Lab 3

Lab 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 1/100 inch, and records of zero precipitation are excluded) at Snoqualmie Falls, Washington. Analyze the data using the following two models.

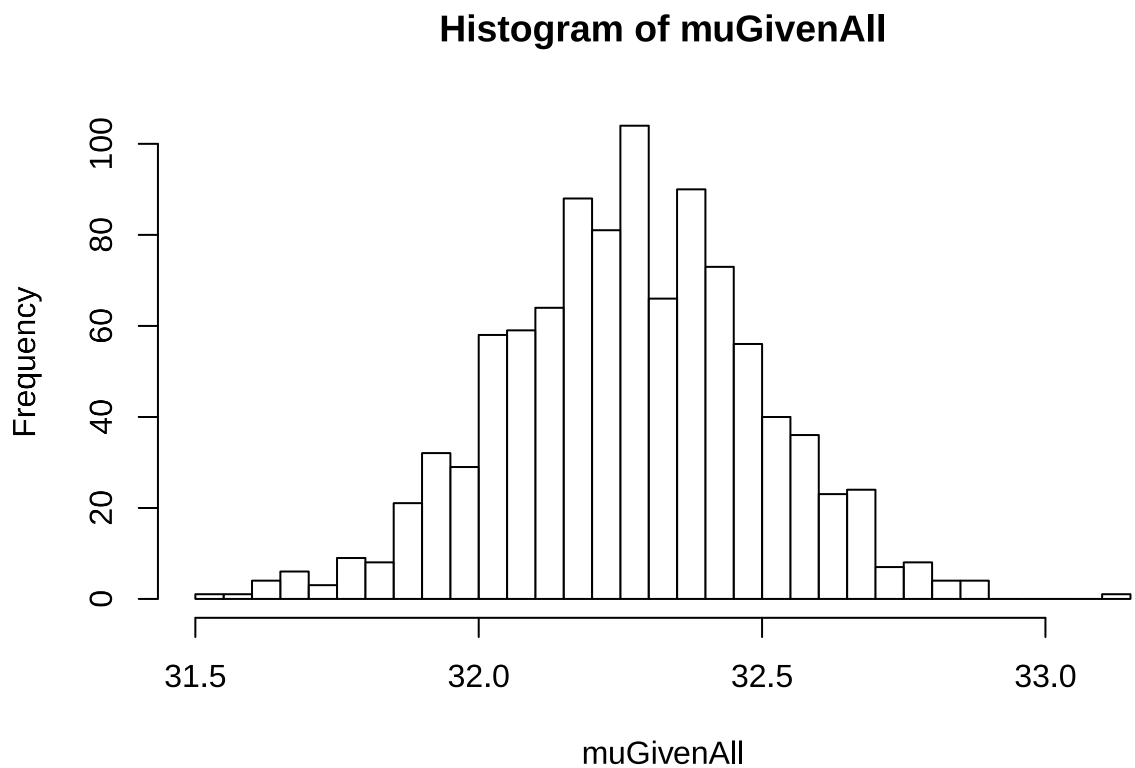
(a) Normal model. Assume the daily precipitation $y_1, ..., y_n$ are independent normally distributed, $y_1, ..., y_n|\mu, \sigma^2 \sim N(\mu, \sigma^2)$ where both μ and σ^2 are unknown. Let $\mu \sim N(\mu_0, \tau_0^2)$ independently of $\sigma^2 \sim Inv-\chi^2(v_0, \sigma_0^2)$.

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

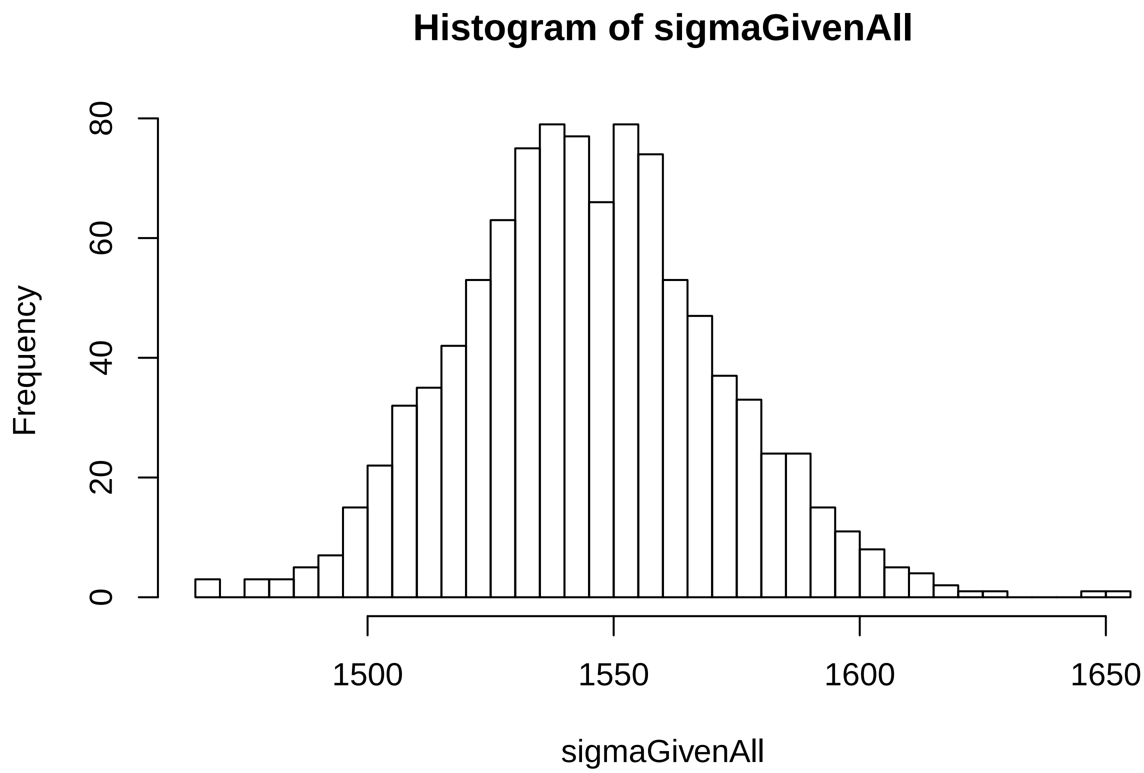
ii. Analyze the daily precipitation using your Gibbs sampler in (a)-i. Evaluate the convergence of the Gibbs sampler by suitable graphical methods, for example by plotting the trajectories of the sampled Markov chains.

```
filename='C:/Users/samue/Documents/LIU/TDDE07/LABS/TDDE07-Bayesian-Learning/lab 3/rainfall.dat'
Data<-read.table(filename,head=TRUE)
y=(as.vector(Data[,1]))
times=1000
tau0Sqr=1000
intitialSigma=1000
mu0=10
yMean=mean(y)
n=length(y)
v0=0

muGivenAll=c(1:times)
sigmaGivenAll=c(1:times)

for (i in 1:times){

  if(i>1){
  sigmaSqr=sigmaGivenAll[i-1]
  }else{
  sigmaSqr=intitialSigma

  }
  taunSqr=1/(n/sigmaSqr+1/tau0Sqr)
  w=n/sigmaSqr/(n/sigmaSqr+1/tau0Sqr)
  muN=w*yMean+(1-w)*mu0

  muGivenAll[i]=rnorm(1,muN,taunSqr)


  scaledSigma=(v0*intitialSigma+sum((y-muGivenAll[i])^2))/(n+v0)
```

