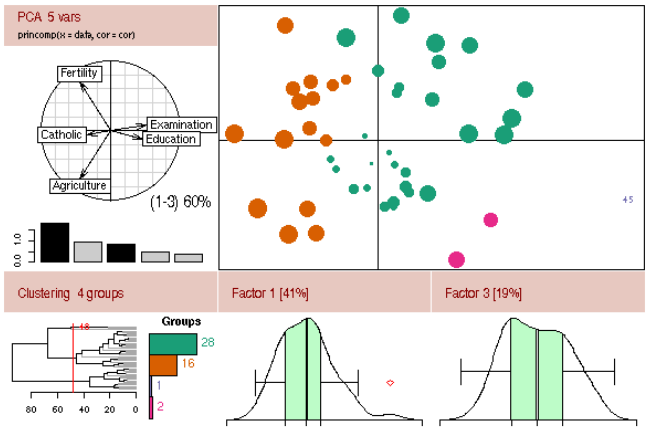
An Introduction to R - I

Objects and data

Olivier Flores

CIRAD, Montpellier

December 7, 2006



What is R?

An integrated suite of software facilities for:

- data manipulation,
- calculation (linear algebra,. . .),
- graphical display,
- programming with a simple, interpreted and object-oriented language: S,
- statistical analysis *via* pre-defined or user-defined routines (also available in *packages*).

Free and *open-source* software!!

Generalities I

- Graphical interface (console) with a *command-line* waiting for the user to interact ($\neq$ *point & click*)!
  > ...
- All things = *objects* (data, graphs, function, results,...) with different modes ("numeric", "character",...) and attributes (length, names,...)
- Objects currently used are stored in a *workspace* which can be saved as a `.Rdata` file and called back during another session

Generalities II

- Case-sensitive language : A $\neq$ a
- Commands are *expressions* (`> a`)
  or *assignments* (`> a <- 1` or `> a = 1`),
- You can recall commands, for correction or re-use
  (arrows $\uparrow$ and $\downarrow$ on the keyboard)
- All typed commands are stored in a *history* which can be recalled

Set up things

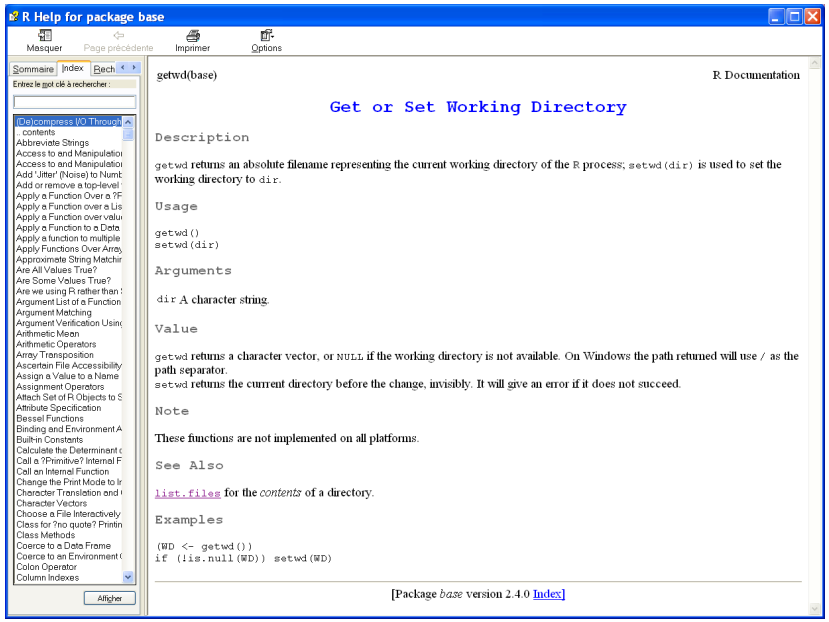
- Download R from the CRAN (Comprehensive R Archive Network) at `http://cran.r-project.org/`,
- Install R (WINDOWS),
- Launch R (shortcut to Rgui.exe),
- Check and change working directory:

```
> getwd()
[1] "C:/Program Files/R-2.4.0"
> setwd("D:/MyDir")
```

