Workshop: Introduction to R

Jean Monlong & Simon Papillon

Human Genetics department

October 21, 2013

Why learning R ?

Useful for your research

- To explore your results. Curiosity and safety !
- To do/understand your analysis. Independence and control !
- To apply the latest Bioinformatics analyzes. Bioconductor !
- To keep track of your analysis. Reproducibility and automation !

- You do it, not some busy bioinformatician.

It's a good time investment

Simple: interpretative language(no compilation needed), no memory, vectorized.

Free: widely used, vast community of R users, good life expectancy.

Multiplatform: Windows, Mac, Unix, it works everywhere.

R vs other languages

Let's create an array, shuffle it and find where is 5.

In C...

```
#include <stdlib.h>
#include <time.h>
int main() {
    int size = 10;
    int *elements = malloc(sizeof(int)*size);
    int i = 0;
    srand(time(NULL));
    for (i = 0; i < size; ++i)
        elements[i] = i;
    for (i = size - 1; i > 0; --i) {
        int w = rand()%i;
        int t = elements[i];
        elements[i] = elements[w];
        elements[w] = t;
    }
    for(i = 0; i < size; ++i) {
        if(elements[i] == 5)
            printf("%d\n", i);
    }
    free(elements);
}
```

In R...

```
which(sample(0:9) == 5)
```

Easy installation

- Install R from
  `http://cran.r-project.org/`

- Additionally, you can get a nice interface through Rstudio Desktop from
  `http://www.rstudio.com/ide/download/desktop`

Data structure - Overview

Unit type

numeric Numbers, e.g. 0, 1, 42, −66.6.

character Words, e.g. “male”, “ENSG0007”,“Vive la France”.

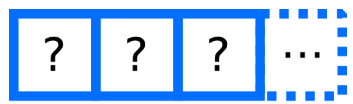
logical Binary, i.e. two possible values: *TRUE* or *FALSE*.

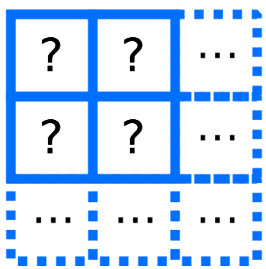
Structure

vector Ordered collection of elements of the same type.

matrix Matrix of element of the same type.

list Flexible container, mixed type possible. Recursive.





Assign a value to an object

Choose an object name

- **Letters**, **numbers**, **dot** or **underline** characters.
- **Starts with a letter** or the dot not followed by a number.
- Correct: “valid.name”, ”valid_name”, ”valid2name3”.
- Incorrect: ”valid name”, ”valid-name”, ”1valid2name3”.

Assign a value

The name of the object followed by the assignment symbol and the value.

```
valid.name_123 = 1
valid.name_123 <- 1

valid.name_123
```

Vectors

vector construction

c	Concatenate function.
1:10	vector with numbers from 1 to 10.
rep	Repeat element several times.

Example

```
luckyNumbers = c(4,8,15,16,23,42)
luckyNumbers
oneToTen = 1:10
tenOnes = rep(1,10)
samples = c("sampA","sampB")
samples
```

Vectors

Manipulation

Using an index between [ ].

vec[i:j] Get or set a vector from i^{th} to j^{th} values.

Characterization

length Number of element in the vector.

names Get or set the names of the vector's values.

Example

```
luckyNumbers[3]
luckyNumbers[2:4]
luckyNumbers[2:4] = c(14,3,9)

length(luckyNumbers)

names(luckyNumbers)
names(luckyNumbers) = c("frank","henry","philip",
                        "steve","tom","francis")
luckyNumbers["philip"]
```

Vectors

Manipulation

sort	Sort a vector.
order	Get the index of the sorted elements.
rev	Reverse a vector.
sample	Shuffle a vector.

Example

```
sort(luckyNumbers)
luckyNumbers[order(luckyNumbers)]

sort(c(luckyNumbers,1:10,tenOnes))

rev(1:10)

sample(1:10)
```

Vectors

Exploration

head/tail Print the first/last values.

On *numeric* vectors:

summary Summary statistics: minimum, mean, maximum, ...

min/max/mean/var Minimum, maximum, average, variance.

sum Sum of the vector's values.