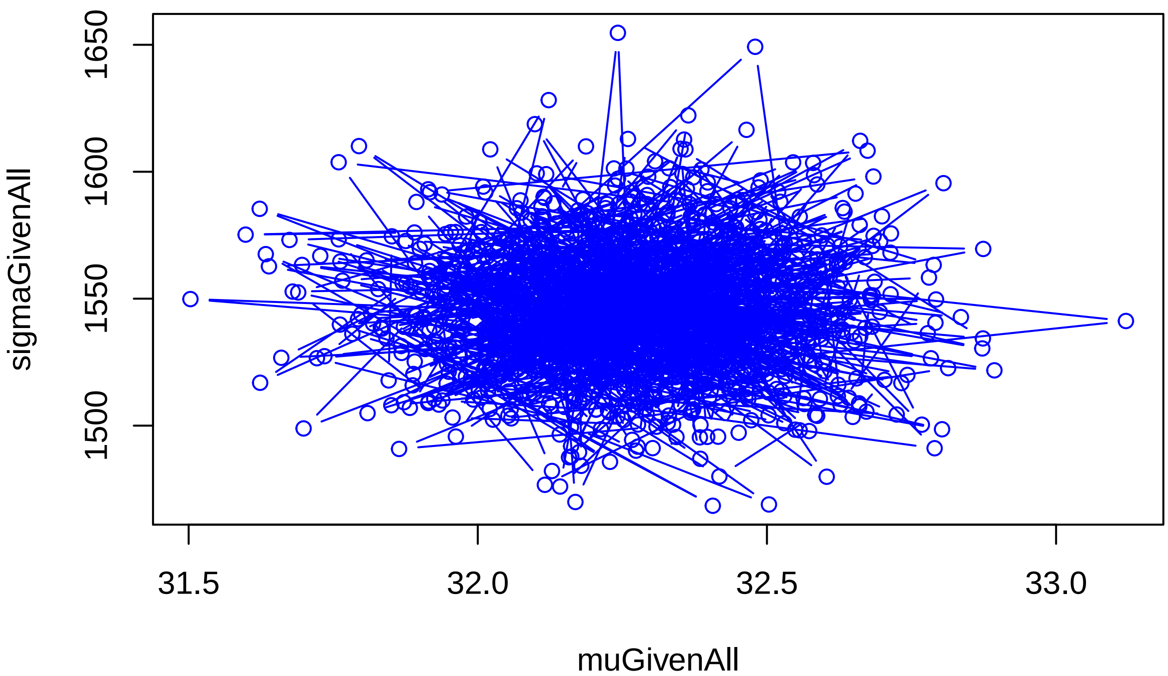
```
  x_draw = rchisq(1,v0+n)
  sigmaGivenAll[i]= ((v0+n)*scaledSigma)/x_draw

}

hist(muGivenAll, breaks = 30)
```




```
hist(sigmaGivenAll, breaks = 30)
```




```r
plot(muGivenAll,sigmaGivenAll,col='blue', type='b')
```



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:

$$p(y_i|\mu, \sigma^2, \sigma) = \pi N(y_i|\mu_1, \sigma_1^2) + (1-\pi) N(y_i|\mu^2, \pi_2^2)$$

, where

$$\mu = (\mu_1, \mu_2)$$

and

$$\sigma^2 = (\sigma_1^2, \sigma_2^2)$$

.

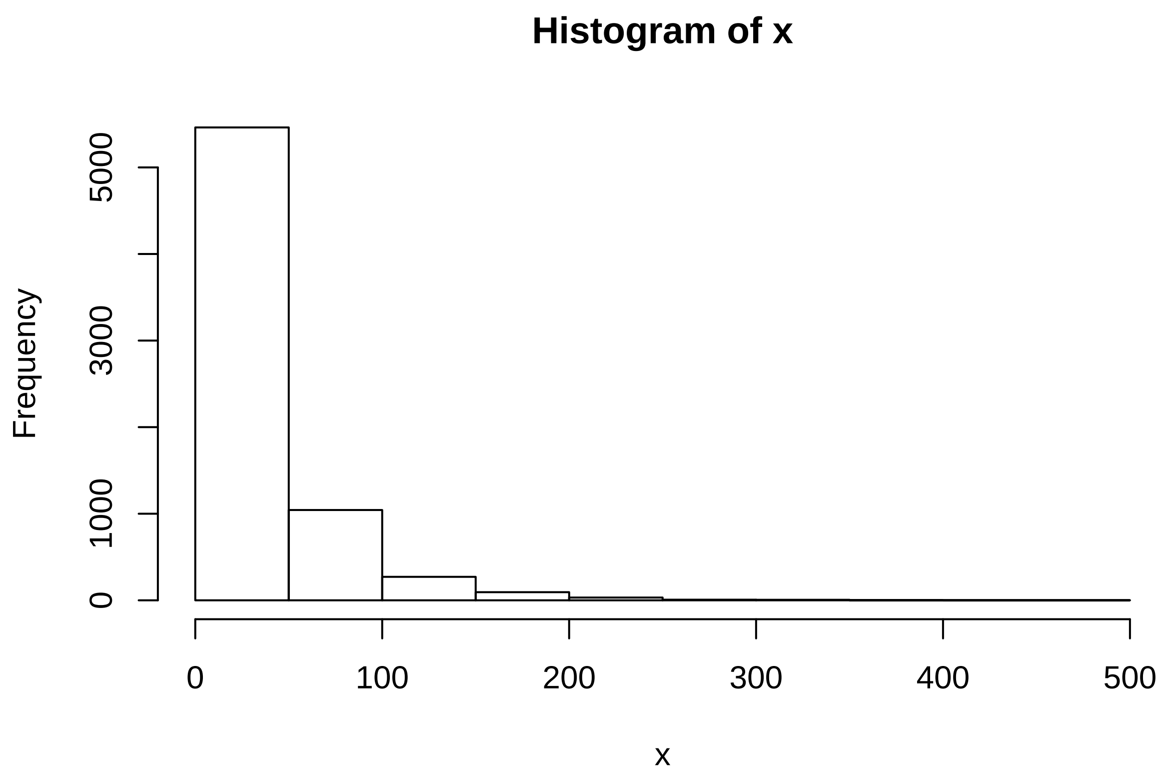
Use the Gibbs sampling data augmentation algorithm in NormalMixtureGibbs.R (available under Lecture 7 on the course page) to analyze the daily precipitation data. Set the prior hyperparameters suitably. Evaluate the convergence of the sampler.

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

##########    BEGIN USER INPUT #################
# Data options
meanHatt=mean(muGivenAll)
sigmaHatt=mean(sigmaGivenAll)

data(y)
```

```
## Warning in data(y): data set 'y' not found
```

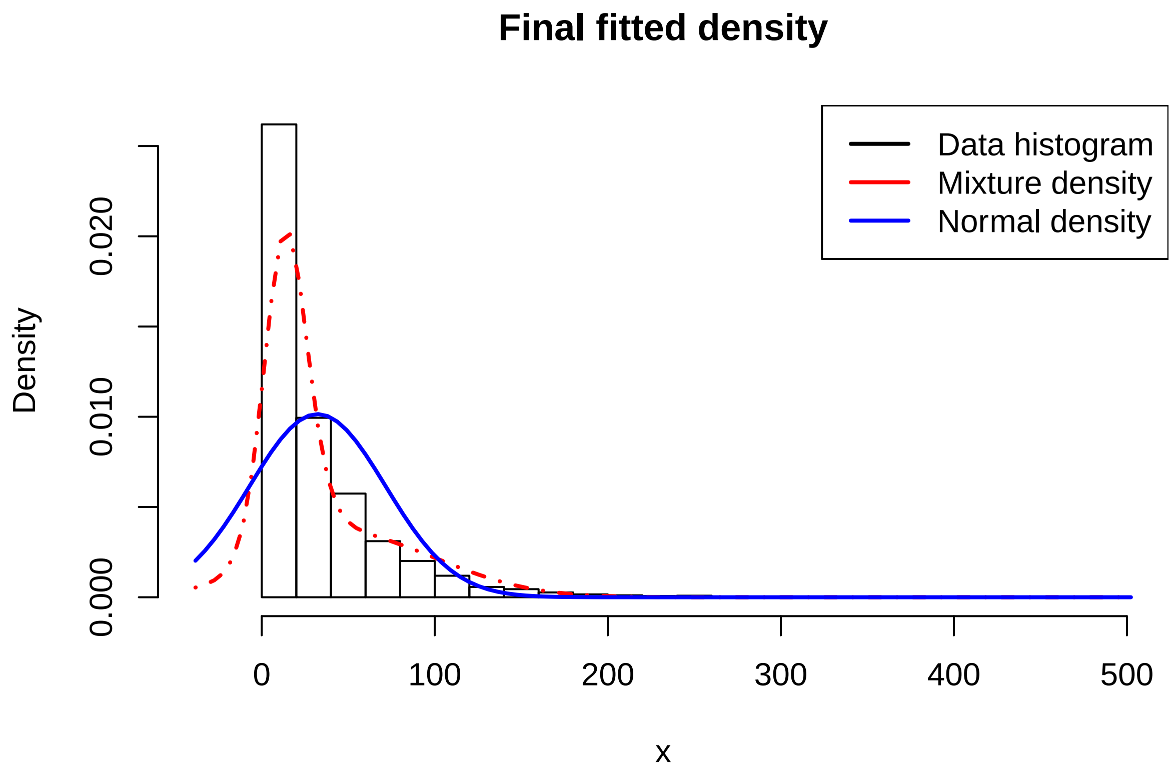
```r
rawData <- y
x <- as.matrix(y)

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

# Prior options
alpha <- 1*rep(1,nComp) # Dirichlet(alpha)
muPrior <- rep(meanHatt,nComp) # Prior mean of mu
tau2Prior <- rep(1500,nComp) # Prior std of mu
sigma2_0 <- rep(sigmaHatt,nComp) # s20 (best guess of sigma2)
nu0 <- rep(10,nComp) # degrees of freedom for prior on sigma2

