Overview

The purpose of most microarray experiments is to identify differential expression.

The analysis phase of a microarray experiment begins with the scanned images and ultimately produces a ranked list of candidate genes.

Using present technology there are three important phases:

- Image Analysis
- Normalisation
- Statistical Analysis

Image Analysis

A scanned slide produces a pair of grey-scale images, one for the green channel and one for the red channel.

The purpose of image analysis is to extract a pair of intensities for each spot (gene).

The major issues are:

- Each spot contains several pixels.
- The spots cannot be assumed to be circular.
- The spots cannot be assumed to be of the same size.
- The spacing of spots is not perfectly regular.
- The background pixels do not always have lower intensities than the spots.

Normalisation

With present technology, the relative intensities of the red and green channels do not accurately reflect the ratio of mRNA in the samples.

Much of this bias is systematic:

- Print tip effects.
- Spatial effects.
- Dye effects.
- Mean intensity effects.

The purpose of normalisation is to identify and subtract as many of these biases as possible.

Statistical Analysis

The normalised spot intensities are used to infer which genes are more likely to be differentially expressed.

We cannot conclude that a certain gene is differentially expressed just because the red and green intensities are different for a single spot. (We would not expect them to be *exactly* the same even if the gene was not differentially expressed)

We need statistical methods to combine the evidence from several slides simultaneously for all genes and obtain an objective answer.

What methods are available?

There is no single software package that automatically performs image analysis, normalisation and statistical analysis.

- There are various methods for image analysis. For example, Genepix and Spot.
- Terry Speed's group has produced the best methods for normalisation that are available in the SMA package.
- Statistical analyses have been developed for very simple designs. Some of these are available in the SMA package. More complicated designs are still an open research problem.

What is R?

R is a powerful and flexible platform for general statistical analysis.

It provides a suitable environment for manipulating microarray data.

It has comprehensive libraries of standard mathematical and statistical functions.

It supports high quality graphics.

It is a programmable environment that allows for easy development of specialised routines at all levels of complexity.

Why use R?

R provides a single environment in which the analysis phase can be performed.

It is needed to run Spot.

There are substantial microarray analysis resources available specifically for R. Most notably the Bioconductor Project and SMA.

It provides good facilities for exploring your data.

When you need to develop your own statistical analysis for a particular problem, it provides all of the basic statistical tools.

Overview of R for Windows

Shown below is a screen shot of an R session.

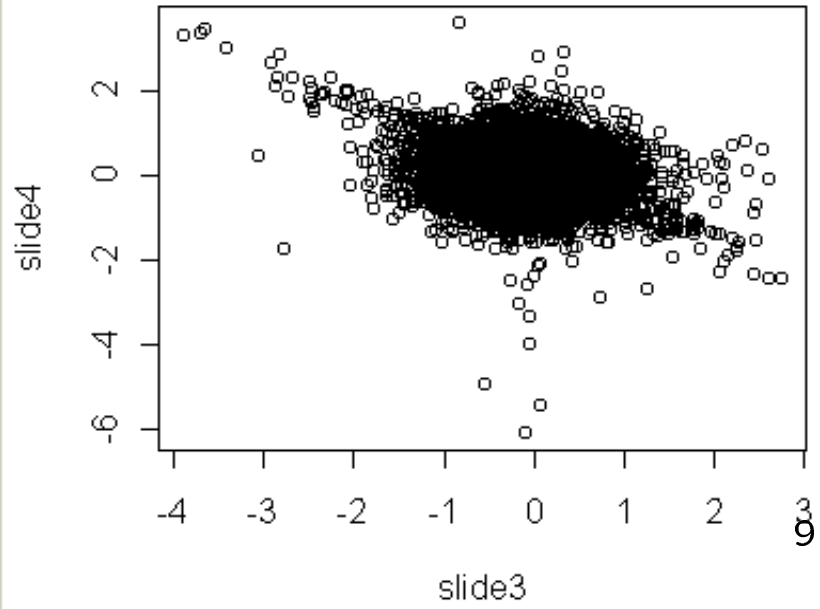
RGui

File History Resize Windows

R Console

```
> objects()
[1] "slide1" "slide2" "slide3" "slide4" "x"      "X"
> mean(slide3)
[1] -0.01327736
> sd(slide3)
[1] 0.4556866
> mean(slide4)
[1] 0.005076649
> sd(slide4)
[1] 0.4542206
> plot(slide3,slide4)
> diff.34<-slide3-slide4
> objects()
[1] "diff.34" "slide1"  "slide2"  "slide3"
> mean(diff.34)
[1] -0.01835401
> sd(diff.34)
[1] 0.7100969
>
```

R Graphics: Device 2 (ACTIVE)



The R Console window

The most important part of the screen is the R Console window.

In this window:

- We type commands to:
  - Produce R output.
  - Produce graphs in the `Graphics` window.
  - To generate new `data objects`.
- The output produced by R is displayed.

R is command driven

Nearly all substantive operations in R are performed by typing commands in the `R Console` window.

There are no menus for performing standard statistical calculations and data manipulations.

For example, to calculate a mean you must type a command such as `mean(x)`.

"Programming" is usually not required.

R is text oriented

Most R output is plain text and appears in the R Console.

Data objects can be displayed as plain text in the R Console.

An important exception is graphics. R provides an excellent facility for producing high quality graphics.

Commands in the R Console

The screenshot showed a simple R session.

In that example, data objects `slide1`, `slide2`, `slide3`, `slide4` have already been created.

