Lab 6 (Bayesian Estimation and MCMC)

Shreya Rao sr3843

06/10/2021

Goals

This lab has two goals. The first goal is to use the **Accept-Reject** algorithm to simulate from a mixture of two normals. The second goal is to utilize Bayesian methods and the the famous **Markov Chain Monte Carlo** algorithm to estimate the mixture parameter δ.

Background: (Mixture)

A mixture distribution is the probability distribution of a random variable that is derived from a collection of other random variables (Wiki). In our case we consider a mixture of two normal distributions. Here we assume that our random variable is governed by the probability density $f(x)$, defined by

$$\begin{aligned} f(x) &= f(x; \mu_1, \sigma_1, \mu_2, \sigma_2, \delta) \\ &= \delta f_1(x; \mu_1, \sigma_1) + (1-\delta) f_2(x; \mu_2, \sigma_2) \\ &= \delta \frac{1}{\sqrt{2\pi\sigma_1^2}} \exp -\frac{1}{2\sigma_1^2}(x-\mu_1)^2 + (1-\delta)\frac{1}{\sqrt{2\pi\sigma_2^2}} \exp -\frac{1}{2\sigma_2^2}(x-\mu_2)^2, \end{aligned}$$

where $-\infty < x < \infty$ and the parameter space is defined by $-\infty < \mu_1, \mu_2 < \infty$, $\sigma_1, \sigma_2 > 0$, and $0 \leq \delta \leq 1$. The **mixture parameter** δ governs how much mass gets placed on the first distribution $f(x; \mu_1, \sigma_1)$ and the complement of δ governs how much mass gets placed on the other distribution $f_2(x; \mu_2, \sigma_2)$.

To further motivate this setting, consider simulating $n = 10,000$ heights from the population of both males and females. Assume that males are distributed normal with mean $\mu_1 = 70$[in] and standard deviation $\sigma_1 = 3$[in] and females are distributed normal with mean $\mu_2 = 64$[in] and standard deviation $\sigma_2 = 2.5$[in]. Also assume that each distribution contributes equal mass, i.e., set the mixture parameter to $\delta = .5$. The distribution of males is governed by

$$f_1(x; \mu_1, \sigma_1) = \frac{1}{\sqrt{2\pi\sigma_1^2}} \exp -\frac{1}{2\sigma_1^2}(x-\mu_1)^2, \quad -\infty < x < \infty,$$

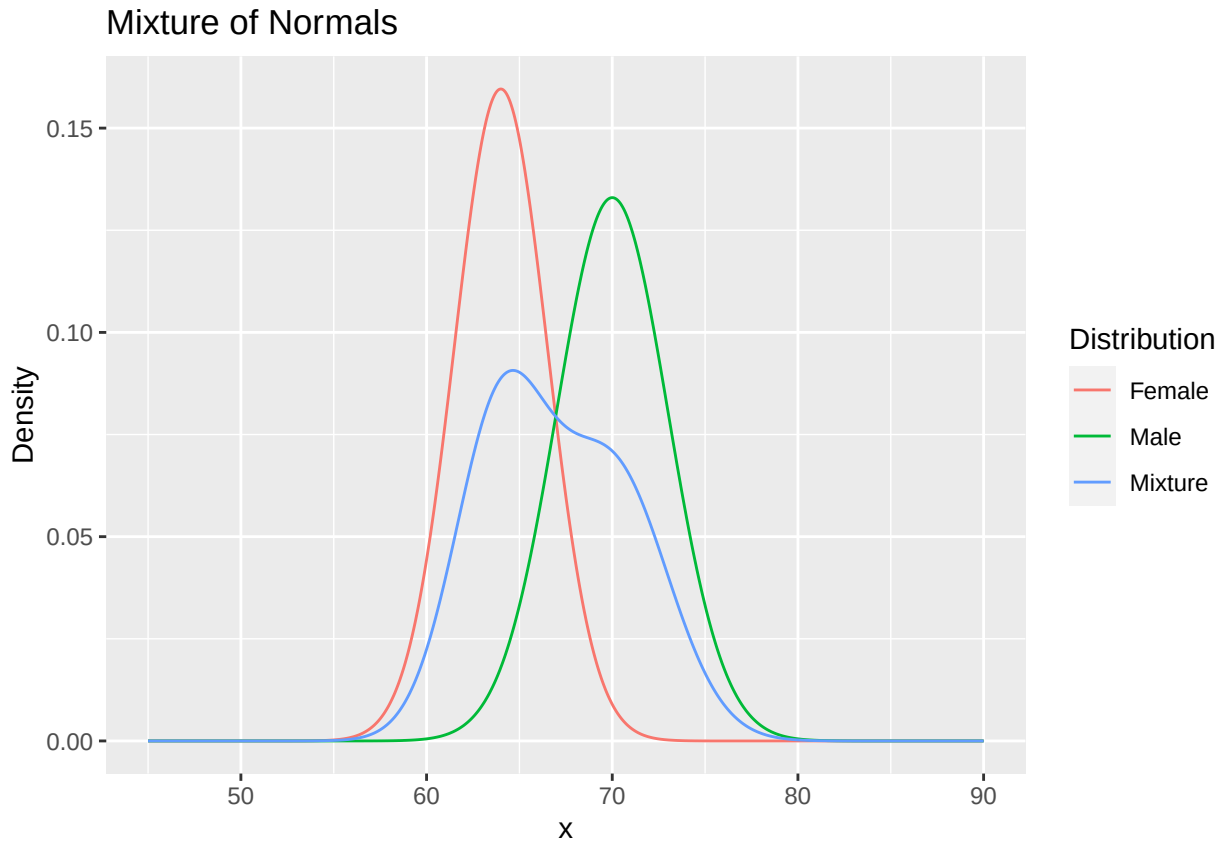
and the distribution of females is governed by

$$f_2(x; \mu_2, \sigma_2) = \frac{1}{\sqrt{2\pi\sigma_2^2}} \exp -\frac{1}{2\sigma_2^2}(x-\mu_2)^2, \quad -\infty < x < \infty.$$

Below shows the pdf of $f_1(x; \mu_1, \sigma_1)$, $f_2(x; \mu_2, \sigma_2)$ and the mixture $f(x)$ all on the same plot.

```
x <- seq(45, 90, by = 0.05)
n.x <- length(x)
f_1 <- dnorm(x, mean = 70, sd = 3)
```

```r
f_2 <- dnorm(x, mean = 64, sd = 2.5)
f <- function(x) {
    return(0.5 * dnorm(x, mean = 70, sd = 3) + 0.5 * dnorm(x,
        mean = 64, sd = 2.5))
}

plot_df <- data.frame(x = c(x, x, x), Density = c(f_1, f_2, f(x)),
    Distribution = c(rep("Male", n.x), rep("Female", n.x), rep("Mixture",
        n.x)))

library(ggplot2)
ggplot(data = plot_df) + geom_line(mapping = aes(x = x, y = Density,
    color = Distribution)) + labs(title = "Mixture of Normals")
```



Part I: Simulating a Mixture of Normals

The first goal is to simulate from the mixture distribution

$$\delta f_1(x; \mu_1, \sigma_1) + (1 - \delta) f_2(x; \mu_2, \sigma_2),$$

where $\mu_1 = 70, \sigma_1 = 3, \mu_2 = 64, \sigma_2 = 2.5, \delta = .5$. We use the accept-reject algorithm to accomplish this task.

