Module 6 exam

AnaïsRey

14 September, 2020

Consigne Vous fournirez un rapport au format pdf généré à partir d'un Rmd (envoyez-nous les 2 fichiers, Rmd et pdf, avec comme nom de fichier "NOM-PRENOM_evaluation-m6-2020" + .Rmd ou .pdf), avec une page d'introduction et deux figures maximum par analyse. Vos travaux doivent être reproductibles, pensez à décrire et justifier les différentes étapes, seuils, extractions ... Bon courage ! Nous sommes disponibles sur Slack en cas de besoin.

https://docs.google.com/document/d/1bjA3WJBF_-rqIV6CUzdRtfrLRSqhVRuLzc8jU1T7aUw/edit#

Load packages

Import the data from the FA model experiment

```
## Set path to files

path <- "/shared/data/projects/dubii2020/data/hbc-MouseKidneyFibrOmics-a39e55a/tables/"

## Get the raw count table from the transcriptomic datataset
trans.count.raw <- read.csv(file=paste(path, "fa/results/counts/raw_counts.csv.gz",
                                       sep=""),
                            header=TRUE)

## Get the Transcripts Per Million normalized count table from the transcriptomic datataset
trans.count.tpm <- read.csv(file=paste(path, "fa/results/counts/tpm.csv.gz", sep=""),
                            header=TRUE)

# Get the raw count table from the proteomic dataset
proteo.count.raw <- read.csv(file=paste(path, "pfa/results/counts/fa_model_counts.csv", sep=""),
                            header=TRUE)
```

Introduction

Summary of the datasets

We are working on two omics datasets: one from transcriptomic experiment and one from proteomic experiment. Both are coming from the same experiment where kidneys of mouses with reversible chemicalfolic acid (FA) induced nephropathy were taken at different times of the treatment (day 0 and several days after the injection) and RNA and protein isolations were performed.

1) Look at the dimension and the contents of each table:

```
dim(trans.count.raw)
```

```
## [1] 46679    19
```

```
head(trans.count.raw)[1:7]
```

```
##              rowname     day1_1      day1_2      day1_3    day14_1    day14_2
## 1 ENSMUSG00000000001 2278.80022 1786.498848 2368.618959 627.758017 559.156031
## 2 ENSMUSG00000000003    0.00000    0.000000    0.000000   0.000000   0.000000
## 3 ENSMUSG00000000028   36.27547   22.147861   39.484949  14.470759  10.167813
```

```
## 4 ENSMUSG00000000031   13.18853    7.151932    1.115304   0.867429   0.000000
## 5 ENSMUSG00000000037    0.00000   27.903214    6.897842   5.692254   1.901719
## 6 ENSMUSG00000000049   30.86001    4.861367   51.466810  26.152649   1.968290
##       day14_3
## 1 611.4338605
## 2   0.0000000
## 3  31.6910193
## 4   0.0000000
## 5   0.6549762
## 6  55.8319868
```

```
dim(proteo.count.raw)
```

```
## [1] 8044   11
```

```
head(proteo.count.raw)[1:7]
```

```
##                    id  normal_1  normal_2    day1_1    day1_2    day2_1   day2_2
## 1 ENSMUSG00000037686  531.2680  651.7200  335.5910  334.8460  197.1740  307.194
## 2 ENSMUSG00000027831  221.6020  266.3590  175.4090  159.4190  234.8080  256.927
## 3 ENSMUSG00000039201   26.0723   29.1331   57.7329   45.8475   81.6009   88.870
## 4 ENSMUSG00000031095 4363.0500 4784.0800 4064.4800 3917.2900 4599.0300 5957.030
## 5 ENSMUSG00000034931  879.2790 1065.3900  914.2870  928.2760 1000.1000 1264.270
## 6 ENSMUSG00000038208   68.2225   89.8871   57.9041   76.3510   84.8474  105.245
```

There is 46679 rows that correspond to genes and 18 columns that correspond to samples in the count table coming from the transcriptomic dataset.

There is 8044 rows that correspond to proteins and 10 columns that correspond to samples in the count table coming from the proteomic dataset.

2) We build a metadata table that will be needed for the analysis of the report

```
get_metaD <- function (x,y){

  ## We build the metadata table by adding the type of dataset,
  ## the names of sample, the condition of sample and sample number

  metadata <- data.frame(
  dataType = y,
  sampleName = colnames(x[-1]))

  metadata <- metadata %>%
  separate(sampleName, c("condition", "sampleNumber"), remove=F)

  ## We transform into factor "condition" and "sampleNumber"
  metadata$condition <-
    factor(metadata$condition)
  metadata$sampleNumber <-
    factor(metadata$sampleNumber)

  ## We specify a Color per condition and we add the color column to the metadata table
  colPerCondition <- c(normal = "#BBFFBB",
                       day1 = "#FFFFDD",
                       day2 = "#FFDD88",
                       day3 = "#FFBB44",
                       day7 = "#FF8800",
                       day14 = "#FF4400")

  metadata$color <- colPerCondition[metadata$condition]

return(metadata)
}

trans.metadata <- get_metaD(x=trans.count.raw, y="transcriptome")
```

```
proteo.metadata <- get_metaD(x=proteo.count.raw, y="proteome")
```

```
kable(trans.metadata, caption="metadata from the transcriptomic dataset")
```

Table 1: metadata from the transcriptomic dataset

dataType	sampleName	condition	sampleNumber	color
transcriptome	day1_1	day1	1	#BBFFBB
transcriptome	day1_2	day1	2	#BBFFBB
transcriptome	day1_3	day1	3	#BBFFBB
transcriptome	day14_1	day14	1	#FFFFDD
transcriptome	day14_2	day14	2	#FFFFDD
transcriptome	day14_3	day14	3	#FFFFDD
transcriptome	day2_1	day2	1	#FFDD88
transcriptome	day2_2	day2	2	#FFDD88
transcriptome	day2_3	day2	3	#FFDD88
transcriptome	day3_1	day3	1	#FFBB44
transcriptome	day3_2	day3	2	#FFBB44
transcriptome	day3_3	day3	3	#FFBB44
transcriptome	day7_1	day7	1	#FF8800
transcriptome	day7_2	day7	2	#FF8800
transcriptome	day7_3	day7	3	#FF8800
transcriptome	normal_1	normal	1	#FF4400
transcriptome	normal_2	normal	2	#FF4400
transcriptome	normal_3	normal	3	#FF4400

```
kable(proteo.metadata, caption="metadata from the proteomic dataset")
```

Table 2: metadata from the proteomic dataset

dataType	sampleName	condition	sampleNumber	color
proteome	normal_1	normal	1	#FF8800
proteome	normal_2	normal	2	#FF8800
proteome	day1_1	day1	1	#BBFFBB
proteome	day1_2	day1	2	#BBFFBB
proteome	day2_1	day2	1	#FFDD88
proteome	day2_2	day2	2	#FFDD88
proteome	day7_1	day7	1	#FFBB44
proteome	day7_2	day7	2	#FFBB44
proteome	day14_1	day14	1	#FFFFDD
proteome	day14_2	day14	2	#FFFFDD

There are some differences between the two omic datasets:

- For each sample, 3 replicates were performed for the transcriptomic dataset whereas only 2 were performed for the proteomic dataset.
- Day 3 was not sampled and/or prepared for the proteomic dataset

3) Before starting we put the gene ID as rownames

```
# For transcriptomic count tables,
# one gene = one row so we can just rename
# the rownames by taking the column "rowname"
trans.count.raw <- trans.count.raw %>% column_to_rownames(var="rowname")
trans.count.tpm <- trans.count.tpm %>% column_to_rownames(var="rowname")

# For proteomic data, there are several rows
# with the same protein name,
# so to "trick" and use the name of the protein as a rowname,
```

```
# I added to each protein the number of the row

proteo.count.raw$nb_row <- 1:nrow(proteo.count.raw)

proteo.count.raw <- proteo.count.raw %>%
  unite("rowname", id, nb_row) %>%
    column_to_rownames(var="rowname")
```

4) We round the raw data as advised in the slack as it seems that the data were already normalized in some way

```
trans.count.raw.arr <- round(trans.count.raw, 0)
trans.count.raw.arr <- as.matrix(trans.count.raw.arr)

proteo.count.raw.arr <- round(proteo.count.raw, 0)
proteo.count.raw.arr <- as.matrix(proteo.count.raw.arr)
```

Analyse d'expression différentielle

Transcriptomic data

Enoncé: Analyse d'expression différentielle pour les données de protéomique et transcriptomique => identifier les gènes/protéines significativement différentiellement exprimés dans le modèle FA en comparant Day 7 à Day 0.

