Math 475: Final Exam

Hannah Zmuda

01/08/2021

Problem 1

part a

Show the outputs of the EM algorithm are consistent with the given parameter equations To find the updated parameters (i.e. the maximized value) we fist need to find the Q-function (E step) then maximize the Q function by taking the derivative in regard to each of the parameters (M step). **E step:** Given the observed likelihood, we can compute the complete likelihood to be:

$$L(\theta|n_{k,i}) = \Pi_{i=0}^{16} \frac{[\pi_i(\theta)]^{(n_i)}}{i!}$$

Given the likelihood equation, we can work out the log likelihood to be:

$$log[L(\theta|n_{k,i})] = \Sigma_{i=0}^{16} n_{k,i} log(\alpha 1_{i=0} + \beta\mu^i e^{-\mu} + (1-\alpha-\beta)\lambda^i e^{-\lambda}))$$

where y is the complete data set and $n_{k,i}$ represent the number of the three different groups. These are further broken down into the zero, typical, and promiscuous groups. To find your Q-function, take the expectation of the log likelihood function:

$$E(n_{z,0}|n_{t,0}, n_{p,0}, \theta^{(t)}) = \Sigma_{i=0}^{16} \frac{z_0(\theta^{(t)})}{n_0 z_0(\theta^{(t)}) + n_0 t_0(\theta^{(t)}) + n_0 p_0(\theta^{(t)})} = z_0^{(t)}(\theta^{(t)})$$

$$E(n_{t,i}|n_{z,i}, n_{p,i}, \theta^{(t)}) = \Sigma_{i=0}^{16} \frac{t_i(\theta^{(t)})}{n_i t_i(\theta^{(t)}) + n_i p_i(\theta^{(t)})} = t_i^{(t)}(\theta^{(t)})$$

$$E(n_{p,i}|n_{z,i}, n_{t,i}, \theta^{(t)}) = \Sigma_{i=0}^{16} \frac{p_i(\theta^{(t)})}{n_i t_i(\theta^{(t)}) + n_i p_i(\theta^{(t)})} = p_i^{(t)}(\theta^{(t)})$$

$$Q(\theta|\theta^{(t)}) = n_{z,0}^{(t)} \Sigma_{i=0}^{16} log(\alpha 1_{i=0}) + n_{t,i}^{(t)} \Sigma_{i=0}^{16} log(\beta\mu^i exp(-\mu)) + n_{p,i}^{(t)} \Sigma_{i=0}^{16} log((1-\alpha-\beta)\lambda^i exp(\lambda)) - n_i log(i!)$$

M step: For the M step of the EM algorithm, we need to maximize the Q function in regard to each parameter then set it equal to zero.

When the derivative is set equal to zero, we find that the updated parameters equal to what we expected:

$$\frac{\partial Q(\theta|\theta^{(t)})}{\partial \alpha^{(t)}} = 0$$

$$\alpha^{(t+1)} = \frac{n_0 z_0(\theta^{(t)} t)}{N}$$

$$\frac{\partial Q(\theta|\theta^{(t)})}{\partial \beta^{(t)}} = 0$$

$$\beta^{(t+1)} = \Sigma_{i=0}^{16} \frac{n_i t_i(\theta^{(t)})}{N}$$

$$\frac{\partial Q(\theta|\theta^{(t)})}{\partial \mu^{(t)}} = \frac{n_i t(\theta^{(t)})}{n_i t(\theta^{(t)})} \frac{-\mu - i}{\mu} = 0$$

$$\mu^{(t+1)} = \frac{\Sigma_{i=0}^{16} i n_i t_i(\theta^{(t)})}{\Sigma_{i=0}^{16} n_i t_i(\theta^{(t)})}$$

$$\frac{\partial Q(\theta|\theta^{(t)})}{\partial \lambda^{(t)}} = \frac{n_i p(\theta^{(t)})}{n_i p(\theta^{(t)})} \frac{-\lambda^{(t)} - i}{\lambda^{(t)}} = 0$$

$$\lambda^{(t+1)} = \frac{\Sigma_{i=0}^{16} i n_i p_i(\theta^{(t)})}{\Sigma_{i=0}^{16} n_i p_i(\theta^{(t)})}$$

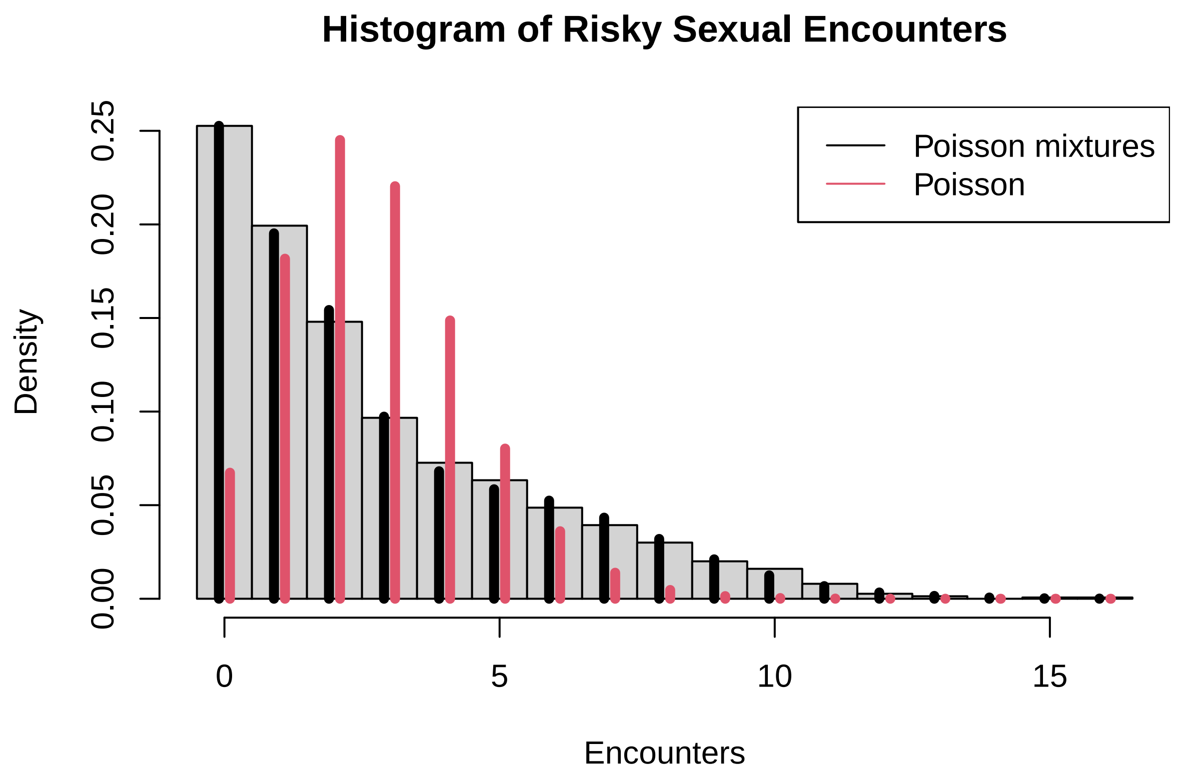
part b and c

```
set.seed(475)
# initialize variables
data = data.frame(enc = 0:16, freq = c(379, 299, 222, 145, 109,
    95, 73, 59, 45, 30, 24, 12, 4, 2, 0, 1, 1))
N = sum(data$freq)
y = rep(data$enc, data$freq)
alpha = 0.5
beta = 0.8
mu = 2
lambda = 15
param = c(alpha, beta, mu, lambda)
param.guess = c(0.1, 0.2, 3, 4)
tol = 1e-10
tol.cur = 100
time = 0
i = 0:16
# functions
log.likelihood <- function(alpha, beta, mu, lambda, x) {
    l = 0
    alpha = exp(alpha)/(1 + exp(alpha))
    beta = exp(beta)/(1 + exp(beta))
    mu = exp(mu)/(1 + exp(mu))
    lambda = exp(lambda)/(1 + exp(lambda))
    for (i in 1:length(x$enc)) {
        e = x$enc[i]
        n = x$freq[i]
        if (e == 0) {
            l = l + n * log(alpha + beta * exp(-mu) + (1 - alpha -
                beta) * exp(-lambda))
            print(l)
        } else {
            l = l + n * log(beta * (mu^e) * exp(-mu) + (1 - alpha -
                beta) * exp(-lambda) * lambda^e) - log(factorial(e))
            print(l)
```

```r
        }
    }
    return(l)
}

# EM Algorithm
while (tol.cur > tol) {
    pi = (beta * exp(-mu) * mu^i) + ((1 - alpha - beta) * exp(-lambda) *
        lambda^i)
    pi[1] = pi[1] + alpha
    z.stat = alpha/(pi[1])
    t.stat = (beta * (mu^i) * exp(-mu))/pi
    p.stat = ((1 - alpha - beta) * exp(-lambda) * lambda^i)/pi
    alpha = (data$freq[1] * z.stat)/N
    beta = sum(data$freq * t.stat)/N
    mu = sum(i * data$freq * t.stat)/sum(data$freq * t.stat)
    lambda = sum(i * data$freq * p.stat)/sum(data$freq * p.stat)
    new.param = c(alpha, beta, mu, lambda)
    tol.cur = sum(abs(new.param - param))
    param = new.param
    time = time + 1
}
param
```

```
## [1] 0.1221661 0.5625419 1.4674746 5.9388889
```

```r
hist(rep(data$enc, data$freq), breaks = -0.5 + c(0:17), freq = F,
    main = "Histogram of Risky Sexual Encounters", xlab = "Encounters")
# hist(y,freq=F)
z = 0:16
prob = (beta * exp(-mu) * mu^z + (1 - alpha - beta) * exp(-lambda) *
    lambda^z)/(factorial(z))
prob[1] = prob[1] + alpha
for (i in 1:length(z)) {
    lines(c(z[i] - 0.1, z[i] - 0.1), c(0, prob[i]), lwd = 5,
        col = 1)
}

pois.hat = mean(y)

for (i in 1:length(z)) {
    lines(c(z[i] + 0.1, z[i] + 0.1), c(0, dpois(z[i], pois.hat)),
        lwd = 5, col = 2)
}

legend("topright", c("Poisson mixtures", "Poisson"), lty = 1,
    col = 1:2)
```



```
# standard error Use log likelihood at theta.hat values (i.e.
# new parameter values)
set.seed(1234)
data = data.frame(enc = 0:16, freq = c(379, 299, 222, 145, 109,
    95, 73, 59, 45, 30, 24, 12, 4, 2, 0, 1, 1))
tol = 1e-10
B = 1000
result.boot <- NULL
alpha <- 1
beta <- 2
mu <- 4
lambda <- 10
param <- c(alpha, beta, mu, lambda)

for (j in 1:B) {
    # randomize samples
    data.boot <- rmultinom(1, sum(data$freq), prob = data$freq/length(data$freq))
    # set initial values
    tol.cur <- 100
    N <- sum(data.boot)
    i = c(0:16)
    # loop
    while (tol.cur > tol) {
        pi = (beta * exp(-mu) * mu^i) + ((1 - alpha - beta) *
            exp(-lambda) * lambda^i)
        pi[1] = pi[1] + alpha
        z.stat = alpha/(pi[1])
        t.stat = (beta * (mu^i) * exp(-mu))/pi
        p.stat = ((1 - alpha - beta) * exp(-lambda) * (lambda^i))/pi

        alpha = (data.boot[1] * z.stat)/N
        beta = sum(data.boot * t.stat)/N
        mu = sum(i * data.boot * t.stat)/sum(data.boot * t.stat)
        lambda = sum(i * data.boot * p.stat)/sum(data.boot *
            p.stat)

        new.param = c(alpha, beta, mu, lambda)
        tol.cur = sum(abs(new.param - param))
        param = new.param
    }
    result.boot <- rbind(result.boot, param)
}
result.boot[B, ]
```

```
## [1] 0.1365674 0.5705522 1.6564927 6.2467511
```

```
c(sd(result.boot[, 1])/sqrt(B), sd(result.boot[, 2])/sqrt(B),
    sd(result.boot[, 3])/sqrt(B), sd(result.boot[, 4])/sqrt(B))
```

```
## [1] 0.0006288919 0.0006627924 0.0034923528 0.0062272633
```

```
cov(result.boot)  #covariance matrix to show standard error
```

```
##               [,1]          [,2]         [,3]        [,4]
## [1,]  0.0003955050 -0.0001790008 0.0015697232 0.001484594
## [2,] -0.0001790008  0.0004392937 0.0001291111 0.001400945
## [3,]  0.0015697232  0.0001291111 0.0121965277 0.013091738
## [4,]  0.0014845938  0.0014009455 0.0130917380 0.038778808
```

```
# pairwise correlation
cor(result.boot)
```

```
##            [,1]        [,2]       [,3]      [,4]
## [1,]  1.0000000 -0.42943889 0.71470853 0.3790831
## [2,] -0.4294389  1.00000000 0.05577864 0.3394271
## [3,]  0.7147085  0.05577864 1.00000000 0.6019799
## [4,]  0.3790831  0.33942709 0.60197988 1.0000000
```

