R

Breaking the ice

Xavier Thibert-Plante* [1]

*McGill University

December 3, 2009

[1] xavier.thibert-plante@mail.mcgill.ca

Introduction

What you won't learn today

- Statistics
- Experimental design

What you will learn today

- Basic usage of high level programming
- Tool to learn more on your own

Introduction

What R is not going to do for you

- Not useful to enter data
- Will not tell you if you are using the right statistical test

What R is going to do for you

- Perform statistical test
- Plot figures

Introduction

Installing R

- Go to web page
  - http://www.r-project.org/
- Download proper version
  - Download CRAN (left frame)
  - Choose mirror (the closer the better)
  - Select proper operating system
- Install it
- Open it

Introduction

Quit

```
> q()
```

or

```
> quit()
```

You can save your session (variables and function and continue later)

Introduction

Setup

- Create a directory with <yourName> on the desktop
- Open R
- Change the directory to <yourName> directory by using File->Change dir
- Open a text editor, such as Notepad, to put your command there first.

Introduction

R graphical interface

- R is a command line interface, your mouse is useless here.
  - Advantage: batch files
  - Inconvenient: when you don't know what to type you feel pretty lonely

Variables

- One value

  ```
  > a <- 1
  ```

  equivalent to a=1

  ```
  > 1 -> a
  ```

- A vector

  ```
  > b <-c(1,2,3)
  ```

Variables

Empty

- Vector

```
> a <- array(NA,dim=10)
> a[4] <- 5
```

- Matrix

```
> b <- matrix(NA,ncol=10,nrow=30)
> b[30,3]<-1
```

Variables

Help

RTFM

```
> help(array)
> help(matrix)
```

Variables

Generating vector and matrix

- Sequence vector

```
> a<-array(seq(1,10,2))
```

- Random vector

```
> a<-array(rnorm(10,mean=15,sd=3))
```

- Sequence matrix

```
> b<-matrix(seq(1,20),ncol=2,nrow=10)
```

- Random matrix

```
> b<-matrix(runif(21),ncol=3,nrow=7)
```

Variables

Simple arithmetic operations

```
> a+a
> a+5
> 1+b
> 5*a
> a*a
```

```
> a*b
> b*b
> a-a
> a-5
> a/2
```

Variables

Structure

- x is a matrix with column and lines

```
x[,1] # refers to the first column
x[,2] # refers to the second column
x[1,] # refers to the first line
```

- x[line,column]

Data

Read the data

Go to my web page

- http://sites.google.com/site/xavierthibertplante
- R workshop
- Download the database into <yourName> folder

Data

Modifying the database

- Remove special character (#$%&?+=-)
- Make sure that the first line is the title of the column without space (’colOne’ vs ’col one’)
- Save as csv (Coma Separated Variable)

Data

Entering your data

- Use excel spreadsheet
- Save as csv (coma separated variable)
- Look at the csv file in a text editor, such as Notepad
- One column must have only one type of cell: number, except for the first one, sometime.
- Go back to the webpage http://sites.google.com/site/xavierthibertplante and download csv version

Data

Load the data

- Open R

```
> x<-read.csv('hendryetAl.csv')
```

- We can now play with the database with the variable x

Data

Structure

- x is a matrix with column and lines

```
x[,1] # refers to the first column
x[,2] # refers to the second column
x[1,] # refers to the first line
```

- x[line,column]

Data

Sanity check

- Number of line in the data before and after loading

```
> length(x[,1])
```

- Number of column:

```
> length(x[1,])
```

Data

Column name

- Syntax: <variableName>$<columnName>

  ```
  > x$Years
  ```

  instead of

  ```
  > x[,18]
  ```

- Note that it is case sensitive:

  ```
  > x$years
  ```

  will not work

Data

Your data set in R

- First line: column name (no space or special character in the name)
- Each column is of one type
- Save as 'csv'
- Look at your file in a text editor (note the separation ";" or "," and the decimal point "." or ",")
- Adapt the option of `read.csv` function
  `> help(read.csv)`
- Load your data
  `yourName<-read.csv('fileName.csv')`
- Test length and names of columns

One variable manipulations

Simple plot

- Reload the database

```
> x<-read.csv('hendryetAl.csv')
```

- Histogram of the Haldanes

```
> hist(x$Haldanes)
```

- Change number of bars

```
> hist(x$Haldanes,breaks=100)
```

One variable manipulations

Test of normality

- Shapiro-Wilk normality test

```
> shapiro.test(x$Haldanes)
```

- Kolmogorov-Smirnov test

```
> ks.test(x$Haldanes,'pnorm',
  mean=mean(x$Haldanes,na.rm=T),
  sd=sd(x$Haldanes,na.rm=T))
```