Genes filtering

We keep only a set of expressed genes by using those with a transcripts per million normalized counts >=1 in at least one sample

```
## Expressed genes
nbexpr <- apply(trans.count.tpm, 1, function(x){length(which(x>=1))})
isexpr <- which(nbexpr>=1)
trans.count.raw.filtered <- trans.count.raw.arr[isexpr,]

dim(trans.count.raw.arr)
```

```
## [1] 46679    18
```

```
dim(trans.count.raw.filtered)
```

```
## [1] 22091    18
```

```
## We remove genes with less than 10 reads in total as we
# want to filter out the genes with very low counts in all conditions
undetectedFeatures <- apply(trans.count.raw.filtered, MARGIN = 1, FUN = sum) < 10

trans.count.raw.filtered <- trans.count.raw.filtered[!undetectedFeatures, ]

dim(trans.count.raw.filtered)
```

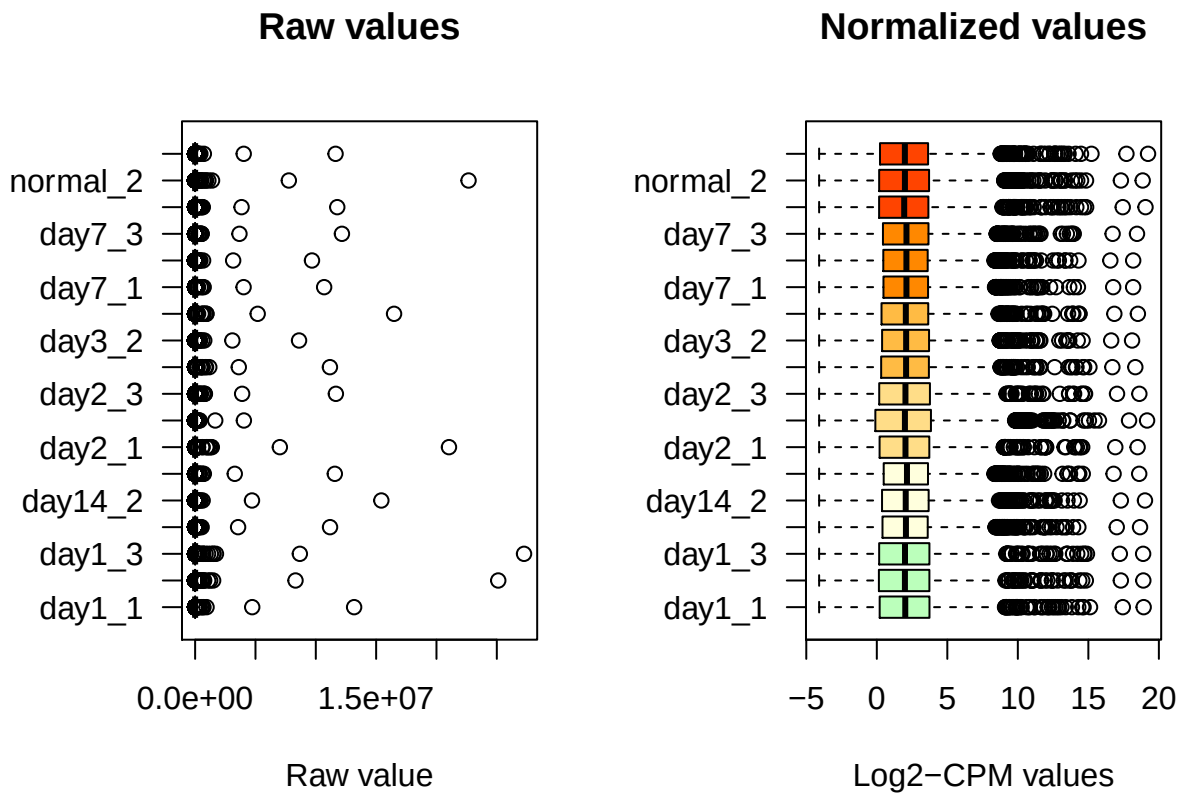
```
## [1] 21028    18
```

Library size normalization with edgeR

```
## We create a DGEList object from the count table
# and give the group indicator for each column
# (here the condition column)

trans.dge <- DGEList(counts=trans.count.raw.filtered,
                     group=trans.metadata$condition)

## We remove genes that are lowly expressed
```

```
keep.exprs <- filterByExpr(trans.dge)
trans.dge.nolow <- trans.dge[keep.exprs,, keep.lib.sizes=FALSE]

## We estimate the factor normalization based on the TMM method
trans.dge.nolow <- calcNormFactors(trans.dge.nolow , method="TMM")

# Important : using calcNormFactors does not change the counts,
# it just updates the column norm.factors
trans.count.norm <- cpm(trans.dge.nolow, log=TRUE)
```

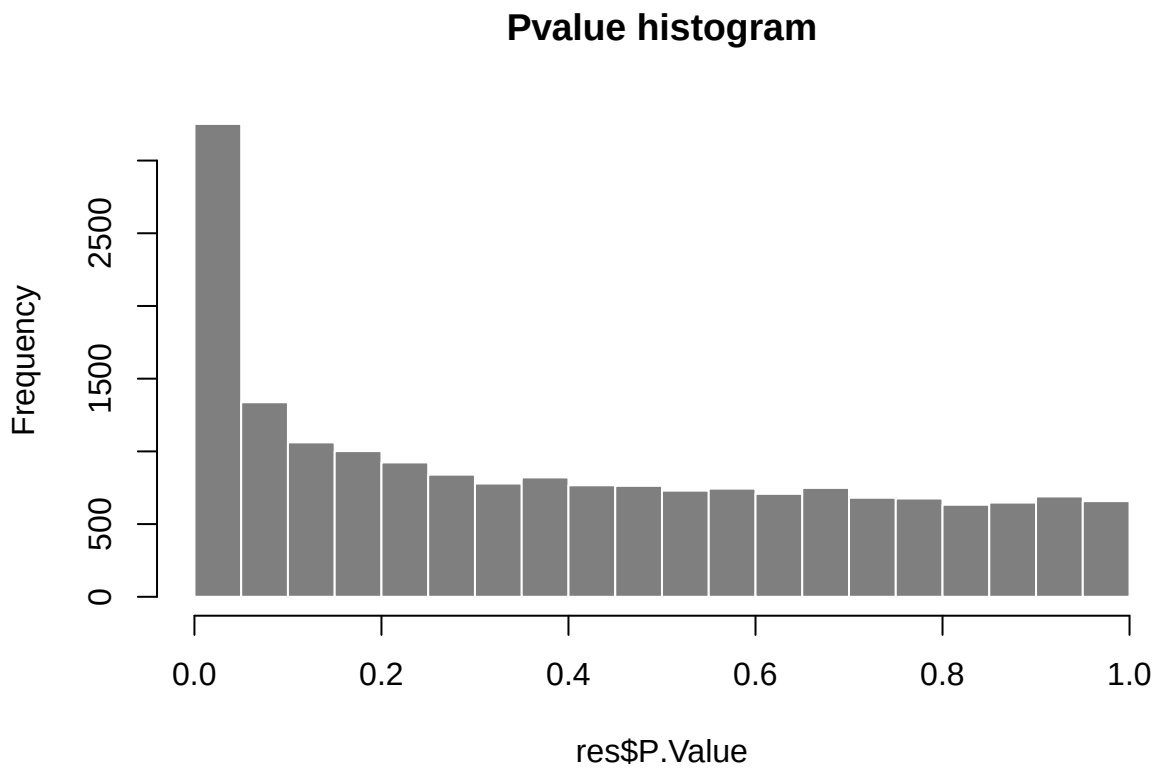
```
# We look at the impact of normalization on data
par(mar = c(4, 6, 5, 1))
par(mfrow = c(1,2))
boxplot(trans.count.raw.arr, col=trans.metadata$color,
        horizontal = TRUE,
        las = 1,
        main = "Raw values",
        xlab = "Raw value")

boxplot(trans.count.norm, col=trans.metadata$color,
        horizontal = TRUE,
        las = 1,
        main = "Normalized values",
        xlab = "Log2-CPM values")
```



