

# R Basics

Laurent Gatto

[lg390@cam.ac.uk](mailto:lg390@cam.ac.uk)

Computational Statistics for Genome Biology

24<sup>th</sup> June 2013

# Plan

## Introduction

## Data types and structures

- Basic data types

- Higher order objects

## Manipulating data

- Subsetting

## Useful functions

## Plotting

## Programming

## Packages

# Plan

## Introduction

## Data types and structures

- Basic data types

- Higher order objects

## Manipulating data

- Subsetting

## Useful functions

## Plotting

## Programming

## Packages

## Hello world

```
x <- 1  ## a variable
x

## [1] 1

x = 2  ## overwrite the value x
x

## [1] 2

y <- length(x)  ## calling a function
y

## [1] 1
```

## Getting help

- ▶ Just ask!
- ▶ `help.start()` and the HTML help button in the Windows GUI.
- ▶ `help` and `?`: `help("data.frame")` or `?help`.
- ▶ `help.search`, `apropos`
- ▶ Online manuals and mailing lists
  
- ▶ Local R user groups

```
ls()
```

```
## [1] "x" "y"
```

```
rm(y)
```

```
ls()
```

```
## [1] "x"
```

# Plan

## Introduction

## **Data types and structures**

Basic data types

Higher order objects

## Manipulating data

Subsetting

## Useful functions

## Plotting

## Programming

## Packages

```
c(1, 3, 9, -1)
```

```
## [1] 1 3 9 -1
```

A vector contains an indexed set of values

- ▶ index starts at 1;
- ▶ all items are of the same storage mode;
- ▶ one of logical, numeric, complex or character,

numeric can further be broken into integer, single and double types (only important when passing these to C or Fortran code, though).



```
mode(1)
```

```
## [1] "numeric"
```

```
typeof(1)
```

```
## [1] "double"
```

```
mode(1L)
```

```
## [1] "numeric"
```

```
typeof(1L)
```

```
## [1] "integer"
```

```
mode("1")

## [1] "character"

typeof("1")

## [1] "character"

mode(TRUE)

## [1] "logical"

typeof(FALSE)

## [1] "logical"

## as we are talking about booleans...
TRUE & TRUE

## [1] TRUE
```

The different modes an types can be retrieved and coerced with the `is.*` and `as.*` functions.

```
x <- 1
typeof(x)

## [1] "double"

y <- as.integer(x)
typeof(y)

## [1] "integer"

is.integer(y)

## [1] TRUE
```

## Special values

NULL

NA

NaN

Inf

-Inf

`is.null()`

`is.na()`

`is.infinite()`

What are the mode and types of these?

All these are objects with a certain class.

```
class(x)
```

```
## [1] "numeric"
```

```
class("a character")
```

```
## [1] "character"
```

## Creating vectors with functions

```
vector(mode = "character", length = 3)
```

```
## [1] "" "" ""
```

```
vector(mode = "numeric", length = 4)
```

```
## [1] 0 0 0 0
```

```
numeric(4)
```

```
## [1] 0 0 0 0
```

## Creating vectors with functions (2)

```
x <- c(1, 4, 7, 10) ## concatenate
```

```
x
```

```
## [1] 1 4 7 10
```

```
y <- 1:5 ## integer sequence
```

```
y
```

```
## [1] 1 2 3 4 5
```

```
z <- seq(from = 1, to = 10, by = 2)
```

```
z
```

```
## [1] 1 3 5 7 9
```

## Arguments by position or name

```
z1 <- seq(from = 1, to = 10, by = 2)
```

```
z2 <- seq(1, 10, 2)
```

```
z1 == z2
```

```
## [1] TRUE TRUE TRUE TRUE TRUE
```

```
all(z1 == z2)
```

```
## [1] TRUE
```

```
identical(z1, z2)
```

```
## [1] TRUE
```



## Vectorised arithmetic

```
x <- 1:5
```

```
y <- 5:1
```

```
x
```

```
## [1] 1 2 3 4 5
```

```
y
```

```
## [1] 5 4 3 2 1
```

```
x + y
```

```
## [1] 6 6 6 6 6
```

```
x^2
```

```
## [1] 1 4 9 16 25
```

## Matrices

are 2-dimensional vectors

```
m <- matrix(1:12, nrow = 4, ncol = 3)
```

```
m
```

```
##      [,1] [,2] [,3]
## [1,]    1    5    9
## [2,]    2    6   10
## [3,]    3    7   11
## [4,]    4    8   12
```

```
dim(m)
```

```
## [1] 4 3
```

What if I don't get the data or dimensions right?