First we must choose the "easy to simulate" distribution $g(x)$. For this problem choose $g(x)$ to be a Cauchy distribution centered at 66 with scale parameter 7.

```r
g <- function(x) {
    s = 7
    l = 66
    return(1/(pi * s * (1 + ((x - l)/s)^2)))
}
```

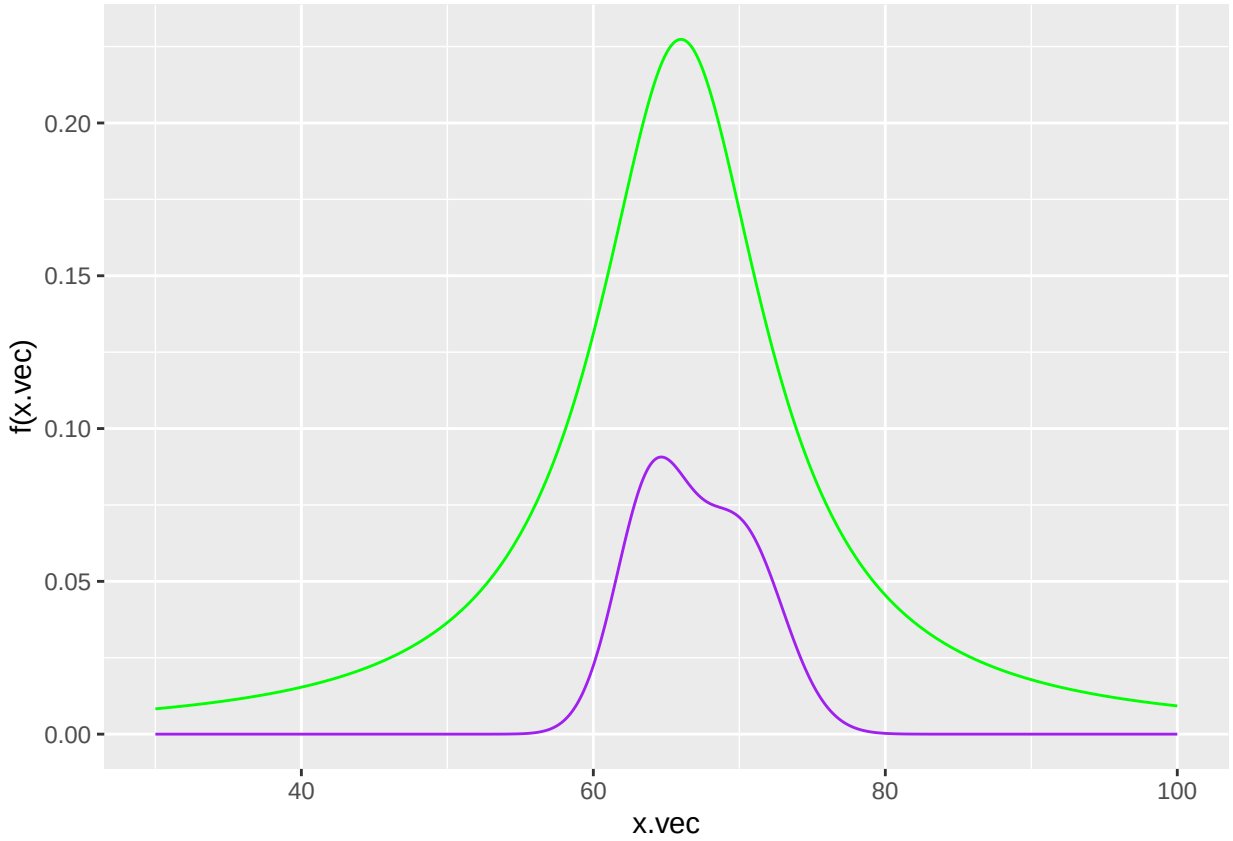
Perform the following tasks

1) Identify a **suitable** value of *alpha* such that your envelope function $e(x)$ satisfies

$$f(x) \leq e(x) = g(x)/\alpha, \quad \text{where} \quad 0 < \alpha < 1.$$

Note that you must choose α so that $e(x)$ is close to $f(x)$. There is not one unique solution to this problem. The below plot shows how $\alpha = .20$ creates an envelope function that is too large. Validate your choice of *alpha* with with a graphic similar to below.

```r
# Choose alpha
alpha <- 0.2

# Define envelope e(x)
e <- function(x) {
    return(g(x)/alpha)
}

# Plot
x.vec <- seq(30, 100, by = 0.1)
ggplot() + geom_line(mapping = aes(x = x.vec, y = f(x.vec)),
    col = "purple") + geom_line(mapping = aes(x = x.vec, y = e(x.vec)),
    col = "green")
```

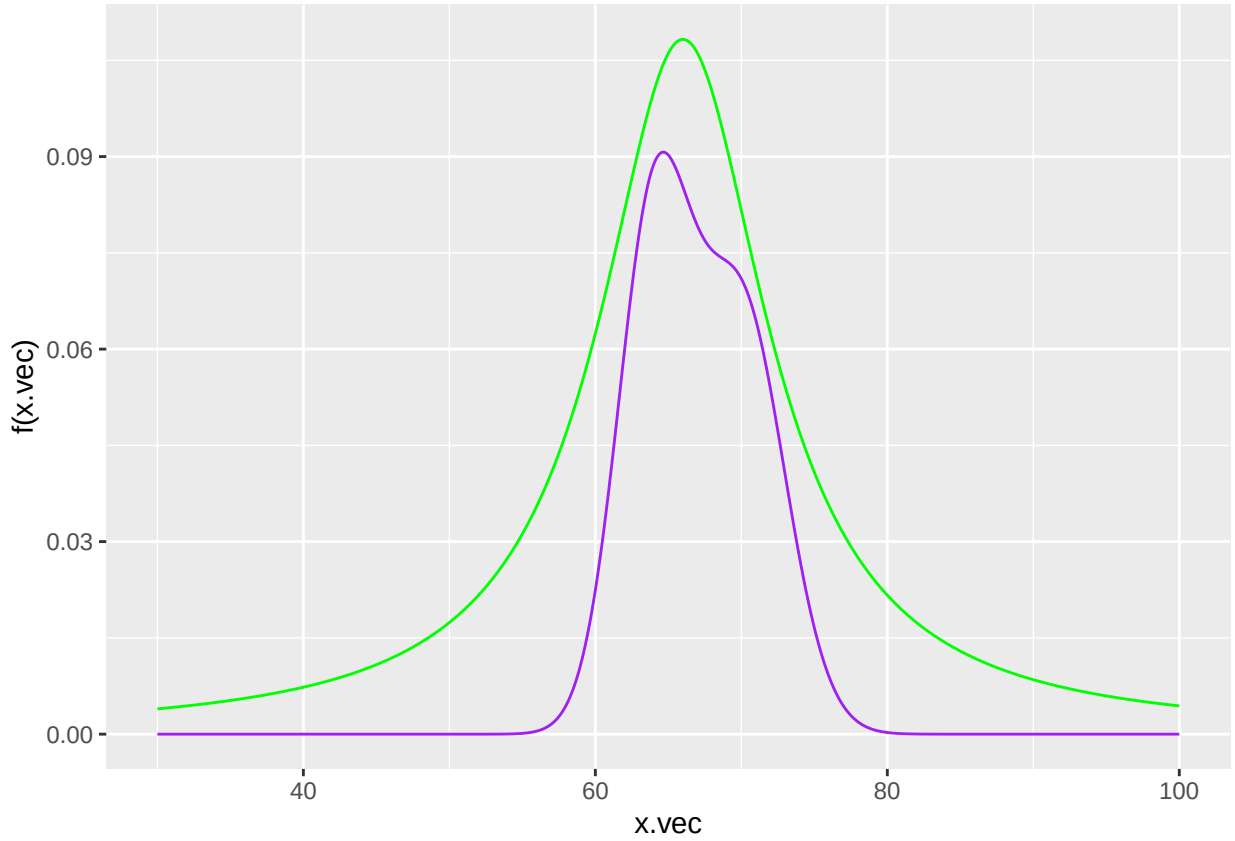



```
# Is g(x)>f(x)?
all(e(x.vec) > f(x.vec))
```

```
## [1] TRUE
```

Solution

```
alpha <- 0.42

# Plot
x.vec <- seq(30, 100, by = 0.1)
ggplot() + geom_line(mapping = aes(x = x.vec, y = f(x.vec)),
    col = "purple") + geom_line(mapping = aes(x = x.vec, y = e(x.vec)),
    col = "green")
```




```
# Is g(x)>f(x)?
all(e(x.vec) > f(x.vec))
```

```
## [1] TRUE
```

2) Write a function named **r.norm.mix()** that simulates **n.samps** from the normal-mixture $f(x)$. To accomplish this task you will wrap a function around the accept-reject algorithm from the lecture notes. Also include the acceptance rate, i.e., how many times did the algorithm accept a draw compared to the total number of trials performed. Your function should return a list of two elements: (i) the simulated vector mixture and (ii) the proportion of accepted cases. Run your function **r.norm.mix()** to simulate 10,000 cases and display the first 20 values. What's the proportion of accepted cases? Compare this number to your chosen α and comment on the result. The code below should help you get started.

