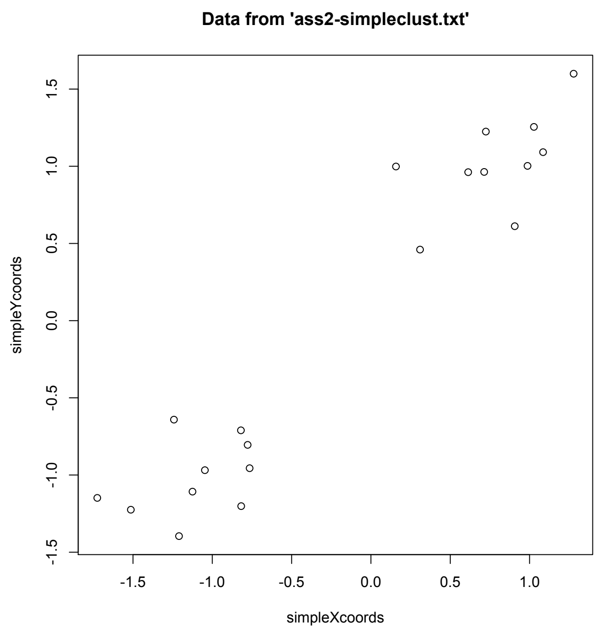
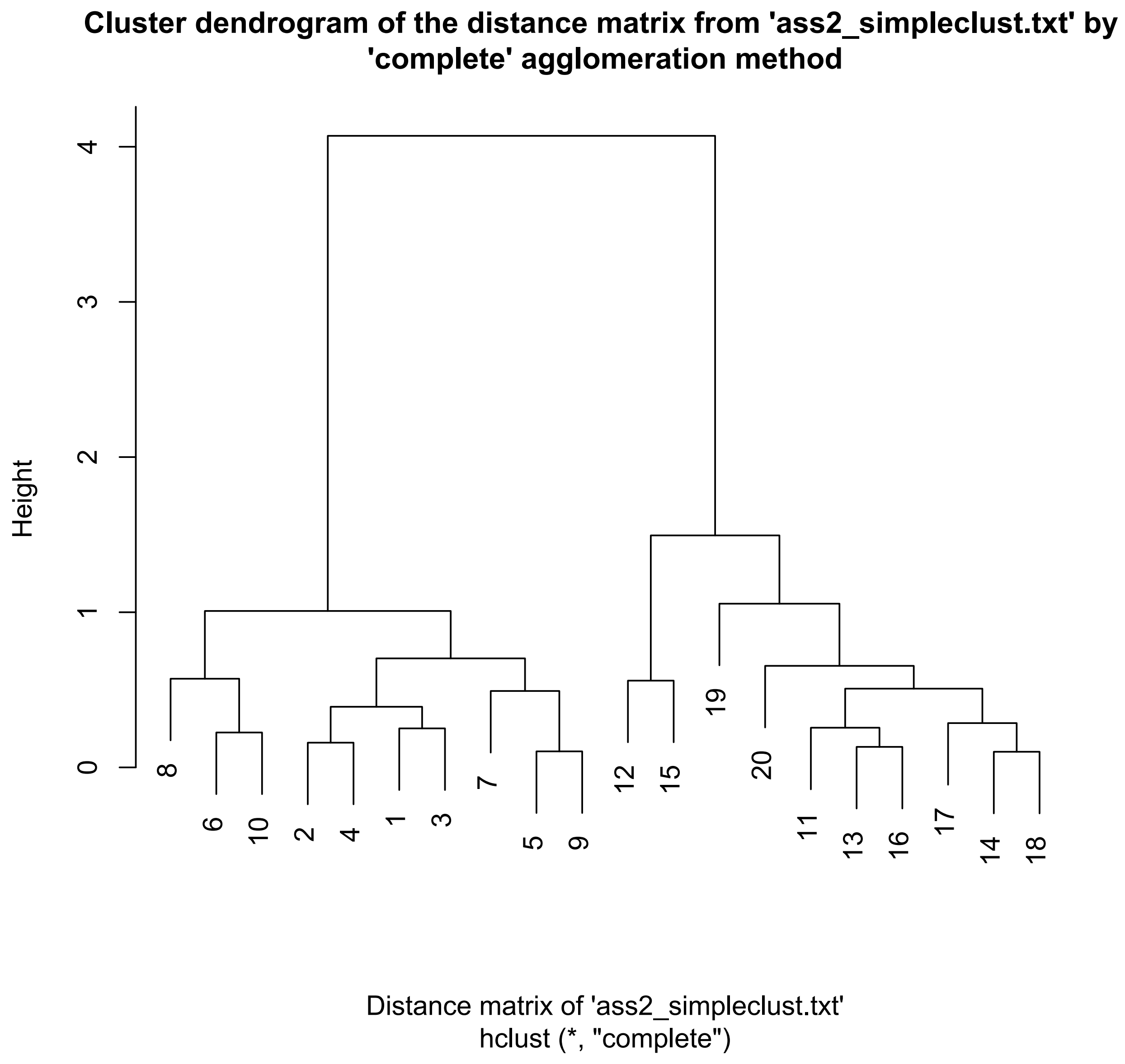
```
#Q1A
#setwd("/Users/siyangli/Documents/Grad1/Statistical modeling and R/")
#the above is where I saved "ass2-simpleclust.txt"
simpleData <- read.table ("ass2-simpleclust.txt", header=FALSE, sep = "",quote="")
names (simpleData) <- c("x", "y")
simpleData
```

```
            x          y
1  -0.7647412 -0.9558243
2  -1.0467150 -0.9686526
3  -0.8178672 -1.2015620
4  -1.1248360 -1.1075870
5  -0.8196271 -0.7104572
6  -1.5135470 -1.2248050
7  -1.2418010 -0.6412505
8  -1.2098340 -1.3957380
9  -0.7772353 -0.8047390
10 -1.7250880 -1.1482980
11  1.0274200  1.2550730
12  0.1577531  0.9985020
13  1.0849540  1.0914870
14  0.6129952  0.9618787
15  0.3095310  0.4602686
16  0.9870233  1.0025200
17  0.7244396  1.2247580
18  0.7137770  0.9633604
19  1.2777040  1.5996010
20  0.9066413  0.6118218
```

