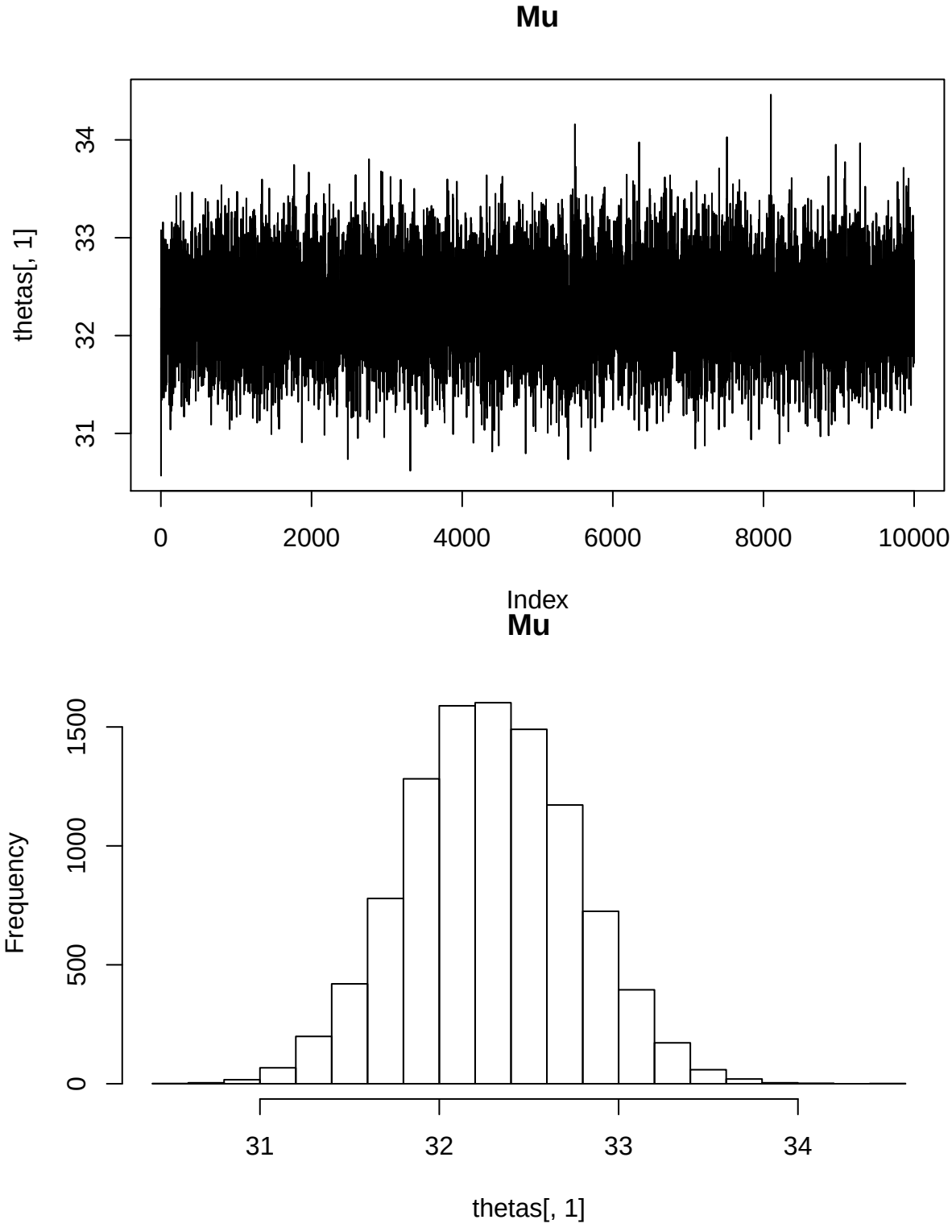
Lab_3_report

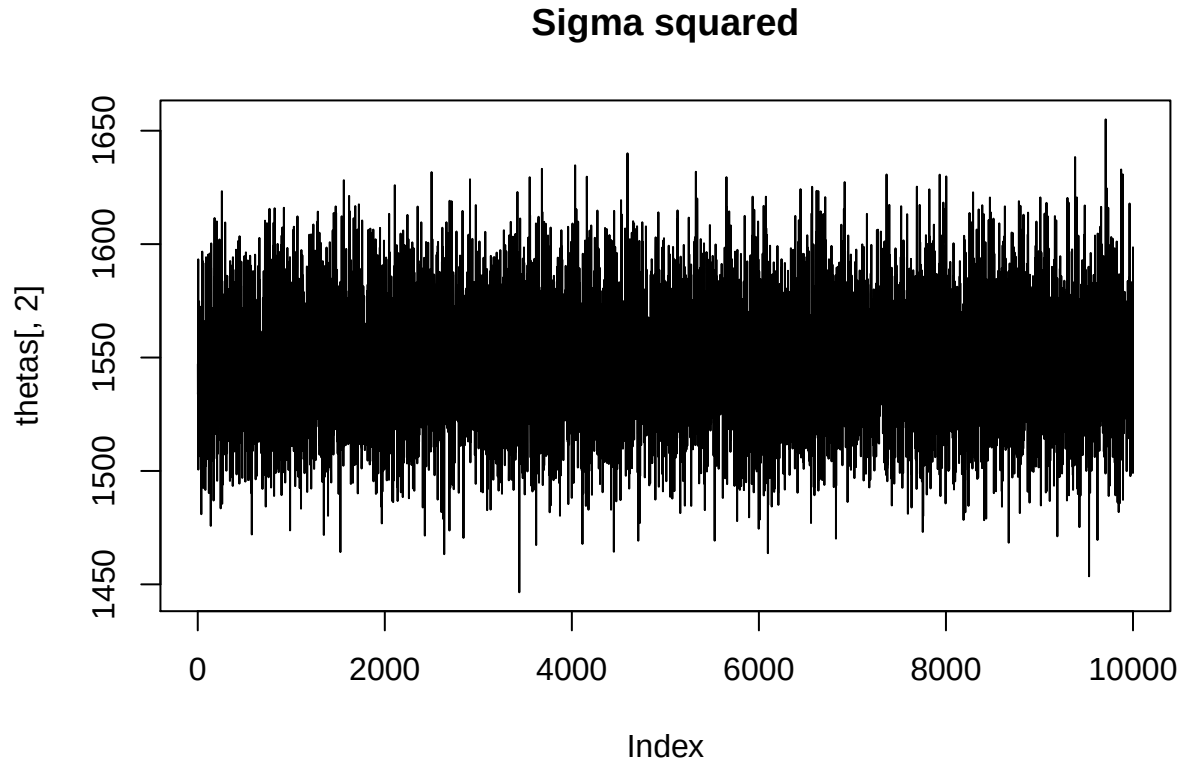
Axel Holmberg (axeho681), Wilhelm Hansson (wilha431)

