Microarray practical

Introduction

In this practical you will analyse microarray data using R. The data files provided are data files that already underwent image analysis and can be downloaded from Canvas (Affy_data.zip). This is rat genomic data from Affymetrix GeneChips, a very popular platform for measuring genome-wide expression levels. In this experiment, some rats receive a treatment while some others are used as a control.

The downloaded files include 6 CEL files, which are text files containing the information from each microarray experiment, and a "experimentdescription.txt" text file containing the description of each microarray (which CEL files corresponds to rats that received treatment and which correspond to the control group). Feel free to open and explore them. Note that in this case there is only one condition, but in more complicated experiments more than one column can be included in the descriptions file.

The aim of this practical is to find genes that are differentially expressed in rats which received a treatment and find relevant information or annotate them. For this purpose, you will use Bioconductor packages. Bioconductor is a widely used open source software for Bioinformatics that provides tools for biological data analysis using R. Some of the packages needed for this practical are specific for Affymetrix ("affy", "affyPLM", "annaffy" and "rgu34a.db") and the package "limma" is used for statistical analysis (not specific to Affymetrix).

To check if these packages are installed you can try to load them using i.e. library(limma) or require(limma). If they are not, you can install them using:

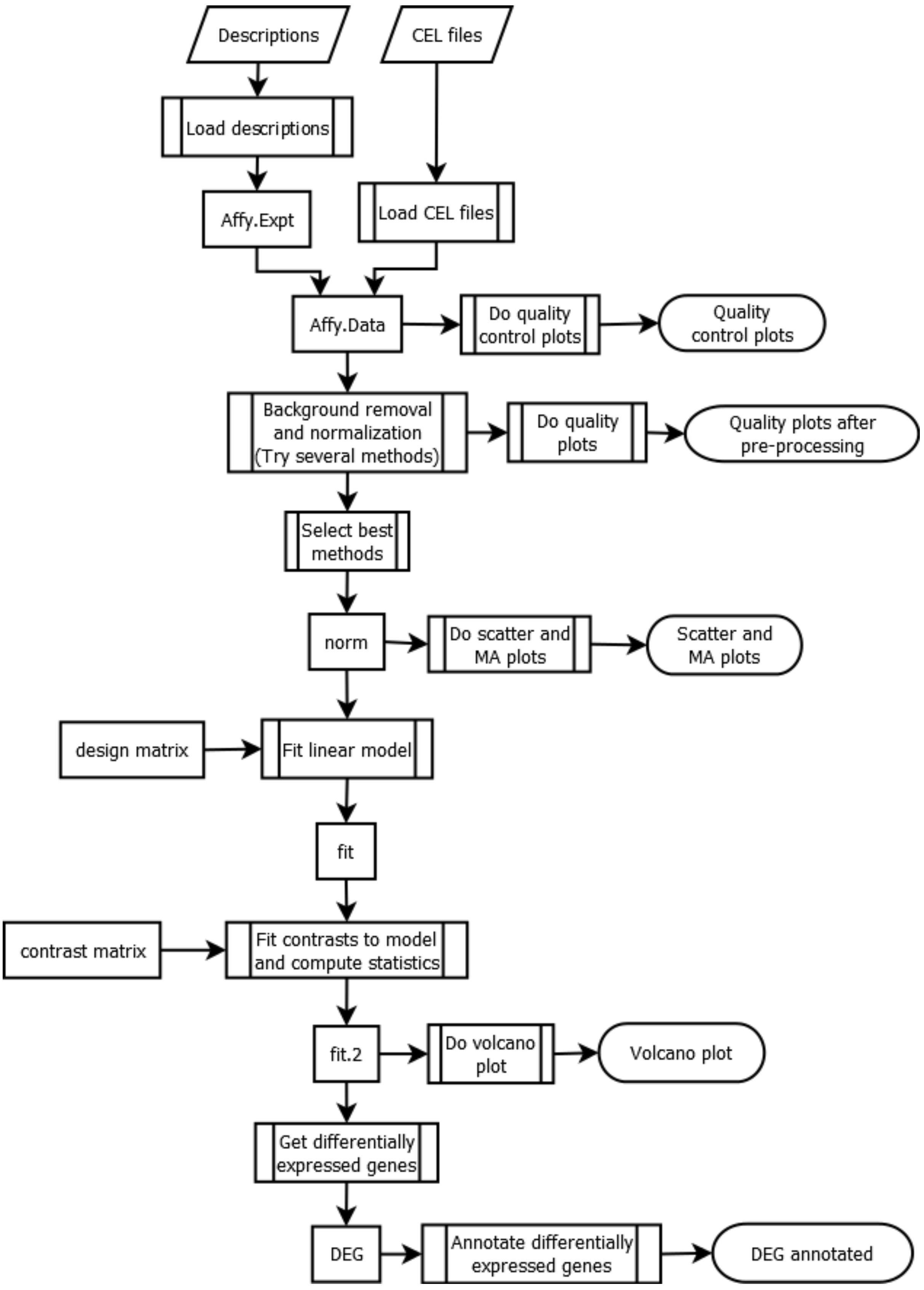
```
if (!require("BiocManager", quietly = TRUE))

    install.packages("BiocManager")

BiocManager::install(c("affy","affyPLM", "limma", "annaffy", "rgu34a.db"))
```