Solution

```
r.norm.mix <- function(n.samps) {
    n <- 0  # counter for number samples accepted
    m <- 0  # counter for number of trials
    samps <- numeric(n.samps)  # initialize the vector of output
    while (n < n.samps) {
        m <- m + 1
        y <- rcauchy(1, location = 66, scale = 7)  #sample from g
        u <- runif(1)
        if (u < f(y)/e(y)) {
            n <- n + 1
```

```
            samps[n] <- y
        }
    }
    return(list(x = samps, alpha.hat = n.samps/m))
}

sim.norm.mix <- r.norm.mix(n.samps = 10000)
head(sim.norm.mix$x, 20)
```

```
##  [1] 61.80217 67.84918 71.63273 67.95398 66.68302 74.59870 60.88830 66.11376
##  [9] 64.63964 68.63727 69.15038 62.64270 63.19513 66.18190 66.31556 61.63554
## [17] 71.46076 66.73278 72.05453 66.18352
```

```
sim.norm.mix$alpha.hat
```

```
## [1] 0.4193575
```

```
alpha
```

```
## [1] 0.42
```

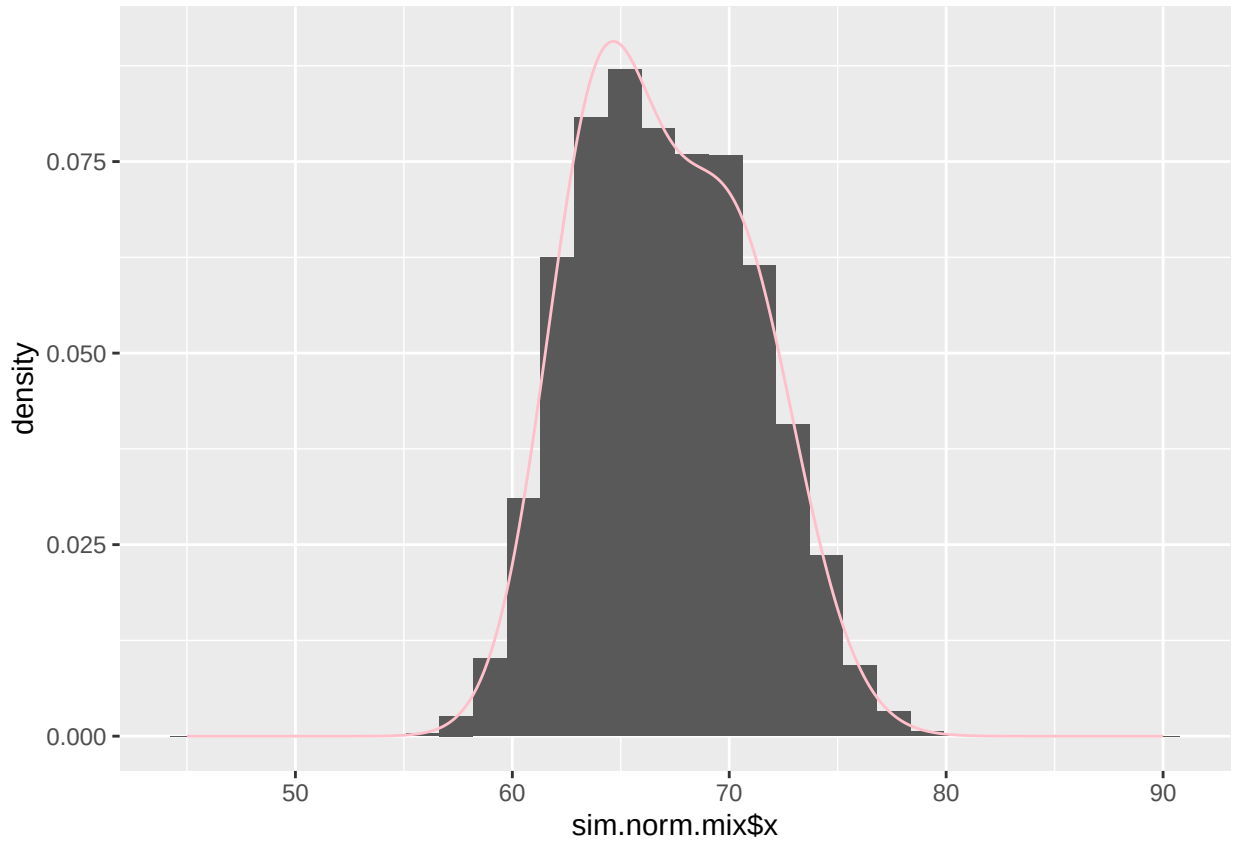
Since alpha = 0.42, we need to expect to accept around 42% of the cases, which is corroborated by alpha.hat. As alpha gets larger, e(x) moves closer to f(x) and % of cases in alpha hat increases.

3) Using **ggplot** or **base R**, construct a histogram of the simulated mixture distribution with the true mixture pdf $f(x)$ overlayed on the plot.

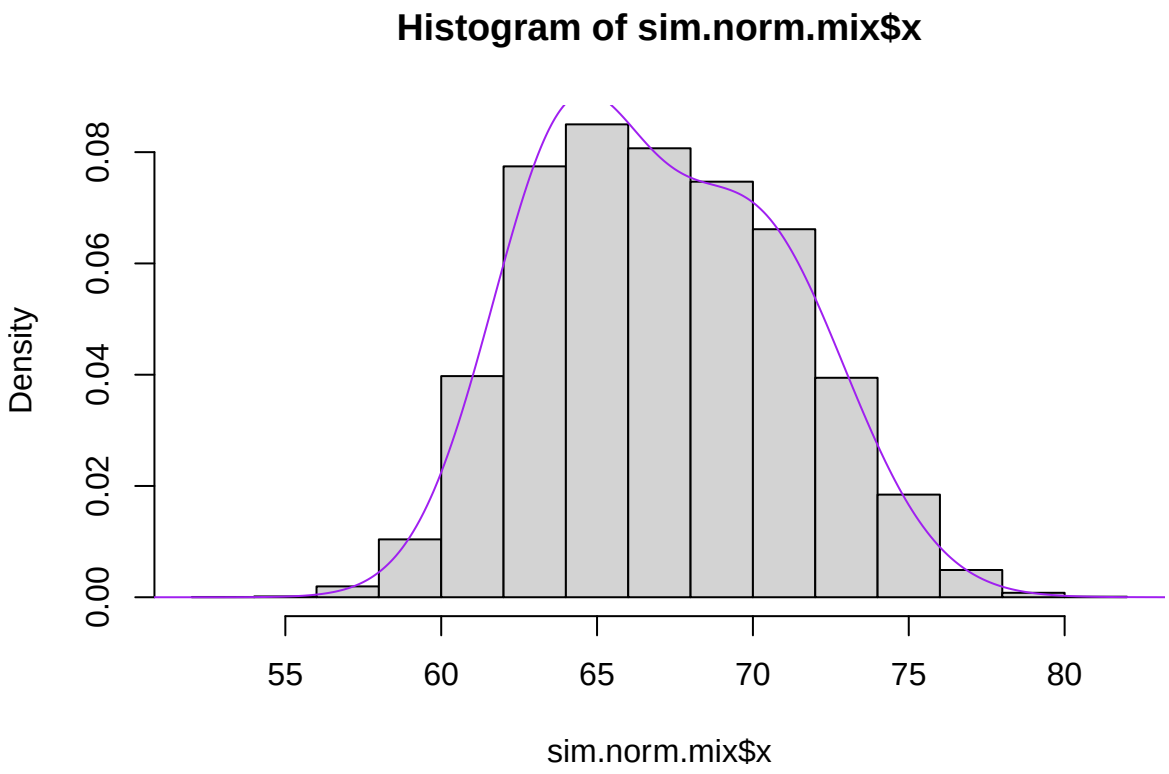
Solution

```
ggplot() + geom_histogram(aes(x = sim.norm.mix$x, y = ..density..)) +
    geom_line(aes(x = x, y = f(x)), col = "pink")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
hist(sim.norm.mix$x, probability = T)
lines(x, f(x), col = "purple")
```



