Homework3

BenLarson

April 14, 2016

For the first problem we set up the a gibbs sampler using an Ising model prior and iid gaussian likelihood given a noisy image y. We use the equation:

$$U(x) = -\alpha \sum x_i - \beta \sum_{<i,j>} x_i x_j + \frac{1}{2\sigma}^2 \sum x_i - y_i^2$$

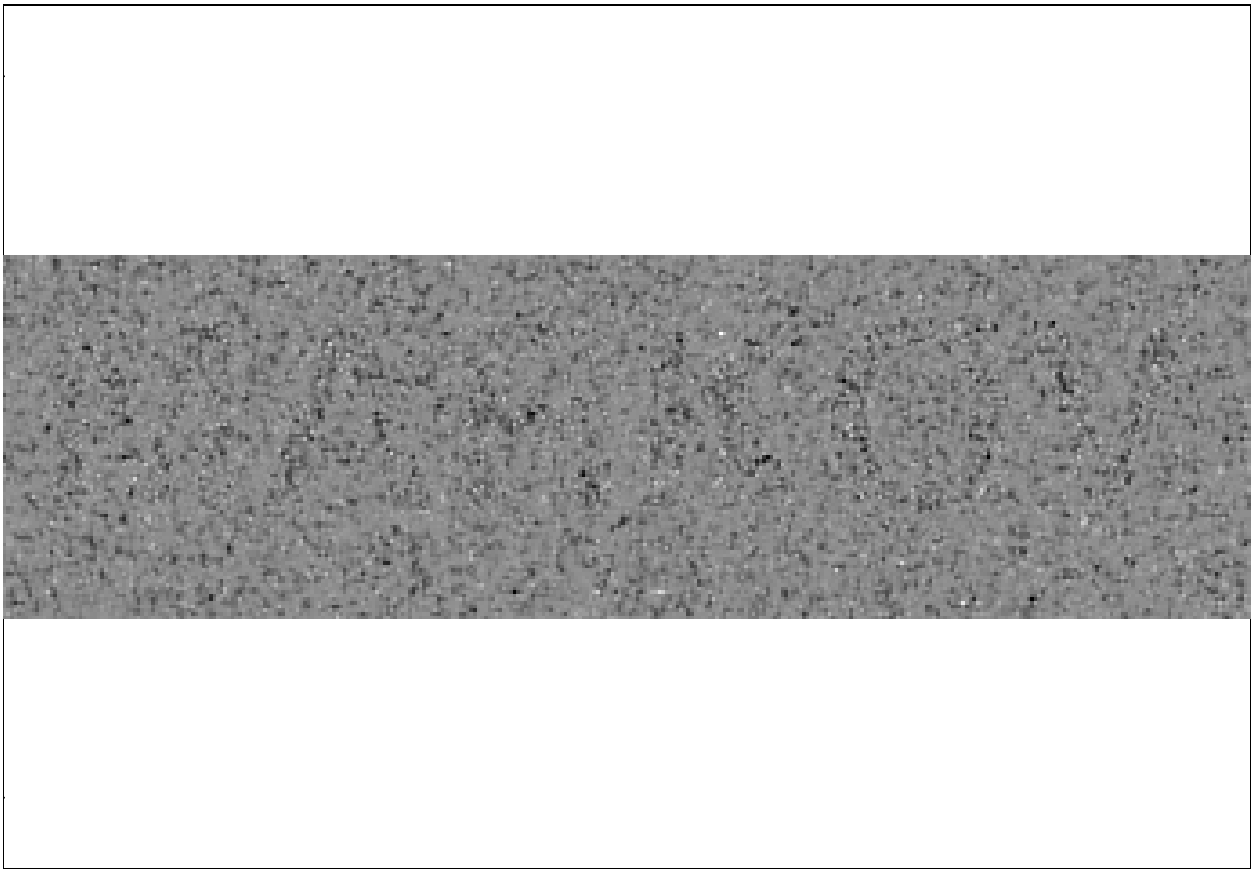
To denoise the image we first create a random image x. The next step is to iterate through this image as you would a markov chain and sample from the bernoulli distribution. The distribution that we sample from is given by finding the probability that the pixel is a 1 or -1 based on exp(-U(x)). Then we compare what the actual pixel value is and find the resulting value to assign to that value. The reason this model works is that it minimizes the energy between pixels that are unlike the given image y while considering the relation to its neighbors. The lowest energy model would then be the denoised image as we can see in the results of our images.

```
## Loading required package: png
```