# MCMC options
nIter <- 20 # Number of Gibbs sampling draws

# Plotting options
plotFit <- TRUE
lineColors <- c("blue", "green", "magenta", 'yellow')
sleepTime <- 0.1 # Adding sleep time between iterations for plotting
################   END USER INPUT ###############

###### Defining a function that simulates from the
rScaledInvChi2 <- function(n, df, scale){
  return((df*scale)/rchisq(n,df=df))
}

####### Defining a function that simulates from a Dirichlet distribution
rDirichlet <- function(param){
  nCat <- length(param)
  piDraws <- matrix(NA,nCat,1)
  for (j in 1:nCat){
    piDraws[j] <- rgamma(1,param[j],1)
  }
  piDraws = piDraws/sum(piDraws) # Diving every column of piDraws by the sum of the elements in that co
  return(piDraws)
}

# Simple function that converts between two different representations of the mixture allocation
S2alloc <- function(S){
  n <- dim(S)[1]
  alloc <- rep(0,n)
  for (i in 1:n){
    alloc[i] <- which(S[i,] == 1)
  }
  return(alloc)
}

# Initial value for the MCMC
nObs <- length(x)
S <- t(rmultinom(nObs, size = 1 , prob = rep(1/nComp,nComp))) # nObs-by-nComp matrix with component 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(
```



Convergence was fast. It converged already after around 10-20 iterations and became pretty stable. We think the mixture of two normal distibutions is reasonalble because some days are rainy and some days aren't. We see that the mixture fits the curve alot better than the single normal distribution. However it doesn't fully approximate the probability density of the "unrainy" days. We think this is because the normal distributions can take negative values which rain can't.

(c) Graphical comparison. Plot the following densities in one figure: 1) a histogram or kernel density estimate of the data. 2) Normal density $N(y_i|\mu, \sigma^2)$ in (a); 3) Mixture of normals density $p(y_i|\mu, \sigma^2, \pi)$ in (b). Base your plots on the mean over all posterior draws.

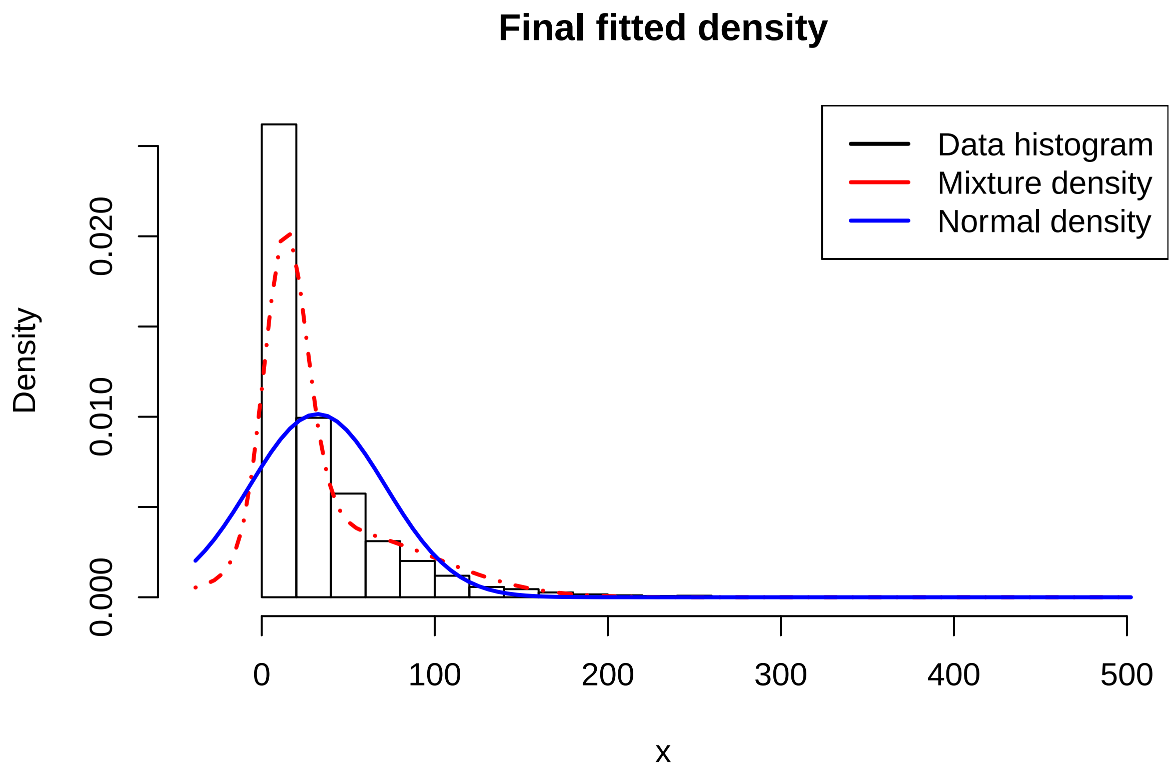
```
meanHatt
```

```
## [1] 32.26568
```

```
sigmaHatt
```

```
## [1] 1545.108
```

```
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 = meanHatt, sd = sigmaHatt^.5), type = "l", lwd = 2, col = "blue")
legend("topright", box.lty = 1, legend = c("Data histogram","Mixture density","Normal density"), col=c(
```



2

Metropolis Random Walk for Poisson regression. Consider the following Poisson regression model

$$y_i|\beta \sim Poisson[exp(x_i^T\beta)], i = 1, ..., n$$

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 (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?

```r
ebay = read.table("C:/Users/samue/Documents/LIU/TDDE07/LABS/TDDE07-Bayesian-Learning/lab 3/eBayNumberOfB

data = ebay[,-2]
model = glm(nBids~PowerSeller+VerifyID +Sealed +Minblem + MajBlem + LargNeg + LogBook + MinBidShare,fami
print(model$coefficients)
```

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

```r
sort(abs(exp(model$coefficients)-1))
```

```
## PowerSeller     Minblem     LargNeg     LogBook     MajBlem    VerifyID
##  0.02033123  0.05085936  0.07322964  0.11368035  0.19818005  0.32599414
##      Sealed MinBidShare (Intercept)
##  0.55868508  0.84954581  1.92250772
```

Answer: MinBidShare have the biggest impact, and Sealed the second. PowerSeller seems to not impact.

b

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

$$\beta \sim \mathcal{N}(\tilde{\beta}, J_y^{-1}(\tilde{\beta}))$$

where β is the posterior mode and $J_y(\tilde{\beta})$ is the negative Hessian at the posterior mode. β and $J_y(\tilde{\beta})$ 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.).

```r
library(mvtnorm)
X = as.matrix(ebay[,2:10])

y = ebay[,1]
prior_cov = t(X)%*%X
log_posterior_prob = function(beta, X, y){
  log_like=0
  for (i in 1:dim(X)[1]) {
    lambda = exp(t(X[i,])%*%beta)
    log_like = log_like +log(dpois(y[i], lambda = lambda))
  }


  prior_log_prob = log(dmvnorm(beta, mean = rep(0,nr_param), sigma = prior_cov))
  tot_log_like = dim(X)[1]*prior_log_prob + log_like
  return(tot_log_like)
}

log_faculty = function(y){
  log_fac = 0
  if (y != 0) {
    for (i in 1:y) {
      log_fac = log_fac + log(i)
```

```r
    }
  }

  return(log_fac)
}

log_posterior_prob = function(beta, X, y){
  log_like=0
  for (i in 1:dim(X)[1]) {
    x_b = t(X[i,])%*%beta
    part = x_b*y[i]-exp(x_b)-log_faculty(y[i])
    log_like = log_like + part
  }


  prior_log_prob = log(dmvnorm(beta, mean = rep(0,nr_param), sigma = prior_cov))
  tot_log_like = dim(X)[1]*prior_log_prob + log_like
  return(tot_log_like)
}
#startvalues for beta vector
nr_param = dim(X)[2]
init_beta = as.vector(rep(0,nr_param))
result = optim(init_beta, log_posterior_prob, gr=NULL, X, y, method=c("BFGS"), control=list(fnscale=-1)

beta_hat = result$par
j_y = -result$hessian
post_cov = solve(j_y)

beta_hat
```

```
## [1]  1.08631089 -0.02956338 -0.37504768  0.43868712 -0.04136304 -0.14881048
## [7]  0.07545489 -0.10113889 -1.80469202
```

