Assignment 6

STAT34800

Seung Chul Lee

Problem A.

i) Simulation Function

To reiterate, the setting we have is

$$D_{ij} \mid \beta, \sigma \sim N(\beta_j, \sigma_j^2), \;\; \beta_j \overset{iid}{\sim} \pi_0 \delta_0 + (1-\pi_0) N(0, \sigma_b^2)$$

assuming $\sigma_j = 1, ^\forall j = 1, \dots, m$. As instructed, I work with $n = 10, \; m = 1000$. I now write the R function to simulate D under this model.

```r
#' @param pi0 a scalar value for mixture proportion of delta component
#' @param sigmab a positive scalar value for SD of normal component
#' @return a list of matrix D, the individual effects, and
#' vector betas, the true treatment effect means
simul_trt = function(pi0, sigmab){
  betas = rep(NA, 1000)
  D = matrix(nrow = 10, ncol = 1000)

  for(i in 1:1000){
    betas[i] = ifelse(runif(1) < pi0, 0, rnorm(1, 0, sigmab))
    D[ , i] = rnorm(10, betas[i], 1)
  }
  return(list(D = D, betas = betas))
}
```

ii) p Value Function

Note that, since D_{ij} follows a normal distribution, we can construct a two-sided Z test statistic as

$$Z_j = \frac{\bar{D}_j}{1/\sqrt{n}} \sim N(0,1), \; \bar{D}_j = \frac{1}{n}\sum_{i=1}^{n} D_{ij}$$

The testing rule will be as follows:

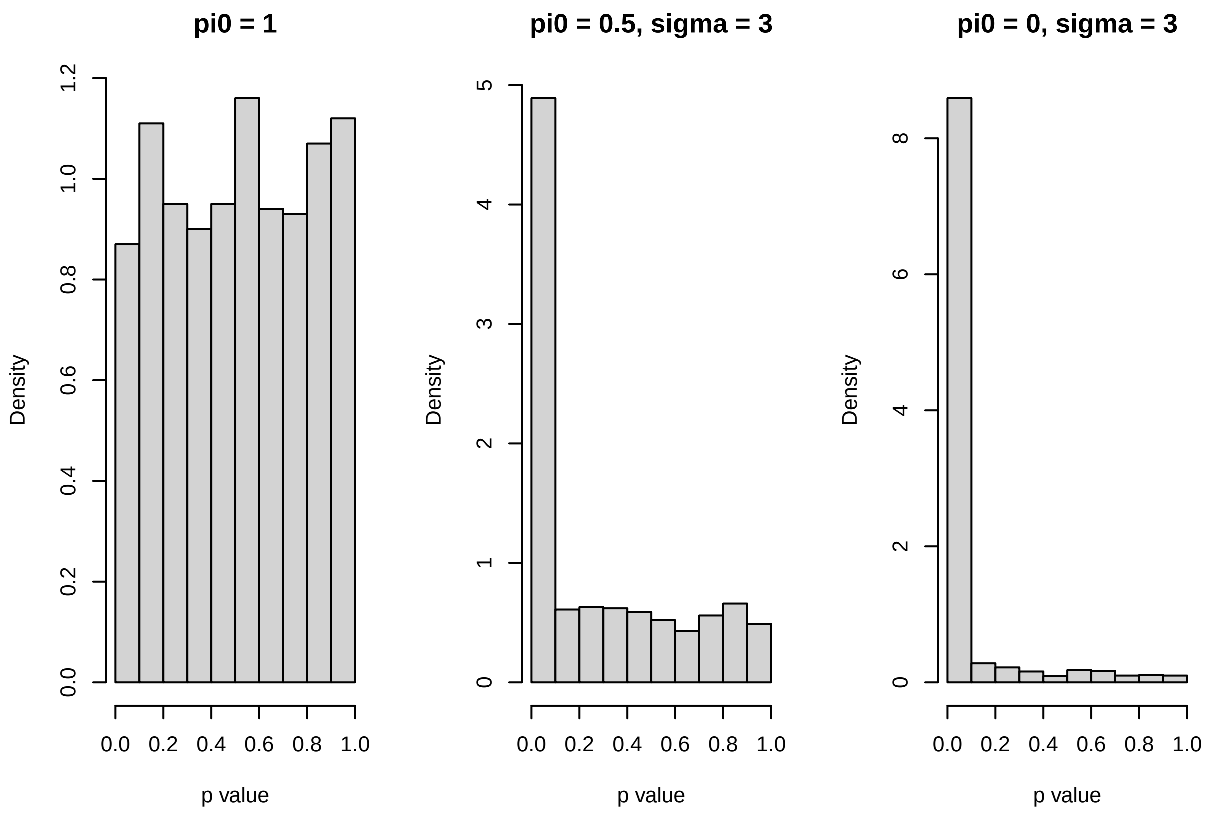
$$\text{Reject } H_j = 0 \text{ if } \; \mathbb{P}\left(Z > |Z_j|\right) < \alpha$$