Bioconductor packages are very well documented. You can access their documentation using i.e. browseVignettes("limma") or to find more information about the usage of a specific function from any package you can use i.e. ?boxplot.

To start working, set the working directory were the files are located using setwd() i.e. setwd("Affy_Data") and it is a good practice to clear the environment and the graphics area using rm(list=ls()) and graphics.off(). Now you are ready to start the analysis. As a quick guideline, the procedure will consist on:

a) Loading the microarray data.
b) Pre-processing.
c) Finding differentially expressed genes.
d) Annotation.

A diagram showing the workflow for the analysis is displayed in the following page.


Descriptions
CEL files
Load descriptions
Affy.Expt
Load CEL files
Affy.Data
Do quality control plots
Quality control plots
Background removal and normalization (Try several methods)
Do quality plots
Quality plots after pre-processing
Select best methods
norm
Do scatter and MA plots
Scatter and MA plots
design matrix
Fit linear model
fit
contrast matrix
Fit contrasts to model and compute statistics
fit.2
Do volcano plot
Volcano plot
Get differentially expressed genes
DEG
Annotate differentially expressed genes
DEG annotated

Loading the microarray data

The first thing to be done is to load the microarray data. For this purpose, the affy package is used:

```
library(affy)

Affy.Expt <- read.AnnotatedDataFrame("experimentdescription.txt", header=TR
UE, row.names=1, sep="\t")


Affy.Data <- ReadAffy(filenames=rownames(pData(Affy.Expt)), phenoData=Affy.
Expt, verbose=TRUE)
```

Take some time to explore all the parameters and understand their inputs. You can see the type of object generated by each one of the functions using:

```
class(Affy.Expt)
```

```
[1] "AnnotatedDataFrame"
attr(,"package")
[1] "Biobase"
```

```
class(Affy.Data)
```

```
[1] "AffyBatch"
attr(,"package")
[1] "affy"
```

Affy.Expt is an AnnotatedDataFrame, an object from Biobase (the base functions from Bioconductor), and Affy.Data is an AffyBatch object from the affy package. You can explore what is inside this objects using the function attributes().

For example:

```
attributes(Affy.Data)
names(attributes(Affy.Data))
```

```
 [1] ".__classVersion__" "cdfName"
 [3] "nrow"              "ncol"
 [5] "assayData"         "phenoData"
 [7] "featureData"       "experimentData"
 [9] "annotation"        "protocolData"
[11] "class"
```

```
attributes(Affy.Data)$annotation
```

```
[1] "rgu34a"
```

Here you found the source for annotation, which will be carried out later. Also check the size of the dimensions of the microarray and see the first lines of the expression data. You can see the whole matrix typing View(exprs(Affy.Data)).

```
dim(Affy.Data)
```

```
Rows Cols
 534  534
```