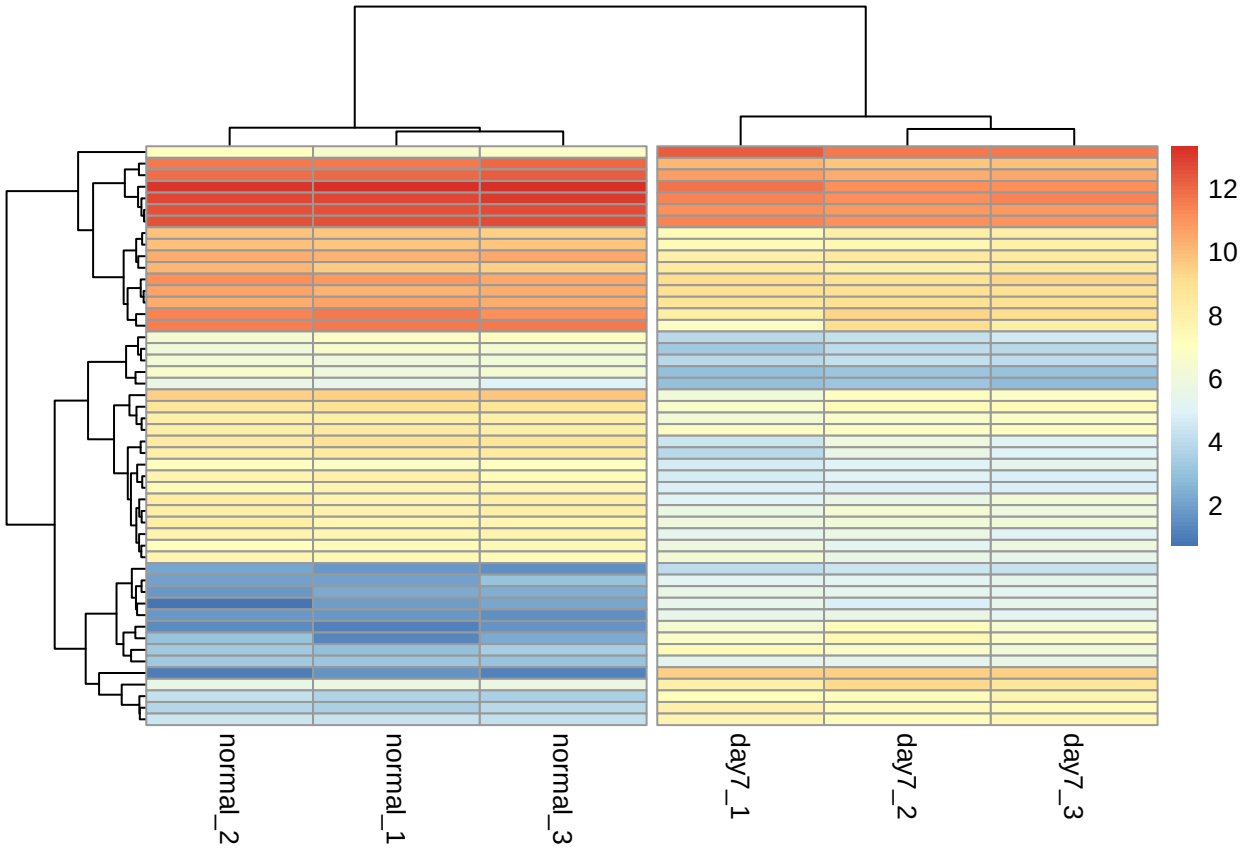
Differential analysis with limma

```
## Voom transformation of normalized data to apply the limma statistical framework
design <- model.matrix(~ 0 + condition, data=trans.metadata)
v <- voom(trans.dge.nolow, design, plot=FALSE)

# Differential analysis based on 'limma'
```

```r
fit <- lmFit(v, design)

## Compare Condition normal with the one ater 7 days
contrast <-makeContrasts(conditionnormal - conditionday7, levels=design)

## Run test
fit2 <- contrasts.fit(fit, contrast)
fit2 <- eBayes(fit2)

## Extract the results
res <- topTable(fit2, number=1e6, adjust.method="BH")

## Pvalue distribution
hist(res$P.Value, main="Pvalue histogram", col="grey50", border="white")
```




```r
## Extract list of DEG
idx.sign <- which(res$adj.P.Val < 0.05 &
                  abs(res$logFC) > 1)
deg <- rownames(res[idx.sign,])

# Heatmap of the 50 most differentially expressed genes
idx.sub <- which(trans.metadata$condition=="normal" | trans.metadata$condition=="day7")
data.sub <- trans.count.norm[deg[1:50],idx.sub]
pheatmap(data.sub,
        cutree_cols = 2,
        show_rownames=FALSE)
```



Proteomic data

Proteins filtering

```
## We remove proteins with less than 10 counts in total as we
# want to filter out the proteins with very low counts in all conditions
undetectedFeatures <- apply(proteo.count.raw.arr, MARGIN = 1, FUN = sum) < 10

proteo.count.raw.filtered <- proteo.count.raw.arr[!undetectedFeatures, ]

dim(proteo.count.raw.arr)
```

```
## [1] 8044   10
```

```
dim(proteo.count.raw.filtered)
```

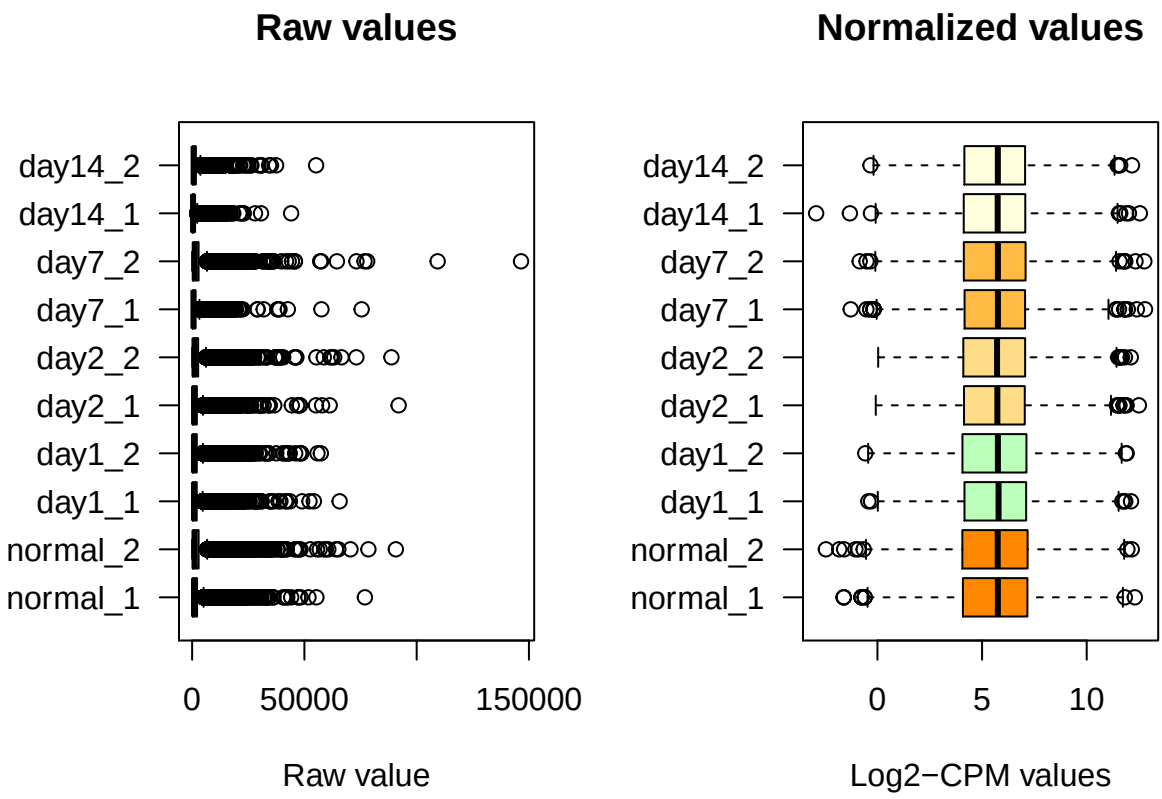
```
## [1] 8044   10
```

Library size normalization with edgeR

```
## We create a DGEList object from the count table and
# give the group indicator for each column
# (here the condition column)

proteo.dge <- DGEList(counts=proteo.count.raw.filtered,
                      group=proteo.metadata$condition)

## We remove genes that are lowly expressed
keep.exprs <- filterByExpr(proteo.dge)
proteo.dge.nolow <- proteo.dge[keep.exprs,, keep.lib.sizes=FALSE]
```

```
## We estimate the factor normalization based on the TMM method
proteo.dge.nolow <- calcNormFactors(proteo.dge.nolow , method="TMM")

# Important : using calcNormFactors does not change the counts,
# it just updates the column norm.factors
proteo.count.norm <- cpm(proteo.dge.nolow, log=TRUE)
```

```
# We look at the impact of normalization on data
par(mar = c(4, 6, 5, 1))
par(mfrow = c(1,2))
boxplot(proteo.count.raw.arr, col=proteo.metadata$color,
        horizontal = TRUE,
        las = 1,
        main = "Raw values",
        xlab = "Raw value")

boxplot(proteo.count.norm, col=proteo.metadata$color,
        horizontal = TRUE,
        las = 1,
        main = "Normalized values",
        xlab = "Log2-CPM values")
```