where α is a significance level. Hence, my p value is $\mathbb{P}\left(Z > |Z_j|\right)$. I now write the R function to calculate this quantity.

```r
pval = function(D){
  return(2*pnorm(abs(colMeans(D)*sqrt(10)), lower.tail = F))
}
```

I now simulate, as instructed, three different samples with 1) $\pi_0 = 1$, 2) $\pi_0 = 0.5$ and $\sigma_b = 3$, and 3) $\pi_0 = 0$ and $\sigma_b = 3$.

```r
set.seed(1234)
sim1 = pval(simul_trt(1, 0)$D) # arbitrary value for sigmab, since not used
sim2 = pval(simul_trt(0.5, 3)$D)
sim3 = pval(simul_trt(0, 3)$D)

par(mfrow = c(1, 3))
hist(sim1, main = "pi0 = 1", xlab = "p value", probability = T)
hist(sim2, main = "pi0 = 0.5, sigma = 3", xlab = "p value", probability = T)
hist(sim3, main = "pi0 = 0, sigma = 3", xlab = "p value", probability = T)
```



I find that, when the null is true, the p values have a distribution close to uniform, consistent with what we have covered in class. When the mixture proportion is half and half, I confirm that there is a concentration of values near 0 (presumably the alternative distribution) and a uniform-like distribution for larger values (presumably the null component). When the alternative is true for all values, there is a much larger concentration around zero and very little mass for the larger values. It is interesting to note that there still is some high p values even when there are no observations from the null distribution.

iii) Benjamini-Hochberg

I define the function `fdr_control_bh()` to calculate false discovery rate using Benjamini-Hochberg rule.

```r
fdr_control_bh = function(p, alpha){
  alpha2 = c(1:1000)*alpha/1000
  p_ = p[order(p)]
  ind = suppressWarnings(max(which(p_ <= alpha2)))
  if(ind == -Inf){
    return(rep(0, 1000))
  }else{
    return(as.numeric(p <= p_[ind]))
  }
}
```

iv) FDR Function

I define a function `emp_fdr` that calculates the empirical false discovery rate given the indicators of rejected hypotheses using Benjamini-Hochberg rule (i.e., the output of `fdr_control_bh`) and ground truth values for β_j.