What if I don't get the data or dimensions right?

```
matrix(1:11, 4, 3) ## recycling
```

```
## Warning: data length [11] is not a sub-multiple  
or multiple of the number of rows [4]
```

```
##      [,1] [,2] [,3]  
## [1,]    1    5    9  
## [2,]    2    6   10  
## [3,]    3    7   11  
## [4,]    4    8    1
```

```
matrix(1:12, 3, 3)
```

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

```
x <- 1:12
class(x)

## [1] "integer"

dim(x)

## NULL

dim(x) <- c(4, 3)
x

##      [,1] [,2] [,3]
## [1,]    1    5    9
## [2,]    2    6   10
## [3,]    3    7   11
## [4,]    4    8   12

class(x)

## [1] "matrix"
```

## Arrays

are n-dimensional vectors

```
array(1:16, dim = c(2, 4, 2))
```

```
## , , 1
```

```
##
```

```
##      [,1] [,2] [,3] [,4]
```

```
## [1,]    1    3    5    7
```

```
## [2,]    2    4    6    8
```

```
##
```

```
## , , 2
```

```
##
```

```
##      [,1] [,2] [,3] [,4]
```

```
## [1,]    9   11   13   15
```

```
## [2,]   10   12   14   16
```

## Lists

are ordered collections of elements that can be arbitrary R objects.

```
(ll <- list(a = 1:3, f = length))

## $a
## [1] 1 2 3
##
## $f
## function (x) .Primitive("length")

ll[1] ## a list of length 1

## $a
## [1] 1 2 3

ll[[1]] ## or ll$a - first element

## [1] 1 2 3
```

```
ll
```

```
## $a
```

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

```
##
```

```
## $f
```

```
## function (x) .Primitive("length")
```

```
ll$f(ll)
```

```
## [1] 2
```

## Data.frames

are 2-dimensional lists (with elements of same length!).

```
dfr <- data.frame(type = c(
  rep("case", 2),
  rep("ctrl", 2)),
  time = rnorm(4))
```

```
dfr
```

```
##   type    time
## 1 case 0.38745
## 2 case 0.04723
## 3 ctrl 0.38085
## 4 ctrl 0.34757
```



```
dfr[1, ]
```

```
##   type   time
```

```
## 1 case 0.3875
```

```
dfr[1, "time"]
```

```
## [1] 0.3875
```

```
dfr$time
```

```
## [1] 0.38745 0.04723 0.38085 0.34757
```

## Environments

are unordered sets of objects.

```
e <- new.env()
e[["a"]] <- 1:3
assign("b", "CSAMA", envir = e)
ls(e)

## [1] "a" "b"

e$a

## [1] 1 2 3

get("b", e)

## [1] "CSAMA"
```

## Names

We have seen that function arguments have names, and named our `data.frame` columns. We can also name `matrix/data.frame` columns and rows, dimensions, and vector items.

```
x <- c(a = 1, b = 2)
```

```
x
```

```
## a b
```

```
## 1 2
```

```
names(x)
```

```
## [1] "a" "b"
```

```
M <- matrix(c(4, 8, 5, 6, 4, 2, 1, 5, 7), nrow=3)
dimnames(M) <- list(year =
                    c(2005, 2006, 2007),
                    "mode of transport" =
                    c("plane", "bus", "boat"))
```

M

```
##           mode of transport
## year    plane bus boat
## 2005      4   6   1
## 2006      8   4   5
## 2007      5   2   7
```

## Factors

for categorical data

```
sample.ExpressionSet$type
```

```
## [1] Control Case Control Case Case  
## [6] Control Case Case Case Control  
## [11] Case Control Case Case Case  
## [16] Control Case Control Case Case  
## [21] Control Control Control Control Case  
## [26] Case  
## Levels: Case Control
```

## Data types summary

	1 dim	2 dims	<i>n</i> dims
same type	vector	matrix	array
diff types	list	data.frame*	list

(\*elements of same length)

## Higher order objects

When the data to be stored is more complex, special objects are created to store and handle it in a specialised manner. These higher order objects are constructed using the data types we have seen so far as building blocks.