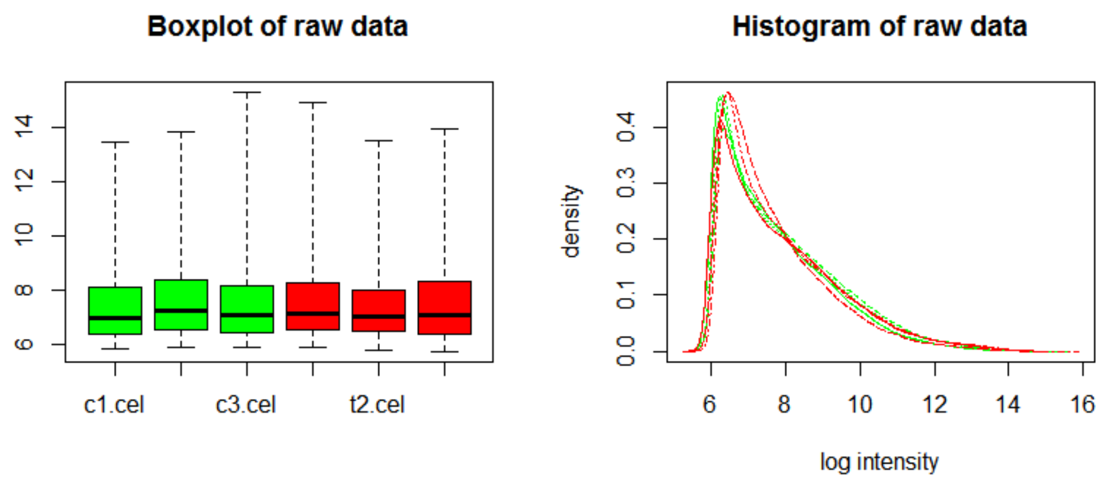
```
head(exprs(Affy.Data)) #first lines of expression data contained in Affy.Data
```

```
   c1.cel  c2.cel  c3.cel  t1.cel  t2.cel  t3.cel
1   110.5   115.0   116.5   124.8   134.5   204.3
2 10359.3 10697.5 11334.0 10685.5  9573.5 16397.8
3   117.3   139.3   125.0   136.8   160.5   145.0
4 10578.5 10829.8 11281.3 10888.8 10032.3 16954.3
5    59.0    71.0    76.3    69.5    77.3    75.0
6   109.5   106.0   115.3    95.0   138.3   123.0
```

Now, you may want to generate some plots to explore your data. The generated plots will be displayed on the graphics area on RStudio. Alternatively, you may want to display the plots in a separated window by typing x11() or save it into a file using png(), tiff(), pdf()… In order to check how they work and the input parameters they take you can use ?png, ?tiff or ?pdf. This information is useful because if your figure is too large for the RStudio graphics area, then you will get a “Error in plot.new() : figure margins too large” error and you will need an alternative solution.

You can generate boxplots and histograms to see the distribution of the data. You will plot controls in green and treatments in red.

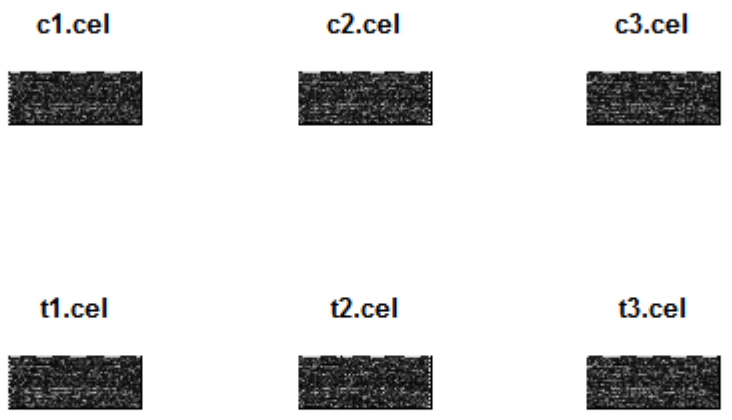
```
colour <- c(rep("green",3),rep("red",3))
par(mfrow=c(1,2))
boxplot(Affy.Data, col= colour, main = "Boxplot of raw data")
hist(Affy.Data,type="l", col=colour,main = "Histogram of raw data")
```