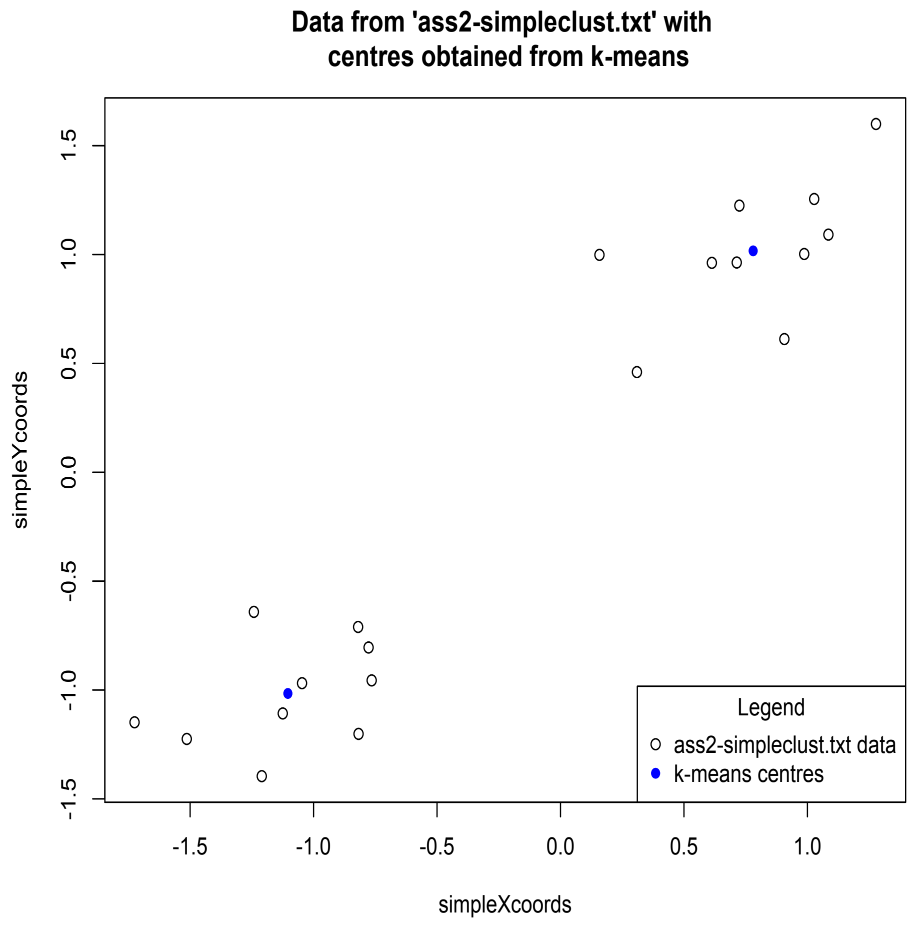
```
# x and y coordinates for one single point, stored as a dataframe

getXcoords <- function (df){ #input is a data.frame with numbers
 return (as.vector(unlist(df$x)))
}
getYcoords <- function (df){
 return (as.vector(unlist(df$y)))
}
simpleXcoords <- getXcoords (simpleData)
simpleYcoords <- getYcoords (simpleData)
plot (simpleXcoords, simpleYcoords, main ="Data from 'ass2-simpleclust.txt'" )
```



```
#Q1B
q1distMat <- dist (simpleData, method = "euclidean", diag=FALSE, upper= FALSE, p=2) #p=power
 of Minkowski distance, p=2 when it is Euclidean
q1completeClust <- hclust (q1distMat) #the default agglomeration method used is "complete"
 and that the given distance matrix is based on distances between single points (not
 clusters)
par () #to visualise current graphical parameters
par (cex.main="1.1") #changed from
plot (q1completeClust,main = "Cluster dendrogram of the distance matrix from
 'ass2_simpleclust.txt' by \n'complete' agglomeration method", xlab = "Distance matrix of
 'ass2_simpleclust.txt'")
```



```
#Q1C
q1kmeans <- kmeans (simpleData, 2) #Default kmeans algorithm is Hartigan and Wong, suggested
 by R
plot (simpleXcoords, simpleYcoords, main ="Data from 'ass2-simpleclust.txt' with \n centres
 obtained from k-means" ) #need to plot the graph again before adding points onto the plot
points (q1kmeans$centers, pch=16, col=12)
q1kmeansLegend <- c("ass2-simpleclust.txt data", "k-means centres")
legend ("bottomright", q1kmeansLegend, col=c(par("col"), 12), pch=c(par("pch"), 16),
title="Legend")
```



```
# I used 2 centres because the data is clearly separated into 2 clusters.
```

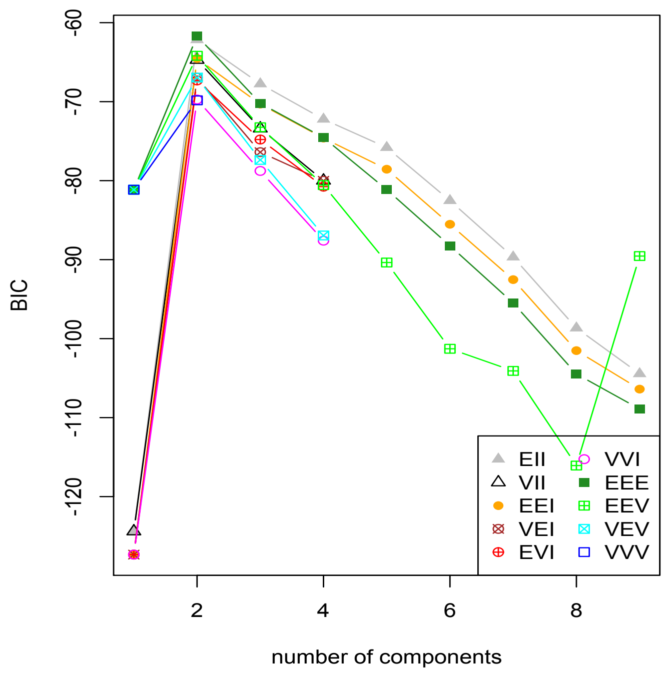
```
#Q1D
#Install mclust package under CRAN binary
library (mclust)
q1Mclust <- Mclust (simpleData) #Gaussian mixture model
q1Mclust
```

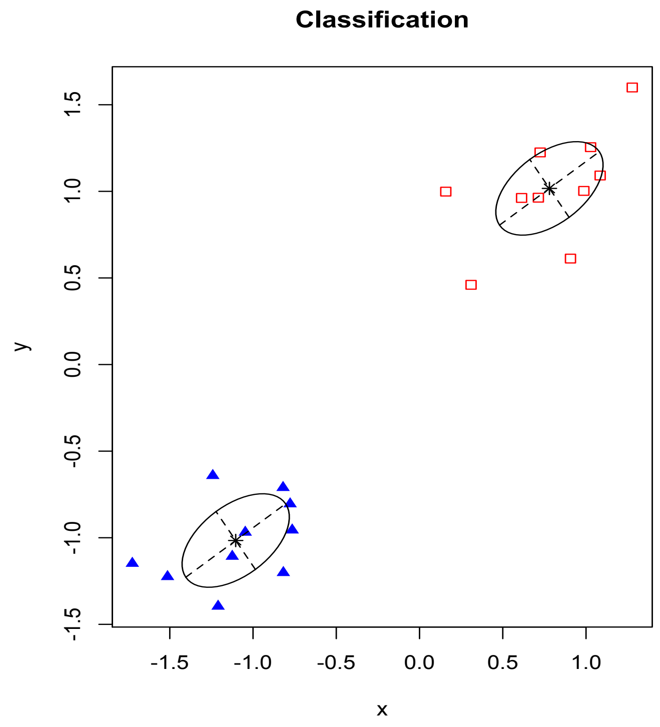
```
best model: elliposidal, equal variance with 2 components
```

```
q1MclustBIC <- mclustBIC (simpleData) #
q1MclustSummary <- summary (q1MclustBIC, data=simpleData)
q1MclustSummary
```

```
classification table:
 1  2
10 10

best BIC values:
    EEE,2     EII,2     EEV,2
-61.68530 -62.16333 -64.19427
```