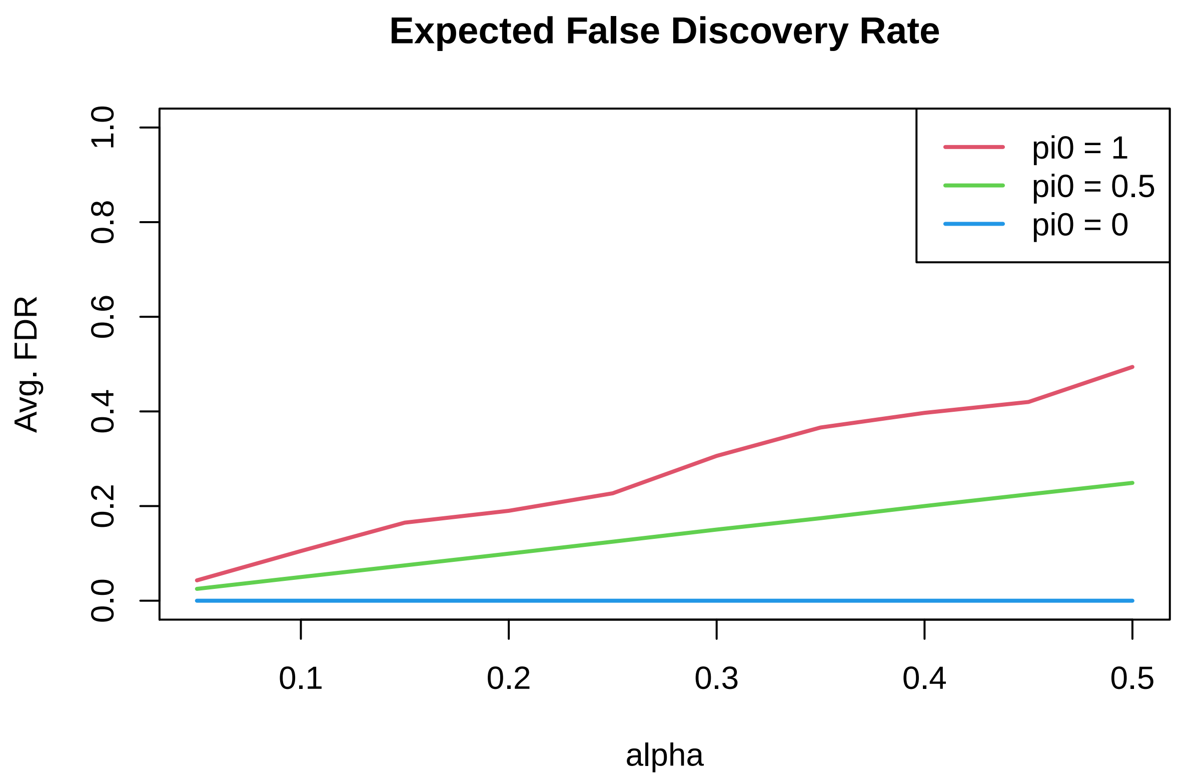
```r
emp_fdr = function(rej, truth){
  ind = which(truth == 0)
  R = sum(rej)
  fdr = ifelse(R == 0, 0, sum(rej[ind])/R)
  return(list(fdr = fdr, R = R))
}
```

v) Simulation Study

I construct a function that, given the number of iterations and values for α, π_0 and σ_b, simulates the data `niter` many times and calculates the empirical expected false discovery rate. For αs from 0.05 to 0.5 increasing with a step size of 0.05, I run a 1,000 trials using each value of α to obtain an empirical estimate of the expected FDR.

```r
fdr_est = function(niter, alpha, pi0, sigmab){
  fdr = rep(NA, niter)

  for(i in 1:niter){
    run = simul_trt(pi0 = pi0, sigmab = sigmab)
    truth = run$betas
    D = run$D

    rej = fdr_control_bh(pval(D), alpha = alpha)

    fdr[i] = emp_fdr(rej, truth)$fdr
  }
  return(mean(fdr))
}

set.seed(123)
alphas = seq(0.05, 0.5, by = 0.05)
case1 = rep(NA, length(alphas))
for(i in 1:length(alphas)){
  case1[i] = fdr_est(1000, alphas[i], 1, 1)
}
```

```r
case2 = rep(NA, length(alphas))
for(i in 1:length(alphas)){
  case2[i] = fdr_est(1000, alphas[i], 0.5, 3)
}

case3 = rep(NA, length(alphas))
for(i in 1:length(alphas)){
  case3[i] = fdr_est(1000, alphas[i], 0, 3)
}

plot(alphas, case1, type = "l", ylim = c(0, 1), lwd = 2, col = 2,
     main = "Expected False Discovery Rate", xlab = "alpha",
     ylab = "Avg. FDR")
lines(alphas, case2, lwd = 2, col = 3)
lines(alphas, case3, lwd = 2, col = 4)
legend("topright", legend = c("pi0 = 1", "pi0 = 0.5", "pi0 = 0"),
       col = c(2, 3, 4), lwd = c(2, 2, 2))
```



The case when $\pi_0 = 0$ is unsurprisingly zero all the time, since there cannot be false discovery given all β_j are from the alternative density. When $\pi_0 = 0.5$, I observe a somewhat modest, monotonic increase in expected false discovery rate with larger values of α. For the case with $\pi_0 = 1$, any discovery will be false. Hence, consistent with this, the false discovery rate increases, also somewhat monotonically, at a higher rate with the larger values of α.

