Homework 3: Combinatorial & EM

> Shixiang Zhu (GTID # 903280826)
>
> Email: shixiang.zhu@gatech.edu

Question 3.1

Implement a random starts local search algorithm for minimizing the AIC for the baseball salary regression problem. Model your algorithm after Example 3.3.

- (a) Change the move strategy from steepest descent to immediate adoption of the first randomly selected downhill neighbor.

Code 1 *Below is a snippet of code for finding k-Neighborhoods of a solution.*

```
k_Neighborhood <- function(sol, p, k=1) {
  # index of positions in solution
  positions  <- 1:p
  # candidates for the positions of changes
  candidates <- combn(positions, k)
  # init neighbors with padding zeros
  neighbors  <- matrix(0, ncol=p, nrow=ncol(candidates))
  for (i in 1:ncol(candidates)){
    # init neighbor with the starting point
    neighbor <- c(sol)
    for (ind in candidates[,i]){
      # turn it to 1 if the change position is not in solution
      if (ind %in% sol){
        neighbor <- setdiff(neighbor, ind)
      }
      # turn it to 0 if the change position is in solution
      else{
        neighbor <- c(neighbor, ind)
      }
    }
    # do zero paddings for neighbor
    neighbors[i,] <- c(neighbor, rep(0, p - length(neighbor)))
  }
  return(neighbors)
}
```

Code 2 *Below is a snippet of code for* `local searching` *via* `steepest descent`

```
  localSearchForLR <- function(
      sol, x=baseballData[,2:28], y=baseballData[,1], k=1, n=1000){
    # Start the clock!
    ptm <- proc.time()

    p   <- ncol(x)           # number of variables
    x$y <- y                 # add y to data
    X       <- x[,sol]           # init x data according to init solution
    lr      <- lm(y ~ ., data=X) # build linear regression model on
filtered data
    lastRes <- AIC(lr)           # calculate AIC for lr model

    iters <- c() # init iteration results of all AIC history
    for (j in 1:n){
      # get neighborhoods for the current solution
      neighbors <- k_Neighborhood(sol, p, k=k)

      # Rule 1: Search all neighborhood
      # calculate lr for every neighborhood of current solution
      res <- c() # init results of AIC
      for (i in 1:nrow(neighbors)){
        neighbor <- neighbors[i,]
        neighbor <- neighbor[neighbor != 0] # k-neighborhood solution
        neighborX   <- x[,neighbor]
        neighborX$y <- y
        lr  <- lm(y ~ ., data=neighborX)    # build linear regression model
        res <- c(res, AIC(lr))              # calculate AIC for lr model
      }
      # find better neighborhood in accordance with its AIC result
      minInd <- which.min(res)
      minRes <- res[minInd]
      sol    <- neighbors[minInd,]
      sol    <- sol[sol != 0]
      # Stopping criterion
      if (minRes >= lastRes){
        # Stop the clock
        dt <- proc.time() - ptm
        result <- list("solution"   = lastSol,
                       "time"       = dt,
                       "iterations" = iters)
        return(result)
      }

      # Logging trace history of solutions
```

```
      iters   <- c(iters, minRes)
      lastRes <- minRes
      lastSol <- sol
    }
    stop("Exceeded allowed number of iterations")
  }
```

> **Below is the results:**
>
> *(Start from (1,2,3,4))*
>
> $solution [1] 3 4 13 14 8 10 27 15 7
>
> $time user system elapsed 0.435 0.017 0.483
>
> $iterations [1] 5486.195 5441.047 5405.131 5389.995 5384.965 5383.118 5381.728 5379.813 5378.474

Code 3 *Below is a snippet of code for* `local searching` *via* `immediate adoption of the first randomly selected downhill neighbor`.

```
  localSearchForLR <- function(
      sol, x=baseballData[,2:28], y=baseballData[,1], k=1, n=1000){
    # Start the clock!
    ptm <- proc.time()

    p   <- ncol(x)           # number of variables
    x$y <- y                 # add y to data
    X       <- x[,sol]           # init x data according to init solution
    lr      <- lm(y ~ ., data=X) # build linear regression model on filtered data
    lastRes <- AIC(lr)           # calculate AIC for lr model

    iters <- c() # init iteration results of all AIC history
    for (j in 1:n){
      # get neighborhoods for the current solution
      neighbors <- k_Neighborhood(sol, p, k=k)

      # Rule 2: Immediate adoption of the first randomly selected downhill neighbor
      # randomly pick neighborhood of current solution
      counter <- 0
      for (i in sample(1:nrow(neighbors))){
        neighbor <- neighbors[i,]
        neighbor <- neighbor[neighbor != 0] # k-neighborhood solution
        neighborX   <- x[,neighbor]
        neighborX$y <- y
```

```
        lr  <- lm(y ~ ., data=neighborX) # build linear regression model on
filtered data
        res <- AIC(lr)                   # calculate AIC for lr model
        # pick the first downhill neiborhood solution
        if (res < lastRes) {
          minRes <- res
          sol    <- neighbor
          break
        }
        counter <- counter + 1
      }
      # stopping criterion
      if (counter >= nrow(neighbors)){
        # Stop the clock
        dt <- proc.time() - ptm
        result <- list("solution"   = lastSol,
                       "time"       = dt,
                       "iterations" = iters)
        return(result)
      }

      # Logging trace history of solutions
      iters   <- c(iters, minRes)
      lastRes <- minRes
      lastSol <- sol
    }
    stop("Exceeded allowed number of iterations")
  }
```

> **Below is the results:**
>
> *(Start from (1,2,3,4))*
>
> $solution [1] 7 10 25 8 13 14 19 9
>
> $time user system elapsed 0.331 0.003 0.358
>
> $iterations [1] 5575.563 5574.578 5562.569 5550.247 5549.765 5549.075 5547.291 5531.017 [9] 5529.764 5525.200 5518.767 5516.779 5515.913 5427.225 5381.288 5380.471 [17] 5378.790 5378.115 5377.333 5375.788 5375.674 5375.553 5375.400 5375.362

- (b) Change the algorithm to employ $\mathbf{2}$-neighborhoods, and compare the results with those of previous runs.
  - **Steepest descent**

    > $solution [1] 13 14 8 10 25 7 19 9

> $time user system elapsed 4.514 0.074 4.686
>
> $iterations [1] 5441.047 5389.995 5381.915 5378.460 5376.452 5375.708 5375.553 5375.362

- **Imediate adoption**

> $solution [1] 3 8 13 14 10 16 4 18 12 15 17 25 11 19
>
> $time user system elapsed 1.776 0.041 1.872
>
> $iterations [1] 5559.430 5546.047 5453.730 5447.718 5446.832 5441.939 5441.937 5441.750 [9] 5441.290 5395.620 5393.999 5392.707 5390.283 5387.542 5387.400 5383.245 [17] 5383.071 5382.792 5381.700 5379.678 5379.004 5378.901 5378.856 5378.627 [25] 5376.532 5375.996 5375.850

Question 3.8

Thirteen chemical measurements were carried out on each of 178 wines from three regions of Italy. These data are available from the website for this book. Using one or more heuristic search methods from this chapter, partition the wines into three groups for which the total of the within-group sum of squares is minimal. Comment on your work and the results. This is a search problem of size 3^p where $p = 178$. If you have access to standard cluster analysis routines, check your results using a standard method like that of Hartigan and Wong.

