hw3

Shweta Singhal

April 8, 2016

```
require(matrixcalc)
```

```
## Loading required package: matrixcalc
```

```
read.image = function(filename){
  library(png)
  y = readPNG(filename)
  ## This is just how I encoded pixel intensities
  ## After this, the labels will be black = -1 and white = +1
  y = y * 20 - 10
  n = nrow(y)
  ## transpose and flip image
  t(y)[,n:1]
}

display.image = function(x, col=gray(seq(0,1,1/256))){
  w = dim(x)[1]
  h = dim(x)[2]
  par(mai = c(0,0,0,0))
  image(x, asp=h/w, col=col)
}
```

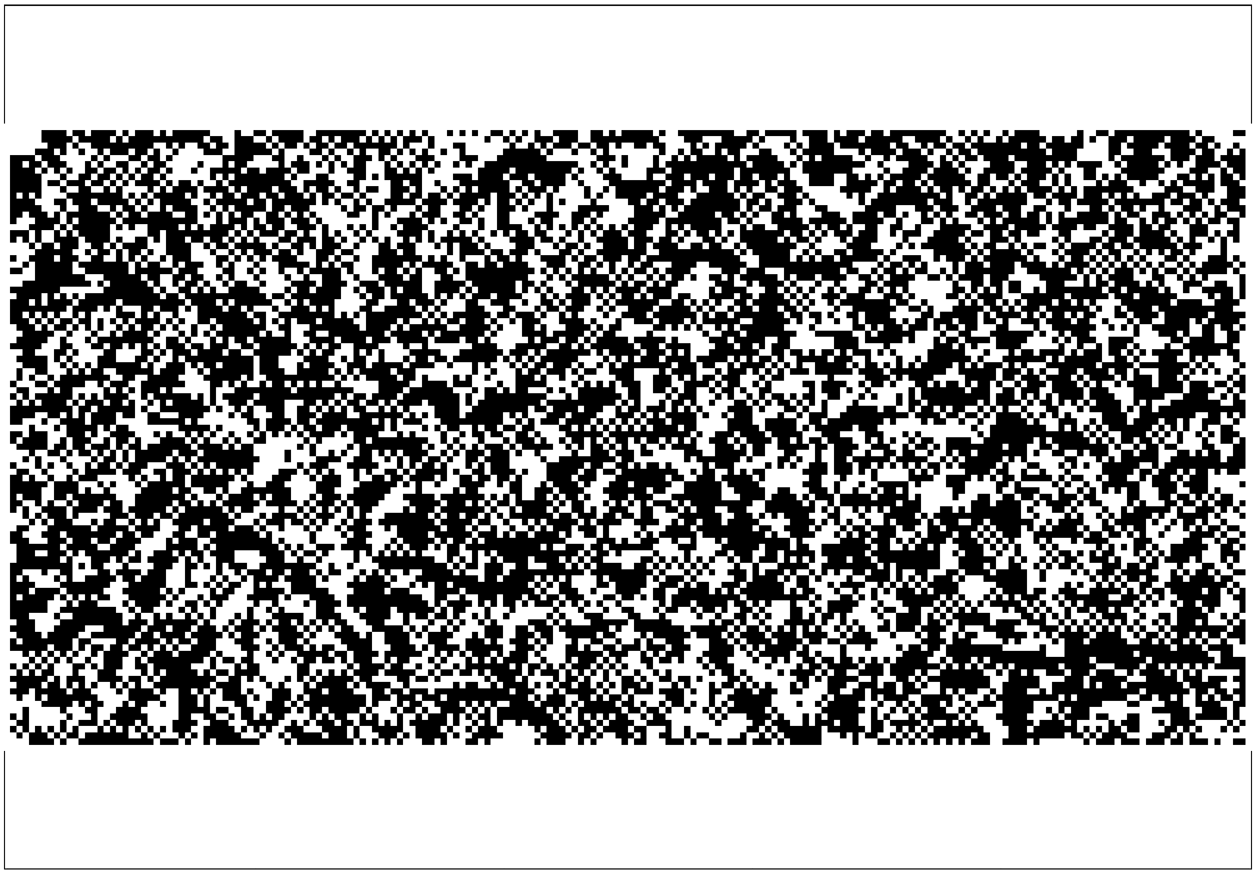
```
gen.rand.image <- function(dimX, dimY){
  rand.i   = sample(c(-1,+1),dimX*dimY ,replace = TRUE, prob = c(0.5,0.5))
  rand.image = matrix(data = rand.i, nrow = dimX, ncol = dimY)
  return(rand.image)
}
```

```
genNearestSamples = function(x, xCord, yCord){
  w = dim(x)[1] # width
  h = dim(x)[2] # height
  samples = c() # create empty vector
  if (xCord > 1) { samples = append(samples, x[xCord - 1, yCord]) } # if the point is not in the first
  if (yCord > 1) { samples = append(samples, x[xCord, yCord - 1]) } # if the point is not on the first
  if (xCord < w) { samples = append(samples, x[xCord + 1, yCord]) } # if the point is not on the last c
  if (yCord < h) { samples = append(samples, x[xCord, yCord + 1]) } # if the point is not on the last r
  return(samples) # return the samples (neighbors)
}

uIsingPrior = function(x, xi, xc, yc ,alpha,beta){
  uXi = - (alpha * (xi)) - ((beta * (xi)) * (sum(genNearestSamples(x=x, xCord = xc, yCord = yc)) ))
  return(uXi)
}
```

Ans.1.a) The ising prior Image Samples:

```r
gen.ising.image <- function(img,a,b){
  newImg = matrix(1,nrow = nrow(img), ncol = ncol(img))
  for(xcord in 2:(nrow(img)-1)){
    for(ycord in 2:(ncol(img)-1)){
      uPlusOne  = uIsingPrior(x = img, xi = +1, xc = xcord, yc = ycord, alpha = a, beta = b)
      uMinusOne = uIsingPrior(x = img, xi = -1, xc = xcord, yc = ycord, alpha = a, beta = b)
      pOfPlusOne = exp(-uPlusOne)/(exp(-uPlusOne) + exp(-uMinusOne))
      pOfMinusOne = 1 - pOfPlusOne
      newImg[[xcord,ycord]] = sample(c(+1,-1),1,replace = TRUE, prob = c(pOfPlusOne, pOfMinusOne ))
    }
  }
  return(newImg)
}
```

