GEO Database: GSE46268 vitamin D

Jean-Yves Sgro

May 9, 2017

Table of Contents

[1. Introduction 2](#_Toc482108941)

[2. Prerequisites 2](#_Toc482108942)

[2.1. CRAN packages 2](#_Toc482108943)

[2.2. Bioconductor modules 3](#_Toc482108944)

[3. knitr 3](#_Toc482108945)

[3.1. Create a new project 3](#_Toc482108946)

[3.2. R Markdown script file 3](#_Toc482108947)

[3.3. Caching 4](#_Toc482108948)

[4. The GEO database 5](#_Toc482108949)

[5. Explore the GEO page 6](#_Toc482108950)

[6. Analysis based on GEO2R script 6](#_Toc482108951)

[6.1. Load the Bioconductor libraries: 7](#_Toc482108952)

[6.2. Command getGEO() and R lists 7](#_Toc482108953)

[6.3. Load the dataset 7](#_Toc482108954)

[6.4. Explore the new dataset structure 10](#_Toc482108955)

[6.5. Samples: "phenotypic" data 10](#_Toc482108956)

[6.6. Sample labels 11](#_Toc482108957)

[6.7. Making groups 11](#_Toc482108958)

[6.8. Log transform 12](#_Toc482108959)

[6.9. Proceed with analysis 14](#_Toc482108960)

[6.10. Update annotations 16](#_Toc482108961)

[6.11. Write output results table into a file 19](#_Toc482108962)

[7. Box plots 19](#_Toc482108963)

[7.1. The complete GEO2R code: 19](#_Toc482108964)

[7.2. Continuation 20](#_Toc482108965)

[8. GEO2R ends 22](#_Toc482108966)

[9. Adding to GEO2R results 22](#_Toc482108967)

[9.1. Simple clustering 22](#_Toc482108968)

[9.2. Principal Component Analysis 23](#_Toc482108969)

[9.3. Heatmaps 28](#_Toc482108970)

[9.4. MA Plot 36](#_Toc482108971)

[10. Online practical exercises for microarray data analysis 38](#_Toc482108972)

[11. Session info 38](#_Toc482108973)

[12. References 39](#_Toc482108974)

# Introduction

This exercise is meant to try knitr (Xie 2016, Xie (2015), Xie (2014)) with a published dataset (Wheelwright et al. 2014) on the GEO database (Barrett et al. 2013).

The title of the dataset is:

**Gene expression profile of human monocytes stimulated with all-trans retinoic acid (ATRA) or 1,25a-dihydroxyvitamin D3 (1,25D3)**

and is available at the GEO web site at <http://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE46268>

Data was in the published paper (Wheelwright et al. 2014):

Wheelwright M, Kim EW, Inkeles MS, De Leon A *et al.* All-trans retinoic acid-triggered antimicrobial activity against Mycobacterium tuberculosis is dependent on NPC2. J Immunol 2014 Mar 1;192(5):2280-90. PMID: [24501203](http://www.ncbi.nlm.nih.gov/pubmed/24501203)

# Prerequisites

## CRAN packages

The following R packages are used: knitr and other CRAN packages will be installed if necessary. Make sure that the "Install Dependencies" button is checked. You can also use line commands such as:

install.packages("knitr", repos="http://cran.us.r-project.org")  
install.packages("rmarkdown", repos="http://cran.us.r-project.org")

## Bioconductor modules

The following BioConductor modules are necessary. To install them run the following code manually if necessary:

source("http://bioconductor.org/biocLite.R")  
biocLite("GEOquery")  
biocLite("limma")  
biocLite("Biobase")  
biocLite("affy")

# knitr

## Create a new project

An **RStudio** project is a set-up that uses a specific directory to contain the data that we want to use for analysis.

To set-up a new project follow do the following in **RStudio**:

1. Click the File menu button, then New Project.
2. Click New Directory.
3. Click Empty Project.
4. Type in the name of the directory to store your project, *e.g.* VitaminD.
5. For now don't check “Create a git repository” (should be unselected by default)
6. Click the Create Project button.

The project will be saved in the default directory: /Users/YourName on the Mac unless you navigate to e.g. the Desktop to more easily find the it later.

## R Markdown script file

You can start a new R Markdown file with the following menu cascade:

**File > New File > R Markdown...**

Within the file there will be some examples data.

***Remove*** everything but keep the "header" that is held between 2 lines with dashes --- as they serve for the final output.

### R Markdown

For this exercise you can use very simple R Markdown for separating paragraphs with the # sign, for example:

# Heading 1  
## Heading 2  
### Heading 3

The rest of the text can be simply plain text.

If you need to review R Markdown syntax you can do so from one of the previous exercises [Using\_RStudio\_II.html](https://dl.dropboxusercontent.com/u/26372912/ipython/RStudio_01/Using_RStudio_II.html) (short URL: <http://bit.ly/21tdUYv> ; or check the complete notes from that session at <http://bit.ly/1TDrmJB>)

For inserting R code use the green button with the white **C** and an orange arrow. Or you can write the necessary tick marks by hand. The code "chunk" can have a ***unique*** optional name label:

```{r optional\_name\_label }  
# This code block starts with 3 backticks and will end the same way.  
# Place the R code within these boundaries.  
```

## Caching

Using knitr for the exercise allows to use the "**cache**" method which is best used with large data. For these microarray data it saves a minute or two, but for larger datasets it can be a substential time saver.

Here we'll set the cache globally so we don't have to repeat the caching request for each code chunk.

To engage the caching add the following code within your .Rmd project file, preferably at the top below the header:

```{r global\_options\_settings, include=TRUE, echo=FALSE }  
# Global options:  
opts\_chunk$set(warning=FALSE, message=FALSE, comment="", cache=TRUE)  
```

These global settings make the following changes:

* warning=FALSE and message=FALSE suppress in the final output any warnings or messages that usually appear in red on the regular R console. These can take a lot of space, be unsightly and are not necessary in the final report.
* comment="" removes the ## mark after all output. If you liked that feature, then you can omit this setting.
* cache=TRUE is the one that is of interest for large datasets... As you work on your project, if a file, a plot or a dataframe does not need to be changed when you make changes to your file, then the cached data will be used and can save a lot of time.
* Within the {r} bracket global\_options\_settings is an optional name for this code "chunk".
* include=TRUE means that we want the code to be included/activated.
* echo=FALSE means that we don't want anything related to this code to appear within the final report.

There are more caching options or methods detailed at <http://yihui.name/knitr/demo/cache/>

# The GEO database

The Gene Expression Omnibus (GEO) is a public repository that archives and freely distributes microarray, next-generation sequencing, and other forms of high-throughput functional genomic data submitted by the scientific community.

