], 'red'), lwd = 2)
    Sys.sleep(sleepTime)
  }

}

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(

#########################    Helper functions    ##############################################
```

Appendix

```
knitr::opts_chunk$set(echo = TRUE, message=FALSE, warning=FALSE, out.width = "320px")
# library used
library(ggplot2)
library(MASS) # To access the mvrnorm() function
library(LaplacesDemon) # for rinvchisq
library(gridExtra) # put plot together
library(mvtnorm)
library(kableExtra)
# read data
rainfall <- read.delim("rainfall.dat", header = FALSE)
colnames(rainfall) <- c("precipitation")
n <- nrow(rainfall)
# implement Gibbs sampling
# set seed
set.seed(123456)

# Gibbs sampling for normal model with non-conjugate prior - L7S15

# prior parameter for 'sigma2'
v_0 <- 1 # how much we trust the prior's var's mean is
sigma2_0 <- var(rainfall$precipitation) # the variacne of the data
# sigma2
sigma2 <- sigma2_0 # need this value for the first simulation

# prior parameter for 'mu'
tau2_0 <- 1 # how much we trust the prior's mu's var is
#tau2_0 <- sigma2_0
########
#Zijife: it's ok to define, since it will change automatically later
########
mu_0 <- mean(rainfall$precipitation)
# mu
mu <- rnorm(n = 1, mean = mu_0, tau2_0) # need this value for the first simulation

# the sampling
nDraws <- 1000
Gibbs_sampling <- data.frame("mu" = numeric(), "sigma2" = numeric())
Gibbs_sampling2 <- data.frame("mu" = numeric(), "sigma2" = numeric())

# for exercise 1 a) ii
# create vairables for mean and varaicne
mean_vec <- vector(mode = "numeric", length = nDraws)
variance_vec <- vector(mode = "numeric", length = nDraws)

for (i in 1:nDraws) {
  # full conditional posterior
  # generate mu - need mu_n (need w to generate mu_n) & tau2_n
  # mu_n - L2S3 - Normal data, known variance - normal prior

  #-- generate mu -------------------------------------------------------- start
  # generate w
```

```r
  w <- (n/sigma2)/((n/sigma2) + (1/tau2_0))
  # generate mu_n
  mu_n <- w * mean(rainfall$precipitation) + (1 - w) * mu_0

  # genearte tau2_n
  tau2_n <- 1 / (n/sigma2 + 1/tau2_0)

  #-- generate mu
  mu <- rnorm(n = 1, mean = mu_n, sd = tau2_n)
  #-- generate mu ---------------------------------------------------------- end

  old_sigma2 <- sigma2
  # generate sigma2 ------------------------------------------------------ start
  # generate v_n - L5s3 - Normal model with normal prior
  v_n <- v_0 + n

  # generate second term - to sample sigma2
  second_term <- (v_0 * sigma2_0 + sum((rainfall$precipitation - mu)^2))/ (n + v_0)

  #-- generate sigma2
  sigma2 <- rinvchisq(n = 1, df = v_n, scale = second_term)
  # generate sigma2 -------------------------------------------------------- end

  # save mu & sigma in df
  Gibbs_sampling2 <- rbind(Gibbs_sampling2, data.frame(mu, sigma2))
  if(i==1){
    sampling <- data.frame("mu" = mu, "sigma2" = sigma2)
  }else{
    sampling <- data.frame("mu" = c(mu, mu), "sigma2" = c(old_sigma2, sigma2))
  }
  # variables we want to save
  Gibbs_sampling <- rbind(Gibbs_sampling, sampling)
  #save the mean of mu and variance
  mean_vec[i] <- mean(Gibbs_sampling2$mu)
  variance_vec[i] <- var(Gibbs_sampling2$sigma2)

}
Gibbs_sampling2$iterations <- 1:nDraws

# plot mu and sigma2

# plot mu
plot_tace_mu <- ggplot(data = Gibbs_sampling2) +
  geom_line(aes(x = iterations, y = mu),color = "#e74c3c") +
  geom_smooth(aes(x = iterations, y = mu),method="gam") +
  ggtitle("Tace of mu")
  # plot sigma2
plot_tace_sigma2 <-ggplot(data = Gibbs_sampling2) +
  geom_line(aes(x = iterations, y = sigma2), color = "#2980b9") +
  geom_smooth(aes(x = iterations, y = sigma2), method = "gam")+
  ggtitle("Tace of sigma2")

grid.arrange(plot_tace_mu, plot_tace_sigma2, nrow = 2)
```

```r
# converge of mean
plot_converge_mu <- ggplot() +
  geom_line(aes(x = Gibbs_sampling2$iterations, y = mean_vec), color = "#e74c3c") +
  labs(title = "Convergence of mu", x = "interation", y = "mean")

