

# Introduction to R for gene expression data analysis

---

Please read this document carefully. If already familiar with R syntax, you may skip the first section and move on to the two study cases. Throughout the document, blue text represents R commands (on lines starting with the '>' character) and output from the R console. This is provided as code examples for you to run during the tutorial, and observe the output.

## R basics

A detailed introduction to R can be found at <http://cran.r-project.org/doc/manuals/R-intro.html>

### Variables and data types:

- R is not a strongly typed programming language, so different value types can be stored in the same variable. Eg:

```
> a=2
```

```
> a
```

```
[1] 2
```

```
> a='2'
```

```
> a
```

```
[1] "2"
```

- Available basic data types: numeric (double, integer), logical, character, vector, matrix.
  - Vector examples:

```
> a=c(1:10)
```

```
> a
```

```
[1] 1 2 3 4 5 6 7 8 9 10
```

```
> a[a>7]
```

```
[1] 8 9 10
```

```
> a[3]
```

```
[1] 3
```

```
> a[-1]
```

```
[1] 2 3 4 5 6 7 8 9 10
```

```
> a[4:8]
```

```
[1] 4 5 6 7 8
```

```
> length(a)
```

```
[1] 10
```

```
> length(a)=3
```

```
> a
```

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

- Matrix examples:

Create a 3X3 matrix:

```
> a=matrix(1:9,3,3)
```

```
> a
```

```
      [,1] [,2] [,3]  
[1,]    1    4    7  
[2,]    2    5    8  
[3,]    3    6    9
```

Access a row in the matrix:

```
> a[1,]
```

```
[1] 1 4 7
```

Access a column:

```
> a[,1]
```

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

Access an element:

```
> a[2,2]
```

```
[1] 5
```

```
> a[2,2]=5.5
```

```
> a
```

```
      [,1] [,2] [,3]  
[1,]    1 4.0    7  
[2,]    2 5.5    8  
[3,]    3 6.0    9
```

Create a 2x2 matrix:

```
> b=array(2:3,1:2,dim=c(2,2))
```

```
> b
```

```
      [,1] [,2]
[1,]    2    2
[2,]    3    3
```

Access more elements in matrix a:

```
> a[b]
```

```
[1] 5.5 9.0
```

- More advanced data types: lists (heterogeneous vectors), data frames (heterogeneous matrices).

Create a list:

```
> l=list("2",3,7,c(4,2,3))
```

```
> l
```

```
[[1]]
```

```
[1] "2"
```

```
[[2]]
```

```
[1] 3
```

```
[[3]]
```

```
[1] 7
```

```
[[4]]
```

```
[1] 4 2 3
```

Access an element in the list:

```
> l[[4]]
```

```
[1] 4 2 3
```

Create a data frame:

```
> d=data.frame(c(1,2,3),c("d","f","t"))
```

```
> d
```

```
  c.1..2..3. c..d....f....t..
1           1           d
2           2           f
```

```
3          3          t
```

Access and element in the data.frame:

```
> d$c.1..2..3.
```

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

- Checking for type, type conversion: `is.numeric (double, integer)`, `is.vector`, `is.matrix`, `as.numeric`, `as.vector`, `as.matrix`. Eg:

```
> a='3.4'
```

```
> is.numeric(a)
```

```
[1] FALSE
```

```
> is.double(a)
```

```
[1] FALSE
```

```
> a=as.double(a)
```

```
> a
```

```
[1] 3.4
```

```
> is.double(a)
```

```
[1] TRUE
```

```
> is.integer(a)
```

```
[1] FALSE
```

```
> a=as.integer(a)
```

```
> a
```

```
[1] 3
```

**File input/output:** we will be using mostly data frames and matrices.

- `read.table()` function: allows reading from text file containing tabular information.

```
>data=read.table("fileIn.txt", header=TRUE)
```

- `write.table()` allows writing a data frame (can be a matrix) to a file.

```
>write.table(data, "fileOut.txt");
```

**Plotting:**

- Various plotting functions exist, and most Bioconductor packages extend these.
- `plot()` - high level function for plotting. E.g. scatter plots, line plots:

```
> x=c(3,5,2,7,4,9,2)
> plot(x)
> plot(x, type='l')
> y=c(4,2,1,7,9,6,2)
> plot(x,y)
```

- `hist()` - histogram

```
>hist(x)
```

- Labels: `xlab`, `ylab` - axis labels, `main` - axis title.