```r
post_cov
```

```
##                [,1]          [,2]          [,3]          [,4]          [,5]
##  [1,]  8.938594e-04 -6.874665e-04 -1.990151e-04 -2.568737e-04 -4.104267e-04
##  [2,] -6.874665e-04  1.326874e-03  1.315401e-05 -2.805127e-04  1.029672e-04
##  [3,] -1.990151e-04  1.315401e-05  7.144150e-03 -6.446590e-04 -8.611320e-05
##  [4,] -2.568737e-04 -2.805127e-04 -6.446590e-04  2.461311e-03  3.133453e-04
##  [5,] -4.104267e-04  1.029672e-04 -8.611320e-05  3.133453e-04  3.431344e-03
##  [6,] -2.130695e-04 -1.625626e-04  1.167772e-04  3.227809e-04  2.321181e-04
##  [7,] -4.655383e-04  2.576643e-04  2.344052e-04  3.014851e-04  3.575370e-05
##  [8,]  3.537432e-05  1.259677e-04 -2.350057e-04 -1.257799e-04  5.705095e-05
##  [9,]  9.844461e-04 -5.094516e-04 -2.383358e-04 -6.596403e-05 -5.486901e-05
##                [,6]          [,7]          [,8]          [,9]
##  [1,] -2.130695e-04 -0.0004655383  3.537432e-05  9.844461e-04
##  [2,] -1.625626e-04  0.0002576643  1.259677e-04 -5.094516e-04
##  [3,]  1.167772e-04  0.0002344052 -2.350057e-04 -2.383358e-04
##  [4,]  3.227809e-04  0.0003014851 -1.257799e-04 -6.596403e-05
##  [5,]  2.321181e-04  0.0000357537  5.705095e-05 -5.486901e-05
##  [6,]  6.353754e-03  0.0002698320 -7.237625e-05  1.775382e-04
##  [7,]  2.698320e-04  0.0030102559 -2.289133e-04 -6.541320e-05
##  [8,] -7.237625e-05 -0.0002289133  8.084577e-04  9.434504e-04
```

```
##  [9,]  1.775382e-04 -0.0000654132  9.434504e-04  4.626835e-03
```

c

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):

$$\theta_p|\theta^{(i-1)} \sim \mathcal{N}(\theta^{(i-1)}, c * \Sigma)$$

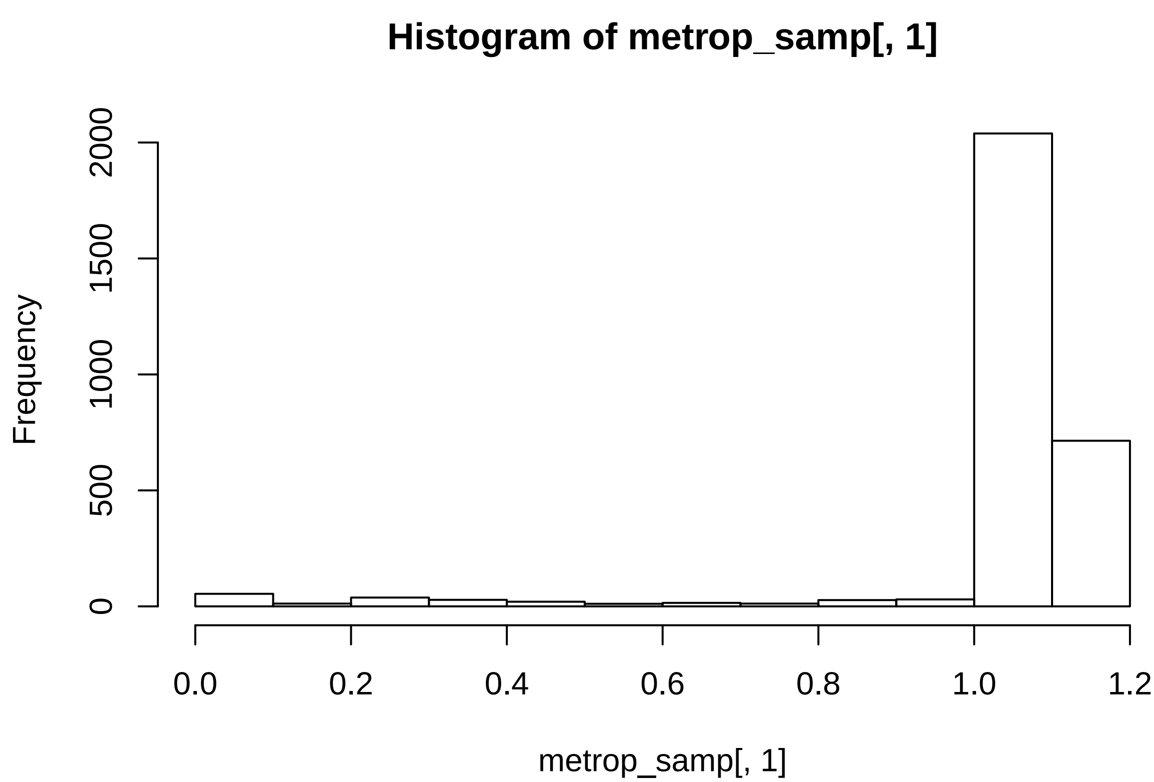
where $\Sigma = J_y^{-1}(\tilde{\beta})$ obtained in b). The value c is a tuning parameter and should be an input to your Metropolis function. The user of your Metropo- lis function should be able to supply her own posterior density function, not necessarily for the Poisson regression, and still be able to use your Metropolis function. This is not so straightforward, unless you have come across function objects in R and the triple dot (...) 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 */beta* in the Poisson regression for the eBay dataset. Assess MCMC convergence by graphical methods.

```r
#2c
library(MASS)
library(LaplacesDemon)
```

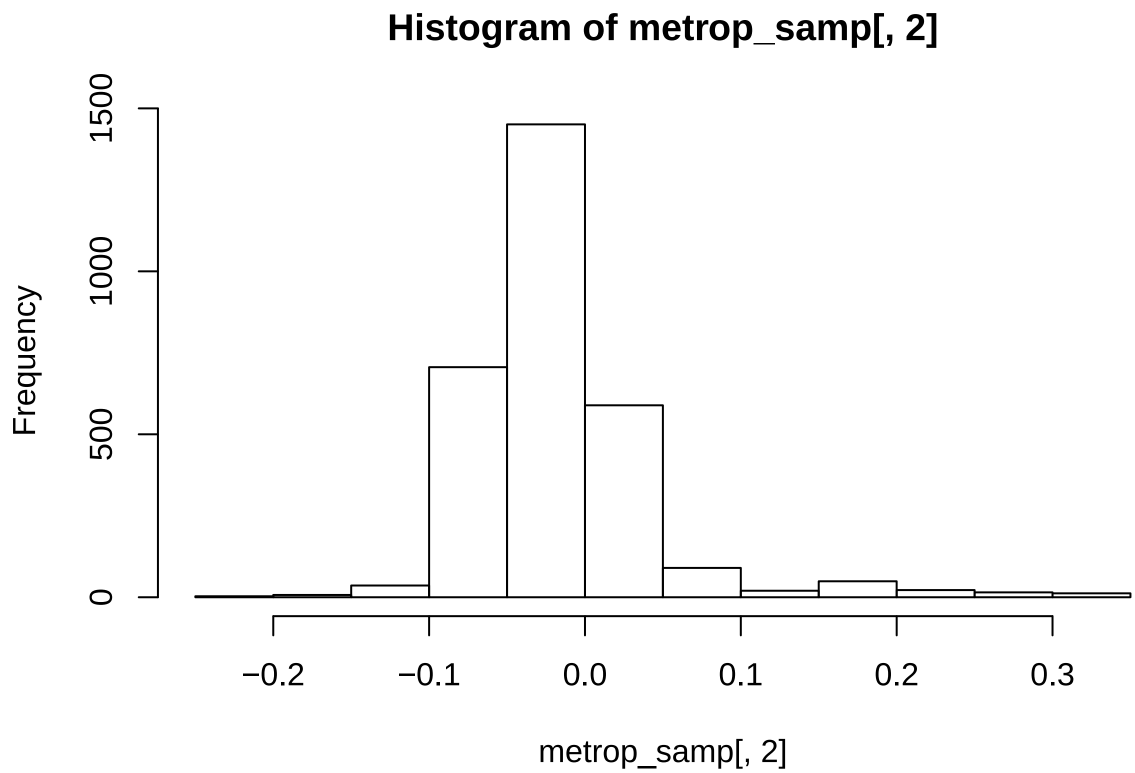
```
##
## Attaching package: 'LaplacesDemon'

## The following objects are masked from 'package:mvtnorm':
##
##     dmvt, rmvt
```

```r
proposal_generator = function(theta_prev, c, cov_mat){
  theta_p = mvrnorm(1, theta_prev, c*cov_mat)
  return(theta_p)
}

alpha_generator = function(proposal, theta_before, log_post_func, ...){

  new_prob = log_post_func(proposal,...)
  old_prob = log_post_func(theta_before, ...)