# converge of sigma
plot_converge_sigma <-ggplot() +
  geom_line(aes(x = Gibbs_sampling2$iterations, y = variance_vec), color = "#2980b9") +
  labs(title = "Convergence of sigma2", x = "interation", y = "variance")

grid.arrange(plot_converge_mu, plot_converge_sigma, nrow = 2)
# plotting the trajectories
# plot data
plot_data_q1aii <- Gibbs_sampling[,1:2]
plot_data_q1aii <- plot_data_q1aii[1:100,] # test- just take the first x rows

plot_mu_vs_sigma_line <- ggplot(plot_data_q1aii,
                                aes(x = plot_data_q1aii$mu, y = plot_data_q1aii$sigma2)) +
  geom_path(aes(color = "Markov chains"),color="#990000") +
  geom_point(aes(x = plot_data_q1aii$mu[1],
                 y = plot_data_q1aii$sigma2[1],
                 color = "start"), size = 3) +
  geom_point(aes(x = plot_data_q1aii$mu[length(plot_data_q1aii$mu)],
                 y = plot_data_q1aii$sigma2[length(plot_data_q1aii$sigma2)],
                 color = "end"), size = 3) +
  ggtitle("mu vs sigma2 - trajectories 100 points") + ylab("sigma2") + xlab("mu") +
  theme(legend.position = "bottom")
plot_mu_vs_sigma_line
# plot data
plot_data_q1aii <- Gibbs_sampling2[,1:2]
plot_data_q1aii <- plot_data_q1aii[1:1000,] # test- just take the first x rows

plot_mu_vs_sigma_line <- ggplot(plot_data_q1aii,
                                aes(x = plot_data_q1aii$mu, y = plot_data_q1aii$sigma2)) +
  geom_point(color = "#990000") +
  ggtitle("mu vs sigma2 - point (1000)") + ylab("sigma2") + xlab("mu") +
  theme(legend.position = "bottom")
plot_mu_vs_sigma_line
# Set the prior hyperparameters suitably
# Evaluate the convergence of the sampler -> need to save mu's and sigma's

##########    BEGIN USER INPUT #################
# Data options
rawData <- rainfall
x <- as.matrix(rainfall$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
```

```r
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", "magenta")
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)
  }
  # Diving every column of piDraws by the sum of the elements in that column.
  piDraws = piDraws/sum(piDraws)
  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)
# nObs-by-nComp matrix with component allocations.
S <- t(rmultinom(nObs, size = 1 , prob = rep(1/nComp,nComp)))
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
```

```r
#ylim <- c(0,2*max(hist(x)$density))

# --------------------- Update - start --------------------
# we want to save mu's and sigma's for every interation
# we have two misxtured model - so two coloums & nIter (100) as rows
simulated_mu <- matrix(data = NA, nrow = nIter, ncol = 2)
simulated_sigma <- matrix(data = NA, nrow = nIter, ncol = 2)
# --------------------- Update - end --------------------

for (k in 1:nIter){
  #message(paste('Iteration number:',k))
  # Just a function that converts between different
  # representations of the group     allocations
  alloc <- S2alloc(S)
  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 - start --------------------
    simulated_mu[k,] <- mu
  # --------------------- Update - end ---------------------

  # 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 - start --------------------
    simulated_sigma[k,] <- sigma2
  # --------------------- Update - end ---------------------



  # 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("topleft", box.lty = 1, legend = c("Data histogram",components, 'Mixture'),
    # col = c("black",lineColors[1:nComp], 'red'), lwd = 2)
    # Sys.sleep(sleepTime)
  }

}

#########################    Helper functions    #############################

plot_simulated_mu <- ggplot() +
  geom_line(aes(x = 1:nIter, y = simulated_mu[,1], color = "mu_1")) +
  geom_line(aes(x = 1:nIter, y = simulated_mu[,2], color = "mu_2")) +
  labs(title = "Convergence of simulated mu components", x = "iteration", y = "mu")

plot_simulated_sigma <- ggplot() +
  geom_line(aes(x = 1:nIter, y = simulated_sigma[,1], color = "sigma2_1")) +
  geom_line(aes(x = 1:nIter, y = simulated_sigma[,2], color = "sigma2_2"))+
  labs(title = "Convergence of simulated sigma components", x = "iteration", y = "simga")


grid.arrange(plot_simulated_mu, plot_simulated_sigma, nrow = 2)
ggplot(data = as.data.frame(x), aes(x = V1)) +
  geom_histogram(aes(y=..density..),
                 colour="black",
                 fill="white",
                 bins=20)
# plot data

plot_data_q1c2 <-  rnorm(n = nDraws, mean = Gibbs_sampling2$mu[dim(Gibbs_sampling2)[1]],
                         sd = sqrt(Gibbs_sampling2$sigma2[dim(Gibbs_sampling2)[1]]))

# the plot
ggplot(data = as.data.frame(plot_data_q1c2), aes(x = plot_data_q1c2)) +
    geom_histogram(aes(y=..density..),
                 colour="black",
                 fill="white",
                 bins=30)+
  geom_density(alpha=.2, colour = "blue", size=1) +
  xlab("values")