Further information about the database can be found at <http://www.ncbi.nlm.nih.gov/geo/info/>

A **general overview** of the database can be found at <http://www.ncbi.nlm.nih.gov/geo/info/>

The data is available in various forms and formats. In summary the various names for data files are summarized in the fillowing table (from <http://bit.ly/1LCJOj9> ; Aug 11, 2015 archive: bit.ly/1p2N65D)

|  |  |
| --- | --- |
| Data type | Description |
| **GEO Platform (GPL)** | These files describe a particular type of microarray. They are annotation files. |
| **GEO Sample (GSM)** | Files that contain all the data from the use of a single chip. For each gene there will be multiple scores including the main one, held in the VALUE column. |
| **GEO Series (GSE)** | Lists of GSM files that together form a single experiment. |
| **GEO Dataset (GDS)** | These are curated files that hold a summarized combination of a GSE file and its GSM files. They contain normalized expression levels for each gene from each sample (i.e. just the VALUE field from the GSM file). |

Today we'll use an experiment on vitamin D stored in a **GSE** file containing multiple samples. Within the database, each of the samples also has an individual **GSM** entry.

# Explore the GEO page

1. Go to the dataset link [GSE46268](http://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE46268)
2. Scroll down the page and click the **[+]More** at the Samples(12) entry
3. Note the name of the samples and their treatments. We'll find this information again later
4. Note the link named [Analyse with GEO2R](http://www.ncbi.nlm.nih.gov/geo/geo2r/?acc=GSE46268). **The R code for this exercise is derived from this option.** Check the "R Script" tab to see the code on the GEO web page.
5. You can read information about this service on the [About GEO2R](http://www.ncbi.nlm.nih.gov/geo/info/geo2r.html) page
6. You can also watch the 4 minutes YouTube video titled [**GEO2R: Analyze GEO Data**](http://youtu.be/EUPmGWS8ik0) on that same page.

# Analysis based on GEO2R script

***Caveat***: this exercise is meant to explore how the GEO database system of GEO2R conducts the computation. The authors of the paper have used other software to draw conclusions on signigicant genes and there is no attempt at this level to reproduce the results of the paper.

## Load the Bioconductor libraries:

library(GEOquery)  
library(Biobase)  
library(limma)  
library(affy)

## Command getGEO() and R lists

The command getGEO() is a function from the packages GEOquery (S. Davis and Meltzer 2007) that can download data directly from the GEO database <http://www.ncbi.nlm.nih.gov/geo/> .

The help command ?getGEO provides a description that stating that *It directs the download (if no filename is specified) and parsing of a GEO SOFT format file into an R data structure[..]*

However, the default command as specified in the **Usage** section is:

getGEO(GEO = NULL, filename = NULL, destdir = tempdir(), GSElimits=NULL,  
GSEMatrix=TRUE,AnnotGPL=FALSE,getGPL=TRUE)

The GSEMatrix=TRUE option is therefore the default and will open the Matrix format and ***NOT*** the SOFT format as implied by the definition. The result will be a different type of R object data structure.

The data exist in various file formats on the GEO database.

|  |  |
| --- | --- |
| Format name | Format |
| **SOFT** | Simple Omnibus Format in Text. |
| **MINiML** | (MIAME Notation in Markup Language - XML format |
| **Matrix** | spreadsheet containing the final, normalized values that are comparable across rows and Samples |

The SOFT format contains both data and annotations and therefore the files are larger. The Matrix format is similar but without the annotation.

## Load the dataset

gset <- getGEO("GSE46268", GSEMatrix =TRUE)

While the format is deemed a "matrix" the gset object created is in fact a list that contains one set of experiment and is therefore a list with one element!

We can see that with this code:

class(gset)

[1] "list"

length(gset)

[1] 1

names(gset)

[1] "GSE46268\_series\_matrix.txt.gz"

Note that the name could have been obtained with the attributes attr() function calling on the list item "names".

attr(gset,"names")

We therefore conclude that **for now** gset is a list with one element that was derived from file *GSE46268\_series\_matrix.txt.gz*.

This result will help explain how the code continues and to better understand that let's make a quick *a parte* into R lists.

Let's create a list L containing 2 objects, for example a vector of numbers vn and a vector of characters vc:

# Create list L  
L <- list(vn=c(2,3,5), vc=c("sun", "moons"))  
# Print list L  
L

$vn  
[1] 2 3 5  
  
$vc  
[1] "sun" "moons"

# Print first item and class of list L  
L[1]

$vn  
[1] 2 3 5

class(L[1])

[1] "list"

# Print content of first item of list L and class  
L[[1]]

[1] 2 3 5

class(L[[1]])

[1] "numeric"

In a similar way, the code continues by checking the length of the gset object which is still a ***list***. This conditional statement checks if the length of the list is greater than 1 and saves the result in "index" variable idx and checks for the pattern GPL570 which is the name in **GEO** for the type of microarray that was used in the experiment.

The gset object is then overwritten with the experimental data contained within the 1rst object of the list (or more if more than 1 have been detected) :

if (length(gset) > 1) idx <- grep("GPL570", attr(gset, "names")) else idx <- 1  
gset <- gset[[idx]]

We can now check again the class and length of gset since this has changed it:

class(gset)

[1] "ExpressionSet"  
attr(,"package")  
[1] "Biobase"

length(gset)

[1] 1

slotNames(gset)

[1] "experimentData" "assayData" "phenoData"   
[4] "featureData" "annotation" "protocolData"   
[7] ".\_\_classVersion\_\_"

The data structure ExpressionSet is a kind of data frame that contains the complete information about the experiment. The names() command not longer applies as gset is no longer a list but the command slotNames() applies to dataframe and can be used.

## Explore the new dataset structure

### Dataset dimensions:

dim(gset)

Features Samples   
 54675 12

Features are the number of "genes" on the array and we see that there are . We can also see that there are 12 samples.

### Dataset structure

Try the following command to get hints from the data structure

str(gset)

## Samples: "phenotypic" data

**"phenotypic" data is the list of samples with their associated attributes and their treatment**

The following command will list all attributes:

pData(phenoData(gset))

We can get the dimensions of the data table with the command:

dim(pData(phenoData(gset)))

[1] 12 38

That will tell us that there are **12** rows, corresponding to the number of samples, and there are **38** columns in the table.

We can list the name of the columns in the table with the command:

colnames(pData(phenoData(gset)))

And we'll print only the first 6: We can list the name of the columns in the table with the command:

colnames(pData(phenoData(gset)))[1:6]

[1] "title" "geo\_accession" "status"   
[4] "submission\_date" "last\_update\_date" "type"

Here is a modified command showing only two columns from the attributes table that seem to be most informative:

pData(phenoData(gset))[ , c(12,13)]

characteristics\_ch1.2 characteristics\_ch1.3  
GSM1127890 individual: donor 1 treatment: control  
GSM1127891 individual: donor 2 treatment: control  
GSM1127892 individual: donor 3 treatment: control  
GSM1127893 individual: donor 4 treatment: control  
GSM1127894 individual: donor 1 treatment: all-trans retinoic acid  
GSM1127895 individual: donor 2 treatment: all-trans retinoic acid  
GSM1127896 individual: donor 3 treatment: all-trans retinoic acid  
GSM1127897 individual: donor 4 treatment: all-trans retinoic acid  
GSM1127898 individual: donor 1 treatment: 1,25a-dihydroxyvitamin D3  
GSM1127899 individual: donor 2 treatment: 1,25a-dihydroxyvitamin D3  
GSM1127900 individual: donor 3 treatment: 1,25a-dihydroxyvitamin D3  
GSM1127901 individual: donor 4 treatment: 1,25a-dihydroxyvitamin D3

## Sample labels

# make proper column names to match toptable   
fvarLabels(gset) <- make.names(fvarLabels(gset))

The function make.names() is from the base pakcage and its descrition is to *make syntactically valid names out of character vectors*.