Example

```
head(samples)
summary(luckyNumbers)
mean(luckyNumbers)
min(luckyNumbers)
```

Vectors

Arithmetic operators

- Simple arithmetic operations over all the values of the vector.
- Or values by values when using vectors of same length.
- Arithmetic operation: +, -, *, /, ** *or* $\wedge$.
- Others exist but let's forget about them for a while.

Example

```
luckyNumbers * 4
luckyNumbers - luckyNumbers
luckyNumbers * 1:length(luckyNumbers)
luckyNumbers ** 2
```

Exercise - Guess my favorite number

Instructions

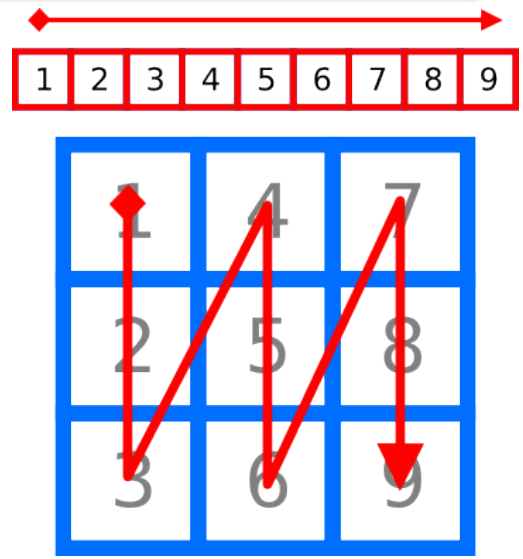
1. Create a vector with 6 *numeric* values
2. Multiply it by 6.
3. Add 21.
4. Divide it by 3
5. Substract 1.
6. Halve it.
7. Substract its original values.

Matrix

Specific to matrices

matrix Create a matrix from a vector.
2^{nd} and 3^{rd} parameters define the number of rows and columns.

mat[i:j,k:l] Subset from the i to j row and k to l column.



Example

```
neo = matrix(1:12,3,4)
neo

neo[1:2,1:3]
neo[1:2,1:3] = matrix(rep(1,6),2,3)
```

Matrix

Specific to matrices

rbind/cbind Concatenate vectors or matrix by row or column.

dim Dimension of the matrix: number of rows and columns.

rownames/colnames Get or set the names of the rows/columns.

Example

```
rbind(neo, neo)
cbind(neo, neo)

dim(neo)
dim(rbind(neo,neo))

colnames(neo) = c("gene1","gene2","gene3","gene4")
rownames(neo) = c("sample1","sample2","sample3")
neo
```

Matrix

Same as vector

- length, head, tail.
- For *numeric* matrix: min, max, sum, mean.
- Arithmetic operations: +, -, *, /.

Example

```
head(mat)
mean(mat)
sum(mat) / length(mat)

mat * 2
mat + mat
```

Exercise

1. Create a matrix with 100 rows and 4 columns with random numbers inside. *Tip:* runif *function for random numbers.*
2. Name the columns. E.g. *sampleA*, *sampleB*, ...
3. Print the name of the column with the largest mean value.
4. Print the name of the column with the largest value.

Functions - apply

New best friend

- Apply a function to row or columns of a 2 dimension data structure (matrix or data frame).
- No manual iteration, the loop is implicit.
- Second parameter: 1 means rows, 2 means columns.

Example

```
apply(mat,1,mean)
apply(mat,2,function(x){
  x.mean = mean(x)
  return(x.mean+1)
})
```

Lists

Flexible container

A list can contain any element type. It does not require elements to be of the same type.

list	Create a list.
l[[i]]	Get or set the i^{th} object of the list.
l$toto	Get or set the element labeled as *toto*.
names	Get or set the names of the list elements.
length	Get the number of element in the list.
str	Output the structure of a R object.

Example

```
l = list(vec=1:10,mat=matrix(runif(25),5))
str(l)
l
l$vec = 1
l
```

Functions - lapply

apply for lists

- Useful way to iterate through lists.

Example

```
file_list <- list.files('.')
files_content <- lapply(file_list, function(file) \{
data <- read.csv(file)
#Do something with the data
return(data)
\})
```

Functions

- Name of the function with parameters between parenthesis.
- E.g. mean(x).