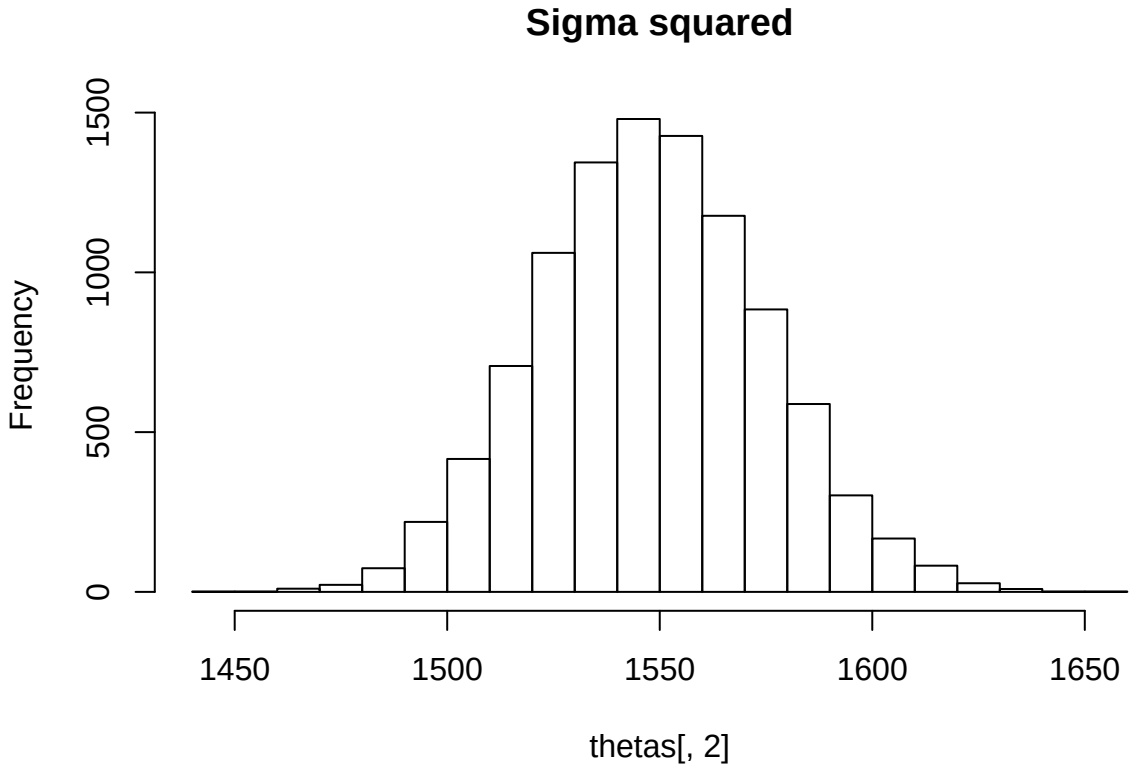
1

a)



As one can see above the values from the Gibbs model varies as it simulates μ for the joint posterior.