Code 4 *Below is a snippet of code for implementing* `k-means` *and visualization of the result via* `t-SNE`

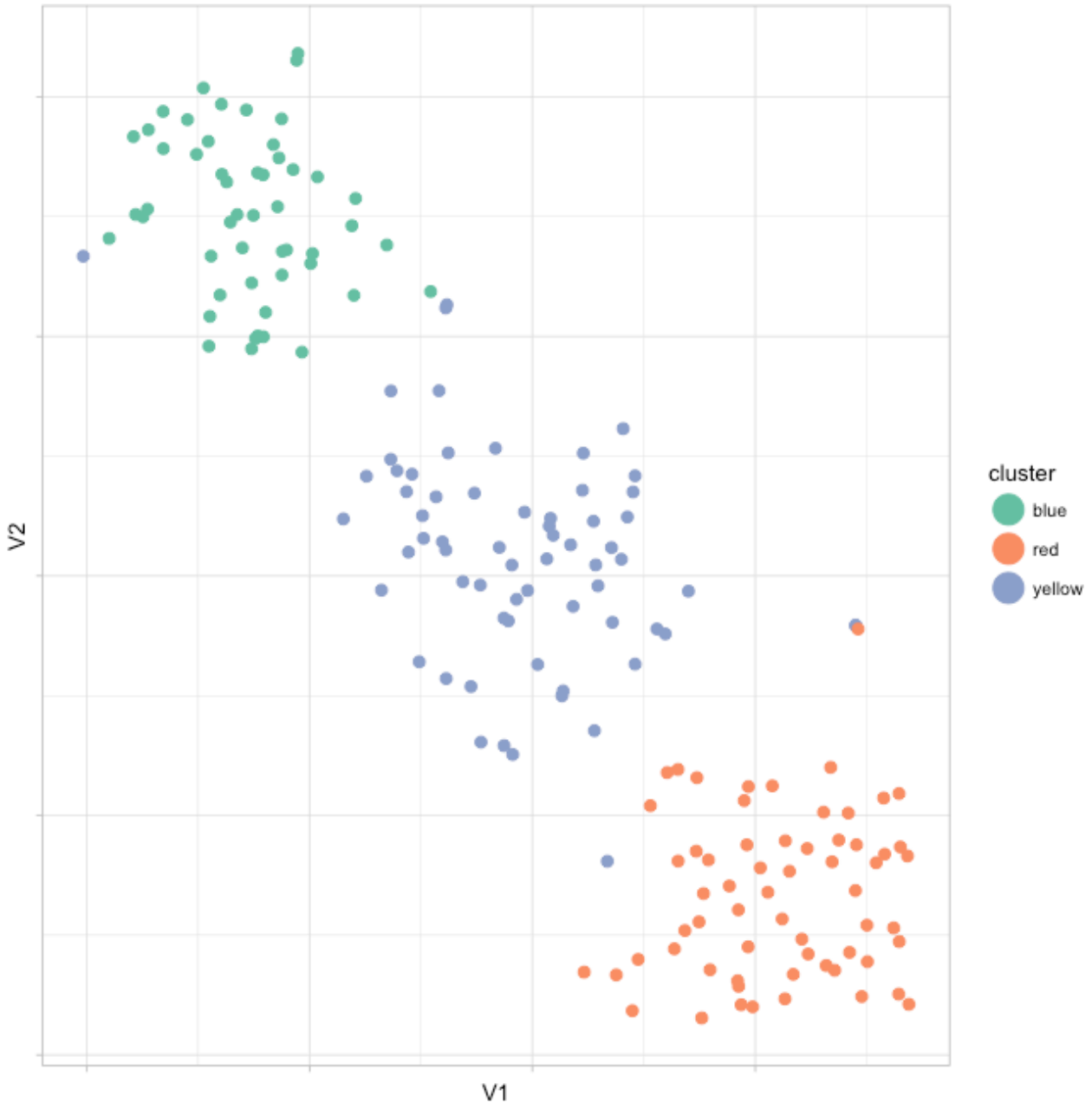
```
# Load data
wineDataPath <- paste(rootPath, "datasets/wine.dat", sep="/")
wineData     <- read.table(wineDataPath, header=TRUE)
# Using K-Means (Hartigan and Wong) as a benchmark
set.seed(20)
wineCluster <- kmeans(wineData[, 2:14], 3, nstart = 20)
# Visualize the result via t-SNE
library(caret)
library(Rtsne)
colorbar <- c("blue", "yellow", "red")
tsneModel <- Rtsne(
  as.matrix(wineData[, 2:14]),
  check_duplicates=FALSE, pca=TRUE, perplexity=30, theta=0.5, dims=2)
tsneEmbeddings         <- as.data.frame(tsneModel$Y)
tsneEmbeddings$cluster <- apply(as.matrix(wineCluster$cluster), 1,
                                function(x) colorbar[x])
# Plot scatterring graph
ggplot(tsneEmbeddings, aes(x=V1, y=V2, color=cluster)) +
  geom_point(size=2.) +
  guides(colour=guide_legend(override.aes=list(size=6))) +
```

```r
  theme_light(base_size=10) +
  theme(axis.text.x=element_blank(),
        axis.text.y=element_blank()) +
  scale_colour_brewer(palette = "Set2")
```

Below is the output of k-means in R.

> K-means clustering with 3 clusters of sizes 51, 65, 62
>
> Cluster means:
>
> x1 x2 x3 x4 x5 x6 x7
>
> 1 0.1643137 0.8698039 0.1862745 0.5223529 -0.07333333 -0.97568627 -1.2111765 2 -0.9236923 -0.3927692 -0.4932308 0.1690769 -0.48830769 -0.07476923 0.0200000 3 0.8329032 -0.3030645 0.3637097 -0.6103226 0.57774194 0.88500000 0.9732258
>
> x8 x9 x10 x11 x12 x13
>
> 1 0.72372549 -0.77764706 0.9390196 -1.1627451 -1.2886275 -0.4062745 2 -0.03338462 0.05830769 -0.8990769 0.4596923 0.2695385 -0.7515385 3 -0.56177419 0.57822581 0.1703226 0.4717742 0.7772581 1.1214516
>
> Clustering vector: [1] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 [40] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 2 1 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2 [79] 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 [118] 2 1 2 2 3 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 [157] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
>
> Within cluster sum of squares by cluster: [1] 326.4147 558.8050 385.7191 (between_SS / total_SS = 44.8 %)
>
> Available components:
>
> [1] "cluster" "centers" "totss" "withinss" "tot.withinss" [6] "betweenss" "size" "iter" "ifault"

We also visualize the output of k-means by applying t-SNE to project points to a 2-D space.



code 5 define how to calculate cost function

```r
# cost = group rss
groupRSS <- function(data, cluster, K=3) {
    for (k in 1:K){
        # get indices of data entries which belongs to a same cluster
        indices <- which(cluster %in% k)
        # calculate mean value for each of attributes in a specific cluster
        mean    <- colMeans(data[indices,], dim=1)
        # get differences between original entries and mean value
        data[indices,] <- sweep(data[indices,], 2, mean)
    }
    return(sum(data^2))
}
```

code 6 define how to find communicated neighborhoods for current solution

```r
# get neighborhood of current solution
neighborhood <- function(sol){
    # init neighbors with padding zeros
    neighbors <- matrix(0, ncol=length(sol), nrow=length(sol)*2)
    # search all neighbors in distance 1.
    j <- 1
    for (i in 1:length(sol)){
        originVal <- sol[i]                         # original value of
candidate position
        candVals  <- setdiff(c(1,2,3), originVal) # candidate values for
this position
        for (val in candVals){
            neighbor      <- sol      # init as original solution
            neighbor[i]   <- val      # make change at specific position
            neighbors[j,] <- neighbor # add this neighbor to matrix
            j <- j + 1
        }
    }
    return(neighbors)
}
```

code 7 below is a snippet of code for `Simulated Annealing`

```r
# Simulated Annealing
simulatedAnnealing <- function(sol, data=wineData[,2:13], n=10000,
step=0.1) {

  # Configuration
  alpha <- 0.01        # cooling rate
  beta  <- 2           # stage rate
```