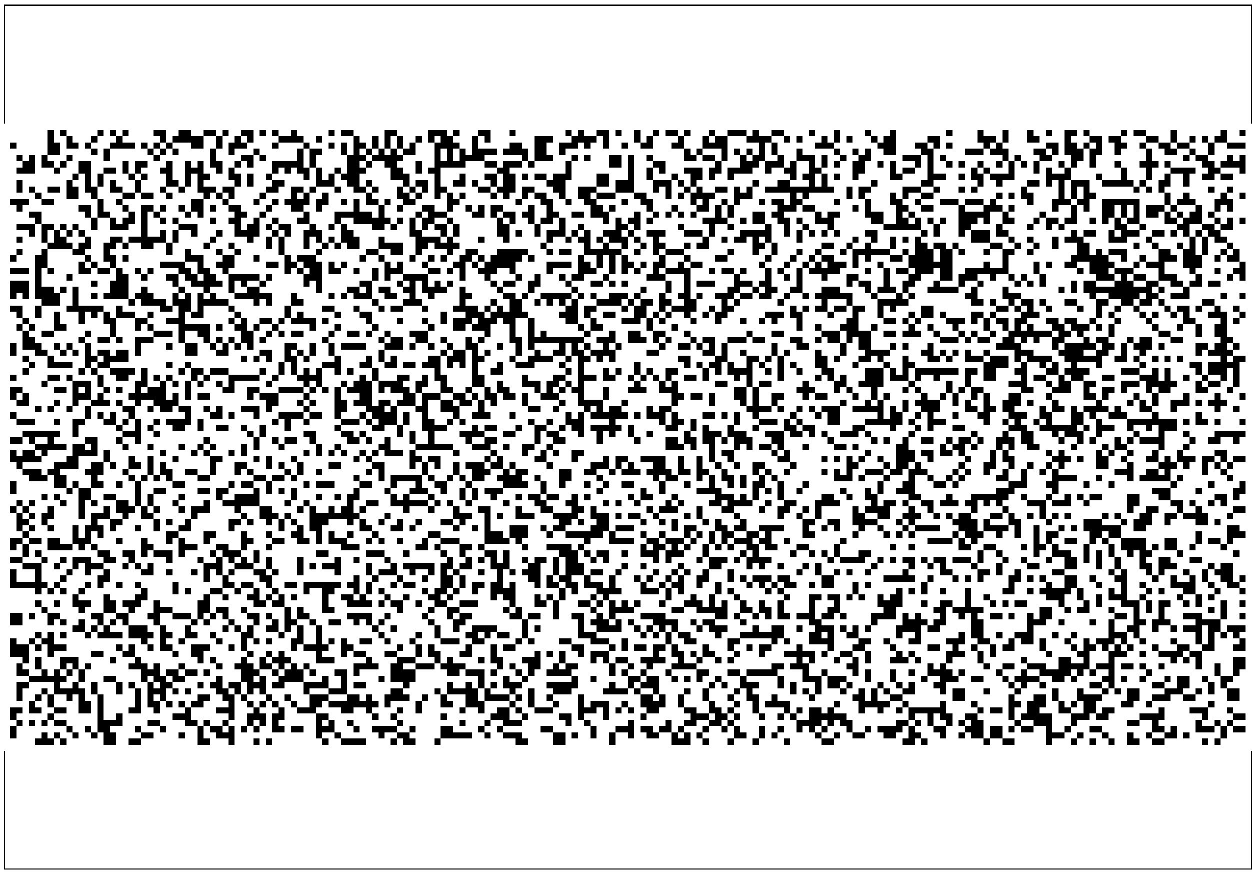
```r
g = gen.rand.image(200,100)

alpha = -0.5
beta = 1
g = gen.ising.image(img = g, a= alpha, b = beta)
display.image(g)
```



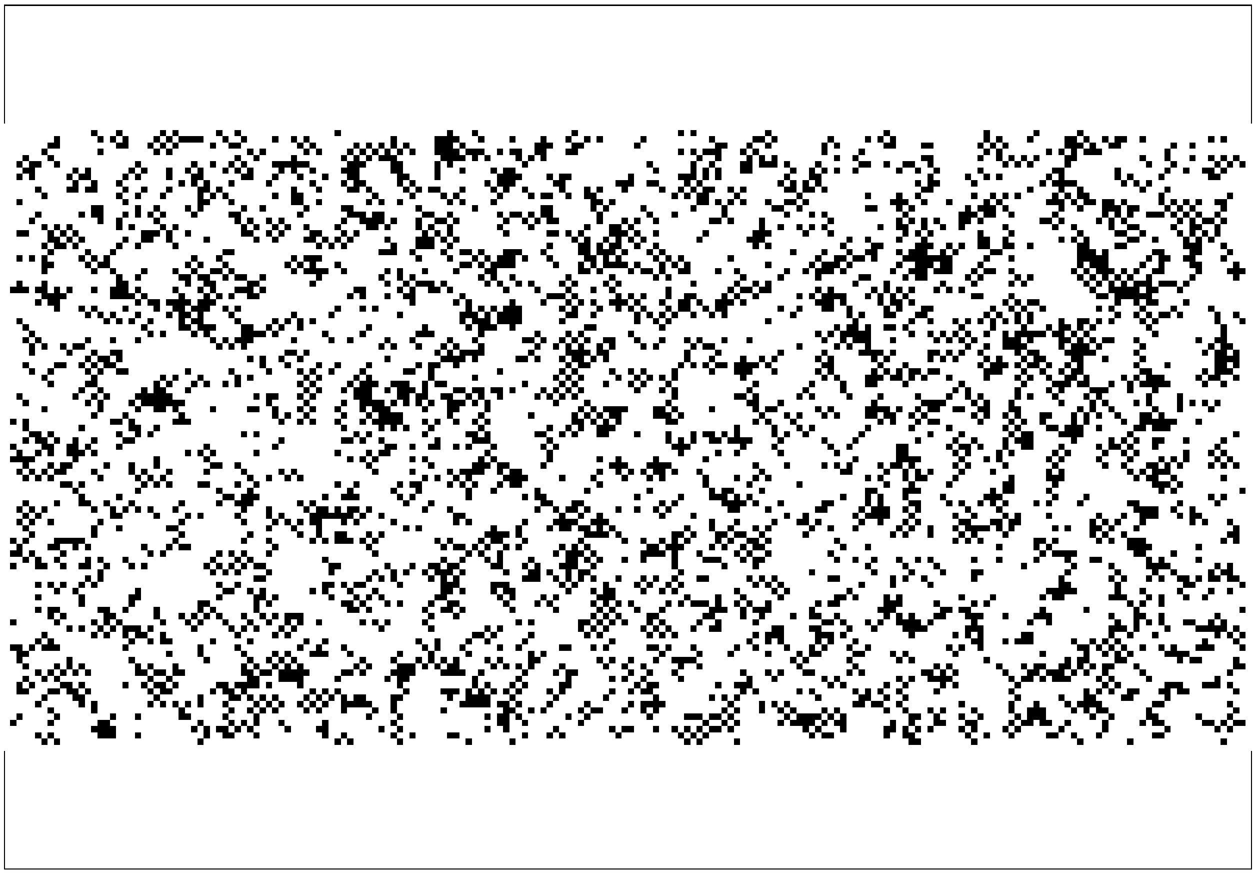
```r
alpha = -0.5
beta = 0.1
g = gen.ising.image(img = g, a= alpha, b = beta)
```

```
display.image(g)
```

```
alpha = 0.5
beta = 0.1
g = gen.ising.image(img = g, a= alpha, b = beta)
display.image(g)
```



```
alpha = 0.5
beta = 1
g = gen.ising.image(img = g, a= alpha, b = beta)
display.image(g)
```