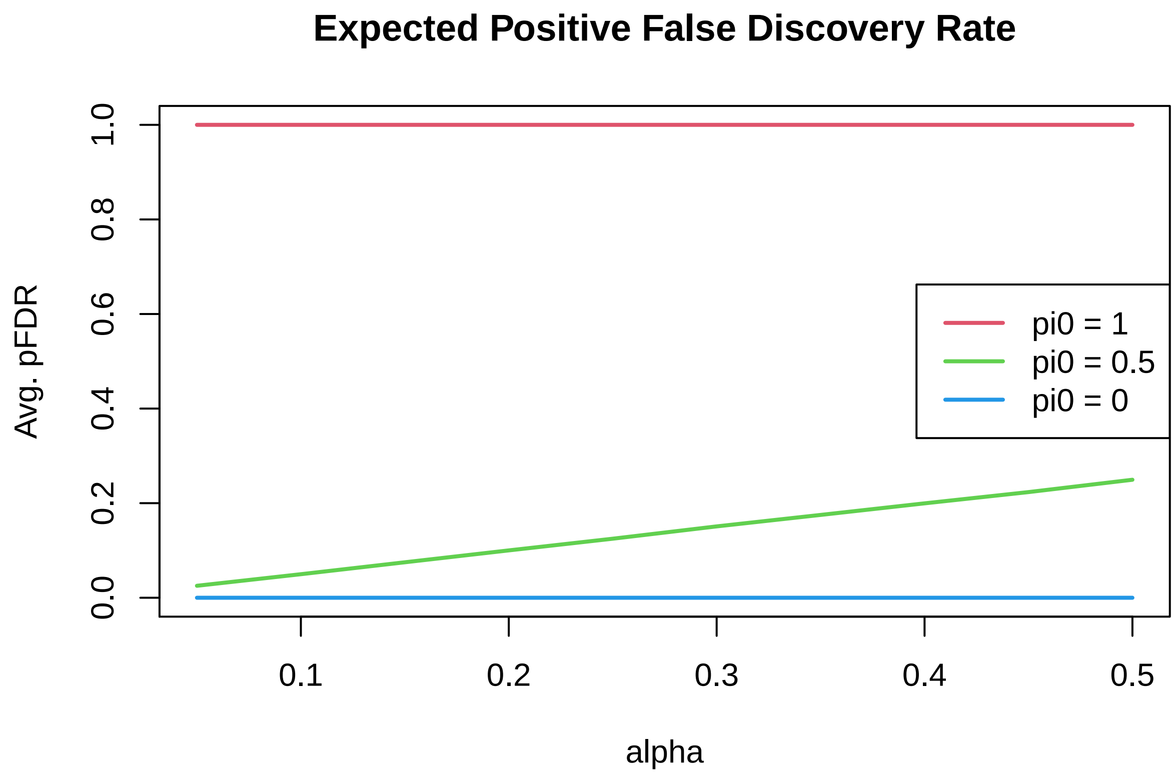
vi) Repeat with pFDR

I tweak the function `fdr_est()` to calculate pFDR instead.

```r
pfdr_est = function(niter, alpha, pi0, sigmab){
  fdr = rep(NA, niter)
  r = rep(NA, niter)

  for(i in 1:niter){
    run = simul_trt(pi0 = pi0, sigmab = sigmab)
    truth = run$betas
    D = run$D

    rej = fdr_control_bh(pval(D), alpha = alpha)

    temp = emp_fdr(rej, truth)
    fdr[i] = temp$fdr
    r[i] = temp$R
  }
  pfdr = ifelse(sum(r) > 0, mean(fdr[r != 0]), 0)
  return(pfdr)
}
```