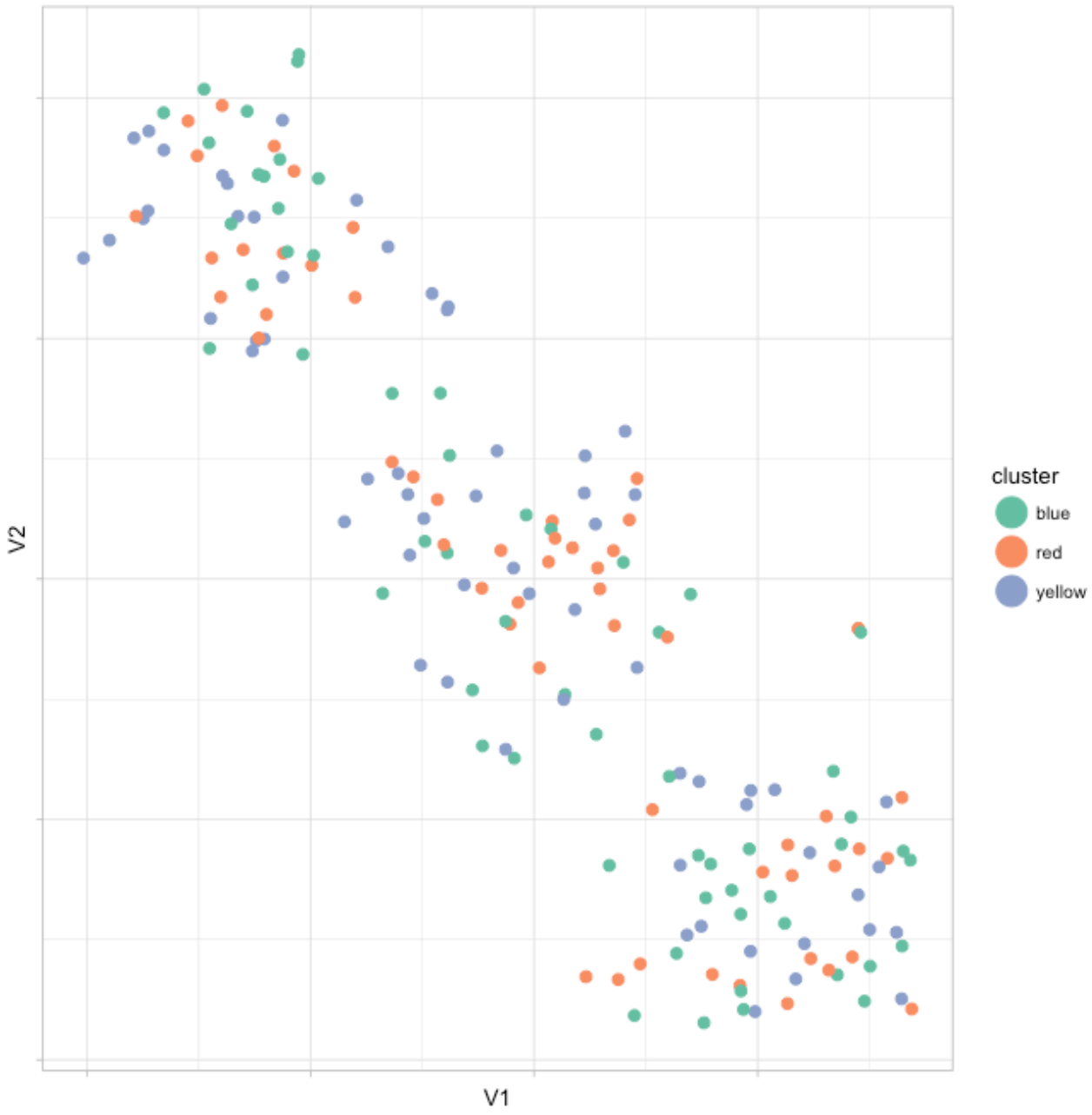
```r
  ptm   <- proc.time() # Start the clock!

  temp  <- 1    # temperature
  stage <- 1    # length of stage m
  p     <- nrow(data) # number of variables
  cost <- groupRSS(data, sol, K=3)

  iters <- c() # init iteration results of cost
  for (j in 1:n){
    for (m in 1:stage){
      # get neighborhoods for the current solution
      neighbors       <- neighborhood(sol)
      res             <- c() # init results for cost
      neighborIndices <- c() # init candidates for neighbor
      for (i in 1:nrow(neighbors)){
        neighbor     <- neighbors[i,]
        neighborCost <- groupRSS(data, neighbor, K=3) # calculate cost for
each neighbor
        if (neighborCost < cost){
          res             <- c(res, neighborCost)
          neighborIndices <- c(neighborIndices, i)
        }
      }
      # stop criterion
      if (length(res) <= 0){
        # Stop the clock
        dt <- proc.time() - ptm
        result <- list("solution"   = sol,
                       "time"       = dt,
                       "iterations" = iters)
        return(result)
      }
      # randomly pick a neighbor from candidates
      candInd      <- sample(1:length(res), 1)
      candNeighbor <- neighbors[neighborIndices[candInd],]
      candCost     <- res[candInd]
      # accept this solution by accept rate
      acceptRate   <- min(1, exp((cost - candCost)/temp))
      print(acceptRate)
      if (sample(c(TRUE,FALSE), size=1, replace=TRUE,
                 prob=c(acceptRate,1-acceptRate))){
        sol   <- candNeighbor
        cost  <- candCost
        iters <- c(iters, cost)
      }
    }
```

```
    # update temperature and stage
    temp  <- temp/(1+alpha*temp)
    stage <- stage * beta
  }
}
```

Below is thee solution:

> [1] 1 2 2 1 3 1 3 3 2 3 1 3 1 3 2 3 2 1 1 2 1 3 1 1 2 3 1 2 1 3 3 3 1 1 2 1 2 1 2 [40] 3 3 2 2 1 3 2 1 1 2 3 1 1 2 1 3 3 2 2 2 2 2 2 3 1 1 1 1 2 2 3 1 1 3 1 2 2 2 1 [79] 3 2 3 3 1 1 1 3 3 2 2 1 3 2 3 2 3 1 2 1 3 1 3 2 3 3 3 2 1 2 3 2 1 2 2 3 2 2 3 [118] 2 3 1 3 3 2 1 2 3 1 1 3 2 2 2 2 3 2 3 3 2 2 2 2 3 2 1 1 3 3 2 1 3 2 1 2 2 2 2 [157] 1 1 1 1 1 1 3 3 1 1 3 1 3 3 2 1 1 3 1 2 3 1 2

And the visualization of Simulated Annealing by t-SNE:



Question 4.1

Recall the peppered moth analysis introduced in Example 4.2. In the field, it is quite difficult to distinguish the *insularia* or *typica* phenotypes due to variations in wing color and mottle. In addition to the 662 moths mentioned in the example, suppose the sample collected by the researchers actually included $n_U = 578$ more moths that were known to be *insularia* or *typica* but whose exact phenotypes could not be determined.

- (a) Derive the EM algorithm for maximum likelihood estimation of p_C, p_I, and p_I for this modified problem having observed data n_C, n_I, n_T, and n_U as given above.

  The observed data are $\mathbf{x} = (n_C, n_I, n_T, n_U)$ and

  the complete data are $\mathbf{y} = (n_{CC}, n_{CI}, n_{CT}, n_{dII}, n_{dIT}, n_{dTT}, n_{nII}, n_{nIT}, n_{nTT})$,

  where $n_{dII}, n_{dIT}, n_{dTT}$ denote the numbers of moth whose phenotypes are determined, and $n_{nII}, n_{nIT}, n_{nTT}$ denote the numbers of moth whose phenotypes could not be determined.

  The mapping from the complete data to the observed data is:

$$\begin{aligned} n_C &= n_{CC} + n_{CI} + n_{CT}, \\ n_I &= n_{dII} + n_{dIT}, \\ n_T &= n_{dTT}, \\ n_U &= n_{nII} + n_{nIT} + n_{nTT} \end{aligned} \tag{1}$$

  According to the question, we are going to estimate the allele probabilities, p_C, p_I, and p_T. The parameters for this problem is $\mathbf{p} = (p_C, p_I)$, similarily for notational brevity we refer to p_T in what follows.

  The complete data log likelihood function is multinomial:

$$\begin{aligned} log\, f\mathbf{Y}(\mathbf{y}|\mathbf{p}) = & n_{CC} log(p_C^2) + n_{CI} log(2p_C p_I) + n_{CT} log(2p_C p_T) \\ & + n_{II} log(p_I^2) + n_{IT} log(2p_I p_T) + n_{TT} log(p_T^2) \\ & + log \begin{pmatrix} n \\ n_{CC}\ n_{CI}\ n_{CT}\ n_{II}\ n_{IT}\ n_{TT} \end{pmatrix} \end{aligned} \tag{2}$$

  Let $\mathbf{Y} = (N_{CC}, N_{CI}, N_{CT}, N_{II}, N_{IT}, N_{TT})$, and $n_{II} = n_{dII} + n_{nII}$, $n_{IT} = n_{dIT} + n_{nIT}$, $n_{TT} = n_{dTT} + n_{nTT}$. none of these frequencies can be observed directly.