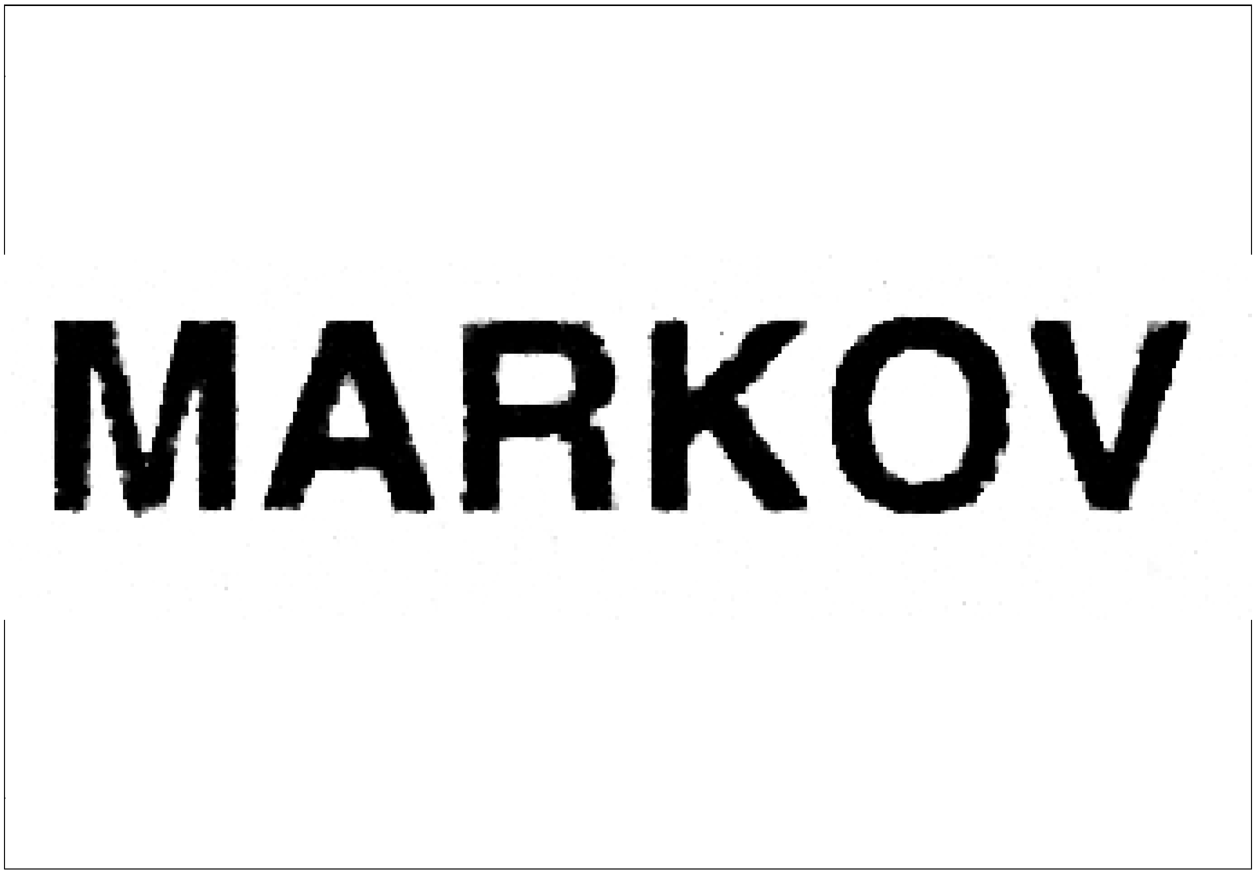


Ans.1.b)

```r
genNeighborMatrix = function(x){
  w = dim(x)[1]
  h = dim(x)[2]
  xUp = shift.up(x)
  xUp[w,] = x[1,]
  xDown = shift.down(x)
  xDown[1,] = x[w,]
  xLeft = shift.left(x)
  xLeft[,h] = x[,1]
  xRight = shift.right(x)
  xRight[,1] = x[,h]
  N = xUp + xDown + xLeft + xRight
  return(N)
}

uIsingPosterior = function(x, xi,alpha,beta,sigma,yi){
  uXi = - (alpha * (xi)) - (beta * (xi) * genNeighborMatrix(x) ) + ((1/(2*sigma)) * ((xi) - (yi))^2)
  return(uXi)
}

gen.posterior.image <- function(img, a, b, s, trueimg, m, n){
  uPlusOne  = uIsingPosterior(x = img, xi = +1, alpha = a, beta = b, sigma =s, yi=trueimg )
  uMinusOne = uIsingPosterior(x = img, xi = -1, alpha = a, beta = b, sigma =s, yi=trueimg)
  pOfPlusOne = exp(-uPlusOne )/(exp(-uPlusOne ) + exp(-uMinusOne))
  pOfMinusOne = 1 - pOfPlusOne
  uRandom = matrix(runif(m*n), m)
  newImg = 2 * (uRandom < pOfPlusOne) - 1
  return(newImg)
}
```