As one can see above the values from the Gibbs model varies as it simulates σ^2 for the joint posterior.

b)

The plots above shows iteration 1, 3, 5 and 1000 of the Gibbs sampling data augmentation algorithm. As one can see it starts at the same place, but as it iterates it splits into two normal distributions, each describing a

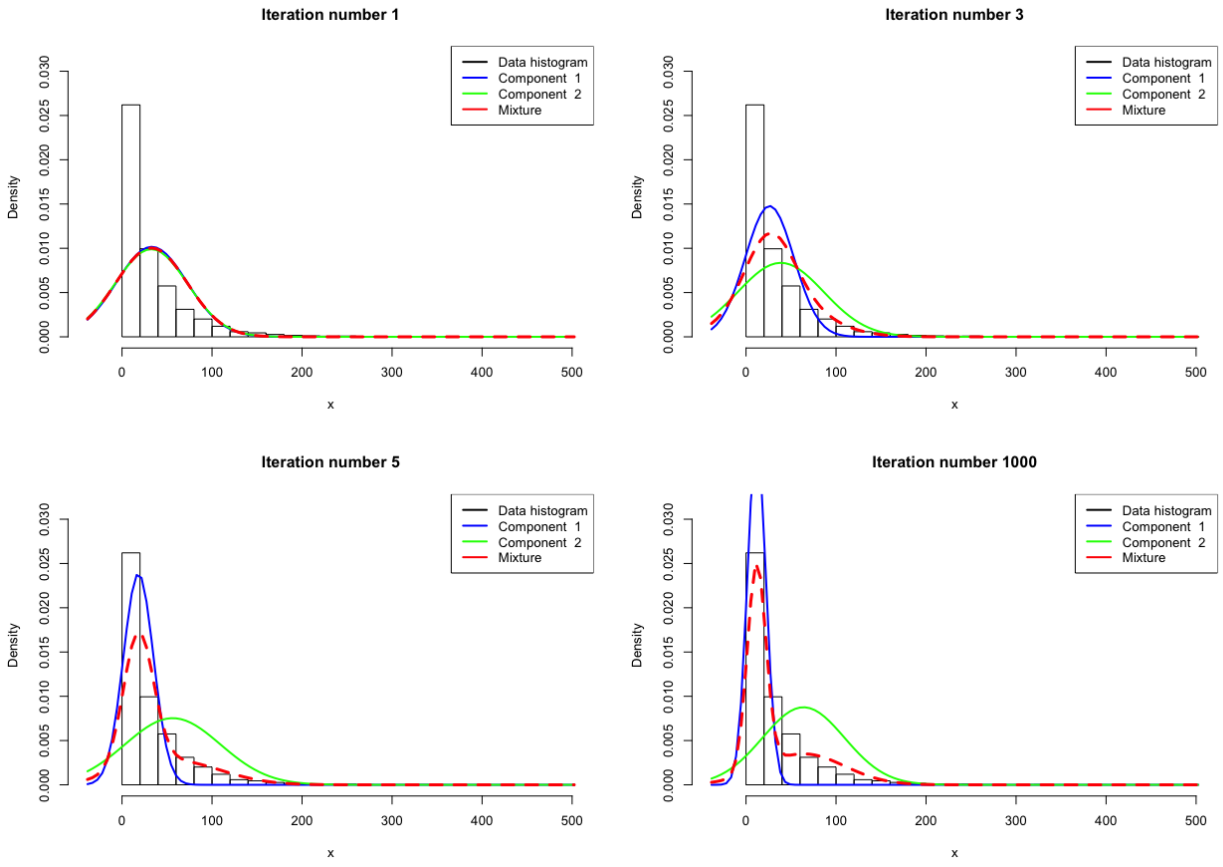


Figure 1: Iterations

part of the data well. The combined value of this can be seen in the dashed red line showing the Mixture normal model.