### R functions:

- Used to write reusable code.
- Syntax: `name <- function(arg_1, arg_2, ...) expression`

```
> add<-function(x,y) {
+ sum=x+y
+ sum
+ }
```

- Usage:

```
> add(2,3)
```

```
[1] 5
```

- If you prefer storing your functions or code in a file, you can run that file using

```
>source("fileName", echo=TRUE)
```

### R packages:

- The main advantage of R stands in the wide availability of free software packages for statistical analysis.
- Loading a package: `library(affy)`, displaying contents of a package : `library(help=affy)`.
- Packages for expression data analysis can be found on the Bioconductor page: <http://www.bioconductor.org/packages/2.6/bioc/> . These can be installed through the R interface menu (Packages->Install packages), after setting the correct repositories (Packages->Select repositories -> select all). Here, we will be using packages *affy* <http://www.bioconductor.org/packages/2.6/bioc/html/affy.html> and *limma*

<http://www.bioconductor.org/packages/2.6/bioc/html/limma.html> . You can find more details and manuals on these web pages.

### Datasets used in this tutorial (Yeast cell cycle):

- Dual-channel: Hasse dataset
- Single-channel: Pramila dataset, Spellman dataset
- Download archive and unpack : <http://www.computing.dcu.ie/~asirbu/R.zip>

## Dual channel microarray data - LIMMA package

### Pramila dataset

We will start with dataset Pramila. For this, we need to load the *limma* package, and change the working directory to that containing the raw data:

```
>library(limma)
```

In the R menu, go to File->Change dir and select the corresponding directory (after unpacking the downloaded archive, this should be Pramila). Please note that in this directory there is a text file called *targets.txt* that contains information on samples hybridised to each chip. This does not come with the raw data, but needs to be created by you. In our case, the data contain time conditioned samples hybridised against a reference.

### Loading raw data

- Reading in data from .gpr file (GenePix microarray image reader):

```
> targets=readTargets("targets.txt")
```

```
> targets$FileName
```

```
[1] "GSM81064.gpr" "GSM81065.gpr" "GSM81066.gpr" "GSM81067.gpr"
"GSM81068.gpr"
```

```
[6] "GSM81069.gpr" "GSM81070.gpr" "GSM81071.gpr" "GSM81072.gpr"
"GSM81073.gpr"
```

```
[11] "GSM81074.gpr" "GSM81075.gpr" "GSM81076.gpr"
```

```
> RG <- read.maimages(targets$FileName, source="genepix")
```

- Contents of RG:

```
> names(RG)
```

```
[1] "R"      "G"      "Rb"     "Gb"     "targets" "genes"  "source"
"printer"
```

R,G are the red and green channel values, Rb and Gb the background noise, genes contains information on chip probes. The can be displayed at the console:

```
> head(RG$genes)
```

	Block	Row	Column	ID	Name
1	1	1	1	YDR407C	TRS120
2	1	1	2	YDR180W	SCC2
3	1	1	3	YAR050W	FL01
4	1	1	4	YKL129C	MY03
5	1	1	5	YOR328W	PDR10
6	1	1	6	YJR138W	

## Visualisation

- Visualisation with MA plots allows plotting log ratios (R vs G) against average spot intensity.

```
> plotMA(RG,"GSM81069")
```

In the resulting plot, most genes need to cluster around 0 M value, which indicates that they are not differentially expressed, compared to the reference sample. Also, there should be no dependence of the M values on the intensity. In case these are not true, normalisation is required.

- Background intensities can be also visualised for each array (this helps identify arrays with large intensities):

```
> boxplot(log2(RG$Rb))
```

- The plot() function can be used to visualise the time series for the red or green channel for a specific gene (gene on position 100 in the example):

```
> plot(RG$G[100,],type='l')
```

- The plotDensities() function can be used to visualise red and green intensities (as density plots) on the different arrays:

## Normalisation

- Normalisation of microarray data starts with background adjustment, then performs within array normalisation to remove effects of average intensities on the log ratios. Several

different approaches for this exist, please see *limma* manual for details. Here, we will use *normexp* background correction and *loess* normalisation.

```
> RGbk <- backgroundCorrect(RG, method="normexp", offset=50)
```

```
> MA <- normalizeWithinArrays(RGbk, method="loess")
```

- We can now look at the corresponding MA plots for RGbk and MA, and observe how the dependence between M and A is reduced:

```
> plotMA(RGbk, "GSM81069")
```

```
> plotMA(MA, "GSM81069")
```

- Also, if we plot densities, we can see that now the two channels in each array overlap better than before loess normalisation:

```
> plotDensities(RGbk)
```

```
> plotDensities(MA)
```

- A further step is between array normalisation, which aims at removing differences between two arrays. This is useful when comparing different arrays or for modelling approaches. We can apply *quantile* normalisation for this.

```
> MA.b=normalizeBetweenArrays(MA, method="quantile")
```