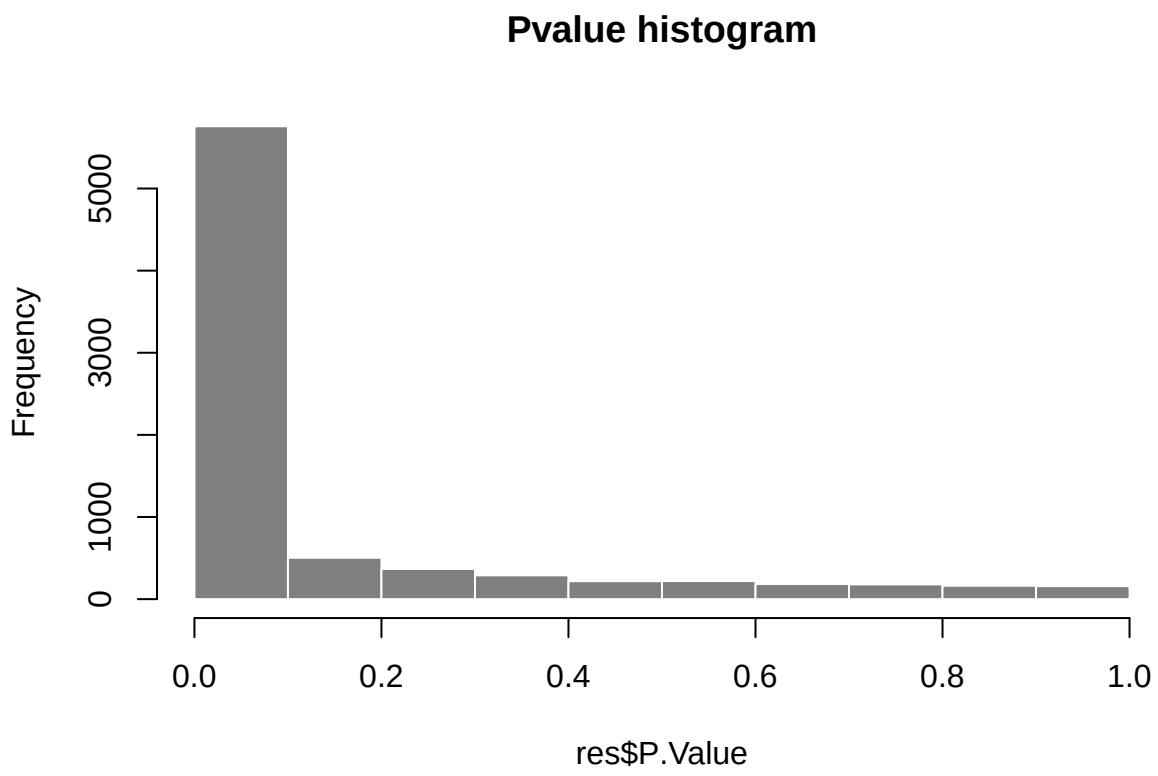

```
dev.off()
```

```
## null device
##           1
```

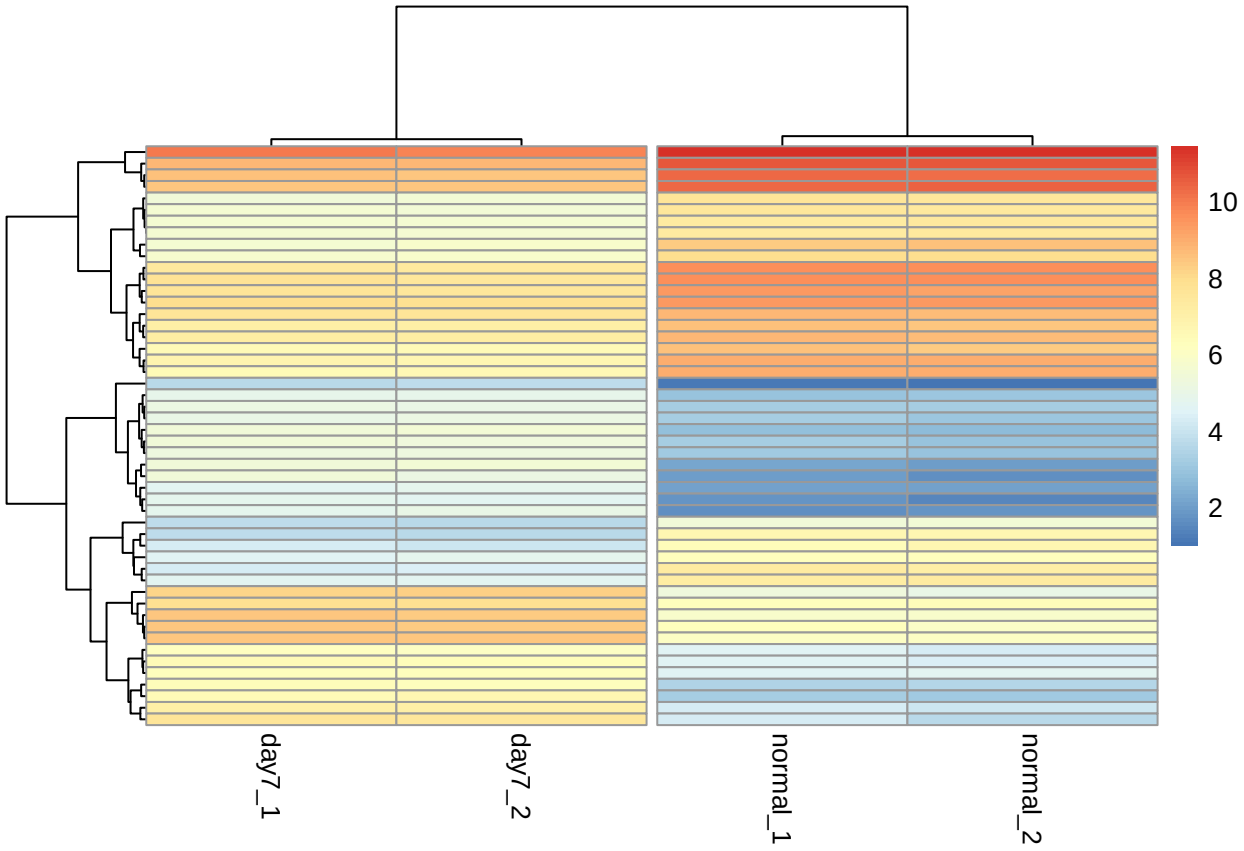
Differential analysis with limma

```
## Voom transformation of normalized data to apply the limma statistical framework
design <- model.matrix(~ 0 + condition, data=proteo.metadata)
v <- voom(proteo.dge.nolow, design, plot=FALSE)
```

```r
# Differential analysis based on 'limma'
fit <- lmFit(v, design)

## Compare Condition normal with the one ater 7 days
contrast <-makeContrasts(conditionnormal - conditionday7, levels=design)

## Run test
fit2 <- contrasts.fit(fit, contrast)
fit2 <- eBayes(fit2)

## Extract the results
res <- topTable(fit2, number=1e6, adjust.method="BH")

## Pvalue distribution
hist(res$P.Value, main="Pvalue histogram", col="grey50", border="white")
```




```r
## Extract list of DEG
idx.sign <- which(res$adj.P.Val < 0.05 &
                    abs(res$logFC) > 1)
deg <- rownames(res[idx.sign,])

# Heatmap of the 50 most differentially expressed proteins
idx.sub <- which(proteo.metadata$condition=="normal" | proteo.metadata$condition=="day7")

data.sub <- proteo.count.norm[deg[1:50],idx.sub]

pheatmap(data.sub,
        cutree_cols = 2,
        show_rownames=FALSE)
```



Analyse multi-omique

Analyse multi-omique (transcripto + protéo) avec, au choix, MOFA, mixOmics, mixKernel ou d'autres outils de factorisation multi-matrices. Vous pouvez soit focaliser sur un time point, soit intégrer les différents time points.