  The expected values of the random parts of **Eq. (2)** are

$$E(N_{CC}|n_C, n_I, n_T, n_U, \mathbf{p}^{(t)}) = n_{CC}^{(t)} = \frac{n_C (p_C^{(t)})^2}{(p_C^{(t)})^2 + 2p_C^{(t)} p_I^{(t)} + 2p_C^{(t)} p_T^{(t)}} \tag{3}$$

$$E(N_{CI}|n_C, n_I, n_T, n_U, \mathbf{p}^{(t)}) = n_{CI}^{(t)} = \frac{2n_C p_C^{(t)} p_I^{(t)}}{(p_C^{(t)})^2 + 2p_C^{(t)} p_I^{(t)} + 2p_C^{(t)} p_T^{(t)}} \tag{4}$$

$$E(N_{CT}|n_C, n_I, n_T, n_U, \mathbf{p}^{(t)}) = n_{CT}^{(t)} = \frac{2n_C p_C^{(t)} p_T^{(t)}}{(p_C^{(t)})^2 + 2p_C^{(t)} p_I^{(t)} + 2p_C^{(t)} p_T^{(t)}} \tag{5}$$

$$E(N_{II}|n_C, n_I, n_T, n_U, \mathbf{p}^{(t)}) = n_{dII}^{(t)} + n_{nII}^{(t)} = \frac{n_I (p_I^{(t)})^2}{(p_I^{(t)})^2 + 2p_I^{(t)} p_T^{(t)}} + \frac{n_U (p_I^{(t)})^2}{(p_I^{(t)})^2 + 2p_I^{(t)} p_T^{(t)} + (p_T^{(t)})^2} \tag{6}$$

$$E(N_{IT}|n_C, n_I, n_T, n_U, \mathbf{p}^{(t)}) = n_{dIT}^{(t)} + n_{nIT}^{(t)} = \frac{2n_I p_I^{(t)} p_T^{(t)}}{(p_I^{(t)})^2 + 2p_I^{(t)} p_T^{(t)}} + \frac{2n_U p_I^{(t)} p_T^{(t)}}{(p_I^{(t)})^2 + 2p_I^{(t)} p_T^{(t)} + (p_T^{(t)})^2} \tag{7}$$

$$E(N_{TT}|n_C, n_I, n_T, n_U, \mathbf{p}^{(t)}) = n_{dTT}^{(t)} + n_{nTT}^{(t)} = \frac{n_U (p_T^{(t)})^2}{(p_I^{(t)})^2 + 2p_I^{(t)} p_T^{(t)} + (p_T^{(t)})^2} + n_T \tag{8}$$

Then differentiating with respect to p_C and p_I yields

$$\frac{dQ(\mathbf{p}|\mathbf{p}^{(t)})}{dp_C} = \frac{2n_{CC}^{(t)} + n_{CI}^{(t)} + n_{CT}^{(t)}}{p_C} - \frac{2n_{TT}^{(t)} + n_{CT}^{(t)} + n_{IT}^{(t)}}{1 - p_C - p_I} \tag{9}$$

$$\frac{dQ(\mathbf{p}|\mathbf{p}^{(t)})}{dp_I} = \frac{2n_{II}^{(t)} + n_{IT}^{(t)} + n_{CI}^{(t)}}{p_I} - \frac{2n_{TT}^{(t)} + n_{CT}^{(t)} + n_{IT}^{(t)}}{1 - p_C - p_I} \tag{10}$$

Setting these derivatives equal to zero and solving for p_C and p_I completes the M step, yielding

$$p_C^{(t+1)} = \frac{2n_{CC}^{(t)} + n_{CI}^{(t)} + n_{CT}^{(t)}}{2n} \tag{11}$$

$$p_I^{(t+1)} = \frac{2n_{II}^{(t)} + n_{IT}^{(t)} + n_{CI}^{(t)}}{2n} \tag{12}$$

$$p_T^{(t+1)} = \frac{2n_{TT}^{(t)} + n_{CT}^{(t)} + n_{IT}^{(t)}}{2n} \tag{13}$$

- (b) Apply the algorithm to find the MLEs.

```r
niters <- 1000
# raw data
nC <- 85
nI <- 196
nT <- 341
nU <- 578
n  <- nC + nI + nT + nU

# init value of p
pC <- 0.1
pI <- 0.1
pT <- 0.8

# standard EM
lastpC <- 0
lastpI <- 0
lastpT <- 1
for (i in 1:niters){
  # E (Estimation) step
  nCC <- (nC*pC^2) / (pC^2 + 2*pC*pI + 2*pC*pT)
  nCI <- (2*nC*pC*pI) / (pC^2 + 2*pC*pI + 2*pC*pT)
  nCT <- (2*nC*pC*pT) / (pC^2 + 2*pC*pI + 2*pC*pT)
  nII <- (nI*pI^2) / (pI^2 + 2*pI*pT) + (nU*pI^2) / (pI^2 + 2*pI*pT + pT^2)
  nIT <- (2*nI*pI*pT) / (pI^2 + 2*pI*pT) + (2*nU*pI*pT) / (pI^2 + 2*pI*pT + pT^2)
  nTT <- nT + (nU*pT^2) / (pI^2 + 2*pI*pT + pT^2)
  # M (Maximization) step
  pC <- (2*nCC + nCI + nCT) / (2*n)
  pI <- (2*nII + nIT + nCI) / (2*n)
  pT <- (2*nTT + nCT + nIT) / (2*n)
  # stop criterion
  if (((lastpC - pC)^2 + (lastpI - pI)^2 + (lastpT - pT)^2) < 10e-5){
    break
  }
  lastpC <- pC
  lastpI <- pI
  lastpT <- pT
}
```

The result is $p_C = 0.03606708$, $p_I = 0.1896198$, $p_T = 0.7743131$.

- (c) Estimate the standard errors and pairwise correlations for $\hat{p_C}$, $\hat{p_I}$ and $\hat{p_I}$ using the SEM algorithm.

```r
# init value
x     <- c(85, 196, 341, 578)
n     <- rep(0,6)
itr   <- 40
p     <- c(0.07, 0.19, 0.74)
p.em  <- p
theta <- matrix(0,3,3)
psi   <- rep(0,3)
r     <- matrix(0,3,3)

# E step
allele.e <- function(x,p){
  n.cc <- (x[1]*(p[1]^2))/((p[1]^2)+2*p[1]*p[2]+2*p[1]*p[3])
  n.ci <- (2*x[1]*p[1]*p[2])/((p[1]^2)+2*p[1]*p[2]+2*p[1]*p[3])
  n.ct <- (2*x[1]*p[1]*p[3])/((p[1]^2)+2*p[1]*p[2]+2*p[1]*p[3])
  n.ii <- (x[2]*p[2]^2) / (p[2]^2 + 2*p[2]*p[3]) + (x[4]*p[2]^2) /
(p[2]^2 + 2*p[2]*p[3] + p[3]^2)
  n.it <- (2*x[2]*p[2]*p[3]) / (p[2]^2 + 2*p[2]*p[3]) +
(2*x[4]*p[2]*p[3]) / (p[2]^2 + 2*p[2]*p[3] + p[3]^2)
  n.tt <- x[3] + (x[4]*p[3]^2) / (p[2]^2 + 2*p[2]*p[3] + p[3]^2)
  n <- c(n.cc,n.ci,n.ct,n.ii,n.it,n.tt)
  return(n)
}

# M step
allele.m <- function(x,n){
  p.c <- (2*n[1]+n[2]+n[3])/(2*sum(x))
  p.i <- (2*n[4]+n[5]+n[2])/(2*sum(x))
  p.t <- (2*n[6]+n[3]+n[5])/(2*sum(x))
  p <- c(p.c,p.i,p.t)
  return(p)
}

# compute em computation
for(i in 1:itr){
  n.em <- allele.e(x,p.em)
  p.em <- allele.m(x,n.em)
}

# init theta
for(j in 1:length(p)){
  theta[,j] <- p.em
  theta[j,j] <- p[j]
}

# main
```

```r
for(t in 1:5){
  n <- allele.e(x,p)
  p.hat <- allele.m(x,n)
  for(j in 1:length(p)){
    theta[j,j] <- p.hat[j]
    n <- allele.e(x,theta[,j])
    psi <- allele.m(x,n)
    for(i in 1:length(p)){
      r[i,j] <- (psi[i]-p.em[i])/(theta[j,j]-p.em[j])
    }
  }
  p <- p.hat
}

# complete information
iy.hat=matrix(0,2,2)
iy.hat[1,1] <- ((2*n.em[1]+n.em[2]+n.em[3])/(p.em[1]^2) +
                  (2*n.em[6]+n.em[3]+n.em[5])/(p.em[3]^2))
iy.hat[2,2] <- ((2*n.em[4]+n.em[5]+n.em[2])/(p.em[2]^2) +
                  (2*n.em[6]+n.em[3]+n.em[5])/(p.em[3]^2))
iy.hat[1,2] <- iy.hat[2,1] <- (2*n.em[6]+n.em[3]+n.em[5])/(p.em[3]^2)

# compute standard errors and correlations
var.hat <- solve(iy.hat)%*%(diag(2)+t(r[-3,-3])%*%solve(diag(2)-
t(r[-3,-3])))
sd.hat  <- c(sqrt(var.hat[1,1]),sqrt(var.hat[2,2]),sqrt(sum(var.hat)))
cor.hat <- c(var.hat[1,2]/(sd.hat[1]*sd.hat[2]),
             (-var.hat[1,1]-var.hat[1,2])/(sd.hat[1]*sd.hat[3]),
             (-var.hat[2,2]-var.hat[1,2])/(sd.hat[2]*sd.hat[3]))
```

```
> var.hat
                [,1]          [,2]
[1,]  1.475087e-05 -5.748342e-06
[2,] -3.078801e-06  1.235657e-04
> sd.hat
[1] 0.003840686 0.011116012 0.011379343
> cor.hat
[1] -0.1346433 -0.2059864 -0.9314149
```