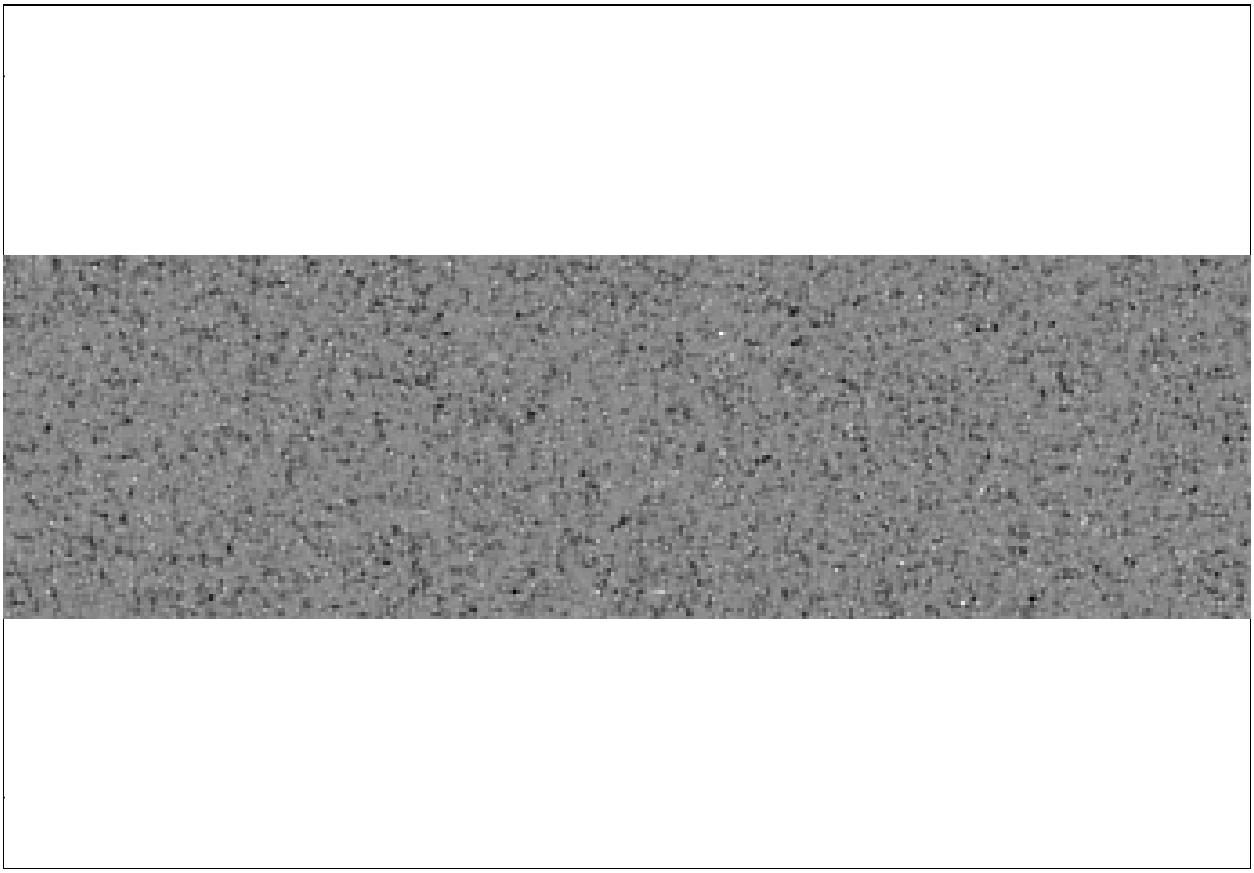
```
## Loading required package: knitr
```

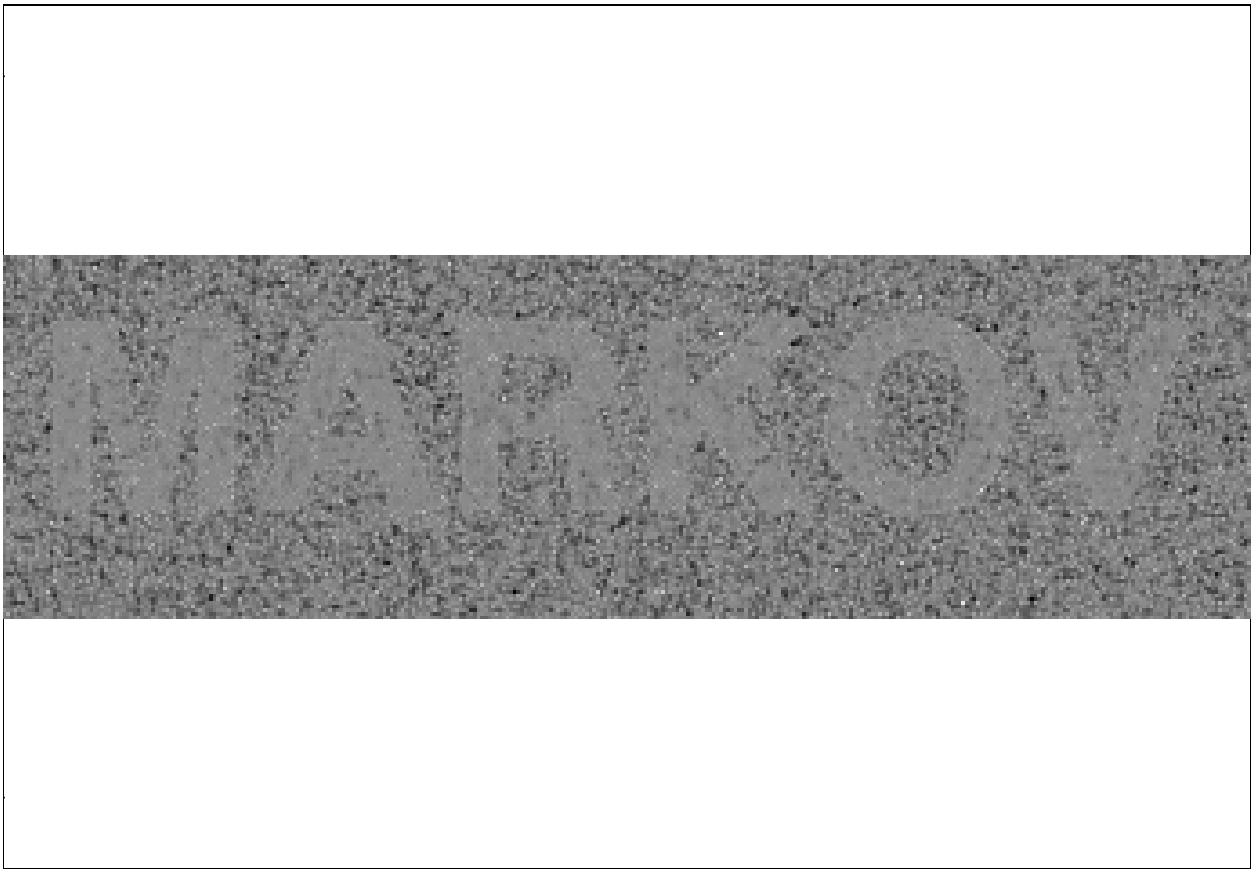
```
## [1] 0.2117647
## [1] 0.7882353
```