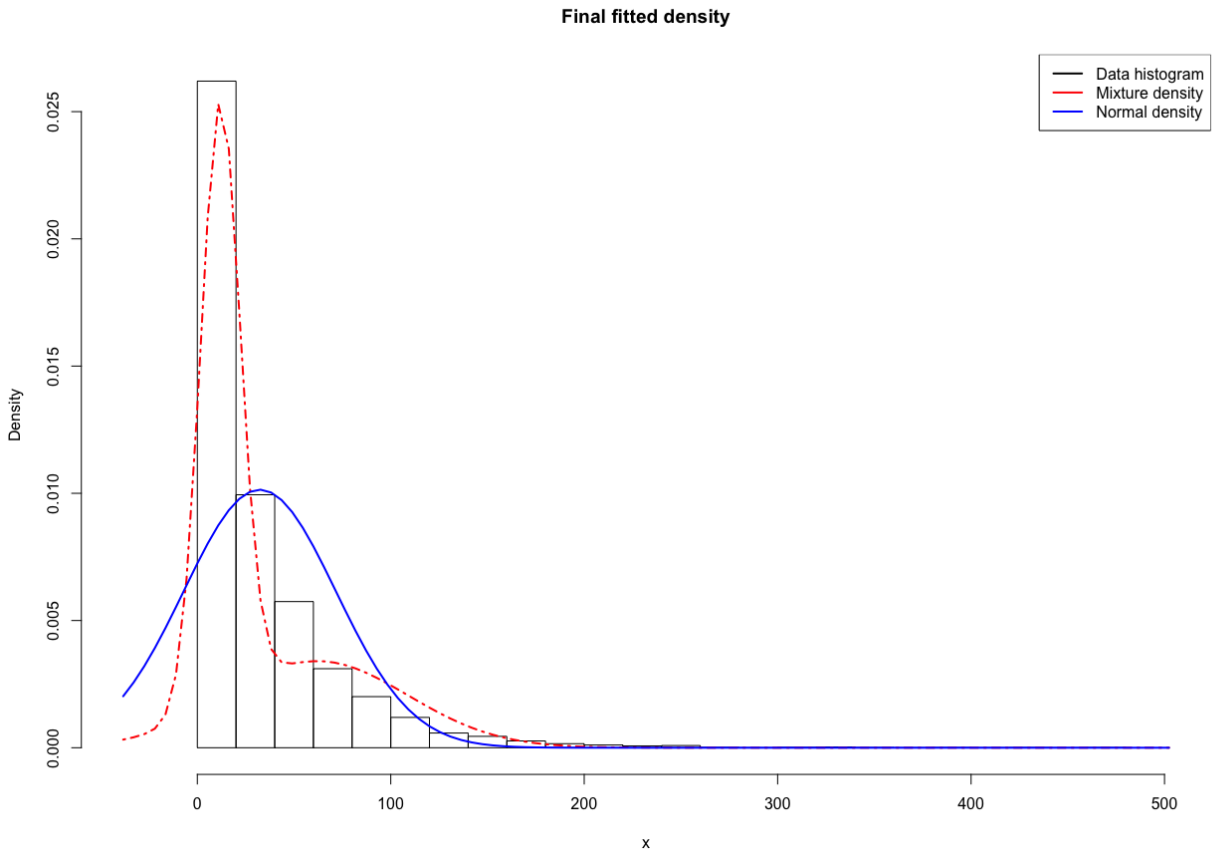


Figure 2: Final Fitted

The plot above shows the final fitted version of the mixture model as well as a normal distribution based on the mean and variance of the original data.

c)

The plot above shows the final fitted version of the mixture model as well as a normal distribution created from values from the Gibbs sampler from 1 a).

One can also see that the normal distribution based on the mean and variance of the original data is almost the exact same as the normal distribution created from values from the Gibbs sampler from 1 a).

2

a)

```
##
## Call:  glm(formula = nBids ~ . - Const, family = poisson, data = data)
##
## Coefficients:
## (Intercept)  PowerSeller     VerifyID       Sealed      Minblem      MajBlem
##     1.07244     -0.02054     -0.39452      0.44384     -0.05220     -0.22087
```