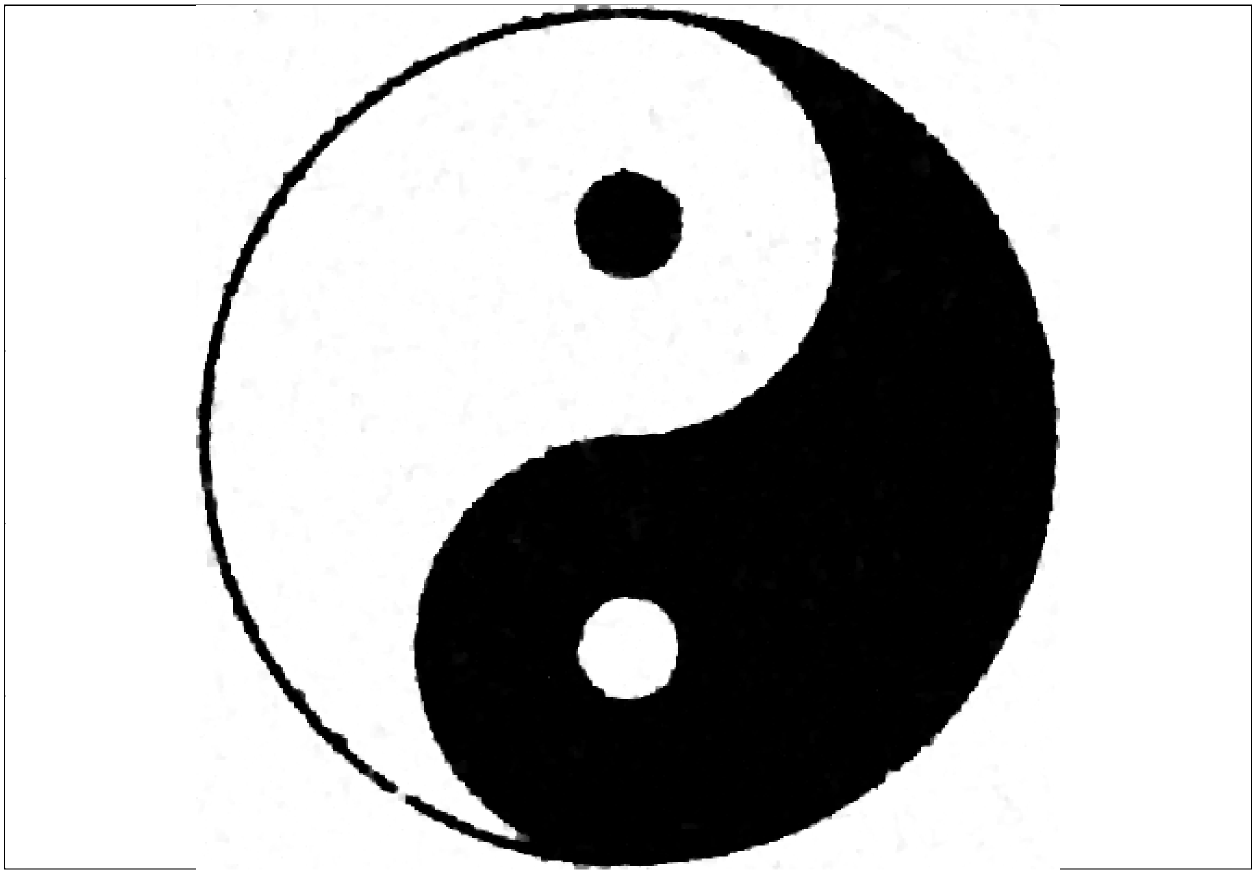
Values of $\alpha = 0.1, \beta = 0.95, \sigma = 1.5$ for “noisy-message.png”

```r
input = read.image("./noisy-message.png")
m=345
n=100

randImg = gen.rand.image(m,n)

alpha = 0.1
beta = 0.95
sigma= 1.19
imgList = list()

samples = 300
burnIn= 10
vecEstimate = numeric()
sRandImg = matrix(0, nrow=m, ncol= n)
for (i in 1:samples){
  imgList[[i]] = gen.posterior.image(img = randImg, a= alpha, b = beta, s=sigma, trueimg=input, m, n)
  randImg = imgList[[i]]
  if (i >= burnIn ){
    sRandImg = sRandImg + randImg
  }
  sigmaEstimate = sqrt(sum((randImg - input)^2)/(m*n))
  vecEstimate = c(vecEstimate, sigmaEstimate)
}

display.image(sRandImg/(samples-burnIn))
```



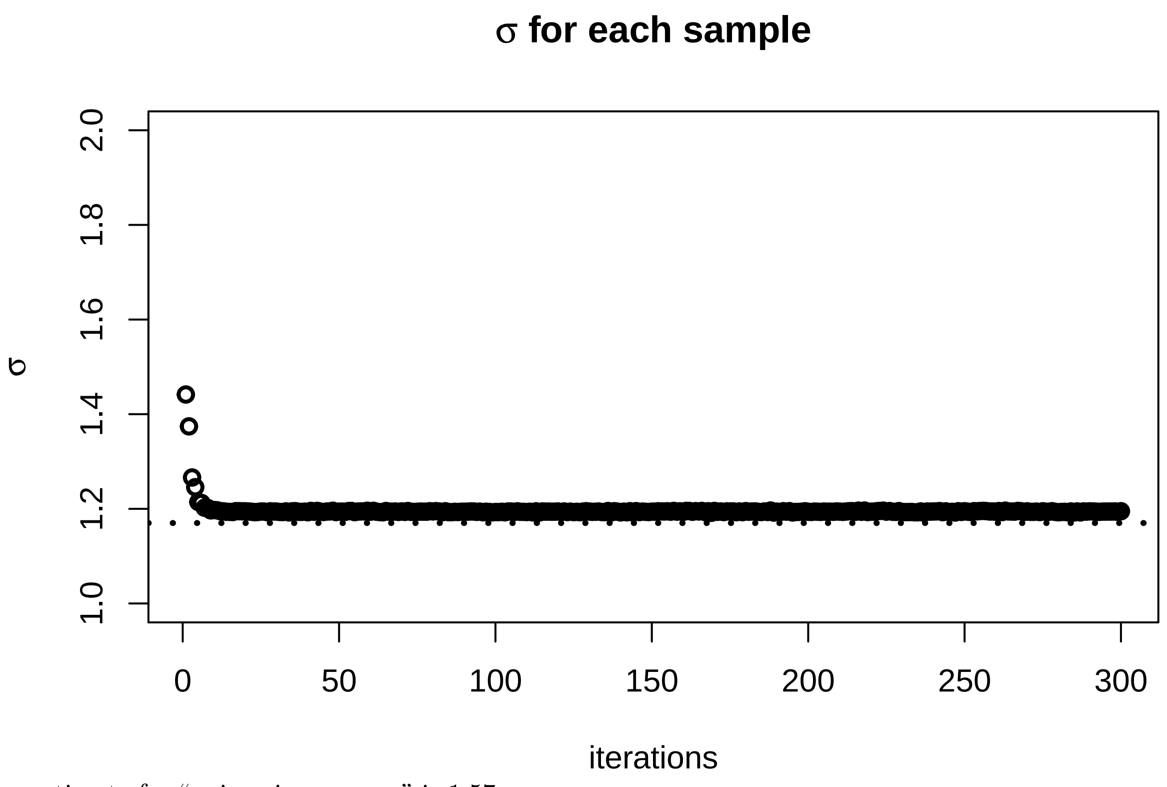
Values of $\alpha = -0.05, \beta = 1.2, \sigma = 1.5$ for "noisy-yinyang.png"

```r
input = read.image("./noisy-yinyang.png")
m=466
n=466
randImg = gen.rand.image(m,n)

alpha = -0.05
beta = 1.2
sigma= 1.59
imgList = list()

samples2 = 500
burnIn = 10
vecEstimate2 = numeric()
sRandImg = matrix(0, nrow=m, ncol= n)
for (i in 1:samples2){
  imgList[[i]] = gen.posterior.image(img = randImg, a= alpha, b = beta, s=sigma, trueimg=input, m, n)
  randImg = imgList[[i]]
  if (i >= burnIn ){
    sRandImg = sRandImg + randImg
  }
  sigmaEstimate = sqrt(sum((randImg - input)^2)/(m*n))
  vecEstimate2 = c(vecEstimate2, sigmaEstimate)
}
display.image(sRandImg/(samples2 - burnIn))
```