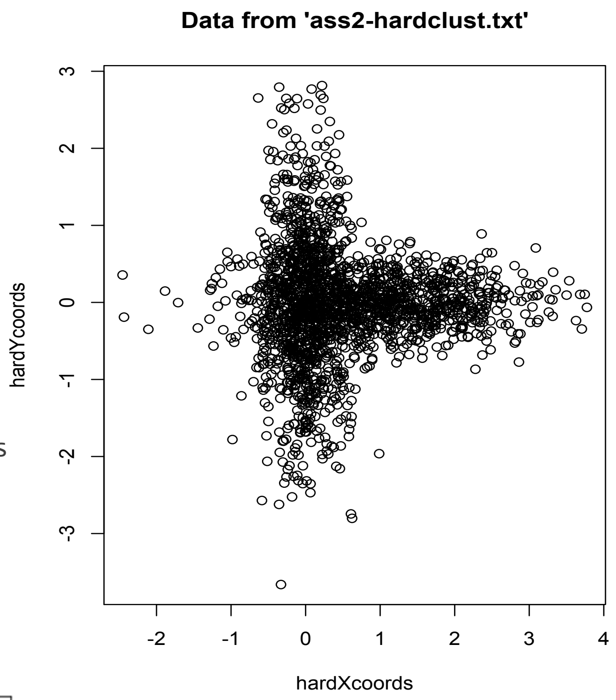
```
plot (q1Mclust, data=simpleData)
```



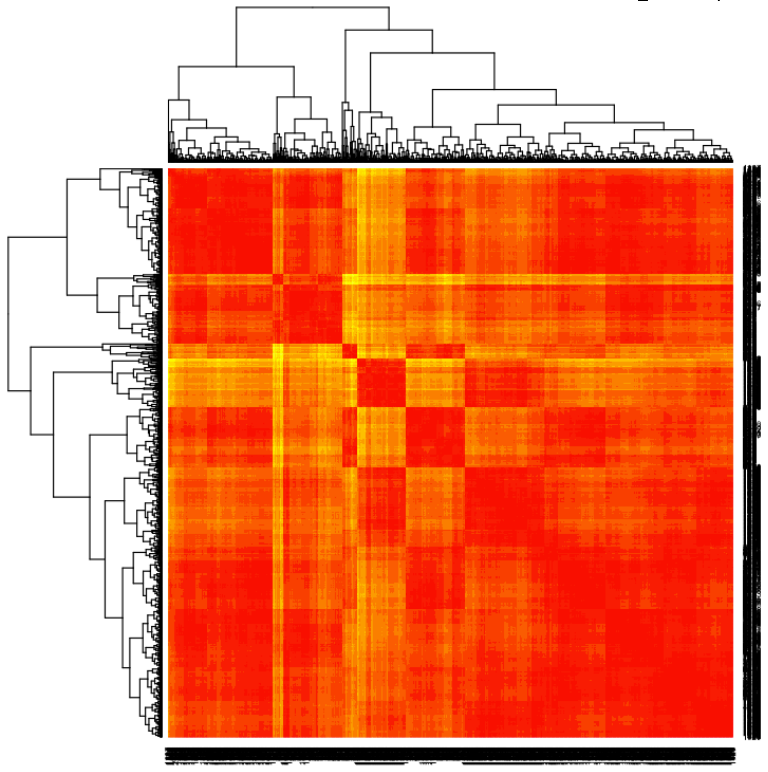


The best model is ellipsoidal, with equal variance and 2 components by Mclust (), which is a Gaussian mixture model (shown above). BIC helps with model selection in statistics. It penalizes for "overfitting", i.e. when the number of parameters is too large. In Mclust(), BIC is calculated as: **$\mathrm{BIC} \equiv 2\ \mathrm{loglik}_M(x, \theta k*) - (\#\ \mathrm{params})_M \log(n)$**
Where loglikM(x, θk∗) is the maximized loglikelihood for the model and data, $(\#\ \mathrm{params})_M$ is the number of independent parameters to be estimated in the model M, and n is the number of observations in the data. Therefore, the larger the number of parameters, the smaller the BIC will be in this particular equation. The best model in Mclust() would be the one with the biggest BIC. In this case, that would be "EEE" ("ellipsoidal, equal variance with 2 components) shown on graph and on summary of BIC results). The data also look like 2 ellipses (diagonal, but rotated on an angle). However, it is important to be cautious with Mclust(), since it could overfit the data. In our case where the clusters are so clearly separated and the spread of the 2 clusters are relatively similar, kmeans() may be more appropriate.

```
#Q2A
hardData <- read.table ("ass2-hardclust.txt")
names (hardData) <- c("x", "y")
hardXcoords <- getXcoords (hardData)
hardYcoords <- getYcoords (hardData)
plot (hardXcoords, hardYcoords, main ="Data from
 'ass2-hardclust.txt'" )
```

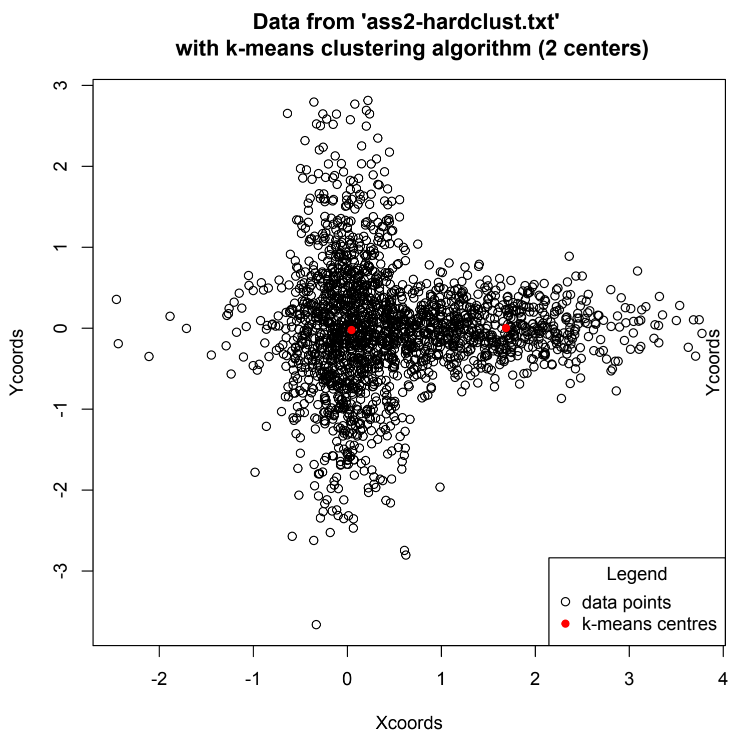


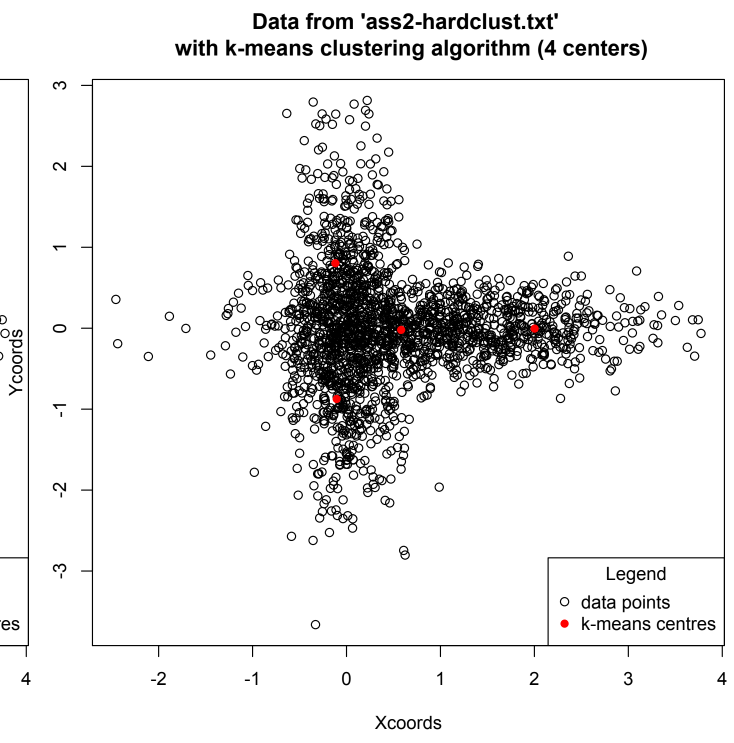
```
#Q2B
q2distMat <- dist (hardData, method = "euclidean",
 diag=FALSE, upper= TRUE, p=2) #p=power of Minkowski
 distance, p=2 when it is Euclidean. dist() calculates
 between ROWS
q2completeClust <- hclust (q2distMat)
q2distMatMA <- as.matrix (q2distMat)
heatmap (q2distMatMA,
 Rowv=as.dendrogram(q2completeClust), Colv="Rowv",
 symm=TRUE)
```