Getting help I

R comes with very detailed help information on objects and the system

- Manuals available on the CRAN website,
- Documentation available from the command line: Description, Usage, Arguments, Value, See Also, **Examples**,...




```
> help(getwd)   or   > ?getwd
```

- Search for a character string in all help documentation:
  `> help.search("directory")`
- For special characters, needs quotation: `> ?"["`

Useful commands I

- `ls()`: lists all objects of the current workspace
- `rm(object1,object2,...)`: remove objects of the workspace (definitely)
- `save(objects,file)`: save objects in the specified *file*
  `save.image()`: save all objects of the workspace in a `.Rdata` file in the working directory
- `load(file)`: load objects contained in *file*, such as a previously saved `.Rdata` file :
- `history(number)`: list the last *number* typed commands
- `q()`: prompt for save and quit R

Objects in R

- Objects in R are characterized by two intrinsic attributes:
  a **mode** and a **length**
- Objects can have non-intrinsic attributes (dimension of matrices,...)

```
attributes(object);
attr(object, name);
attr(object, name) = value
```

- The *class* is a special attribute all objects have:

```
class(object)
```

  Functions behave differently depending on the class:

```
print(object); plot(object);...
```

Objects in R

Objects can be composed of elements of one or several modes

Objects	Mode	Several mode allowed
vector	numeric, character, complex or logical	No
factor	numeric, character	
matrix	numeric, character, complex or logical	
list	numeric, character, complex or logical	Yes
data.frame	numeric, character, complex or logical	

Matrices are a particular type of *array*
(*array* = table with n dimensions, $n = 2$ for matrices)

Functions

- Functions are particular objects of class *function*
- Functions require **arguments**, some of which must be specified (*mandatory*) while others are *optional*
- Arguments can have **default** values which are used when not specifically defined by the user
- Some functions can be called without arguments (`ls()`, `...`)
- Functions can return objects (the *value* of the function) of various classes

All these information and more can be found in the help documentation.

Simple example

- ```
  > x=2
  > mode(x)
  [1] "numeric"
  > length(x)
  [1] 1
  ```
- Check mode: `is.numeric(x)`, `is.character(x)`,...

  ```
  > is.numeric(x)
  [1] TRUE
  ```
- Transform mode: `as.numeric(x)`, `as.character(x)`,...

  ```
  > as.character(x)
  [1] "2"
  ```
  ```

Vectors and factors I

1. **Vectors**
   - concatenate elements of **one** mode with the command `c()`:
     ```
     > x=c(1,2,3)
     > y=c("Age","Size","Weight","Color")
     ```
   - use specific commands:
     ```
     > x=vector("numeric",length=3); y=character(4)
     ```
   - check mode and transform into vector:
     `is.vector(object); as.vector(object)`

Vectors and factors II

2 Factors

- = vectors of categorial variables with different levels (*ordered* or *unordered*)

```
> x = as.factor(c(1,2,3)) or factor(c(1,2,3))
> x
[1] 1 2 3
Levels:  1 2 3
```

- check mode and transform into factor: `is.factor(object)`; `as.factor(object)`
- check or set the levels of a factor:

```
> levels(x)
> levels(x) = c("a","b","c")
```

Some manipulations

- **Numbers**: x = 2; y = 3;
  classical operations: `x+y; x-y; x*y; x/y, x^y,...`
  functions: `log(x); cos(x);...`
  comparisons:

```
> x==y
[1] FALSE
```

  also `x!=y; x<y;...`

- **Vectors**: `x=c(1,2,3); y=c(4,5,6);`
  1. element by element operations : `x+y; x*y; x^y; log(x);...`

```
> x*y
[1] 4 10 18
```

  2. operations on one vector: `sum(x); length(x); min(x); max(x); range(x); mean(x); sd(x);...`
  3. extracting element *i* of x: `> x[i]`

```
> x[2] = x[3]   # put the third element in the second
```

- **Factors**: no arithmetic operations on factors

Matrices

Matrices are collection of vectors with an attribute *dimension* which is a vector of length 2: [number of rows, number of columns]

Basic commands: `dim; nrow; ncol; names; dimnames; colnames; rownames;`