Let's look at how microarray data is handled in Bioconductor. The eSet model has been re-used for other technologies.

```
library(Biobase)
data(sample.ExpressionSet)
sample.ExpressionSet

## ExpressionSet (storageMode: lockedEnvironment)
## assayData: 500 features, 26 samples
##   element names: exprs, se.exprs
## protocolData: none
## phenoData
##   sampleNames: A B ... Z (26 total)
##   varLabels: sex type score
##   varMetadata: labelDescription
## featureData: none
## experimentData: use 'experimentData(object)'
## Annotation: hgu95av2
```



```
class(sample.ExpressionSet)
```

```
## [1] "ExpressionSet"
```

```
## attr(,"package")
```

```
## [1] "Biobase"
```

```
slotNames(sample.ExpressionSet)
```

```
## [1] "experimentData"      "assayData"
```

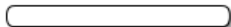
```
## [3] "phenoData"           "featureData"
```

```
## [5] "annotation"          "protocolData"
```

```
## [7] ".__classVersion__"
```

```
?ExpressionSet
```

- assayData** expression values in identical sized matrices.
- phenoData** sample annotation in AnnotatedDataFrame.
- featureData** feature annotation in AnnotatedDataFrame.
- experimentData** description of the experiment as a MIAME object  
(see ?MIAME for more details).
- annotation** type of chip as a character.
- protocolData** scan dates as a character.



## The assayData slot

Stored the expression data of the assay.

```
exprs(sample.ExpressionSet)[1:4, 1:3]
```

```
##           A           B           C
## AFX-MurIL2_at 192.74  85.753 176.76
## AFX-MurIL10_at 97.14 126.196  77.92
## AFX-MurIL4_at  45.82   8.831  33.06
## AFX-MurFAS_at  22.54   3.601  14.69
```

```
dim(sample.ExpressionSet)
```

```
## Features  Samples
##         500      26
```

## The `phenoData` slot

stores the meta data about the samples.

```
phenoData(sample.ExpressionSet)

## An object of class 'AnnotatedDataFrame'
##   sampleNames: A B ... Z (26 total)
##   varLabels: sex type score
##   varMetadata: labelDescription
```

## The `featureData` slot

stores the meta data about the features.

```
featureData(sample.ExpressionSet)
```

```
## An object of class 'AnnotatedDataFrame': none
```

```
## NULL
```

## AnnotatedDataFrame

consists of a collection of samples and the values of variables measured on those samples. There is also a description of each variable measured. AnnotatedDataFrame associates a `data.frame` with its metadata.

```
head(pData(sample.ExpressionSet))
```

```
##      sex    type score
## A Female Control  0.75
## B  Male     Case   0.40
## C  Male     Control 0.73
## D  Male     Case   0.42
## E Female     Case   0.93
## F  Male     Control 0.22
```

# Plan

## Introduction

## Data types and structures

- Basic data types

- Higher order objects

## **Manipulating data**

- Subsetting

## Useful functions

## Plotting

## Programming

## Packages



- ▶ One of the most powerful features of R is its ability to manipulate subsets of vectors and arrays.
- ▶ As seen, subsetting is done with, `[]`, `[, ]`, ...

## Subsetting with positive indices

```
x <- 1:10
```

```
x[3:7]
```

```
## [1] 3 4 5 6 7
```

```
x[9:11]
```

```
## [1] 9 10 NA
```

```
x[0:1]
```

```
## [1] 1
```

```
x[c(1, 7, 2, NA)]
```

```
## [1] 1 7 2 NA
```

## Assignments with positive indices

```
x[2] <- 20  
x[4:5] <- x[4:5] * 100  
## x[1:6] ?
```

## Assignments with positive indices

```
x[2] <- 20  
x[4:5] <- x[4:5] * 100  
## x[1:6] ?
```

```
x[1:6]  
## [1] 1 20 3 400 500 6
```

## Subsetting with negative indices

```
x <- 1:10  
## x[-c(3:7)] ?
```

## Subsetting with negative indices

```
x <- 1:10  
## x[-c(3:7)] ?
```

```
x[-c(3:7)]  
## [1] 1 2 8 9 10
```

## Subsetting with logical predicates

```
x[c(TRUE, TRUE, rep(FALSE, 8))]
```

```
## [1] 1 2
```

```
x > 5
```

```
## [1] FALSE FALSE FALSE FALSE FALSE TRUE TRUE
```

```
## [8] TRUE TRUE TRUE
```

```
x[x > 5]
```

```
## [1] 6 7 8 9 10
```