- Give numbers, not its meaning

Two variables manipulations

Plot

- Haldanes as a function of the years

```
> plot(x$Years,x$Haldanes)
```

- With labels

```
> plot(x$Years,abs(x$Haldanes),
xlab='Years', ylab='Haldanes',
main='Absolute values')
```

- With limit on the axis

```
> plot(x$Years,abs(x$Haldanes),
xlim=c(0,100), ylim=c(0,2))
```

Two variables manipulations

Subset of data

```
> x$Years
> x$Years>100
> x$Years==111
> x$Years!=124
> cond<-x$Years>100
> x$Years[cond]
> x$Haldanes[cond]
> x$Haldanes[x$Years>100]
```

Two variables manipulations

List of conditions

>	Greater
>=	Greater or equal
<	Smaller
<=	Smaller or equal
==	Equal
!=	Not equal
&	And
|	Or

Two variables manipulations

Subset of data

```
> cond<-x$Years>100 & x$Years<113
> cond<-x$Years<100 | x$Years>113
> cond<-x$Years>100 & x$Haldanes>=0
> x$Years[cond]
> x$Haldanes[cond]
```

Two variables manipulations

More on the plot

- Differentiate the direction

```
> y<-array(1,length(x$Haldanes))
> y[x$Haldanes<0]<-2
> plot(x$Years,abs(x$Haldanes),col=y,
xlab='Years',ylab='Haldanes')
> legend('topright',
c('positive','negative'), col=c(1,2),
pch=c(1,1))
```

Save information

Figures

- Lazy way: click file-> save as -> jpeg
- Prefered option

```
> jpeg('fileName.jpg')
```

- More option with the command line

```
> help(jpeg)
```

Save information

Variables

```
> help(save)
> save(x,y,z,file='saveXYZ.RData')
```

The whole workspace

```
> save.image(file='workspace.RData')
```

Save information

Get the infomation back

```
> load('saveXYZ.RData')
> load('workspace.RData')
```

What was loaded

```
> ls()
```

Linear models

- Haldane function of generation length?
- Syntax: `x$Haldanes~ x$GLength`

```
> lm(x$Haldanes~ x$GLength)
```

- More information:

```
> summary(lm(x$Haldanes~ x$GLength))
```

Linear models

How it look like?

```
> plot(x$Haldanes~ x$GLength)
```

This is equivalent to

```
> plot(x$GLength,x$Haldanes)
```

Linear models

- Absolute values of Haldanes function of generation length?

```
> lm(x$HaldanesAbs~ x$GLength)
```

- More information:

```
> summary(lm(x$HaldanesAbs~ x$GLength))
```

Linear models

How it look like?

```
> plot(x$HaldanesAbs~ x$GLength)
```

- We want the regression line on the graph
  ```
  > help.search('regression')
  ```
- Google : R regression line plot
- The function is not in the base package!

New packages

Install

```
> install.packages()
```

- Select somewhere close (Canada or USA)
- Select the package you want (car)

New packages

Loading and using the new library

```
> library(car)
> help(regLine)
> rg<-x$HaldanesAbs~ x$GLength
> plot(rg,xlab='Glength',ylab='Absolute
Haldanes')
> regLine(lm(rg))
```

Produce a linear model from your data

Linear models

Your turn

```
> lm(yourName$a~ yourName$b)
> summary(lm(yourName$a~ yourName$b))
```

Linear models

More factor

- Two factor

```
> m1<- x$HaldanesAbs~ x$GLength+x$Years
```

- Interaction term

```
> m2<- x$HaldanesAbs~ x$GLength:x$Years
```

- Two factor + interaction term:

```
> m3<- x$HaldanesAbs~ x$GLength*x$Years
```

equivalent to:

```
> m3<- x$HaldanesAbs~ x$GLength +
x$Years + x$GLength:x$Years
```

Produce a more complicated linear model from your data

ANOVA

Setup

- Look at the file `CH25PR07.txt` in a text editor
- Read a table

```
> z<-read.table('CH25PR07.txt')
```

- Give names to the column

```
> names(z) <- c('response', 'category',
'replicat', 'coVar')
```

- Shortcut to column name

```
> attach(z)
> response
> detach(z)
> response
> attach(z)
```