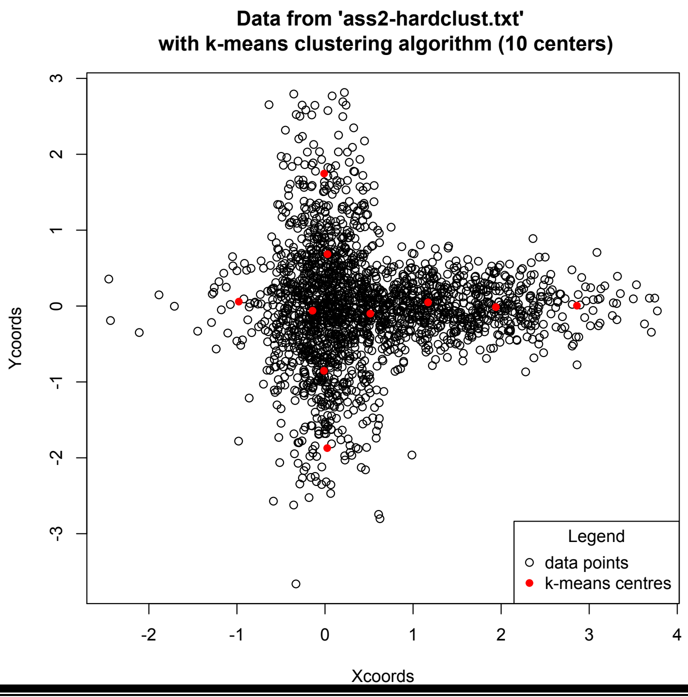


The heat map is symmetrical (as we set it to be in the parameters), and it has a diagonal from the top left corner to the bottom right corner. So we could just look at either one of the halves. The red color signifies a small number (close or equal to 0) in the distance-matrix as the diagonal is where all the points are compared to themselves. The top left corner, the bottom left corner and the bottom right corner are all very closely related (a very small distance between the data, because they are red patches). The patch that is in between the red patches is more ambiguous as they alternate between being closely related and not closely related (yellow). The right bottom patch probably corresponds to the big cluster of points between the 2 ellipses, while the other smaller red patches correspond to the ellipse themselves.

```
#Q2C

# Plots K means with x number of centers

kmeansAndPlot <- function (data, numberOfCenters){ #data is in the form of a data.frame, no
 need to input the algorithm method because we will always be using the default (Hartigan and
 Wong), no need to change the number of iterations (always 10), nstart is always 1.
 kmeansResults <- kmeans (data, numberOfCenters)
 #Calls other functions
 dataXcoords <- getXcoords (data)
 dataYcoords <- getYcoords (data)
 plotCentersKmeans (dataXcoords, dataYcoords, kmeansResults$centers)
}

plotCentersKmeans <- function (Xcoords, Ycoords, centersCoords) {
 plot (Xcoords, Ycoords, main = paste("Data from 'ass2-hardclust.txt' \n with k-means
 clustering algorithm (", nrow(centersCoords), " centers)", sep="" ))
 points (centersCoords, pch=16, col=34)
 kmeansLegend <- c("data points", "k-means centres")
 legend ("bottomright", kmeansLegend, col=c(par("col"), 34, "red"), pch=c(par("pch"), 16),
 title="Legend")

}

kmeansAndPlot (hardData, 2)
kmeansAndPlot (hardData, 4)
kmeansAndPlot (hardData, 10)
```





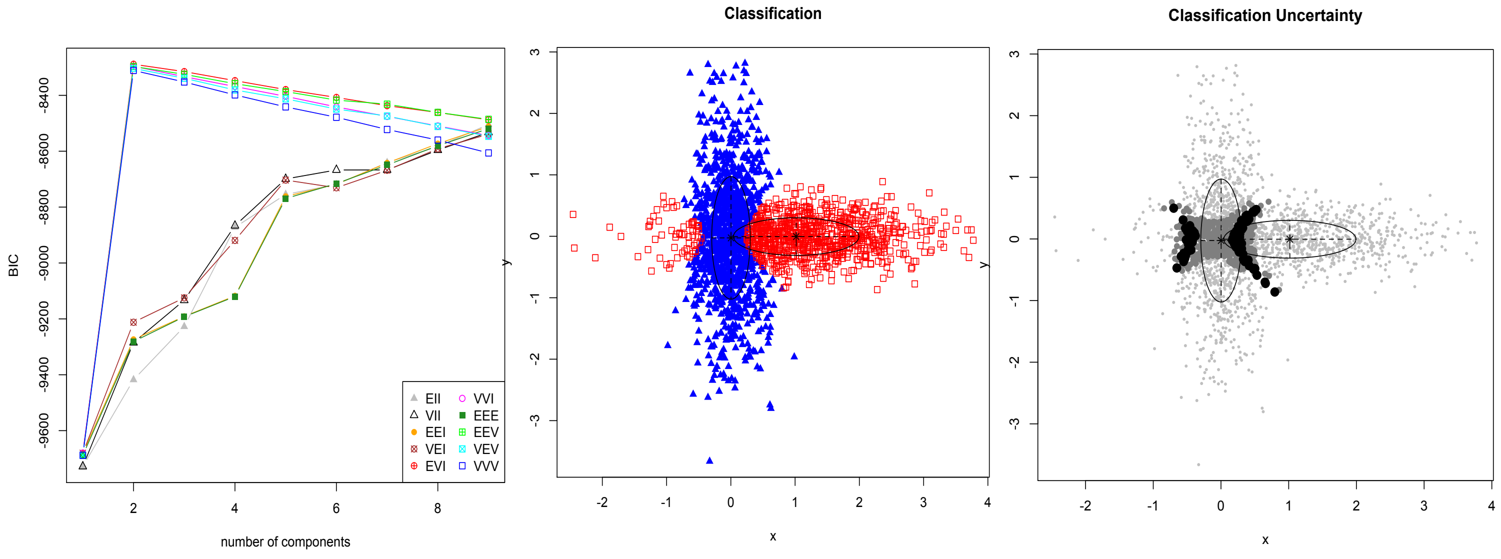


I see that kmeans () doesn't cluster this set of data very well. The default clustering algorithm is Hartigan and Wong (R recommends this as a better algorithm generally). The objective function of kmeans() is to minimize the sum of squared errors. In this particular case, this algorithm doesn't work well because the data is clearly organized into two clusters, one vertical and one horizontal. There is a large area of overlap between the clusters, the objective function of minimizing the sum of squared errors would not be appropriate this case.

```
#Q2D
q2Mclust <- Mclust (hardData)
q2Mclust
```

```
best model: diagonal, equal volume with 2 components
```

```
plot (q2Mclust, data=hardData)
```



```
q2MclustBIC <- mclustBIC (hardData)
q2MclustSummary <- summary (q2MclustBIC, data=hardData)
q2MclustSummary
```

```
classification table:
   1    2
1115  885

best BIC values:
    EVI,2     VVI,2     EEV,2
-8288.759 -8295.998 -8296.798
```

