

Gabriella Rustici, Audrey Kauffmann (05/03/2009)

## Microarray data analysis using Bioconductor

### 1 Introduction

During this practical session, we will give an overview of the fundamental steps in the analysis of microarray data. We will first assess the quality of raw data downloaded from ArrayExpress. Data normalization algorithm will then be applied and the quality metrics re-run on the normalized data, giving the opportunity to compare the quality reports before and after normalization. Then, we will identify the differentially expressed genes with a linear model approach, visualize the processed data with clustering and perform a Gene Set Enrichment analysis.

The following R packages are required for this practical: ArrayExpress, arrayQualityMetrics, affy, limma, GSEABase, moe430a.db, genefilter, KEGG.db

### 2 Importing the data from ArrayExpress

The package ArrayExpress contains the function **ArrayExpress** which downloads a dataset from the ArrayExpress repository and converts it into a Bioconductor object.

```
> library("ArrayExpress")
> AEset = ArrayExpress("E-MEXP-886")
```

To avoid repeating this step, we can save the object in the local directory and use it later by using the function **load**.

```
> save(AEset, file = "AEset.RData")
```

E-MEXP-886 is an Affymetrix experiment, thus, the object AEset is an **AffyBatch** which is the standard Bioconductor object for Affymetrix data. The function class allows finding out which is the class of an object.

```
> class(AEset)
```

By using the function **ArrayExpress**, we have created an object that contains the expression values and the sample annotations. The sample annotation is obtained from the SDRF file from the database and is stored in the phenoData slot of the created object. The function pData gives access to the phenoData slot of an object.

```
> AEset
> colnames(pData(AEset))
```

### 3 Pre-processing

The main pre-processing steps are the quality assessment, the log transformation, the background correction, the normalization and the summarization of the probesets.

#### 3.1 Quality assessment before normalization

The package arrayQualityMetrics contains the function **arrayQualityMetrics** which is called with the following arguments:

- **expressionset**: is an object of class ExpressionSet, AffyBatch, BeadLevelList, NChannelSet, RGList, MAList, marrayRaw or marrayNorm.
- **outdir**: is the directory in which the result files are created.
- **force**: if TRUE, if outdir already exists, the files of the report contained in it will be overwritten.
- **do.logtransform**: if TRUE, the data are log transformed before the analysis.
- **split.plots**: if the number of studied arrays is more than 50 it is advised to define a number of experiments to represent on the density plots.
- **intgroup**: name of the column of the phenoData to be used to draw a color side bar next to the heatmap.

By calling the **arrayQualityMetrics** function, we will create a HTML report with several diagnostics plots.

To feed the argument **intgroup**, we can choose a relevant column from the sample annotation. In the SDRF files; the column names starting with "Factor" are the ones designing the biological groups of a dataset. We can extract these column names using the function **grep**.

```
> fac = colnames(pData(AEset))[grep("Factor", colnames(pData(AEset)))]
> fac
```

In this particular example, there is only one factor, the 'genotype'. We can now run the quality report with the genotype taken into account.

```
> library("arrayQualityMetrics")
> arrayQualityMetrics(expressionset = AEset, outdir = "QArav",
+ force = FALSE, do.logtransform = TRUE, intgroup = fac)
```

A report named **QMreport.html** is produced in the subdirectory **QArav**. The report contains a series of diagnostic plots as .png images. Each .png is linked to corresponding .pdf files that provide high quality images for publication purposes. All the images are saved in the subdirectory **QArav**.

## 3.2 Understanding the quality metrics report

The first entry in the report is a summary table. Arrays identified as having a potential problem or as being outliers are marked by an asterisk. Outliers are detected on the MA-plot, spatial distributions of the features' intensities, boxplot, heatmap, RLE and NUSE (Fig. 1).