MARKOV




MARKOV



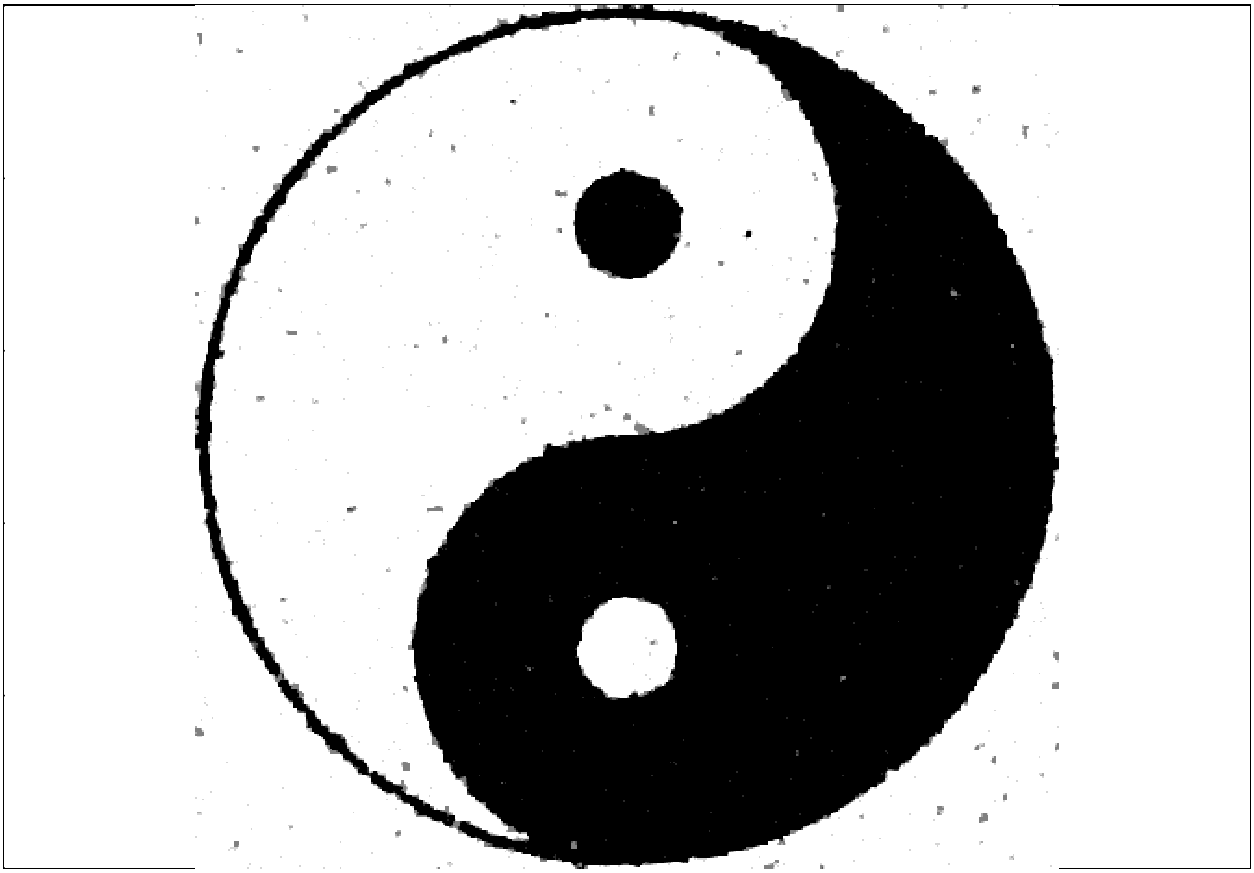


#1A
The above images are with no posterior. Just the alpha and beta terms for the markov image. The first image is the original.

```
## [1] 1.198078
```


```
## [1] 0.1176471
## [1] 0.9137255
```

```
## [1] 1.599562
```



1B

These are the results for the noisy images, Markov and Yingyang. The best alpha and beta I found were:

$$\alpha = -0.07, \beta = 1, \sigma = 0.2$$

1C

To estimate σ from each iteration I took an initial value and found a new image. This new image I subtracted from the noisy image y and then used the variance of this result to pass into the next iteration. The final value I found was ** $\sigma = 1.25$ **.

2A

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

$$\propto \prod_{i=1}^{n} \frac{1}{1+e^{-x_i^T\beta}}^{y} (1-e^{-x_i^T\beta})^{(1-y)} e^{\frac{1}{2*\sigma^2}\sum 0-\beta^2}$$

$$= log(\prod_{i=1}^{n} \frac{1}{1+e^{-x_i^T\beta}}^{y} (1-e^{-x_i^T\beta})^{(1-y)} e^{\frac{1}{2*\sigma^2}\sum 0-\beta^2})$$

for simplicity I will call $X = -x_i^T\beta$

$$= \sum [y_i log(\frac{1}{1+e^X}) + (1-y_i) log(1-\frac{1}{1+e^X})] - \frac{1}{2\sigma^2}(B^TB)^2$$

$$= \sum[-y_i log(1+e^X) + (1-y_i)X - log(1+e^X) + y_i log(1+e^X)] - \frac{1}{2\sigma^2}(B^T B)^2$$

$$= \sum[-(1-y_i)X - log(1+e^X)] - \frac{1}{2\sigma^2}(B^T B)^2$$

We now plug in X and realize that the log is on the left hand side of the equation to show that this is equivalent to $exp(-U(\beta))$.

$$log(U(\beta)) = \sum[(1-y_i)x_i^T\beta - log(1+e^{-x_i^T\beta})] - \frac{1}{2\sigma^2}(B^T B)^2$$

$$U(\beta) = exp\sum[(1-y_i)x_i^T\beta - log(1+e^{-x_i^T\beta})] - \frac{1}{2\sigma^2}(B^T B)^2$$