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(
rm(list = ls())
data <- read.table("eBayNumberOfBidderData.dat", head = TRUE)
mdl.2a <- glm(nBids ~ ., data = data[-2], family = poisson)
summary(mdl.2a)
y <- data[,1]
X <- as.matrix(data[,2:10])

# rename X columns
covNames <- names(data)[2:10]
m <- dim(X)[2]

# setting up the prior
mu <- as.vector(rep(0,m))
Sigma <- 100 * solve(t(X)%*%X)

LogPostLogistic <- function(beta, y, X, mu, Sigma){

  M <- length(beta)

  # log the likelihood of Possion Distribution
  logLik <- -sum(exp(X%*%beta)) + sum(y%*%(X%*%beta))
  if (abs(logLik) == Inf){
    logLik <- -20000
    # Likelihood is not finite, stear the optimizer away from here!
    # by Teacher's idea
  }

  # prior follows multi-normal distribution with
  logPrior <- dmvnorm(beta, mu, Sigma, log = TRUE)

  # cuz we logarithmize the likelihood and prior,
  # posterior is the sum of them
  return(logLik + logPrior)
}

initVal <- as.vector(rnorm(dim(X)[2]))

OptimResults <- optim(initVal,
                   LogPostLogistic,
                   y = y,
                   X = X,
                   mu = mu,
                   Sigma = Sigma,
                   method = c("BFGS"),
                   control = list(fnscale=-1),
                   hessian = TRUE)  # output hessian matrix


# Printing the results to the screen
postMode <- OptimResults$par

# Posterior covariance matrix is -inv(Hessian)
postCov <- -solve(OptimResults$hessian)

# Computing approximate standard deviations.
approxPostStd <- sqrt(diag(postCov))

# Naming the coefficient by covariates
names(postMode) <- covNames
names(approxPostStd) <- covNames

print('The posterior mode is:')
print(postMode)

print('The approximate posterior standard deviation is:')
print(approxPostStd)
set.seed(12345)
RWMSampler <- function(LogPostFunc, num, c,...){

  # In random walk, all draws are from sample proposal
  draws <- data.frame(matrix(nrow = num, ncol = 9))
  colnames(draws) <- colnames(X)
  draws[1,] <- mvrnorm(n = 1, postMode, c*postCov)
  count <- 1
  i <- 0
  while(count<num) {
    i <- i+1
    theta.old <- as.numeric(draws[count,])
    theta.new <- mvrnorm(n = 1, theta.old, c*postCov)
    u <- runif(1,0,1)
    # LogPostFunc is the distribution of theta/posterior we assume
    left  <- LogPostFunc(theta.new,...)
    right <- LogPostFunc(theta.old,...)
    a <- min(1, exp(left-right))
    if(u<a){
      count <- count + 1
      draws[count,] <- theta.new
    }
  }
  cat("acceptance probability: ",num/i,"\n")
  return(draws)
}

c <- 0.35
num <- 1200
RW <- RWMSampler(LogPostFunc = LogPostLogistic,
           num = num,
           c = c,
           y = y, X = X, mu = postMode, Sigma = postCov)

Index <- 1:num
data.2c <- cbind(RW, Index)
cnames <- colnames(data.2c)[1:9]
f <- function(cname){
  ggplot(data.2c, aes_string(x = Index, y = cname)) +
    geom_line()
}
plot(arrangeGrob(grobs = lapply(cnames, f)))
f <- function(cname){
  ggplot(data.2c, aes_string(x = cname)) +
    geom_histogram(aes(y=..density..),
                   colour="black",
                   fill="white",
                   bins=30)+
  geom_density(alpha=.2, colour = "blue", size=1)
}
plot(arrangeGrob(grobs = lapply(cnames, f)))

t <- round(t(mdl.2a$coefficients), 3)
colnames(t) <- names(postMode)
results <- rbind(t, round(t(postMode),3),round(colMeans(RW),3))
results <- cbind(step = c("Q2a","Q2b","Q2c"), results)

kable(results) %>%
  kable_styling(latex_options="scale_down")
# the given R file - NormalMixtureGibbs.R

# Estimating a simple mixture of normals
# Author: Mattias Villani, IDA, Linkoping University. http://mattiasvillani.com

##########    BEGIN USER INPUT #################
# Data options
data(faithful)
rawData <- faithful
x <- as.matrix(rawData['eruptions'])

# Model options
nComp <- 4    # 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 <- 1000 # Number of Gibbs sampling draws

# Plotting options
plotFit <- TRUE
```

```r
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)
  }
  # Diving every column of piDraws by the sum of the elements in that column.
  piDraws = piDraws/sum(piDraws)
  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 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){
  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 =
  }

  # 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),
    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("topleft", box.lty = 1, legend = c("Data histogram",components, 'Mixture'),
           col = c("black",lineColors[1:nComp], 'red'), lwd = 2)
    Sys.sleep(sleepTime)
  }

}

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(
#########################    Helper functions    ##############################################
```