I decided to use mixKernel

Normalization of datasets

We already normalized the two datasets in the first part of the report so I will used those datasets

```
# We check the dimensions of the transcripto and proteo dataset
dim(trans.count.norm) # we have 18465 genes for 18 samples
```

```
## [1] 18465    18
```

```
dim(proteo.count.norm) # we have 8044 proteins for 10 samples
```

```
## [1] 8044   10
```

Filter dataset

I decided to integrate different time point but I need to remove the samples that were not done for both datasets: Day 3 and each third replicate for each sample was not sampled and/or prepared for the proteomic dataset.

```
# samples for transcripto dataset
colnames(trans.count.norm)
```

```
##  [1] "day1_1"   "day1_2"   "day1_3"   "day14_1"  "day14_2"  "day14_3"
##  [7] "day2_1"   "day2_2"   "day2_3"   "day3_1"   "day3_2"   "day3_3"
## [13] "day7_1"   "day7_2"   "day7_3"   "normal_1" "normal_2" "normal_3"
```

```r
# samples for proteo dataset -->
# we are going to subset the transcripto dataset
# with the samples found in the proteo dataset to have the
# same samples in both dataset

col_tokeep <- colnames(proteo.count.norm)
trans.filt <- as.data.frame(trans.count.norm) %>% dplyr::select(any_of(col_tokeep))

colnames(trans.filt) == col_tokeep
```

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

Multiple kernel computation

Individual kernel computation

We build individiual kernel for each dataset

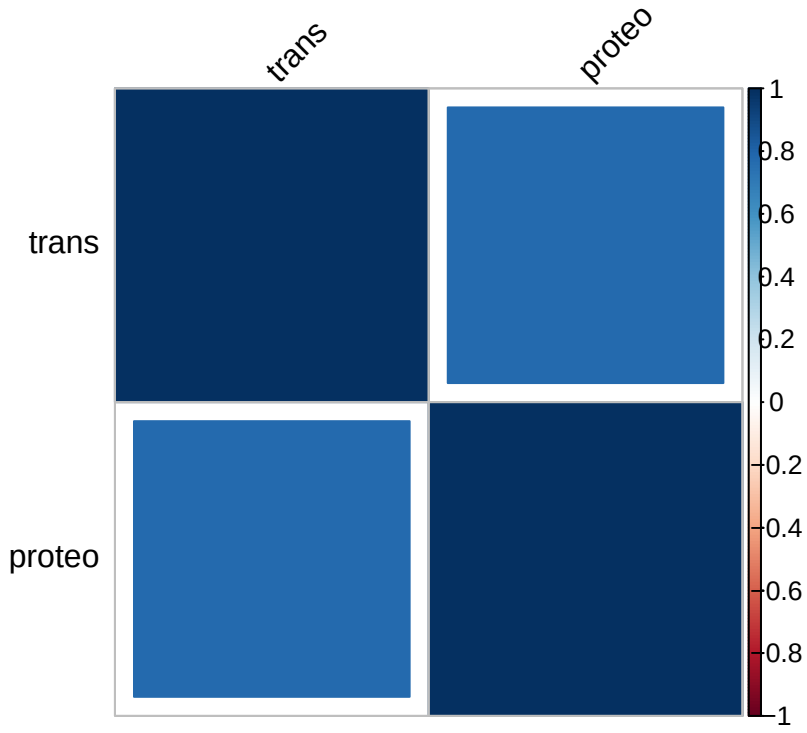
```r
# First, we transpose the datasets as compute.kernel needs the dataframe with
# conditions as rows and genes as columns
trans.filt.t <- t(trans.filt)
proteo.kernel.t <- t(proteo.count.norm)

# Then, we compute each kernel using the linear function as datasets were normalized
trans.kernel <- compute.kernel(trans.filt.t, kernel.func = "linear")
proteo.kernel <- compute.kernel(t(proteo.count.norm), kernel.func = "linear")

# check dimensions
dim(trans.kernel$kernel)
```

```
## [1] 10 10
```

```r
dim(proteo.kernel$kernel)
```

```
## [1] 10 10
```

```r
# A general overview of the correlation structure between datasets
cim.kernel(trans = trans.kernel,
           proteo = proteo.kernel,
           method = "square")
```



It seems that both datasets are positively and strongly correlated

Combined kernel computation

We combined the created kernels for the methof full-UMKL, this method computes a kernel that minimizes the distortion between tht two imput kernels.

```
meta.kernel <- combine.kernels(trans = trans.kernel,
                               proteo = proteo.kernel,
                               method = "full-UMKL")
```

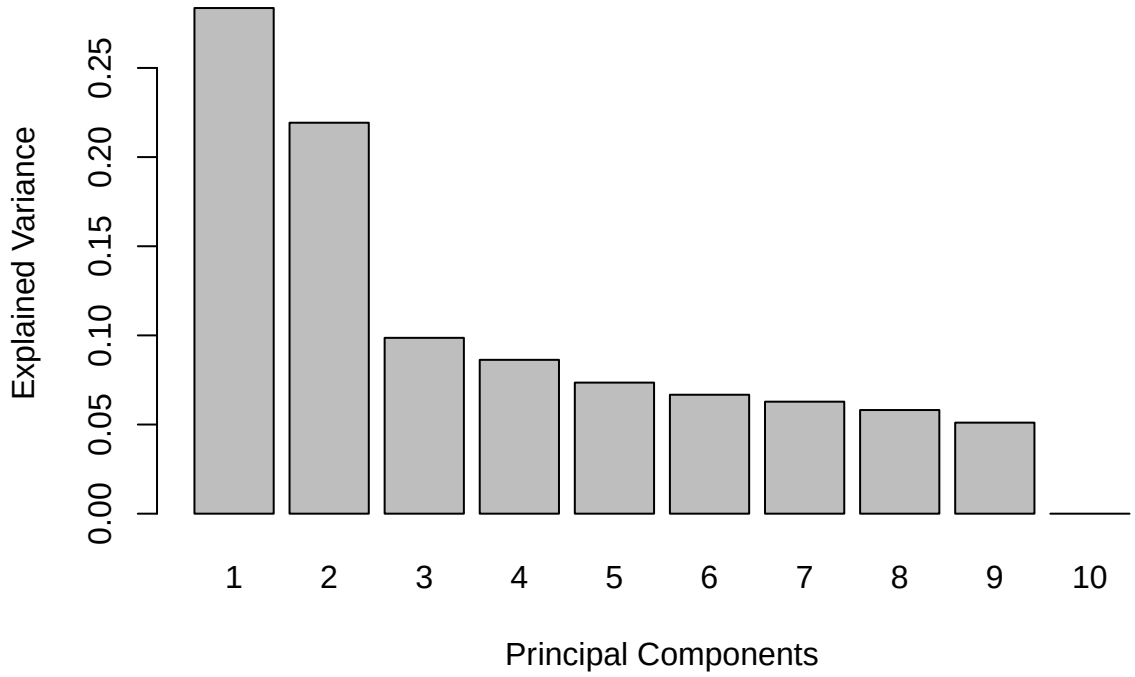
Exploratory analysis with KPCA

We do a KPCA for the 10 first most important components

```
kernel.pca.result <- kernel.pca(meta.kernel, ncomp = 10)
```

With the two first axes, we summarize 0.5 of explained variances as we can see on the plot of eigen values:

```
plot(kernel.pca.result)
```

