BIO 754 - Lecture 11

04-05-2017

Contents

Permutation (randomization) tests (continued)

You had calculated the sex effect permutation results as homework. We start by loading that object `nmat6_aovp_perm2`:

```
load("liver_transcriptome_v1.Rdata")
load("liver_transcriptome_v2.Rdata")
load("liver_transcriptome_v3.Rdata")
load("liver_transcriptome_v4.Rdata")
load("liver_transcriptome_v5.Rdata")
```

Let's calculate the permutation p-value for sex again:

```
obs = mean(nmat6_aovp[,2] < 0.05) # number of genes
exp = sapply(nmat6_aovp_perm2, function(x) mean(x[,2] < 0.05) )
mean(exp >= obs)
```

```
## [1] 0.5
```

As you see, the overall result, in terms of genes with `p<0.05` for the sex effect, is not significant (but it could be significant at more stringent p-value cutoffs - we haven't checked).

The structure of permutation tests

Here, it is important that you use the **same reshuffled** vector on all genes each permutation round. Otherwise you will be randomizing not just across individuals, but also **across genes**, which are actually not independent. You will thus remove the effect of individual differences: e.g. some individuals having extreme expression levels for many genes. Consequently, you will obtain **very uniform** p-value distributions. Instead, we want to keep everything as is, retain individual variation for the whole transcriptome.

Likewise, we wish to retain the grouping based on sex.

It is also important that you retain the sex effect in second place inside the `aov` function (`y ~ species2 + randomsex2`), when testing for sex. This is because: in random permutations, `randomsex2` will frequently become correlated, i.e. **confounded**, with the species effect. For example all macaques could appear as male.

If we had placed `randomsex2` before `species2` in the ANOVA model, `randomsex2` could sometimes explain most of the variance in the expression, and appear very significant. This we do not wish - it will be biasing ourselves against the sex effect.

Let's try to demonstrate these effects, using only the first 2000 genes, and 4 permutations, for the sake of time:

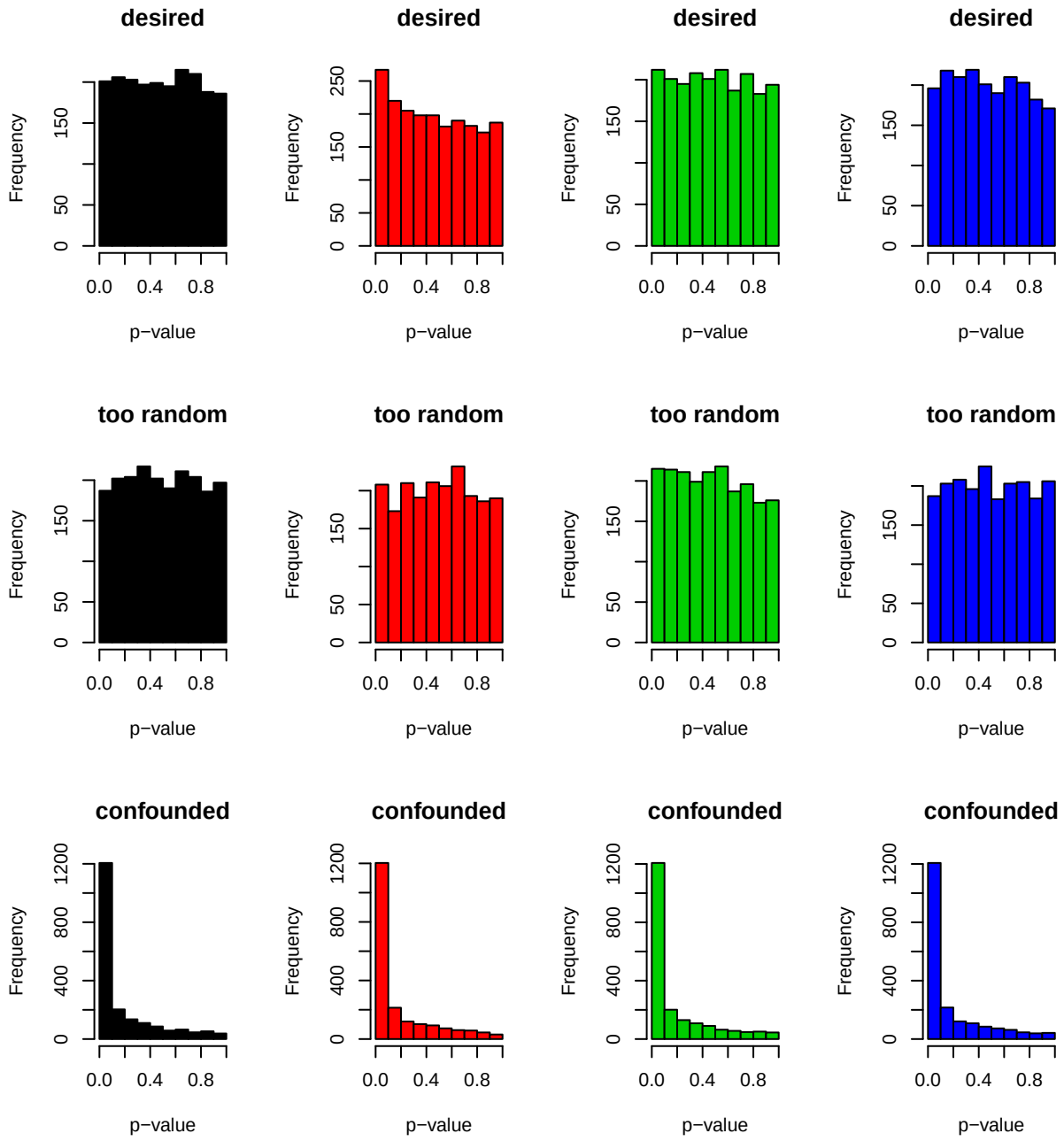
```
# first check some examples
y = nmat6[1,]
randomsex2 = sample(sex2)
anova(aov(y ~ species2 * randomsex2))$Pr[1:3]
randomsex2 = sample(sex2)
anova(aov(y ~ species2 * randomsex2))$Pr[1:3]

toomuchrandomization = lapply(1:4, function(i) {
  t(apply(nmat6[1:2000,], 1, function(y) {
    # randomize separately for each gene
    anova(aov(y ~ species2 * sample(sex2)))$Pr[1:3]
  }))
})
```