  alpha = exp(new_prob - old_prob)

  return(min(alpha, 1))
}

metropolis = function(init_theta, c, nr_iter, cov_mat,log_post_func, ...){

  metrop_sim = matrix(data=1, nrow = nr_iter, ncol = length(init_theta))
  metrop_sim[1,] = init_theta

  for (i in 2:nr_iter) {
    proposal = proposal_generator(metrop_sim[i-1,],c,cov_mat);

    #calculate apha
    alpha = alpha_generator(proposal, metrop_sim[i-1,],log_post_func, ...)
    #accept the proposal with proability alpha
```

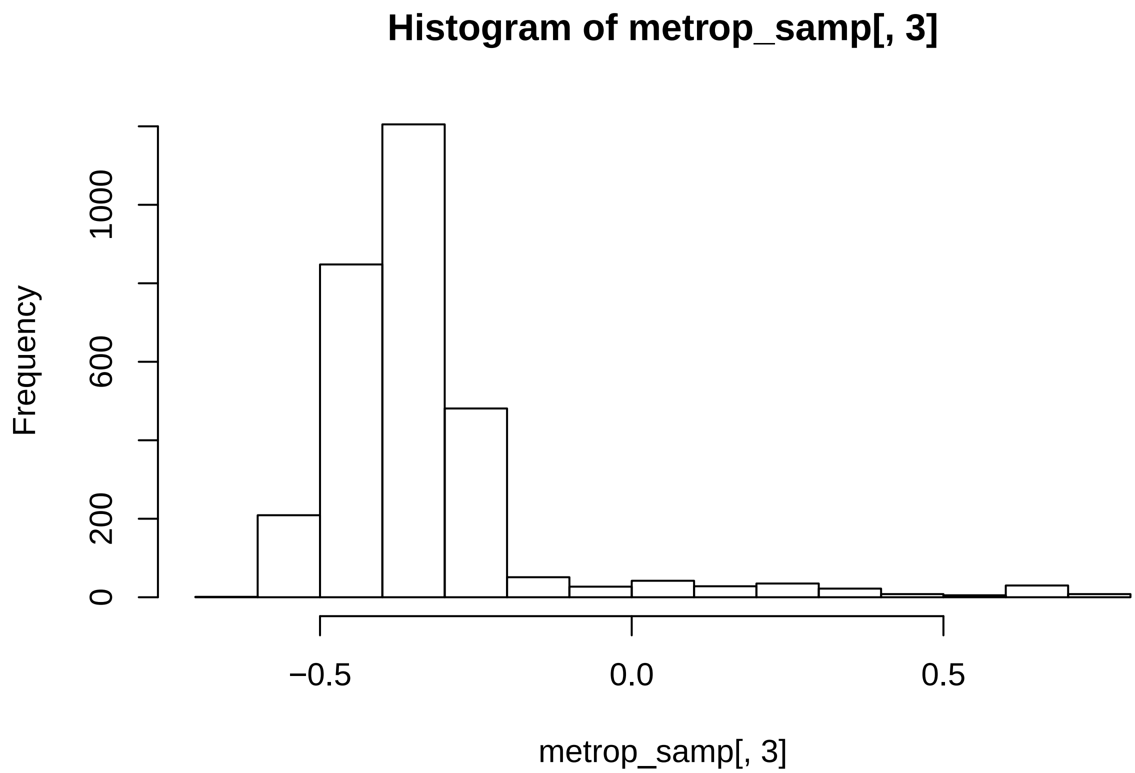
```r
    if( rbern(1, alpha)){
      metrop_sim[i,] = proposal
    }else{
      metrop_sim[i,] = metrop_sim[i-1,]
    }
  }
  return(metrop_sim)
}
init_theta = rep(0,length(beta_hat))
c=1
nr_iter=3000
cov_mat = post_cov

#proposal_generator(init_theta, c, cov_mat)
metrop_samp = metropolis(init_theta, c, nr_iter, cov_mat,log_posterior_prob,X,y)