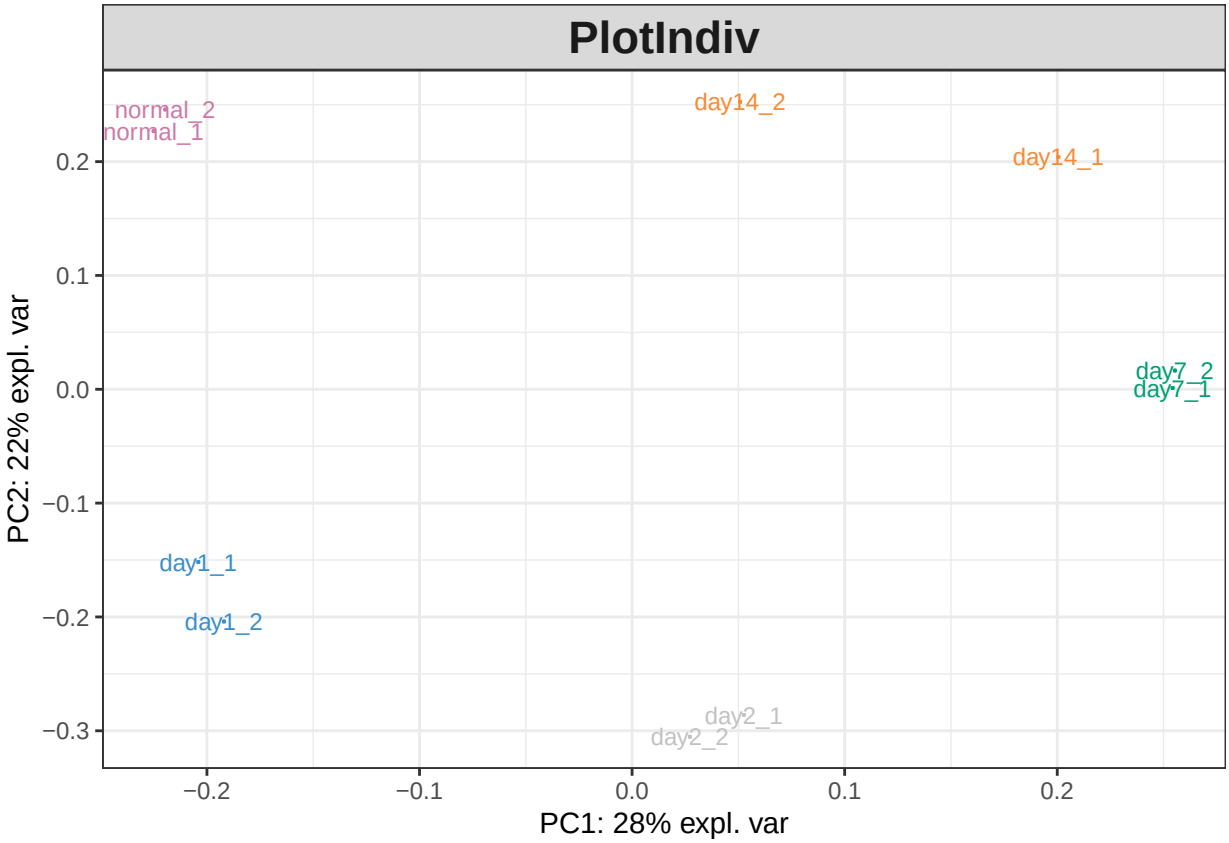


We plot the two first axes of the KPCA with the function of mixOmics

```
# We retrieve metadata related to the samples
metaD_forPCA <- rownames(kernel.pca.result$X) %>%
  as.data.frame() %>%
  rename(sampleName=".") %>%
  inner_join(proteo.metadata)
```

```
## Joining, by = "sampleName"
```

```
plotIndiv(kernel.pca.result,
          comp = c(1, 2),
          ind.names = TRUE,
          group= as.vector(metaD_forPCA$condition))
```



Construction de réseau avec WGCNA

Consigne : Reconstruct the co-expression network from all the time points of the FA transcriptomics data.

Filter

Propose to filter and remove all the zero expressed genes, the NAs and the less informative genes from the transcriptomics data. (I remove all the genes that are not expressed in at least 9 out of the 18 conditions (expression > 1 TPM in 9) and then filter with the coefficient of variation > 0.75).

```
# we look at the dimension of the expression data
head(trans.count.tpm)
```

```
##                      day1_1   day1_2   day1_3  day14_1  day14_2  day14_3
## ENSMUSG00000000001 18.217598 7.318595 6.181582 6.343883 5.270735 4.532972
## ENSMUSG00000000003  0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
## ENSMUSG00000000028  0.698490 0.218534 0.248198 0.352222 0.230849 0.565890
## ENSMUSG00000000031  0.237079 0.065881 0.006545 0.019711 0.000000 0.000000
## ENSMUSG00000000037  0.000000 0.117586 0.018518 0.059173 0.018440 0.004995
## ENSMUSG00000000049  0.919727 0.074244 0.500737 0.985273 0.069168 1.543102
##                      day2_1   day2_2   day2_3   day3_1   day3_2   day3_3
## ENSMUSG00000000001 7.917233 6.416987 3.548104 4.038720 9.599544 6.384702
## ENSMUSG00000000003 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
## ENSMUSG00000000028 2.671730 0.281002 1.419609 0.510979 0.180966 0.805545
## ENSMUSG00000000031 0.014207 0.000000 0.056274 0.007380 0.000000 0.000000
## ENSMUSG00000000037 0.217848 0.000000 0.190351 0.037585 0.063837 0.000000
## ENSMUSG00000000049 0.148542 0.250424 0.032027 0.014441 0.895128 0.136669
##                      day7_1    day7_2   day7_3 normal_1 normal_2 normal_3
## ENSMUSG00000000001 11.555237 10.538166 8.340840 3.615573 6.467585 5.427865
## ENSMUSG00000000003  0.000000  0.000000 0.000000 0.000000 0.000000 0.000000
## ENSMUSG00000000028  0.169244  0.170429 0.784728 0.132484 0.094683 0.028430
```

```
## ENSMUSG00000000031  0.000000  0.019420 0.029434 0.014474 0.000000 0.000000
## ENSMUSG00000000037  0.086099  1.063765 0.005117 0.246735 0.037644 0.000000
## ENSMUSG00000000049  1.266932  0.254115 0.967421 0.774452 0.502925 0.676517
```

```
dim(trans.count.tpm)
```

```
## [1] 46679    18
```

```
# we have 46679 genes and 18 samples

# we remove zeros and NAs
faD <- trans.count.tpm[apply(trans.count.tpm, 1, function(row) all(row !=0 )), ]
faD <- faD[complete.cases(faD),]

# small quality control of WGCNA
gsgFA = goodSamplesGenes(faD, verbose = 3)
```

```
##  Flagging genes and samples with too many missing values...
##   ..step 1
```

```
gsgFA$allOK # all genes are OK we can pursue
```

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

```
# We keep only informative genes so we decided
# to remove genes which are not expressed in at least
# 9 out of the 18 conditions (expression > 1 TPM in 9)
# and then filter with the coefficient of variation > 0.75

faD <- faD[apply(faD, 1, sum) >= 9,]
faD <- faD[apply(faD, 1, CoefVar) >= 0.75, ]

dim(faD) # we have now 4530 genes that will be used in the network
```

```
## [1] 4530   18
```

```
# We transpose the dataframe as WGCNA needs the dataframe with
# conditions as rows and genes as columns
faD.t <- t(faD)
head(faD.t)[,1:2] # we look at the first columns to check
```

```
##         ENSMUSG00000000028 ENSMUSG00000000049
## day1_1            0.698490           0.919727
## day1_2            0.218534           0.074244
## day1_3            0.248198           0.500737
## day14_1           0.352222           0.985273
## day14_2           0.230849           0.069168
## day14_3           0.565890           1.543102
```

```
dim(faD.t)
```

```
## [1]   18 4530
```

Network reconstruction

Then apply the first part of the network reconstruction steps as we saw them on the WGCNA course until the module predictions. Instead of using WGCNA's module prediction routines, apply a universal threshold of 0.5 on the adjacency matrix, and obtain an adjacency matrix that is reduced in size. This is the network.

Choice of the soft-thresholding power

We first identify the soft-thresholding power to which co-expression similarity is raised to calculate adjacency

```
## Chose a set of soft-thresholding powers
powers = c(c(1:10), seq(from = 12, to=20, by=2))
```