Problem 2

part a: Metropolis and M-H Algorithm

We are seeking to estimate $\alpha, \beta, \mu, \lambda$ using the Metropolis Sampler and Metropolis-Hastings Algorithm.

Because the proposal distribution is symmetric, it can be canceled out in the ratio calculation, and we can then focus on the ratio of the target distribution with proposed values based on the value in the previous chain and the previous values and chain.

part 2: MCMH

```r
set.seed(5)
df <- c(379, 299, 222, 145, 109, 95, 73, 59, 45, 30, 24, 12,
    4, 2, 0, 1, 1)
alpha <- 0.23
beta <- 0.25
mu <- 3.5
lambda <- 3.5
N <- 1000
chain <- matrix(nrow = N, ncol = 4)
chain[1, ] <- c(alpha, beta, mu, lambda)
minn = -0.5
maxx = 0.5
reject = 0
count = 0
pi_i <- function(alpha, beta, mu, lambda, i) {
    if (i == 0)
        return(alpha + beta * exp(-mu) + (1 - alpha - beta) *
            exp(-lambda)) else return(beta * (mu^i) * exp(-mu) + (1 - alpha - beta) *
        (lambda^i) * exp(-lambda))
}
loglike <- function(alpha, beta, mu, lambda) {
    sum.out <- 0
    alpha = exp(alpha)/(1 + exp(alpha))
    beta = exp(beta)/(1 + exp(beta))
    mu = exp(mu)
    lambda = exp(lambda)
    for (i in 1:length(df)) {
        sum.out <- sum.out + df[i] * (log(pi_i(alpha, beta, mu,
            lambda, i - 1)) - log(factorial(i - 1)))
    }
    return(sum.out)
}
options(warn = -1)
for (i in 2:N) {
    alpha.star <- chain[i - 1, 1] + runif(1, minn, maxx)
    while (alpha.star < 0 || alpha.star > 1) {
        alpha.star <- chain[i - 1, 1] + runif(1, minn, maxx)
    }
    beta.star <- chain[i - 1, 2] + runif(1, minn, maxx)
    while (beta.star < 0 || beta.star > 1) {
        beta.star <- chain[i - 1, 2] + runif(1, minn, maxx)
    }
    mu.star <- chain[i - 1, 3] + runif(1, -0.1, 0.1)
    lambda.star <- chain[i - 1, 4] + runif(1, -0.1, 0.1)
    ratio <- exp(loglike(alpha.star, beta.star, mu.star, lambda.star) -
        loglike(chain[i - 1, 1], chain[i - 1, 2], chain[i - 1,
            3], chain[i - 1, 4]))
    options(warn = -1)
    if (is.nan(ratio)) {
        ratio = 0
    }
    if (runif(1) < ratio) {
        chain[i, ] <- c(alpha.star, beta.star, mu.star, lambda.star)
    } else {
        chain[i, ] <- chain[i - 1, ]
        reject <- reject + 1
    }
    count = count + 1
}
options(warn = -1)
c(mean(chain[, 1]), mean(chain[, 2]), mean(chain[, 3]), mean(chain[,
    4]))
```

```
## [1] 0.9015040 0.9565949 1.3769800 3.9394893
```

```r
print("Rejection Rate:")
```

```
## [1] "Rejection Rate:"
```

```r
100 * (reject/count)
```

```
## [1] 95.8959
```

```r
# Part d: Bootstrap the variances Estimate th estimate of the
# variances (1/B-1)*sum(var(sample median) - mL), mL = 1/B *
# sum(var(sample median))
B <- 1000
y.a <- chain[, 1]
y.b <- chain[, 2]
y.m <- chain[, 3]
y.l <- chain[, 4]
mN.a <- rep(0, B)
mN.b <- rep(0, B)
mN.m <- rep(0, B)
mN.l <- rep(0, B)
for (k in 1:B) {
    samp.a <- y.a[sample(1:N, replace = TRUE)]
    samp.b <- y.b[sample(1:N, replace = TRUE)]
    samp.m <- y.m[sample(1:N, replace = TRUE)]
    samp.l <- y.l[sample(1:N, replace = TRUE)]
    mN.a[k] <- mean(samp.a)
    mN.b[k] <- mean(samp.b)
    mN.m[k] <- mean(samp.m)
    mN.l[k] <- mean(samp.l)
}
var.boot <- c((1/(B - 1)) * sum((mN.a - mean(mN.a))^2), (1/(B -
    1)) * sum((mN.b - mean(mN.b))^2), (1/(B - 1)) * sum((mN.m -
    mean(mN.m))^2), (1/(B - 1)) * sum((mN.l - mean(mN.l))^2))

cat("Variance Estimate from bootstrap", var.boot, "\n")
```

```
## Variance Estimate from bootstrap 1.755429e-05 1.561581e-05 0.0001500258 3.185464e-06
```

The variance estimates from the Metropolis algorithm are smaller than the standard error from part 1c. This could be becuase the MCMC is more accurate in estimating parameters than the EM algorithm. But they could also be smaller becuase the chain did not explore a large sample space and instead stayed in a smaller space during each chain iteration. This can happen when the proposal distribution has a realatively small range or small sigma value. While it might converge faster, the estimates will be less accurate.

```r
# Part e: Bootstrap 95% CI
mean.a <- rep(0, B)
mean.b <- rep(0, B)
mean.m <- rep(0, B)
mean.l <- rep(0, B)
t.a <- rep(0, B)
t.b <- rep(0, B)
t.m <- rep(0, B)
t.l <- rep(0, B)
samp.a <- rep(0, N)
samp.b <- rep(0, N)
samp.m <- rep(0, N)
samp.l <- rep(0, N)
obs.a <- mean(chain[, 1])
obs.b <- mean(chain[, 2])
obs.m <- mean(chain[, 3])
obs.l <- mean(chain[, 4])
for (i in 1:B) {
    samp.a <- y.a[sample(1:N, replace = TRUE)]
    samp.b <- y.b[sample(1:N, replace = TRUE)]
    samp.m <- y.m[sample(1:N, replace = TRUE)]
    samp.l <- y.l[sample(1:N, replace = TRUE)]
    mean.a[i] <- mean(samp.a)
    mean.b[i] <- mean(samp.b)
    mean.m[i] <- mean(samp.m)
    mean.l[i] <- mean(samp.l)
    var.a <- (1/mean.a[i]) * (sd(samp.a)^2)
    var.b <- (1/mean.b[i]) * (sd(samp.b)^2)
    var.m <- (1/mean.m[i]) * (sd(samp.m)^2)
    var.l <- (1/mean.l[i]) * (sd(samp.l)^2)
    t.a[i] <- (mean.a[i] - obs.a)/(sqrt(var.a))
    t.b[i] <- (mean.b[i] - obs.b)/(sqrt(var.b))
    t.m[i] <- (mean.m[i] - obs.m)/(sqrt(var.m))
    t.l[i] <- (mean.l[i] - obs.l)/(sqrt(var.l))
}
```

```r
# bootstrap percentile
j <- (alpha/2) * B
k <- (1 - (alpha/2)) * B
mean.a <- sort(mean.a)
mean.b <- sort(mean.b)
mean.m <- sort(mean.m)
mean.l <- sort(mean.l)
marg.stats <- matrix(c(mean.a[j], mean.a[k], mean.b[j], mean.b[k],
    mean.m[j], mean.m[k], mean.l[j], mean.l[k]), ncol = 2, byrow = TRUE)
colnames(marg.stats) <- c("Lower", "Upper")
rownames(marg.stats) <- c("Alpha perc CI", "Beta perc CI", "Mu perc CI",
    "Lambda perc CI")
as.table(marg.stats)
```

```
##                    Lower     Upper
## Alpha perc CI  0.8962495 0.9063370
## Beta perc CI   0.9516110 0.9610925
## Mu perc CI     1.3627591 1.3922407
## Lambda perc CI 3.9371382 3.9415126
```

```r
# bootstrap t
a.t <- c(mean(mean.a) - quantile(t.a, 0.025, na.rm = TRUE) *
    sd(mean.a), (mean(mean.a) - quantile(t.a, 0.975, na.rm = TRUE) *
    sd(mean.a)))
b.t <- c(mean(mean.b) - quantile(t.b, 0.025, na.rm = TRUE) *
    sd(mean.b), (mean(mean.b) - quantile(t.b, 0.975, na.rm = TRUE) *
    sd(mean.b)))
m.t <- c(mean(mean.m) - quantile(t.m, 0.025, na.rm = TRUE) *
    sd(mean.m), (mean(mean.m) - quantile(t.m, 0.975, na.rm = TRUE) *
    sd(mean.m)))
l.t <- c(mean(mean.l) - quantile(t.l, 0.025, na.rm = TRUE) *
    sd(mean.l), (mean(mean.l) - quantile(t.l, 0.975, na.rm = TRUE) *
    sd(mean.l)))
print("Studentized t bootsrap")
```

```
## [1] "Studentized t bootsrap"
```

```r
a.t  #alpha
```

```
##      2.5%     97.5%
## 0.9016554 0.9011743
```

```r
b.t  #beta
```

```
##      2.5%     97.5%
## 0.9566930 0.9561976
```

```r
m.t  #mu
```

```
##     2.5%    97.5%
## 1.378680 1.376837
```

```
l.t  #lambda
```

```
##     2.5%    97.5%
## 3.939646 3.939166
```

```
# bootstrap percentile (check), student (check), BCAa 95% CI
library(boot)
```

```
##
## Attaching package: 'boot'
```

```
## The following object is masked from 'package:lattice':
##
##     melanoma
```

```
B <- 1000
y.a <- chain[, 1]
y.b <- chain[, 2]
y.m <- chain[, 3]
y.l <- chain[, 4]

mean.fun <- function(y.a, i) {
    m <- mean(y.a[i])
    n <- length(i)
    v <- (n - 1) * var(y.a[i])/n^2
    c(m, v)
}
a.boot <- boot(y.a, mean.fun, R = 1000)
boot.ci(a.boot, type = c("perc", "stud", "bca"))
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = a.boot, type = c("perc", "stud", "bca"))
##
## Intervals :
## Level    Studentized          Percentile            BCa
## 95%   ( 0.8935,  0.9087 )   ( 0.8936,  0.9087 )   ( 0.8932,  0.9084 )
## Calculations and Intervals on Original Scale
```

```
b.boot <- boot(y.b, mean.fun, R = 1000)
boot.ci(b.boot, type = c("perc", "stud", "bca"))
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = b.boot, type = c("perc", "stud", "bca"))
##
## Intervals :
```

```
## Level     Studentized          Percentile            BCa
## 95%   ( 0.9476,  0.9643 )   ( 0.9477,  0.9644 )   ( 0.9468,  0.9634 )
## Calculations and Intervals on Original Scale
```

```r
m.boot <- boot(y.m, mean.fun, R = 1000)
boot.ci(m.boot, type = c("perc", "stud", "bca"))
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = m.boot, type = c("perc", "stud", "bca"))
##
## Intervals :
## Level     Studentized          Percentile            BCa
## 95%   ( 1.354,  1.404 )   ( 1.354,  1.402 )   ( 1.354,  1.403 )
## Calculations and Intervals on Original Scale
```

```r
l.boot <- boot(y.l, mean.fun, R = 1000)
boot.ci(l.boot, type = c("perc", "stud", "bca"))
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = l.boot, type = c("perc", "stud", "bca"))
##
## Intervals :
## Level     Studentized          Percentile            BCa
## 95%   ( 3.935,  3.943 )   ( 3.936,  3.943 )   ( 3.935,  3.943 )
## Calculations and Intervals on Original Scale
```