2C

To implement the code for the hamiltonian function from Radford Neal paper we need to define the function "U" and "grad_U". We have our U function from above. This takes as input the data x, y, and our vector beta. Beta can be a scalar or vector, but in our iris data set it was a vector. The U function will give us an potential energy in the hamiltonian equations. GradU will give us a position, or position vector in our case. To get gradU we take the derivate of U with respect to beta:

$$d/d\beta U(\beta) = d/d\beta exp\sum[(1-y_i)x_i^T\beta - log(1+e^{-x_i^T\beta})] - \frac{1}{2\sigma^2}(B^T B)^2$$

Intermediate steps:

$$d/d\beta(x_i^T\beta) = (x_i^T)^T = x_i$$

$$d/d\beta(y_i x_i^T\beta) = y_i(x_i^T)^T = y_i x_i$$

$$d/d\beta(log(x)) = 1/x * dx = \frac{1}{1+e^{-x_i^T\beta}} * e^{-x_i^T\beta} * (-x_i) = \frac{x_i^T e^{-x_i^T\beta}}{1+e^{-x_i^T\beta}}$$

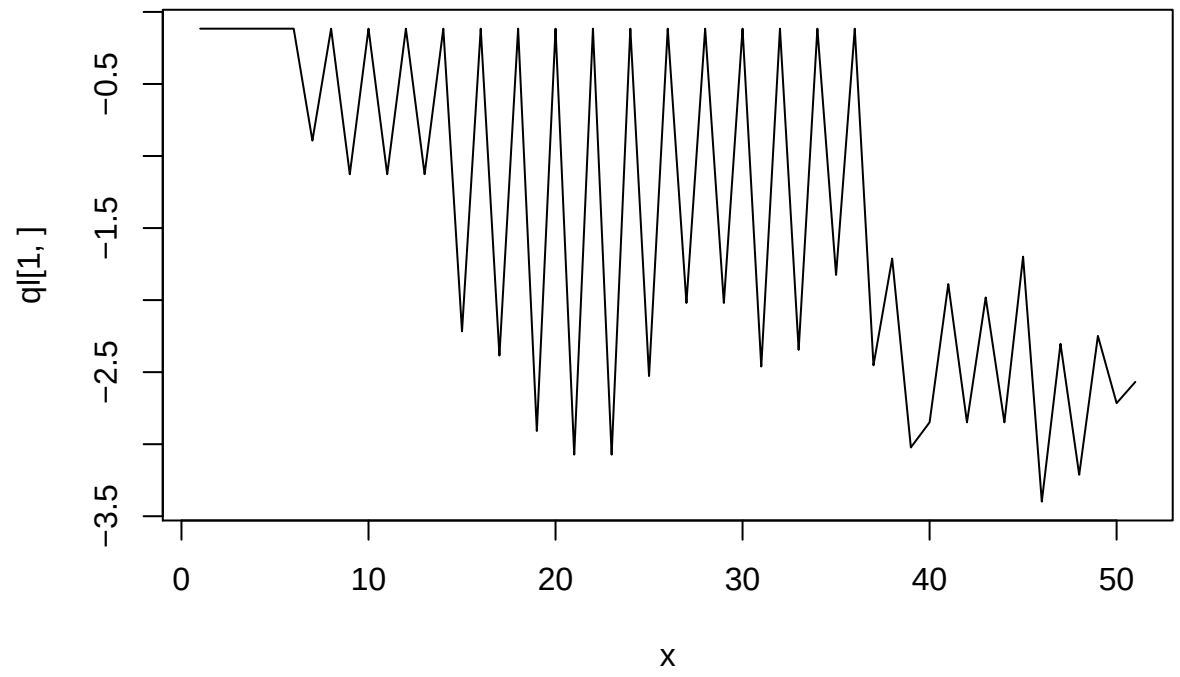
$$d/d\beta = \frac{1}{2\sigma^2}(B^T B)^2 = \frac{2\beta}{2\sigma^2} = \frac{\beta}{\sigma^2}$$

Finally together we get:

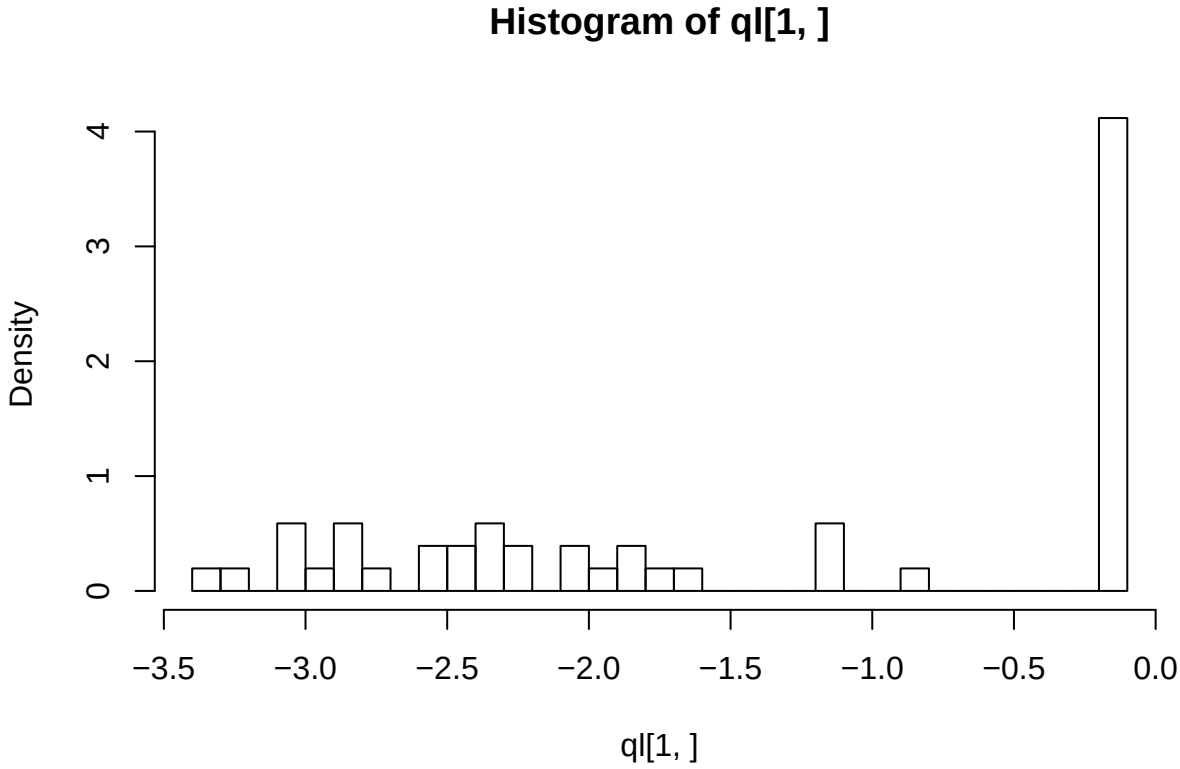
$$d/d\beta U(\beta) = \sum[x_i - y_i x_i^T - \frac{x_i^T e^{-x_i^T\beta}}{1+e^{-x_i^T\beta}}] - \frac{\beta}{\sigma^2}$$

```
## Acceptance rate = 1
```