hist(metrop_samp[,1])
```



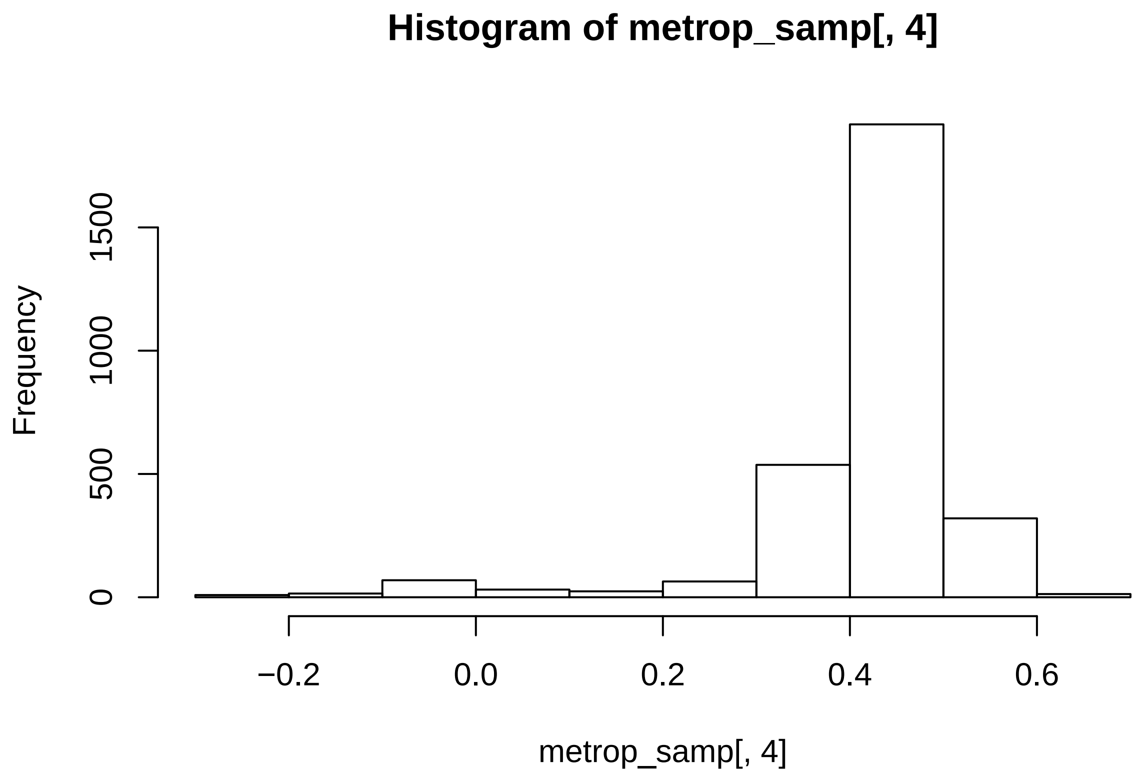

```r
hist(metrop_samp[,2])
```



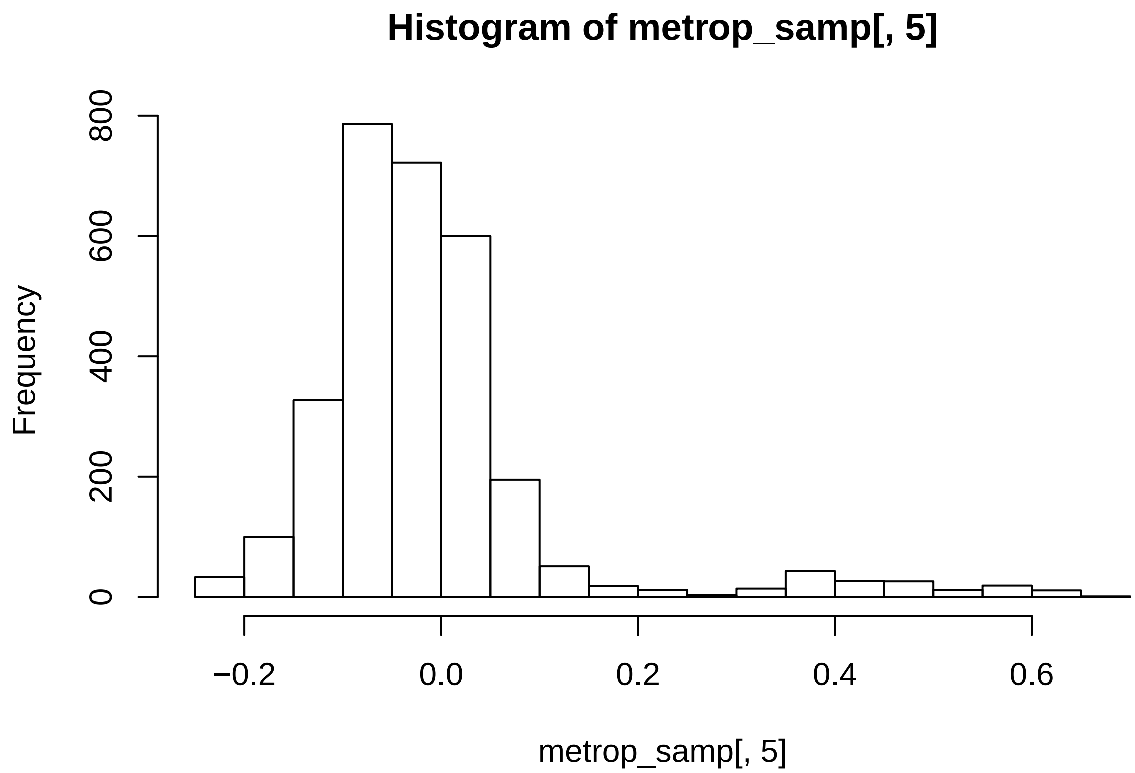

```r
hist(metrop_samp[,3])
```



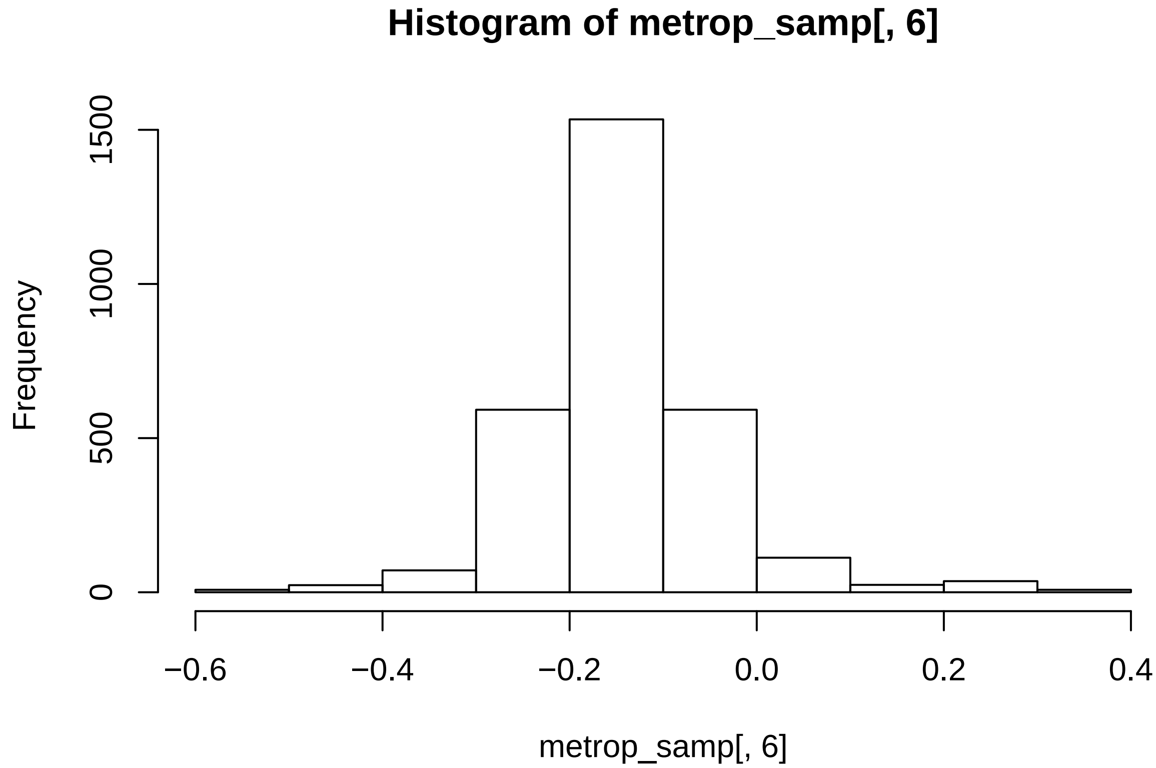

```
hist(metrop_samp[,4])
```



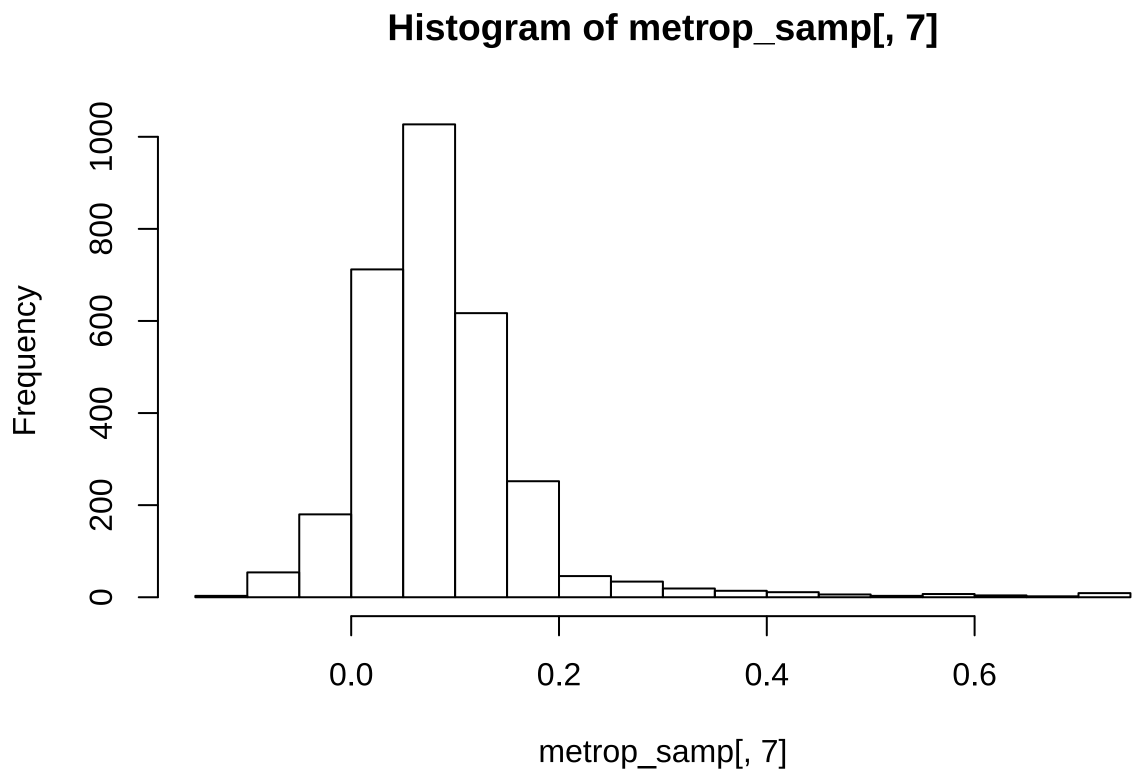

```r
hist(metrop_samp[,5])
```



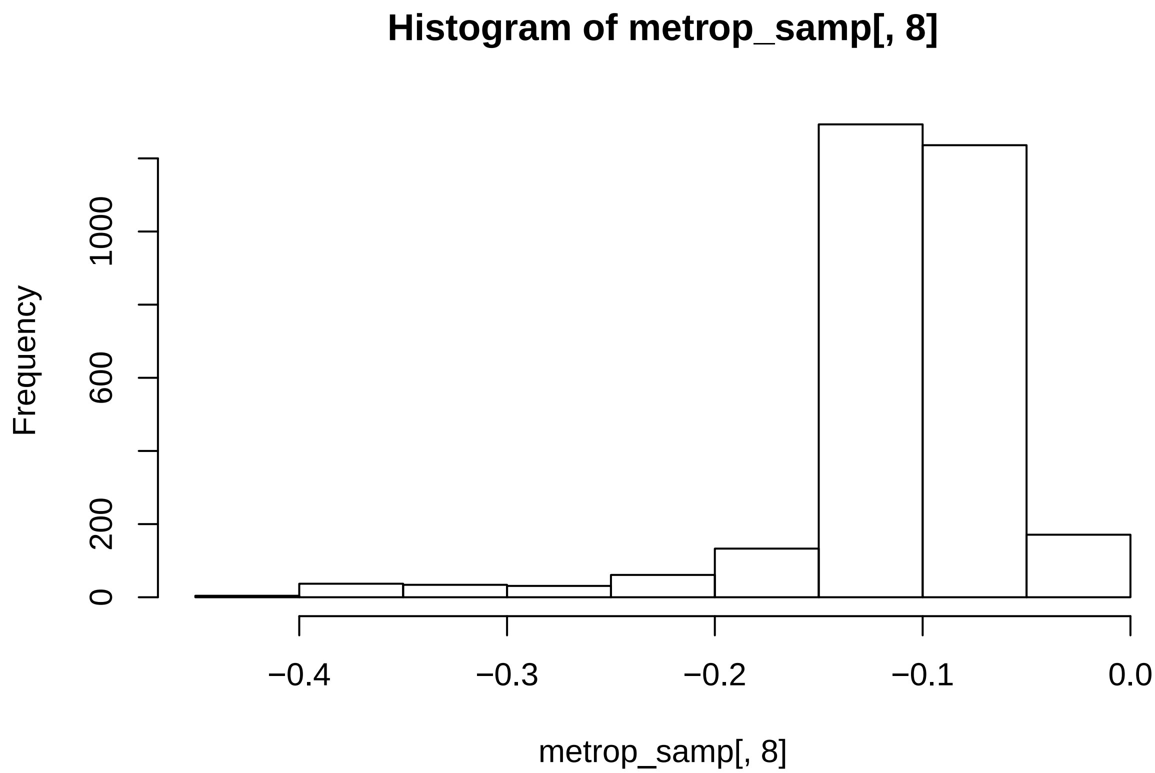

```
hist(metrop_samp[,6])
```



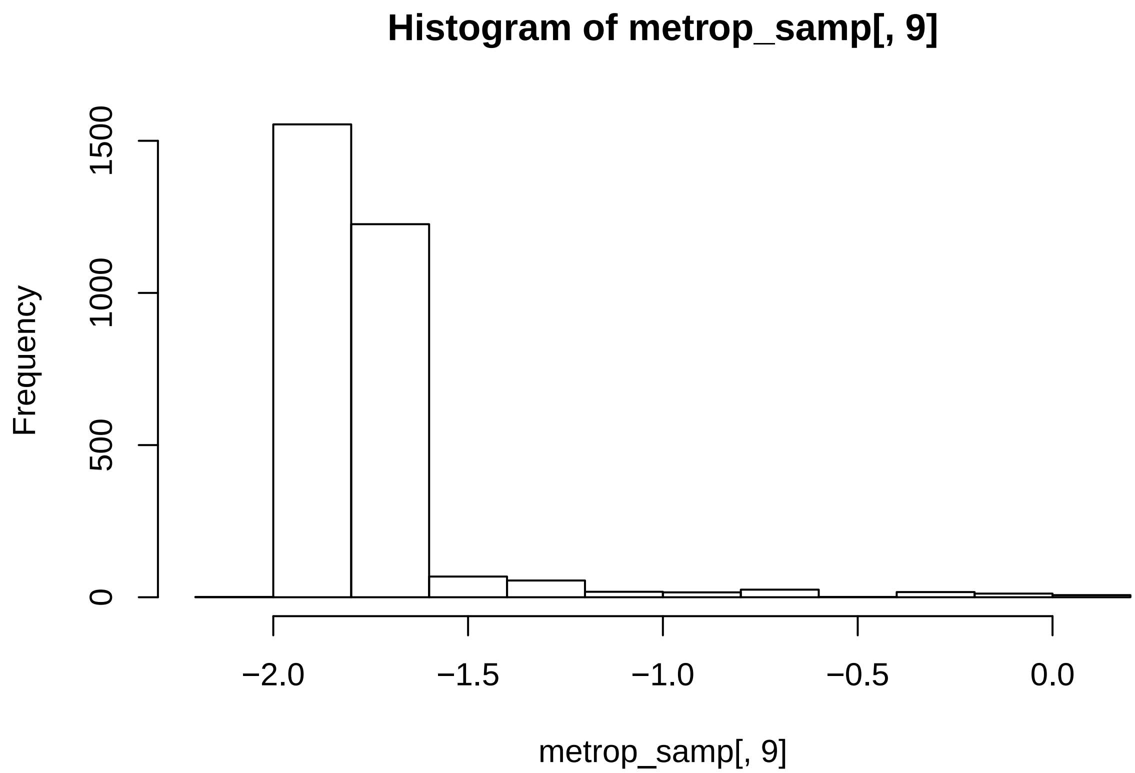

```
hist(metrop_samp[,7])
```




```
hist(metrop_samp[,8])
```



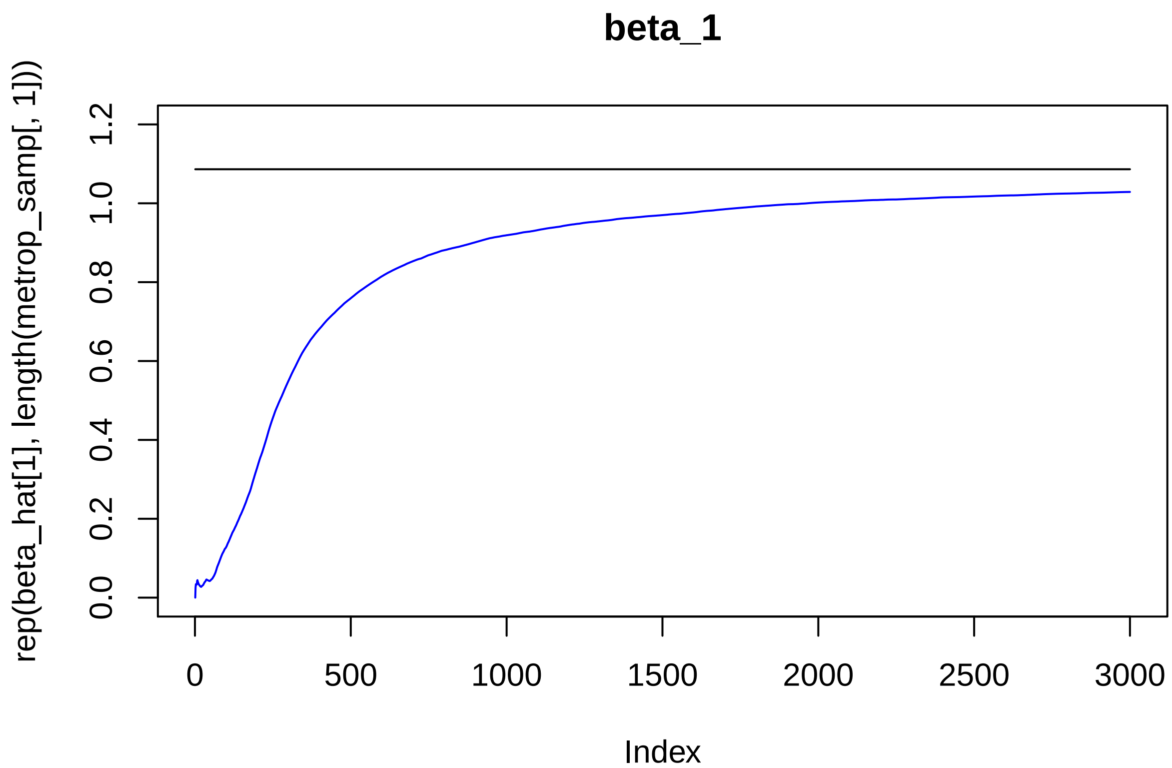

```
hist(metrop_samp[,9])
```



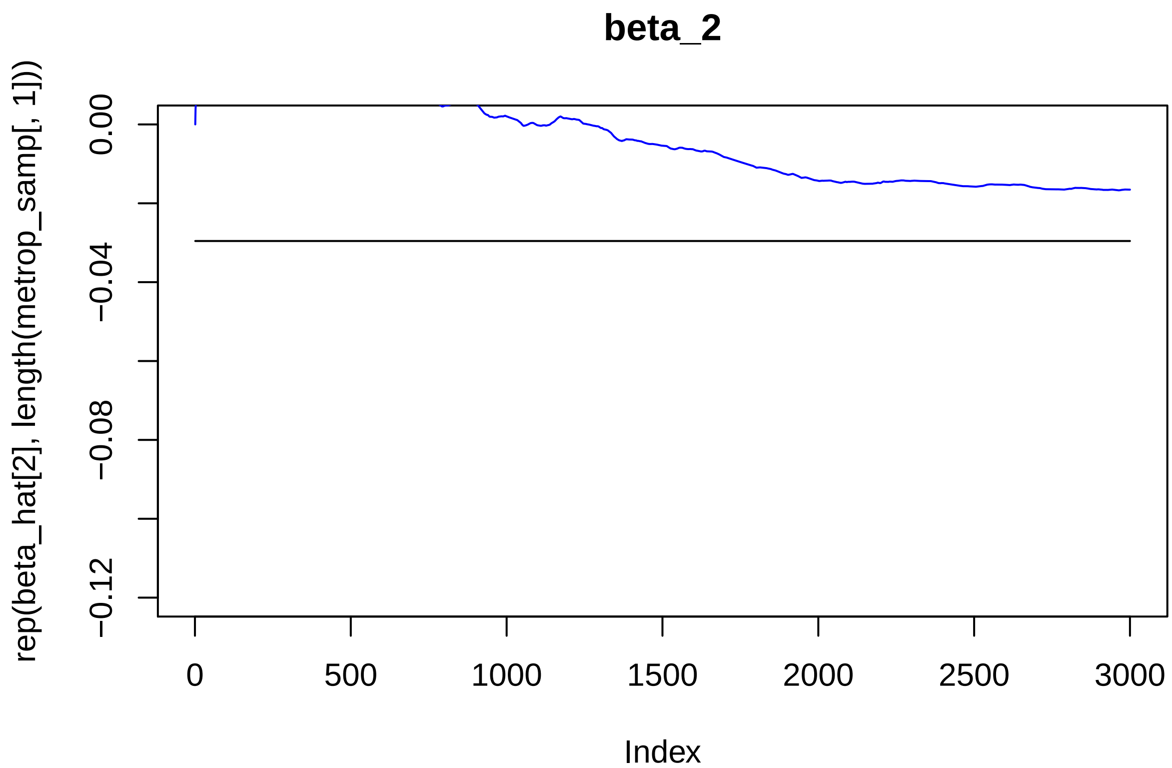

```
beta_hat
```

```
## [1]  1.08631089 -0.02956338 -0.37504768  0.43868712 -0.04136304 -0.14881048
## [7]  0.07545489 -0.10113889 -1.80469202
```