I now run the function again to estimate the pFDR.

```r
set.seed(1234)
case1_p = rep(NA, length(alphas))
for(i in 1:length(alphas)){
  case1_p[i] = pfdr_est(1000, alphas[i], 1, 1)
}

case2_p = rep(NA, length(alphas))
for(i in 1:length(alphas)){
  case2_p[i] = pfdr_est(1000, alphas[i], 0.5, 3)
}

case3_p = rep(NA, length(alphas))
for(i in 1:length(alphas)){
  case3_p[i] = pfdr_est(1000, alphas[i], 0, 3)
}

plot(alphas, case1_p, type = "l", ylim = c(0, 1), lwd = 2, col = 2,
     main = "Expected Positive False Discovery Rate", xlab = "alpha",
     ylab = "Avg. pFDR")
lines(alphas, case2_p, lwd = 2, col = 3)
lines(alphas, case3_p, lwd = 2, col = 4)
legend("right", legend = c("pi0 = 1", "pi0 = 0.5", "pi0 = 0"),
       col = c(2, 3, 4), lwd = c(2, 2, 2))
```



The pFDR for the case with $\pi_0 = 1$ is always 1. This is because all of the discoveries are true and thus we have $V/R = 1$. For the other cases, the probability $\mathbb{P}(R > 0) \approx 1$, i.e., there will be at least one mistake almost surely. Hence, the graphs do not change much from the FDR result.

Problem B.

1. Latent Variable Formulation

Our model is

$$\mathbb{P}(X_1, \dots, X_T) = \sum_z \mathbb{P}(X \mid Z)\mathbb{P}(Z)$$

where $Z = (Z_1, \dots, Z_T)$ is the vector of latent variables $Z_i \in \{1, 2\}$ that indicates the population of origin for X_i, and $\mathbb{P}(X \mid Z)$ is the emission probability. In addition, without knowing the means, $X_i \mid Z_i = k \sim N(\mu_k, \sigma^2)$. Also, for $i = 1$, $Z_1 \sim Mult(1, \pi)$, where $\pi = (\pi_1, \pi_2)$ and $\pi_k = \mathbb{P}(Z_l = k),\ k = 1, 2$.

2. EM Algorithm

The expected complete data log likelihood Q is

$$Q = \sum_t \sum_k w_{tk} \left(-\frac{1}{2}\log(2\pi\sigma^2) + \frac{(x_t - \mu_k)^2}{2\sigma^2} \right)$$

To find μ_k, the M step is

$$\frac{dQ}{d\mu_k} = \sum_t w_{tk}(x_t - \mu_k) \Rightarrow \mu_k = \frac{\sum_t w_{tk}x_t}{\sum_t w_{tk}}$$

Using what I have learned about HMM, given an initialization of $\mu^{(0)} = (\mu_1^{(0)}, \mu_2^{(0)})$, the forward probability α_{tk} is

$$\begin{aligned}
\alpha_{tk}^{(0)} &= p(X_1, \dots, X_t \mid Z_t = k, \mu_k^{(0)}) \\
\alpha_{t+1,k}^{(0)} &= \sum_j \alpha_{tj} P_{jk} p(X_{t+1} \mid Z_{t+1} = k, \mu_k^{(0)}) \\
&= \sum_j \alpha_{tj} P_{jk} N(X_{t+1}; \mu_k^{(0)}, \sigma^2)
\end{aligned}$$

assuming σ is fixed and given. Similarly, the backward probability β_{tk} is

$$\begin{aligned}
\beta_{tk}^{(0)} &= p(X_{t+1}, \dots, X_T \mid Z_t = k, \mu_k^{(0)}) \\
\beta_{tk}^{(0)} &= \sum_j p(X_{t+1}, \dots, X_T, Z_{t+1} = j \mid Z_t = k, \mu_k^{(0)}) \\
&= \sum_j \beta_{t+1,j}^{(0)} P_{kj} p(X_{t+1} \mid Z_{t+1} = j, \mu_j^{(0)}) \\
&= \sum_j \beta_{t+1,j}^{(0)} P_{kj} N(X_{t+1}; \mu_j^{(0)}, \sigma^2)
\end{aligned}$$

Then, I can define the responsibilities as

$$w_{tk}^{(0)} = \mathbb{P}(Z_t = k \mid X_1, \dots, X_T, \mu_k) = \frac{\alpha_{tk}^{(0)} \beta_{tk}^{(0)}}{\sum_{k'} \alpha_{tk'}^{(0)} \beta_{tk'}^{(0)}}$$

Thus, using the analytical result from the M step, I can update the estimate for μ_k as follows:

$$\mu_k^{(1)} = \frac{\sum_t w_{tk} x_t}{\sum_t w_{tk}}$$

Repeat this process until convergence.