## Subsetting with logical predicates

```
## x[c(TRUE, FALSE)] ?
```



## Subsetting with logical predicates

```
## x[c(TRUE, FALSE)] ?
```

```
x[c(TRUE, FALSE)] ## recycled
```

```
## [1] 1 3 5 7 9
```

## Subsetting by names

```
x <- c(a = 1, b = 2, c = 2)
```

```
x[c("a", "c")]
```

```
## a c
```

```
## 1 2
```

```
x[c("a", "d")]
```

```
##      a <NA>
```

```
##      1  NA
```

## Subsetting matrices

```
M <- matrix(1:12, 3)
M[1, ] ## row -> vector (or drop = FALSE)

## [1] 1 4 7 10

M[, 1] ## column -> vector (or drop = FALSE)

## [1] 1 2 3

M[2, 3] <- 0
M

##      [,1] [,2] [,3] [,4]
## [1,] 1    4    7    10
## [2,] 2    5    0    11
## [3,] 3    6    9    12
```

## Subsetting matrices (2)

```
M < 9
```

```
##      [,1] [,2] [,3] [,4]
## [1,] TRUE TRUE  TRUE FALSE
## [2,] TRUE TRUE  TRUE FALSE
## [3,] TRUE TRUE FALSE FALSE
```

```
M[M < 9] <- -1
```

```
M
```

```
##      [,1] [,2] [,3] [,4]
## [1,]  -1  -1  -1  10
## [2,]  -1  -1  -1  11
## [3,]  -1  -1   9  12
```

## Subsetting lists

```
ll <- list(a = 1:3, b = "CSAMA", c = length)
ll[1]  ## still a list, but of length 1

## $a
## [1] 1 2 3

ll[[1]]  ## first element of the list

## [1] 1 2 3
```

## Subsetting ExpressionSet instances

It is reasonable to expect that subsetting operations work also for higher order objects.

```
sample.ExpressionSet[1:10, 1:2]

## ExpressionSet (storageMode: lockedEnvironment)
## assayData: 10 features, 2 samples
##   element names: exprs, se.exprs
## protocolData: none
## phenoData
##   sampleNames: A B
##   varLabels: sex type score
##   varMetadata: labelDescription
## featureData: none
## experimentData: use 'experimentData(object)'
## Annotation: hgu95av2
```

# Plan

## Introduction

## Data types and structures

- Basic data types

- Higher order objects

## Manipulating data

- Subsetting

## Useful functions

## Plotting

## Programming

## Packages

## Data IO

**read.table** creates a `data.frame` from a spreadsheet file.

**write.table** writes a `data.frame/matrix` to a spreadsheet (tsv, csv).

**save** writes an binary representation of R objects to a file (cross-platform).

**load** load a binary R file from disk.

Specialised data formats often have specific i/o functionality (microarray CEL files, XML, ...)



```
read.table("./Data/data.csv", sep = ",",  
           header = TRUE, row.names = 1)
```

```
##           A  B C  D  
## 1 -0.15330 10 x 10  
## 2 -0.13868  3 n  9  
## 3 -0.43323  2 f  8  
## 4  1.64569  4 o  7  
## 5  0.23381  6 b  6  
## 6  0.98770  9 m  5  
## 7 -0.25565  7 c  4  
## 8 -0.74719  1 l  3  
## 9 -0.02001  5 e  2  
## 10 -0.95000  8 v  1
```

```
read.csv("./Data/data.csv", row.names = 1)
```

```
##           A  B C  D
## 1 -0.15330 10 x 10
## 2 -0.13868  3 n  9
## 3 -0.43323  2 f  8
## 4  1.64569  4 o  7
## 5  0.23381  6 b  6
## 6  0.98770  9 m  5
## 7 -0.25565  7 c  4
## 8 -0.74719  1 l  3
## 9 -0.02001  5 e  2
## 10 -0.95000  8 v  1
```

```
x <- read.csv("../Data/data.csv", row.names = 1)
save(x, file = "../Data/data.rda")
rm(x)
load("../Data/data.rda")
x[1:3, ]
```

```
##           A  B C  D
## 1 -0.1533 10 x 10
## 2 -0.1387  3 n  9
## 3 -0.4332  2 f  8
```

## String manipulation (1)

```
paste("abc", "def", sep = "-")
```

```
## [1] "abc-def"
```

```
paste0("abc", "def")
```

```
## [1] "abcdef"
```