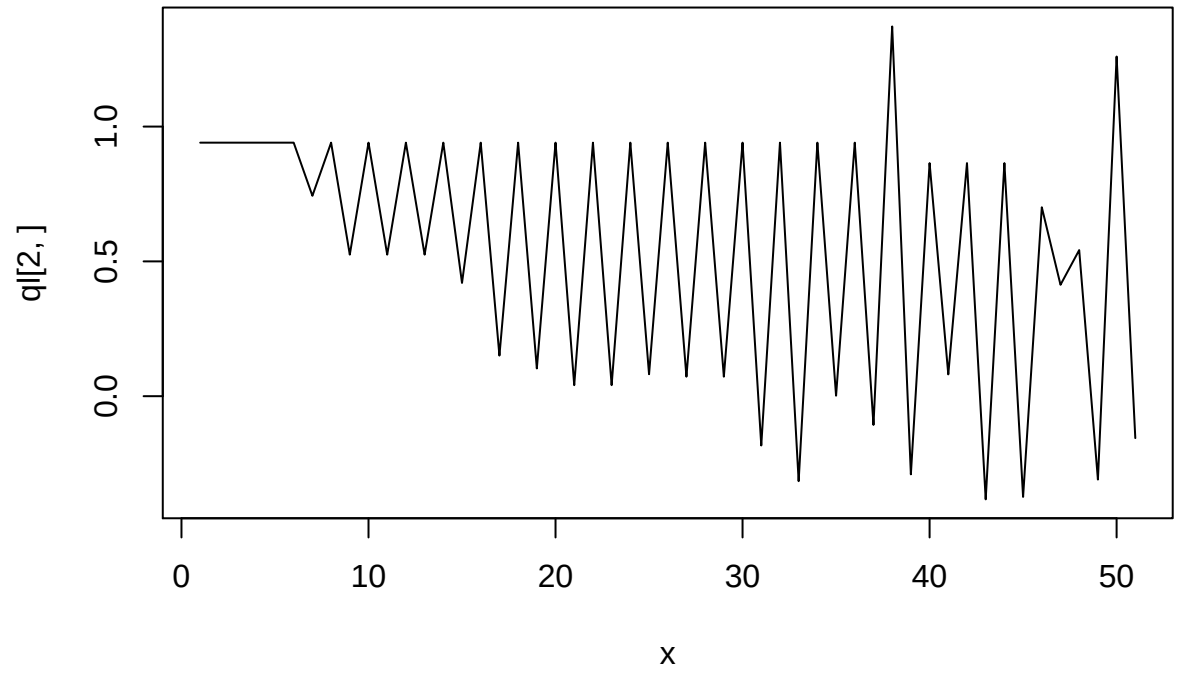
```
plot(x,ql[1,],type="l")
```



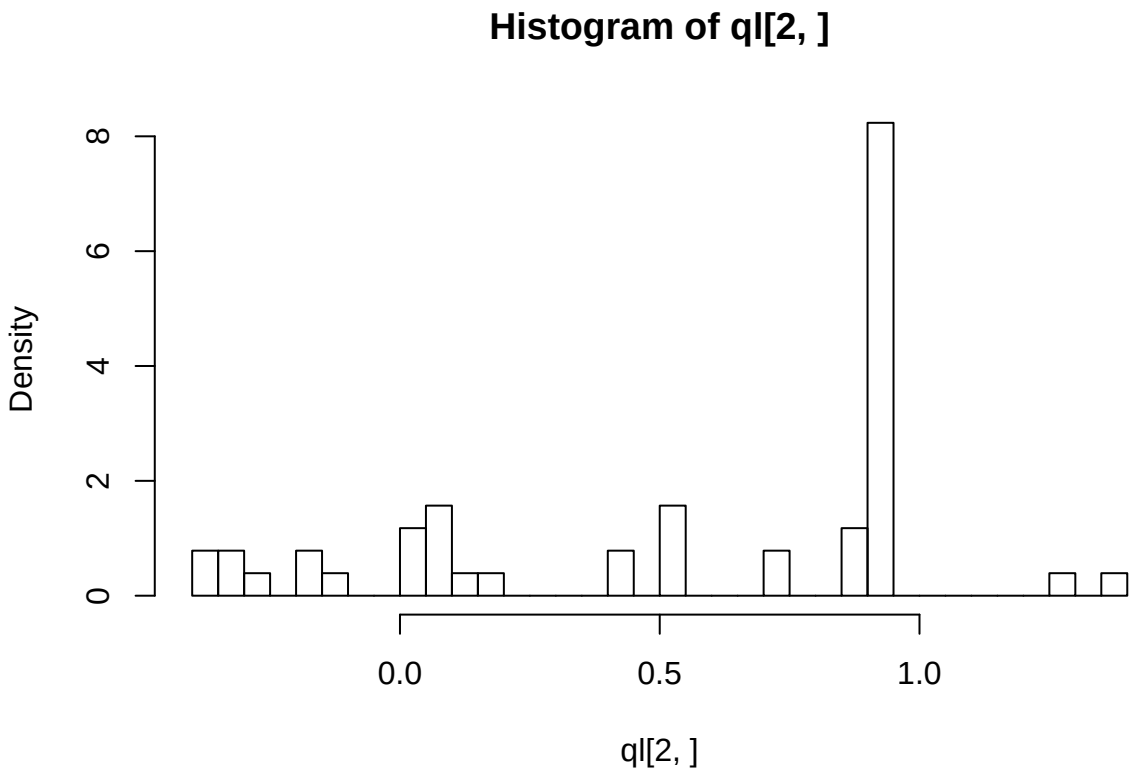

```
hist(ql[1,],freq=FALSE,breaks=40)
```