I now define my function for running the EM algorithm for the Hidden Markov Model. I first copy and paste the function `emit` from the vignette to use it within my function. The function takes in a vector of Markov chain data `X`, a vector of initialization for the component means `mu`, a scalar for the common standard deviation of the components `sd`, an integer of the possible latent states `K`, a matrix of transition probabilities `P`, and a vector of prior probabilities `pi`. It returns a list of 1) a vector of estimated component means and 2) a vector of incomplete log likelihood values[1]. I formulate the algorithm so that it stops when the updates to the incomplete log likelihood are substantially small. Note that I take the log of α and β values before multiplying them, as it was mentioned in the vignette that α values can be very small and may cause numerical problems.

```r
emit = function(k, x){
  dnorm(x, mean = k, sd = sd)
}

#' @param X a vector of the Markov chain data
#' @param mu a vector of initialization of component means
#' @param sd a scalar for the common standard deviation of the components
#' @param K an integer of the possible latent states
#' @param P a matrix of transition probabilities
#' @param pi a vector of prior probabilities
#' @return a list: 1) a vector of estimated component means and 2) a vector
```

[1] $\ell = \log\left(\sum_k \alpha_{Tk}\right)$

```r
#' of incomplete log likelihood values
em_hmm <- function(X, mu, sd, K, P, pi){
  t. = length(X)
  loglike = c()
  delta = 1

  while(delta > 1e-6){
    # Forward probabilities
    alpha = matrix(nrow = t., ncol = K)
    for(k in 1:K){
      alpha[1, k] = pi[k] * emit(mu[k], X[1])
    }
    for(t in 1:(t.-1)){
      m = alpha[t, ] %*% P
      for(k in 1:K){
        alpha[t+1, k] = m[k] * emit(mu[k], X[t])
      }
    }
    # Backward probabilities
    beta = matrix(nrow = t., ncol = K)
    for(k in 1:K){
      beta[t., k] = 1
      }
    for(t in (t.-1):1){
      for(k in 1:K){
        beta[t, k] = sum(beta[t+1, ] * P[k, ] * emit(mu, X[t+1]))
      }
    }
    # EM for Component Means
    logab = log(alpha) + log(beta)
    prob = exp(logab)/rowSums(exp(logab))
    mu_next = c(sum(X*prob[, 1]) / sum(prob[, 1]),
                sum(X*prob[, 2]) / sum(prob[, 2]))
    loglike = c(loglike, log(sum(alpha[t., ])))

    if(length(loglike) < 2){
      delta = 1
    }else{
      delta = loglike[length(loglike)] - loglike[length(loglike)-1]
    }
    mu = mu_next
  }

  return(list(mu = mu, loglike = loglike))
}
```