Now allow confounding sex with species effects. More specifically, `sex2` is uncorrelated with `species2`, because the design is balanced. But each time `randomsex2` becomes correlated with `species2`, it is able to explain a significant part of the variance:

```
sexeffectconfounded = lapply(1:4, function(i) {
  t(apply(nmat6[1:2000,], 1, function(y) {
    randomsex2 = sample(sex2)
    # writing sex before species means gives priority to the former
    anova(aov(y ~ randomsex2 * species2 ))$Pr[1:3]
  }))
})
```

Compare the results:

```
par(mfrow=c(3,4))
for (i in 1:4) {
  hist(nmat6_aovp_perm2[[i]][1:2000,2], col=i, xlab='p-value', main='desired')
}
for (i in 1:4) {
  hist(toomuchrandomization[[i]][1:2000,2], col=i, xlab='p-value', main='too random')
}
for (i in 1:4) {
  hist(sexeffectconfounded[[i]][1:2000,2], col=i, xlab='p-value', main='confounded')
}
```



If we had used `toomuchrandomization` to represent the H0, this could increase the Type I error rate (rejecting a true H0). This because we randomize the whole dataset and do not allow any excess of low p-values (e.g. as in the 2nd case of `nmat6_aovp_perm2` in the figure above). For example, outlier individuals vanish under the `toomuchrandomization` scheme.

If we used the `sexeffectconfounded`, we would never be able to detect a slight but significant sex effect, because we would be practically comparing the sex effect with the species effect (which is much stronger).

Before continuing, let's save the permutation results as a separate file as it will be large.

```
save(nmat6_aovp_perm2, toomuchrandomization, sexeffectconfounded, file="liver_transcriptome_v6.Rdata")
```

Clustering genes

Heatmaps using `heatmap.2`

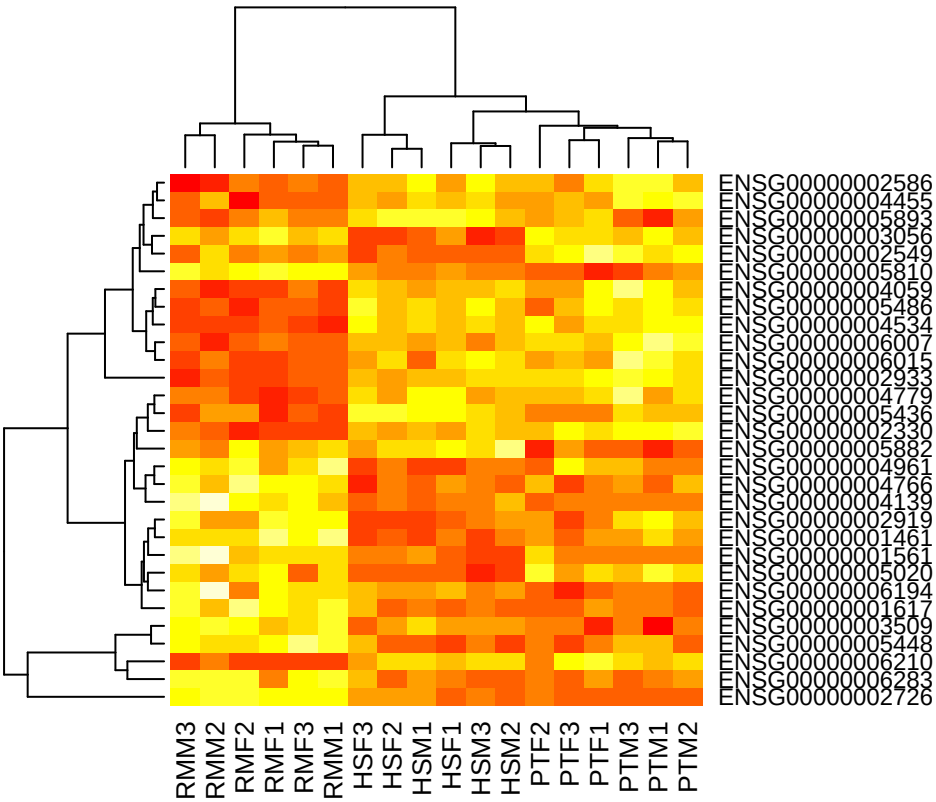
Now we will learn to cluster genes based on their expression profiles (as opposed to clustering individuals). Our goal is to sort genes that show differential expression w.r.t. the species effect, group them according to their profiles, study the DE patterns, and then test for possible functional differences among these gene groups.