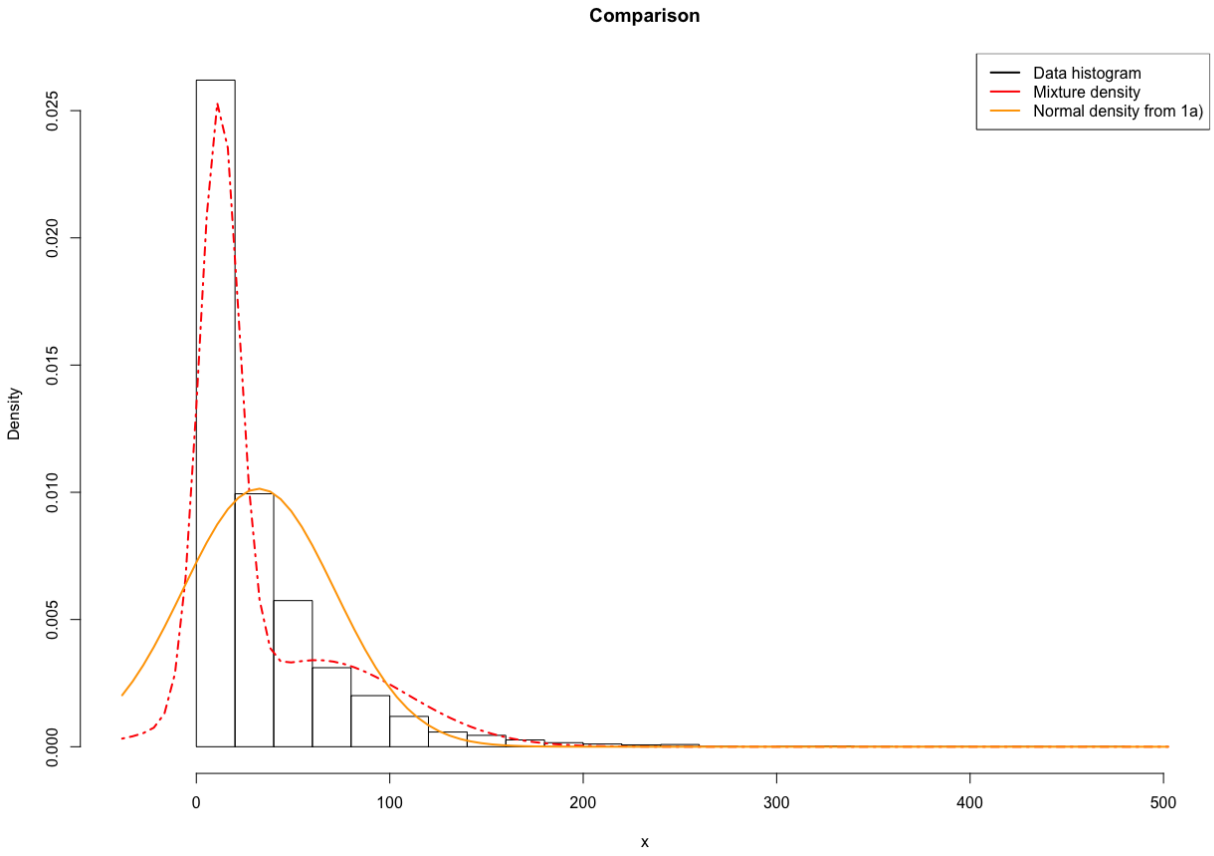


Figure 3: Comparison

```
##     LargNeg      LogBook  MinBidShare
##     0.07067     -0.12068     -1.89410
##
## Degrees of Freedom: 999 Total (i.e. Null);  991 Residual
## Null Deviance:       2151
## Residual Deviance: 867.5     AIC: 3610
```

The interesting part to note in the model above are the coefficients for each variable as these will be compared to in b) and c). The ones that are most significant are MinBidShare, Sealed and VerifyID.

b)