3. Check Implementation

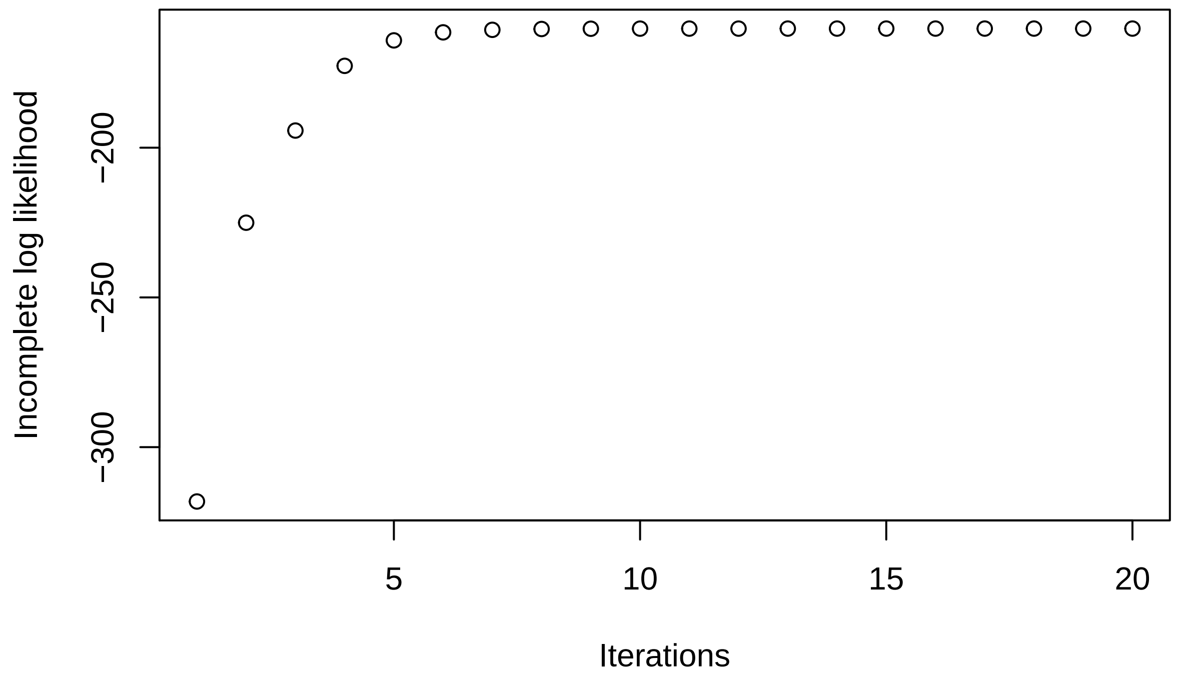
I use the same sample code in the vignette to run my function. I feel this is useful for comparison.

```r
set.seed(1)
t. = 200
K = 2
```

```r
sd = 0.4
P = cbind(c(0.9, 0.1), c(0.1, 0.9))

# Simulate the latent (Hidden) Markov states
Z = rep(0, t.)
Z[1] = 1
for(t in 1:(t.-1)){
  Z[t+1] = sample(K, size = 1, prob = P[Z[t],])
}

# Simulate the emitted/observed values
X = rnorm(t., mean = Z, sd = sd)

pi = c(0.5, 0.5) # Assumed prior distribution on Z_1

alpha = matrix(nrow = t., ncol = K)
```

I run the algorithm and first plot the incomplete log likelihood. I choose (0, 1) as my initial μ values.

```r
run = em_hmm(X, mu = c(0, 1), sd = sd, K = K, P = P, pi = pi)
plot(run$loglike, ylab = "Incomplete log likelihood", xlab = "Iterations")
```



The EM algorithm seems to work well, with monotonic increase in log likelihood for each iteration. Convergence also appears to have been obtained. As is characteristic of EM, the log likelihood improves quite quickly and takes a while to settle onto a point.

```r
run$mu
```