One approach is to use hierarchical clustering again. Here we can apply the `heatmap` function. This will cluster both genes and samples, and also plot the names of genes and samples (which takes too much time, so we suppress this behaviour):

```
spDEGenes = rownames(nmat6)[nmat6_aovq[,1] < 0.05]
length(spDEGenes)
```

```
## [1] 3097
```

```
# just for the first 100 genes
heatmap( head(nmat6[spDEGenes, ], 30) )
```



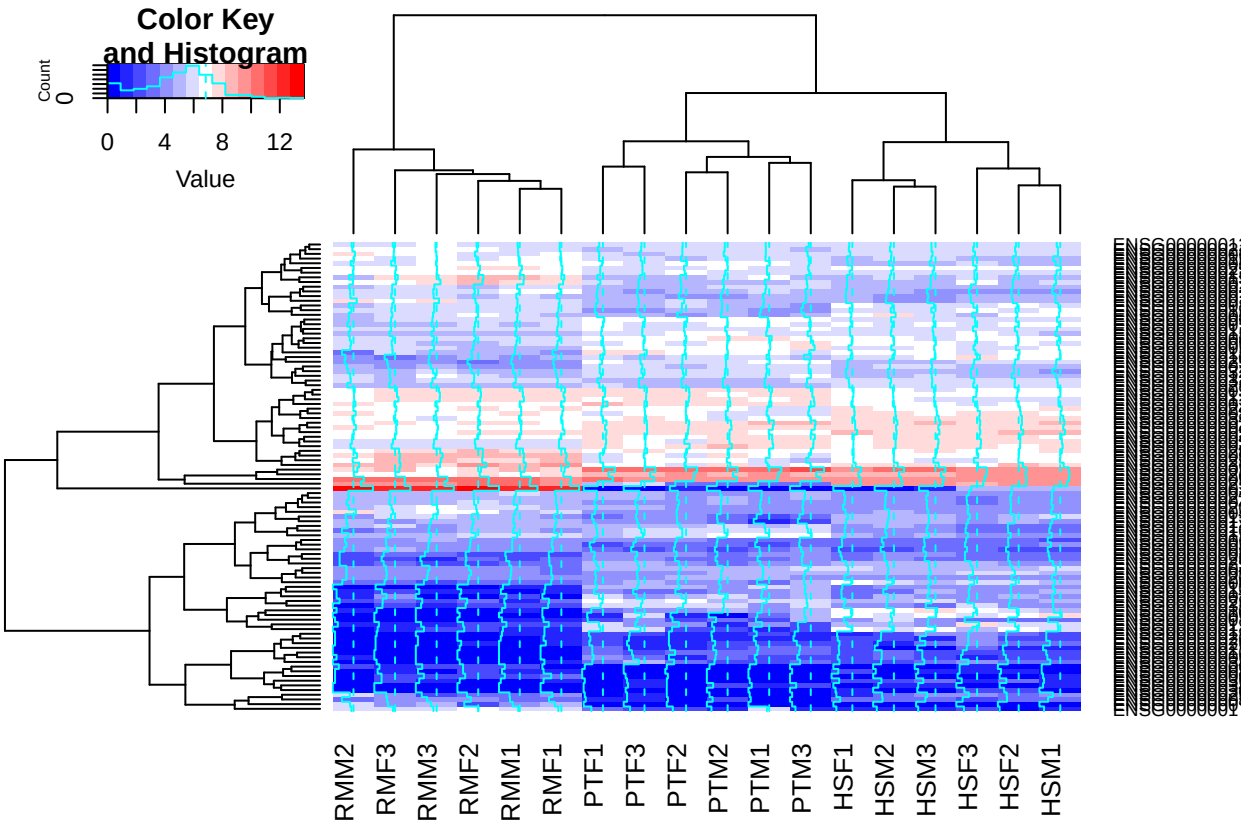
An even better option is the `heatmap.2` function of the `gplots` package:

```
library("gplots")
```

```
##
## Attaching package: 'gplots'
```

```
## The following object is masked from 'package:stats':
##
##     lowess
```

```
heatmap.2( head(nmat6[spDEGenes,], 100), col=bluered )
```



Centering and scaling using `scale`

This looks nicer. But the problem is, it groups genes based on their average expression levels, rather than variation among species.