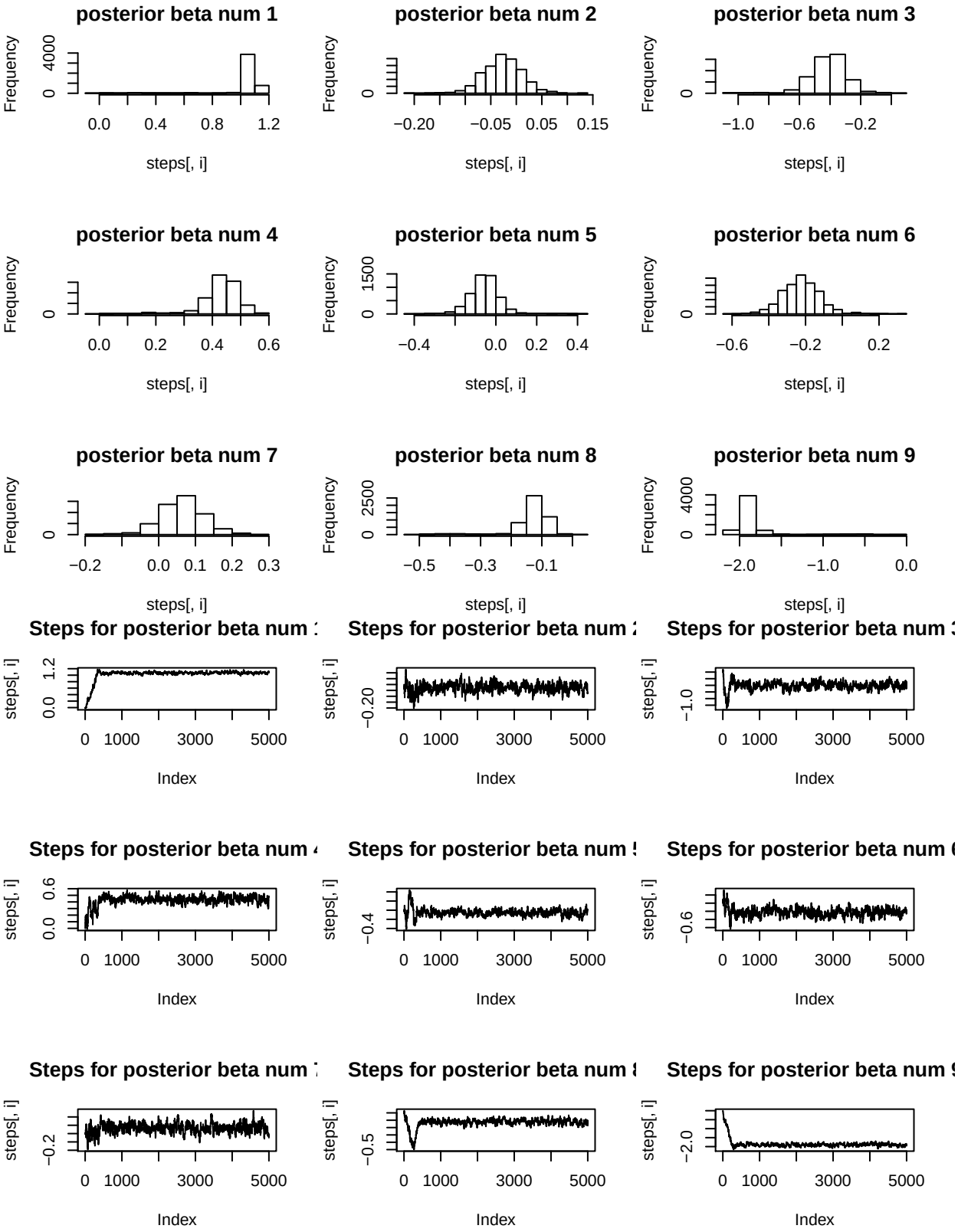
$J_y^{-1}(\tilde{\beta})$ is:

```
##                [,1]          [,2]          [,3]          [,4]          [,5]
##  [1,]  9.454667e-04 -7.138995e-04 -2.741525e-04 -2.709022e-04 -4.454562e-04
##  [2,] -7.138995e-04  1.353078e-03  4.024965e-05 -2.948992e-04  1.142968e-04
##  [3,] -2.741525e-04  4.024965e-05  8.515492e-03 -7.824897e-04 -1.013613e-04
##  [4,] -2.709022e-04 -2.948992e-04 -7.824897e-04  2.557772e-03  3.577153e-04
##  [5,] -4.454562e-04  1.142968e-04 -1.013613e-04  3.577153e-04  3.624626e-03
##  [6,] -2.772235e-04 -2.082682e-04  2.282530e-04  4.532323e-04  3.492351e-04
##  [7,] -5.128371e-04  2.801788e-04  3.313573e-04  3.376477e-04  5.843972e-05
##  [8,]  6.436946e-05  1.181860e-04 -3.191858e-04 -1.311049e-04  5.854048e-05
##  [9,]  1.109944e-03 -5.685718e-04 -4.292790e-04 -5.759616e-05 -6.437198e-05
##                [,6]          [,7]          [,8]          [,9]
##  [1,] -2.772235e-04 -5.128371e-04  6.436946e-05  1.109944e-03
##  [2,] -2.082682e-04  2.801788e-04  1.181860e-04 -5.685718e-04
##  [3,]  2.282530e-04  3.313573e-04 -3.191858e-04 -4.292790e-04
##  [4,]  4.532323e-04  3.376477e-04 -1.311049e-04 -5.759616e-05
##  [5,]  3.492351e-04  5.843972e-05  5.854048e-05 -6.437198e-05
##  [6,]  8.365211e-03  4.048646e-04 -8.975877e-05  2.622266e-04
##  [7,]  4.048646e-04  3.175037e-03 -2.541764e-04 -1.063199e-04
##  [8,] -8.975877e-05 -2.541764e-04  8.384734e-04  1.037434e-03
##  [9,]  2.622266e-04 -1.063199e-04  1.037434e-03  5.054784e-03
```

$\tilde{\beta}$ is:

```
## [1]  1.06983642 -0.02050871 -0.39302285  0.44356195 -0.05247189 -0.22125729
## [7]  0.07070931 -0.12022173 -1.89199799
```