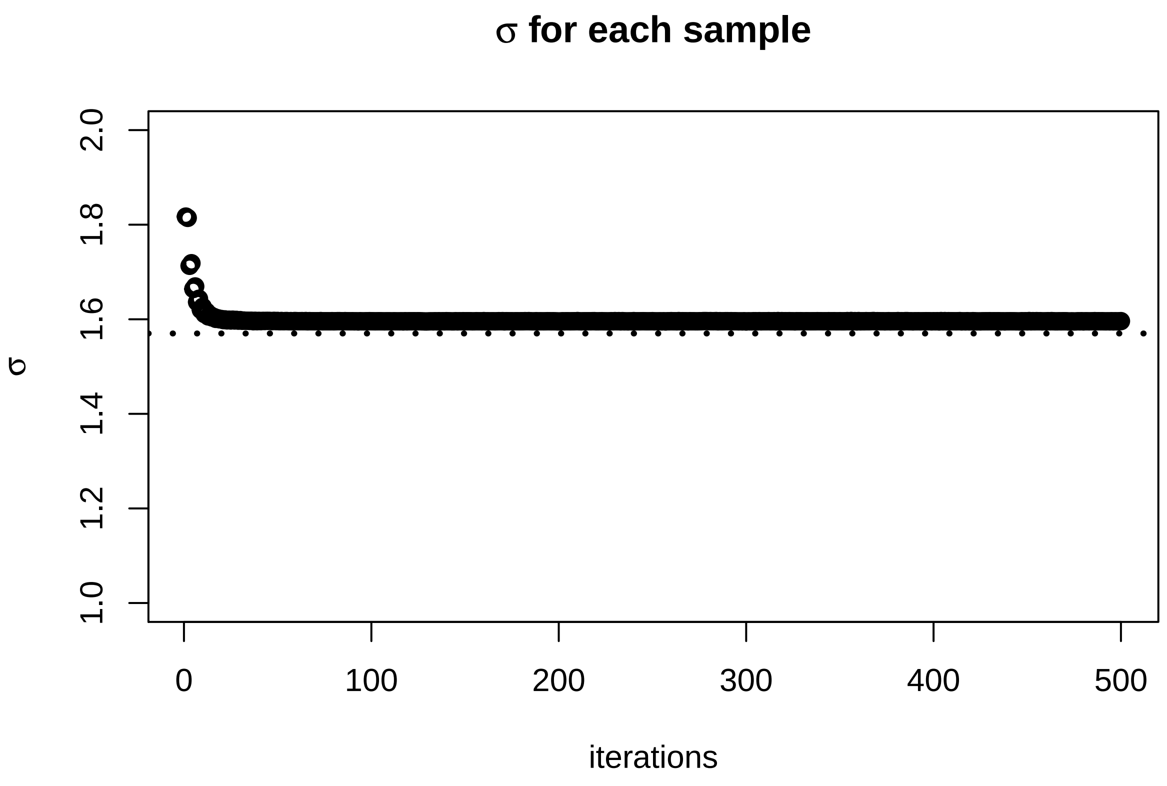
Ans.1.c) σ estimate for “noisy-message.png” $= 1.2$

```
plot(1:samples, vecEstimate, lwd=2, main=expression(bold(paste(sigma, " for each sample"))), ylab=expre
abline(h=1.17,col='black',lwd=3,lty=3)
```



σ estimate for "noisy-yinyang.png" is 1.57

```
plot(1:samples2, vecEstimate2, lwd=2, main=expression(bold(paste(sigma, " for each sample"))), ylab=exp
abline(h=1.57,col='black',lwd=3,lty=3)
```



Ans 2.a) For Un-normalized Posterior of

$$p(\beta|y;x,\sigma) \propto \prod_{i=1}^{n} p(y_i|\beta;x_i)p(\beta;\sigma)$$

y distribution is bernoulli distribution, therefore:

$$p(y_i|\beta;x_i) = \left(\frac{1}{1+\exp^{-x_i^T\beta}}\right)^{y_i}\left(1-\frac{1}{1+\exp^{-x_i^T\beta}}\right)^{1-y_i}$$

$$p(\beta;\sigma) \propto \exp^{-\frac{(\|\beta\|)^2}{2\sigma^2}}$$

Therefore, the posterior distribution of $P(\beta)$ with prior as gaussian is given below:

$$p(\beta|y;x,\sigma) \propto \left\{\prod_{i=1}^{n}\left(\frac{1}{1+\exp^{-x_i^T\beta}}\right)^{y_i}\left(1-\frac{1}{1+\exp^{-x_i^T\beta}}\right)^{1-y_i}\right\}\exp^{-\frac{(\|\beta\|)^2}{2\sigma^2}}$$

Ans.2.b)

$$p(\beta|y;x,\sigma) \propto \exp^{-U(\beta)}$$

$$\exp^{-U(\beta)} \propto \left\{\prod_{i=1}^{n}\left(\frac{1}{1+\exp^{-x_i^T\beta}}\right)^{y_i}\left(1-\frac{1}{1+\exp^{-x_i^T\beta}}\right)^{1-y_i}\right\}\exp^{-\frac{(\|\beta\|)^2}{2\sigma^2}}$$