## Making groups

Below is code to help extract the name of treatment. This is not part of the original GEO2R code.

# NOTE:  
# The following code is added to the GEO2R code to automatically extract   
# the name of the treatments from column 13 of the phenoData  
tr <- levels(unique(pData(phenoData(gset))[13])[,1])  
tr1 <- gsub("treatment: ", "", tr[1])  
tr2 <- gsub("treatment: ", "", tr[2])  
tr3 <- gsub("treatment: ", "", tr[3])  
# The variables tr1, tr2 and tr3 are used wihthin   
# the knitr text file to name the treatments in the  
# paragraph below

The names of treatments are within the 13th column of the phenoData information and are saved within object tr. The text **"treatment: "** is removed by *gobal substitution* (command gsub) to an empty string ("") leaving the treatment name available.

*Note*:In the text below this table, the .RMD file used to create this document used this string to write the name of the treatment without any need for manual copy/paste.

|  |  |
| --- | --- |
| Object name | Extracted named treatment |
| tr | treatment: 1,25a-dihydroxyvitamin D3, treatment: all-trans retinoic acid, treatment: control |
| tr1 | 1,25a-dihydroxyvitamin D3 |
| tr2 | all-trans retinoic acid |
| tr3 | control |

Groups were created using the GEO2R web interface which resulted in the following vectors to be created to distinguished three groups labeled G0, G1 and G2 which, based on the phenotypic data we explored above would correspond to treatments named ***1,25a-dihydroxyvitamin D3***, ***all-trans retinoic acid***, and ***control***.

# group names for all samples  
sml <- c("G0","G0","G0","G0","G1","G1","G1","G1","G2","G2","G2","G2");

This means that the software has idendified 3 groups of 4 that make up our samples! It is not surprising as this set-up had to be done by hand within the web interface.

## Log transform

This is an important step for better statistical evaluation of microarray data.

First the observed expression values are extracted by the function exprs() from the Biobase package and stored into object ex taking care of removing NA values (na.rm=T). Negative values (for which a log cannot be calculated) are replaced with NaN and at the end the original expression values are replaced by their log: exprs(gset) <- log2(ex).

# log2 transform  
ex <- exprs(gset)  
qx <- as.numeric(quantile(ex, c(0., 0.25, 0.5, 0.75, 0.99, 1.0), na.rm=T))  
LogC <- (qx[5] > 100) ||  
 (qx[6]-qx[1] > 50 && qx[2] > 0) ||  
 (qx[2] > 0 && qx[2] < 1 && qx[4] > 1 && qx[4] < 2)  
if (LogC) { ex[which(ex <= 0)] <- NaN  
 exprs(gset) <- log2(ex) }

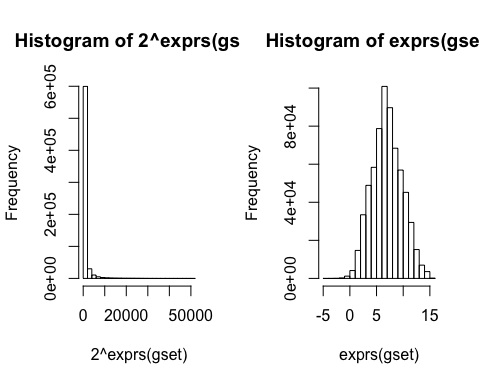
There is some internal calculation on quantiles, and the replacement of negative or zero values (ex <= 0) by NaN before taking the log2 of the intensities.

The final result is an ***"expression set"*** which contains the **log2** transform of the intensity values.

The log transformation is a necessary step to change the data into a form to which statistics can be better applied: it gives the data a more "Normal" distribution as can be seen by a histogram.

We can compare a histogram of values before and after log2 transformation:

# Split graphics in 2 sections  
par(mfrow=c(1,2))  
# Before log2 - since it's already been log2 trasnform   
# we use 2^ to compute back the original value:  
hist(2^exprs(gset), breaks=25)  
#After log2  
hist(exprs(gset), breaks=25)



# Return graphics to default  
par(mfrow=c(1,1))

## Proceed with analysis

**Set up the data and proceed with analysis**

This uses a method from the limma package (Ritchie et al. 2015).

The following code "chunk" prepares an experimental **design matrix** where samples on rows are represented within a matrix by either 1 or 0 and columns are the names of the sample groups G0, G1 and G2 obtained with the as.factor() and levels() functions.

The makeContrasts() function creates pair-wise comparison between groups.

The limma package then uses Baysian statistical methods (eBayes()) to fit the data to the model.

The data is adjusted for False Discovery Rate (adjust="fdr") and sorted by column B which represents the *log-odds that the gene is differentially expressed*.

Finally, the top 250 "genes" are extracted into a table object tT (the default export number is 10, see ?topTable.)

FDR: **False Discovery Rate**[[1]](#footnote-1) The False discovery rate (FDR) is one way of conceptualizing the rate of type I errors in null hypothesis testing when conducting multiple comparisons. FDR-controlling procedures are designed to control the expected proportion of rejected null hypotheses that were incorrect rejections ("false discoveries").(Benjamini and Hochberg 1995)

# set up the data and proceed with analysis  
fl <- as.factor(sml)  
gset$description <- fl  
design <- model.matrix(~ description + 0, gset)  
colnames(design) <- levels(fl)  
fit <- lmFit(gset, design)  
cont.matrix <- makeContrasts(G2-G0, G1-G0, G2-G1, levels=design)  
fit2 <- contrasts.fit(fit, cont.matrix)  
fit2 <- eBayes(fit2, 0.01)  
tT <- topTable(fit2, adjust="fdr", sort.by="B", number=250)

Side exercise: check content of objects from code above  
- sml  
- fl  
- design before and after colnames() change  
- cont.matrix

The best 250 are kept in the final table. This number is arbitrary, and if left unspecified would provide a table with all results.

We can count the number of columns of the Table:

dim(tT)

[1] 250 23

We can see the name of the columns:

colnames(tT)

[1] "ID" "GB\_ACC"   
 [3] "SPOT\_ID" "Species.Scientific.Name"   
 [5] "Annotation.Date" "Sequence.Type"   
 [7] "Sequence.Source" "Target.Description"   
 [9] "Representative.Public.ID" "Gene.Title"   
[11] "Gene.Symbol" "ENTREZ\_GENE\_ID"   
[13] "RefSeq.Transcript.ID" "Gene.Ontology.Biological.Process"  
[15] "Gene.Ontology.Cellular.Component" "Gene.Ontology.Molecular.Function"  
[17] "G2...G0" "G1...G0"   
[19] "G2...G1" "AveExpr"   
[21] "F" "P.Value"   
[23] "adj.P.Val"

We can view the first 5 witin a table with the command:

View(tT[1:5,])

We can also view the first **top 3** and specify only a few of the most interesting columns (for printing space sake.)  
Note the name of the column headers.

head(tT)[1:3,c(2,11:12,17:20, 22:23)]

GB\_ACC Gene.Symbol ENTREZ\_GENE\_ID G2...G0 G1...G0  
226099\_at AI924426 ELL2 22936 2.7737814 -0.04501778  
206504\_at NM\_000782 CYP24A1 1591 9.4846618 -0.88549603  
212685\_s\_at AI608789 TBL2 26608 0.4468399 3.36551801  
 G2...G1 AveExpr P.Value adj.P.Val  
226099\_at 2.818799 11.341555 6.384254e-10 1.722549e-05  
206504\_at 10.370158 7.853788 6.519834e-10 1.722549e-05  
212685\_s\_at -2.918678 11.480685 9.451573e-10 1.722549e-05

## Update annotations

The annotations are the default options and can be updated with the latest NCBI annotations.