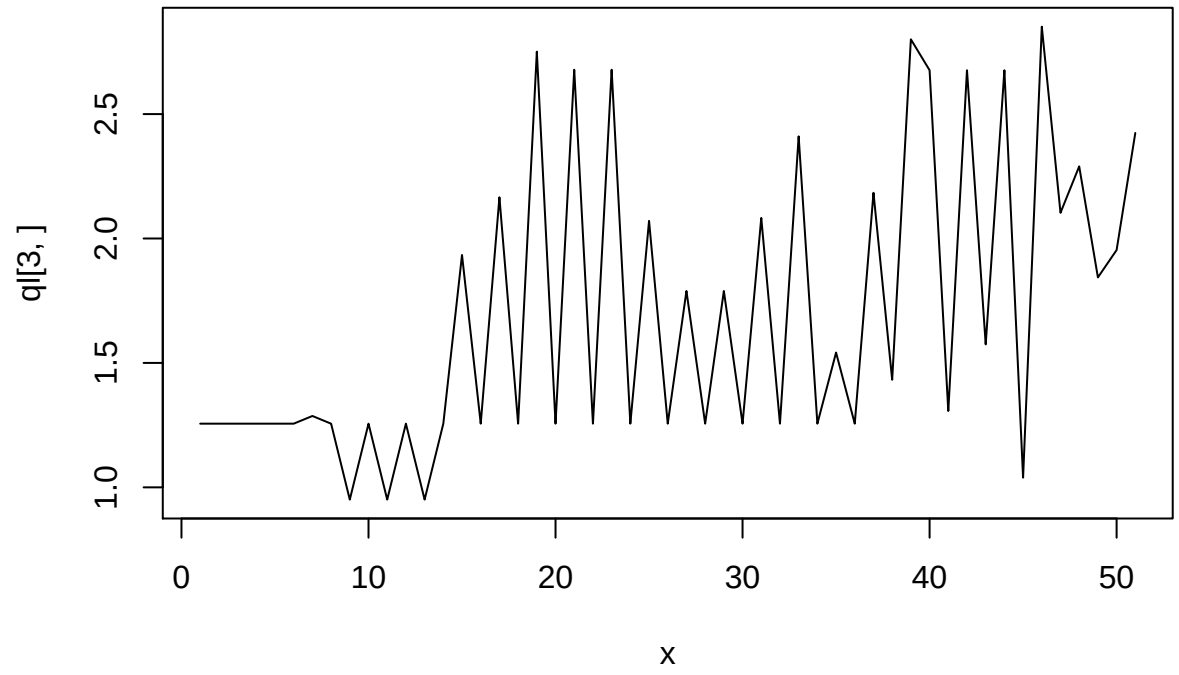
```
plot(x,ql[2,],type="l")
```



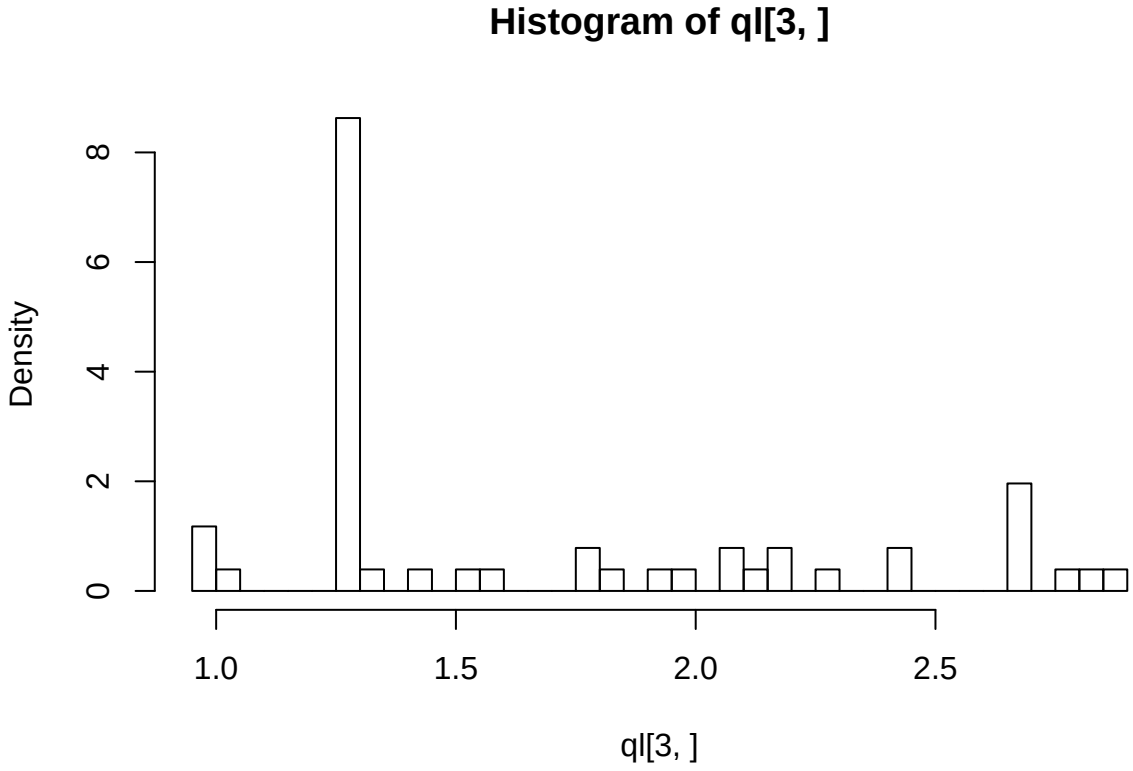

```
hist(ql[2,],freq=FALSE,breaks=40)
```