- (d) Estimate the standard errors and pairwise correlations for $\hat{p_C}$, $\hat{p_I}$ and $\hat{p_I}$ by bootstrapping.

```r
# init values
x     <- c(85, 196, 341, 578)
n     <- rep(0,6)
p     <- rep(1/3,3)
itr   <- 40
theta <- matrix(0,3,10000)
set.seed(0)

# E step
allele.e <- function(x,p){
  n.cc <- (x[1]*(p[1]^2))/((p[1]^2)+2*p[1]*p[2]+2*p[1]*p[3])
  n.ci <- (2*x[1]*p[1]*p[2])/((p[1]^2)+2*p[1]*p[2]+2*p[1]*p[3])
  n.ct <- (2*x[1]*p[1]*p[3])/((p[1]^2)+2*p[1]*p[2]+2*p[1]*p[3])
  n.ii <- (x[2]*p[2]^2) / (p[2]^2 + 2*p[2]*p[3]) + (x[4]*p[2]^2) /
(p[2]^2 + 2*p[2]*p[3] + p[3]^2)
  n.it <- (2*x[2]*p[2]*p[3]) / (p[2]^2 + 2*p[2]*p[3]) +
(2*x[4]*p[2]*p[3]) / (p[2]^2 + 2*p[2]*p[3] + p[3]^2)
  n.tt <- x[3] + (x[4]*p[3]^2) / (p[2]^2 + 2*p[2]*p[3] + p[3]^2)
  n <- c(n.cc,n.ci,n.ct,n.ii,n.it,n.tt)
  return(n)
}

# M step
allele.m <- function(x,n){
  p.c <- (2*n[1]+n[2]+n[3])/(2*sum(x))
  p.i <- (2*n[4]+n[5]+n[2])/(2*sum(x))
  p.t <- (2*n[6]+n[3]+n[5])/(2*sum(x))
  p <- c(p.c,p.i,p.t)
  return(p)
}

# main
for(i in 1:itr){
  n <- allele.e(x,p)
  p <- allele.m(x,n)
}

theta[,1] <- p
for(j in 2:10000){
  n.c <- rbinom(1, sum(x), x[1]/sum(x))
  n.i <- rbinom(1, sum(x) - n.c, x[2]/(sum(x)-x[1]))
  n.t <- rbinom(1, sum(x) - n.c - n.i, x[2]/(sum(x)-x[1]-x[2]))
  n.u <- sum(x) - n.c - n.i - n.t
  x.new <- c(n.c, n.i, n.t, n.u)
  n <- rep(0,6)
  p <- rep(1/3,3)
```

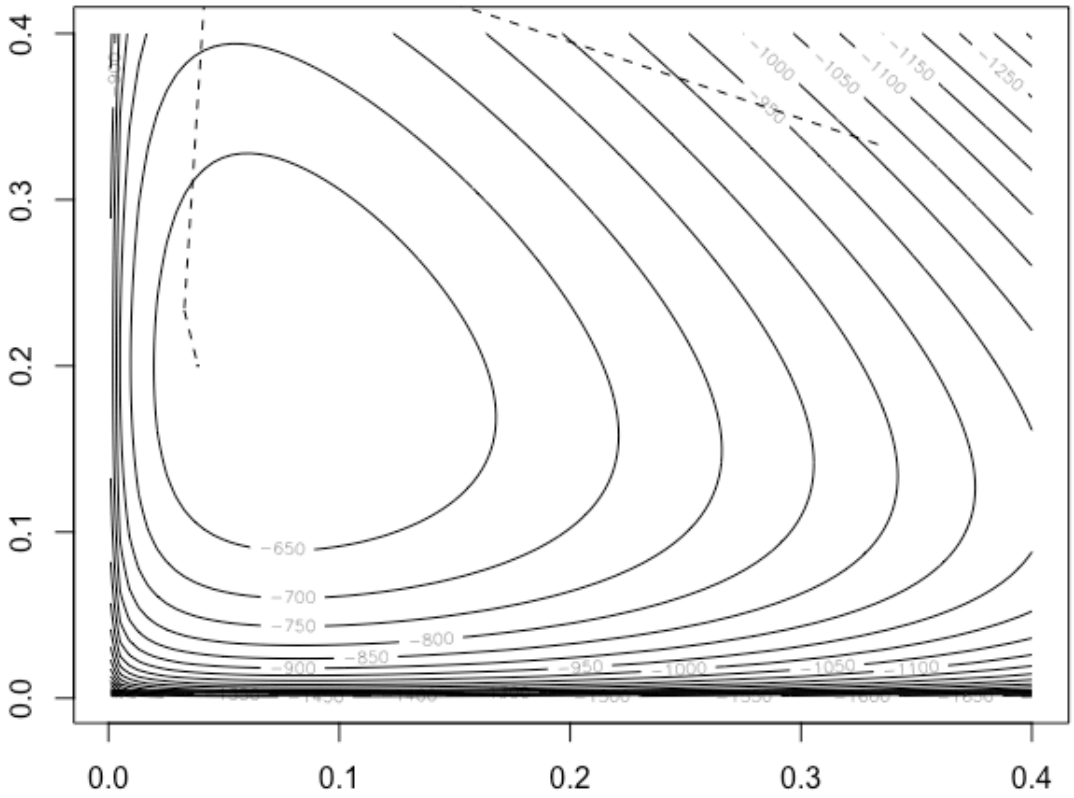
```
  for(i in 1:itr){
    n <- allele.e(x.new,p)
    p <- allele.m(x.new,n)
  }
  theta[,j] <- p
}

sd.hat  <- c(sd(theta[1,]), sd(theta[2,]), sd(theta[3,]))
cor.hat <- c(cor(theta[1,],theta[2,]), cor(theta[1,],theta[3,]),
             cor(theta[2,],theta[3,]))
```

```
> sd.hat
[1] 0.003843569 0.017248652 0.017413335
> cor.hat
[1] -0.06836572 -0.15300647 -0.97545267
```