Each microarray has a "platform" ID code specific to that array. Here we find the code for the array with annotation(gset) which gives the result ***GPL570*** and this is passed onto the next command to download the corresponsing "platform" file from NCBI/GEO and stored within platf.

The operations that follow check the name of "column" information in tT and gset and "merge" into a final version based on the ***common*** column ID.

# load NCBI platform annotation  
gpl <- annotation(gset)  
platf <- getGEO(gpl, AnnotGPL=TRUE)  
ncbifd <- data.frame(attr(dataTable(platf), "table"))  
  
# replace original platform annotation  
tT <- tT[setdiff(colnames(tT), setdiff(fvarLabels(gset), "ID"))]  
tT <- merge(tT, ncbifd, by="ID")  
tT <- tT[order(tT$P.Value), ] # restore correct order

Side exercise:  
- check the output of commands colnames(tT) and fvarLabels(gset)  
- check the meaning of setdiff. (Hint: in package BiocGenerics)

With the updated annotations the number of columns has changed:

dim(tT)

[1] 250 28

THe new annotation has completly rearranged and changed many of the columns, including the addtion of Gene Ontology.  
We can see the name of the columns

colnames(tT)

[1] "ID" "G2...G0"   
 [3] "G1...G0" "G2...G1"   
 [5] "AveExpr" "F"   
 [7] "P.Value" "adj.P.Val"   
 [9] "Gene.title" "Gene.symbol"   
[11] "Gene.ID" "UniGene.title"   
[13] "UniGene.symbol" "UniGene.ID"   
[15] "Nucleotide.Title" "GI"   
[17] "GenBank.Accession" "Platform\_CLONEID"   
[19] "Platform\_ORF" "Platform\_SPOTID"   
[21] "Chromosome.location" "Chromosome.annotation"  
[23] "GO.Function" "GO.Process"   
[25] "GO.Component" "GO.Function.ID"   
[27] "GO.Process.ID" "GO.Component.ID"

The statistical columns are now at the begining of the table.

We can print the first 3 with some specific columns. Note that the row names are changed to a number, probably a ranking number.

head(tT)[1:3,c(1:10)]

ID G2...G0 G1...G0 G2...G1 AveExpr F  
202 226099\_at 2.7737814 -0.04501778 2.818799 11.341555 256.8290  
84 206504\_at 9.4846618 -0.88549603 10.370158 7.853788 255.8234  
122 212685\_s\_at 0.4468399 3.36551801 -2.918678 11.480685 238.6761  
 P.Value adj.P.Val  
202 6.384254e-10 1.722549e-05  
84 6.519834e-10 1.722549e-05  
122 9.451573e-10 1.722549e-05  
 Gene.title Gene.symbol  
202 elongation factor, RNA polymerase II, 2 ELL2  
84 cytochrome P450, family 24, subfamily A, polypeptide 1 CYP24A1  
122 transducin (beta)-like 2 TBL2

The file would be written in the current directory.

**The table can be made nicer with kable()**

kable() is aknitr function to render tables.

kable(head(tT)[1:3,c(1:10)])

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | ID | G2...G0 | G1...G0 | G2...G1 | AveExpr | F | P.Value | adj.P.Val | Gene.title | Gene.symbol |
| 202 | 226099\_at | 2.7737814 | -0.0450178 | 2.818799 | 11.341555 | 256.8290 | 0 | 1.72e-05 | elongation factor, RNA polymerase II, 2 | ELL2 |
| 84 | 206504\_at | 9.4846618 | -0.8854960 | 10.370158 | 7.853788 | 255.8234 | 0 | 1.72e-05 | cytochrome P450, family 24, subfamily A, polypeptide 1 | CYP24A1 |
| 122 | 212685\_s\_at | 0.4468399 | 3.3655180 | -2.918678 | 11.480685 | 238.6761 | 0 | 1.72e-05 | transducin (beta)-like 2 | TBL2 |

## Write output results table into a file

The code provided write.table(tT, file=stdout(), row.names=F, sep="\t") would print the 250 results onto the console as the given file name is stdout(). To write the complete list we would have to update the number= to that of the number of "genes" and provide a proper filename such as "results.txt" printed with quotes.

# Box plots

The next portion of the code is meant to create box plots. The code is redundant by downloading again, creating groups again etc. but that makes that chunk of code independent from the rest.  
Some slight difference can be noted for the ex object created here with a specification as to the ordering of the samples, so that samples are presented together by group.

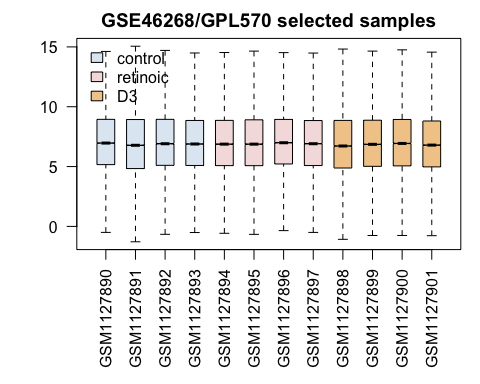
## The complete GEO2R code:

################################################################  
# Boxplot for selected GEO samples  
library(Biobase)  
library(GEOquery)  
  
# load series and platform data from GEO  
  
gset <- getGEO("GSE46268", GSEMatrix =TRUE)  
if (length(gset) > 1) idx <- grep("GPL570", attr(gset, "names")) else idx <- 1  
gset <- gset[[idx]]  
  
# group names for all samples in a series  
sml <- c("G0","G0","G0","G0","G1","G1","G1","G1","G2","G2","G2","G2")  
  
# order samples by group  
ex <- exprs(gset)[ , order(sml)]  
sml <- sml[order(sml)]  
fl <- as.factor(sml)  
labels <- c("control","retinoic","D3")  
  
# set parameters and draw the plot  
palette(c("#dfeaf4","#f4dfdf","#f2cb98", "#AABBCC"))  
dev.new(width=4+dim(gset)[[2]]/5, height=6)  
par(mar=c(2+round(max(nchar(sampleNames(gset)))/2),4,2,1))  
title <- paste ("GSE46268", '/', annotation(gset), " selected samples", sep ='')  
boxplot(ex, boxwex=0.6, notch=T, main=title, outline=FALSE, las=2, col=fl)  
legend("topleft", labels, fill=palette(), bty="n")

## Continuation

We can start where the first change occurs: with the updating of the ex object:

################################################################  
# Boxplot for selected GEO samples  
# (removed chunk)  
  
# order samples by group  
ex <- exprs(gset)[ , order(sml)]  
sml <- sml[order(sml)]  
fl <- as.factor(sml)  
labels <- c("control","retinoic","D3")  
  
# set parameters and draw the plot  
palette(c("#dfeaf4","#f4dfdf","#f2cb98", "#AABBCC"))  
# dev.new(width=4+dim(gset)[[2]]/5, height=6)  
par(mar=c(2+round(max(nchar(sampleNames(gset)))/2),4,2,1))  
title <- paste ("GSE46268", '/', annotation(gset), " selected samples", sep ='')  
boxplot(ex, boxwex=0.6, notch=T, main=title, outline=FALSE, las=2, col=fl)  
legend("topleft", labels, fill=palette(), bty="n")



The dev.new() options shows the plot in a new window on R or RStudio when running interactively but does **not** include it within an RMarkdown report.