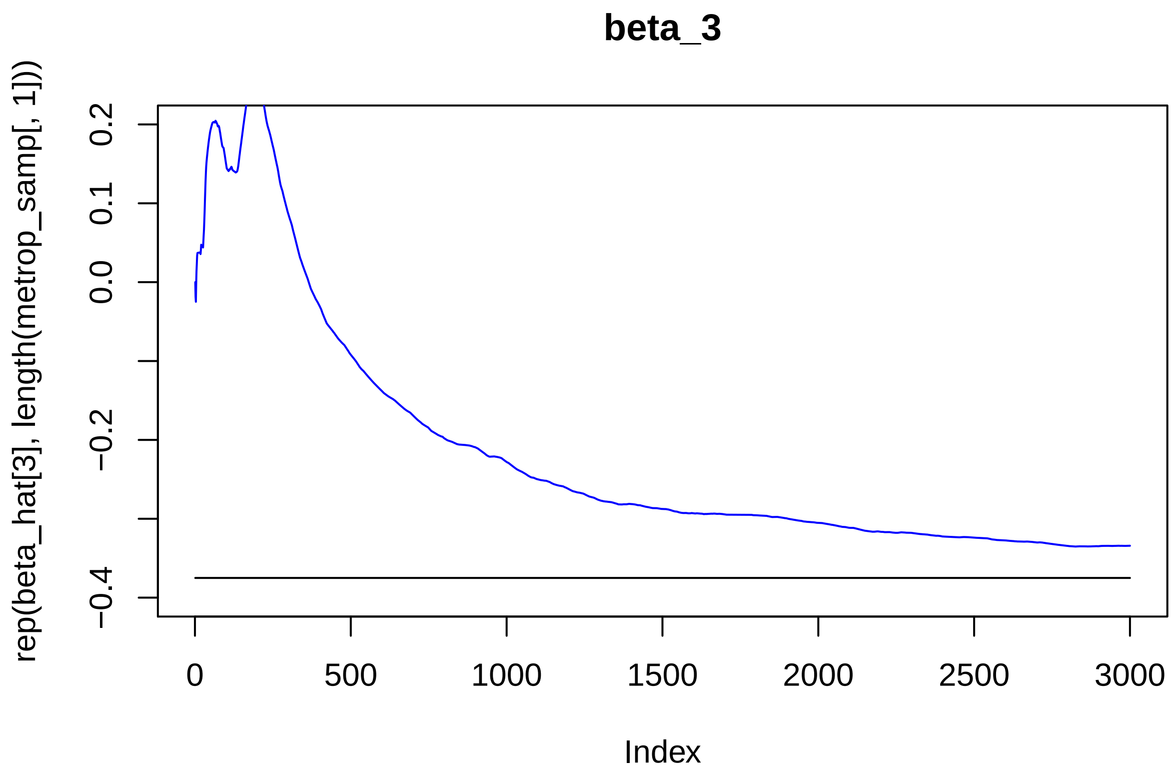
```
#y1 = cumsum(metrop_samp[,1]) / seq_along(metrop_samp[,1])
y1 = cumsum(metrop_samp[,1]) / 1:length(metrop_samp[,1])
y2 = cumsum(metrop_samp[,2]) / 1:length(metrop_samp[,2])
y3 = cumsum(metrop_samp[,3]) / 1:length(metrop_samp[,3])

#convergence plotted with beta_hat, beta by optimization from maximum posterior liklehood.
plot(rep(beta_hat[1],length(metrop_samp[,1])), type="l", main="beta_1", ylim = c(0,1.2))
lines(y1, type="l", col="blue")
```




```r
plot(rep(beta_hat[2],length(metrop_samp[,1])), type="l", main="beta_2", ylim = c(-0.12,0))
lines(y2, type="l", col="blue")
```




```r
plot(rep(beta_hat[3],length(metrop_samp[,1])), type="l", main="beta_3",ylim = c(-0.4,0.2))