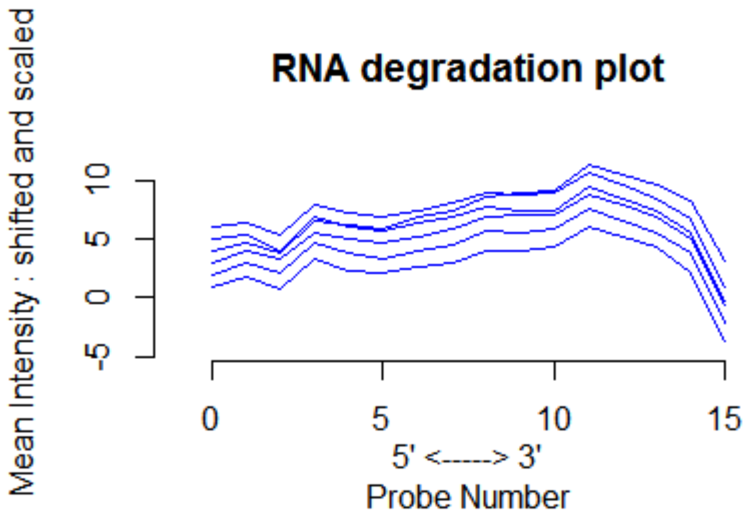


You can plot the chip images for each one of the arrays at the probe level and a RNA degradation plot. Use the help and check the documentation to find more information about the functions used to generate the plots. Check the function sapply as well, as it is not strictly related to microarray analysis but it comes in handy when dealing with vectors in R.

```
par(mfrow=c(2,3))
```

```
chip<-sapply(1:ncol(exprs(Affy.Data)), function(x) image(Affy.Data[,x]))
RNA.deg <- AffyRNAdeg(Affy.Data)
par(mfrow=c(1,1))
plotAffyRNAdeg(RNA.deg)
```





Pre-processing

Pre-processing for microarray data consists of three steps: background correction to remove the effect of unspecific binding, normalization between arrays and summarization to combine all the information from the multiple probes in the same array (genes are represented more than once in the same array). You will perform preprocessing using the threestep() function from the affyPLM package.

```
library(affyPLM)
```

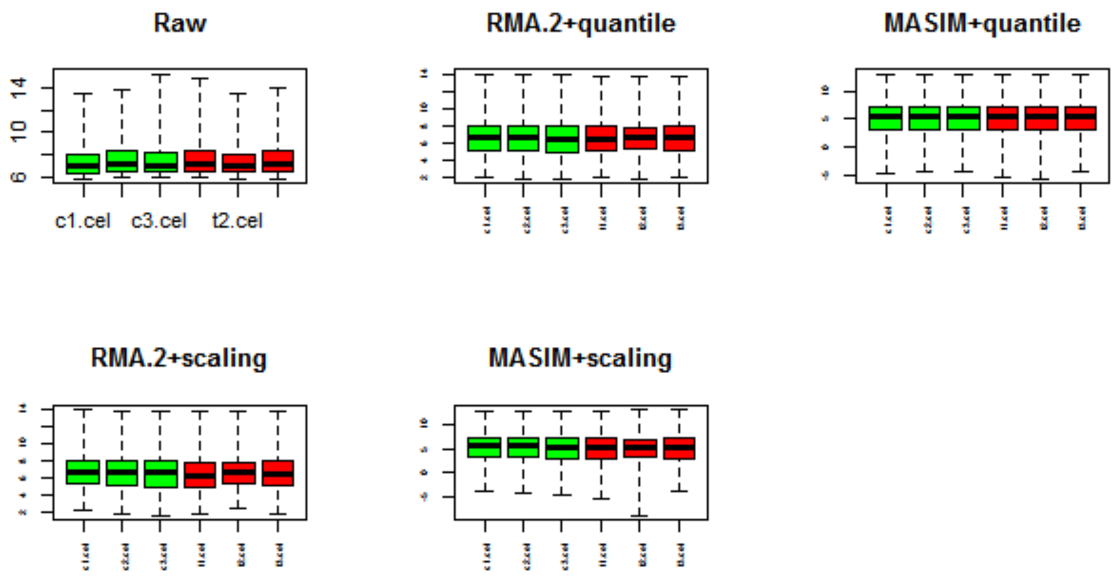
To make sure you are using a good background correction and normalization method it is a good idea to try several combinations. Here we are trying “RMA.2” and “MASIM” for background correction and “quantile” and “scaling” for normalization.