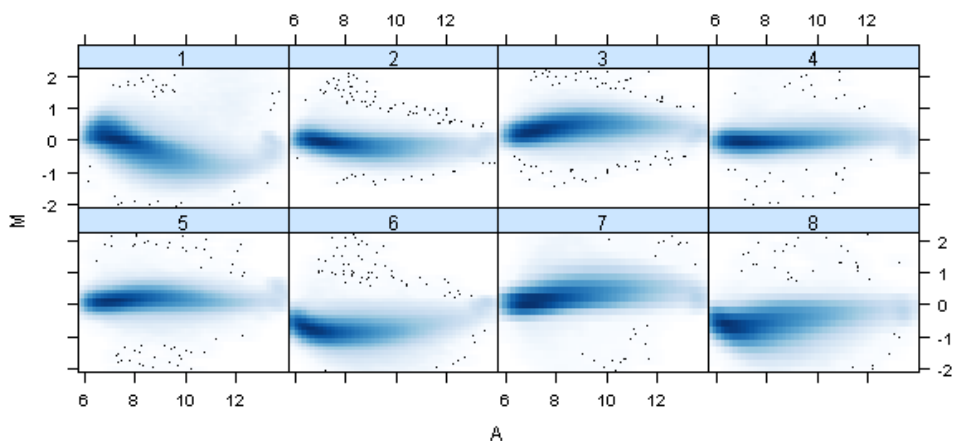
Array #	Array Name	MA-plot	Spatial distribution	Boxplots/Density plots	Heatmap	RLE	NUSE
1	E-MEXP-886-raw-cel-1227302827.cel						*
2	E-MEXP-886-raw-cel-1227302800.cel						
3	E-MEXP-886-raw-cel-1227302845.cel		*				
4	E-MEXP-886-raw-cel-1227302836.cel						
5	E-MEXP-886-raw-cel-1227302791.cel						
6	E-MEXP-886-raw-cel-1227302809.cel	*		*	*		
7	E-MEXP-886-raw-cel-1227302867.cel						
8	E-MEXP-886-raw-cel-1227302889.cel	*		*			
9	E-MEXP-886-raw-cel-1227302818.cel		*				
10	E-MEXP-886-raw-cel-1227302782.cel						

**Fig. 1: Summary table for the quality report on E-MEXP-886 raw data. Many of the issues will be resolved after normalization.**

The report is then subdivided in the following sessions:

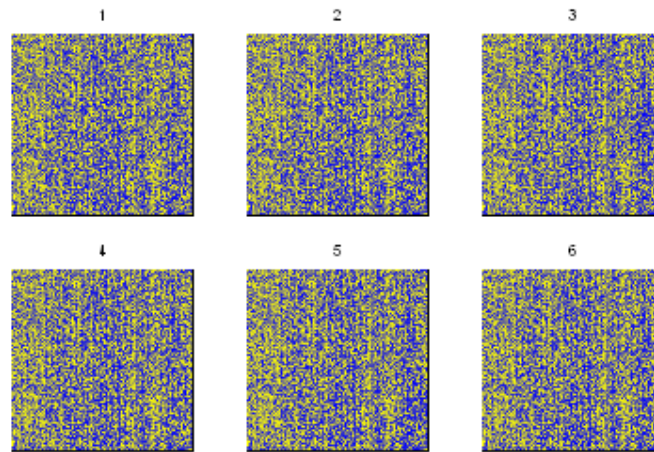
- Individual array quality
- Homogeneity between arrays
- Between array comparison
- Variance mean dependency
- Affymetrix specific plots

In the 'Individual array quality' session you will find the MA plots, which allow the evaluation of the dependence between the intensity levels and the distribution of the ratios (Fig. 2). For two-colour arrays, a probe's  $M$ -value is the log-ratio of the two intensities (y axis) and the  $A$ -value is the mean of their logarithms (x axis). In the case of one colour arrays, the  $M$ -value is computed by dividing the intensity by the median intensity of the same probe across all arrays.