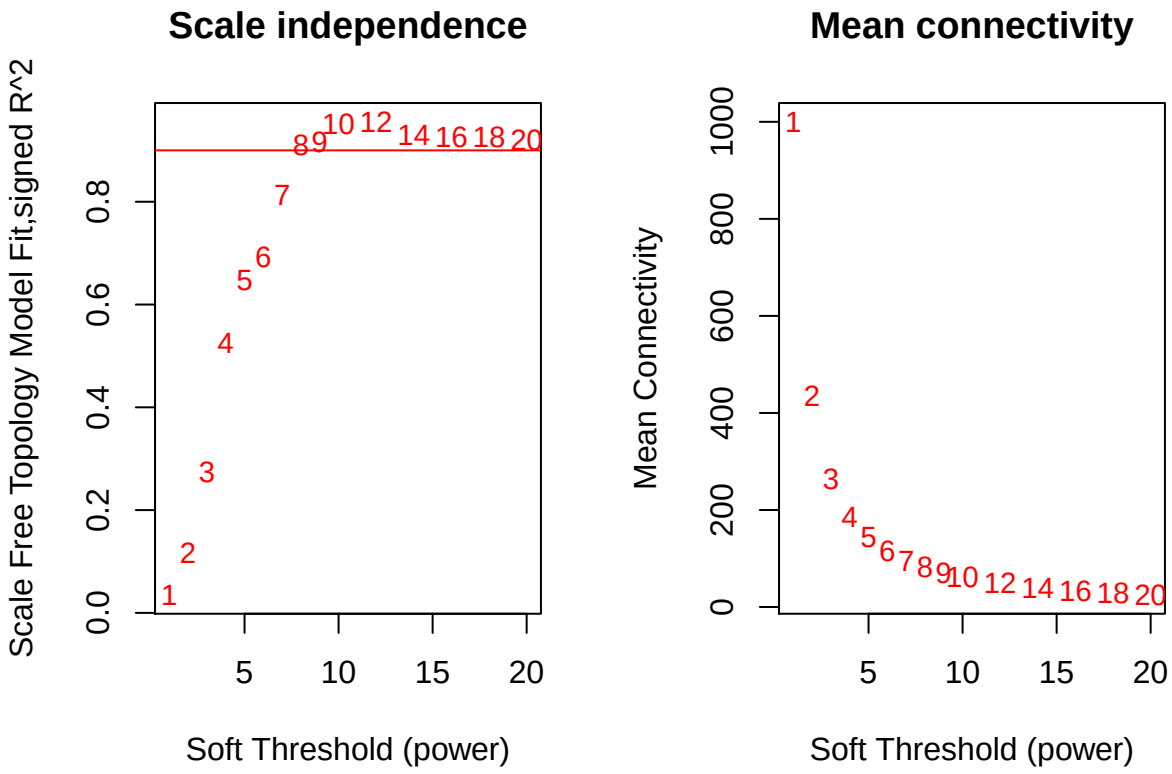
```r
# Call the network topology analysis function
sft = pickSoftThreshold(faD.t, powerVector = powers, verbose = 5)
```

```
## pickSoftThreshold: will use block size 4530.
##  pickSoftThreshold: calculating connectivity for given powers...
##    ..working on genes 1 through 4530 of 4530
```

```
## Warning: executing %dopar% sequentially: no parallel backend registered
```

```
##    Power SFT.R.sq  slope truncated.R.sq mean.k. median.k. max.k.
## 1      1   0.0351 -0.417          0.851  1000.0    979.00   1640
## 2      2   0.1160 -0.474          0.813   436.0    413.00    916
## 3      3   0.2750 -0.507          0.902   264.0    248.00    628
## 4      4   0.5260 -0.726          0.887   187.0    160.00    499
## 5      5   0.6470 -0.843          0.879   143.0    113.00    426
## 6      6   0.6930 -0.941          0.824   115.0     89.10    373
## 7      7   0.8130 -0.960          0.871    96.0     69.70    333
## 8      8   0.9100 -0.950          0.929    81.6     55.60    300
## 9      9   0.9160 -0.971          0.917    70.6     45.70    274
## 10    10   0.9510 -1.030          0.954    62.0     38.00    261
## 11    12   0.9550 -1.110          0.953    49.2     27.40    238
## 12    14   0.9310 -1.170          0.923    40.3     20.30    220
## 13    16   0.9260 -1.200          0.915    33.8     15.30    206
## 14    18   0.9270 -1.220          0.914    28.9     11.70    194
## 15    20   0.9200 -1.230          0.904    25.1      8.92    183
```

```r
# Plot the results:
par(mfrow = c(1,2));cex1 = 0.9;
# Scale-free topology fit index as a function of the soft-thresholding power
plot(sft$fitIndices[,1], -sign(sft$fitIndices[,3])*sft$fitIndices[,2],
     xlab="Soft Threshold (power)",ylab="Scale Free Topology Model Fit,signed R^2",type="n",
     main = paste("Scale independence"));
text(sft$fitIndices[,1], -sign(sft$fitIndices[,3])*sft$fitIndices[,2],
     labels=powers,cex=cex1,col="red");
# this line corresponds to using an R^2 cut-off of h
abline(h=0.90,col="red")
# Mean connectivity as a function of the soft-thresholding power
plot(sft$fitIndices[,1], sft$fitIndices[,5],
     xlab="Soft Threshold (power)",ylab="Mean Connectivity", type="n",
     main = paste("Mean connectivity"))
text(sft$fitIndices[,1], sft$fitIndices[,5], labels=powers, cex=cex1,col="red")
```



We choose the power 8, which is the lowest power for which the scale-free topology fit index reaches 0.90.

Build network and module detection

```
### -- we build the network and detect module using
# an automatic block-wise network construction and
# module detection method
# we choose the mimnimum module size relatively high
# (here 30) as it is better to have large modules

cor <- WGCNA::cor # to avoid conflict with other packages (https://programmersought.com/article/90752004413/)

net = blockwiseModules(faD.t, power = 8,
                       TOMType = "unsigned", minModuleSize = 30,
                       reassignThreshold = 0, mergeCutHeight = 0.25,
                       numericLabels = TRUE, pamRespectsDendro = FALSE,
                       saveTOMs = TRUE,
                       verbose = 3)
```

```
##  Calculating module eigengenes block-wise from all genes
##    Flagging genes and samples with too many missing values...
##     ..step 1
##  ..Working on block 1 .
##     TOM calculation: adjacency..
##     ..will not use multithreading.
##      Fraction of slow calculations: 0.000000
##     ..connectivity..
##     ..matrix multiplication (system BLAS)..
##     ..normalization..
##     ..done.
##    ..saving TOM for block 1 into file blockwiseTOM-block.1.RData
##  ....clustering..
##  ....detecting modules..
```

```
##  ....calculating module eigengenes..
##  ....checking kME in modules..
##      ..removing 1 genes from module 1 because their KME is too low.
##      ..removing 1 genes from module 2 because their KME is too low.
##  ..merging modules that are too close..
##      mergeCloseModules: Merging modules whose distance is less than 0.25
##        Calculating new MEs...
```

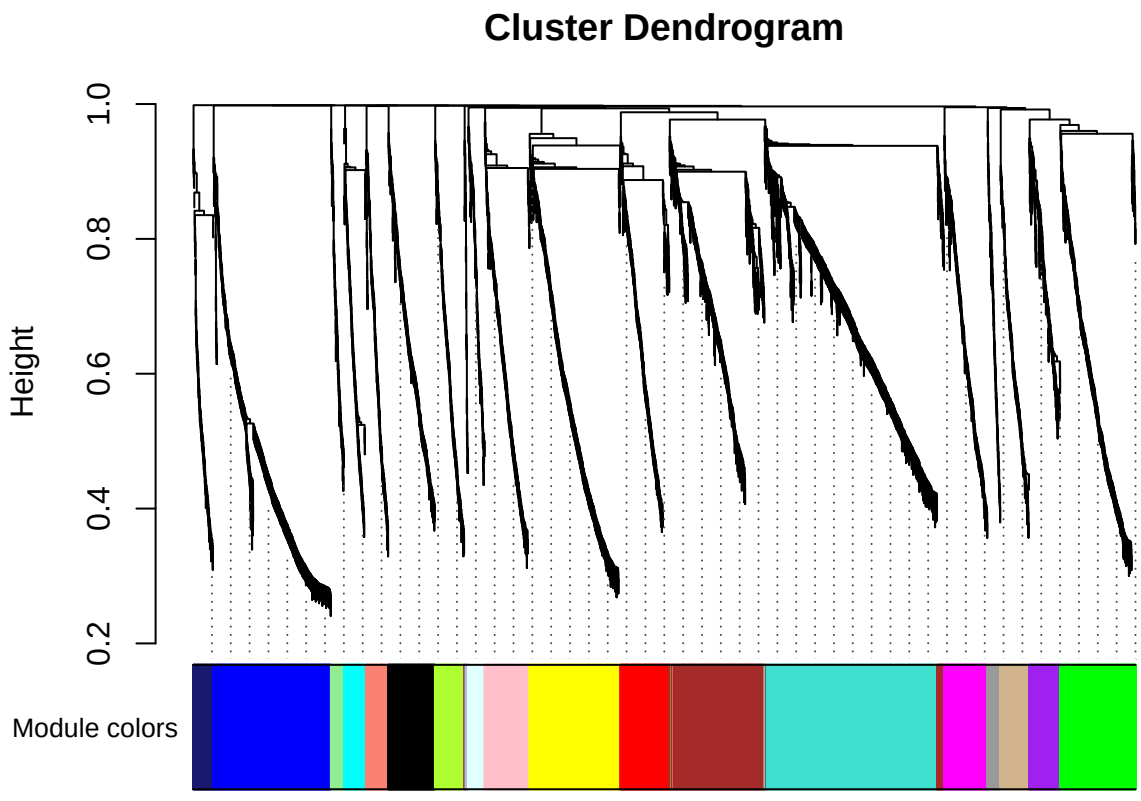
```
### -- We have a look at the detected modules
table(net$colors)
```

```
##
##   0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
##  12 826 564 492 438 362 236 224 218 208 153 142 139 108 105  97  80  63  63
```

We see we have 17 modules (from 1 to 18 with the 1 with the highest number of genes and the last one the lowest number of genes), the label 0 indicates that 12 genes are not associated to a specific module.