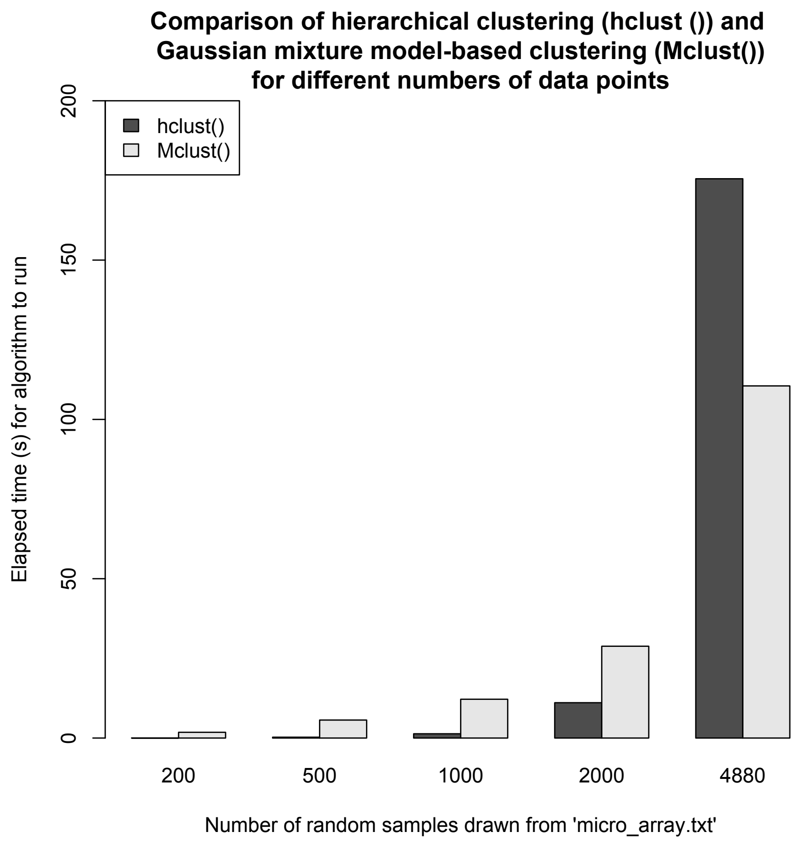
As with question Q1D, I think that mclust () found the right model. The BICs are shown on the first graph, while the "Classification" graph shows all the points in 2 different clusters. The centres are shown with an asterisk with the standard deviations drawn as ellipses around the centres. The best BIC (recall from Q1D that we are looking for the biggest BIC according to R's algorithm) is for the model EVI with 2 centres, which means diagonal distribution with equal volumes and variable shapes oriented on the coordinate axes (no rotation). The classification uncertainty shows a relatively low uncertainty (grey), except around the area where the 2 clusters overlap.

```
#Q3A need to do hclust() and Mclust() and time the R commands on randomly drawn samples from
our list
MicroArray <- read.table ("microarray_data.txt", header=T, sep="\t")
#Data is arranged by columns (each experiment is a new column)
matrixMA <- as.matrix (MicroArray)
matrixMA <- matrixMA [rowSums(is.na(matrixMA))==0,] #removes all genes with missing
 datapoints
MicroArray <- as.data.frame (matrixMA)

#randomly drawing samples
totalGenes <- nrow (MicroArray)
colMicroArray <- length (MicroArray)

sampling <- function (sampleNumber){
 geneIndices <- sample (totalGenes, sampleNumber) #default of replace is FALSE, which makes
 sense for our purposes. This randomly samples integers between 1:totalGenes (number of genes
 in the entire data set).
 randomSamples <- sapply (1:sampleNumber, function(x)matrixMA[geneIndices[x],])
 randomSamples <- t(randomSamples) #transposes the matrix, since dist only performs between
 rows
 randomSamplesNoName <- randomSamples [,2:colMicroArray] #only takes the numeric values for
 each of the gene that got sampled
class (randomSamplesNoName) <- "numeric"
return (randomSamplesNoName)

}

timeCalculation <- function (randomGenes, clustMethod){
 if (clustMethod =="hclust"){
      q3distMat <- dist (randomGenes, method = "euclidean", diag=FALSE, upper= FALSE, p=2)
      q3completeClust <- hclust(q3distMat)
 }else if (clustMethod =="Mclust"){
      Mclust (randomGenes)
 }
}

random200 <- sampling (200)
timeH200 <- system.time (timeCalculation(random200, "hclust")) #system CPU time is the third
 number, so timeH200 [3] would call this time
timeM200 <- system.time (timeCalculation(random200, "Mclust"))
random500 <- sampling (500)
timeH500 <- system.time (timeCalculation(random500, "hclust"))
timeM500 <- system.time (timeCalculation(random500, "Mclust"))
random1000 <- sampling (1000)
timeH1000 <- system.time (timeCalculation(random1000, "hclust"))
timeM1000 <- system.time (timeCalculation(random1000, "Mclust"))
random2000 <- sampling (2000)
timeH2000 <- system.time (timeCalculation(random2000, "hclust"))
timeM2000 <- system.time (timeCalculation(random2000, "Mclust"))

#No need to sample 5000 genes, our data set only has 4880 genes
random4880 <-matrixMA[,2:colMicroArray]
timeH4880 <- system.time (timeCalculation(random4880, "hclust"))
```

```
timeM4880 <- system.time (timeCalculation(random4880, "Mclust"))
timeH <- cbind (timeH200[3], timeH500[3], timeH1000[3], timeH2000[3], timeH4880[3])
timeM <- cbind (timeM200[3], timeM500[3], timeM1000[3], timeM2000[3], timeM4880[3])

combinedData <- rbind (timeH, timeM)
rownames (combinedData) <- cbind ("hclust()", "Mclust()")
colnames (combinedData) <- cbind (200, 500, 1000, 2000, 4880)
combinedData
```

```
           200   500   1000   2000    4880
hclust() 0.012 0.260  1.335 11.085 175.523
Mclust() 1.796 5.645 12.173 28.812 110.511
```

```
barplot (combinedData, main = "Comparison of hierarchical clustering (hclust ()) and
 \nGaussian mixture model-based clustering (Mclust())\nfor different numbers of data points",
 xlab="Number of random samples drawn from 'micro_array.txt'", ylab="Elapsed time (s) for
 algorithm to run", legend = TRUE, beside=TRUE, args.legend = list(x="topleft"),
 ylim=c(0,200)) #legend=TRUE will just print the names of the rows of combined, args.legend
 sets the legend to topleft of the graph
```