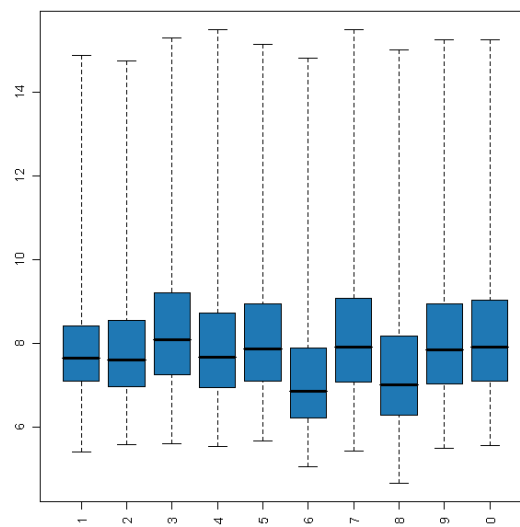
**Fig. 2: MA plots.  $M$  and  $A$  are defined as :  $M = \log_2(I1) - \log_2(I2)$  and  $A = 1/2 (\log_2(I1) + \log_2(I2))$  where  $I1$  is the intensity of the array studied and  $I2$  is the intensity of a "pseudo"-array, which have the median values of all the arrays. Typically, we expect the mass of the distribution in an MA plot to be concentrated along the  $M = 0$  axis, and there should be no trend in the mean of  $M$  as a function of  $A$ .**

In the same session, a false colour representation of each array's spatial distribution of feature intensities (Fig. 3) helps in identifying spatial effects that may be caused by, for example, gradients in the hybridization chamber, air bubbles or printing problems.

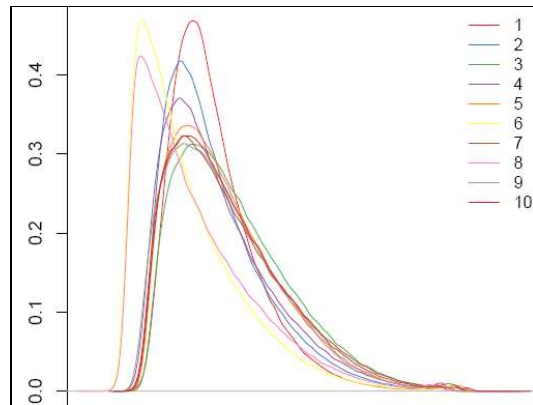


**Fig. 3: 'Spatial distribution of feature intensities' plots**

To assess the homogeneity between the arrays, boxplots of the  $\log_2$  intensities (Fig. 4) and density estimate plot (Fig. 5) are presented in the 'Homogeneity between arrays' session.

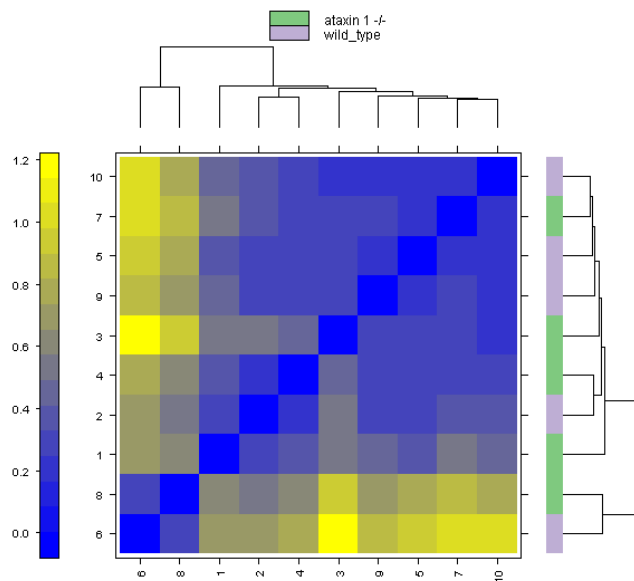


**Fig. 4: Boxplots of the  $\log_2$ (Intensities). Each box corresponds to one array. If the arrays are homogeneous, the boxes should have similar width and y position.**