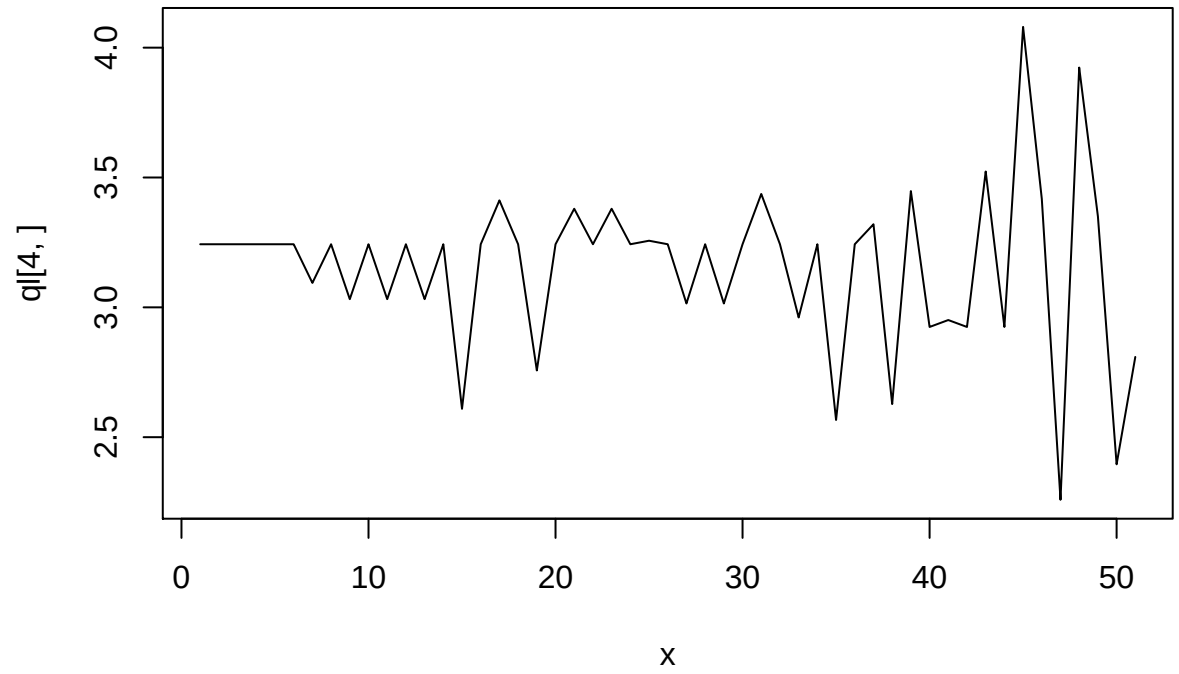

```
plot(x,ql[3,],type="l")
```



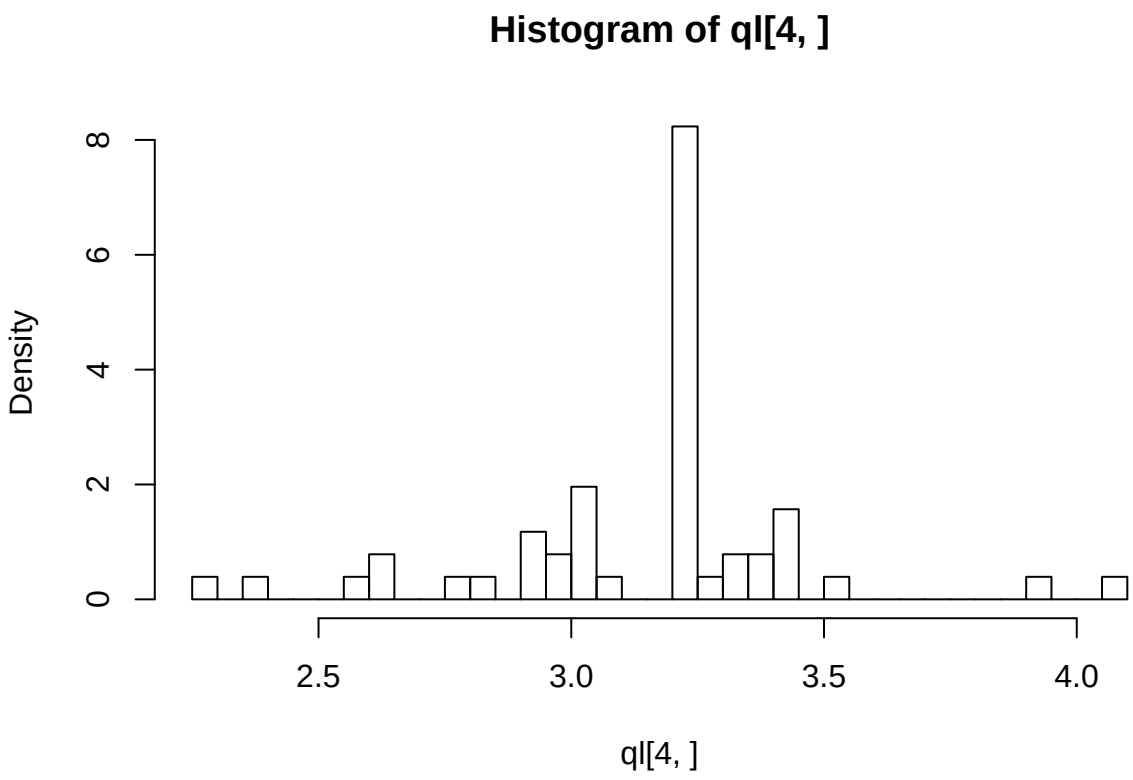

```
hist(ql[3,],freq=FALSE,breaks=40)
```



```r
plot(x,ql[4,],type="l")
```




```r
hist(ql[4,],freq=FALSE,breaks=40)
```