To visualize the relationship between genes clustering and detected modules (ie to see where the "cutting the branches" of the gene tree was perfomed with the blockwiseModules), we perform the hierarchical clustering tree and add below the modules (each module has its specific color).

```
### -- We have a look at the gene tree and the associated modules
# Convert labels to colors for plotting
mergedColors = labels2colors(net$colors)
# Plot the dendrogram and the module colors underneath
plotDendroAndColors(net$dendrograms[[1]], mergedColors[net$blockGenes[[1]]],
                    "Module colors",
                    dendroLabels = FALSE, hang = 0.03,
                    addGuide = TRUE, guideHang = 0.05)
```




```
# save some results
moduleLabels = net$colors
moduleColors = labels2colors(net$colors)
MEs = net$MEs
geneTree = net$dendrograms[[1]]
```

Export network to Cytoscape

```
# --- We prepare the dataframe needed to create the network
# that will be imported into Cytoscape

# We frst recalculate the topological overlap
TOM = TOMsimilarityFromExpr(faD.t, power = 8)
```

```
## TOM calculation: adjacency..
## ..will not use multithreading.
##  Fraction of slow calculations: 0.000000
## ..connectivity..
## ..matrix multiplication (system BLAS)..
## ..normalization..
## ..done.
```

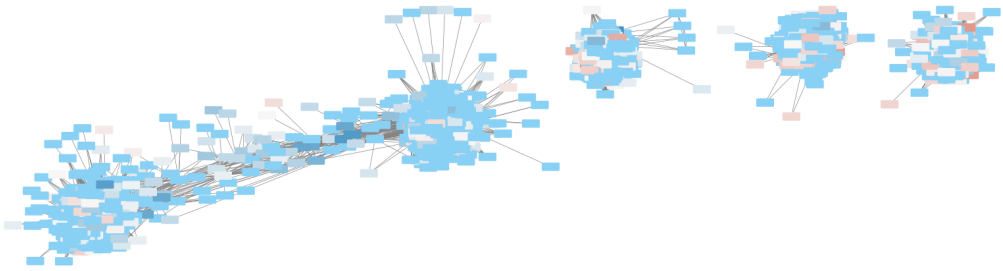
```
geneNames <- rownames(faD)

# We select the modules
# We choose 5 modules by selecting the bigger ones and
# also by not taking into account the one (grey color) that contains all the not-assigned genes

mods <- c("turquoise", "blue","brown","yellow","green")
inModule <- is.finite(match(moduleColors, mods))
modGenes <- geneNames[inModule]
modTOM <- TOM[inModule, inModule]
dimnames(modTOM) <- list(modGenes, modGenes)

# In order to reduce the adjaceny matrix, we apply a
# universal threshold of 0.25 on the adjacency matrix

cyt = exportNetworkToCytoscape(modTOM,
                               edgeFile = paste("CytoscapeInput-edges-0.25",
                                                paste(mods, collapse="-"), ".txt", sep=""),
                               nodeFile = paste("CytoscapeInput-nodes-0.25",
                                                paste(mods, collapse="-"), ".txt", sep=""),
                               threshold = 0.25,
                               altNodeNames = modGenes,
                               nodeAttr = moduleColors[inModule])
```

Import it to Cytoscape with aMatReader plugin. Visualize, analyze the network and superimpose the proteomics data on it.

Color reseau cytoscpae

Colorez dans le réseau choisi les noeuds en fonction des données de protéomiques avec un gradient de couleur correspondant au fold-change des données de protéomique.

Here my network where I decrease the option threshold to 0.25 in exportNetworkToCytoscape function to have some modules which were connected. To be honest, I am not sure of what I did, I just surimposed the proteimic dataat and changed the color in function of fold-change, I did not try to “arrange” or “make more readable” the network.



R session info

```
sessionInfo()
```

```
## R version 3.6.3 (2020-02-29)
## Platform: x86_64-conda_cos6-linux-gnu (64-bit)
## Running under: CentOS Linux 7 (Core)
##
## Matrix products: default
## BLAS/LAPACK: /shared/mfs/data/software/miniconda/envs/r-3.6.3/lib/libopenblasp-r0.3.9.so
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] parallel  stats4    stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] dplyr_1.0.2                 mixKernel_0.4
##  [3] reticulate_1.16             mixOmics_6.10.9
##  [5] ggplot2_3.3.2               lattice_0.20-41
##  [7] MASS_7.3-51.6               DescTools_0.99.38
##  [9] WGCNA_1.69                  fastcluster_1.1.25
## [11] dynamicTreeCut_1.63-1       pheatmap_1.0.12
## [13] tidyr_1.1.2                 edgeR_3.28.1
## [15] limma_3.42.2                DESeq2_1.26.0
## [17] SummarizedExperiment_1.16.1 DelayedArray_0.12.3
## [19] BiocParallel_1.20.1         matrixStats_0.56.0
## [21] Biobase_2.46.0              GenomicRanges_1.38.0
```

```
## [23] GenomeInfoDb_1.22.1         IRanges_2.20.2
## [25] S4Vectors_0.24.4            BiocGenerics_0.32.0
## [27] tibble_3.0.3                knitr_1.29
##
## loaded via a namespace (and not attached):
##   [1] backports_1.1.9        Hmisc_4.4-1            corrplot_0.84
##   [4] plyr_1.8.6             igraph_1.2.5           splines_3.6.3
##   [7] digest_0.6.25          foreach_1.5.0          htmltools_0.5.0
##  [10] GO.db_3.10.0           magrittr_1.5           checkmate_2.0.0
##  [13] memoise_1.1.0          cluster_2.1.0          doParallel_1.0.15
##  [16] Biostrings_2.54.0      annotate_1.64.0        rARPACK_0.11-0
##  [19] jpeg_0.1-8.1           colorspace_1.4-1       blob_1.2.1
##  [22] xfun_0.17              crayon_1.3.4           RCurl_1.98-1.2
##  [25] jsonlite_1.7.1         genefilter_1.68.0      Exact_2.0
##  [28] impute_1.60.0          survival_3.2-3         iterators_1.0.12
##  [31] ape_5.4-1              glue_1.4.2             gtable_0.3.0
##  [34] zlibbioc_1.32.0        XVector_0.26.0         phyloseq_1.30.0
##  [37] Rhdf5lib_1.8.0         scales_1.1.1           mvtnorm_1.1-1
##  [40] DBI_1.1.0              Rcpp_1.0.5             xtable_1.8-4
##  [43] htmlTable_2.0.1        tmvnsim_1.0-2          foreign_0.8-76
##  [46] bit_4.0.4              preprocessCore_1.48.0  Formula_1.2-3
##  [49] htmlwidgets_1.5.1      RColorBrewer_1.1-2     ellipsis_0.3.1
##  [52] farver_2.0.3           pkgconfig_2.0.3        XML_3.99-0.3
##  [55] nnet_7.3-14            locfit_1.5-9.4         labeling_0.3
##  [58] tidyselect_1.1.0       rlang_0.4.7            reshape2_1.4.4
##  [61] AnnotationDbi_1.48.0   munsell_0.5.0          tools_3.6.3
##  [64] generics_0.0.2         RSQLite_2.2.0          ade4_1.7-15
##  [67] evaluate_0.14          biomformat_1.14.0      stringr_1.4.0
##  [70] yaml_2.2.1             bit64_4.0.5            purrr_0.3.4
##  [73] nlme_3.1-147           compiler_3.6.3         rstudioapi_0.11
##  [76] png_0.1-7              e1071_1.7-3            geneplotter_1.64.0
##  [79] stringi_1.5.3          highr_0.8              RSpectra_0.16-0
##  [82] Matrix_1.2-18          psych_2.0.8            vegan_2.5-6
##  [85] permute_0.9-5          multtest_2.42.0        vctrs_0.3.4
##  [88] pillar_1.4.6           lifecycle_0.2.0        data.table_1.13.0
##  [91] bitops_1.0-6           corpcor_1.6.9          lmom_2.8
##  [94] R6_2.4.1               latticeExtra_0.6-29    gridExtra_2.3
##  [97] gld_2.6.2              codetools_0.2-16       LDRTools_0.2-1
## [100] boot_1.3-25            rhdf5_2.30.1           withr_2.2.0
## [103] mnormt_2.0.2           GenomeInfoDbData_1.2.2 mgcv_1.8-31
## [106] expm_0.999-5           quadprog_1.5-8         grid_3.6.3
## [109] rpart_4.1-15           class_7.3-17           rmarkdown_2.3
## [112] base64enc_0.1-3        ellipse_0.4.2
```