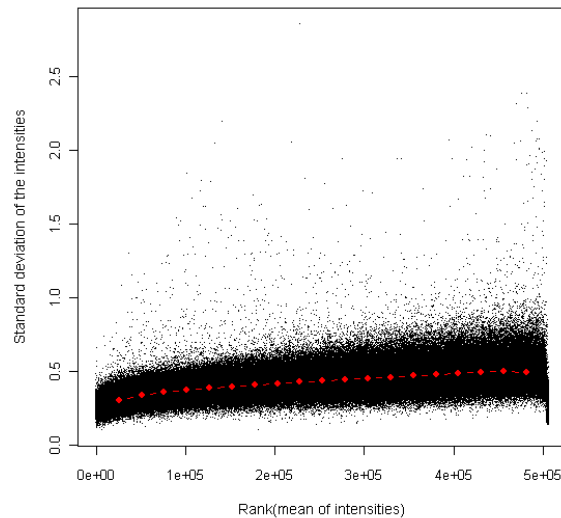
**Fig. 5: Density estimate plot. Arrays whose distributions are very different from the others should be considered for possible problems. A small bump on the right tail of the graph could be indicative of saturation effects.**

The 'Between array comparison' session shows a heatmap of between array distances, computed as the mean absolute difference of the M-value for each pair of arrays. The dendrogram can serve to check if the experiments cluster in accordance with the sample classes; if the clustering is not biologically meaningful at this stage, it should not be regarded as a problem since the data is still un-normalized.



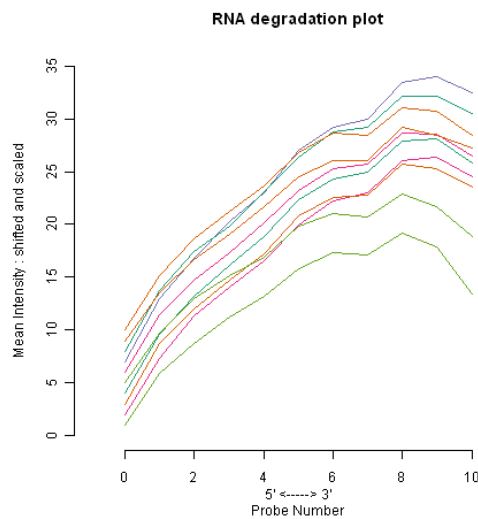
**Fig. 6: Between array comparison. This graph shows a false color heatmap representation of between arrays distances, computed as the median absolute difference of the M-value for each pair of arrays. Arrays whose distance matrix entries are way different give cause for suspicion. The dendrogram on this plot also can serve to check if, without any probe filtering, the arrays cluster accordingly to a biological meaning.**

The variance mean dependency plot (Fig. 7) shows, for each feature, the empirical standard deviation of the intensities of all the arrays (y axis) versus the rank of the mean of intensities of the arrays (x axis).

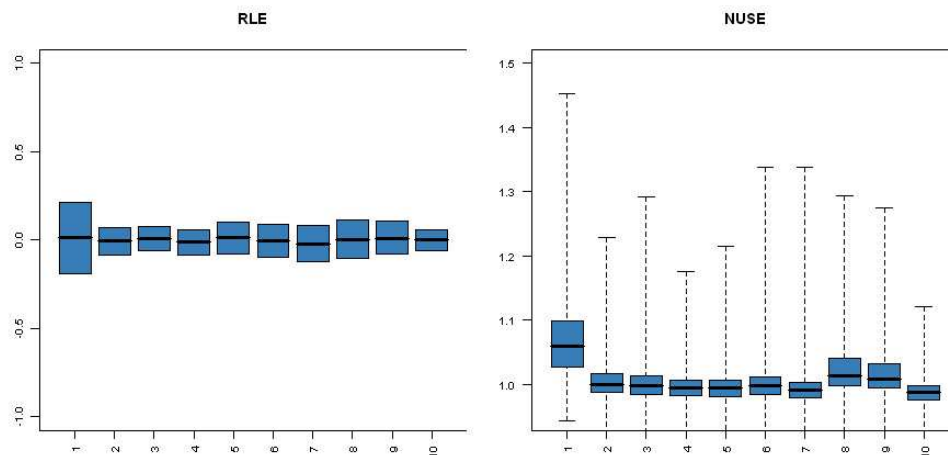


**Fig. 7: Variance mean dependency plot.** The red dots, connected by lines, show the running median of the standard deviation. Normally, higher intensities have higher variance. When the curve is really skewed, VSN normalization is recommended. After VSN, the curve should be approximately horizontal, showing no substantial trend.