We usually do not study information on absolute expression levels; we do not wish to compare genes w.r.t. their expression levels. Rather, we are interested in variation in expression among groups. Therefore we wish to cluster genes with *different means* but the *same expression patterns*. For that reason, when clustering genes, it is reasonable to **scale** all genes to `mean=0`, and `sd=1`, so that fixed expression level differences between two genes are practically ignored:

```
# let's choose the first gene
y = nmat6[1,]
par(mfrow=c(1,2))
boxplot(y ~ species2, col=2:4, ylab="Expression")
# scale
sy = (y-mean(y))/sd(y)
sy
```

```
##        HSM1        PTF1        RMM1        HSF1        PTM1        RMF1
## -2.22147179  0.05641317 -0.34675259 -0.22832184  0.25507656  0.23186622
##        RMF2        HSM2        PTF2        RMM2        HSF2        PTM2
##  0.40414715 -0.11945816  0.30131103  2.40216529 -0.38175785  1.22697103
##        RMM3        RMF3        HSM3        PTF3        PTM3        HSF3
##  0.88930293 -0.51315211 -0.34747338 -0.16163155  0.23356821 -1.68080232
```

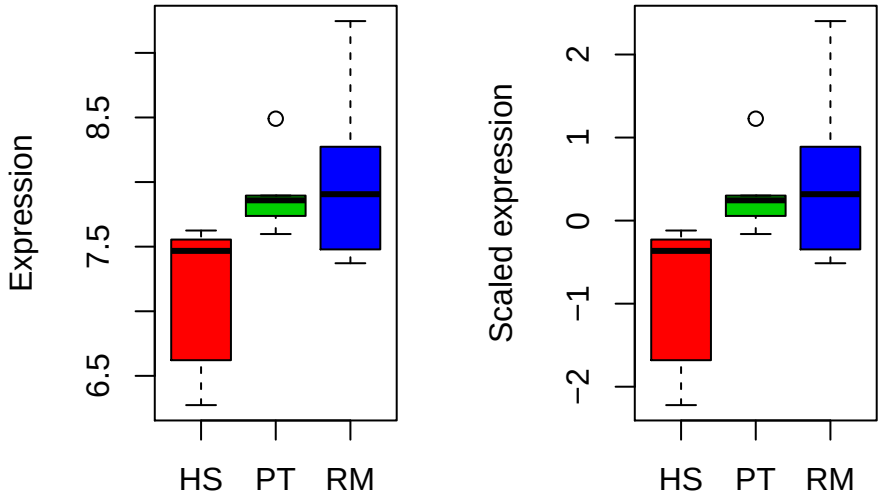
```r
mean(sy)
```

```
## [1] -1.349711e-16
```

```r
sd(sy)
```

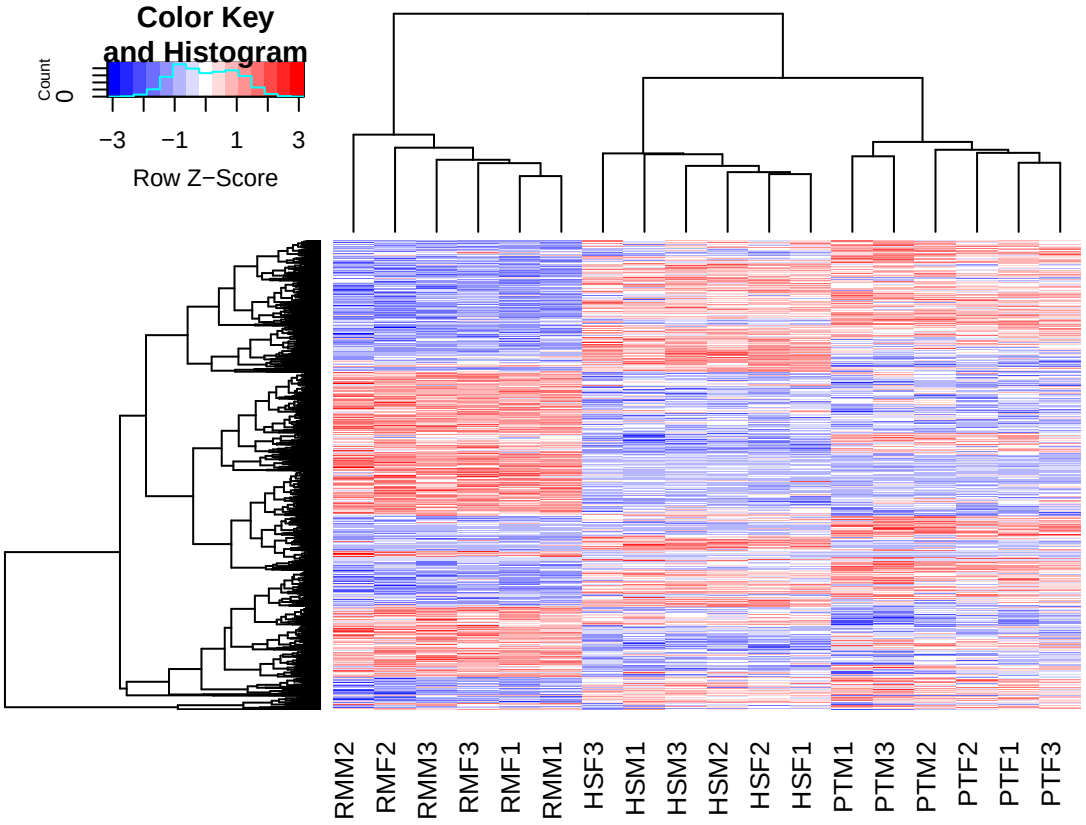
```
## [1] 1
```

```r
# the same pattern, but check the y-axis
boxplot(sy ~ species2, col=2:4, ylab="Scaled expression")
```



Now we plot a heatmap on the row-scaled matrix. `heatmap.2` can scale the input matrix if required:

```r
heatmap.2( nmat6[spDEGenes,], col=bluered, labRow = "", trace = "none", scale = "row")
```



What do the different blocks represent? How many different expression patterns would you predict there are?