Part II: Bayesian Statistics and MCMC

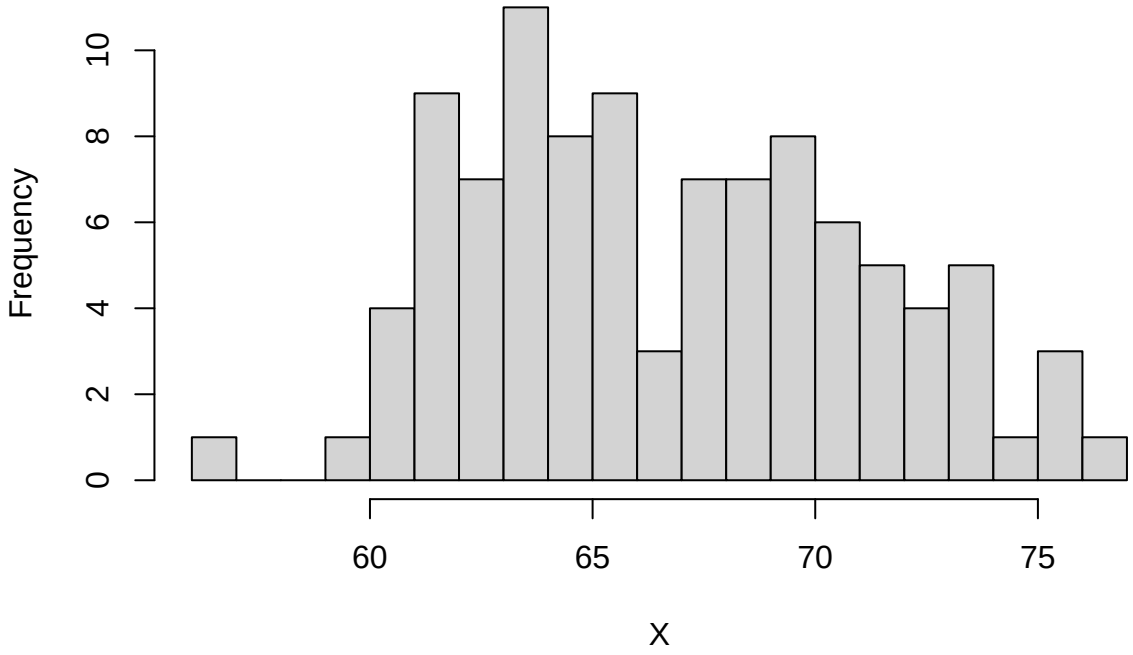
Suppose that the experimenter collected 100 cases from the true mixture-normal distribution $f(x)$. To solve problems (4) through (8) we analyze one realized sample from our function **r.norm.mix()**. In practice this dataset would be collected and not simulated. Uncomment the below code to simulate our dataset **x**. If you failed to solve Part I, then read in the csv file **mixture_data.csv** posted on Canvas.

Solution

```
# Simulate data set.seed(1983) x <- r.norm.mix(n.samps=100)$x
# head(x) hist(x,breaks=20,xlab='X',main='')

# Or read data
x <- read.csv("mixture_data.csv")$x
head(x)
```

```
## [1] 71.66666 63.91096 67.06554 65.49516 70.34363 65.69982
```

```
hist(x, breaks = 20, xlab = "X", main = "")
```



Further, suppose that we know the true heights and standard deviations of the two normal distributions but the mixture parameter δ is unknown. In this case, we know $\mu_1 = 70$, $\sigma_1 = 3$, $\mu_2 = 64$, $\sigma_2 = 2.5$. The goal of this exercise is to utilize **maximum likelihood** and **MCMC Bayesian techniques** to estimate mixture parameter δ.

Maximum likelihood Estimator of Mixture Parameter

4) Set up the likelihood function $L(\delta|x_1, \ldots, x_{100})$ and define it as **mix.like()**. The function should have two inputs including the parameter **delta** and data vector **x**. Evaluate the likelihood at the parameter values **delta=.2**, **delta=.4,** and **delta=.6**. Note that all three evaluations will be very small numbers. Which delta ($\delta = .2, .4, .6$) is the most likely to have generated the dataset **x**?
Solution

```
mix.like <- function(delta, x) {
    ll.func <- prod(delta * dnorm(x, mean = 70, sd = 3) + (1 -
        delta) * dnorm(x, mean = 64, sd = 2.5))
    return(ll.func)
}

mix.like(0.2, x = x)
```

```
## [1] 4.546128e-129
```

```r
mix.like(0.4, x = x)
```

```
## [1] 1.967793e-124
```

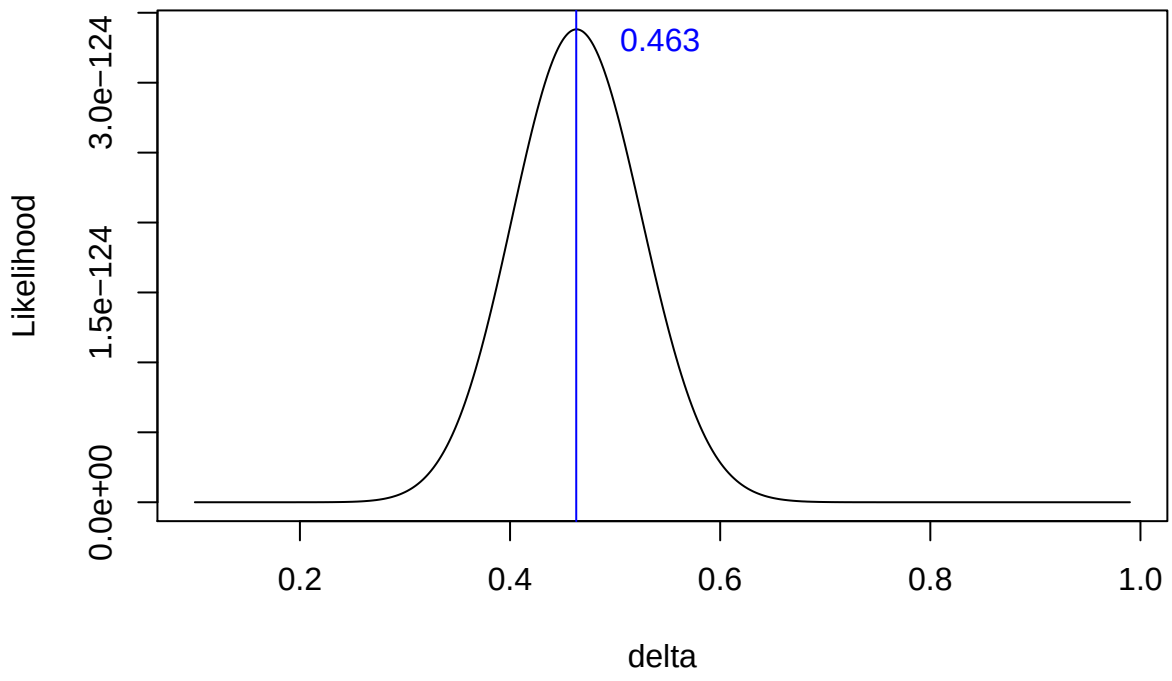
```r
mix.like(0.6, x = x)
```

```
## [1] 2.838963e-125
```