The last session of the report contains four plots evaluating Affymetrix-specific metrics, if the input object is an **AffyBatch**: the RNA degradation plot (Fig. 8) from the **affy** package, the relative log expression (RLE) and the normalized unscaled standard error (NUSE) boxplots (Fig. 9) from the **affyPLM** package and the QC stat plot from the **simpleaffy** package (Fig. 10). A density distribution of the log<sub>2</sub> intensities grouped by the matching of the probes is also provided (Fig. 11).



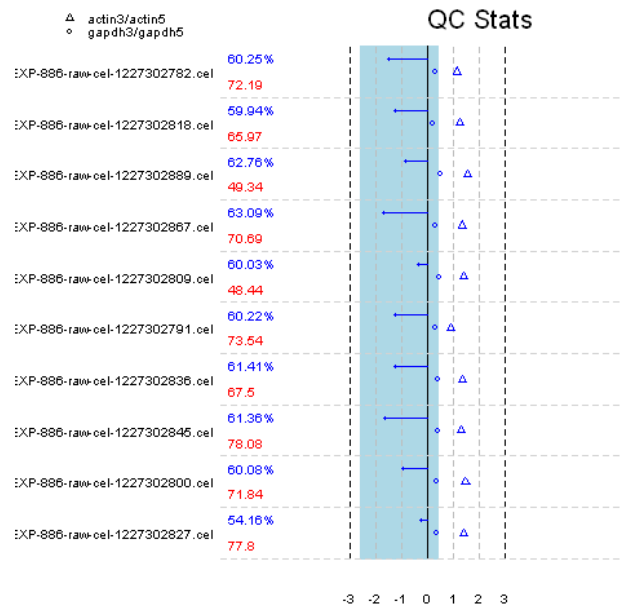
**Fig. 8: RNA degradation plot.** Each curve corresponds to a single chip and visualizes the chip-averaged dependency between probe intensity and probe position. Since RNA degradation characteristically starts from the 5' end, one expects that 5' end probes show lower intensities than 3' end probes. The trend should be consistent across arrays.



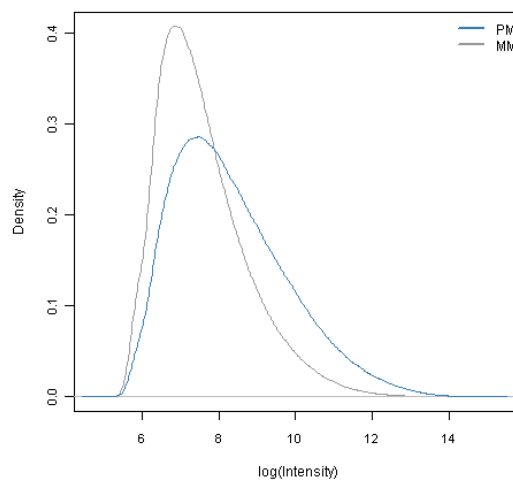
**Fig. 9: RLE and NUSE boxplots.**

The relative log expression (RLE) plot compares the expression levels on each chip to an artificial median of all chips in the experiment. Often, it is reasonable to assume that despite biological variability, the majority of genes do not change their expression across experimental conditions. RLE boxplots are then expected to be centered on zero and to show a small inter-quartile range. Deviating boxplots often indicate problematic chips.

The normalized unscaled standard error (NUSE) plot visualizes the chip-wise distribution of standard error estimates obtained for each gene on each array when performing the robust multi-chip probe-level fit. Standardization accounts for the fact that variability might considerably differ between genes and assures that the median standard error across arrays is 1 for each gene. Aberrant chips can be easily identified by their increased median residuum. As rule of thumb, medians higher than 1.05 are alarming.