K-means clustering

Here we can also use the k-means algorithm, although hiearchical clustering (`hclust`) and other functions are also available.

With k-means we need to choose the number of groups `k` in advance. There are methods to decide which is best, but they do not always work perfectly. A common approach is trying multiple `k` and confirming that the **downstream results** do not depend on the choice of `k`. Here we'll choose `k=9`, not for any particular reason.

Also note that k-means is a **heuristic algorithm**. You can get different results on different runs. And it might sometimes not converge.

For k-means, we need to scale the rows of the matrix ourselves, for which we can use the `scale` function. This centers a matrix *columns* to `mean=0` and scales to `sd=1`.

A last point: it might be preferred to run clustering only using DE genes, to focus the rest of the study on differential expression patterns.

```
# center and scale rows
snmat6 = t( scale( t( nmat6 ) ) )
head(snmat6)
```

```
##                      HSM1        PTF1        RMM1       HSF1       PTM1
## ENSG00000000003 -2.2214718  0.05641317 -0.34675259 -0.2283218  0.2550766
## ENSG00000000005 -0.8488278  0.23674941  1.34358138 -0.8488278 -0.8488278
## ENSG00000000419 -1.1793918  0.63280031 -1.71877979  1.0407810 -0.4173648
## ENSG00000000457  0.4626583 -0.27857166 -0.36691681 -0.3087339  2.0616890
```

```
## ENSG00000000460  1.8419800 -0.37056538 -0.40248243 -0.5013011  0.5123615
## ENSG00000000938  0.5315166 -0.01028352 -0.03381372 -0.6297712  2.0633735
##                       RMF1       RMF2       HSM2        PTF2       RMM2
## ENSG00000000003  0.23186622  0.4041471 -0.1194582  0.30131103  2.4021653
## ENSG00000000005 -0.84882779  0.8133052  0.2367494  2.49574789 -0.8488278
## ENSG00000000419 -0.42681915  0.7446611  0.3525870 -0.01523095  1.6666685
## ENSG00000000457 -0.84697216 -0.8504087 -1.1389593  1.68273133  0.3542160
## ENSG00000000460 -1.35535994 -0.7988927 -0.4844542 -0.12883054 -1.0805561
## ENSG00000000938 -0.08129359 -1.2876148 -0.9193627  0.85043367 -0.5570455
##                      HSF2       PTM2        RMM3       RMF3        HSM3
## ENSG00000000003 -0.3817578  1.2269710  0.88930293 -0.5131521 -0.34747338
## ENSG00000000005 -0.8488278  0.3814930 -0.84882779  0.8646538 -0.84882779
## ENSG00000000419  0.1233517 -0.5479359  0.74034639 -1.8895427 -0.32066235
## ENSG00000000457 -1.0847221  1.6869316  0.46265827 -0.8050920 -0.45483458
## ENSG00000000460  1.9029746 -0.4024824 -1.03940783 -0.1288305  1.53506266
## ENSG00000000938  0.1623586 -1.4014023 -0.06351781 -1.0254977 -0.08841513
##                       PTF3       PTM3       HSF3
## ENSG00000000003 -0.161631545  0.2335682 -1.6808023
## ENSG00000000005  0.556562085  0.7106079 -0.8488278
## ENSG00000000419  0.004944411  1.6269519 -0.4173648
## ENSG00000000457 -0.582570017  0.7457852 -0.7388884
## ENSG00000000460 -0.641754137  0.9467809  0.5957577
## ENSG00000000938  0.234508555 -0.0690900  2.3249172
```

```
rowMeans(head(snmat6))
```

```
## ENSG00000000003 ENSG00000000005 ENSG00000000419 ENSG00000000457
##   -1.349229e-16    2.158767e-17   -1.169011e-16    3.700743e-16
## ENSG00000000460 ENSG00000000938
##    1.541976e-17    4.969983e-16
```

```
apply(head(snmat6), 1, sd)
```

```
## ENSG00000000003 ENSG00000000005 ENSG00000000419 ENSG00000000457
##               1               1               1               1
## ENSG00000000460 ENSG00000000938
##               1               1
```

Choose 9 clusters:

```
km10 = kmeans( snmat6[spDEGenes,] , centers = 10)
# the cluster identities
head(km10$cluster)
```

```
## ENSG00000001461 ENSG00000001561 ENSG00000001617 ENSG00000002330
##               3               3               6               4
## ENSG00000002549 ENSG00000002586
##               7              10
```

```
# number of genes
km10$size
```

```
##  [1] 273 259 317 312 215 553 238 328 130 472
```

```
# same thing
table( km10$cluster )
```

```
##
##   1   2   3   4   5   6   7   8   9  10
```

```
## 273 259 317 312 215 553 238 328 130 472
```