lines(y3, type="l", col="blue")
```

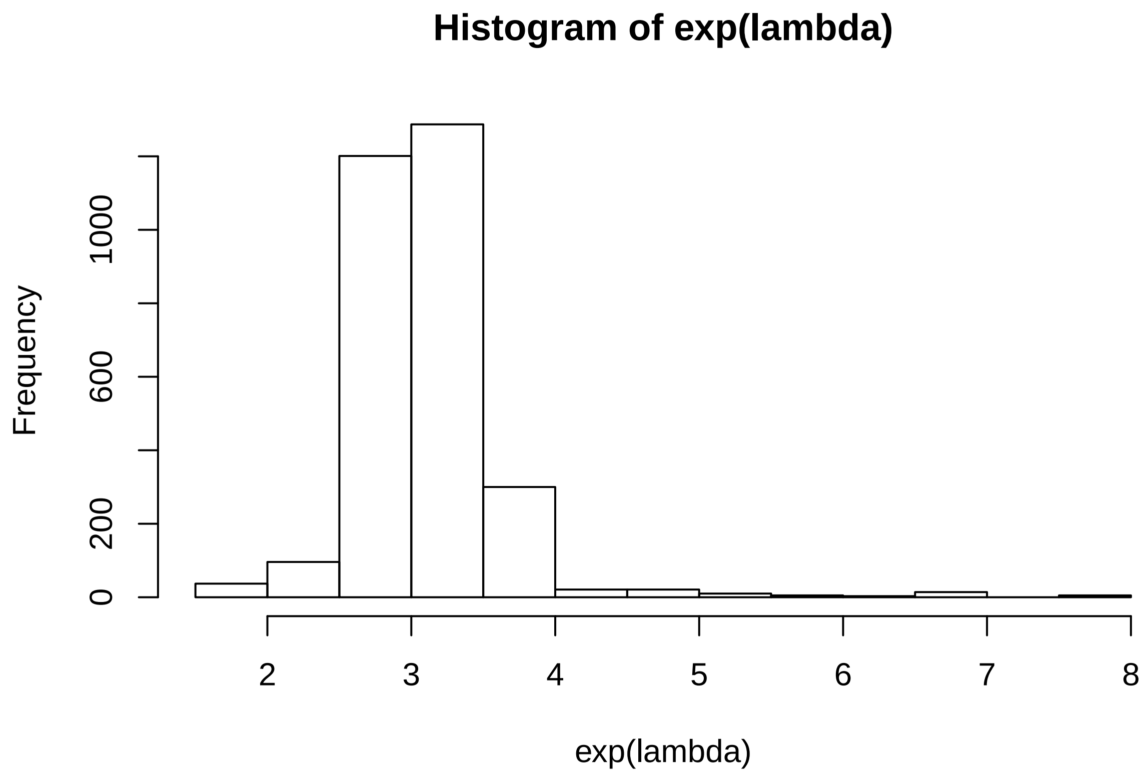


d

Use the MCMC draws from c) to simulate from the predictive distribution of the number of bidders in a new auction with the characteristics below. Plot the predictive distribution. What is the probability of no bidders in this new auction?

- PowerSeller = 1
- VerifyID = 1
- Sealed = 1
- MinBlem = 0
- MajBlem = 0
- LargNeg = 0
- LogBook = 1
- MinBidShare = 0.5

```
# 2d
x_new = c(1,1,1,1,0,0,0,1,0.5)

x_b_pred = metrop_samp%*%x_new
lambda = exp(x_b_pred)
hist(exp(lambda))
```




```
#poisson for all observed lambda to get y
y_prob_0 = rep(0,nr_iter)

for (i in 1:nr_iter) {

  y_prob_0[i] = dpois(0,lambda[i])

}
sum(y_prob_0)/nr_iter
```

```
## [1] 0.3301381
```