**Fig. 10: QC stat plot.** Each row shows the (i) % present and average background, (ii) scale factors and (iii) GAPDH /  $\beta$ -actin ratios for an individual chip. (i) % present and average background will vary according to the samples being processed, and Affymetrix suggests simply that they should be similar. (ii) The blue stripe in the image represents the range where scale factors are within 3-fold of the mean for all chips. Scale factors are plotted as a line from the centre line of the image. A line to the left corresponds to a down-scaling, to the right, to an up-scaling. If any scale factors fall outside this '3-fold region', they are coloured red, otherwise they are blue. (iii) GAPDH and  $\beta$ -actin 3'/5' ratios are plotted as circles and triangles respectively. Affymetrix states that  $\beta$ -actin should be below 3 and GAPDH around 1. Any that fall outside these thresholds (1.25 for GAPDH) are coloured red; the rest are blue. In short, everything in the figure should be blue - red highlights a problem!



**Fig. 11: PM and MM density distribution plot.** Blue, density estimate of intensities of perfect match probes (PM) and gray the mismatch probes (MM). Since MM probes have poorer

*hybridization than PM probes, we expect that the PM curve should be shifted to the right of the MM curve.*

### 3.3 Normalization

Some arrays are highlighted as outliers in the report. It is recommended to try different types of normalization to see if these arrays are still outliers.

As an example here, we will only perform RMA (Robust Multi-Array Average expression measure) normalization. This method includes the background correction, the normalization and the summarization steps.

```
> library("affy")
> rAEset = rma(AEset)
```

### 3.4 Quality assessment after normalization

As the RMA normalization summarizes the probesets, the object rAEset is now an **ExpressionSet** which is a standard object for one colour arrays.

```
> arrayQualityMetrics(expressionset = rAEset, outdir = "QAnorm",
+ force = FALSE, intgroup = fac)
```

A report named **QMreport.html** is produced in the subdirectory **QAnorm**.

Open up the report after normalization. In the new file all plots are present except: (i) the 'spatial distribution of feature intensities' plots, since the summarization of probe sets is done during normalization, and (ii) the Affymetrix specific plots.

The first array (E-MEXP-886-raw-cel-1227302827.cel) is still identified as an outlier after normalization. This is indicated by:

- a wider spread in the MA plot,
- a slightly different boxplot distribution,
- a density distribution quite dissimilar from the other arrays, and
- very different distance matrix entries in the heatmap.

No problem is present in the 'Variance mean dependency' graph.

To avoid introducing noise and loosing statistical power, we will remove array #1 from the dataset before proceeding with the analysis.

```
> cAEset = rma(AEset[, -1])
Background correcting
Normalizing
Calculating Expression
```

## 4 Differential Expression

To identify the differentially expressed genes, we will use a moderated t-test. What we aim to achieve is a sensible ranking of genes based on evidence for differential expression.

For this, we will use the **limma** package. The **limma** approach incorporates average differential expression and a measure of variability similar to that used in t-statistics (i.e. standard error).

However the gene-wise standard errors are modified.

## 4.1 Data preparation

We first need to define the groups we want to use to fit the model. The names of the groups have to start with a letter, contain one word only and be without special character. In this case we will use the genotype as discriminating factor.

```
> library("limma")
> groups = pData(cAeset)[, fac]
> groups[groups == "wildfitype"] = "WT"
> groups[groups == "ataxin 1 -/-"] = "KO"
> f = factor(groups)
```

**f** is a factor containing the groups WT and KO. We will use this factor to create the model matrix that will be used to fit a linear model.