The solution is to simply comment out (#) the dev.new() line so that the final plot is included in the final report.

Within the plot() command the option las=2 write the names of the samples as vertical labels on the x axis.

One important feature to look at in box-plots is the horizontal mark within the boxes that represents the median value. The box itself is made of the data from the 25th to the 75th percentile. If the values are similar than it is possible to use the data to compare the samples against each other.

Learn more about box plots at <https://en.wikipedia.org/wiki/Box_plot>

# GEO2R ends

The GEO2R R scripts only contains the code that we have explored. On the interactive web site, there are other possibilities, for example to show the expression values of specific genes across all samples.

Details of the GEO2R method can be found on the GEO web site at <http://www.ncbi.nlm.nih.gov/geo/info/geo2r.html>

# Adding to GEO2R results

There are many other calculations or plots that can be added to the GEO2R analysis. Here are a few examples.

We may use some of the objects defined above as well, for example groups names.

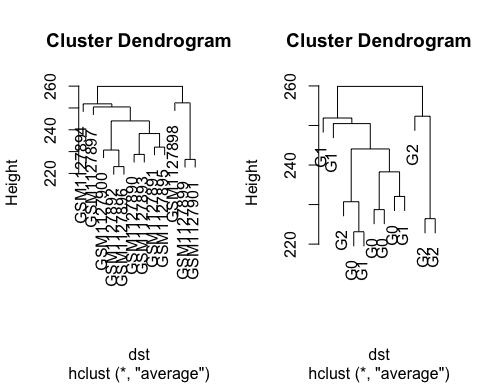
## Simple clustering

A simple cluster of samples can be obtained with:

# calculate a distance matrix between each sample (each array)  
dst <- dist(t(exprs(gset)))  
# Hierarchical cluster analysis on above distance matrix  
hh <- hclust(dst, method="average")

We can then plot the tree by sample name or by group name using the fl object created previously:

# We will plot both of them on the same plot  
par(mfrow=c(1,2))  
# plot default is by sample name  
plot(hh)  
# label sample by group  
plot(hh, label=fl)



par(mfrow=c(1,1))

Ideally all groups would cluster together, but it is common to find, as here, elements of a groups that cluster with anohter.

## Principal Component Analysis

The central idea of principal component analysis (PCA) is to reduce the dimensionality of a data set consisting of a large number of interrelated variables, while retaining as much as possible of the variation present in the data set. This is achieved by transforming to a new set of variables, the principal components (PCs), which are uncorrelated, and which are ordered so that the first ***few*** retain most of the variation present in all of the original variables[[2]](#footnote-2) (Jolliffe 2002).

PCA can be computed with the prcomp() function from the stats package installed by default in all R installation. There is therefore no package to install.

prcomp() will a principal components analysis on the complete set of 54675 genes or probes within the gene expression levels inside the dataset (exprs(gset)). Function t() transposes the data matrix. Results are stored in object PC subsequenlty analysed by the predict() function to compute values for each sample. Results are placed into object scores.

PC=prcomp(t(exprs(gset)))  
scores = predict(PC)

As stressed in the definition above the first ***few*** components are useful.

Exercise:  
\* Can you determine how many components were calculated?  
\* Explore the class and content of scores.  
(Hint: class() and dim(), print())  
Can you print only the values of the first 2 PC?

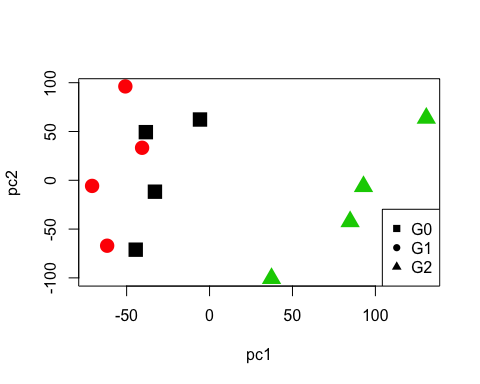
### PCA Plot

To make the plot code easier to follow it is best to prepare a few variables. Variable fl was defined previously as a factor for the groups and is used below to identify the shape of samples on the plot as well as the colors

# extract PC1 and PC2  
pc1 <- scores[ ,1]  
pc2 <- scores[ ,2]  
# Create a vector of number for choosing the plot symbol  
# We add 14 to reach the symbols that are filled  
shape <- as.numeric(fl) + 14

We can now make a plot for the first 2 principal components and then add a legend:

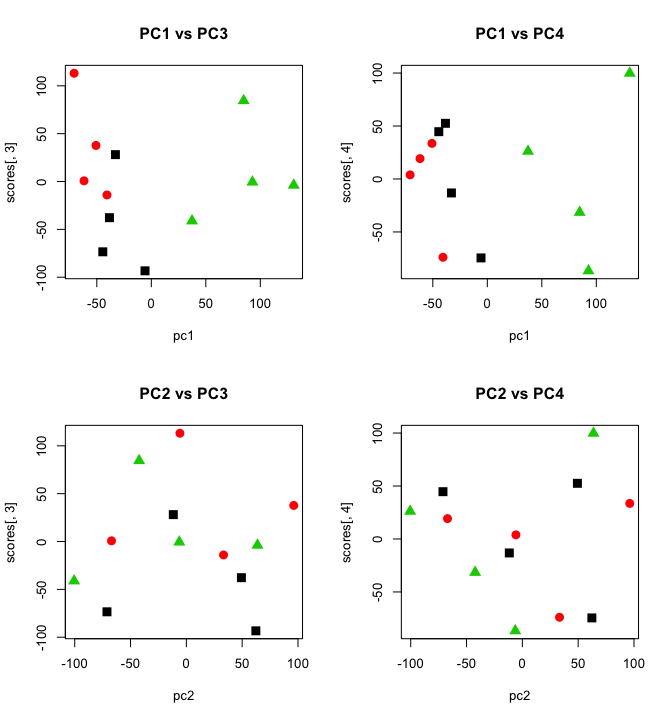
plot(pc1, pc2, col=fl, pch=shape, cex=2)  
# legend("topright", pch=unique(shape), paste(unique(fl)))  
legend("bottomright", pch=unique(shape), paste(unique(fl)))



cex=2 controls the size of the plotted shapes, here making them bigger.

The location of the legend can be controlled by a set of coordinates or predefined locations as used here. See ?legend for more details.