delta = 0.4 is most likely

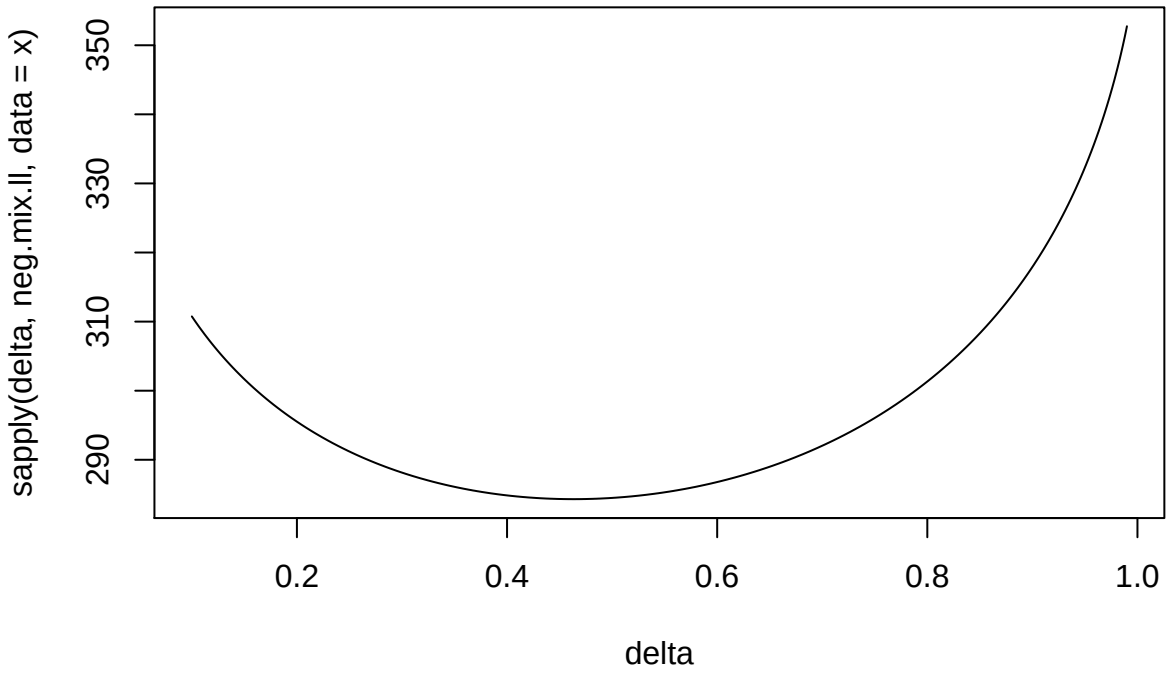
5) Compute the maximum likelihood estimator of mixture parameter δ. To accomplish this task, apply your likelihood function **mix.like()** across the vector **seq(.1,.99,by=.001)**. The solution to this exercise is given below.

```r
# apply likelihood function across seq vector 0.1-0.99
delta <- seq(0.1, 0.99, by = 0.001)
MLE.values <- sapply(delta, mix.like, x = x)

# which delta value has the max likelihood?
delta.MLE <- delta[which.max(MLE.values)]

# plotting MLE values with max MLE value
plot(delta, MLE.values, ylab = "Likelihood", type = "l")
abline(v = delta.MLE, col = "blue")
text(x = delta.MLE + 0.08, y = mix.like(delta = 0.45, x = x),
    paste(delta.MLE), col = "blue")
```



Optional:

```
neg.mix.ll <- function(delta, data = x) {
    return(-1 * sum(log(delta * dnorm(x, mean = 70, sd = 3) +
        (1 - delta) * dnorm(x, mean = 64, sd = 2.5))))
}

plot(delta, sapply(delta, neg.mix.ll, data = x), type = "l")
```




```
nlm(neg.mix.ll, p = 0.5)
```

```
## Warning in log(delta * dnorm(x, mean = 70, sd = 3) + (1 - delta) * dnorm(x, :
## NaNs produced

## Warning in nlm(neg.mix.ll, p = 0.5): NA/Inf replaced by maximum positive value

## Warning in log(delta * dnorm(x, mean = 70, sd = 3) + (1 - delta) * dnorm(x, :
## NaNs produced

## Warning in nlm(neg.mix.ll, p = 0.5): NA/Inf replaced by maximum positive value

## $minimum
## [1] 284.3022
##
```

```
## $estimate
## [1] 0.4632656
##
## $gradient
## [1] 0.0001152785
##
## $code
## [1] 1
##
## $iterations
## [1] 3
```

MCMC

6) Run the Metropolis-Hastings algorithm to estimate mixture parameter δ. In this exercise you will assume a Beta($\alpha = 10, \beta = 10$) prior distribution on mixture parameter δ. Some notes follow:

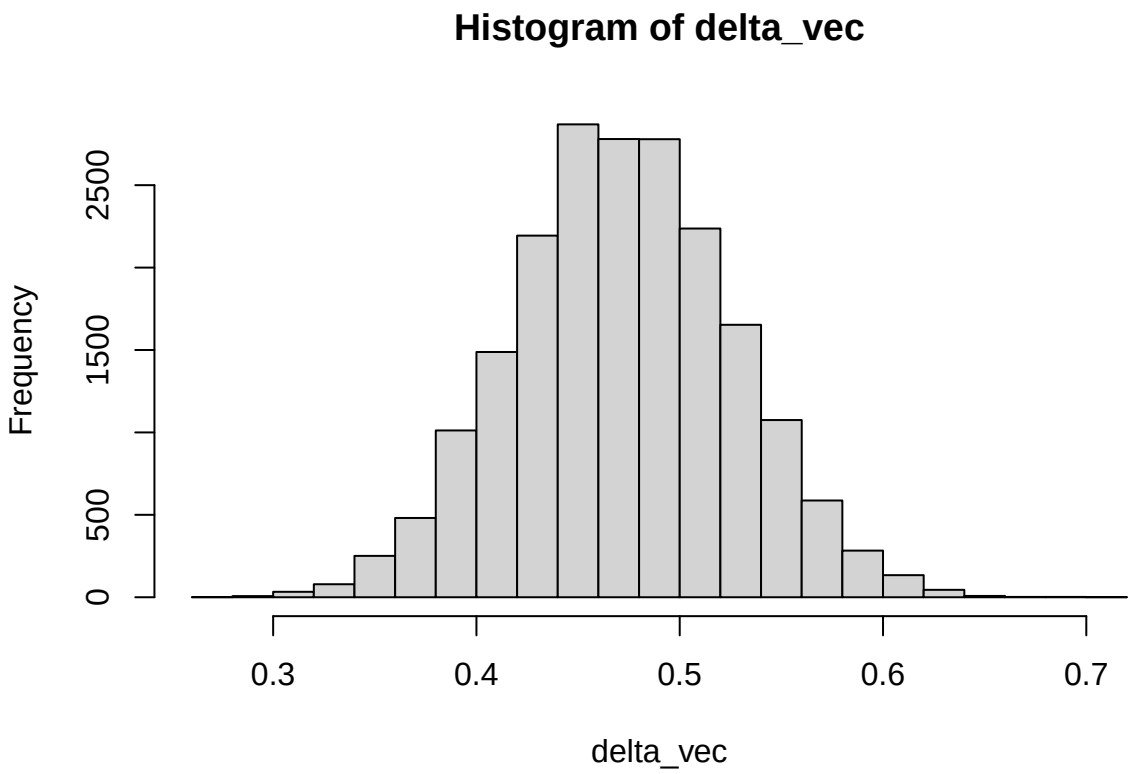
   - Run 20000 iterations. I.e., simulate 20000 draws of $\delta^{(t)}$
   - Proposal distribution Beta($\alpha = 10, \beta = 10$)
   - Independence chain with Metropolis-Hastings ratio:

$$R(\delta^{(t)}, \delta^*) = \frac{L(\delta^*|x_1, \ldots, x_{100})}{L(\delta^{(t)}|x_1, \ldots, x_{100})}$$

   Display the first 20 simulated cases of $\delta^{(t)}$.