The estimates from the Metropolis algorithm are slightly off from the EM algorithm estimates. While there is a high rejection rate that would be becuase I had to add a conditional to stop NaN variables entering (I debugged the whole thing and unfortunately that was the only solution that worked) which of course contributes to the high rejection rate. With that being said, MCMC algorithms, such as the Metropolis Sampler, can have a difficult time converging so typically additional steps need to be taken (such as reparamertization), making it harder to implement compared to the EM algorithm.

Moreover we do not see a large difference in the 95% Confidence intervals, though that can be because of the small amount of variance seen and the high rejection rate.

In statistics, typically the BCa and student t-test for 95% confidence intervals outperform the percentile bootstrap method to find the 95% confidence interval. It is hard to tell based on these three methods which is better, especially because of the high rejection rate and small variance calculated in part d (wouldn't expect to see a wide range of values).

part b: The Metropolis-Hastings Algorithm

```r
# Metropolis-Hastings Algorithm
set.seed(5)
```

```r
df <- c(379, 299, 222, 145, 109, 95, 73, 59, 45, 30, 24, 12,
    4, 2, 0, 1, 1)
alpha <- 0.23
beta <- 0.25
mu <- 3.5
lambda <- 3.5
N <- 1000
chain <- matrix(nrow = N, ncol = 4)
chain[1, ] <- c(alpha, beta, mu, lambda)
minn = -0.5
maxx = 0.5
reject = 0
pi_i <- function(alpha, beta, mu, lambda, i) {
    if (i == 0)
        return(alpha + beta * exp(-mu) + (1 - alpha - beta) *
            exp(-lambda)) else return(beta * (mu^i) * exp(-mu) + (1 - alpha - beta) *
        (lambda^i) * exp(-lambda))
}
loglike <- function(alpha, beta, mu, lambda) {
    sum.out <- 0
    alpha = exp(alpha)/(1 + exp(alpha))
    beta = exp(beta)/(1 + exp(beta))
    mu = exp(mu)
    lambda = exp(lambda)
    for (i in 1:length(df)) {
        sum.out <- sum.out + df[i] * (log(pi_i(alpha, beta, mu,
            lambda, i - 1)) - log(factorial(i - 1)))
    }
    return(sum.out)
}
options(warn = -1)
for (i in 2:N) {
    alpha.star <- chain[i - 1, 1] + rexp(1, 1)
    while (alpha.star < 0 || alpha.star > 1) {
        alpha.star <- chain[i - 1, 1] + rexp(1, 1)
    }
    beta.star <- chain[i - 1, 2] + rexp(1, 1)
    while (beta.star < 0 || beta.star > 1) {
        beta.star <- chain[i - 1, 2] + rexp(1, 1)
    }
    mu.star <- chain[i - 1, 3] + rexp(1, 1)
    lambda.star <- chain[i - 1, 4] + rexp(1, 1)
    ratio <- exp(loglike(alpha.star, beta.star, mu.star, lambda.star) -
        loglike(chain[i - 1, 1], chain[i - 1, 2], chain[i - 1,
            3], chain[i - 1, 4]))
    options(warn = -1)
    if (is.nan(ratio)) {
        ratio = 0
    }
    if (runif(1) < ratio) {
        chain[i, ] <- c(alpha.star, beta.star, mu.star, lambda.star)
    } else {
        chain[i, ] <- chain[i - 1, ]
        reject <- reject + 1
    }
}
options(warn = -1)
c(mean(chain[, 1]), mean(chain[, 2]), mean(chain[, 3]), mean(chain[,
    4]))
```

```
## [1] 0.8097695 0.6942583 3.5139465 4.4168919
```

```
print("Rejection Rate:")
```

```
## [1] "Rejection Rate:"
```

```
100 * (reject/N)
```

```
## [1] 99.7
```

part c: Comparing Proposal Distributions

```
# Metropolis-Hastings Algorithm
set.seed(5)
df <- c(379, 299, 222, 145, 109, 95, 73, 59, 45, 30, 24, 12,
    4, 2, 0, 1, 1)
alpha <- 0.23
beta <- 0.25
mu <- 3.5
lambda <- 3.5
N <- 1000
chain <- matrix(nrow = N, ncol = 4)
chain[1, ] <- c(alpha, beta, mu, lambda)
minn = -0.5
maxx = 0.5
reject = 0
pi_i <- function(alpha, beta, mu, lambda, i) {
    if (i == 0)
        return(alpha + beta * exp(-mu) + (1 - alpha - beta) *
            exp(-lambda)) else return(beta * (mu^i) * exp(-mu) + (1 - alpha - beta) *
        (lambda^i) * exp(-lambda))
}
loglike <- function(alpha, beta, mu, lambda) {
    sum.out <- 0
    alpha = exp(alpha)/(1 + exp(alpha))
    beta = exp(beta)/(1 + exp(beta))
    mu = exp(mu)
    lambda = exp(lambda)
    for (i in 1:length(df)) {
        sum.out <- sum.out + df[i] * (log(pi_i(alpha, beta, mu,
            lambda, i - 1)) - log(factorial(i - 1)))
    }
    return(sum.out)
}
options(warn = -1)
for (i in 2:N) {
    alpha.star <- rgamma(1, 2)
    while (alpha.star < 0 || alpha.star > 1) {
        alpha.star <- rgamma(1, 2)
    }
    beta.star <- rgamma(1, 2)
    while (beta.star < 0 || beta.star > 1) {
        beta.star <- rgamma(1, 2)
    }
    mu.star <- rgamma(1, 2)
    lambda.star <- rgamma(1, 2)
    ratio <- exp(loglike(alpha.star, beta.star, mu.star, lambda.star) -
        loglike(chain[i - 1, 1], chain[i - 1, 2], chain[i - 1,
            3], chain[i - 1, 4]))
    options(warn = -1)
    if (is.nan(ratio)) {
        ratio = 0
    }
    if (runif(1) < ratio) {
        chain[i, ] <- c(alpha.star, beta.star, mu.star, lambda.star)
    } else {
        chain[i, ] <- chain[i - 1, ]
        reject <- reject + 1
    }
}
options(warn = -1)
cat("Estimates from an exponential distribution", "\n", c(mean(chain[,
    1]), mean(chain[, 2]), mean(chain[, 3]), mean(chain[, 4])),
    "\n")
```

```
## Estimates from an exponential distribution
##  0.807985 0.8487188 1.314986 4.013694
```

```
print("Rejection Rate:")
```

```
## [1] "Rejection Rate:"
```

```
100 * (reject/N)
```

```
## [1] 99.6
```

```
# Metropolis-Hastings Algorithm
set.seed(5)
df <- c(379, 299, 222, 145, 109, 95, 73, 59, 45, 30, 24, 12,
    4, 2, 0, 1, 1)
alpha <- 0.23
beta <- 0.25
mu <- 3.5
lambda <- 3.5
```

```r
N <- 1000
chain <- matrix(nrow = N, ncol = 4)
chain[1, ] <- c(alpha, beta, mu, lambda)
minn = -0.5
maxx = 0.5
reject = 0
pi_i <- function(alpha, beta, mu, lambda, i) {
    if (i == 0)
        return(alpha + beta * exp(-mu) + (1 - alpha - beta) *
            exp(-lambda)) else return(beta * (mu^i) * exp(-mu) + (1 - alpha - beta) *
        (lambda^i) * exp(-lambda))
}
loglike <- function(alpha, beta, mu, lambda) {
    sum.out <- 0
    alpha = exp(alpha)/(1 + exp(alpha))
    beta = exp(beta)/(1 + exp(beta))
    mu = exp(mu)
    lambda = exp(lambda)
    for (i in 1:length(df)) {
        sum.out <- sum.out + df[i] * (log(pi_i(alpha, beta, mu,
            lambda, i - 1)) - log(factorial(i - 1)))
    }
    return(sum.out)
}
options(warn = -1)
for (i in 2:N) {
    alpha.star <- rchisq(1, 2)
    while (alpha.star < 0 || alpha.star > 1) {
        alpha.star <- rchisq(1, 2)
    }
    beta.star <- rchisq(1, 2)
    while (beta.star < 0 || beta.star > 1) {
        beta.star <- rchisq(1, 2)
    }
    mu.star <- rchisq(1, 2)
    lambda.star <- rchisq(1, 2)
    ratio <- exp(loglike(alpha.star, beta.star, mu.star, lambda.star) -
        loglike(chain[i - 1, 1], chain[i - 1, 2], chain[i - 1,
            3], chain[i - 1, 4]))
    options(warn = -1)
    if (is.nan(ratio)) {
        ratio = 0
    }
    if (runif(1) < ratio) {
        chain[i, ] <- c(alpha.star, beta.star, mu.star, lambda.star)
    } else {
        chain[i, ] <- chain[i - 1, ]
        reject <- reject + 1
    }
}
options(warn = -1)
cat("Estimates from an exponential distribution", "\n", c(mean(chain[,
    1]), mean(chain[, 2]), mean(chain[, 3]), mean(chain[, 4])),
    "\n")
```

```
## Estimates from an exponential distribution
##  0.6649768 0.7794704 1.369096 4.753034
```

```
print("Rejection Rate:")
```

```
## [1] "Rejection Rate:"
```

```
100 * (reject/N)
```

```
## [1] 99.5
```