Taking log on both sides:

$$-U(\beta) = \left\{\sum_{i=1}^{n} y_i \log\frac{1}{1+\exp^{-x_i^T\beta}} + (1-y_i)\log(1-\frac{1}{1+\exp^{-x_i^T\beta}})\right\} - \frac{(\|\beta\|)^2}{2\sigma^2}$$

$$-U(\beta) = \left\{\sum_{i=1}^{n} y_i \log\frac{1}{1+\exp^{-x_i^T\beta}} + (1-y_i)\log\frac{\exp^{-x_i^T\beta}}{1+\exp^{-x_i^T\beta}})\right\} - \frac{(\|\beta\|)^2}{2\sigma^2}$$

$$-U(\beta) = \left\{\sum_{i=1}^{n} -y_i\log(1+\exp^{-x_i^T\beta}) + (1-y_i)\log(\exp^{-x_i^T\beta}) - (1-y_i)\log(1+\exp^{-x_i^T\beta})\right\} - \frac{(\|\beta\|)^2}{2\sigma^2}$$

$$-U(\beta) = \left\{\sum_{i=1}^{n} -y_i\log(1+\exp^{-x_i^T\beta}) - (1-y_i)x_i^T\beta - (1-y_i)\log(1+\exp^{-x_i^T\beta})\right\} - \frac{(\|\beta\|)^2}{2\sigma^2}$$

Solving this equation :

$$U(\beta) = \sum_{i=1}^{n}(1-y_i)x_i^T\beta + \log(1+\exp^{-x_i^T\beta}) + \frac{(\|\beta\|)^2}{2\sigma^2}$$

Hence Proved.

Ans.2.c)

The implementation of the Hamiltonian Monte Carlo sampling can be seen in hmc.r. The label for versicolor is chosen as 0 and virginica is chosen as 1.

Ans.2.d)

```r
source("~/Documents/Probabilistic Modelling/hw3/temp/hmc.r");

## X
data.versicolor = as.data.frame.matrix(subset(iris, iris$Species==c("versicolor")));
data.virginica = as.data.frame.matrix(subset(iris, iris$Species==c("virginica")));
data.training <- rbind(data.versicolor[1:30,], data.virginica[1:30,]);
data.test <- rbind(data.versicolor[31:50,], data.virginica[31:50,]);

numtrainings <- nrow(data.training);
numtests <- nrow(data.test);

## Y
label.training = 1:numtrainings;
label.training[which(data.training$Species == "versicolor")] = 0;
label.training[which(data.training$Species == "virginica")] = 1;
label.test = 1:numtests;
label.test[which(data.test$Species == "versicolor")] = 0;
label.test[which(data.test$Species == "virginica")] = 1;

data.training = data.training[-grep('Species',colnames(data.training))];
data.training = cbind(data.training, matrix(1,numtrainings,1));
data.training = as.matrix(data.training);

data.test = data.test[-grep('Species', colnames(data.test))];
data.test = cbind(data.test, matrix(1,numtests,1));
data.test = as.matrix(data.test);

L = 30;
numBurnin = 50;
dim = 5;
epsilon = 0.05;
sigma = 1.0;
numSamples = 1000;

current_beta = matrix(c(-1,0,-1,0,-1),1,5)

beta = current_beta
sampleBeta = matrix(rep(0,5),numSamples,5)

for(i in 1:numBurnin){
  beta = HMC(data.training, label.training, epsilon, L, beta, sigma)
}

rejectCount = 0
sampleBeta[1,] = t(beta)
for(i in 1:(numSamples-1)){
  beta =  HMC(data.training, label.training, epsilon, L, t(data.matrix(sampleBeta[1,])), sigma)
  sampleBeta[i+1,] = t(beta)

  if(all(sampleBeta[i,] == sampleBeta[i+1,]))
    rejectCount = rejectCount + 1
}
```

```
cat(paste("Acceptance rate =",1-(rejectCount / numSamples), "\n"))
```

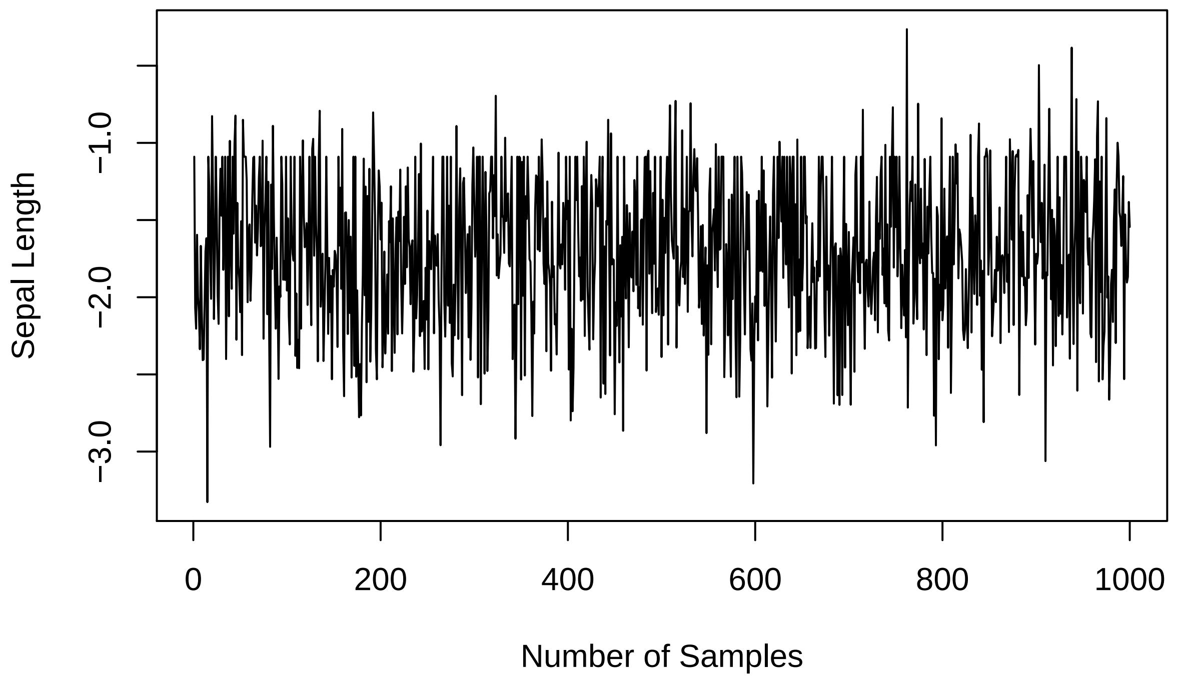
```
## Acceptance rate = 0.987
```

```
## Computing predictive posterior probability
y.predict = matrix(-1, numtests, 1);
for(i in 1:numtests){
  sum.predict = 0;
  for(j in 1:numSamples){
    beta = as.matrix(sampleBeta[j,]);
    x = data.test[i,];
    de = x%*%beta;
    pr = 1/(1+exp(-de));
    sum.predict = sum.predict + pr;
  }
  prob.predict = sum.predict / numSamples;
  if(prob.predict >= 0.5) {
    y.predict[i]=1;
  } else  {
    y.predict[i] = 0;
  }
}
```