Solution

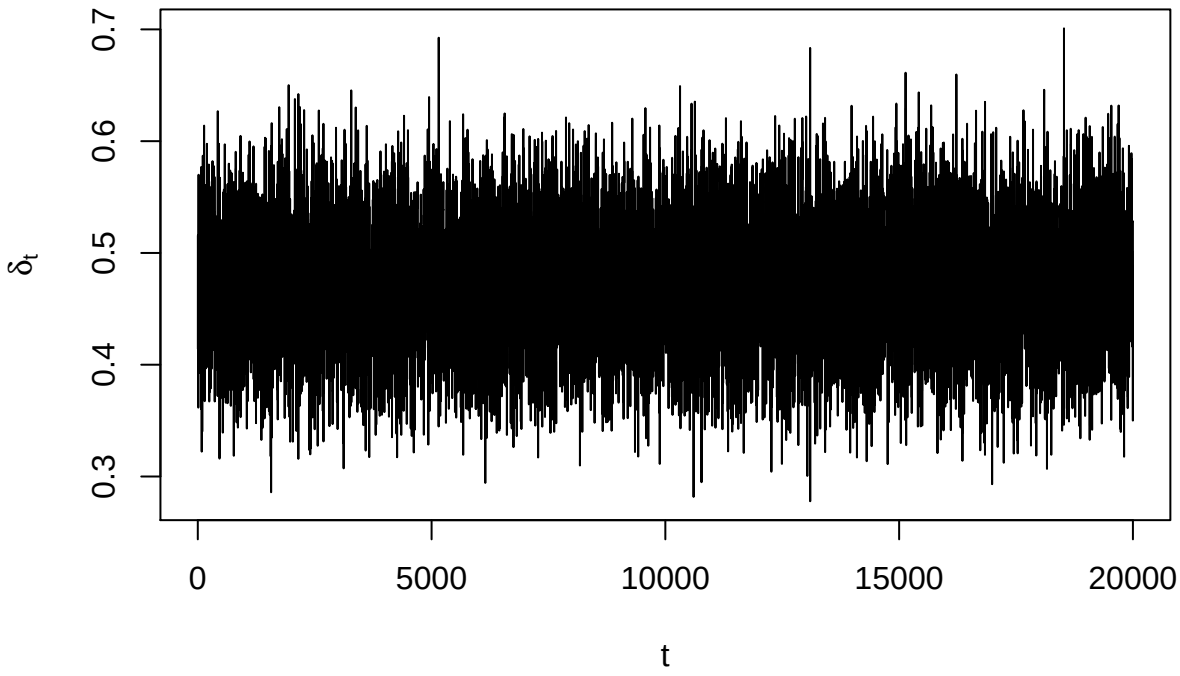
```
# draw initial theta (delta_0 in this case)
delta_0 <- rbeta(1, shape1 = 10, shape2 = 10)
n.samps <- 20000  #number of iterations
delta_vec <- rep(NA, n.samps + 1)
delta_vec <- delta_0

# MCMC loop
for (t in 1:n.samps) {
    delta_star <- rbeta(1, shape1 = 10, shape2 = 10)  #draw delta* from proposal
    delta_t <- delta_vec[t]  #delta_(t)
    # compute MH ratio
    MH_ratio <- mix.like(delta = delta_star, x = x)/mix.like(delta = delta_t,
        x = x)
    # select new case
    prob_vec <- c(min(MH_ratio, 1), 1 - min(MH_ratio, 1))
    delta_vec[t + 1] <- sample(c(delta_star, delta_t), 1, prob = prob_vec)
}


hist(delta_vec, breaks = 30)
```



7) Construct a lineplot of the simulated Markov chain from exercise (6). The vertical axis is the simulated chain $\delta^{(t)}$ and the horizontal axis is the number of iterations.

Solution

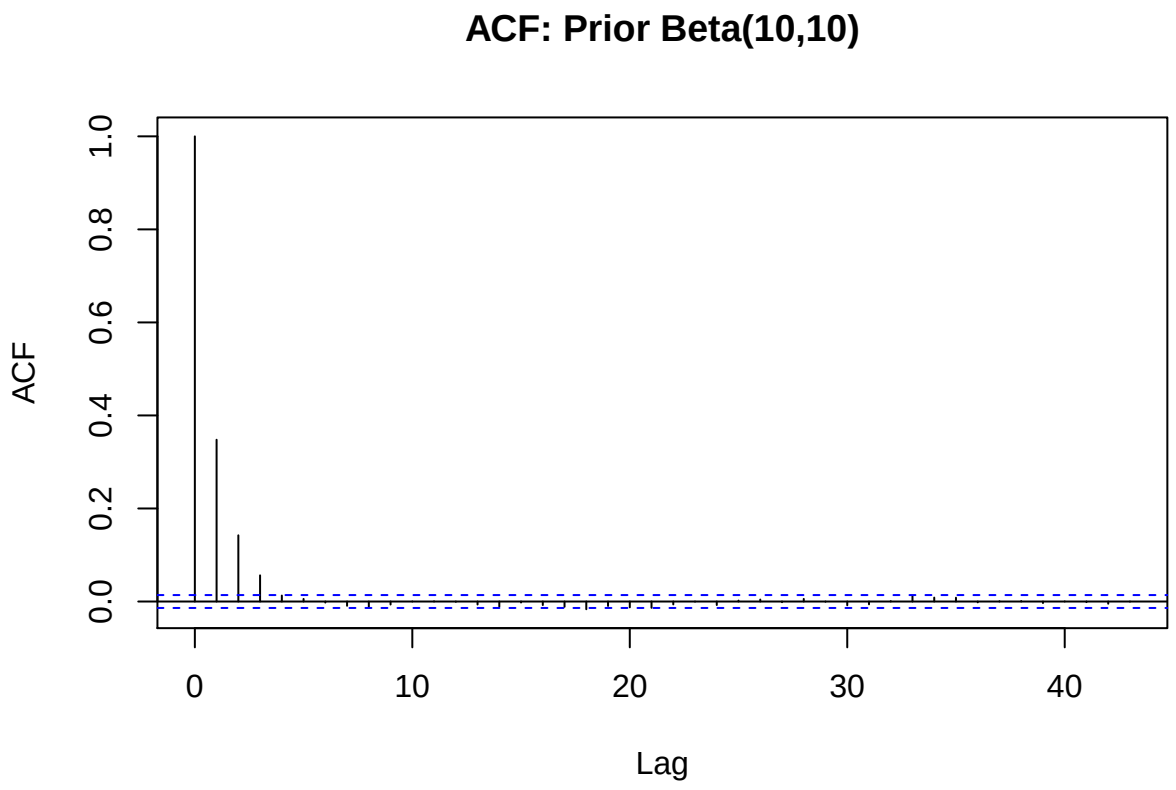
```
# lineplot of simulated Markov chain (vertical axis is the
# simulated chain)
plot(delta_vec, type = "l", xlab = "t", ylab = expression(delta[t]))
```



8) Plot the empirical autocorrelation function of your simulated chain $\delta^{(t)}$. I.e., run the function **acf()**. A quick decay of the chain's autocorrelations indicate good mixing properties.

Solution

```
# Plotting the empirical autocorrelation function of
# simulated chain
acf(delta_vec, main = "ACF: Prior Beta(10,10)")
```



9) Compute the empirical Bayes estimate $\hat{\delta}_B$ of the simulated posterior distribution $\pi(\delta|x_1, \ldots, x_n)$. To solve this problem, simply compute the sample mean of your simulated chain $\delta^{(t)}$ after discarding a 20% burn-in.