- If we plot densities again, we can see that all arrays overlap significantly.

```
> plotDensities(MA.b)
```

## Spellman dataset

Several other types of files can be automatically read by function `read.maimages()`. To use these, the parameter 'source' needs to be changed into e.g. '*agilent*' or '*smd*', etc. In case the file format is not available, we can read from text files containing tabular data, by specifying the labels for the red and green channel, and for the background. We can do this for the Spellman dataset (change working directory to Spellman):

```
> targets=readTargets("targets.txt")
```

```
> targets$FileName
```

```
[1] "13.txt" "14.txt" "15.txt" "16.txt" "17.txt" "18.txt" "19.txt"  
"20.txt" "21.txt" "22.txt" "23.txt"
```

```
[12] "24.txt" "25.txt" "26.txt" "27.txt" "28.txt" "29.txt" "30.txt"
```

```
> RG <- read.maimages ( targets$FileName, columns = list ( R =
"ScanAlyze:RESULT.CH1I_MEAN", G = "ScanAlyze:RESULT.CH2I_MEAN", Rb =
"ScanAlyze:RESULT.CH1B_MEDIAN", Gb = "ScanAlyze:RESULT.CH2B_MEDIAN"))
```

- Having the data loaded, we can now proceed to the analysis as above. You should try and adapt the above commands to this new dataset.

## Affymetrix data - affy & limma packages

- Change the working directory to Hasse, and load the *affy* package:

```
> library(affy)
```

- Reading in the raw data:

```
> data= ReadAffy()
```

- Data visualisation:

Box plots for each array:

```
> boxplot(data)
```

Density graphs for each array:

```
>hist(data)
```

MA plots of each of the first 4 arrays against a median reference:

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

```
> MAplot(data[,1:4], method="smoothScatter")
```

MA plots of first three arrays (against each other):

```
> MAplot(data[,1:3], method="smoothScatter", pairs=TRUE)
```

- Data normalisation and computation of expression values:

RMA

```
>rma=rma(Data)
```

MAS5

```
>mas5=mas5(Data)
```

Custom normalisation, using *expresso* to control all steps:

```
> custom = expresso(Data, normalize.method = "qspline",
bgcorrect.method="rma", pmcorrect.method="pmonly",
summary.method="liwong")
```

For details on available methods for each normalisation step, you can see *affy* Bioconductor web page . There are several commands that also display this information:

```
> bgcorrect.methods()
```

```
[1] "bg.correct" "mas"          "none"          "rma"
```

```
> express.summary.stat.methods()
```

```
[1] "avgdiff"      "liwong"       "mas"          "medianpolish"
"playerout"
```

```
> pmcorrect.methods()
```

```
[1] "mas"          "methods"      "pmonly"       "subtractmm"
```

```
> normalize.AffyBatch.methods()
```

```
[1] "constant"      "contrasts"    "invariantset" "loess"
"methods"        "qspline"
```

```
[7] "quantiles"      "quantiles.robust"
```

- Normalised data visualisation:

Same as above for `hist` and `boxplot`, just using `exprs(rma)`, `exprs(mas5)` or `exprs(custom)` instead of `data`. For example, to obtain the *boxplot* of expression values for each array after RMA normalisation (which can be compared to that before normalisation):

```
>boxplot( exprs(rma))
```

- Writing expression values to file:

```
>write.exprs(rma,file="normalisedData.txt")
```

## Remarks

This document represents a quick introduction to microarray data processing using R, and includes pre-processing only. However, the *limma* package offers tools for modelling and differential expression analysis, which are detailed in the user guide on the *limma* web page <http://www.bioconductor.org/packages/2.6/bioc/html/limma.html> . However, keep in mind that, in general, in order to perform differential expression analysis, datasets need to contain replicates. A good source for microarray data is GEO database (<http://www.ncbi.nlm.nih.gov/geo/> ), where you can look for different types of experiments (not necessarily time series, as these examples here), and try to perform differential expression analysis.

Also, for those interested in Next Generation Sequencing data, GEO does contain these as well, and a recent R package for analysis is DESeq <http://www.bioconductor.org/packages/release/bioc/html/DESeq.html>

A few datasets that may be of interest (time series data):

Drosophila microarray data:

Single channel:

<http://www.fruitfly.org/cgi-bin/ex/insitu.pl>

Dual-channel :

<http://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE14086>

Drosophila NGS (RNAseq) data:

<http://www.ncbi.nlm.nih.gov/sites/entrez?db=sra&term=SRP001065>