Do your own

function To define functions.

- All the object created within the function are temporary.

return Define what will be returned by the function.

Example

```
almostMean = function(x){
  x.mean = mean(x)
  return(x.mean+1)
}
almostMean(0:10)
x.mean
```

Functions - Exercise

Create a function that:

1. returns the average of the minimum and maximum value of a vector.
2. returns the power of a number.
   Two parameters: base and exponent.
   Returns: $base^{exponent}$.

   Tips: $base^{exponent} = e^{exponent \times ln(base)} =$ exp(exponent * log(base))

Conditions

Logical tests

==	Are both values equal.
$>$ or $\geq$	Is the left value greater (greater or equal) than the right value.
$<$ or $\leq$	Is the left value smaller (smaller or equal) than the left value.
!	Is a NOT operator that negates the value of a test.
|	Is an OR operator used to combine logical tests. Returns TRUE if either are TRUE.
&	Is an AND operator used to combine logical tests. Returns TRUE if both are TRUE

Example

```
test <- 2 + 2 == 4     ## (TRUE)
test && !test          ## (FALSE)
test || !test          ## (TRUE)
```

Conditions

Boolean

Any logical tests can be vectorized (compare 2 vectors).

|	Is a OR operator for vectorized application.
&	Is an AND operator for vectorized application.
which	Returns the index of the vectors with *TRUE* values.
any	Take a vector of *logical* and return *TRUE* if at least one value is *TRUE*.
%in%	Vectorized any. See example/supp material.

Example

```
c(TRUE, TRUE) & c(TRUE, FALSE) -> TRUE, FALSE
which(5:10 == 6)
any(9>1:10)
any(9>1:10 & 8<=1:10)
luckyNumbers[which(luckyNumbers %in% c(16,42,-66.6))]
```

Conditions - Exercise

Create a function that:

1. remove values below 3 from a vector.
2. remove values below a specified threshold from a vector.

Testing conditions

if else

Test if a condition, if *TRUE* run some instruction, if *FALSE* something else (or nothing).

```
if( Condition ){
...   Instructions
}
```

Example

```
if(length(luckyNumbers)>3){
  cat("Too many lucky numbers.\n")
  luckyNumbers = luckyNumbers[1:3]
} else if(length(luckyNumbers)==3){
  cat("Just enough lucky numbers.\n")
} else {
  cat("You need more lucky numbers.\n")
}
```

if else - Exercise

Create a function that classify the average value of a vector. It returns:

- *low* if the average if below 3.
- *medium* if the average if between 3 and 7.
- *high* if the average if above 7.

Loops

for loops

Iterate over the element of a container and run instructions.

```
for(v in vec){
...  Instruction
}
```

while loops

Run instructions as long as a condition is *TRUE*.

```
while( CONDITION ){
...  Instruction
}
```

Example

```
facto = 1
for(n in 1:10){
   facto = facto * n
}
```

Loops - Exercise

Write a function that computes the mean values of the columns:

1. using the apply function.
2. using a for loop.
3. (using a while loop.)

Import/export data

R objects

save Save R objects into a file. Usual extension: *.RData*. file= parameter to specify file name.

save.image Save the entire R environment.

load Load R objects from a (*.RData*) file. verbose to print the names of the objects loaded.

Example

```
save(luckyNumbers, tenOnes, mat, file="uselessData.RData")
load(file="uselessData.RData")
load(file="dataForBasicPlots.RData",verbose=TRUE)
```

Import/export data - Text files

Easy but important

- What data structure is the more appropriate ? vector, matrix ?
- Does R read/write the file the way you want ?
- The extra parameters of the functions are your allies.

read.table

To read a data.frame from a multi-column file.

file=	the file name.
header=	*TRUE* use the first line for the column names. Default: *FALSE*.
as.is=	*TRUE* read the values as simple type, no complex type inference, **recommended**. Default: *FALSE*.
sep=	the *character* that separate each column. By default, a white-space or end of line.