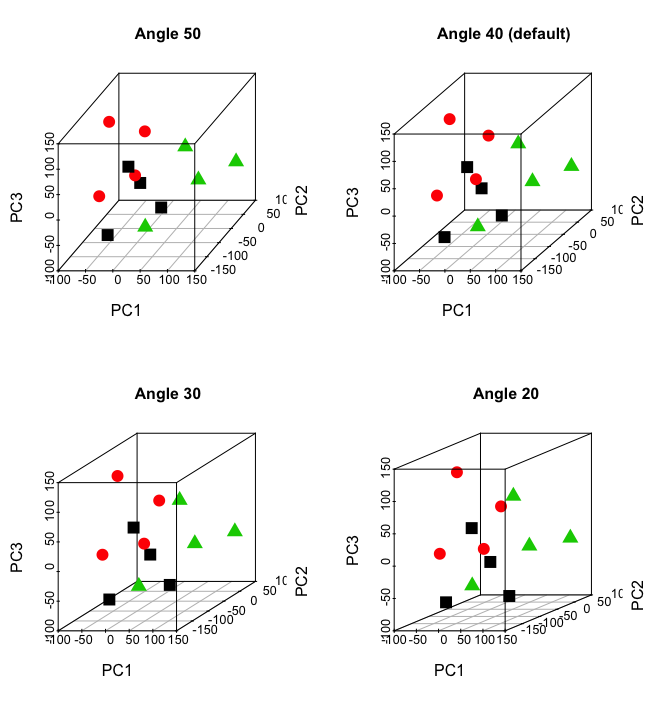
From this simple plot it seems that group G2 is more different than the other 2 groups.

Exercise:  
\* Make a similar plot for PC1 vs PC3  
\* Make a similar plot for PC1 vs PC4  
\* What happens if you use PC2 as horizontal axis?  
\* (Hints: it is not necessary to create pc objects, simply use scores[,3], scores[,4] etc.)  
You would get images similar to this: 

*Notes*: there are methods to plot this data in 3D with interactive (mouse) rotation with the package rgl and the plotlm3d function[[3]](#footnote-3) but it is also possible to use the CRAN package scatterplot3d for static plots:

# Install scatterplot3d if wanted  
install.packages("scatterplot3d", repos="http://cran.us.r-project.org")

library(scatterplot3d)  
#  
par(mfrow=c(2,2))  
  
# Angle 50   
scatterplot3d(scores[,1], scores[,2], scores[,3], xlab="PC1", ylab="PC2", zlab="PC3", pch=shape, color=as.numeric(fl), main="Angle 50", cex.symbols=2.0, angle=50)  
  
# Angle 40 (default)  
scatterplot3d(scores[,1], scores[,2], scores[,3], xlab="PC1", ylab="PC2", zlab="PC3", pch=shape, color=as.numeric(fl), main="Angle 40 (default)", cex.symbols=2.0, angle=40)  
  
# Angle 30   
scatterplot3d(scores[,1], scores[,2], scores[,3], xlab="PC1", ylab="PC2", zlab="PC3", pch=shape, color=as.numeric(fl), main="Angle 30", cex.symbols=2.0, angle=30)  
   
# Angle 20   
scatterplot3d(scores[,1], scores[,2], scores[,3], xlab="PC1", ylab="PC2", zlab="PC3", pch=shape, color=as.numeric(fl), main=" Angle 20", cex.symbols=2.0, angle=20)



par(mfrow=c(1,1))

## Heatmaps

Additional packages needed from both CRAN (for gplots) and Bioconductor (for RColorBrewer). Run the following installation code if necessary:

source("http://bioconductor.org/biocLite.R")  
biocLite("RColorBrewer")  
install.packages("gplots")

Load libraries:

library(RColorBrewer)  
library(gplots)

-->

The heatmap can be constructed for each "contrast" for which we calculated a differential expression with limma when we constructed the contrast matrix cont.matrix during the calculations with limma:

Contrasts  
Levels G2 - G0 G1 - G0 G2 - G1  
 G0 -1 -1 0  
 G1 0 1 -1  
 G2 1 0 1

We can "grab" each contrast name by selecting each element of the column names, for example colnames(cont.matrix)[2] yields **G1 - G0** which we can plot below with a false discovery rate (FDR) of 0.01.

The number of genes defined here is the number of "probesets" representing the genes onto the array.

The object completeTopTable will contain the differential expression table values for the selected contrast.

From this table we can select only those genes/probesets that match the chosen criterion that FDR should be less than the defined cut-off value, for example FDR < 0.01.

This creates the object selected which is a logical list of all the genes with TRUE or FALSE and can be used as a "selector" to select the same genes within the gset expression values that are then stored in object esetSel.

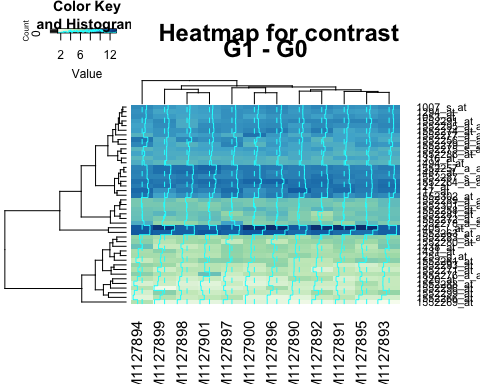
# Define FDR cut-off, typically 0.05 or 0.01  
FDR\_cutoff <- 0.01  
  
# Calculate the number of genes  
numGenes <- nrow(exprs(gset))  
  
# Extract a "contrast" from the contrast matrix  
contrast <- colnames(cont.matrix)[2]  
  
# Select the differential expression for this specific contrast  
# for all genes and sorted by columns "B" which represents   
# the log-odds that a gene is differentially expressed.  
completeTopTable <- topTable(fit2,coef=contrast, adjust="fdr", sort.by="B", number=numGenes)  
  
# Create a logical selector containing TRUE or FALSE  
# that defines if the gene meets the criterion about FDR  
selected <- completeTopTable$adj.P.Val < FDR\_cutoff  
  
# Create a subset of the expression set only for selected genes  
esetSel <- gset[selected]  
  
# Check some esetSel properties. Dimensions  
dim(esetSel)

Features Samples   
 43 12

# Calculate the number of genes that are selected. This number   
# should be the same as the number of "Features" above.  
sum(selected)

[1] 43

# Create a heatmap with heatmap.2 that allows more colors  
# color gradient to represent the expression values of gene   
# heatmap.2(exprs(esetSel))  
hmcol <- colorRampPalette(brewer.pal(9,"GnBu"))(100)  
heatmap.2(exprs(esetSel), col=hmcol, main=c(" Heatmap for contrast " , contrast ))



The color of the plotted genes depends on the palette chosen. By default the colors would be red-to-yellow but here we access one of the "green-blue" palettes from the RColorBrewer package. See ?brewer.pal for all the color palettes descriptions.

From the total number of genes (54675) there are **43** that are selected for the plot. The number of genes is the number of "probesets" onto the array.