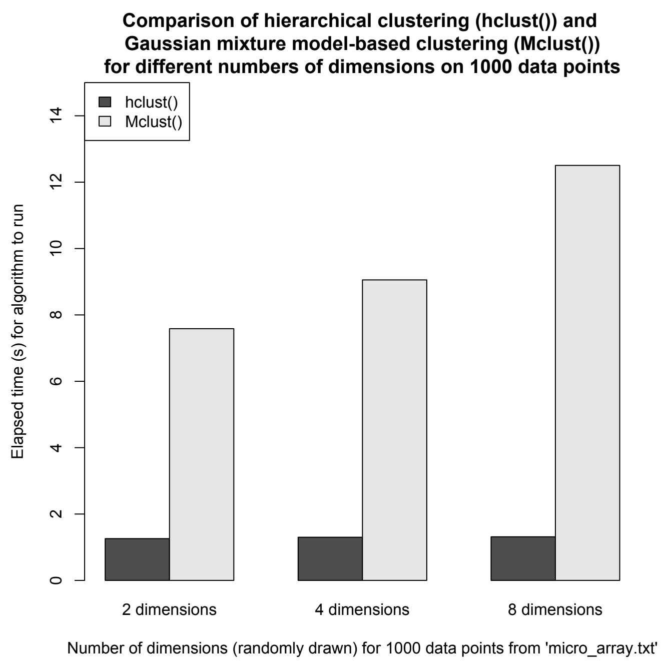


```
The elapsed time increases as more data points are used in the clustering analyses. However,
 hclust() runs relatively fast for smaller data sets as it is faster than Mclust() at 200,
 500, 1000, and 2000 points (8 dimensions). However, the time for hclust() to run grows much
 faster as the data set size increases, shown by the 4880 data points: hclust() becomes much
 slower than Mclust().
```

```
#Q3B, let's keep the number of datapoints at 1000
random1000for3B <- sampling (1000)

randomMicroArrayExp <- function (numberOfExperiments){
 columns3B <- sample (1:colMicroArray-1, numberOfExperiments) #samples between the column
 numbers that contain experimental values in random1000for3B, which only has numeric values
class (random1000for3B) <- "numeric"
return (random1000for3B[,columns3B]) #returns all the columns that were selected under the
 random column numbers
}
random1000_2exps <- randomMicroArrayExp (2)
random1000_4exps <- randomMicroArrayExp (4)
#for the 8 microarray experiments, no need to sample, since we only have 8 micro array
 experiments in total!
random1000_8exps <- random1000for3B
timeH2exps <- system.time (timeCalculation(random1000_2exps, "hclust"))
timeM2exps <- system.time (timeCalculation(random1000_2exps, "Mclust"))
timeH4exps <- system.time (timeCalculation(random1000_4exps, "hclust"))
timeM4exps <- system.time (timeCalculation(random1000_4exps, "Mclust"))
timeH8exps <- system.time (timeCalculation(random1000_8exps, "hclust"))
timeM8exps <- system.time (timeCalculation(random1000_8exps, "Mclust"))
timeHdimensions <- cbind (timeH2exps[3], timeH4exps[3], timeH8exps[3])
timeMdimensions <- cbind (timeM2exps[3], timeM4exps[3], timeM8exps[3])
combinedDims<- rbind (timeHdimensions, timeMdimensions)
rownames (combinedDims) <- cbind ("hclust()", "Mclust()")
colnames (combinedDims) <- cbind ("2 dimensions", "4 dimensions", "8 dimensions")
combinedDims
```

```
         2 dimensions 4 dimensions 8 dimensions
hclust()        1.259        1.301        1.314
Mclust()        7.585        9.055       12.505
```

```
barplot (combinedDims, main = "Comparison of hierarchical clustering (hclust()) and
 \nGaussian mixture model-based clustering (Mclust())\nfor different numbers of dimensions on
 1000 data points", xlab="Number of dimensions (randomly drawn) for 1000 data points from
 'micro_array.txt'", ylab="Elapsed time (s) for algorithm to run", legend = TRUE,
 beside=TRUE, args.legend = list(x="topleft"), ylim=c(0,15)) #legend=TRUE will just print the
 names of the rows of combined, args.legend sets the legend to topleft of the graph
```



```
Increase in the number of dimensions increases the time that it takes Mclust () to run, but
 doesn't affect hclust () at a fixed number of data points.
```

```
#Q3C
q3 <- cbind
 (as.numeric(matrixMA[,2]),as.numeric(matrixMA[,3]),as.numeric(matrixMA[,4]),as.numeric(matri
 xMA[,5]),as.numeric(matrixMA[,6]),as.numeric(matrixMA[,7]),as.numeric(matrixMA[,8]),
 as.numeric(matrixMA[,9]))  #they were somehow stored as strings in a matrix before and this
 is the only way to convert them. Actually, could do class(x) <- "numeric", but didn't want
 to change the entire code.

q3mclust <- Mclust (q3)
q3mclust
```

```
best model: ellipsoidal, equal shape with 6 components
```

```
Best model: ellipsoidal, equal shape with 6 components. But 6 clusters may not always be
 right, since the question asks for us to use kmeans (), the below tries a few different
 centres for kmeans ()
```

```
q3_4kmeans <- kmeans (q3, 4)
q3_5kmeans <- kmeans (q3, 5)
q3_6kmeans <- kmeans (q3, 6)
q3_7kmeans <- kmeans (q3, 7)
q3_8kmeans <- kmeans (q3, 8)

q3_4kmeans$size
```

```
 [1] 1763  656 2453    8
```

```
q3_5kmeans$size
```

```
[1] 1128  855 1635   38 1224
```

```
q3_6kmeans$size
```

```
[1] 1076 1040    8 1593  145 1018
```

```
q3_7kmeans$size
```

```
[1] 1413  827  664 1010    8  839  119
```

```
q3_8kmeans$size
```

```
[1] 1019  646  107 1072  336  666 1026    8
```

```
Cluster of 8 consistently appears with centres >= 6. Because of the information that we were
 given by the question, we can assume that this cluster of 8 data points is the consistently
 highly expressed genes. To check, we can look at the centers for kmeans () under 6 centers
```

```
q3_6kmeans$centers
```

```
         [,1]       [,2]         [,3]       [,4]       [,5]       [,6]        [,7]        [,8]
1 -0.30553903 -0.1206413 -0.136440520 -0.2913383 -0.1506877 -0.9228067 -0.09254647 -0.03329926
2  0.28615385  0.1099038 -0.081182692 -0.1840577  0.3154038 -0.5004519  0.45731731  0.15115385
3  3.95500000  3.9825000  4.142500000  4.5850000  4.5275000  2.8587500  3.38500000  2.55500000
4  0.07532957 -0.0269366  0.001726303  0.4189642  0.0863779 -0.2814878  0.25924043 -0.08081607
5  0.50075862  0.1266897  0.122965517  2.7551724  0.6223448  0.2002069  0.66075862  0.17482759
6  0.48610020  0.1480059  0.119125737  0.6590570  0.4824263  0.2914931  0.88402750  0.12376228
```

```
#Center 3 (size = 8), has a center that is consistently positive across all dimensions, this
is the small cluster of genes that we are looking for.

MAwithClust <- cbind (matrixMA, classification=q3_6kmeans$cluster) #binds the cluster
 classifications to the end of the matrix
q3cluster <- MAwithClust [which(MAwithClust[,10]==3),] #the center number changes every time
 you run the algorithm, one way to avoid this would be to rearrange the centers such that the
 most positive center would always be listed as the first center.
q3cluster
```

```
     UID       dby....DBY7286.Low.Pi.vs..High.Pi..dby
 pho4c..PHO4c.vs..wild.type..pho4c pho80..pho80.mutant.vs.wild.type..pho80
 pho81..PHO81c.vs..wild.type..exp.1..pho81..
 pho812.PHO81c.vs..wild.type..exp.2..pho812.
[1,] "YJL012C" " 2.96"                                      " 3.23"
 " 3.13"                                      " 3.74"
 " 3.19"
[2,] "YHR215W" " 4.34"                                      " 4.14"
 " 4.66"                                      " 5.65"
 " 5.34"
[3,] "YBR093C" " 3.22"                                      " 2.85"
 " 4.34"                                      " 4.44"
 " 4.19"
[4,] "YDR281C" " 3.46"                                      " 3.03"
 " 3.43"                                      " 4.06"
 " 3.76"
[5,] "YHR136C" " 4.67"                                      " 4.58"
 " 5.29"                                      " 4.53"
 " 5.34"
[6,] "YAR071W" " 3.82"                                      " 4.30"
 " 4.45"                                      " 5.30"
 " 5.06"
[7,] "YPL019C" " 4.05"                                      " 4.27"
 " 4.36"                                      " 5.10"
 " 4.22"
[8,] "YBR296C" " 5.12"                                      " 5.46"
 " 3.48"                                      " 3.86"
 " 5.12"
```

```
     pho85..pho85.delete.vs..wild.type..pho85
 lowpho.NBW7.strain.High.Pi.vs..Low.Pi..exp.2..lowpho.
 nbw....NBW7.strain.low.Pi.vs..High.Pi..exp.1..nbw.... classification
[1,] " 2.12"                                        " 2.55"
 " 2.06"                                                      "3"
[2,] " 3.75"                                        " 4.57"
 " 3.04"                                                      "3"
[3,] " 3.31"                                        " 3.55"
 " 1.23"                                                      "3"
[4,] "-0.08"                                        " 1.97"
 " 1.23"                                                      "3"