The cluster means are stored in the list element named `centers`:

```
head( km10$centers )
```

```
##          HSM1       PTF1       RMM1        HSF1       PTM1       RMF1
## 1 -0.06962736 -1.0674227  1.0807163  0.01941314 -1.2061545  1.0658732
## 2  1.18467229 -0.5074743 -0.6273273  1.11738027 -0.6636594 -0.6237204
## 3 -0.97178664 -0.2498558  1.0385659 -0.79927375 -0.2426888  0.9483251
## 4 -0.13473884  0.9798602 -1.0038262 -0.02722453  1.1533209 -1.0081906
## 5 -1.25210336  0.4176547  0.6124235 -1.02596647  0.4994003  0.6085101
## 6 -0.62323014 -0.6552190  1.2567578 -0.54402001 -0.6994957  1.2199244
##         RMF2        HSM2       PTF2       RMM2         HSF2       PTM2
## 1  1.1532256 -0.07861778 -0.8737128  1.0215342  0.000181266 -0.7613261
## 2 -0.5953302  1.23137186 -0.6063258 -0.5134425  1.171713448 -0.5749643
## 3  0.8362792 -0.76203853 -0.2115913  1.3627171 -0.938642716 -0.2310836
## 4 -1.0577219  0.01090392  0.9333012 -1.0704654 -0.096891089  1.0003927
## 5  0.6490631 -1.00462919  0.5427050  0.4094718 -1.125203437  0.5776676
## 6  1.3321768 -0.58459289 -0.5266223  1.0221627 -0.610022329 -0.5304834
##         RMM3       RMF3        HSM3       PTF3        PTM3        HSF3
## 1  0.9828208  0.9679316 -0.13482721 -0.6758487 -1.14138434 -0.28277467
## 2 -0.6526996 -0.5575730  1.22393487 -0.4247684 -0.56148851  0.97970091
## 3  1.2348708  1.0803408 -0.85579553 -0.2510644 -0.01864483 -0.96863303
## 4 -0.8706014 -0.8726803  0.04235421  0.8630834  1.24146761 -0.08234383
## 5  0.7181056  0.6931432 -1.17417663  0.4110163  0.71235303 -1.26943523
## 6  1.1988686  1.1907155 -0.62057548 -0.5472046 -0.62089912 -0.65824076
```

```
# means of genes in cl1
km10$centers[1,]
```

```
##         HSM1         PTF1         RMM1         HSF1         PTM1
## -0.069627360 -1.067422676  1.080716303  0.019413141 -1.206154469
##         RMF1         RMF2         HSM2         PTF2         RMM2
##  1.065873233  1.153225626 -0.078617776 -0.873712754  1.021534169
##         HSF2         PTM2         RMM3         RMF3         HSM3
##  0.000181266 -0.761326138  0.982820797  0.967931562 -0.134827214
##         PTF3         PTM3         HSF3
## -0.675848702 -1.141384341 -0.282774667
```

```
# same as:
genes = names( which( km10$cluster == 1 ) )
colMeans( snmat6[ genes, ] )
```

```
##         HSM1         PTF1         RMM1         HSF1         PTM1
## -0.069627360 -1.067422676  1.080716303  0.019413141 -1.206154469
##         RMF1         RMF2         HSM2         PTF2         RMM2
##  1.065873233  1.153225626 -0.078617776 -0.873712754  1.021534169
##         HSF2         PTM2         RMM3         RMF3         HSM3
##  0.000181266 -0.761326138  0.982820797  0.967931562 -0.134827214
##         PTF3         PTM3         HSF3
## -0.675848702 -1.141384341 -0.282774667
```

Note that we may get different results each time we run:

```
set.seed(1)
km10 = kmeans( snmat6[spDEGenes,] , centers = 10)
km10$size
```

```
##  [1] 400 261 260 129 298 330 557 166 312 384
```

```
km10 = kmeans( snmat6[spDEGenes,] , centers = 10)
km10$size
```

```
##  [1] 250 341 306 312 323 490 244 422 241 168
```

```
# let's set the seed to 1 so that everyone gets the same result
set.seed(1)
km10 = kmeans( snmat6[spDEGenes,] , centers = 10)
sort( km10$size )
```

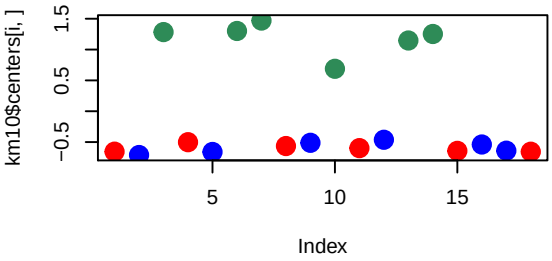
```
##  [1] 129 166 260 261 298 312 330 384 400 557
```

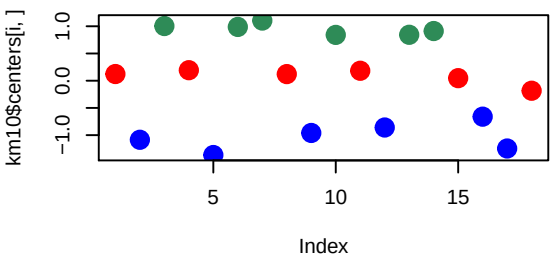
Now plot the cluster means:

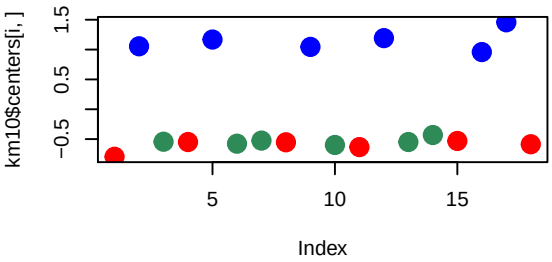
```
par( mfrow=c(5,2) )
# choose colors for each species using levels of species2
as.numeric(species2)
```