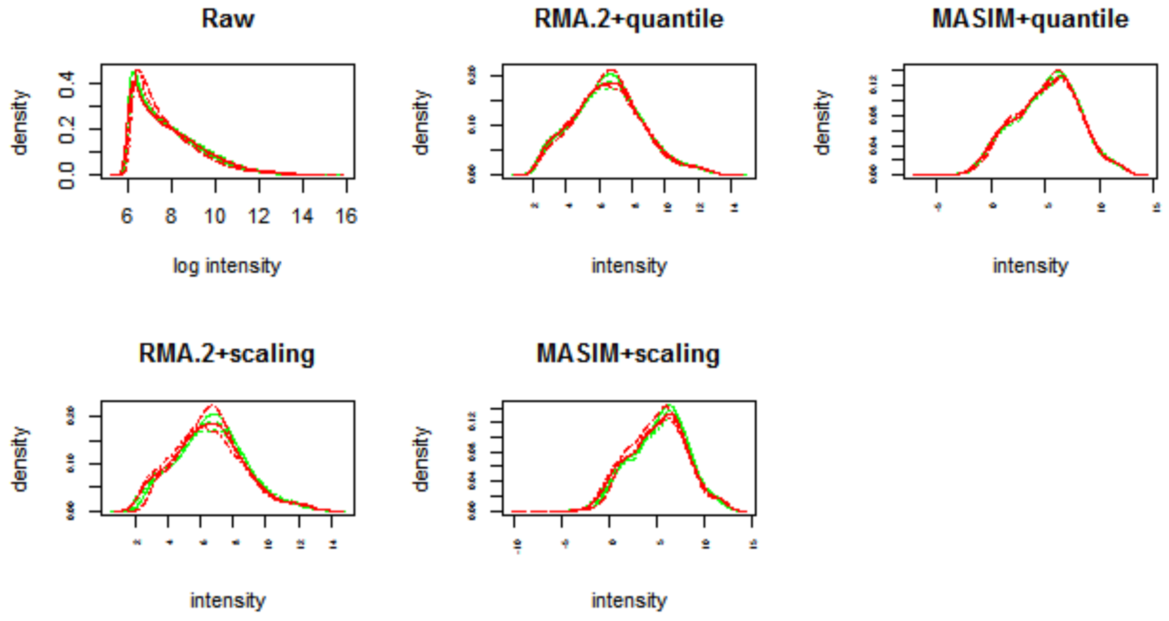
```
norm.rma2.quantile <- threestep(Affy.Data, background.method="RMA.2",
        normalize.method="quantile",summary.method="median.polish")

norm.masim.quantile <- threestep(Affy.Data, background.method="MASIM",
        normalize.method="quantile",summary.method="median.polish")

norm.rma2.scaling <- threestep(Affy.Data, background.method="RMA.2",
        normalize.method="scaling",summary.method="median.polish")


norm.masim.scaling <- threestep(Affy.Data, background.method="MASIM",

        normalize.method="scaling",summary.method="median.polish")
```

Then boxplots and histograms are useful to know which methods are better for our data.

```
# Compare boxplots
par(mfrow=c(2,3))

boxplot(Affy.Data, col= colour, main = "Raw")

boxplot(norm.rma2.quantile, las=3, cex.axis=0.5, pch=".",
        col=colour,main = "RMA.2+quantile")

boxplot(norm.masim.quantile, las=3, cex.axis=0.5, pch=".",
        col=colour,main = "MASIM+quantile")

boxplot(norm.rma2.scaling, las=3, cex.axis=0.5, pch=".",
        col=colour,main = "RMA.2+scaling")

boxplot(norm.masim.scaling, las=3, cex.axis=0.5, pch=".",
        col=colour,main = "MASIM+scaling")
```

```
# Compare histograms
par(mfrow=c(2,3))

hist(Affy.Data, col=colour,main = "Raw")

hist(norm.rma2.quantile, las=3, cex.axis=0.5, pch=".",
        col=colour,main = "RMA.2+quantile")

hist(norm.masim.quantile, las=3, cex.axis=0.5, pch=".",
        col=colour,main = "MASIM+quantile")

hist(norm.rma2.scaling, las=3, cex.axis=0.5, pch=".",
        col=colour,main = "RMA.2+scaling")

hist(norm.masim.scaling, las=3, cex.axis=0.5, pch=".",
        col=colour,main = "MASIM+scaling")

par(mfrow=c(1,1))
```