## String manipulation (2)

```
month.name[1:4]

## [1] "January" "February" "March" "April"

grep("Feb", month.name)

## [1] 2

grep("Feb", month.name, value = TRUE)

## [1] "February"

grepl("Feb", month.name)

## [1] FALSE TRUE FALSE FALSE FALSE FALSE FALSE
## [8] FALSE FALSE FALSE FALSE FALSE
```

## String manipulation (3)

```
month.name[1]
```

```
## [1] "January"
```

```
length(month.name[1])
```

```
## [1] 1
```

```
nchar(month.name[1])
```

```
## [1] 7
```

## String manipulation (4)

```
strsplit("abc-def", "-")
```

```
## [[1]]
```

```
## [1] "abc" "def"
```

## String manipulation (4)

```
strsplit("abc-def", "-")
```

```
## [[1]]
```

```
## [1] "abc" "def"
```

```
strsplit(c("abc-def", "ghi-jkl"), "-")
```

```
## [[1]]
```

```
## [1] "abc" "def"
```

```
##
```

```
## [[2]]
```

```
## [1] "ghi" "jkl"
```



## Comparing and matching (1)

```
set.seed(1)
x <- sample(letters[1:10], 6)
y <- sample(letters[1:10], 6)
x

## [1] "c" "d" "e" "g" "b" "h"

y

## [1] "j" "f" "i" "a" "b" "g"
```

## Comparing and matching (2)

```
intersect(x, y)
```

```
## [1] "g" "b"
```

```
setdiff(x, y)
```

```
## [1] "c" "d" "e" "h"
```

```
union(x, y)
```

```
## [1] "c" "d" "e" "g" "b" "h" "j" "f" "i" "a"
```

## Comparing and matching (3)

```
x %in% y
```

```
## [1] FALSE FALSE FALSE TRUE TRUE FALSE
```

```
x == y
```

```
## [1] FALSE FALSE FALSE FALSE TRUE FALSE
```

```
match(x, y)
```

```
## [1] NA NA NA 6 5 NA
```

## Generating data (1)

```
seq(1, 7, 3)
```

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

```
rep(1:2, 2)
```

```
## [1] 1 2 1 2
```

```
rep(1:2, each = 2)
```

```
## [1] 1 1 2 2
```

## Generating data (2)

```
runif(5)
```

```
## [1] 0.6870 0.3841 0.7698 0.4977 0.7176
```

```
rnorm(5)
```

```
## [1] 2.4047 0.7636 -0.7990 -1.1477 -0.2895
```

## About the data

```
table(sample(letters, 100, replace = TRUE))
```

```
##
```

```
## a b c d e f g h i j k l m n o p q r s t u v w x y
```

```
## 2 2 4 4 2 2 4 2 6 4 5 7 9 3 1 3 5 3 5 5 6 4 5 2 2
```

```
## z
```

```
## 3
```

```
summary(rnorm(100))
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
```

```
## -1.6800 -0.8280 -0.0081 -0.0089  0.6090  2.6600
```

```
head(x)
```

```
## [1] "c" "d" "e" "g" "b" "h"
```

```
tail(x)
```

```
## [1] "c" "d" "e" "g" "b" "h"
```

```
M <- matrix(rnorm(1000), ncol = 4)
```

```
head(M)
```

```
##           [,1]      [,2]      [,3]      [,4]
## [1,]  0.7796 -0.3399 -1.44689 -0.1658
## [2,]  0.7132  0.6063  1.01951  0.5571
## [3,] -0.5429  1.3411  1.17855  1.4443
## [4,]  0.8858  0.7673 -0.01026  0.9014
## [5,] -0.3486  0.1937  0.26862 -0.2220
## [6,] -1.0081  1.1406  1.34203  0.1062
```