Solution

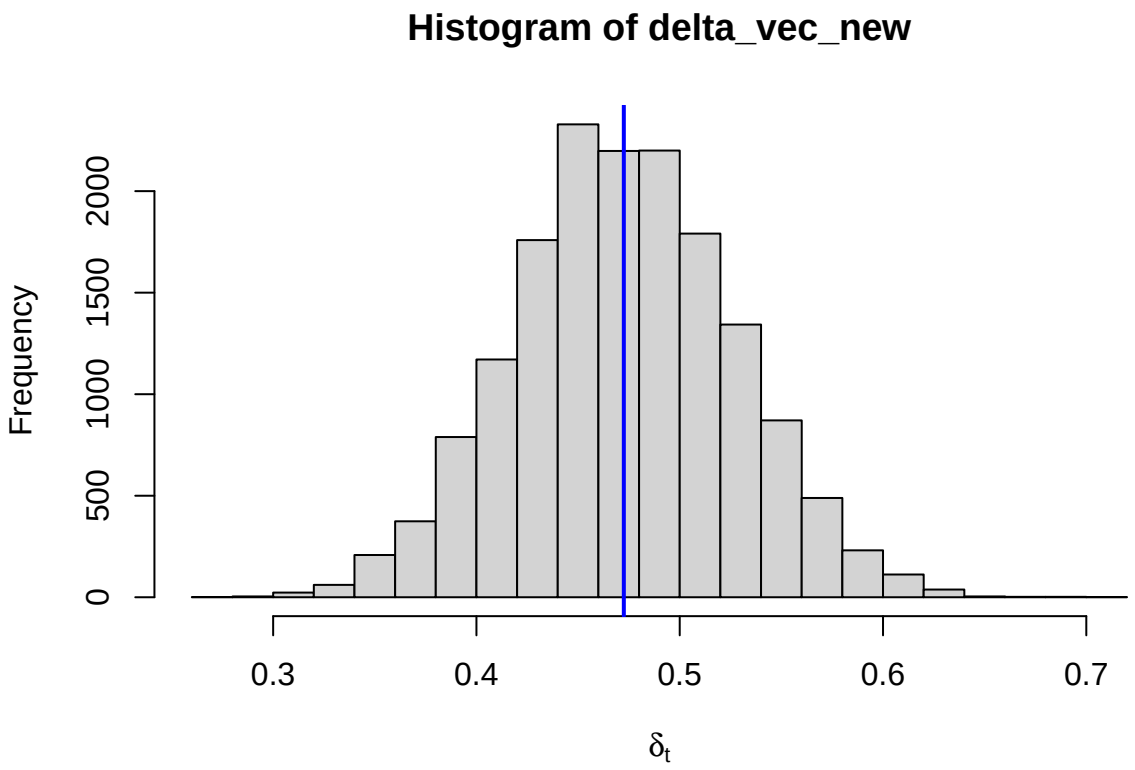
```
# discard the first 20% of the cases subset delta_vec from
# 4001 to 20000
delta_vec_new <- delta_vec[(floor(0.2 * length(delta_vec)) +
    1):length(delta_vec)]

mean(delta_vec_new)
```

```
## [1] 0.4725019
```

10) Construct a histogram of the simulated posterior $\pi(\delta|x_1, \ldots, x_n)$ after discarding a 20% burn-in.

Solution

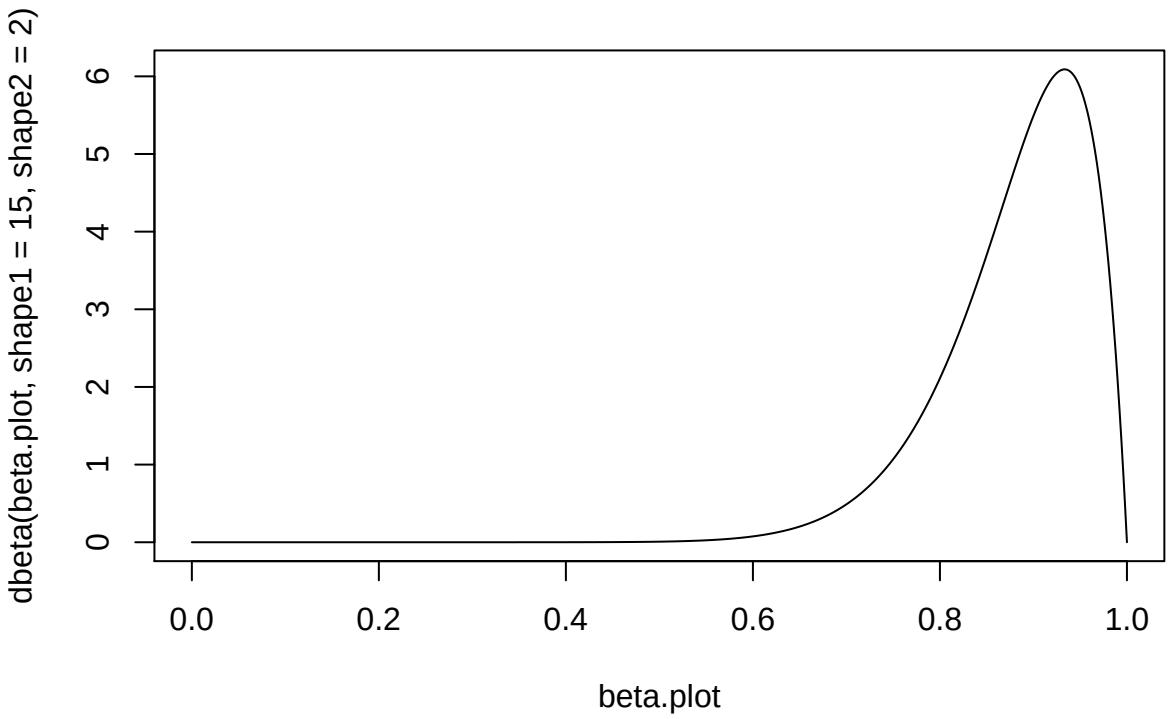
```
# histogram of the simulated posterior after discarding a 20%
# burn-in
hist(delta_vec_new, breaks = 30, xlab = expression(delta[t]))
abline(v = mean(delta_vec_new), col = "blue", lwd = 2)
```



11) Run the Metropolis-Hastings algorithm to estimate the mixture parameter δ using a Beta($\alpha = 15, \beta = 2$) prior distribution on mixture parameter δ. Repeat exercises 6 though 10 using the updated prior.

Solution

```
beta.plot <- seq(0, 1, length = 500)
plot(beta.plot, dbeta(beta.plot, shape1 = 15, shape2 = 2), type = "l")
```

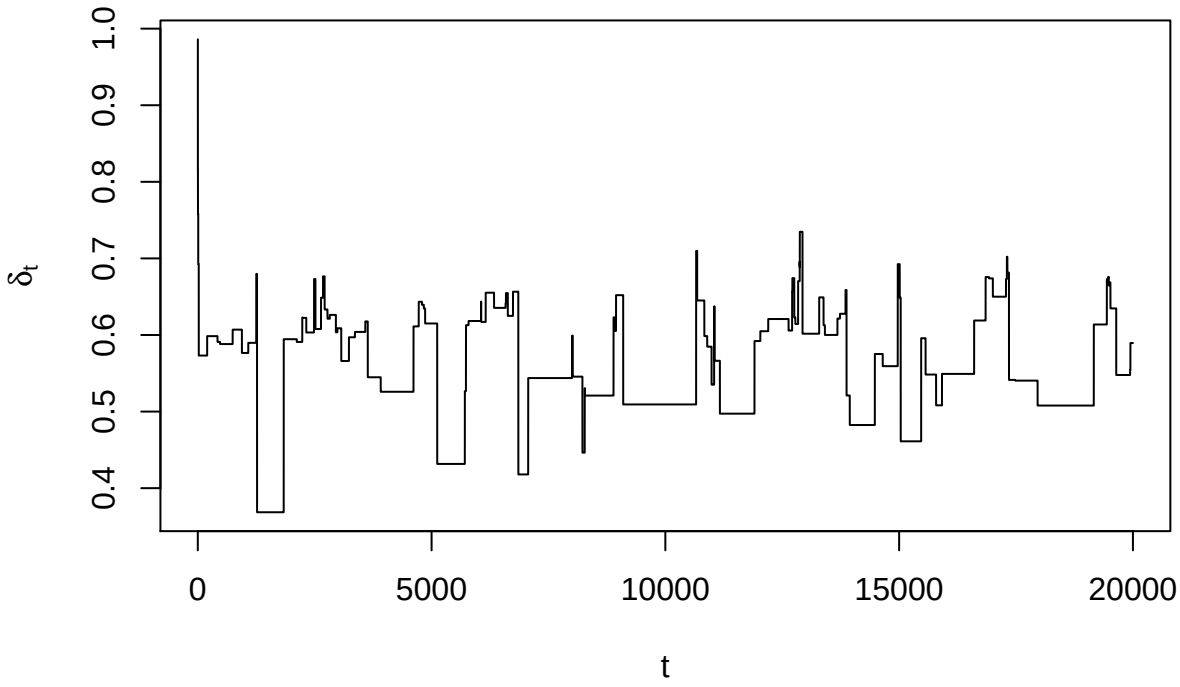



```
delta_0 <- rbeta(1, shape1 = 15, shape2 = 2)
n.samps <- 20000
delta_vec <- rep(NA, n.samps + 1)
delta_vec = delta_0

for (t in 1:n.samps) {
    delta_star <- rbeta(1, shape1 = 15, shape2 = 2)
    delta_t <- delta_vec[t]
    MH_ratio <- mix.like(delta = delta_star, x = x)/mix.like(delta = delta_t,
        x = x)
    prob_vec <- c(min(1, MH_ratio), 1 - min(1, MH_ratio))
    delta_vec[t + 1] <- sample(c(delta_star, delta_t), 1, prob = prob_vec)
}
```

lineplot:

Construct a lineplot of the simulated Markov chain from exercise (6). The vertical axis is the simulated chain $\delta^{(t)}$ and the horizontal axis is the number of iterations.