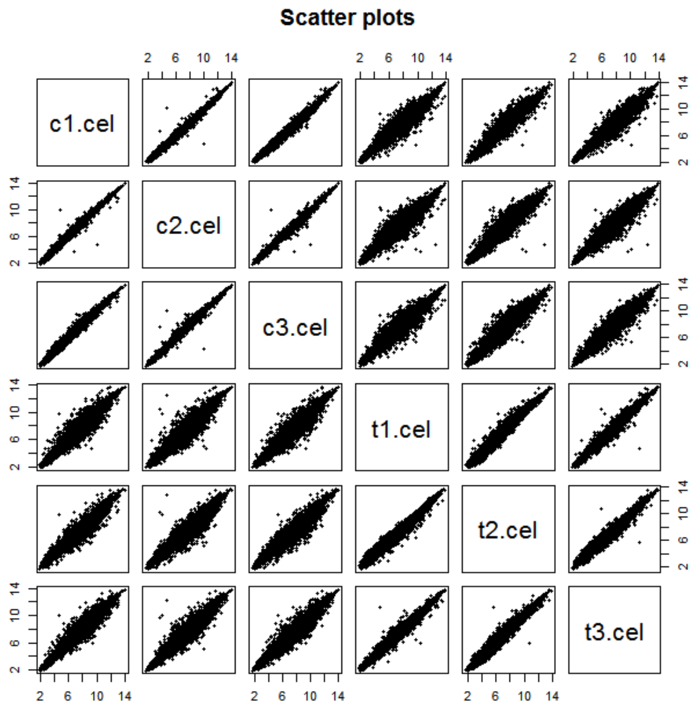


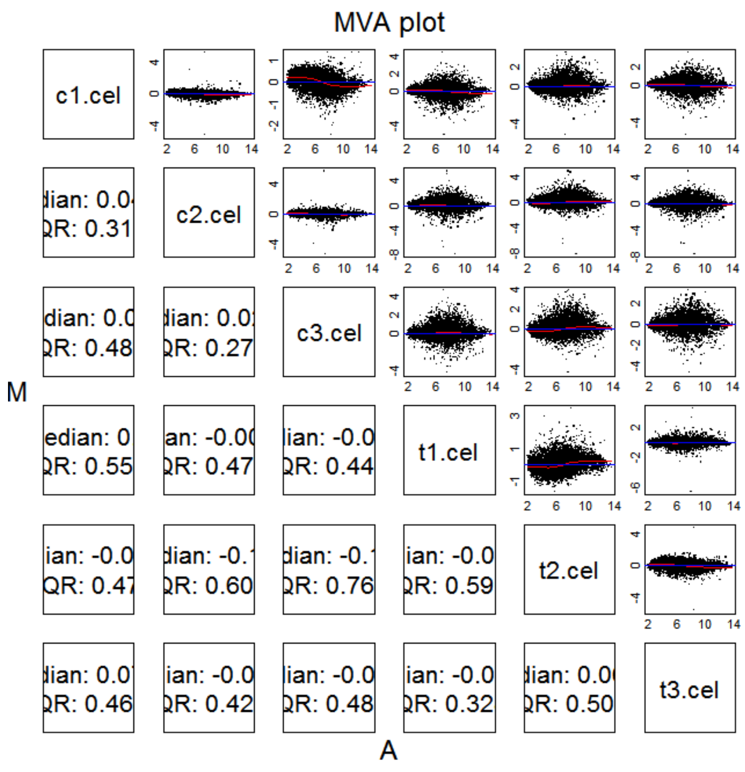
From these plots we can observe that quantile normalization work better than scaling. We are going to use “RMA.2” and “quantile”, but you can try more methods for the threestep() function or for another function, i.e. expresso() from the affy package.