Changing the proposal distribution should (in theory) help with convergence and parameter estimation, because unlike the random walk where one chooses a symmetrical distribution (even if the target is no where near that) M-H algorithm chooses a non-symmetric proposal distribution.

```
# Part d: Bootstrap the variances for MH
B <- 1000
y.a <- chain[, 1]
y.b <- chain[, 1]
y.m <- chain[, 1]
y.l <- chain[, 1]
mN.a <- rep(0, B)
mN.b <- rep(0, B)
mN.m <- rep(0, B)
mN.l <- rep(0, B)
for (k in 1:B) {
    samp.a <- y.a[sample(1:N, replace = TRUE)]
    samp.b <- y.b[sample(1:N, replace = TRUE)]
    samp.m <- y.m[sample(1:N, replace = TRUE)]
    samp.l <- y.l[sample(1:N, replace = TRUE)]
    mN.a[k] <- mean(samp.a)
    mN.b[k] <- mean(samp.b)
    mN.m[k] <- mean(samp.m)
    mN.l[k] <- mean(samp.l)
}
var.boot <- c((1/(B - 1)) * sum((mN.a - mean(mN.a))^2), (1/(B -
    1)) * sum((mN.b - mean(mN.b))^2), (1/(B - 1)) * sum((mN.m -
    mean(mN.m))^2), (1/(B - 1)) * sum((mN.l - mean(mN.l))^2))

cat("Variance Estimate from bootstrap", var.boot, "\n")
```

```
## Variance Estimate from bootstrap 3.051364e-05 2.93202e-05 3.093468e-05 2.731045e-05
```

```
# Part e: Bootstrap 95% CI bootstrap percentile, student,
# BCAa
mean.fun <- function(y.a, i) {
    m <- mean(y.a[i])
    n <- length(i)
```

```r
    v <- (n - 1) * var(y.a[i])/n^2
    c(m, v)
}
a.boot <- boot(y.a, mean.fun, R = 1000)
boot.ci(a.boot, type = c("perc", "stud", "bca"))
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = a.boot, type = c("perc", "stud", "bca"))
##
## Intervals :
## Level    Studentized          Percentile            BCa
## 95%   ( 0.6543,  0.6754 )   ( 0.6538,  0.6748 )   ( 0.6536,  0.6741 )
## Calculations and Intervals on Original Scale
```

```r
b.boot <- boot(y.b, mean.fun, R = 1000)
boot.ci(b.boot, type = c("perc", "stud", "bca"))
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = b.boot, type = c("perc", "stud", "bca"))
##
## Intervals :
## Level    Studentized          Percentile            BCa
## 95%   ( 0.6533,  0.6751 )   ( 0.6543,  0.6755 )   ( 0.6549,  0.6768 )
## Calculations and Intervals on Original Scale
```

```r
m.boot <- boot(y.m, mean.fun, R = 1000)
boot.ci(m.boot, type = c("perc", "stud", "bca"))
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = m.boot, type = c("perc", "stud", "bca"))
##
## Intervals :
## Level    Studentized          Percentile            BCa
## 95%   ( 0.6533,  0.6746 )   ( 0.6548,  0.6756 )   ( 0.6548,  0.6756 )
## Calculations and Intervals on Original Scale
```

```r
l.boot <- boot(y.l, mean.fun, R = 1000)
boot.ci(l.boot, type = c("perc", "stud", "bca"))
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
```

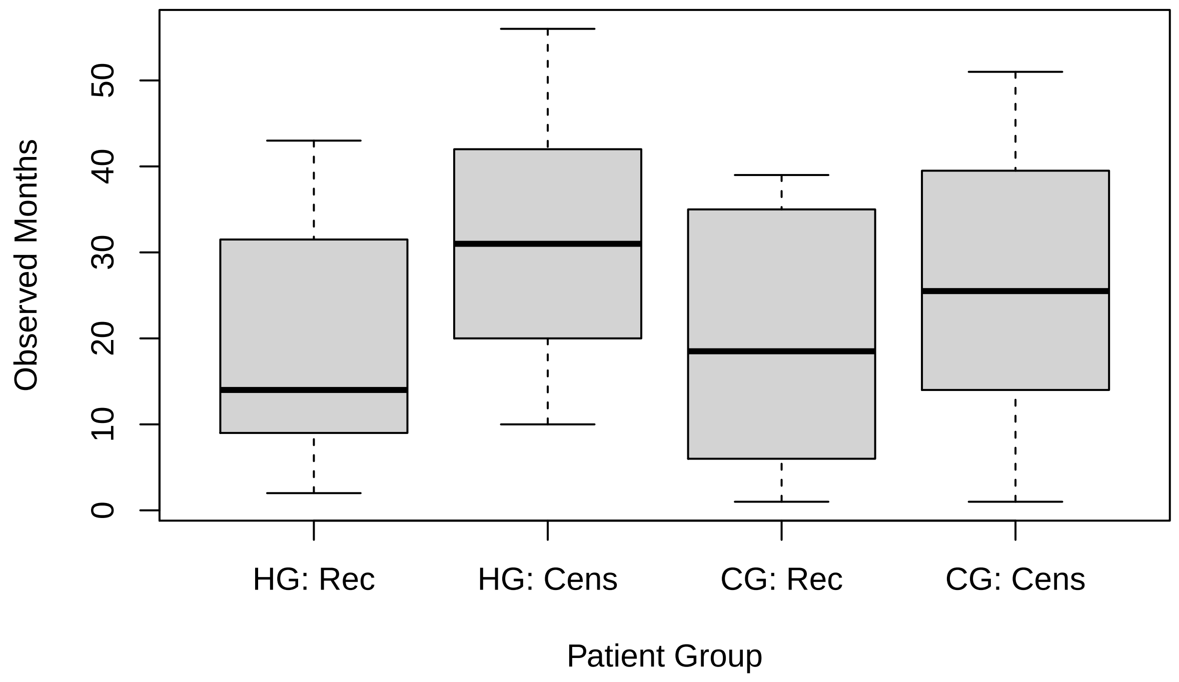
```
## CALL :
## boot.ci(boot.out = l.boot, type = c("perc", "stud", "bca"))
##
## Intervals :
## Level    Studentized          Percentile            BCa
## 95%   ( 0.6530,  0.6750 )   ( 0.6542,  0.6758 )   ( 0.6537,  0.6754 )
## Calculations and Intervals on Original Scale
```

Problem 3

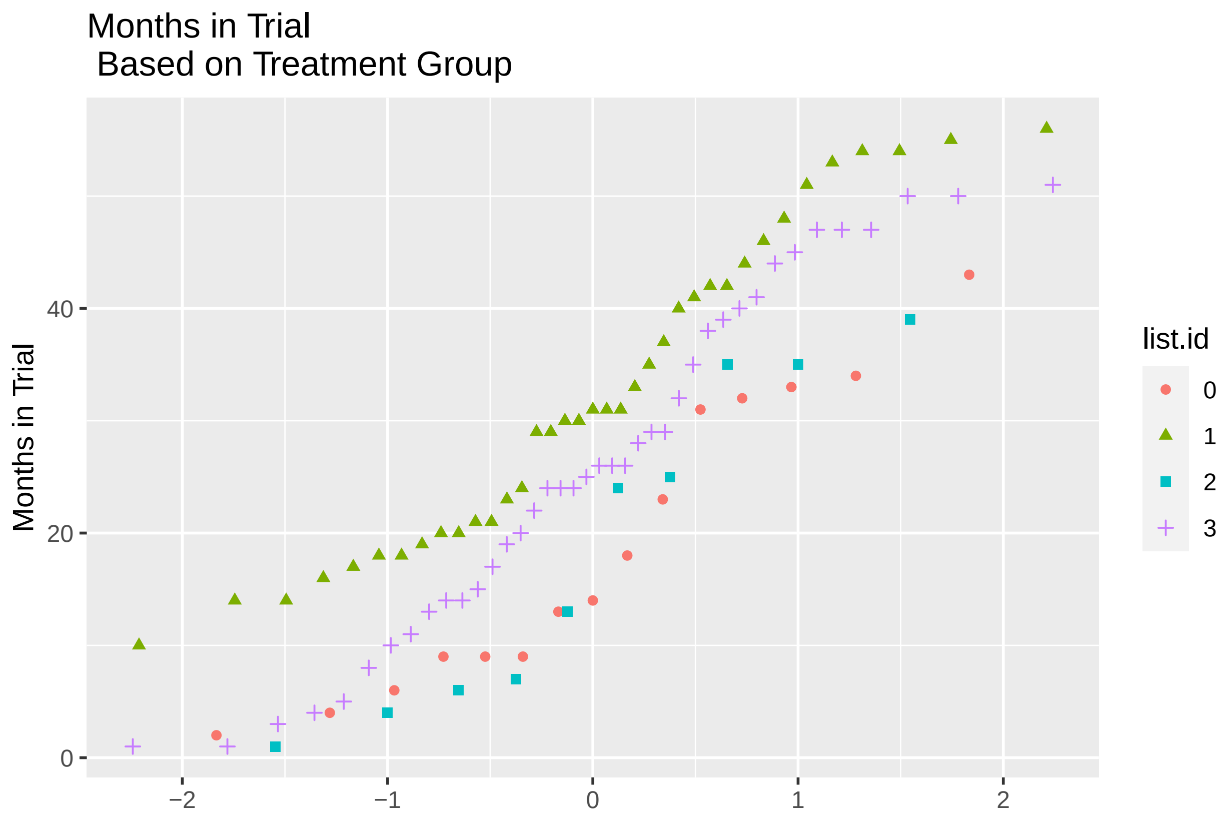
part a:

From the given data for the clinical trial, we can see from the box-and-whisker plot that the Hormone group with the censored time has the most patients out of the four groups. Not only that, but the means are more spread apart within the hormone group (difference of 13.68) compared to the control group (difference of 7.2). Based on the combined Normal QQ plot, the Control Group with Censored time most closely follows a normal distribution ("3" from list.id legend). The two from the hormone group of Recurrence and Censor times (0 and 1, respectively), do not closely align with a normal distribution.

```r
set.seed(12345)
# initialize data sets
hormone.rec <- c(2, 4, 6, 9, 9, 9, 13, 14, 18, 23, 31, 32, 33,
    34, 43)
lab.h.r <- rep(0, length(hormone.rec))
hormone.cens <- c(10, 14, 14, 16, 17, 18, 18, 19, 20, 20, 21,
    21, 23, 24, 29, 29, 30, 30, 31, 31, 31, 33, 35, 37, 40, 41,
    42, 42, 44, 46, 48, 51, 53, 54, 54, 55, 56)
lab.h.c <- rep(1, length(hormone.cens))
control.rec = c(1, 4, 6, 7, 13, 24, 25, 35, 35, 39)
lab.c.r <- rep(2, length(control.rec))
control.cens = c(1, 1, 3, 4, 5, 8, 10, 11, 13, 14, 14, 15, 17,
    19, 20, 22, 24, 24, 24, 25, 26, 26, 26, 28, 29, 29, 32, 35,
    38, 39, 40, 41, 44, 45, 47, 47, 47, 50, 50, 51)
lab.c.c <- rep(3, length(control.cens))
list.id <- c(lab.h.r, lab.h.c, lab.c.r, lab.c.c)
dat.df <- c(hormone.rec, hormone.cens, control.rec, control.cens)

cancer.dat <- data.frame(dat.df, list.id)
cancer.dat$list.id <- as.factor(cancer.dat$list.id)

# part a: Plots and quantiles
dat.list <- list(hormone.rec, hormone.cens, control.rec, control.cens)
boxplot(dat.list, range = 0, names = c("HG: Rec", "HG: Cens",
    "CG: Rec", "CG: Cens"), xlab = "Patient Group", ylab = "Observed Months")
```

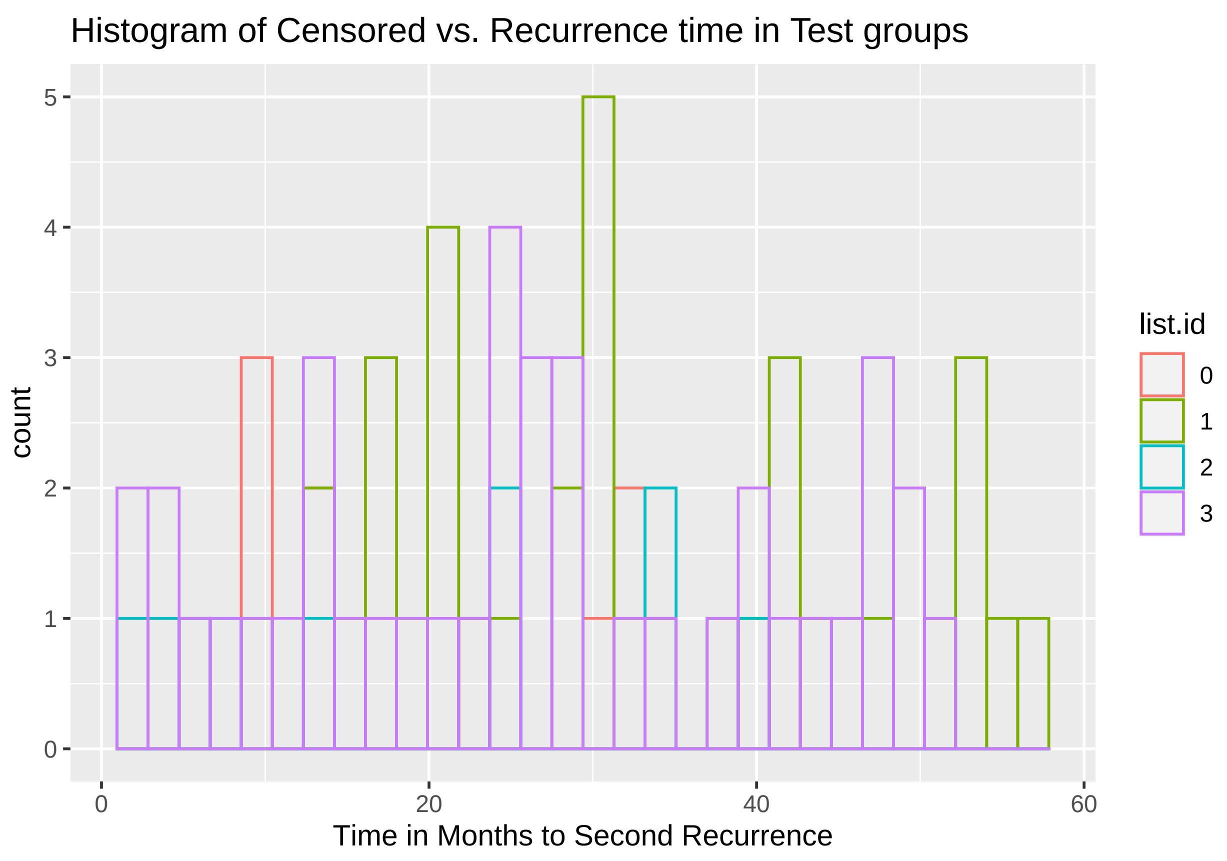


```
p <- qplot(sample = dat.df, data = cancer.dat, color = list.id,
    shape = list.id)
p + labs(title = "Months in Trial \n Based on Treatment Group",
    y = "Months in Trial")
```



```
ggplot(cancer.dat, aes(x = dat.df, fill = list.id, color = list.id)) +
    geom_histogram(position = "identity", fill = "transparent") +
    labs(title = "Histogram of Censored vs. Recurrence time in Test groups",
        x = "Time in Months to Second Recurrence")
```

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



part b:

Given the likelihood and prior, we can use those to calculate the conditional distributions of θ and τ. To do this we will first need to calculate the joint probability density function:

$$P(y|\theta,\tau) \propto L(\theta,\tau|y)f(\theta,\tau)$$

$$\pi(\theta,\tau|y) \propto \pi(\theta,\tau,y) = L(\theta,\tau|y)f(\theta,\tau)$$

$$P(\theta,\tau|y) = \theta^{\Sigma\delta_i^c+\Sigma\delta_i^H+2a}\tau^{\Sigma\delta_i^H+b}exp(-\theta(\Sigma x_i^C + c) - \tau\theta(\Sigma x_i^H + d))$$

With this we can calculate the conditional distributions for the parameters:

$$P(\tau|\theta,y) = \tau^{\Sigma\delta_i^H+b}exp(-\tau\theta(\Sigma x_i^H + d)) \propto Gamma(\tau|\Sigma\delta_i^H + b + 1, \Sigma x_i^C\Sigma x_i^H + c + d$$

$$P(\theta|\tau,y) = \theta^{\Sigma\delta_i^c+\Sigma\delta_i^H+a}exp(-\theta(\Sigma x_i^C+c)-\tau\theta(\Sigma x_i^H+d)) \propto Gamma(\theta|\Sigma\delta_i^c+\Sigma\delta_i^H+a+1, \Sigma x_i^C+\tau\Sigma x_i^H+c+d\tau$$

```r
# part c: Gibbs Sampler Initialize and prepare recurrence and
# censored data based on patient group
delta.h <- c(rep(1, length(hormone.rec)), rep(0, length(hormone.cens)))
delta.c <- c(rep(1, length(control.rec)), rep(0, length(control.cens)))
dat.hormone <- data.frame(x = c(hormone.rec, hormone.cens), delta.h)
dat.control <- data.frame(x = c(control.rec, control.cens), delta.c)
a = 3
b = 1
c = 60
```

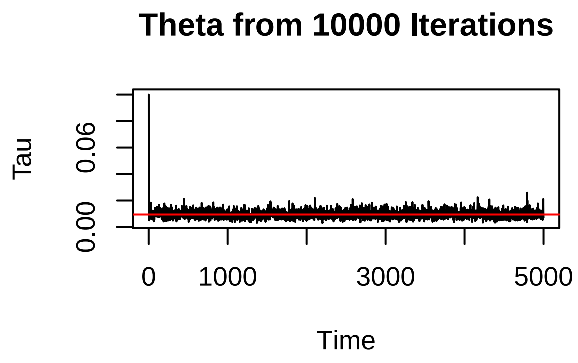
```r
d = 120

# FUNCTIONS
gibbs <- function(dat.control, dat.hormone, n, burn, hyperparameter) {
    mat <- matrix(ncol = 2, nrow = n)
    tau <- 0.1
    theta <- 0.1
    mat[1, ] <- c(theta, tau)
    a <- hyperparameter[1]
    b <- hyperparameter[2]
    c <- hyperparameter[3]
    d <- hyperparameter[4]
    for (i in 2:n) {
        tau <- mat[i - 1, 2]
        one.tau <- a + 1 + sum(dat.control$delta.c) + sum(dat.hormone$delta.h)
        two.tau <- sum(dat.control$x) + (tau * sum(dat.hormone$x)) +
            c + (d * tau)
        mat[i, 1] <- rgamma(1, one.tau, two.tau)
        theta <- mat[i, 1]
        one.theta <- b + 1 + sum(dat.hormone$delta.h)
        two.theta <- theta * sum(dat.hormone$x) + (theta * d)
        mat[i, 2] <- rgamma(1, one.theta, two.theta)
    }
    burn <- burn + 1
    return(mat[burn:n, ])
}

gibby <- gibbs(dat.control, dat.hormone, 1000, 100, c(3, 1, 60,
    120))
colMeans(gibby)
```

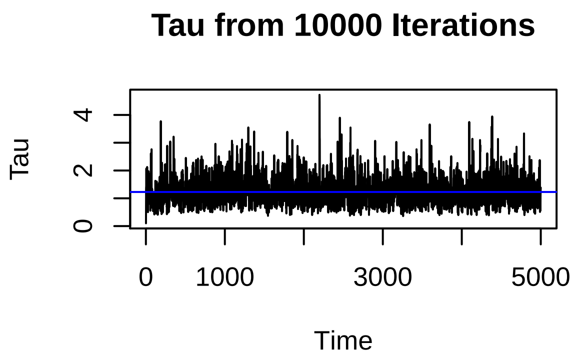
```
## [1] 0.009277404 1.253663033
```

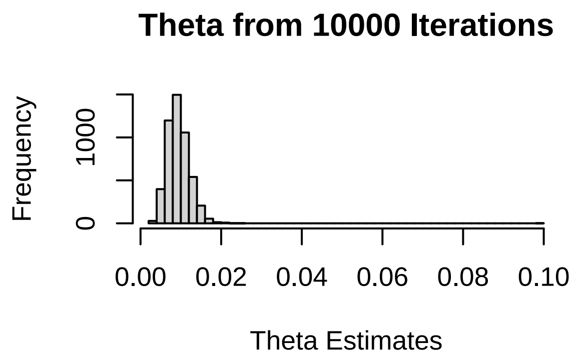
```r
gibbies <- gibbs(dat.control, dat.hormone, 5000, 0, c(3, 1, 60,
    120))
colMeans(gibbies)
```