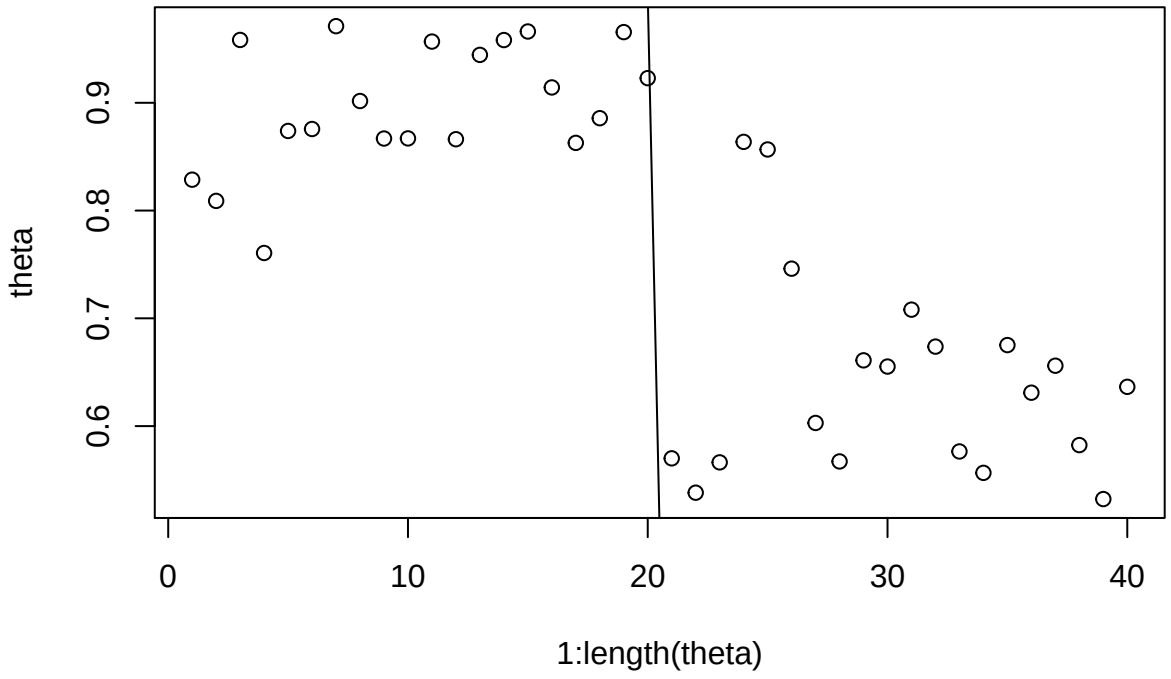



```r
#
resul = testsamples(ql)
```

```
correct = resul$correct
theta = resul$theta
theta = rowMeans(theta)
print('Monte Carlo estimate of p(~y|y)')
```

```
## [1] "Monte Carlo estimate of p(~y|y)"
```

```
plot(1:length(theta),theta)
lines(yt)
```




```
print(mean(theta))
```

```
## [1] 0.7702862
```

```
print("Zero-one Loss")
```

```
## [1] "Zero-one Loss"
```

```
print(correct)
```

```
## [1] 0.7711366
```

2: D,E, and F

IMPORTANT NOTE, 0 is the classifier for versicolor, 1 is the classifier for virignica. So in the Monte Carlo plot the first 20 indexes should be virinica, and 20:40 are versicolor. This is consitent with all my data.

Part D. We are asked to find the monte carlo estimate of the p(y~|y) for each testing data point. From the hamiltonian sampling I returned a vector of possible classification functions beta. Each row was a classifier for the entire data set. With each row lower in the beta matrix the classifier gets better and better as a result of more sampling from the hamiltonian code. I then saved each possible y~ for each beta, and averaged the probability for each y~. This gives us a final vector of how close our beta vectors came to assigning the right probability to the true data set.

Part E. Each trace plot is over each beta variable from the list of betas returned by the hamiltonian. I noticed that because the trace plots have a lot of movement up and down that is indicative of high acceptance rate.

Part F. Comparing the class labels I found iteratively what the correct number of classificaions were for each beta value over all test data. So if it labeled correctly I gave it a 1, else I skipped. Then I divided by the total number of test data that I was applying this classifcation to to get the expected value. Choosing the parameters I found that high epsilon would give inf values. This is probably due to the step size of the hamiltonian creating huge values that were putting the position vectors out of a reasonable range. I didn't have time to really change simga a lot, and high values past a certain point didn't make a noticable difference I could understand.

$$\sigma = 1, \epsilon = 0.05, L = 100$$