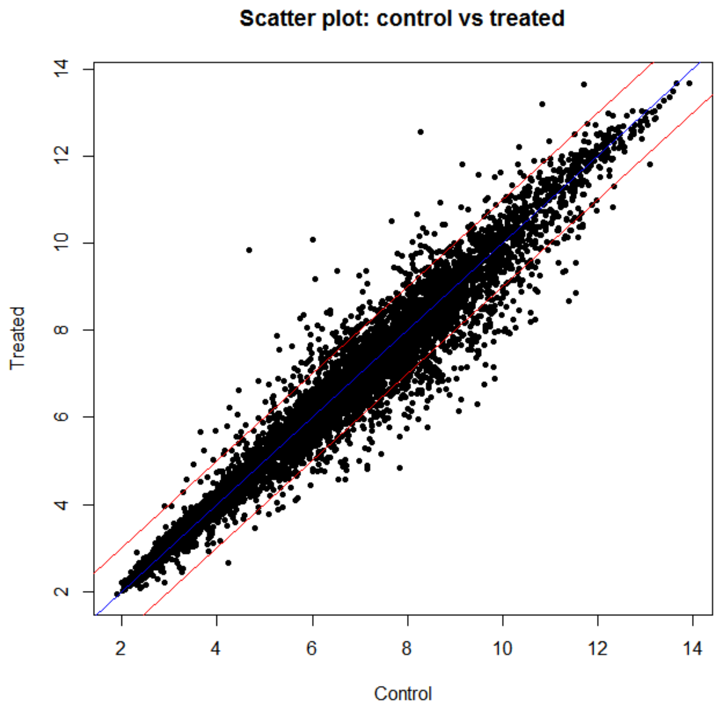
```
norm<-norm.rma2.quantile
```

With the pre-processed data you will generate scatter plots of all arrays against each other, MA plots (M (red/green ratio) vs A (average intensity) and a scatter plot (control vs treated)

```
norm.exprs<-as.data.frame(exprs(norm))
# Scatter matrix of all arrays against one another
pairs(norm.exprs, pch=20,main="Scatter plots")
# MA plots of all arrays against one another (log-intensity vs log-ratio
mva.pairs(norm.exprs,log.it=F)

# To compare with the raw data:

mva.pairs(exprs(Affy.Data))

# Scatter plot of Control vs treated
plot(x = rowMeans(norm.exprs[,1:3]), y = rowMeans(norm.exprs[,4:6]),
        pch=20, xlab = "Control", ylab = "Treated",
        main="Scatter plot: control vs treated")

abline(0,1, col = "blue")
abline(1,1, col = "red")
abline(-1,1, col = "red")
```







Finding differentially expressed genes

In order to perform the statistical analysis and find the differentially expressed genes we are using the limma package. The workflow of limma for microarray analysis is well described in their documentation.