- (e) Implement the EM gradient algorithm for these data. Experiment with step halving to ensure ascent and with other step scalings that may speed convergence.

```
# init values
x   <- c(85, 196, 341, 578)
n   <- rep(0,6)
p   <- rep(1/3,3)
itr <- 40
prob.values     <- matrix(0,3,itr+1)
prob.values[,1] <- p
alpha.default   <- 2

# E step
allele.e <- function(x,p){
  n.cc <- (x[1]*(p[1]^2))/((p[1]^2)+2*p[1]*p[2]+2*p[1]*p[3])
  n.ci <- (2*x[1]*p[1]*p[2])/((p[1]^2)+2*p[1]*p[2]+2*p[1]*p[3])
  n.ct <- (2*x[1]*p[1]*p[3])/((p[1]^2)+2*p[1]*p[2]+2*p[1]*p[3])
  n.ii <- (x[2]*p[2]^2) / (p[2]^2 + 2*p[2]*p[3]) + (x[4]*p[2]^2) /
(p[2]^2 + 2*p[2]*p[3] + p[3]^2)
  n.it <- (2*x[2]*p[2]*p[3]) / (p[2]^2 + 2*p[2]*p[3]) +
(2*x[4]*p[2]*p[3]) / (p[2]^2 + 2*p[2]*p[3] + p[3]^2)
  n.tt <- x[3] + (x[4]*p[3]^2) / (p[2]^2 + 2*p[2]*p[3] + p[3]^2)
  n <- c(n.cc,n.ci,n.ct,n.ii,n.it,n.tt)
  return(n)
}

allele.l <- function(x,p){
```

```
  l <- ( x[1]*log(2*p[1] - p[1]^2) + x[2]*log(p[2]^2 + 2*p[2]*p[3]) +
          2*x[3]*log(p[3]) )
  return(l)
}

# gradient estimation
Q.prime <- function(n,p){
  da <- (2*n[1]+n[2]+n[3])/(p[1]) - (2*n[6]+n[3]+n[5])/(p[3])
  db <- (2*n[4]+n[5]+n[2])/(p[2]) - (2*n[6]+n[3]+n[5])/(p[3])
  dQ <- c(da,db)
  return(dQ)
}

Q.2prime <- function(n,p){
  da2 <- -(2*n[1]+n[2]+n[3])/(p[1]^2) - (2*n[6]+n[3]+n[5])/(p[3]^2)
  db2 <- -(2*n[4]+n[5]+n[2])/(p[2]^2) - (2*n[6]+n[3]+n[5])/(p[3]^2)
  dab <- -(2*n[6]+n[3]+n[5])/(p[3]^2)
  d2Q <- matrix(c(da2,dab,dab,db2), nrow=2, byrow=TRUE)
  return(d2Q)
}

# main
l.old <- allele.l(x,p)
for(i in 1:itr){
  alpha <- alpha.default
  n <- allele.e(x,p)
  p.new <- p[1:2] - alpha*solve(Q.2prime(n,p))%*%Q.prime(n,p)
  p.new[3] <- 1 - p.new[1] - p.new[2]
  if(p.new > 0 && p.new < 1){l.new <- allele.l(x,p.new)}
  # REDUCE ALPHA UNTIL A CORRECT STEP IS REACHED
  while(p.new < 0 || p.new > 1 || l.new < l.old){
    alpha <- alpha/2
    p.new <- p[1:2] - alpha*solve(Q.2prime(n,p))%*%Q.prime(n,p)
    p.new[3] <- 1 - p.new[1] - p.new[2]
    if(p.new > 0 && p.new < 1){l.new <- allele.l(x,p.new)}
  }
  p <- p.new
  prob.values[,i+1] <- p
  l.old <- l.new
}
```

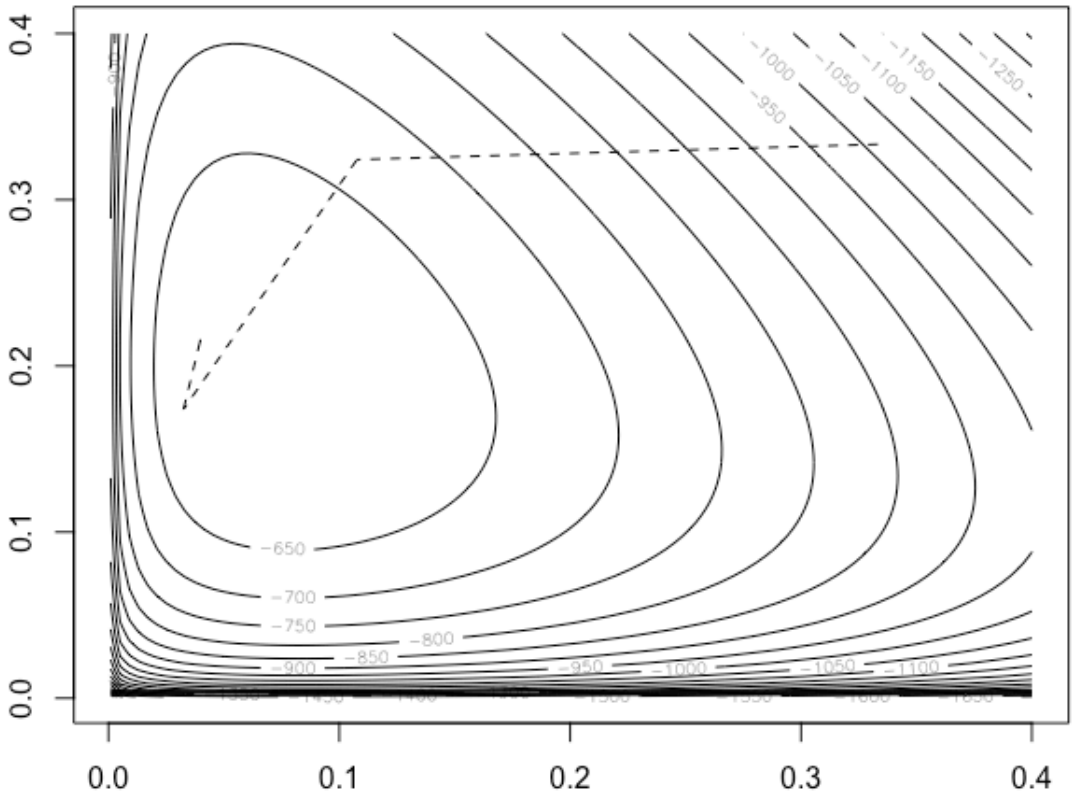


- (f) Implement Aitken accelerated EM for these data. Use step halving.

```r
# init values
x   <- c(85, 196, 341, 578)
n <- rep(0,6)
p <- rep(1/3,3)
itr <- 40
prob.values <- matrix(0,3,itr+1)
prob.values[,1] <- p
alpha.default <- 2

# E step
allele.e <- function(x,p){
  n.cc <- (x[1]*(p[1]^2))/((p[1]^2)+2*p[1]*p[2]+2*p[1]*p[3])
  n.ci <- (2*x[1]*p[1]*p[2])/((p[1]^2)+2*p[1]*p[2]+2*p[1]*p[3])
  n.ct <- (2*x[1]*p[1]*p[3])/((p[1]^2)+2*p[1]*p[2]+2*p[1]*p[3])
  n.ii <- (x[2]*p[2]^2) / (p[2]^2 + 2*p[2]*p[3]) + (x[4]*p[2]^2) /
(p[2]^2 + 2*p[2]*p[3] + p[3]^2)
  n.it <- (2*x[2]*p[2]*p[3]) / (p[2]^2 + 2*p[2]*p[3]) +
(2*x[4]*p[2]*p[3]) / (p[2]^2 + 2*p[2]*p[3] + p[3]^2)
```

```r
  n.tt <- x[3] + (x[4]*p[3]^2) / (p[2]^2 + 2*p[2]*p[3] + p[3]^2)
  n <- c(n.cc,n.ci,n.ct,n.ii,n.it,n.tt)
  return(n)
}

# M step
allele.m <- function(x,n){
  p.c <- (2*n[1]+n[2]+n[3])/(2*sum(x))
  p.i <- (2*n[4]+n[5]+n[2])/(2*sum(x))
  p.t <- (2*n[6]+n[3]+n[5])/(2*sum(x))
  p <- c(p.c,p.i,p.t)
  return(p)
}

allele.l <- function(x,p){
  l <- ( x[1]*log(2*p[1] - p[1]^2) + x[2]*log(p[2]^2 + 2*p[2]*p[3]) +
         2*x[3]*log(p[3]) )
  return(l)
}

allele.iy <- function(n,p){
  iy.hat<-matrix(0,2,2)
  iy.hat[1,1] <- ((2*n[1]+n[2]+n[3])/(p[1]^2) +
                  (2*n[6]+n[3]+n[5])/(p[3]^2))
  iy.hat[2,2] <- ((2*n[4]+n[5]+n[2])/(p[2]^2) +
                  (2*n[6]+n[3]+n[5])/(p[3]^2))
  iy.hat[1,2] <- iy.hat[2,1] <- (2*n[6]+n[3]+n[5])/(p[3]^2)
  return(iy.hat)
}

allele.l.2prime <- function(x,p){
  l.2prime <- matrix(0,2,2)
  l.2prime[1,1] <- ( (-x[1]*(2-2*p[1])^2)/((2*p[1]-p[1]^2)^2) -
                     2*x[1]/(2*p[1]-p[1]^2) -
                     (4*x[2])/((-2*p[1]-p[2]+2)^2) -
                     2*x[3]/(p[3]^2))
  l.2prime[2,2] <- ( (-4*x[2]*p[3]^2)/((p[2]^2 + 2*p[2]*p[3])^2) -
                     2*x[2]/(p[2]^2 + 2*p[2]*p[3]) -
                     2*x[3]/(p[3]^2))
  l.2prime[1,2] <- ((-2*x[2])/((-2*p[1]-p[2]+2)^2) -
                    2*x[3]/(p[3]^2))
  l.2prime[2,1] <- l.2prime[1,2]
  return(l.2prime)
}

# main
```

```
l.old <- allele.l(x,p)
for(i in 1:itr){
  alpha <- alpha.default
  n <- allele.e(x,p)
  p.em <- allele.m(x,n)
  p.new <- (p[1:2] - alpha*solve(allele.l.2prime(x,p))%*%
              allele.iy(n,p)%*%(p.em[1:2]-p[1:2]))
  p.new[3] <- 1 - p.new[1] - p.new[2]
  if(p.new > 0 && p.new < 1){l.new <- allele.l(x,p.new)}
  # REDUCE ALPHA UNTIL A CORRECT STEP IS REACHED
  while(p.new < 0 || p.new > 1 || l.new < l.old){
    alpha <- alpha/2
    p.new <- (p[1:2] - alpha*solve(allele.l.2prime(x,p))%*%
                allele.iy(n,p)%*%(p.em[1:2]-p[1:2]))
    p.new[3] <- 1 - p.new[1] - p.new[2]
    if(p.new > 0 && p.new < 1){l.new <- allele.l(x,p.new)}
  }
  p <- p.new
  prob.values[,i+1] <- p
  l.old <- l.new
}
```