## 4.2 Moderated t-test

We first need to create a design matrix, which provides a representation of the different RNA targets that have been hybridized to the arrays.

```
> design = model.matrix(~f)
> colnames(design) = c("WTvsRef", "KOvsWT")
> design
  WTvsRef KOvsWT
1  1  1
2  1  0
3  1  0
4  1  1
5  1  1
6  1  0
7  1  0
8  1  1
9  1  1
```

We can now estimate the fold change and standard errors by fitting a linear model (using the **lmFit** function) for each gene given the groups of arrays and then compute a moderated t-statistics of differential expression by applying empirical Bayes shrinkage of the standard errors towards a common value (using the **eBayes** function).

```
> fit = lmFit(cAeset, design)
> fit2 = eBayes(fit)
```

To extract a table of the top-ranked genes from a linear model fit we first use **topTable** and then we subset its output. The **topTable** function is also used to perform hypothesis testing and adjust the p-values for multiple testing (see **topTable** arguments).

```
> results = topTable(fit2, coef = "KOvsWT", adjust = "BH", number = nrow(cAeset))
> topgenes = results[results[, "P.Value"] < 0.001, ]
```

A number of summary statistics are presented by **topTable**. Take a look at the **colnames** of the **results** object

```
> colnames(results)
```

```
[1] "ID" "logFC" "AveExpr" "t" "P.Value" "adj.P.Val" "B"
```

The following statistics are available:

- **logFC**: log2-fold change between two or more experimental conditions
- **AveExpr**: average log2-expression level of that gene across all arrays (or channels) in the experiment
- **t**: moderated t-statistic
- **P.Value**: associated p-value
- **adj.P.Val**: associated p-value after adjustment for multiple testing
- **B**: B-statistic or log-odds that the gene is differentially expressed

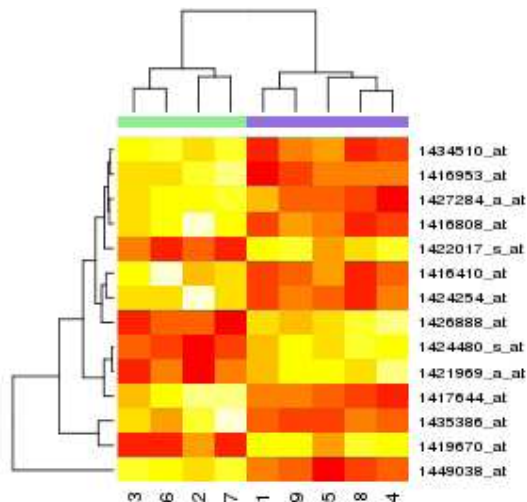
### 4.3 Visualisation

We can draw a heatmap of the arrays using the significant genes selected thanks to the moderated t-test to check that the samples cluster as expected (Fig. 12).

```
> m = exprs(cAeSet[topgenes[, "ID"], ])
> colnames(m) = 1:9
```

**m** is a matrix containing the normalized expression values for the top-ranked genes.

```
> colours = c("lightgreen", "mediumpurple")
> col = colours[f]
> heatmap(m, ColSideColors = col, margin = c(5, 8))
```



*Fig. 12: Heatmap of the 14 top-ranked genes in the 9 samples. Notice how the arrays cluster accordingly to a biological meaning.*

## 5 Gene Set Enrichment analysis

The library **GSEABase** is dedicated to Gene Set Enrichment Analysis (GSEA). GSEA is a computational method that determines whether an a priori defined set of genes shows statistically significant, concordant differences between two biological states (e.g. genotypes).

We first need to load the **GSEABase** library and the **annotation package** corresponding to our object. Then, we can collect the KEGG annotation and create an incidence matrix with the probes IDs within all the KEGG pathways.

```
> library("GSEABase")
> annotation(cAEset)
> library("moe430a.db")
> gsc = GeneSetCollection(cAEset, setType = KEGGCollection())
> Am = incidence(gsc)
> dim(Am)
```

We can subset our object so that only the probes belonging to a pathway are kept. And then we can compute a simple t-test only on those probes.

```
> nsF = cAEset[colnames(Am), ]
> library("genefilter")
> rtt = rowttests(nsF, fac)
> rttStat = rtt$statistic
```

We are interested in the pathways in which at least ten probes are present.

```
> selectedRows = (rowSums(Am) > 10)
> Am2 = Am[selectedRows, ]
```

We compute the t-test per pathway and adjust the t-values by the size of the pathway.

```
> tA = as.vector(Am2 %*% rttStat)
> tAadj = tA/sqrt(rowSums(Am2))
> names(tA) = names(tAadj) = rownames(Am2)
```

We can retrieve the pathway with the highest difference between the KO for the ataxin gene and the WT.