[5,] " 3.42"                                        " 3.22"
 " 3.49"                                                      "3"
[6,] " 3.79"                                        " 4.28"
 " 2.99"                                                      "3"
[7,] " 2.67"                                        " 3.55"
 " 3.43"                                                      "3"
[8,] " 3.89"                                        " 3.39"
 " 2.97"                                                      "3"
```

```
#Confirmation that these genes have consistent increased expression as all dimensions are
 positive

q3clusterNames <- q3cluster[,1]
q3clusterNames <- as.data.frame (q3clusterNames) #needs a data frame for getGOinList
q3clusterNames
```

```
  q3clusterNames
1        YJL012C
2        YHR215W
3        YBR093C
4        YDR281C
5        YHR136C
6        YAR071W
7        YPL019C
8        YBR296C
```

```
# Download org.Sc.sgd.db package and GO.db package
library ("org.Sc.sgd.db")
ygenes <- org.Sc.sgdGO
library ("GO.db")

#Some functions from previous assignment
#Function that gets all the GO terms in a gene list

getGOinList <- function (geneList){ #geneList should be a dataframe and the first column
 should be entrez genes
 sampleGOIDs <- list () #empties list from previous usage
 sampleGOIDs <- sapply (1:nrow(geneList),
 function(x)names(ygenes[[as.character(geneList[x,1])]]))
 #geneList[i,1] looks up the entrez ID through the entire list and the names function
 retrieves all the GO IDs that is associated with each entrez gene. I used as.character here
 to convert the entrez ID into a string, as I said previously that looking up with the index
 number would be incorrect. Instead of doing this in a loop, I chose to use the sapply
 function because this is much faster (it applies the function simultaneously to all vectors
 in the list).
geneList$GOIDs <- sampleGOIDs
return (geneList)
}

clusterGO <- getGOinList (q3clusterNames)
justGOsample <- unique (unlist (clusterGO$GOIDs))
justGOsample
```

```
[1] "GO:0006797" "GO:0007034" "GO:0016237" "GO:0042144" "GO:0005773" "GO:0005774"
 "GO:0005783" "GO:0016020" "GO:0016021" "GO:0031310" "GO:0033254" "GO:0008976" "GO:0008150"
 "GO:0016311" "GO:0000324" "GO:0003993"
[17] "GO:0016787" "GO:0016791" "GO:0006796" "GO:0008361" "GO:0016036" "GO:0005576"
 "GO:0009277" "GO:0030287" "GO:0017111" "GO:0047429" "GO:0003674" "GO:0009266" "GO:0005737"
 "GO:0004857" "GO:0004860" "GO:0000329"
[33] "GO:0006810" "GO:0006817" "GO:0055085" "GO:0005886" "GO:0005315" "GO:0015293"
 "GO:0015319"
```

```
#All 39 unique GO terms from our list of genes (8)

#All the ORFs that are linked to a GO, this is our universe...
orfsWithGO <- mappedkeys (ygenes)
numberOfGenesUniverse <- nrow(orfsWithGO)
orfsWithGO <- as.data.frame (orfsWithGO)
universeGO <- getGOinList (orfsWithGO)
justGOuniverse <- unique (unlist(universeGO$GOIDs)) #All GO terms that are present in our
 universe

hyperTest <- function (sampleGOcounts, universeGOcounts){
	p <- phyper (sampleGOcounts-1, universeGOcounts, nrow(orfsWithGO), nrow(clusterGO) ,
 lower.tail = FALSE, log.p=FALSE) #totalGO is the total number of GO terms, because we
 already tested the GO terms that are not in our sample list, but we still need to correct
 for it. 6359 genes in our universe and 8 genes in our sample
	return (p)
}

checkingEveryGOwithHyper <- function (GOIDs, sampleGenes){
#no need for universe, always the same. The sampleGenes should have GOIDs as a header and
 should be a data.frame.

sampleListTOI <- vector () #empties vector from previous usage
universeListTOI <- vector ()
hyperTestResults <- vector ()
for (i in 1:length(GOIDs)){ #for every GO (vector of strings)
sampleListCounts <- sum(sapply (sampleGenes$GOIDs, function(x)GOIDs[i]%in%x)) #Use sapply to
 look for the GO ID in our sample list's column "GOIDs". Sapply returns boolean variables,
 and the sum just takes the number of terms that are TRUE in sapply. So this just returns the
 number of genes in the list with the GOID that we're interested in. Since we are using a
 boolean function, it would ignore GO terms appearing more than once under the same gene
 (avoiding annotation mistakes)
universeListCounts <- sum(sapply (universeGO$GOIDs, function(x)GOIDs[i]%in%x))
#The universe is always the same (to simplify things)
sampleListTOI <- append (sampleListTOI, sampleListCounts)
universeListTOI <- append (universeListTOI, universeListCounts)
hyperTestResults <- append (hyperTestResults, hyperTest(sampleListCounts,
 universeListCounts))
#Calls the function with these parameters.

}

sampleGenesAllPvalues <- list (GOIDs=GOIDs, sample_counts=sampleListTOI,
 universe_counts=universeListTOI, hypergeom_pvalues=hyperTestResults) #this puts all info
 together into one var as R functions can't return more than 1 var
sampleGenesAllPvalues <- as.data.frame (sampleGenesAllPvalues) #converts to df, required
 format for following functions
return (sampleGenesAllPvalues)

}


#Significance test
significanceTest <- function (alpha, geneList){ #geneList should be in the format of the
 output of checkingEveryGOwithHyper
 significantIndices <- which(geneList$hypergeom_pvalues < alpha)
 significantTerms <- list(GOIDs = geneList$GOIDs[significantIndices], sample_counts =
 geneList$sample_counts[significantIndices], universe_counts =
 geneList$universe_counts[significantIndices],
 hypergeom_pvalues=geneList$hypergeom_pvalues[significantIndices])
 significantTerms <- as.data.frame (significantTerms)
 return (significantTerms)
}

retrieveGOterm <- function (significantTerms){ #significantList is the combined matrix from
 printListWithoutGOterm
 if (nrow (significantTerms)==0){
       return ("No significant terms")
 }else{
 GOterms <- sapply (1:nrow(significantTerms),
 function(x)as.character(Term(as.character(significantTerms[[x,1]]))))
 significantTerms <- cbind (GOterms=GOterms, significantTerms)
 significantTerms <- as.data.frame (significantTerms)
 return(significantTerms)
}
}
```

```
GOclustAllP <- checkingEveryGOwithHyper (justGOsample, clusterGO)
print (data.frame (GOclustAllP))
```

```
        GOIDs sample_counts universe_counts hypergeom_pvalues
1  GO:0006797             2               8      3.853952e-05
2  GO:0007034             2              17      1.856128e-04
3  GO:0016237             2              10      6.182193e-05
4  GO:0042144             2              29      5.479002e-04
5  GO:0005773             3             162      7.689555e-04
6  GO:0005774             3             134      4.461889e-04
7  GO:0005783             2             418      8.298848e-02
8  GO:0016020             4            1676      6.440501e-02
9  GO:0016021             4            1309      3.317401e-02
10 GO:0031310             1               2      2.513944e-03
11 GO:0033254             2               4      8.289671e-06
12 GO:0008976             1               1      1.257862e-03
13 GO:0008150             2            1220      3.774901e-01
14 GO:0016311             3              32      6.279337e-06
15 GO:0000324             2              97      5.898169e-03
16 GO:0003993             3               9      1.089627e-07
17 GO:0016787             3             640      3.004125e-02
18 GO:0016791             3              29      4.640623e-06
19 GO:0006796             2               8      3.853952e-05
20 GO:0008361             1              30      3.697290e-02
21 GO:0016036             1               4      5.020779e-03