The name of the Affymetrix probeset is listed on the right hand side of the heatmap. The function help (?heatmap.2) reveals that the annotation of the "genes" is by default the row names of the object plotted (rownames(x) for object x) and indeed:

rownames(esetSel)

[1] "1007\_s\_at" "1053\_at" "117\_at" "121\_at"   
 [5] "1255\_g\_at" "1294\_at" "1316\_at" "1320\_at"   
 [9] "1405\_i\_at" "1431\_at" "1438\_at" "1487\_at"   
[13] "1494\_f\_at" "1552256\_a\_at" "1552257\_a\_at" "1552258\_at"   
[17] "1552261\_at" "1552263\_at" "1552264\_a\_at" "1552266\_at"   
[21] "1552269\_at" "1552271\_at" "1552272\_a\_at" "1552274\_at"   
[25] "1552275\_s\_at" "1552276\_a\_at" "1552277\_a\_at" "1552278\_a\_at"  
[29] "1552279\_a\_at" "1552280\_at" "1552281\_at" "1552283\_s\_at"  
[33] "1552286\_at" "1552287\_s\_at" "1552288\_at" "1552289\_a\_at"  
[37] "1552291\_at" "1552293\_at" "1552295\_a\_at" "1552296\_at"   
[41] "1552299\_at" "1552301\_a\_at" "1552302\_at"

Therefore, more manipulation is necessary to obtain the name gene symbol rather than the probeset to be listed.

### Finding the gene symbols

First we have to wonder and verify the class of object esetSel which should be an "annotated data frame for expression sets".

class(esetSel)

[1] "ExpressionSet"  
attr(,"package")  
[1] "Biobase"

This is the main final type of data structure for many microarray experiment based on the Affymetrix platform. It contains the data and other annotations listed in various "slots."

slotNames(esetSel)

[1] "experimentData" "assayData" "phenoData"   
[4] "featureData" "annotation" "protocolData"   
[7] ".\_\_classVersion\_\_"

We have explored before how to look at, extract and modify the phenoData. For this subset, the complete phenotypic data information would have been transfered. You can verify that with the commands that we know, for example pData(esetSel).

The "slot" of interest now is featureData which contains the actual data. While we used a helper function pData() before, we can also access slot information with help of the @ nomenclature as shown below:

esetSel@featureData

An object of class 'AnnotatedDataFrame'  
 featureNames: 1007\_s\_at 1053\_at ... 1552302\_at (43 total)  
 varLabels: ID GB\_ACC ... Gene.Ontology.Molecular.Function (16  
 total)  
 varMetadata: Column Description labelDescription

However, this compact output is not yet very useful and we need a little more work to get to what we need. For example it would be nice to know the name of the various columns that make up this table of data:

colnames(esetSel@featureData)

[1] "ID" "GB\_ACC"   
 [3] "SPOT\_ID" "Species.Scientific.Name"   
 [5] "Annotation.Date" "Sequence.Type"   
 [7] "Sequence.Source" "Target.Description"   
 [9] "Representative.Public.ID" "Gene.Title"   
[11] "Gene.Symbol" "ENTREZ\_GENE\_ID"   
[13] "RefSeq.Transcript.ID" "Gene.Ontology.Biological.Process"  
[15] "Gene.Ontology.Cellular.Component" "Gene.Ontology.Molecular.Function"

We should now be able to access and list the gene symbols, and we use the $ annotation to extract these values:

esetSel@featureData$Gene.Symbol

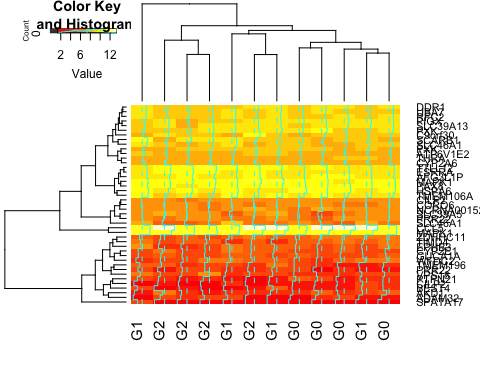
[1] DDR1 RFC2 HSPA6 PAX8 GUCA1A UBA7   
 [7] THRA PTPN21 CCL5 CYP2E1 EPHB3 ESRRA   
[13] CYP2A6 SCARB1 TTLL12 NCRNA00152 WFDC2 MAPK1   
[19] MAPK1 ADAM32 SPATA17 PRR22 PRR22 PXK   
[25] PXK VPS18 C9orf30 SLC46A1 SLC46A1 TIMD4   
[31] SLC39A5 ZDHHC11 ATP6V1E2 AFG3L1P CILP2 CILP2   
[37] PIGX TMEM196 SLC39A13 BEST4 AKD1 CORO6   
[43] TMEM106A   
21050 Levels: ADAM32 AFG3L1P AKD1 ALG10 ARMCX4 ATP6V1E2 BEST4 ... FAM86B1

Now that we know where these are and how to access them we should be able to change the annotation of the heatmap with these gene symbols:

heatmap.2(exprs(esetSel), labRow=esetSel@featureData$Gene.Symbol)

But we can take this opportunity to also change the label for the samples and use the group label instead by using the sml object containing the list of groups and passing this information with the labCol paramter:

heatmap.2(exprs(esetSel), labRow=esetSel@featureData$Gene.Symbol, labCol=sml)



Note that the heatmap colors are that of the default since we have not specified here the color command as previously using col=hmcol in the command.

We can finalize this version of the heatmap by adding one more optional parameter named ColSideColors to color a horizontal side-bar to distinguish the groups. This is an optional parameter.

We know that the groups are contained within the object sml, an object of class "character" that we need to transform into numeric values that can be factors. The as.numeric() function alone applied to sml would yield a new vector of only NA values. That is why we first apply the function as.factor() to force this output:

as.numeric(sml)

[1] NA NA NA NA NA NA NA NA NA NA NA NA

as.numeric(as.factor(sml))

[1] 1 1 1 1 2 2 2 2 3 3 3 3

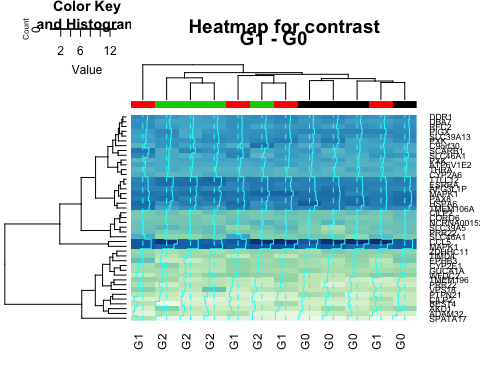
Now that we have a vector reprenting the groups as numbers, we need to choose a palette and pass these values. We can store this into a new object that we'll use the in final heatmap command:

# prepare vector of colors for heatmap  
colside <- palette(brewer.pal(8,"Dark2"))[as.numeric(as.factor(sml))]  
# print values  
colside

[1] "black" "black" "black" "black" "red" "red" "red"   
 [8] "red" "green3" "green3" "green3" "green3"

We can now make a final heatmap with all the additions we made, starting from a previous command:

heatmap.2(exprs(esetSel), col=hmcol, main=c(" Heatmap for contrast " , contrast ), labRow=esetSel@featureData$Gene.Symbol, labCol=sml, ColSideColors=colside)



### Exercise

Can you add color sides for the rows?

Can you plot the other contrasts?

## MA Plot

We have seen MA plots in the previous workshop.