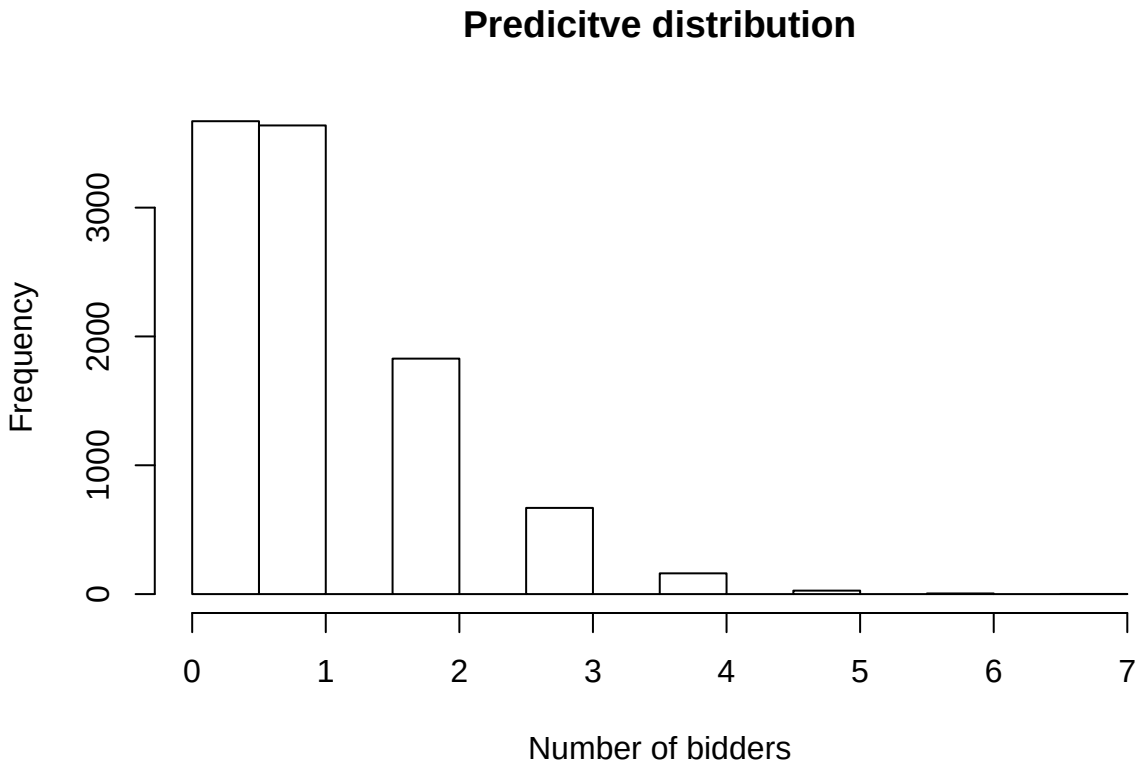
c)



As one can see in the histograms above the MCMC converges to about the same value as the coefficients for the GLM model in a). However in the case of bayesian approch we are presented with a distrubution for $\tilde{\beta}$

rather than a singular value. The line plots show how the value for each individual beta changed with each iteration. Here the line plots are squished together for the sake of the report. In RStudio we studied each plot one by one and came to the conclusion mentioned earlier.

d)



In the histogram above one can see the predicitve distribution for the number of bidders given the vector (1, 1, 1, 1, 0, 0, 0, 1, 0.5) where the feature Const is set to 1 according to previous data. The $\tilde{\beta}$ is calculated using the median of the MCMC draws in c). Using these values and the given feature vector λ is calculated according to $\lambda = exp(x^T\beta)$. The predictive distriibution was simulated using $y|\beta \sim Poisson[\lambda]$. Using this simulated distribution the probablility of 0 bidders was calulated to be 0.3671.