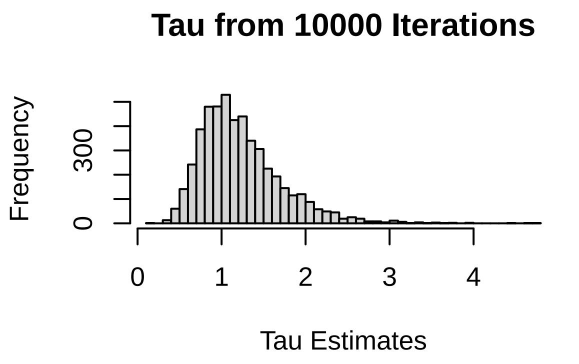
```
## [1] 0.009430104 1.227765240
```

```r
# Convergence diagnostics
par(mfrow = c(2, 2))
plot(ts(gibbies[, 1]), main = "Theta from 10000 Iterations",
    ylab = "Tau")
abline(h = mean(gibbies[, 1]), col = "red")
plot(ts(gibbies[, 2]), main = "Tau from 10000 Iterations", ylab = "Tau")
abline(h = mean(gibbies[, 2]), col = "blue")
hist(gibbies[, 1], 40, main = "Theta from 10000 Iterations",
    xlab = "Theta Estimates")
hist(gibbies[, 2], 40, main = "Tau from 10000 Iterations", xlab = "Tau Estimates")
```







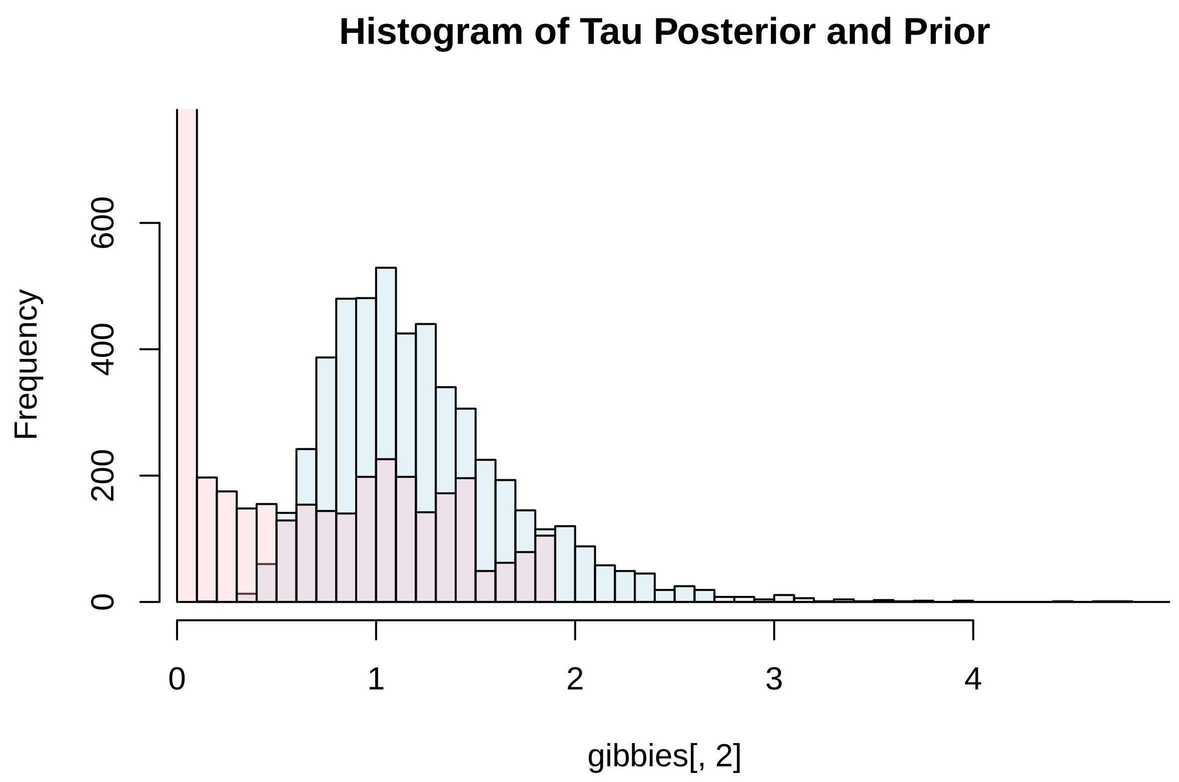


```
# part d: summary statistics marginal mean
marg.means <- colMeans(gibbies)
# sd
marg.sd <- c(sd(gibbies[, 1]), sd(gibbies[, 2]))
# 95% Confidence interval
marg.ci.theta <- CI(gibbies[, 1], ci = 0.95)
marg.ci.tau <- CI(gibbies[, 2], ci = 0.95)

marg.stats <- matrix(c(marg.means[1], marg.means[2], marg.sd[1],
    marg.sd[2], marg.ci.theta[3], marg.ci.tau[3], marg.ci.theta[1],
    marg.ci.tau[1]), ncol = 2, byrow = TRUE)
colnames(marg.stats) <- c("Theta", "Tau")
rownames(marg.stats) <- c("Marginal Mean", "Standard Deviation",
    "95% CI upper", "95% CI lower")
as.table(marg.stats)
```

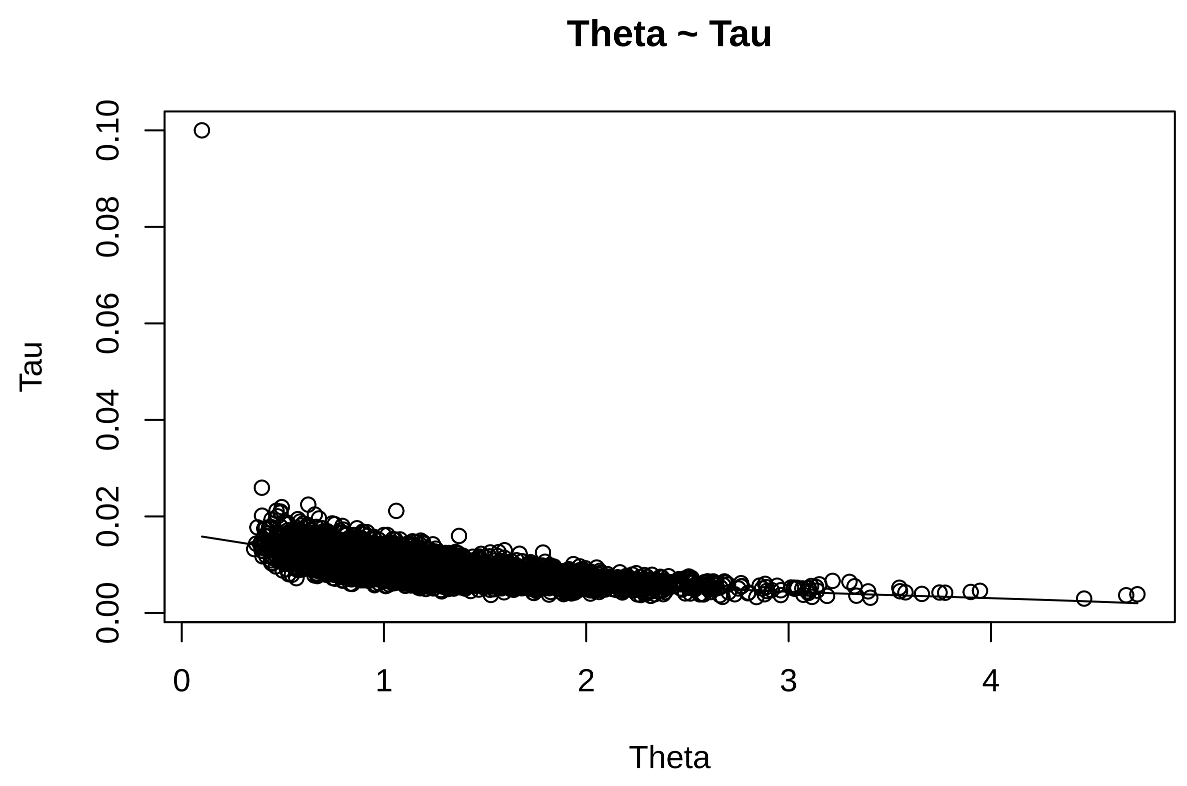
```
##                          Theta         Tau
## Marginal Mean      0.009430104 1.227765240
## Standard Deviation 0.002968410 0.484634152
## 95% CI upper       0.009347805 1.214328860
## 95% CI lower       0.009512402 1.241201620
```

```
# part e: graphing the prior and posterior distributions
# Posterior is the theta and tau estimates from above with
# the prior being a gamma distribution
```

```r
par(mfrow = c(1, 1))
c1 <- rgb(173, 216, 230, max = 255, alpha = 80, names = "lt.blue")
c2 <- rgb(255, 192, 203, max = 255, alpha = 80, names = "lt.pink")
theta.pos <- hist(gibbies[, 1], breaks = 40, plot = FALSE)
tau.pos <- hist(gibbies[, 2], breaks = 40, plot = FALSE)
theta.prior <- hist(dexp(quantile(gibbies[, 1])), breaks = 40,
    plot = FALSE)
tau.prior <- hist(dgamma(quantile(gibbies[, 2]), shape = 1 +
    1 + sum(dat.hormone$delta.h), rate = gibbies[, 1] * sum(dat.hormone$x) +
    (gibbies[, 1] * 120)), breaks = 120, plot = FALSE)

plot(tau.pos, col = c1, main = "Histogram of Tau Posterior and Prior",
    ylim = range(1:750))
plot(tau.prior, col = c2, add = TRUE)
```



```r
# part f: Interpreting result for the drug company Linear
# regression model to compare the theta and tau
scatter.smooth(gibbies[, 2], gibbies[, 1], xlab = "Theta", ylab = "Tau",
    main = "Theta ~ Tau")
```



```
cor(gibbies[, 2], gibbies[, 1])
```

```
## [1] -0.6475986
```

```
# inverse correlation, indicating that there is not

# part g: sensitivity analysis
hyp.half <- gibbs(dat.control, dat.hormone, 10000, 2500, c(1.5,
    2, 30, 60))
half.mean <- colMeans(hyp.half)
half.sd <- c(sd(hyp.half[, 1]), sd(hyp.half[, 2]))
half.ci.theta <- CI(hyp.half[, 1], ci = 0.95)
half.ci.tau <- CI(hyp.half[, 2], ci = 0.95)

hyp.dou <- gibbs(dat.control, dat.hormone, 10000, 2500, c(6,
    0.5, 120, 240))
dou.mean <- colMeans(hyp.dou)
dou.sd <- c(sd(hyp.dou[, 1]), sd(hyp.dou[, 2]))
dou.ci.theta <- CI(hyp.dou[, 1], ci = 0.95)
dou.ci.tau <- CI(hyp.dou[, 2], ci = 0.95)

# Summary statistics based on hyperparameters
hyp.table <- matrix(c(half.mean[1], half.mean[2], dou.mean[1],
    dou.mean[2], half.sd[1], half.sd[2], dou.sd[1], dou.sd[2],
    half.ci.theta[3], half.ci.tau[3], dou.ci.theta[3], dou.ci.tau[3],
```

```r
    half.ci.theta[1], half.ci.tau[1], dou.ci.theta[1], dou.ci.tau[1]),
    ncol = 4, byrow = TRUE)
colnames(hyp.table) <- c("Theta, Halved", "Tau, Halved", "Theta, Doubled",
    "Tau Doubled")
rownames(hyp.table) <- c("Marginal Mean", "Standard Deviation",
    "CI 95% Lower", "CI 95% Upper")
as.table(hyp.table)
```

```
##                    Theta, Halved Tau, Halved Theta, Doubled Tau Doubled
## Marginal Mean        0.007445801 1.774570157    0.011443877 0.897629263
## Standard Deviation   0.002442879 0.786136092    0.002895754 0.335451688
## CI 95% Lower         0.007390505 1.756775677    0.011378330 0.890036190
## CI 95% Upper         0.007501096 1.792364637    0.011509423 0.905222335
```

part f and g:

Based on the summary statistics produced in part d, I would say that the recurrence times are not statistically significant from the control group. This would mean that the clinical trial was not a huge success because the drug has very little affect on combating recurring episodes of breast cancer. We can see from the linear regression graph and the correlation output (-0.6475986) that the two parameters are inversely related.