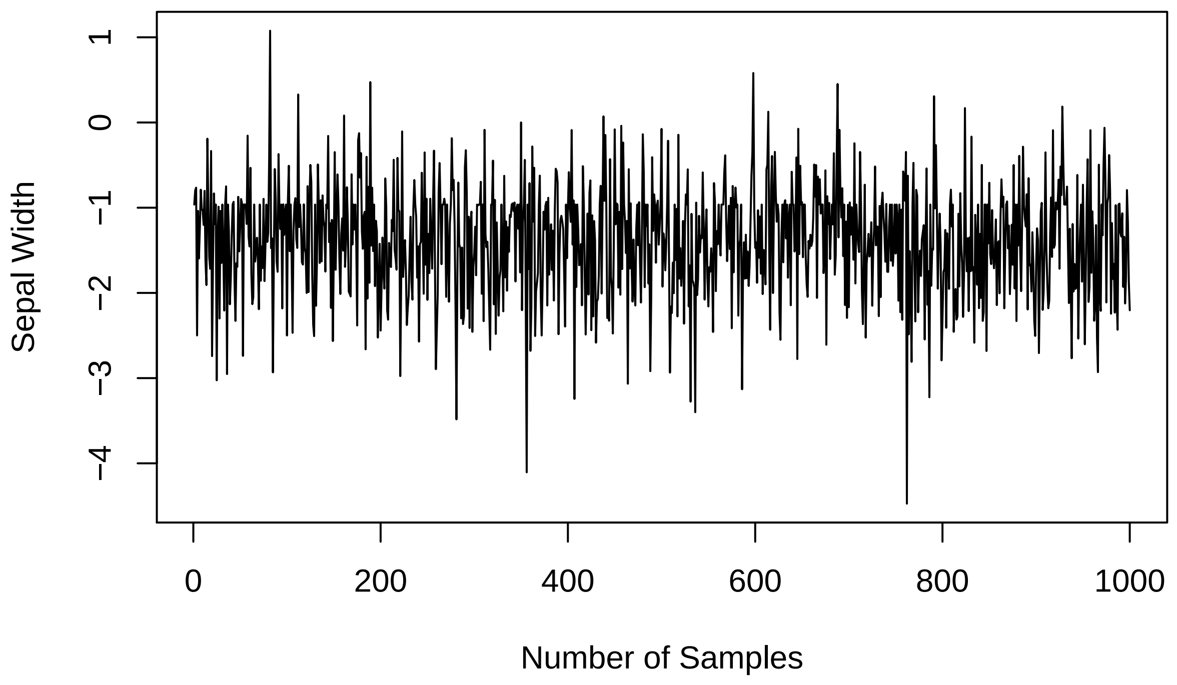
I sampled 1000 beta samples from the posterior, leaving 50 for burn-in and used 1000 for the testing stage. I got an acceptance rate about 99.1%. For each sampled beta, I find the probability for each data point that it has a certan label. If the avaerage probability is greater than 0.5 than that data point got the label 1 else 0. Using this method I predicted the $\hat{y}$ labels with accuracy of about 97.5%.

Ans.2.e)

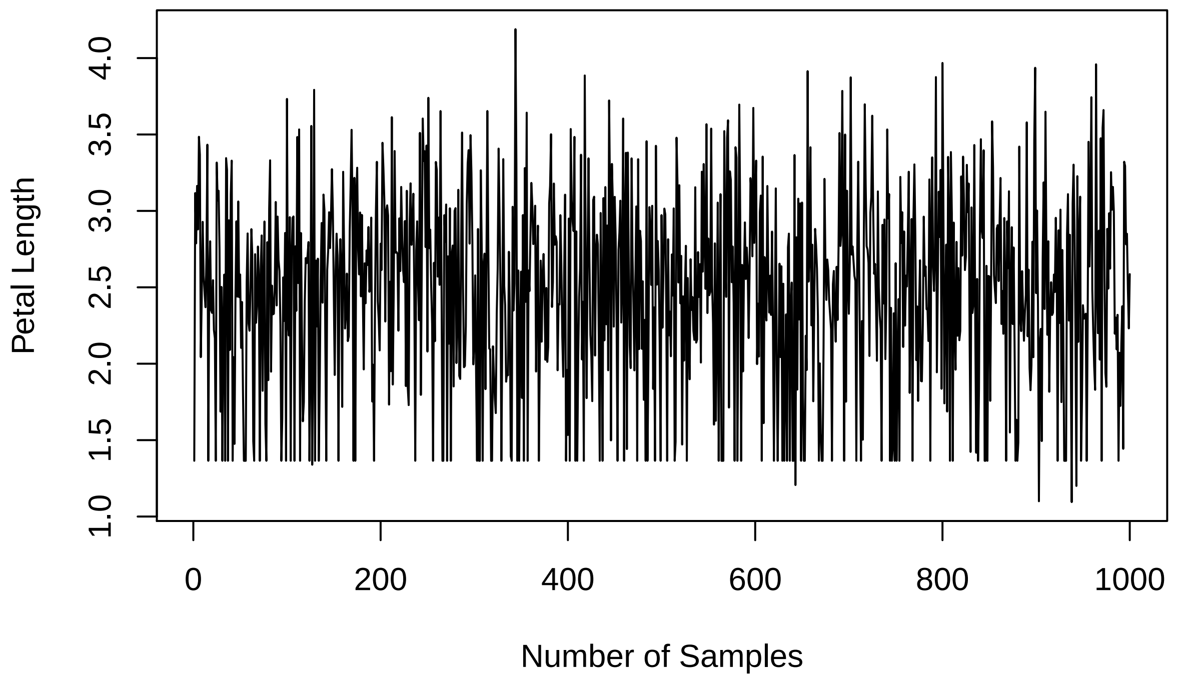
```
colname = c("Sepal Length","Sepal Width","Petal Length","Petal Width","Constant");

trace.plot = function(q,col_name){
  plot(1:length(q), q, type='l',xlab ="Number of Samples",ylab=col_name);
}

trace.plot(sampleBeta[,1],colname[1])
```