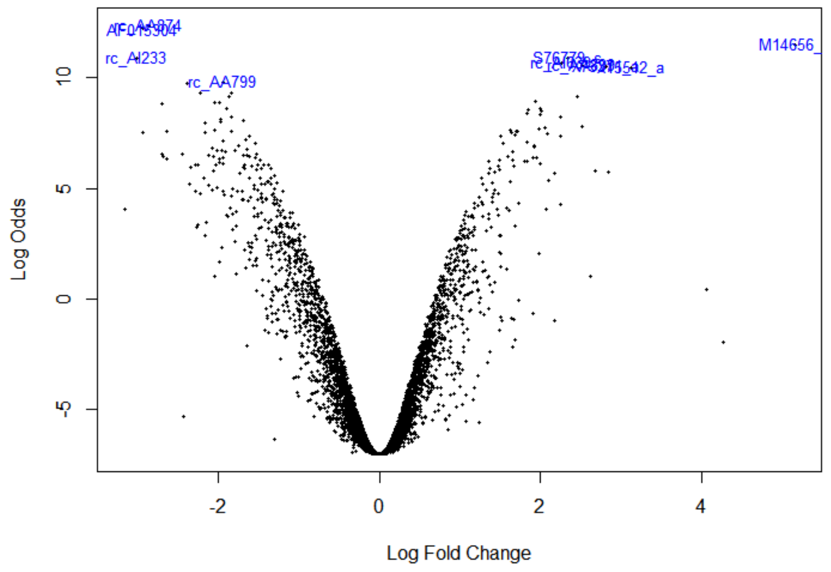
You need to create a design matrix (defining the conditions of each microarrays), then create a contrast matrix (with the comparisons of interest), fit a linear model, fit the contrasts to the linear model and calculate statistics.

```
library(limma)
# Create design matrix
design <- model.matrix(~ 0+factor(c(1,1,1,2,2,2)))
colnames(design)<- c("Control", "Treated")

# Create contrasts
contrast.matrix <- makeContrasts(Treated-Control, levels=design)

# Fit linear model to the data
fit <- lmFit(norm.exprs, design)

# Fit contrasts to linear model
fit.2 <- contrasts.fit(fit, contrast.matrix)

# Compute moderated t-statistics
fit.2 <- eBayes(fit.2)
```

Take some time to explore the design and contrasts matrix and to read the documentation of the functions you applied. You might also want to check the limma documentation and think how would you prepare your design and contrast matrix if you had more than one condition.

Then you can visually see the differentially expressed genes by displaying a volcano plot. A volcano plot is performed plotting fold change vs. a measurement of statistical significance, typically the P-value. However, the volcanoplot() function included in limma plots log(fold change) vs B-value. The B-value is a measurement of the probability of a gene to be differentially expressed. A B-value of 1.5 is equivalent to a exp(1.5)/(1+exp(1.5)) = 0.818 = 81.8% probability of a gene to be differentially expressed. Additionally the volcanoplot() function provides the option to highlight a given number of top genes.

```
volcanoplot(fit.2, coef=1, highlight = 10, names=rownames(norm.exprs))
```



A list of the differentially expressed genes can be obtained using the topTable() function. This function allows to filter by P-value and $\log_2$(FC), to select the maximum number of genes to be included in the table (select “Inf” for infinite for all the genes) and to sort them.

```
DEG<-topTable(fit.2, coef=1, number = Inf, adjust.method = "BH", p.value = 
0.01, lfc = 0.5, sort.by = "logFC")
```

To write these result into a CSV file that you can open in Excel you can use the function write.csv().

```
write.csv(DEG, file="DEG.csv", row.names=T)
```

The most interesting features of this table include:

a) The log2(FC): Fold change between two conditions in logarithmic values. If you indicated in your contrast matrix “Treated-Control”. Then positive values would be overexpressed and negative values would be underexpressed in the treated rats.
b) AveExpr: This is the average expression of this gene in all the microarrays
c) Adj.P.value: Here we applied Benjamini–Hochberg procedure (“BH”), so in this case it is equivalent to the false discovery rate.
d) B: B-value, explained previously.

Annotation

For annotation, you will use the libraries “annaffy” and “rgu34a.db”. You knew you need the “rgu34a.db” data package for annotation when you typed attributes(Affy.Data)$annotation.

```
library(annaffy)
library(rgu34a.db)
```

The use of annaffy is very simple. You just have to provide the sample IDs you want to annotate, the information you want to retrieve and the data package that contains the annotation data. The sample IDs we want to annotate available can be known using aaf.handler(). We want to annotate the 10 genes with higher log (FC) from the differentially expressed genes table. As we already sorted the genes by log(FC), we just have to select the first 10 names of the row names.

```
annot.columns <- aaf.handler()[c(1:3,7,11:12)]

anntable <- aafTableAnn(probeids = rownames(DEG)[1:10],
        chip = "rgu34a.db", colnames = annot.columns)
```

Then the annotation table can be easily saved into an HTML file.

```
saveHTML(anntable, "DEG_rat.html", title = "DE genes, rat experiment")
```