Example

```
input.data = read.table("fileToRead.txt", as.is=TRUE,
                                          header=TRUE, sep="\t")
```

Import/export data - Text files

write.table

To write a data.frame in a multi-column file.

df	the matrix or data.frame to write.
file=	the file name.
col.names=	*TRUE* print the column names in the first line. Default: *TRUE*.
row.names=	*TRUE* print the rows names in the first columns. Default: *TRUE*.
quote=	*TRUE* surround character by quotes(″). Default: *TRUE* → messy.
sep=	the *character* that separate each column. By default, a white-space.

Example

```
write.table(resToWrite, file="fileToRead.txt", col.names=TRUE,
                          row.names=FALSE, quote=FALSE, sep="\t")
```

Basic plotting

hist

Plot the value distribution of a vector.

x The vector with the values to plot.

plot

Plot one vector against the other.

x The first vector to plot. *x-axis.*

y The second vector to plot. *y-axis.*

type How the points are plotted. "p" as points, "l" joined by lines.

Example

```
hist(mat.ge[,1])
plot(mat.ge[,1],mat.ge[,2])
```

Basic plotting

Common parameters

main=	A title for the plot.
xlab=/ylab=	A name for the x/y axis.
xlim=/ylim	A vector of size two defining the desired limit on the x/y axis.

Example

```
hist(mat.ge[,1],main="A basic graph",
                xlab="first column values")

plot(mat.ge[,1],mat.ge[,2],main="Another basic graph",
  xlab="first column values",ylab="second column values")
```

Basic plotting

Extra parameters

col	the colour of the points/lines. 1:black, 2:red, ...
pch	Shape of the points. 1:circle, 2:triangle, ...
lty	Shape of the lines. 1:plain, 2:dotted, ...

Extra functions

lines	Same as plot but super-imposed to the existent one.
abline	Draw vertical/horizontal lines.

Example

```
plot(mat.ge[,1],mat.ge[,2],main="Another basic graph",
  xlab="first column values",ylab="second column values")
lines(mat.ge[,1],mat.ge[,3],type="p",col=2,pch=2)
abline(h=0,lty=2)
```

Basic plotting

boxplot

Plot the distribution (quantiles/median/outliers) of variables.

x The matrix (or list) of distributions

Example

```
boxplot(mat.ge)
```

Basic plotting - Exercise

Plot:

1. the distribution of the average gene(row) expression. Add a vertical dotted line to mark their average value.
2. the expression(row) of *gene333* against *gene666*. Superimpose in red triangles the expression(row) of *gene333* against *gene667*.
3. the distribution of the first 10 samples(column) as a box plot.

Type coercion.

- Automatic conversion of an object to another type, e.g numeric→character, logical→numeric.
- Awareness for debugging.
- Useful sometimes.

Example

```
is.numeric( c(1:10,"eleven") )

logical.vector = c(TRUE,TRUE,FALSE,TRUE,FALSE)
sum(logical.vector)
mean(logical.vector)
```

character operations

paste	Paste several *character* into one.
grep	Search a pattern in a vector and return the index when matched.
grepl	Search a pattern in a vector and return *TRUE* if found.
strsplit	Split *character* into several.

Example

```
sample.name = "Ob5cU8eN4mE"
file.name = paste("pathToYourDirectory/greatAnalysis-",
                                    sample.name,".txt",sep="")

which(sample.names=="controlA" & sample.names=="controlB")
grep("control",sample.names)
```

Save your plot into a *pdf/png*

Open a connection to a output file, plot as usual, close the connection.

pdf	Open the connection to a *pdf* output.
png	Open the connection to a *png* output.
dev.off()	Close the connection

Example

```
pdf("myNicePlot.pdf")
plot(...)
dev.off()
```

One-liner quiz

Instructions

Write R command to address each question. Only one-line command allowed. The shorter the better.

Questions

1. From a matrix of *numeric*, compute the proportion of columns with average value higher than 0.
2. From a matrix of *numeric*, print the name of the columns with the highest value.
3. From a matrix of *numeric*, print the rows with only positive values.
4.

Thank you !!

If you're interested in potentially **more sessions**, in different format (**more often, more specific**), maybe some kind of **R club**, let us know through the **survey** or by email.

jean.monlong@mail.mcgill.ca