They contain the log intensity ratios for 4 different slides in a microarray experiment.

```
> objects()
[1] "slide1" "slide2" "slide3" "slide4"
> mean(slide3)
[1] -0.01327736
> sd(slide3)
[1] 0.4556866
> mean(slide4)
[1] 0.005076649
> sd(slide4)
[1] 0.4542206
> plot(slide3,slide4)
```

```
> # slide3 and slide4 are dye swapped replicates
> # so it is useful to calculate the difference
> diff.34<-slide3-slide4
> objects()
[1] "diff.34" "slide1"  "slide2"  "slide3"  "slide4"
>
```

In the `R Console`:

- `">"` is the prompt. It means that R is waiting for a command to be typed.
- `"objects()"` is the command that causes R to list the available data objects.
- `mean(slide3)` is the command that causes R to calculate the mean of the 16128 log ratios in slide3.

  The result is -0.01327736.
- `"sd(slide3)"` is the command that produces the standard deviation of those values.
- `"plot(slide3,slide4)"` produces the scatter plot.

- The symbol `"#"` indicates that the rest of the line is a comment and not a command.
- The command `"diff.34 <- slide3-slide4"` creates a new data object called diff.34 containing the differences between corresponding log-ratios from slides 3 and 4.
  - `"slide3-slide4"` is the part of the command that tells R to calculate the difference.
  - If we just typed `"slide3-slide4"`, R would calculate the 16128 differences and display them on the screen.
  - `"."` is the preferred symbol for separating words in an object name. `"_"` cannot be used.
  - `"<-"` is the assignment operator. There must not be a space between `"<"` and `"-"`.

Data In R

R has several different data structures.

Vectors

- The basic structure is a `vector`.
  - A list of numbers.
- In the simplest case, a vector can be just one number.
  - These can be created with an assignment such as

```
> a<-1.5
> b<- -2
> mean.1<-mean(slide1)
```

- The value of a vector can displayed by typing its name.

```
> a
[1] 1.5
> b
[1] -2
> mean.1
[1] 0.0032243
```

- Vectors containing more than one number can be created and accessed in various ways.

```
> # Create a vector
> A<-c(2,3,5,7,11,13,17,19)
> #Check whether A is a vector
> is.vector(A)
[1] TRUE
> #Find length of vector
> length(A)
[1] 8
> #List contents of A
> A
[1]  2  3  5  7 11 13 17 19
> #Select a single element of A
> A[4]
[1] 7
> #Select a range of elements
> A[3:6]
[1]  5  7 11 13
```

Matrices

A matrix in R is a two-way array of numbers.

```
> #Here is the matrix B
> B
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[4,]   10   11   12
> # It is a matrix
> is.matrix(B)
[1] TRUE
> # But not a vector
> is.vector(B)
[1] FALSE
> # It has 4 rows and 3 columns
> dim(B)
[1] 4 3
```

```
> #We can select:
> # Single elements
> B[2,2]
[1] 5
> # Blocks of elements
> B[2:4,1:2]
     [,1] [,2]
[1,]    4    5
[2,]    7    8
[3,]   10   11
> # Whole Rows
> B[2,]
[1] 4 5 6
> # Whole Columns
> B[,1]
[1]  1  4  7 10
>
```

Data Frames

A data frame is like a matrix except that:

1. Columns have names.

2. Some columns can have numeric and others character data.

```
> # Slides.data is a data frame containing Unigene
> # and Accession codes together with the log ratios
> # for the slides data.
> Slides.data
   Unigene Accession Slide1   Slide2  Slide3   Slide4
1  Mm.1703  AW542954 -0.181  0.00881 -0.1847 -0.37102
2 Mm.34695  AW553426  0.194  0.10606 -0.0910  0.17478
3  Mm.3433  AW555322  1.841 -1.30757  0.8488 -1.58469
4  Mm.1139  AW556256 -0.154  0.41093  0.0161 -0.00463
5 Mm.70127  AW546379 -0.243  0.19362  0.0390 -0.01326
> # We can select columns as we do with matrices
> Slides.data[,3]
[1] -0.18084  0.19382  1.84112 -0.15396 -0.24293
> # But also by their names, using "$"
> Slides.data$Slide1
[1] -0.18084  0.19382  1.84112 -0.15396 -0.24293
>
```

Lists

A list is a complex data structure that can contain any combination of data types.

In the example below, `Slides` is a list containing four matrices, `R, G, Rb, Gb`

Each matrix contains two columns (corresponding to two slides) and two rows (corresponding to two genes)

```
> is.list(Slides)
[1] TRUE
> attributes(Slides)
$names
[1] "R"  "G"  "Rb" "Gb"
```

```
> Slides
$R
         [,1]     [,2]
[1,] 566.9048 950.1646
[2,] 255.0877 331.5844
$G
         [,1]      [,2]
[1,] 941.7778 1055.1646
[2,] 501.5789  426.3506
$Rb
     [,1] [,2]
[1,]   53   46
[2,]   60   55
$Gb
     [,1] [,2]
[1,]   47   31
[2,]   61   52
> # Components of a list can be extracted using $
> Slides$R
         [,1]     [,2]
[1,] 566.9048 950.1646
[2,] 255.0877 331.5844
```

- Lists are used extensively by packages like `Spot` and `SMA`.
- They provide the mechanism for passing several pieces of information to or from functions.

The workspace image

- All data objects created in an `R` session are stored in the file `.RData`
- When you give the command `objects()` it lists all of the objects stored in `.Rdata`
- The `.Rdata` file is used for permanent storage of your data objects between sessions.
- This means that next time you start `R` all of the previously created data objects will be available.