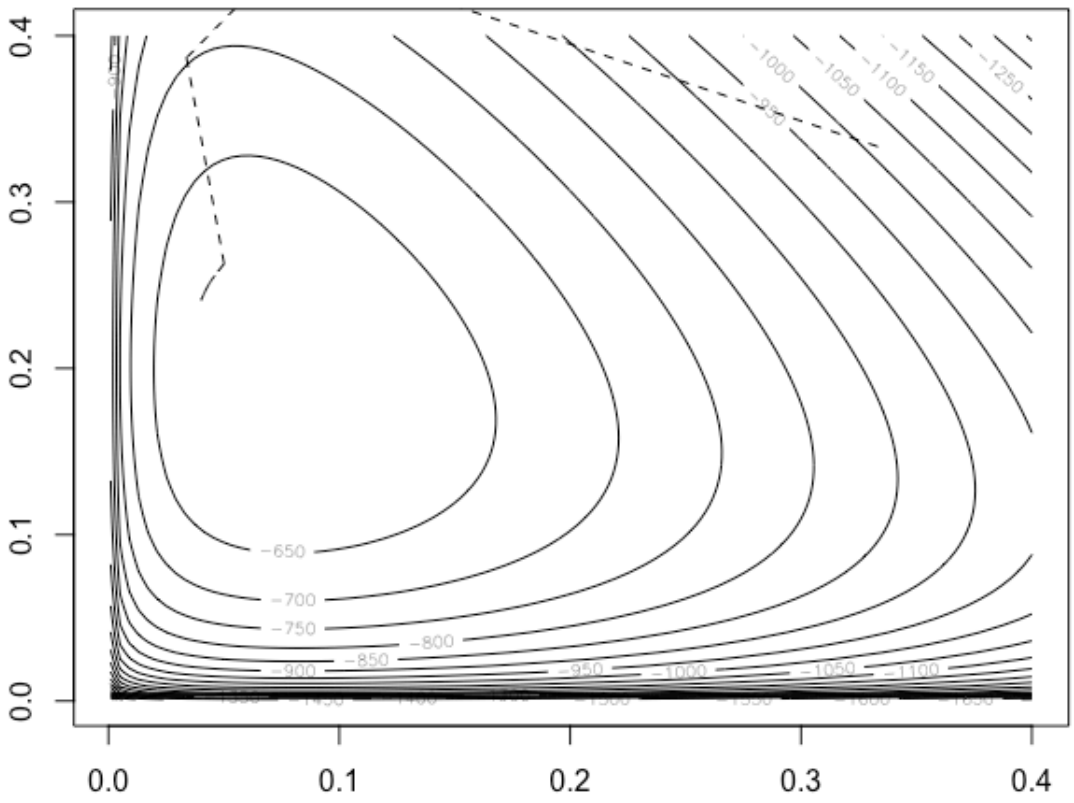


- (g) Implement quasi-Newton EM for these data. Compare performance with and without step halving.

```
# init values
x   <- c(85, 196, 341, 578)
n <- rep(0,6)
p <- rep(1/3,3)
itr <- 20
m <- matrix(0,2,2)
b <- matrix(0,2,2)
prob.values <- matrix(0,3,itr+1)
prob.values[,1] <- p
alpha.default <- 2

# E step
allele.e <- function(x,p){
  n.cc <- (x[1]*(p[1]^2))/((p[1]^2)+2*p[1]*p[2]+2*p[1]*p[3])
  n.ci <- (2*x[1]*p[1]*p[2])/((p[1]^2)+2*p[1]*p[2]+2*p[1]*p[3])
  n.ct <- (2*x[1]*p[1]*p[3])/((p[1]^2)+2*p[1]*p[2]+2*p[1]*p[3])
```

```
  n.ii <- (x[2]*p[2]^2) / (p[2]^2 + 2*p[2]*p[3]) + (x[4]*p[2]^2) / (p[2]^2 + 2*p[2]*p[3] + p[3]^2)
  n.it <- (2*x[2]*p[2]*p[3]) / (p[2]^2 + 2*p[2]*p[3]) + (2*x[4]*p[2]*p[3]) / (p[2]^2 + 2*p[2]*p[3] + p[3]^2)
  n.tt <- x[3] + (x[4]*p[3]^2) / (p[2]^2 + 2*p[2]*p[3] + p[3]^2)
  n <- c(n.cc,n.ci,n.ct,n.ii,n.it,n.tt)
  return(n)
}

allele.l <- function(x,p){
  l <- ( x[1]*log(2*p[1] - p[1]^2) + x[2]*log(p[2]^2 + 2*p[2]*p[3]) +
           2*x[3]*log(p[3]) )
  return(l)
}

# gradient estimation
Q.prime <- function(n,p){
  da <- (2*n[1]+n[2]+n[3])/(p[1]) - (2*n[6]+n[3]+n[5])/(p[3])
  db <- (2*n[4]+n[5]+n[2])/(p[2]) - (2*n[6]+n[3]+n[5])/(p[3])
  dQ <- c(da,db)
  return(dQ)
}

Q.2prime <- function(n,p){
  da2 <- -(2*n[1]+n[2]+n[3])/(p[1]^2) - (2*n[6]+n[3]+n[5])/(p[3]^2)
  db2 <- -(2*n[4]+n[5]+n[2])/(p[2]^2) - (2*n[6]+n[3]+n[5])/(p[3]^2)
  dab <- -(2*n[6]+n[3]+n[5])/(p[3]^2)
  d2Q <- matrix(c(da2,dab,dab,db2), nrow=2, byrow=TRUE)
  return(d2Q)
}

# main
l.old <- allele.l(x,p)
for(i in 1:itr){
  alpha <- alpha.default
  n <- allele.e(x,p)
  m <- Q.2prime(n,p) - b
  p.new <- p[1:2] - alpha*solve(m)%*%Q.prime(n,p)
  p.new[3] <- 1 - p.new[1] - p.new[2]
  if(p.new > 0 && p.new < 1){l.new <- allele.l(x,p.new)}
  # REDUCE ALPHA UNTIL A CORRECT STEP IS REACHED
  while(p.new < 0 || p.new > 1 || l.new < l.old){
    alpha <- alpha/2
    p.new <- p[1:2] - alpha*solve(m)%*%Q.prime(n,p)
    p.new[3] <- 1 - p.new[1] - p.new[2]
    if(p.new > 0 && p.new < 1){l.new <- allele.l(x,p.new)}
```

```
  }
  at <- p.new[1:2]-p[1:2]
  n <- allele.e(x,p.new)
  bt <- Q.prime(n,p)-Q.prime(n,p.new)
  vt <- bt - b%*%at
  ct <- as.numeric(1/(t(vt)%*%at))
  b <- b + ct*vt%*%t(vt)
  p <- p.new
  prob.values[,i+1] <- p
  l.old <- l.new
}
```



- (h) Compare the effectiveness and efficiency of the standard EM algorithm and the three variants in (e), (f), and (g). Use step halving to ensure ascent with the three variants. Base your comparison on a variety of starting points. Create a graph analogous to Figure 4.3.