Appendix for code

```
data <- read.delim("rainfall.dat", header = FALSE, sep = "\n")[, 1]

## 1
### a)

#Prior values
mu_0 <- mean(data)
sigma_0 <- sd(data)
tau_0 <- 5
v_0 <- 10
n <- length(data)
iterations <- 10000

#Prepping thetas for gibbs sampling
thetas_col <- c(rep(0, iterations))
thetas <- cbind(thetas_col, thetas_col)
thetas[1, ] = c(mu_0, sigma_0 ^ 2)

#Function for draws for mu from the conditional posterior
mu_draw_cond_post <- function(sigma_2) {
    tau_n_2 <- 1 / (n / sigma_2 + 1 / tau_0 ^ 2)    #!
    raw <- (n / sigma_2) / (n / sigma_2 + 1 / tau_0 ^ 2)#!
    mu_n <- raw * mu_0 + (1 - raw) * mu_0
    return(rnorm(1, mu_n, sqrt(tau_n_2)))
}

#Fucntion for random inverse chi squared
randominvchisq <- function(vn, sigman, ndraw) {
    return(vn * sigman / rchisq(ndraw, vn))
}

#Function for draws for sigma from the conditional posterior
sigma_draw_cond_post <- function(mu) {
    v_n <-  n + v_0 #!
    sigma_n <- (v_0 * sigma_0 ^ 2 + sum((data - mu) ^ 2)) / v_n #!
    return(randominvchisq(v_n, sigma_n, 1))
}

#Function for draws from the h
gibbs_draw <- function(theta_n) {
    mu <- mu_draw_cond_post(theta_n[2])
    sigma_2 <- sigma_draw_cond_post(mu)
    return(c(mu, sigma_2))
}


#Draws for the set number of observations. First is already set above, hence we start at 2
for (i in 2:iterations) {
    thetas[i, ] <- gibbs_draw(thetas[i - 1, ])
}

plot(thetas[, 1], type = "l", main = "Mu")
```

```r
plot(thetas[, 2], type = "l", main = "Sigma squared")

hist(thetas[, 1], main = "Mu")
hist(thetas[, 2], main = "Sigma squared")


### b)

##########    BEGIN USER INPUT #################
# Data options

x <- as.matrix(read.delim("rainfall.dat", header = FALSE, sep = "\n"))

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

# Prior options
alpha <- 10 * rep(1, nComp) # Dirichlet(alpha)
muPrior <- rep(mu_0, nComp) # Prior mean of mu
tau2Prior <- rep(tau_0 ^ 2, nComp) # Prior std of mu
sigma2_0 <- rep(var(x), nComp) # s20 (best guess of sigma2)
nu0 <- rep(v_0, 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 <-
    0.05 # 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 tha
    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 a
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 alloc
    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
        )
        #   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("black", "red", "blue"),
    lwd = 2
)



#c)
par(mfrow = c(1, 1))

hist(
    x,
    breaks = 20,
    freq = FALSE,
    xlim = c(xGridMin, xGridMax),
    main = "Comparison"
)
lines(
    xGrid,
    mixDensMean,
    type = "l",
    lwd = 2,
    lty = 4,
    col = "red"
```

```
)
lines(
    xGrid,
    dnorm(xGrid, mean(thetas[, 1]), sd = mean(sqrt(thetas[, 2]))),
    type = "l",
    lwd = 2,
    col = "orange"
)
legend(
    "topright",
    box.lty = 1,
    legend = c("Data histogram", "Mixture density", "Normal density from 1a)"),
    col = c("black", "red", "orange"),
    lwd = 2
)
```

```
setwd("~/Programming/TDDE07/Lab 3")
data <- read.table("eBayNumberOfBidderData.dat", header = TRUE)

# a)

library(glmnet)

model <- glm(nBids ~ . - Const, family = poisson, data = data)

#The covariates that affects the B more is MinBidShare, Sealed and VerifyID.


# b)
library(mvtnorm)
X <- as.matrix(data[, -1])
y <- as.vector(data$nBids)
XTX_inv <- solve(t(X) %*% X)
mu = rep(0, ncol(X))
initBeta <-
    as.vector(rmvnorm(1, mean = rep(0, nrow(XTX_inv)), sigma = (100 * XTX_inv)))


LogPostPoisson <- function(betaVect, y, X, mu, Sigma) {
    linPred <-  X %*% betaVect
    lambda <- exp(linPred)

    logLik <- sum(-log(factorial(y)) + y * linPred - lambda)

    if (abs(logLik) == Inf)
        logLik = -20000
    # Likelihood is not finite, stear the optimizer away from here!

    # evaluating the prior
    logPrior <- dmvnorm(betaVect, mu, Sigma, log = TRUE)

    #print(logLik + logPrior)
    #print(betaVect)
    # add the log prior and log-likelihood together to get log posterior
```

```
    return(logLik + logPrior)
}


OptimResults <-
    optim(
        initBeta,
        LogPostPoisson,
        gr = NULL,
        y,
        X,
        mu,
        100 * XTX_inv,
        method = c("BFGS"),
        control = list(fnscale = -1),
        hessian = TRUE
    )


#Results 2 b)
inverse_hessian <- solve(-OptimResults$hessian)
B_tilde <- OptimResults$par


# c)


sigmaInput <- inverse_hessian
initalThetaInput <- c(rep(0, nrow(sigmaInput)))
c_Input <- 0.5
noIterationsInput <- 5000



rwmSampler <-
    function(logPostFunc,
                    sigma,
                    initalTheta,
                    c,
                    noIterations,
                    ...) {
        outSteps <- matrix(nrow = noIterations, ncol = length(initalTheta))
        outSteps[1, ] <- initalTheta

        for (i in c(2:noIterations)) {
            nextTheta <- as.vector(rmvnorm(1, outSteps[i - 1, ], c * sigma))
            r <-
                exp(logPostFunc(nextTheta, ...) - logPostFunc(outSteps[i - 1, ], ...))


            if (r > runif(1)) {
                outSteps[i, ] <- nextTheta
            } else{
```

```
                outSteps[i, ] <- outSteps[i - 1, ]
            }
        }
        return(outSteps)
    }



steps <-
    rwmSampler(
        LogPostPoisson,
        sigmaInput,
        initalThetaInput,
        c_Input,
        noIterationsInput,
        y,
        X,
        mu,
        100 * XTX_inv
    )


for (i in c(1:9)) {
    hist(steps[, i], main = paste("posterior beta num", i))

}

for (i in c(1:9)) {
    plot(steps[, i],
             type = "l",
             main = paste("Steps for posterior beta num", i))

}



#d)

b_tilde <- c()
features <- c(1, 1, 1, 1, 0, 0, 0, 1, 0.5)
for (i in c(1:9)) {
    b_tilde[i] <- median(steps[, i])
}
lambda_tilde <- exp(t(features) %*% b_tilde)

predicted_draws <- rpois(10000, lambda_tilde)

hist(predicted_draws, main = "Predicitve distribution")

prob_for_zero_bidders <-
    sum(ifelse(predicted_draws == 0, 1, 0)) / (length(predicted_draws))
```