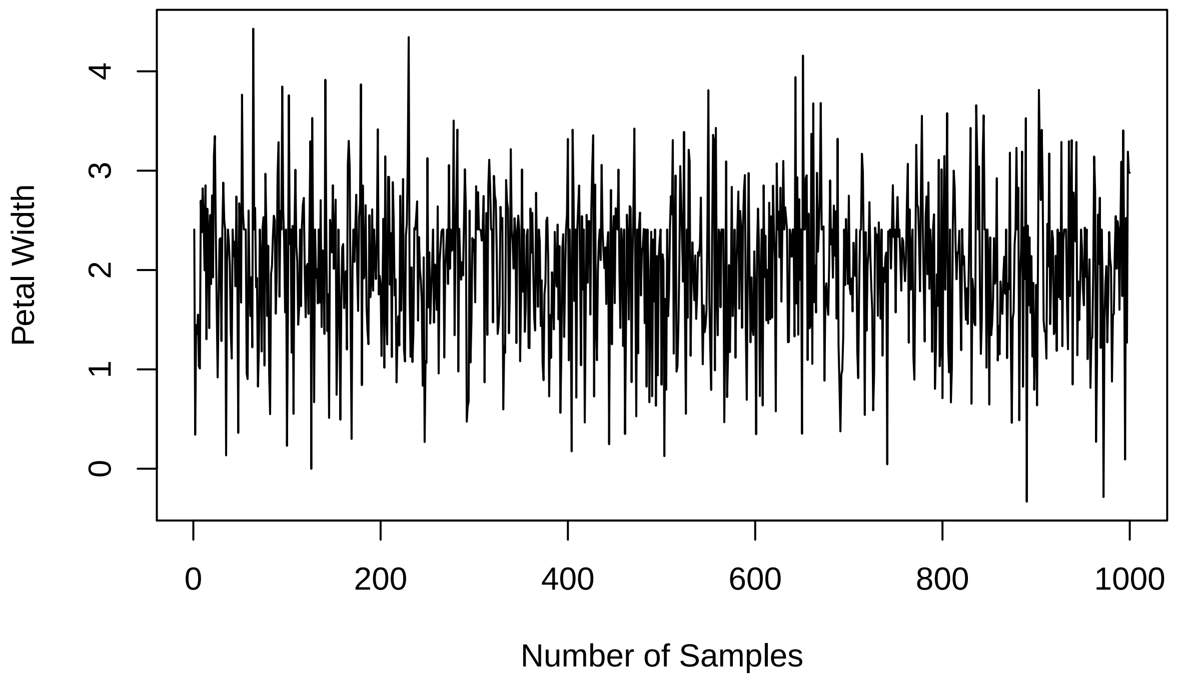
```
trace.plot(sampleBeta[,2],colname[2])
```



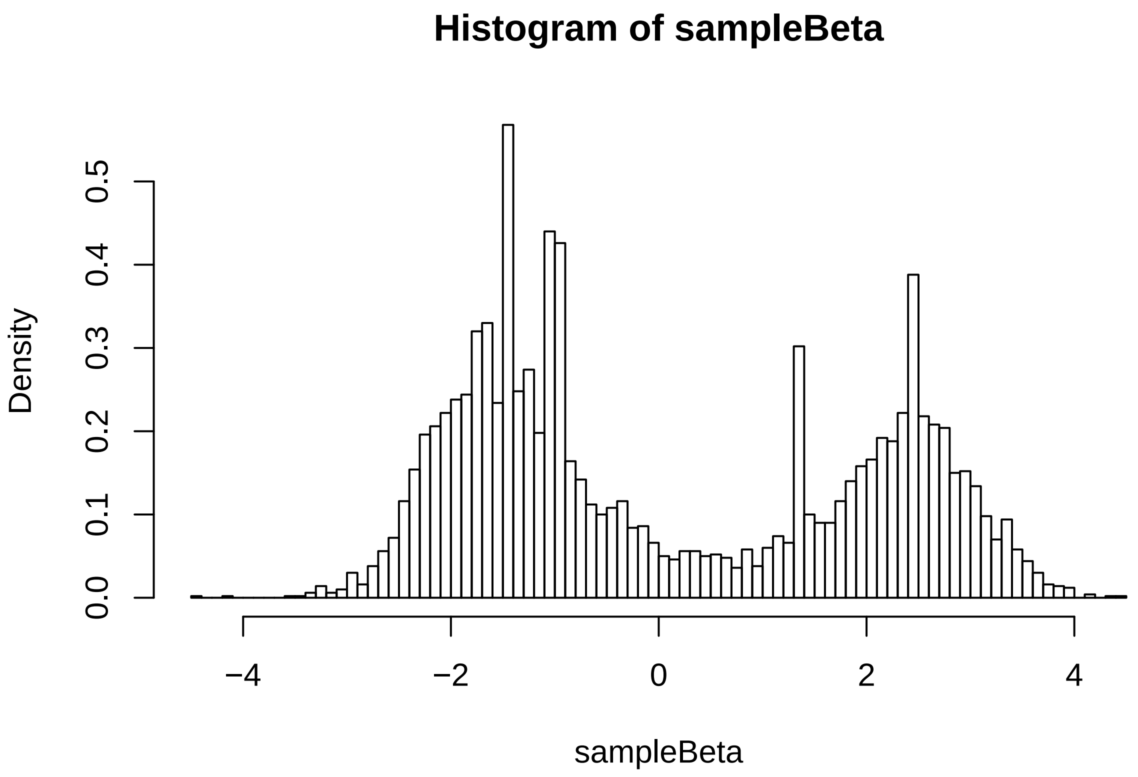
```
trace.plot(sampleBeta[,3],colname[3])
```



```r
trace.plot(sampleBeta[,4],colname[4])
```



```r
hist(sampleBeta, freq=FALSE, breaks=80)
```



Ans.2.f)

```
## average error rate
error = 0;
for(i in 1:numtests){
  if(y.predict[i] != label.test[i]){
    error = error + 1;
  }
}

cat(paste("Average error rate =", error/numtests, "\n"))
```

```
## Average error rate = 0.025
```

$\sigma = 1, \epsilon = 0.05$ and $L = 30$ As mention earlier I have got the acceptance rate of about 97.5%. Here I also calculated an average beta from the beta sample calculated earlier. Then I calculated the probability that each data point have a specific label. If that probability is greater than 0.5 I assign a category of 1 to that data point. Here, I have got the accuracy of predicting labels about 97.5%