# Plan

## Introduction

## Data types and structures

- Basic data types

- Higher order objects

## Manipulating data

- Subsetting

## Useful functions

## Plotting

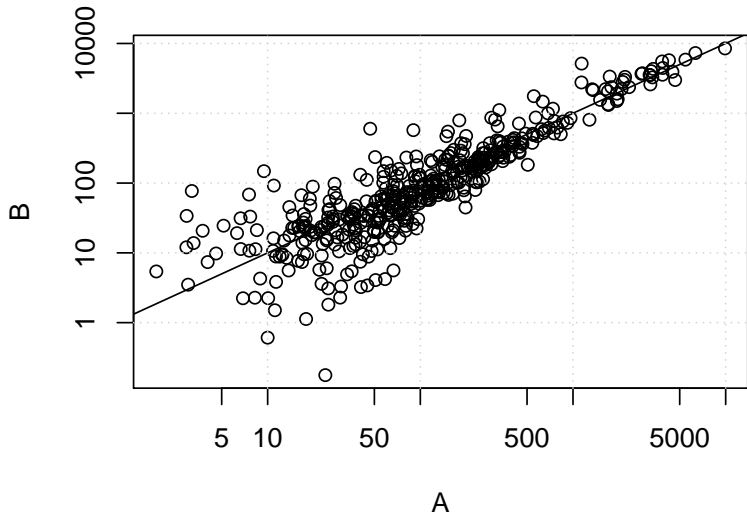
## Programming

## Packages



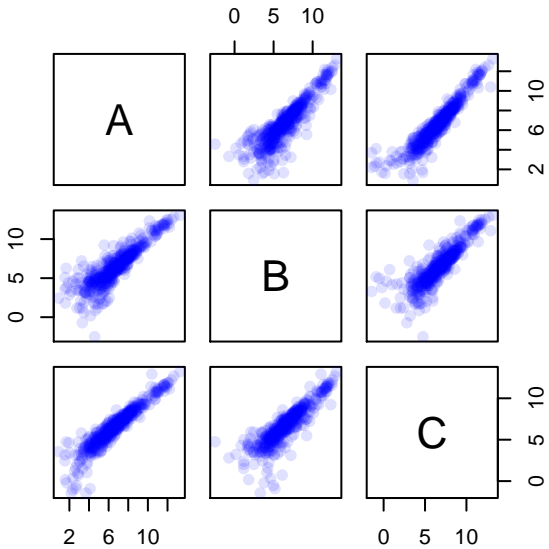
- ▶ Scatterplots with `plot`
- ▶ Boxplots with `boxplot`
- ▶ Barplots with `barplot`
- ▶ Histograms with `hist`
- ▶ `smoothScatter`

```
plot(exprs(sample.ExpressionSet[, 1]),  
      exprs(sample.ExpressionSet[, 2]),  
      log = "xy",  
      xlab = sampleNames(sample.ExpressionSet)[1],  
      ylab = sampleNames(sample.ExpressionSet)[2])  
abline(0, 1)  
grid()
```

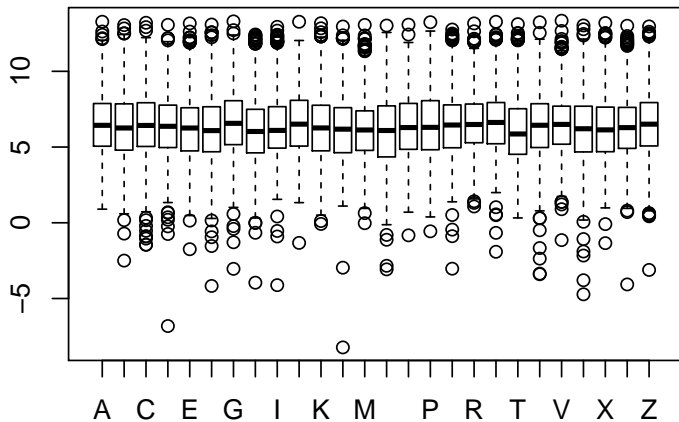


To create subplots, one can use `par(mfrow = c(2,2))`, `layout`, or (for scatterplots)

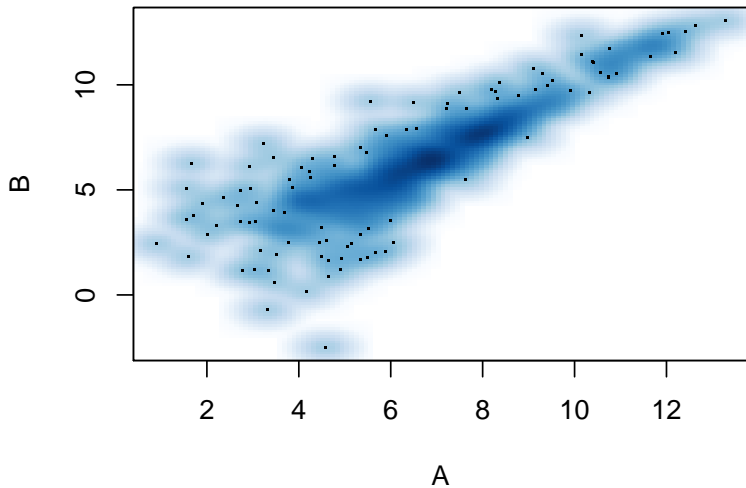
```
pairs(log2(exprs(sample.ExpressionSet)[, 1:4])),  
      pch = 19,  
      col = "#0000FF20")
```



```
boxplot(log2(exprs(sample.ExpressionSet)))
```



```
smoothScatter(log2(exprs(sample.ExpressionSet)[, 1:2]))
```