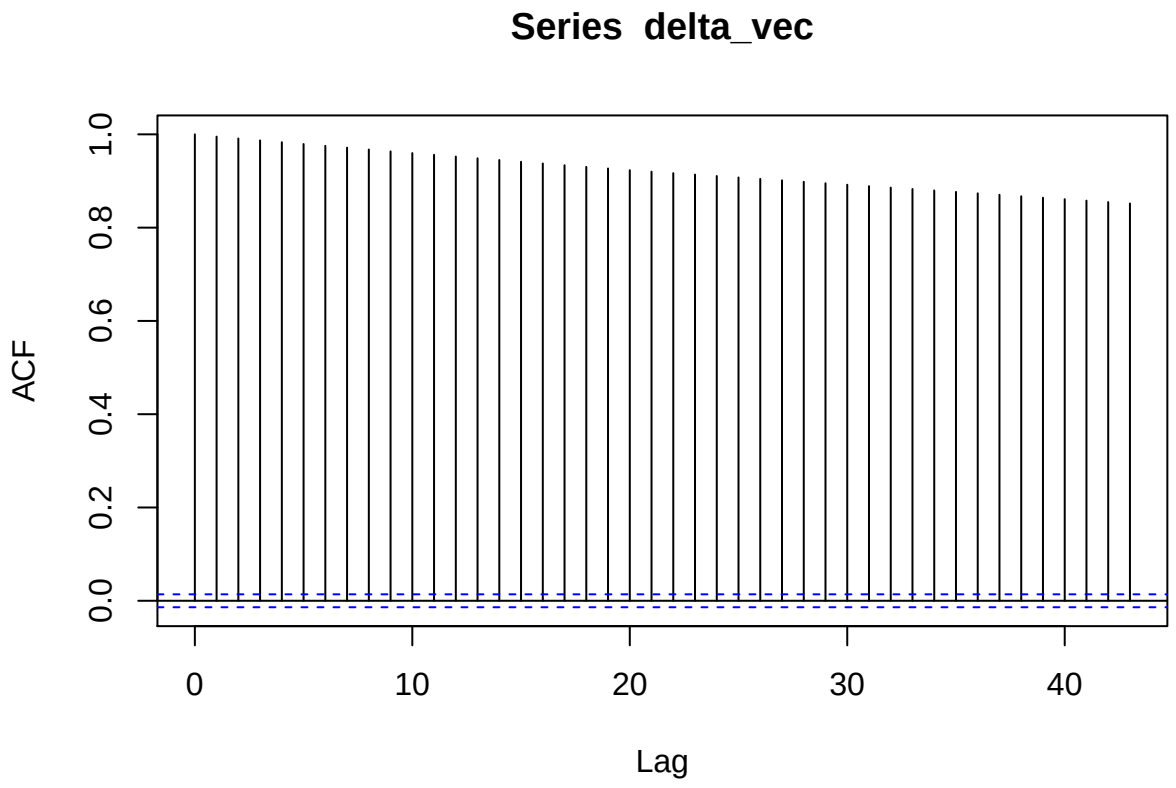
```
plot(delta_vec, type = "l", xlab = "t", ylab = expression(delta[t]))
```



ACF:

Plot the empirical autocorrelation function of your simulated chain $\delta^{(t)}$. I.e., run the function **acf()**. A slow decay of the chain's autocorrelations indicate poor mixing properties.

```
acf(delta_vec)
```



Bayes estimate:

Compute the empirical Bayes estimate $\hat{\delta}_B$ of the simulated posterior distribution $\pi(\delta|x_1, \ldots, x_n)$. To solve this problem, simply compute the sample mean of your simulated chain $\delta^{(t)}$ after discarding a 20% burn-in. Your answer should be close to the MLE.

Solution

```
delta_vec_80 <- delta_vec[(floor(0.2 * length(delta_vec)) + 1):length(delta_vec)]
mean(delta_vec_80)
```

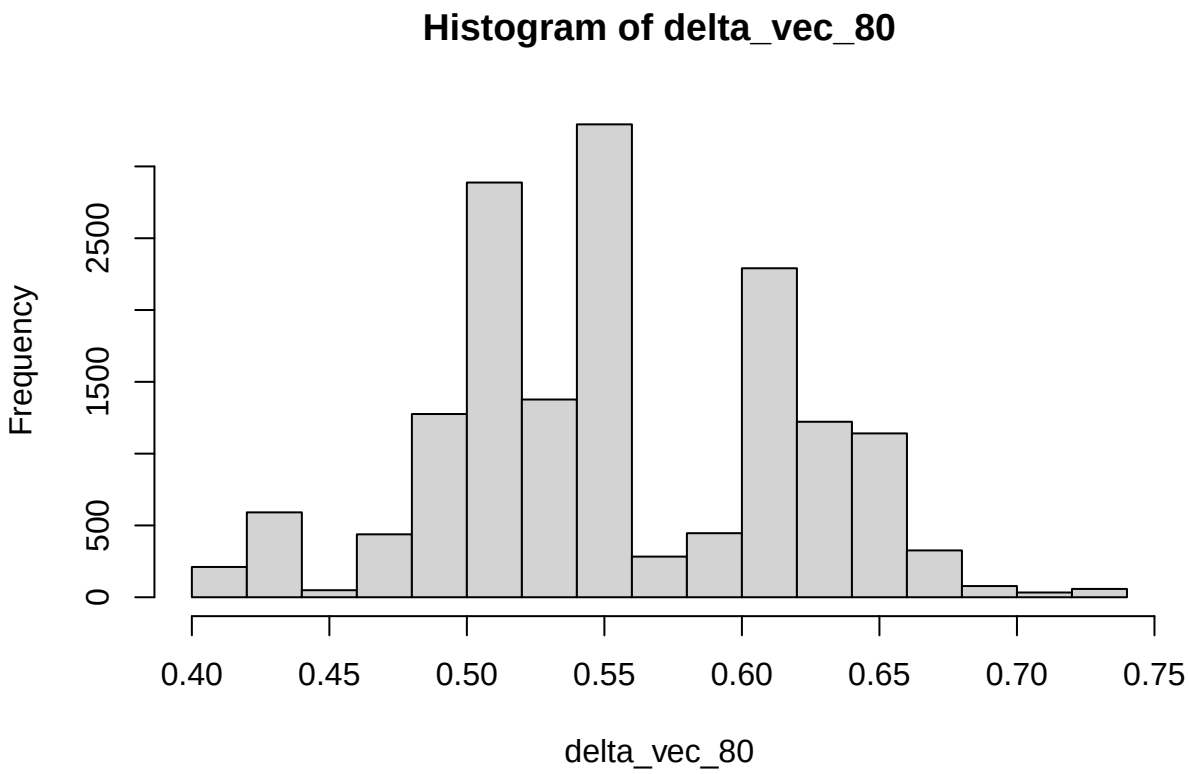
```
## [1] 0.5535371
```

MLE value is 0.463

Posterior: Construct a histogram of the simulated posterior $\pi(\delta|x_1, \ldots, x_n)$ after discarding a 20% burn-in.

Solution

```
hist(delta_vec_80)
```


Histogram of delta_vec_80
Frequency
0
500
1500
2500
0.40
0.45
0.50
0.55
0.60
0.65
0.70
0.75
delta_vec_80