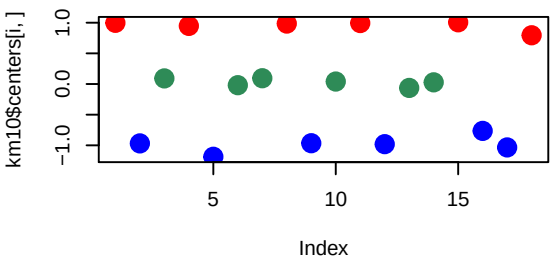
```
##  [1] 1 2 3 1 2 3 3 1 2 3 1 2 3 3 1 2 2 1
```

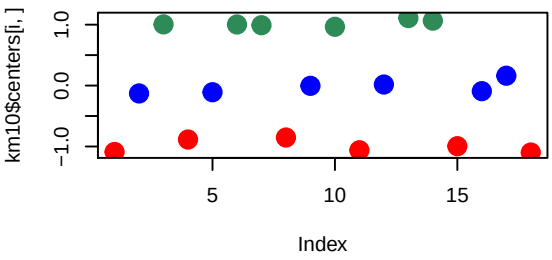
```
colx = c("red", "blue", "seagreen")[as.numeric(species2)]
# increase point size with cex
for( i in 1:10) {
  plot( km10$centers[i,], col = colx, pch = 19, cex=2)
}
```

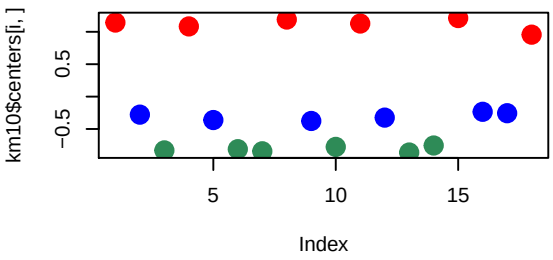

km10$centers[i,]
−0.5
0.5
1.5
5
10
15
Index

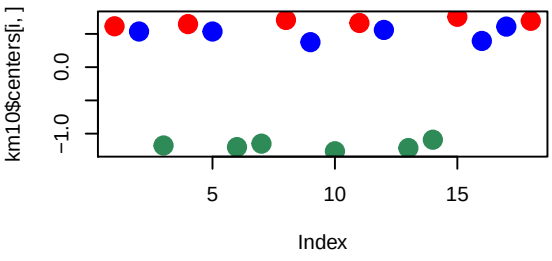

km10$centers[i,]
−1.0
0.0
1.0
5
10
15
Index

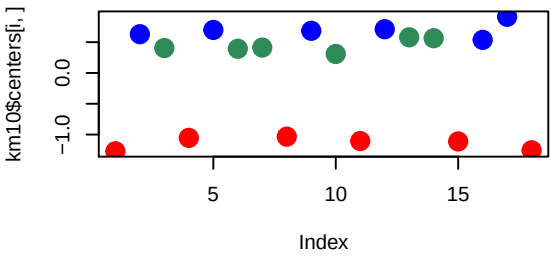

km10$centers[i,]
−0.5
0.5
1.5
5
10
15
Index

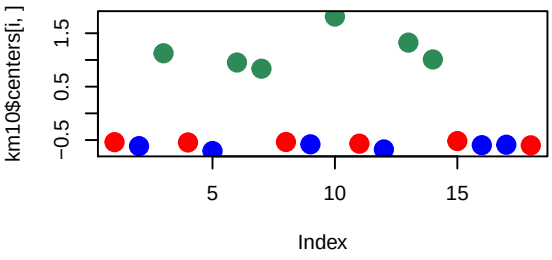

km10$centers[i,]
−1.0
0.0
1.0
5
10
15
Index

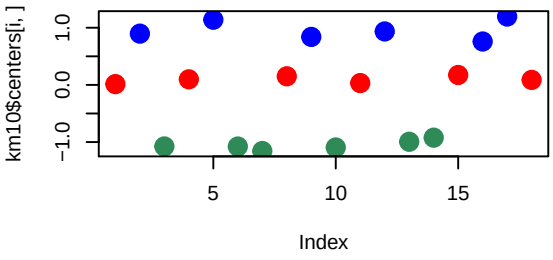

km10$centers[i,]
−1.0
0.0
1.0
5
10
15
Index


km10$centers[i,]
−0.5
0.5
5
10
15
Index


km10$centers[i,]
−1.0
0.0
5
10
15
Index


km10$centers[i,]
−1.0
0.0
5
10
15
Index


km10$centers[i,]
−0.5
0.5
1.5
5
10
15
Index


km10$centers[i,]
−1.0
0.0
1.0
5
10
15
Index

But this looks ugly and confusing. Let's try to reorder the samples according to species:

```
which(species2 == 'HS')
```

```
## [1]  1  4  8 11 15 18
```

```
reorder = c(which(species2 == 'HS'), which(species2 == 'PT'), which(species2 == 'RM') )
reorder
```