We have not covered lattice and ggplot2.

## References

- ▶ <http://gallery.r-enthusiasts.com/allgraph.php>
- ▶ R Graphics manual:  
[http://rgm3.lab.nig.ac.jp/RGM/r\\_image\\_list](http://rgm3.lab.nig.ac.jp/RGM/r_image_list)
- ▶ <http://www.cookbook-r.com/Graphs/> (ggplot2)
- ▶ *ggplot2: Elegant Graphics for Data Analysis*, Hadley Wickham (2009)
- ▶ *Lattice: Multivariate Data Visualization with R*, Deepayan Sarkar (2008)



# Plan

## Introduction

## Data types and structures

- Basic data types

- Higher order objects

## Manipulating data

- Subsetting

## Useful functions

## Plotting

## Programming

## Packages

## Flow control

```
for (var in seq) expr  
while (cond) expr  
repeat expr break
```

```
for (i in 1:4) {  
  ## bad  
  print(i^2)  
}
```

```
## [1] 1  
## [1] 4  
## [1] 9  
## [1] 16
```

```
(1:4)^2 ## good
```

```
## [1] 1 4 9 16
```

## The apply family and friends

- ▶ Applies a function to each element of an input, being a list or a vector (`sapply`, `lapply`), a matrix or a data frame (`apply`) or an environment (`eapply`).
- ▶ Same functionality than an explicit `for` loop, but often more elegant, function-centric, **not** faster.

```
sapply(month.name[1:2], paste0, "_2012")
```

```
##           January           February  
## "January_2012" "February_2012"
```

```
lapply(month.name[1:2], paste0, "_2012")
```

```
## [[1]]  
## [1] "January_2012"  
##  
## [[2]]  
## [1] "February_2012"
```

## Conditionals

```
if (cond) expr1 else expr2  
ifelse(cond, expr1, expr2)  
switch
```

```
x <- 2
if (x > 0) {
  ## bad
  log2(x)
} else {
  log2(-x)
}

## [1] 1

log2(abs(x)) ## better

## [1] 1
```

## Exception handling

`try(exprs)` will either return the value of the expression `expr`, or an object of class `try-error`.

`tryCatch` provides a more configurable mechanism for condition handling and error recovery.



## Writing functions

```
myFun <- function(param1, param2, ...) {  
  ## function body  
  ## acting on copies of the params  
  ans <- param1 + param2  
  return(ans)  
}
```

## Function facts

- ▶ Single return value.
- ▶ To return multiple items, use a list or a proper object (see OO programming).
- ▶ The return value is either the last statement, or explicit return using `return` (can be called from any where in a function)

## Function facts (cont.)

- ▶ Functions act on a pass-by-copy semantic.

```
x <- 1
f <- function(x) {
  x <- x + 10
  x
}
f(x)
## [1] 11

x
## [1] 1
```

## Function facts (cont.)

- ▶ Functions live/act in their own environment and have access to *global* variables.

```
x <- 1
f <- function() {
  x <- x + 10
  x
}
f()
## [1] 11
x
## [1] 1
```

# Plan

## Introduction

## Data types and structures

- Basic data types

- Higher order objects

## Manipulating data

- Subsetting

## Useful functions

## Plotting

## Programming

## Packages

- ▶ Primary mechanism to distribute R software is via packages.
- ▶ Packages are installed in libraries (directories) on your hard disk, and they are loaded with the `library` function.
- ▶ There are software, data and annotation packages.
- ▶ The Comprehensive R Archive Network (CRAN) is the main package repository. It provides an automatic build framework for package authors.
- ▶ The Bioconductor project manages its own CRAN-style repository.
- ▶ R-forge – <https://r-forge.r-project.org/>