ANOVA

Nominal term

- Everything is considered numeric as default
- Define the categorie RDexp: nominal

```
> category<-factor(category)
```

ANOVA

First steps

- Write the model

```
> mod1<-response~ category
```

- Take a look at the model

```
> boxplot(mod1)
```

- Linear model of the data

```
> mod1.lm<-lm(mod1)
```

- Vizualize the model

```
> plot(mod1.lm)
```

- Get the information out of the model

```
> summary(mod1.lm)
```

- Perform the ANOVA

```
> anova(mod1.lm)
```

ANCOVA

Models

- Sanity check:
```
> as.factor(category)
```
- Look at the data:
```
> plot(response~
coVar,pch=as.numeric(category))
```
- Full model:
```
> ResFull<-response~ category+ coVar +
category:coVar
```
- Common slope, different intercept:
```
> ResCD<-response~ category+ coVar
```
- Everything in common:
```
> ResE<-response~ coVar
```

ANCOVA

Models

- Linear model

```
> ResFull.lm<-lm(ResFull)
> ResCD.lm<-lm(ResCD)
> ResE.lm<-lm(ResE)
```

- Look at the models

```
> plot(ResFull.lm)
> plot(ResCD.lm)
> plot(ResE.lm)
```

ANCOVA

Analysis

- Get the infomarion from the models:

```
> summary(ResFull.lm)
> summary(ResCD.lm)
> summary(ResE.lm)
```

- ANCOVA

```
> anova(ResFull.lm, ResE.lm)
> anova(ResFull.lm, ResCD.lm, ResE.lm)
```

ANCOVA

Useful command

- Verify the hypothesis of equal variance within group

```
> tapply(response, category, var,
na.rm=TRUE)
```

- Verify the hypothesis of normality in a group

```
> tapply(response, category,
function(x) shapiro.test(x))
```

Simple Programming

- Loops:
  for
  while
- Conditions:
  if
- No help!

Simple Programming

Loops

```
> for (dummyVariable in array){
>     operation
> }
```

or

```
> while (condition){
>     operation
>     change the condition
> }
```

Simple Programming

Loop example

Fibonacci

```
> x<- array(1,dim=10)
> for (i in seq(3,length(x))) {
>     x[i]<-x[i-1]+x[i-2]
> }
```

or

```
> x<- array(1,dim=10)
> i<-3
> while (i <= length(x)) {
>     x[i]<-x[i-1]+x[i-2]
>     i<-i+1
> }
```

Simple Programming

Condition

```
> if (condition) {
>     operation 1
> } else {
>     operation 2
> }
```

Simple Programming

List of conditions

>	Greater
>=	Greater or equal
<	Smaller
<=	Smaller or equal
==	Equal
!=	Not equal
&	And
|	Or

Simple Programming

Condition example

```
> if (x>0) {
>     x<-x+1
> } else {
>     x<-x-1
> }
```

Simple Programming

Your first program: chaos

- Logistic equation: $x_{t+1} = r \times x_t(1 - x_t))$
- Run simulation of 1000 generations
- Set your initial population size
- Try different r values between 3 and 4
- Plot the time serie

Simple Programming

Chaos solution

```
> x<-array(dim=1000)
> x[1]=0.5
> r<-3.7
> for (t in 2:length(x)){
>    x[t]<-r*x[t-1]*(1-x[t-1])
> }
```

Functions

Your first function

You want to be able to change you parameter faster: write a function

Functions

Before

```
> x<-array(dim=1000)
> x[1]=0.5
> r<-3.7
> for (t in 2:length(x)){
>    x[t]<-r*x[t-1]*(1-x[t-1])
> }
```

Functions

After

```
> chaos <- function(time,r,popS){
>     x<-array(dim=time)
>     x[1]=popS
>     for (t in 2:length(x)){
>         x[t]<-r*x[t-1]*(1-x[t-1])
>     }
>     return(x)
> }
```

Functions

Callign your function

```
> chaos(1000,3.5,0.5)
> c<-chaos(1000,3.5,0.5)
> plot(c)
> plot(chaos(1000,3.5,0.5))
```

Functions

General

All procedure that you will repeat more than once: do a function of it

Conclusion

- RTFM
  `help(functionName)` and
  `help.search("what your looking for")`
- Text editors are your best compagnions
  - Verify the format of your data csv and others
  - To write down the command BEFORE you put them in the R console
- You will NEVER screw up your data in R if you load them from a file

Acknowledgement

- Thanks you!