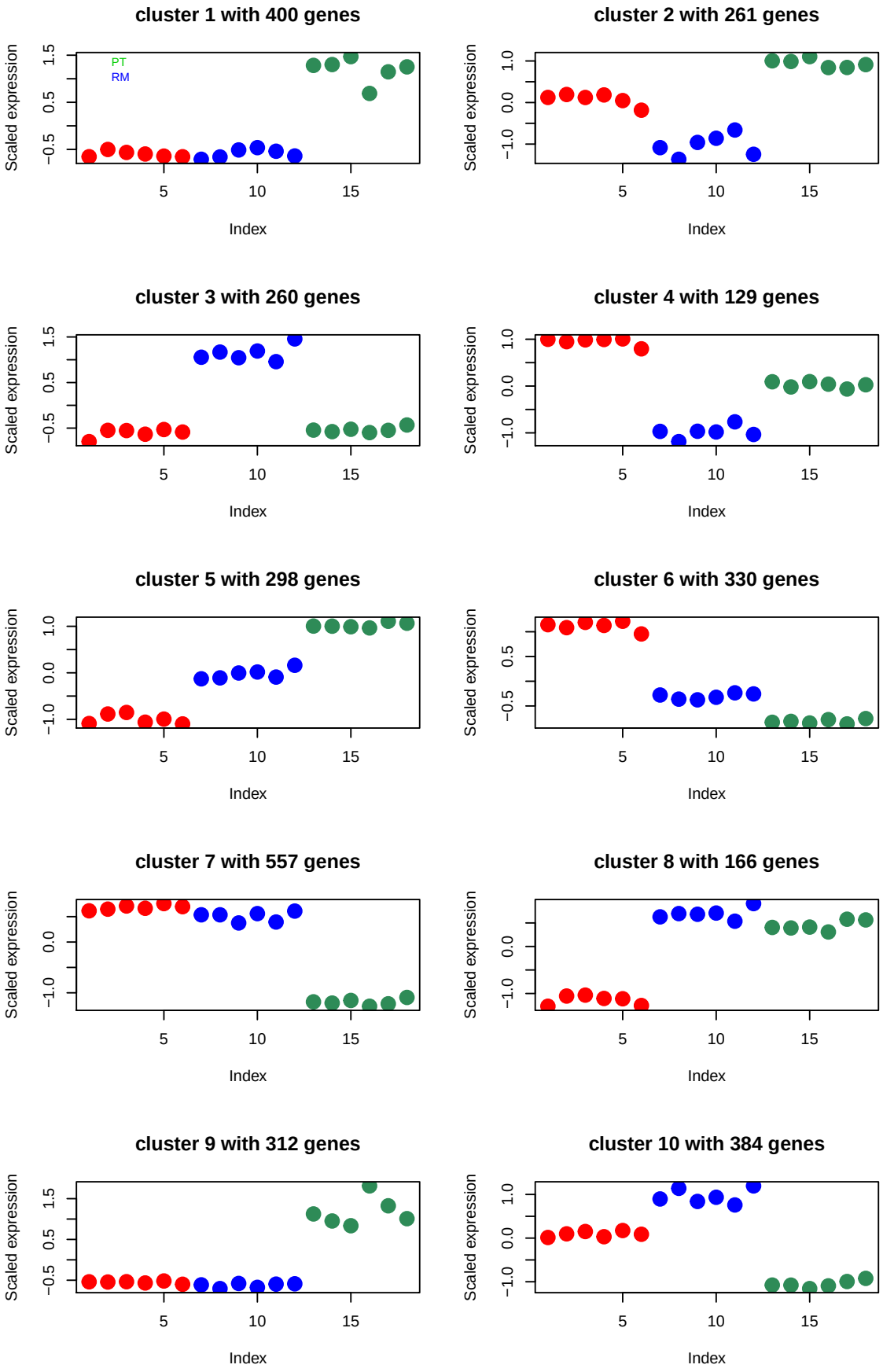
```
##  [1]  1  4  8 11 15 18  2  5  9 12 16 17  3  6  7 10 13 14
```

```
km10$centers[1, reorder]
```

```
##       HSM1       HSF1       HSM2       HSF2       HSM3       HSF3
## -0.6553865 -0.5019261 -0.5643468 -0.5981599 -0.6424803 -0.6564051
##       PTF1       PTM1       PTF2       PTM2       PTF3       PTM3
## -0.7100454 -0.6602986 -0.5117302 -0.4625724 -0.5393716 -0.6398186
##       RMM1       RMF1       RMF2       RMM2       RMM3       RMF3
##  1.2832220  1.3013284  1.4703374  0.6881776  1.1469426  1.2525336
```

Now plot:

```
par( mfrow=c(5,2) )
for( i in 1:10)  {
  colx2 = colx[reorder]
  maintext = paste( 'cluster', i, 'with', km10$size[i], 'genes')
  plot( km10$centers[i, reorder], col=colx2, pch=19, cex=2, main=maintext, ylab="Scaled expression")
  if (i == 1) legend(1, 2, text.col = 2:4, legend = levels(species2), bty="n", cex=0.8)
}
```


cluster 1 with 400 genes
cluster 2 with 261 genes
cluster 3 with 260 genes
cluster 4 with 129 genes
cluster 5 with 298 genes
cluster 6 with 330 genes
cluster 7 with 557 genes
cluster 8 with 166 genes
cluster 9 with 312 genes
cluster 10 with 384 genes
Scaled expression
Index
PT
RM

This looks better (although we have no idea about the variation within clusters).

Which clusters appear to have **human-specific expression**? Any guesses? What could they represent?