  Please the details in each subsection of this question.

Question 4.5

- (a) Show the following algorithms can be used to calculate $\alpha(i,h)$ and $\beta(i,h)$.

$$\begin{aligned}\alpha(0,h) &= P(O_0 = o_0, H_0 = h) \\ &= P(O_0 = o_0 | H_0 = h) \cdot P(H_0 = h) \\ &= \pi(h) e(h, o_0)\end{aligned} \tag{14}$$

$$\begin{aligned}\alpha(i+1,h) &= P(\mathbf{O}_{\leq i+1} = \mathbf{o}_{\leq i+1}, H_{i+1} = h) \\ &= \sum_{h^* \in H} P(\mathbf{O}_{\leq i} = \mathbf{o}_{\leq i}, H = h^*) \cdot P(O_{i+1} = o_{i+1} | H_{i+1} = h) \cdot P(H_{i+1} = h | H_i = h^*) \\ &= \sum_{h^* \in H} \alpha(i, h^*) e(h, o_{i+1}) p(h^*, h)\end{aligned} \tag{15}$$

$$\begin{aligned}\beta(i-1,h) &= P(\mathbf{O}_{>i-1} = \mathbf{o}_{>i-1} | H_{i-1} = h) \\ &= \sum_{h^* \in H} P(\mathbf{O}_{>i} = \mathbf{o}_{>i} | H = h^*) \cdot P(O_i = o_i | H_i = h) \cdot P(H_i = h | H_i = h^*) \\ &= \sum_{h^* \in H} \beta(i, h^*) e(h^*, o_i) p(h, h^*)\end{aligned} \tag{16}$$

- (b) Prove that these random variables have the following expectations.

$$\begin{aligned}E_Q(N(h) | \mathbf{O} = \mathbf{o}) &= P(H_0 = h | \mathbf{O} = \mathbf{o}, \theta) \\ &= \frac{P(\mathbf{O}_{>0} = \mathbf{o}, O_0 = o_0 | H_o = h, \theta) \cdot P(H_o = h | Q)}{P(\mathbf{O} = \mathbf{o} | \theta)} \\ &= \frac{\beta(0,h) e(h, o_0) \pi(h)}{P(\mathbf{O} = \mathbf{o} | \theta)} \\ &= \frac{\alpha(0,h)\beta(0,h)}{P(\mathbf{O} = \mathbf{o} | \theta)}\end{aligned} \tag{17}$$

$$\begin{aligned}E_Q(N(h, h^*) | \mathbf{O} = \mathbf{o}) &= E_Q(\sum_{i=0}^{n-1} \mathbb{I}(H_i = h, H_{i+1} = h^*) | \mathbf{O} = \mathbf{o}) \\ &= \sum_{i=0}^{n-1} E_Q(\mathbb{I}(H_i = h, H_{i+1} = h^*) | \mathbf{O} = \mathbf{o}) \\ &= \sum_{i=0}^{n-1} \frac{P(H_i = h, H_{i+1} = h^*, \mathbf{O} = \mathbf{o} | \theta)}{P(\mathbf{O} = \mathbf{o} | \theta)} \\ &= \sum_{i=0}^{n-1} \frac{P(H_i = h, H_{i+1} = h^*, \mathbf{O}_{\leq i} = \mathbf{o}_{\leq i}, O_{i+1} = o_{i+1}, \mathbf{O}_{>i+1} = \mathbf{o}_{>i+1} | \theta)}{P(\mathbf{O} = \mathbf{o} | \theta)} \\ &= \sum_{i=0}^{n-1} \frac{\alpha(i,h) e(h^*, o_{i+1}) \beta(i+1, h^*) P(h, h^*))}{P(\mathbf{O} = \mathbf{o} | \theta)}\end{aligned} \tag{18}$$

$$
\begin{aligned}
E_Q(N(h,o)|\mathbf{O}=\mathbf{o}) &= E_Q(\sum_{i=0}^{n-1}\mathbb{I}(H_i=h,O_i=o)|\mathbf{O}=\mathbf{o}) \\
&= \sum_{i\in O_i=o} E_Q(\mathbb{I}(H_i=h)|\mathbf{O}=\mathbf{o}) \\
&= \sum_{i\in O_i=o} \frac{P(H_i=h,\mathbf{O}_{\leq i}=\mathbf{o}_{\leq i},\mathbf{O}_{>i}=\mathbf{o}_{>i}|\theta)}{P(\mathbf{O}=\mathbf{o}|\theta)} \\
&= \sum_{i\in O_i=o} \frac{\sum_{h\in H} P(\mathbf{O}_{>i}=\mathbf{o}_{>i}|H_i=h,\mathbf{O}_{\leq i}=\mathbf{o}_{\leq i},\theta)\alpha(i,h)}{P(\mathbf{O}=\mathbf{o}|\theta)} \\
&= \sum_{i\in O_i=o} \frac{\alpha(i,h)\beta(i,h)}{P(\mathbf{O}=\mathbf{o}|\theta)}
\end{aligned} \tag{19}
$$

- (c) Prove that the Baum-Welch algorithm is an EM algorithm.

  Take one component of π as an example.

$$
max_\pi \sum E(N(h)|\theta^{(t)},\mathbf{O}) log\, \pi(h) \tag{20}
$$

$$
s.t. \sum_{h\in H^*} \pi(h) = 1 \tag{21}
$$

  We apply Lagrangian Multiplier.

$$
Lang(\pi,\lambda) = \sum_{h\in H} E(N(h)|Q^{(t)},\mathbf{O}) log\, \pi(h) + \lambda(\sum_{h\in H}\pi(h)-1) \tag{22}
$$

  And solve $0 = \frac{\partial}{\partial\pi(h)} Lang(\pi,\lambda)$, we can get

$$
\pi(h) = \frac{E(N(h)|Q^{(t)},\mathbf{O})}{\sum_{h\in H} E(N(h)|Q^{(t)},\mathbf{O})} \tag{23}
$$

$$
\lambda = -\sum_{h\in H} E(N(h)|Q^{(t)},\mathbf{O}) \tag{24}
$$

- (d) Use the Baum-Welch algorithm to estimate p, d, and s.

```
require(HMM)
hmm1 = initHMM(c('dim',"penny"), c("tail","Head"), c(0.5,0.5),
matrix(c(0.25,0.75,0.75,0.25),2),
        matrix(c(0.25,0.5,0.75,0.5),2))
O = read.table('coin.dat', header = TRUE)$outcome
observation = O
observation[O == 1] = 'Head'
```

```
observation[O == 2] = 'tail'
B1 = exp(backward(hmm1,observation))
A1 =exp(forward(hmm1, observation))
baumWelch(hmm1, observation, maxIterations=100, delta=1E-9, pseudoCount = 0)

hmm2 = initHMM(c('dim',"penny"), c("tail","Head"), c(0.5,0.5), matrix(c(0.5,0.5,0.5,0.5),2),
               matrix(c(0.5,0.5,0.5,0.5),2))
baumWelch(hmm2, observation, maxIterations=100, delta=1E-9, pseudoCount = 0)

hmm3 = initHMM(c('dim',"penny"), c("tail","Head"), c(0.5,0.5), matrix(c(0.1,0.9,0.9,0.1),2),
               matrix(c(0.1,0.1,0.9,0.9),2))
baumWelch(hmm3, observation, maxIterations=100, delta=1E-9, pseudoCount = 0)

hmm4 = initHMM(c('dim',"penny"), c("tail","Head"), c(0.5,0.5), matrix(c(1/4,2/3,3/4,1/3),2),
               matrix(c(1/4,2/3,3/4,1/3),2))
baumWelch(hmm3, observation, maxIterations=100, delta=1E-9, pseudoCount = 0)
```

Below is the results:

> *hmm*States [1] "dim" "penny"
>
> *hmm*Symbols [1] "tail" "Head"
>
> *hmm*startProbs dim penny 0.5 0.5
>
> *hmm*transProbs
>
> from dim penny dim 0.1 0.9 penny 0.9 0.1
>
> *hmm*emissionProbs
>
> states tail Head dim 0.45 0.55 penny 0.45 0.55
>
> $difference [1] 7.000000e-01 1.353259e-14