For the hyperparameters, we can see from the table of summary statistics, that changing the hyperparameters affects our tau term more than the theta term. This can be seen in the 95% CI. This would be important to understand where these numbers come from and how to better estimate them before completing another clinical trial, because the hyperparameters could affect whether the drug is (statistically) effective.

Problem 4

```r
set.seed(475)

# part b: mean recurrence time
m.r.h <- mean(hormone.rec)
m.r.c <- mean(control.rec)
cat("Mean Recurrence Time for Hormone Group", m.r.h, "\n")
```

```
## Mean Recurrence Time for Hormone Group 18.66667
```

```r
cat("Mean Recurrence Time for Control Group", m.r.c, "\n")
```

```
## Mean Recurrence Time for Control Group 18.9
```

```r
tau.est <- gibbies[, 2]
theta.est <- gibbies[, 1]
better <- c(0.032429, 0.76652)
hormone.estimate <- rexp(length(theta.est), rate = mean(tau.est) *
    mean(theta.est))
control.estimate <- rexp(length(theta.est), rate = mean(theta.est))
cat("Mean Recurrence Time for Hormone Group", mean(hormone.estimate),
    "\n")
```

```
## Mean Recurrence Time for Hormone Group 87.97597
```

```r
cat("Mean Recurrence Time for Control Group", mean(control.estimate),
    "\n")
```

```
## Mean Recurrence Time for Control Group 104.0848
```

```r
# part c:
B = 1000  #Bootstrap Iteration
alpha <- 0.05
means.hormone <- rep(0, B)
means.control <- rep(0, B)
for (i in 1:B) {
    means.hormone[i] <- mean(sample(hormone.estimate, size = 1:length(hormone.estimate),
        replace = TRUE))
    means.control[i] <- mean(sample(control.estimate, size = 1:length(control.estimate),
        replace = TRUE))
}
j <- (alpha/2) * B
k <- (1 - (alpha/2)) * B
cat("95% CI for Hormone Group:", c(means.hormone[j], means.hormone[k]),
    "\n")
```

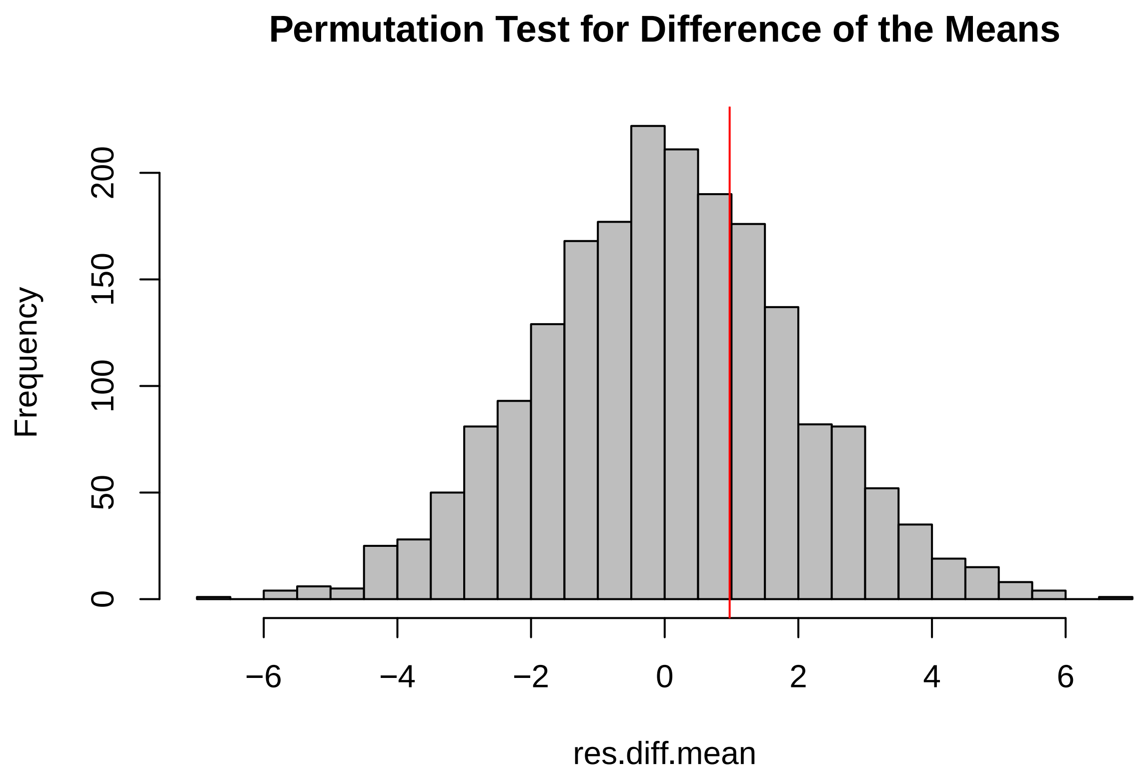
```
## 95% CI for Hormone Group: 123.3696 103.0919
```

```r
cat("95% CI for Control Group:", c(means.control[j], means.control[k]),
    "\n")
```

```
## 95% CI for Control Group: 68.94879 234.4083
```

Because the 95% confidence interval for the hormone group falls into the 95% confidence interval for the control group, we can say that the drug has little to no effect on stopping the recurrence of cancer.

```r
# part d: Permutation Tests Use permutation tests to show
# that there is no difference between the mean recurrence
# times Null hypothesis: the mean recurrence time between the
# two groups is equal
P <- 2000  #number of permutations
est <- c(control.estimate, hormone.estimate)
id <- c(rep(0, length(control.estimate)), rep(1, length(hormone.estimate)))
df <- data.frame(est, id)
theta.h <- mean(df[df$id == 1, 1])
theta.c <- mean(df[df$id == 0, 1])
res.diff.mean <- rep(0, P)
for (p in 1:P) {
    perm <- sample(nrow(df))
    dat <- transform(df, id = id[perm])
    theta.h <- mean(dat[dat$id == 1, "est"])
    theta.c <- mean(dat[dat$id == 0, "est"])
    res.diff.mean[p] <- theta.h - theta.c
}
obs.diff.mean <- theta.h - theta.c
hist(res.diff.mean, breaks = 25, col = "gray", main = "Permutation Test for Difference of the Means")
abline(v = obs.diff.mean, col = "red")
```



```
p <- mean(c(obs.diff.mean, res.diff.mean) >= obs.diff.mean)
p
```

```
## [1] 0.3103448
```

```
quantile(res.diff.mean, probs = c(0.025, 0.975))
```

```
##      2.5%     97.5%
## -3.800114  3.945974
```

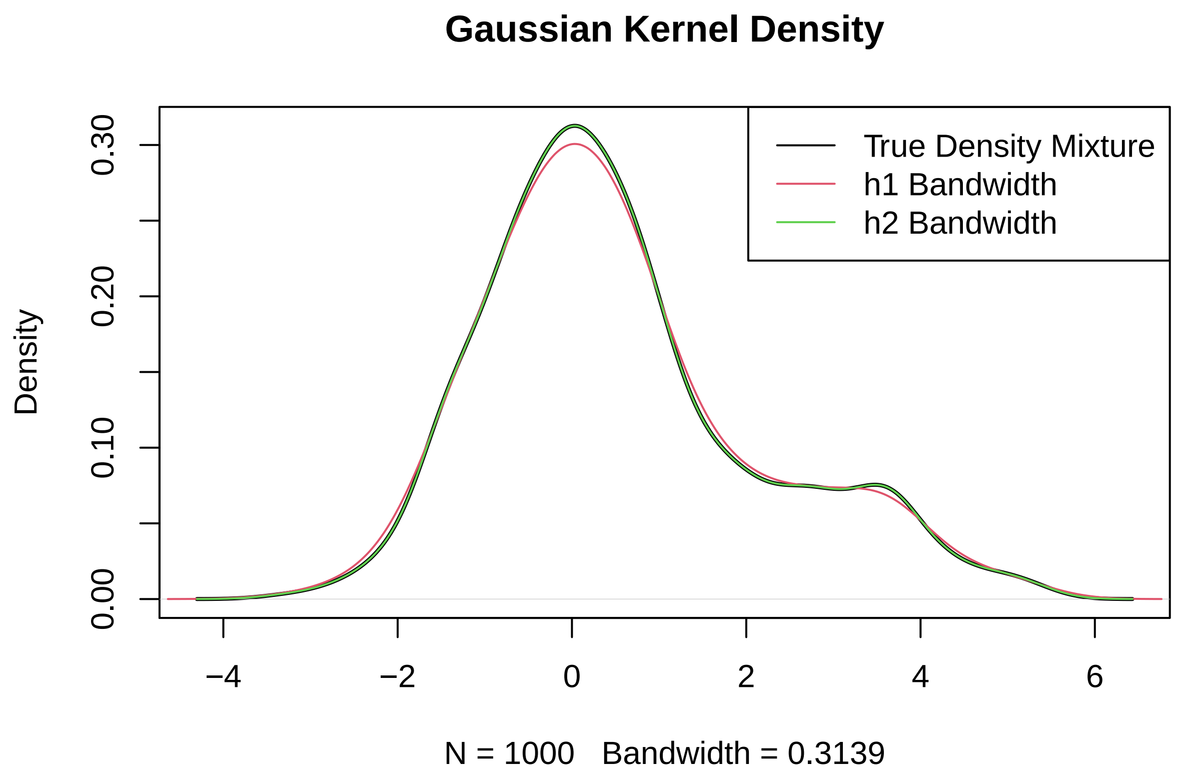
```
obs.diff.mean
```

```
## [1] 0.9722439
```

Based on the placement of the observed test statistic (mean) on the histogram, and the ALS (p-value) we must reject the alternative hypothesis that the means of the recurrence times are different and accept the null hypothesis. This supports previous conclusions made about the clinical trial data (i.e. the drug is not effective against recurring episodes of cancer).