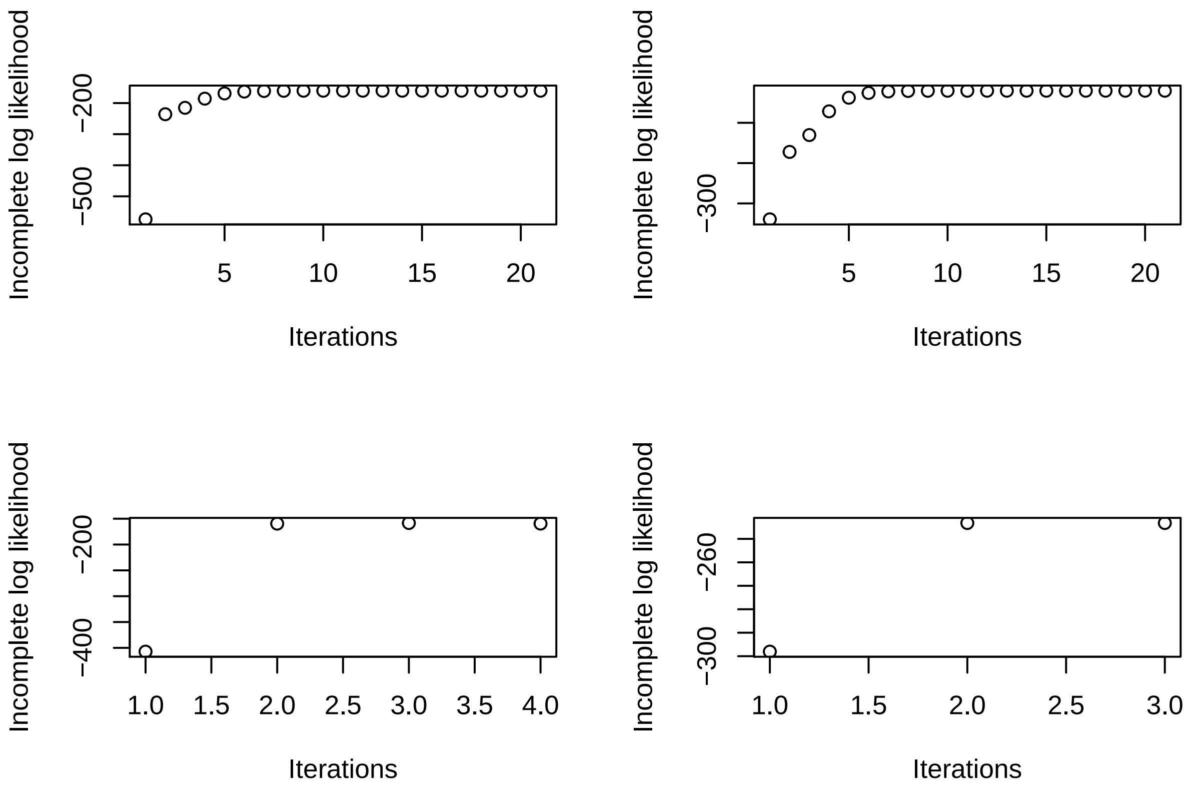
```
## [1] 1.072880 1.799902
```

The estimated component means are approximately (1.073, 1.800), which are reasonably close to the true values of (1, 2).

4. Run Multiple Times

I try running the algorithm with four different points, namely (-2, 2), (1, -3), (3, 0.5), and (1, 1). I first plot the log likelihood.

```r
run1 = em_hmm(X, mu = c(-2, 2), sd = sd, K = K, P = P, pi = pi)
run2 = em_hmm(X, mu = c(1, -3), sd = sd, K = K, P = P, pi = pi)
run3 = em_hmm(X, mu = c(3, 0.5), sd = sd, K = K, P = P, pi = pi)
run4 = em_hmm(X, mu = c(1, 1), sd = sd, K = K, P = P, pi = pi)

par(mfrow = c(2, 2))
plot(run1$loglike, ylab = "Incomplete log likelihood", xlab = "Iterations")
plot(run2$loglike, ylab = "Incomplete log likelihood", xlab = "Iterations")
plot(run3$loglike, ylab = "Incomplete log likelihood", xlab = "Iterations")
plot(run4$loglike, ylab = "Incomplete log likelihood", xlab = "Iterations")
```



An initial look suggests that the last run may have settled at a local optimum. I check the explicit values.

```r
max_lik = c(max(run1$loglike), max(run2$loglike), max(run3$loglike),
            max(run4$loglike))
mu = rbind(run1$mu, run2$mu, run3$mu, run4$mu)
result = as.data.frame(cbind(mu, max_lik))
colnames(result) = c("mu1", "mu2", "max_lik")
rownames(result) = c("Run 1", "Run 2", "Run 3", "Run 4")
result
```

```
##             mu1      mu2   max_lik
## Run 1 1.072880 1.799902 -160.2246
## Run 2 1.799902 1.072880 -160.2246
## Run 3 1.804449 1.073299 -158.3818
## Run 4 1.299578 1.299578 -243.2427
```

To confirm, Run 1 through Run 3 seems to have converged to a reasonable value with approximately equal log likelihood values. Of course, the EM is agnostic of the ordering of μ_k, so some of the values are in the opposite order. The ordering seems to depend on the relative size of the initialized values. However, Run 4 clearly is far apart from the ground truth of (1, 2), with estimates of (1.3, 1.3). The log likelihood is clearly a lot less than the other runs at -243.24. Hence, there exists local optima in the log likelihood, of which one should be mindful when running EM.