22 GO:0005576             2             102      6.495209e-03
23 GO:0009277             1              88      1.041751e-01
24 GO:0030287             1               9      1.125692e-02
25 GO:0017111             1              92      1.086116e-01
26 GO:0047429             1               4      5.020779e-03
27 GO:0003674             2            2008      6.075893e-01
28 GO:0009266             1               1      1.257862e-03
29 GO:0005737             1            2119      8.999344e-01
30 GO:0004857             1               7      8.767755e-03
31 GO:0004860             1               3      3.768249e-03
32 GO:0000329             1             118      1.368534e-01
33 GO:0006810             1             817      6.199565e-01
34 GO:0006817             1              10      1.249886e-02
35 GO:0055085             1             306      3.135301e-01
36 GO:0005886             1             355      3.526208e-01
37 GO:0005315             1               5      6.271539e-03
38 GO:0015293             1               8      1.001322e-02
39 GO:0015319             1               1      1.257862e-03
```

```
#All P-values associated with each tested GO term by the hypergeometric test
```

```
alpha <- 0.05
GOclustSigP <- significanceTest (alpha, GOclustAllP)
final <- retrieveGOterm (GOclustSigP)
final
```

```
                                             GOterms      GOIDs sample_counts universe_counts hypergeom_pvalues
1                      polyphosphate metabolic process GO:0006797             2               8      3.853952e-05
2                                  vacuolar transport GO:0007034             2              17      1.856128e-04
3                                     microautophagy GO:0016237             2              10      6.182193e-05
4                        vacuole fusion, non-autophagic GO:0042144             2              29      5.479002e-04
5                                            vacuole GO:0005773             3             162      7.689555e-04
6                                  vacuolar membrane GO:0005774             3             134      4.461889e-04
7                                integral to membrane GO:0016021             4            1309      3.317401e-02
8                       intrinsic to vacuolar membrane GO:0031310             1               2      2.513944e-03
9                  vacuolar transporter chaperone complex GO:0033254             2               4      8.289671e-06
10                      polyphosphate kinase activity GO:0008976             1               1      1.257862e-03
11                                  dephosphorylation GO:0016311             3              32      6.279337e-06
12                                fungal-type vacuole GO:0000324             2              97      5.898169e-03
13                           acid phosphatase activity GO:0003993             3               9      1.089627e-07
14                                 hydrolase activity GO:0016787             3             640      3.004125e-02
15                                phosphatase activity GO:0016791             3              29      4.640623e-06
16                          phosphate metabolic process GO:0006796             2               8      3.853952e-05
17                             regulation of cell size GO:0008361             1              30      3.697290e-02
18             cellular response to phosphate starvation GO:0016036             1               4      5.020779e-03
19                                extracellular region GO:0005576             2             102      6.495209e-03
20                  cell wall-bounded periplasmic space GO:0030287             1               9      1.125692e-02
21         nucleoside-triphosphate diphosphatase activity GO:0047429             1               4      5.020779e-03
22                    response to temperature stimulus GO:0009266             1               1      1.257862e-03
23                           enzyme inhibitor activity GO:0004857             1               7      8.767755e-03
24                     protein kinase inhibitor activity GO:0004860             1               3      3.768249e-03
25                                 phosphate transport GO:0006817             1              10      1.249886e-02
26 inorganic phosphate transmembrane transporter activity GO:0005315             1               5      6.271539e-03
27                                 symporter activity GO:0015293             1               8      1.001322e-02
28          sodium:inorganic phosphate symporter activity GO:0015319             1               1      1.257862e-03
```

```
#All significant GO terms at a=0.05, no multiple hypothesis testing correction


#Multiple testing correction with Bonferroni
BonfAllP <- GOclustAllP
BonfAllP$hypergeom_pvalues <-  p.adjust (BonfAllP$hypergeom_pvalues, method = "bonferroni",
 length(justGOuniverse)) #this corrects for ALL GO terms in the universe, because we removed
 GO terms that we knew that wouldn't be enriched for (because they didn't exist in the sample
 list), but they also count as hypotheses. There are 4422 GO terms in the
BonfsigTerms <- significanceTest (alpha, BonfAllP)
BonfsigGOterms <- retrieveGOterm (BonfsigTerms)
names (BonfsigGOterms) <- c("GOterms", "GOIDs", "sample_counts", "universe_counts",
 "corrected_pvalues")
BonfsigGOterms
```

```
                                GOterms      GOIDs sample_counts universe_counts corrected_pvalues
1 vacuolar transporter chaperone complex GO:0033254             2               4      0.0366569265
2                     dephosphorylation GO:0016311             3              32      0.0277672304
3             acid phosphatase activity GO:0003993             3               9      0.0004818333
4                  phosphatase activity GO:0016791             3              29      0.0205208327
```

```
#Multiple testing correction with FDR
FDRAllP <- GOclustAllP
FDRAllP$hypergeom_pvalues <-  p.adjust (FDRAllP$hypergeom_pvalues, method = "fdr",
 length(justGOuniverse))
FDRsigTerms <- significanceTest (alpha, FDRAllP)
FDRsigGOterms <- retrieveGOterm (FDRsigTerms)
names (FDRsigGOterms) <- c("GOterms", "GOIDs", "sample_counts", "universe_counts",
 "corrected_pvalues")
FDRsigGOterms
```

```
                               GOterms      GOIDs sample_counts universe_counts corrected_pvalues
1        polyphosphate metabolic process GO:0006797             2               8      0.0284036287
2                        microautophagy GO:0016237             2              10      0.0390537933
3 vacuolar transporter chaperone complex GO:0033254             2               4      0.0091642316
4                     dephosphorylation GO:0016311             3              32      0.0091642316
5              acid phosphatase activity GO:0003993             3               9      0.0004818333
6                   phosphatase activity GO:0016791             3              29      0.0091642316
7             phosphate metabolic process GO:0006796             2               8      0.0284036287
```

As usual, the FDR is more lenient than the Bonferroni correction test – 7 significant GO terms versus 5 significant GO terms from the Bonferroni. In both multiple hypothesis correction results, enriched GO terms seem to be involved in **phosphate metabolic processes** (such as phosphatase, dephosphorylation, and polyphosphate processes). Most significant result being acid phosphatase activity, p=0.00048). Vacuolar transport is also significant.