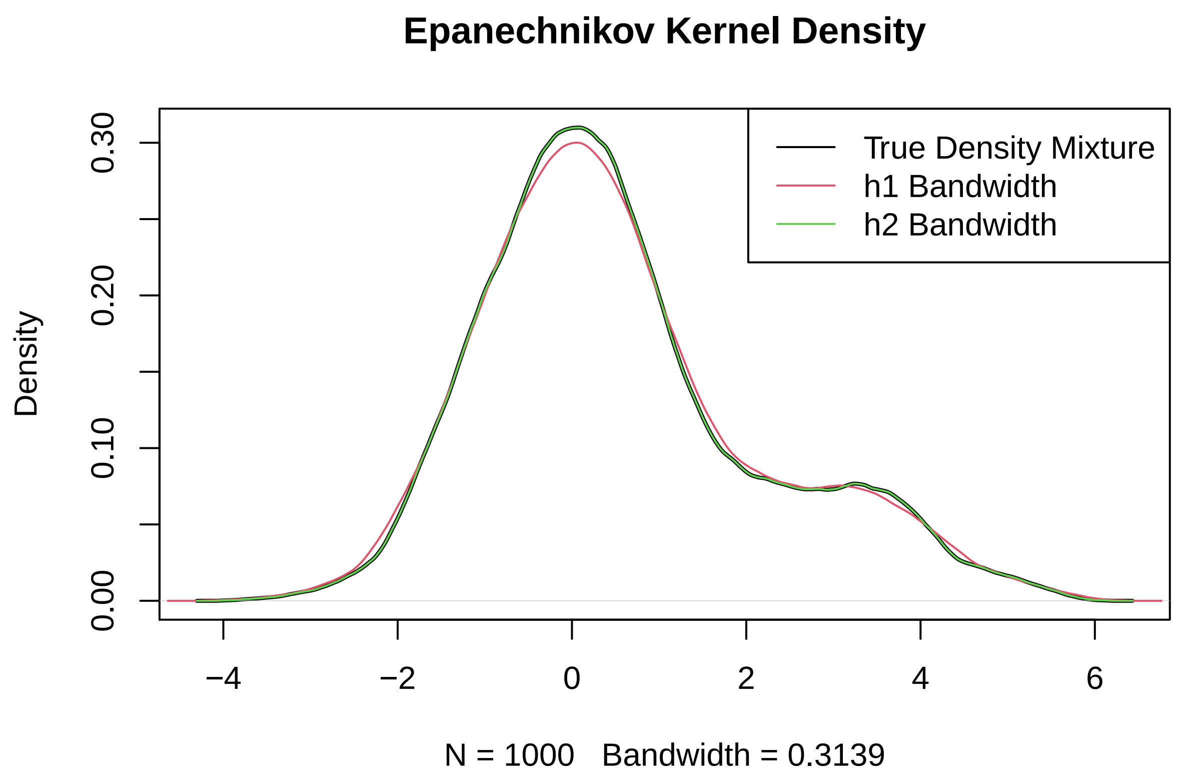
Problem 5

```r
set.seed(1000)
n <- 1000  #sample size
comp <- sample(1:2, prob = c(0.8, 0.2), size = n, replace = TRUE)
mu = c(0, 3)
stan.dev = sqrt(c(1, 1))
samp <- rnorm(n, mean = mu[comp], sd = stan.dev[comp])

h1 <- 1.06 * (n^(-1/5)) * sd(samp)
h2 <- 0.9 * (n^(-1/5)) * min(sd(samp), (IQR(samp))/1.34)

plot(density(samp), col = 1, main = "Gaussian Kernel Density",
    lwd = 2)  #true density estimate
lines(density(samp, bw = h1), col = 2)
lines(density(samp, bw = h2), col = 3)
legend("topright", c("True Density Mixture", "h1 Bandwidth",
    "h2 Bandwidth"), lty = 1, col = 1:3)
```

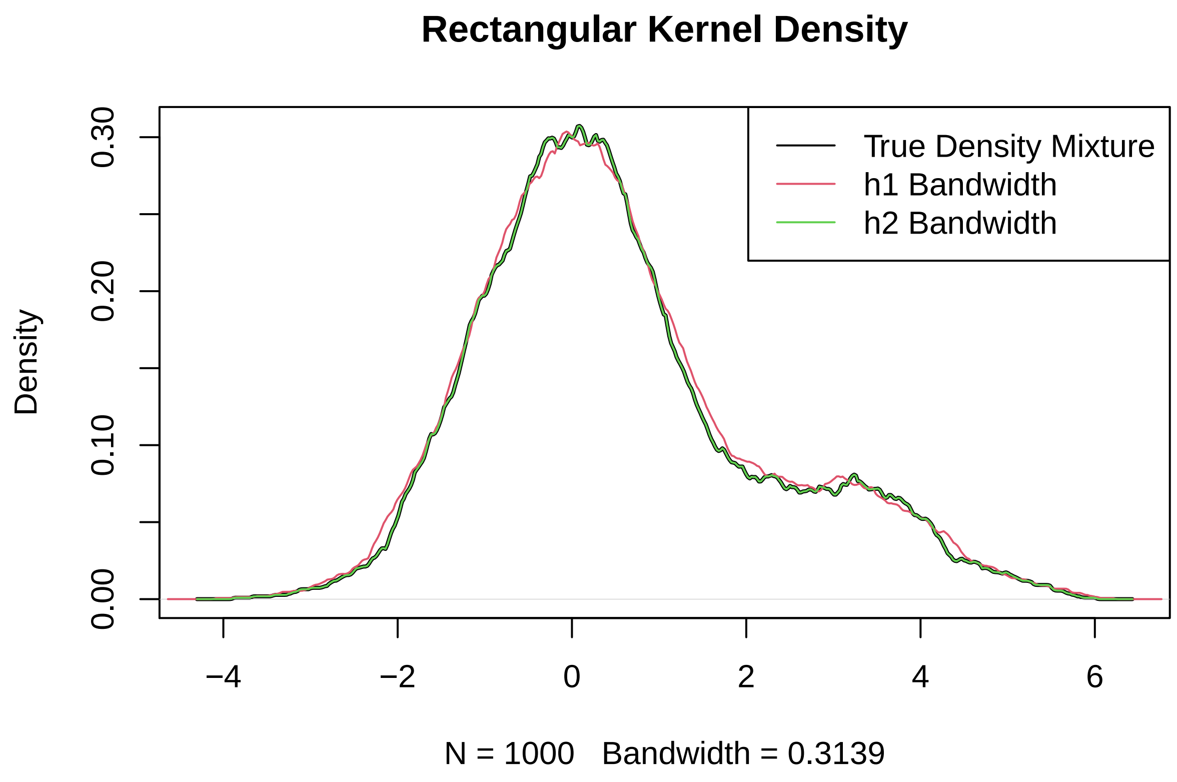


```r
# part b
plot(density(samp, kernel = "epanechnikov"), col = 1, main = "Epanechnikov Kernel Density",
    lwd = 2)  #true density estimate
lines(density(samp, bw = h1, kernel = "epanechnikov"), col = 2)
lines(density(samp, bw = h2, kernel = "epanechnikov"), col = 3)
```

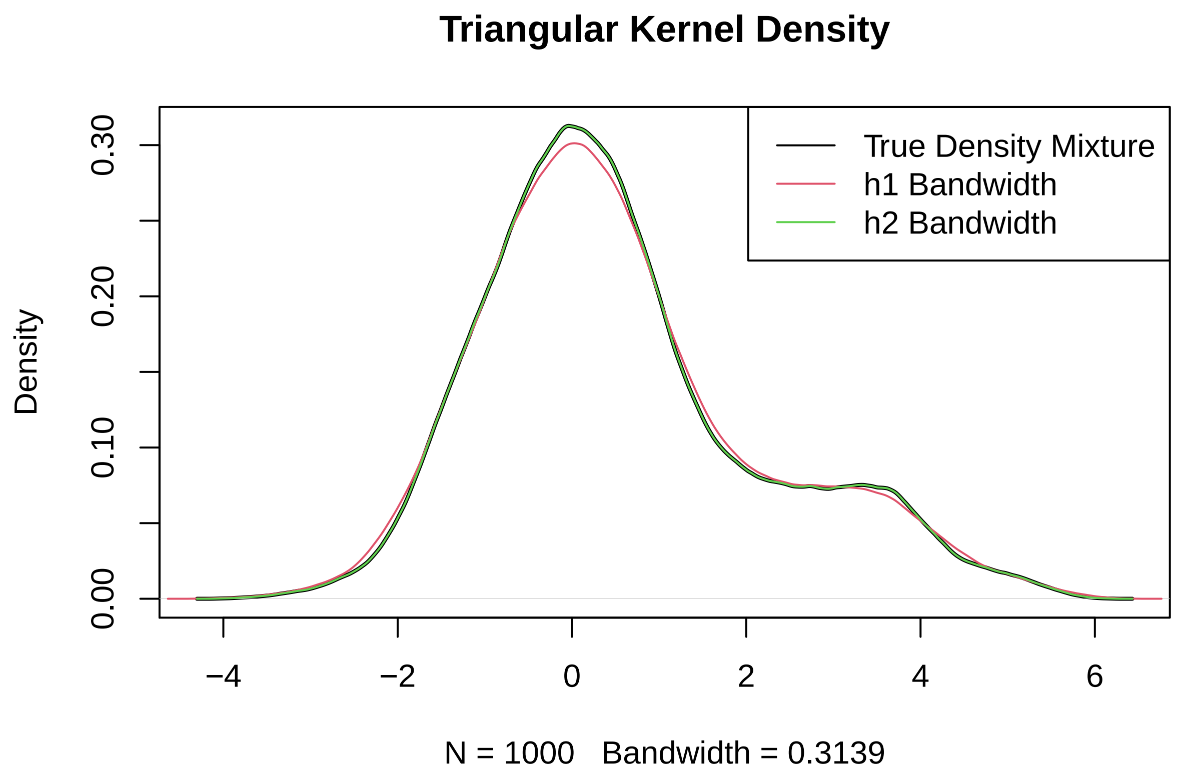
```r
legend("topright", c("True Density Mixture", "h1 Bandwidth",
    "h2 Bandwidth"), lty = 1, col = 1:3)
```



```r
plot(density(samp, kernel = "rectangular"), col = 1, main = "Rectangular Kernel Density",
    lwd = 2)  #true density estimate
lines(density(samp, bw = h1, kernel = "rectangular"), col = 2)
lines(density(samp, bw = h2, kernel = "rectangular"), col = 3)
legend("topright", c("True Density Mixture", "h1 Bandwidth",
    "h2 Bandwidth"), lty = 1, col = 1:3)
```



```r
plot(density(samp, kernel = "triangular"), col = 1, main = "Triangular Kernel Density",
    lwd = 2)  #true density estimate
lines(density(samp, bw = h1, kernel = "triangular"), col = 2)
lines(density(samp, bw = h2, kernel = "triangular"), col = 3)
legend("topright", c("True Density Mixture", "h1 Bandwidth",
    "h2 Bandwidth"), lty = 1, col = 1:3)
```



Based on the plots from above: The h2 bandwidth is a better smoothing parameter compared to using the h1 bandwidth because the h2 bandwidth plot fits over the true density mixture plots. Moreover, this shows that while changing the kernel type will change the shape of the mixture plot, the h2 bandwidth still fits over the true density mixture better than the h1 bandwidth plot. The h1 bandwidth plot is hardly affected when changing the kernel.