## Package installation

- ▶ From within R , using `install.packages` - takes care of dependencies.
- ▶ Update all installed packages with `update.packages`.
- ▶ For Bioconductor packages, use `biocLite`:

```
source("http://www.bioconductor.org/biocLite.R")  
## or, if you have already done so in the past  
library("BiocInstaller")  
biocLite("packageName")
```

## Getting information about packages

- ▶ CRAN/Bioconductor/R-forge web pages
- ▶ Documentation

```
help(package = "Biobase")
```

- ▶ Vignettes (mandatory for Bioconductor packages)

```
vignette(package = "Biobase")
```

```
vignette("Bioconductor", package = "Biobase")
```

- ▶ Demos

```
demo("lattice", package = "lattice")
```



```
packageDescription("Biobase")
```

```
## Package: Biobase
## Title: Biobase: Base functions for Bioconductor
## Version: 2.21.4
## Author: R. Gentleman, V. Carey, M. Morgan, S. Falcon
## Description: Functions that are needed by many other packages or
##             which replace R functions.
## Suggests: tools, tkWidgets, ALL
## Depends: R (>= 2.10), BiocGenerics (>= 0.3.2), utils
## Imports: methods, BiocGenerics
## Maintainer: Bioconductor Package Maintainer
##             <maintainer@bioconductor.org>
## License: Artistic-2.0
## Collate: tools.R strings.R environment.R vignettes.R packages.R
##             .....
## LazyLoad: yes
## biocViews: Infrastructure, Bioinformatics
## Packaged: 2013-06-11 02:32:03 UTC; biocbuild
## Built: R 3.1.0; x86_64-unknown-linux-gnu; 2013-06-18 00:33:27 UTC;
##             unix
##
## -- File: /home/lgatto/R/x86_64-unknown-linux-gnu-library/3.1/Biobase/Meta/pa
```

## Package versions

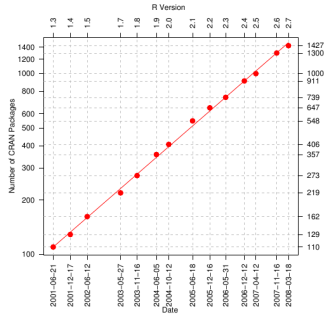
- ▶ New Bioconductor releases appear twice a year. Bioconductor versions are **tied** to R versions.
- ▶ Stable packages versions are  $x.y.z$  where  $x \geq 1$  and  $y$  is even
- ▶ Devel packages have  $y$  odd.

**Bioconductor** 636 reviewed packages

**CRAN** 3889 packages

**R-forge** 1313 projects

(19<sup>th</sup> June 2012)



## Finding packages

- ▶ BiocViews – <http://bioconductor.org/packages/release/BiocViews.html>.
- ▶ CRAN Task Views – <http://cran.r-project.org/web/views/>.
- ▶ sos to search inside contributed R packages.

```
toLatex(sessionInfo())
```

- ▶ R Under development (unstable) (2013-06-16 r62969),  
x86\_64-unknown-linux-gnu
- ▶ Locale: LC\_CTYPE=en\_GB.UTF-8, LC\_NUMERIC=C,  
LC\_TIME=en\_GB.UTF-8, LC\_COLLATE=en\_GB.UTF-8,  
LC\_MONETARY=en\_GB.UTF-8, LC\_MESSAGES=en\_GB.UTF-8,  
LC\_PAPER=C, LC\_NAME=C, LC\_ADDRESS=C, LC\_TELEPHONE=C,  
LC\_MEASUREMENT=en\_GB.UTF-8, LC\_IDENTIFICATION=C
- ▶ Base packages: base, datasets, graphics, grDevices, methods,  
parallel, stats, utils
- ▶ Other packages: Biobase 2.21.4, BiocGenerics 0.7.2, knitr 1.2
- ▶ Loaded via a namespace (and not attached): digest 0.6.3,  
evaluate 0.4.3, formatR 0.7, stringr 0.6.2, tools 3.1.0

## References

- ▶ W. N. Venables, D. M. Smith and the R Development Core Team, An Introduction to R (get it with `help.start()`)
- ▶ R. Gentleman, R Programming for Bioinformatics, CRC Press, 2008

- ▶ This work is licensed under a CC BY-SA 3.0 License.
- ▶ Course web page and more material:  
<https://github.com/lgatto/TeachingMaterial>

Thank you for your attention