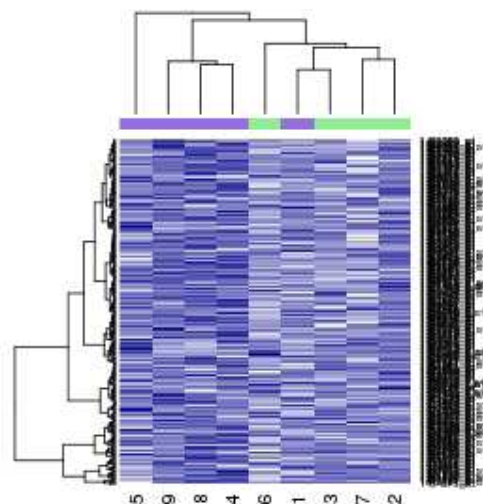
```
> library(KEGG.db)
> smPW = which(tAadj == min(tAadj))
> pwName = KEGGPATHID2NAME[[names(smPW)]]
> pwName
[1] "Neuroactive ligand-receptor interaction"
```

The E-MEXP-886 experiment description tells us that "*Ataxin 1 (Atxn1) is a protein of unknown function associated with cerebellar neurodegeneration in spinocerebellar ataxia type1 (SCA1). SCA1 is caused by an expanded polyglutamine within Atxn1 by gain-of-function mechanisms. Lack of Atxn1 in mice triggers motor deficits in the absence of neurodegeneration or apparent neuropathological abnormalities. We extracted RNA from cerebellum of Atxn1-null mice and 5 WT. Cerebellar gene expression profiles at 15 weeks of age were generated using Affymetrix MOE430A*

arrays. Identifying the molecular pathways regulated by *Atxn1* can provide insights into the early molecular mechanisms underlying neuronal dysfunction".

Interestingly, the most enriched KEGG pathway is "Neuroactive ligand-receptor interaction". To end, we can easily represent a heatmap of the pathway in the arrays using the function `KEGG2heatmap` (Fig. 13).

```
> KEGG2heatmap(names(smPW), nsF, "moe430a", col = colorRampPalette(c("white",
+ "darkblue"))(256), ColSideColors = col, margin = c(5, 8))
```



**Fig. 13: Heatmap of the significant genes belonging to the Neuroactive ligand-receptor interaction KEGG pathway.**

## 6 Session info

```
> sessionInfo()
```

R version 2.8.1 (2008-12-22)

Base packages: base, datasets, graphics, grDevices, grid, methods, splines, stats, tools, utils

Other packages: affy~1.20.2, affyPLM~1.18.0, annotate~1.20.1, AnnotationDbi~1.4.2, ArrayExpress~1.2.2, arrayQualityMetrics~1.8.1, beadarray~1.10.0, Biobase~2.2.1, DBI~0.2-4, gcrma~2.14.1, genefilter~1.22.0, geneplotter~1.20.0, graph~1.20.0, GSEABase~1.4.0, hwriter~0.93, KEGG.db~2.2.5, lattice~0.17-17, latticeExtra~0.5-4, limma~2.16.4, marray~1.20.0, matchprobes~1.14.1, moe430a.db~2.2.5, moe430acdf~2.3.0, preprocessCore~1.4.0, RColorBrewer~1.0-2, RSQLite~0.7-1, simpleaffy~2.18.0, sma~0.5.15, survival~2.34-1, vsn~3.8.0, XML~1.94-0.1, xtable~1.5-4

Loaded via a namespace (and not attached): affyio~1.10.1, cluster~1.11.11, KernSmooth~2.22-22