An improved method, which is basically a scaled, 45 degree rotation of the R vs. G plot is an MA-plot.[2] The MA-plot is a plot of the distribution of the red/green intensity ratio ('M') plotted by the average intensity ('A'). M and A are defined by the following equations. Review: <https://en.wikipedia.org/wiki/MA_plot>

The MA plots were defined at a time where the experimental and control experiments were both on the same array and labelled with red and green dyes but the same principles and calculations can apply to simple arrays.

A is the average log-expression and is used as the horizontal axis.

M is the log fold-change and is used on the vertical axis.

Horizontal lines are often shown at -1 and 1 for and representing the **2-fold** change limit.

### Plot our selected genes

With the same selected genes we can also create an MA plot for the same contrast.

The simplest command would be plotMA(fit2[,1]) to plot the first contrast ([,1]) but we can make it a bit better and mark significant genes.

We prepare an object named status to define if the gene is "significant" and this information will be automatically be plotted (top left) together with the legend.

# Prepare the "status" parameter to distinguish  
# differentially expressed genes on the plot  
# Check the number of genes (probesets)  
number\_genes <- dim(exprs(gset))[1]  
# create a new object as long as the number of gens  
# Each position will contain "" (nothing) at first  
status <- character (length=number\_genes)  
# fill object with "not changing" at every position  
status <- rep ("not changing", number\_genes)  
# add a name attribute, in the form of gene number  
names (status) <- seq (1, number\_genes, 1)   
# change the value to "significant" for the selected genes  
status [selected] <- "significant"  
  
# Make the MA plot using the "status" option to show selected genes  
# The "values" option can separate options to be plotted  
# (Compare with the simpler plot command below.)  
# The plot symbols are chosen with the "pch" option.  
plotMA(fit2[,1], status=status, values=c("not changing", "significant"), col=c("blue","red"), pch=c(46,20))  
# Simpler plots  
# plotMA(fit2[,1]) # no colors  
# plotMA(fit2[,1], status=status, col=c("red","blue"))  
# Add text info  
text(x=12, y=9, labels=paste("FDR = ", FDR\_cutoff), col="black", font=2)  
# Add horizontal lines at +-log2(2) to mark 2X fold change  
abline(h=c(1,-1), col="green")



### Exercise

Can you plot the other contrasts?

Can you alter colors?

Can you change plot symbols?

# Online practical exercises for microarray data analysis

A document titled "*A Tutorial Review of Microarray Data Analysis*" provides statistical background on analysis and is accompanied by a series of exercises.

The document is available in PDF at <http://www.ub.edu/stat/docencia/bioinformatica/microarrays/ADM/slides/A_Tutorial_Review_of_Microarray_data_Analysis_17-06-08.pdf> (and is archived at: <http://bit.ly/1QAuuT9>)

The practical exercises page is at <http://www.ub.edu/stat/docencia/bioinformatica/microarrays/ADM/practicalExs.htm> (archived: <http://bit.ly/1Lbo91D>)

# Session info

sessionInfo()

R version 3.2.3 (2015-12-10)  
Platform: x86\_64-apple-darwin13.4.0 (64-bit)  
Running under: OS X 10.10.5 (Yosemite)  
  
locale:  
[1] en\_US.UTF-8/en\_US.UTF-8/en\_US.UTF-8/C/en\_US.UTF-8/en\_US.UTF-8  
  
attached base packages:  
[1] parallel stats graphics grDevices utils datasets methods   
[8] base   
  
other attached packages:  
[1] gplots\_2.17.0 RColorBrewer\_1.1-2 affy\_1.48.0   
[4] limma\_3.26.8 GEOquery\_2.36.0 Biobase\_2.30.0   
[7] BiocGenerics\_0.16.1 knitr\_1.12.3   
  
loaded via a namespace (and not attached):  
 [1] magrittr\_1.5 zlibbioc\_1.16.0 highr\_0.5.1   
 [4] stringr\_1.0.0 caTools\_1.17.1 tools\_3.2.3   
 [7] KernSmooth\_2.23-15 htmltools\_0.3 gtools\_3.5.0   
[10] yaml\_2.1.13 digest\_0.6.9 preprocessCore\_1.32.0  
[13] affyio\_1.40.0 formatR\_1.2.1 codetools\_0.2-14   
[16] bitops\_1.0-6 RCurl\_1.95-4.8 evaluate\_0.8   
[19] rmarkdown\_0.9.5 gdata\_2.17.0 stringi\_1.0-1   
[22] BiocInstaller\_1.20.1 XML\_3.98-1.4

# References

Barrett, T., S. E. Wilhite, P. Ledoux, C. Evangelista, I. F. Kim, M. Tomashevsky, K. A. Marshall, et al. 2013. “NCBI GEO: archive for functional genomics data sets–update.” *Nucleic Acids Res.* 41 (Database issue): D991–995.

Benjamini, Yoav, and Yosef Hochberg. 1995. “Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing.” *J. Roy. Statist. Soc. Ser. B* 57 (1): 289–300. [http://links.jstor.org/sici?sici=0035-9246(1995)57:1<289:CTFDRA>2.0.CO;2-E](http://links.jstor.org/sici?sici=0035-9246(1995)57:1%3c289:CTFDRA%3e2.0.CO;2-E).

Davis, Sean, and Paul Meltzer. 2007. “GEOquery: A Bridge Between the Gene Expression Omnibus (Geo) and Bioconductor.” *Bioinformatics* 14: 1846–7.

Jolliffe, I.T. 2002. *Principal Component Analysis*. 2nd ed. Boca Raton, Florida: Springer-Verlag New York. <http://www.springer.com/us/book/9780387954424>.

Ritchie, Matthew E, Belinda Phipson, Di Wu, Yifang Hu, Charity W Law, Wei Shi, and Gordon K Smyth. 2015. “limma Powers Differential Expression Analyses for RNA-Sequencing and Microarray Studies.” *Nucleic Acids Research* 43 (7): e47.

Wheelwright, M., E. W. Kim, M. S. Inkeles, A. De Leon, M. Pellegrini, S. R. Krutzik, and P. T. Liu. 2014. “All-trans retinoic acid-triggered antimicrobial activity against Mycobacterium tuberculosis is dependent on NPC2.” *J. Immunol.* 192 (5): 2280–90.

Xie, Yihui. 2014. “Knitr: A Comprehensive Tool for Reproducible Research in R.” In *Implementing Reproducible Computational Research*, edited by Victoria Stodden, Friedrich Leisch, and Roger D. Peng. Chapman; Hall/CRC. <http://www.crcpress.com/product/isbn/9781466561595>.

———. 2015. *Dynamic Documents with R and Knitr*. 2nd ed. Boca Raton, Florida: Chapman; Hall/CRC. <http://yihui.name/knitr/>.

———. 2016. *Knitr: A General-Purpose Package for Dynamic Report Generation in R*. <http://yihui.name/knitr/>.

1. <https://en.wikipedia.org/wiki/False_discovery_rate> [↑](#footnote-ref-1)
2. <http://www.springer.com/us/book/9780387954424> [↑](#footnote-ref-2)
3. <https://github.com/sr1919/R/blob/master/ext/protm3d.R> [↑](#footnote-ref-3)