```
> x=c(1,2,3)
> matrix(x, nrow=1, ncol=3, byrow=FALSE, dimnames=NULL)
> matrix(1:6, 2, 3)
```

```
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

```
> matrix(1:6, 2, 3, byrow=TRUE)
```

```
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

Operations on Matrices I

- Element by element: `X+Y; X-Y; X*Y; X^Y; log(X);...`

```
> X = matrix(1:6,2,3); Y = matrix(c(5,9,6,3,1,2),2,3)
> Y-X
```

```
     [,1] [,2] [,3]
[1,]    4    3   -4
[2,]    7   -1   -4
```

- Matrix product and other operations: `X%*%Y; t(X)`
- Extraction:
  - element (i,j): `X[i,j]`
  - row i: `X[i,]`
  - column j: `X[,j]`

```
> X = matrix(1:6,2,3)
> X[1,3]
[1] 5
```

Operations on Matrices II

- Add rows or column to an existing matrix: `rbind` or `cbind`

```
> X=matrix(1:4,2,2)
```

```
> rbind(X,X)
     [,1] [,2]
[1,]    1    3
[2,]    2    4
[3,]    1    3
[4,]    2    4
```

```
> cbind(X,X)
     [,1] [,2] [,3] [,4]
[1,]    1    3    1    3
[2,]    2    4    2    4
```

- Apply function on rows or columns: `apply(X, MARGIN, FUN)`

```
> apply(X,1,sum)
[1,] 4 6
> apply(X,2,sum)
[1,] 3 7
```

Lists

- An R **list** is an ordered collection of objects known as its *components*.
- Components can be objects of all types: vectors, matrices, lists,...
- Components may be named.

• List without name

```
> L1=list(c(1,2),X)
> L1
[[1]]
[1] 1 2

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

• List with names

```
> L2=list(A=c(1,2), B=X)
> L2
$A
[1] 1 2

$B
     [,1] [,2]
[1,]    1    3
[2,]    2    4
```

Operation on lists I

- Names of a list

```
> names(L1)   > names(L2)
[1] NULL      [1] "A" "B"
```

- Extract one element

```
> L1[[1]]   > L2$A
[1] 1 2     [1] 1 2
```

!!! L1[[1]] $\neq$ L1[1] !!!

- Add components, concatenate lists

```
> L1[[3]]="one more";
L3 = c(L1, b = 10); L4 = c(L1, L2)
```

Operation on lists II

- Apply a function to the components of a list:
  `lapply(X, FUN)` (see also *sapply*)

```
> lapply(L1,sum)
[[1]]
[1] 3
[[2]]
[1] 10
```

- Transform lists in vectors
  (only if all components share the same mode):
  `unlist(list)`

Data frames

A *data frame* is a special type of list displayed in matrix form.
Components (= columns of the matrix) can have different modes and attributes but must have the same length (if not, elements are recycled).

```
> x = 1:2; y = "a"; z = 1:3
> data = data.frame(count = x, level = y)
[1]
      count  level
 [1,]     1    "a"
 [2,]     2    "a"
> data = data.frame(x,z)
Error in data.frame(x, y) :  arguments imply differing
number of rows:  2, 3
```

Operations on data frames

- Extraction:
  - element $[i, j]$: `data[i, j]`
    `data[2, 1] = data$count[2] = 2`
  - row i: `data[i, ]`
  - column j: `data[, j]` or `data[[j]]`
    `data[, 2] = data$level = c("a","a")`
- All operations on matrices if all elements are numeric, but results will be a matrix, not a data frame
- All operations on lists but acting on the columns only (components of the list).

Indexing and sequences I

1. **Indexing**

   → Define indices to extract *subsets* of an object along a dimension: *length>1* for vectors and lists, *rows* or *columns* for matrices and data frames.

```
x = c(5,3,-8,2,-10,-15,21)
```

   - given a set of indices:

```
> x[c(1, 4, 5)]
[1] 5 2 -10
```

   - using a boolean expression *bool* with correct length:

     `x[bool]` : all elements of x that verify the condition `bool`

```
> x[x>0]
[1] 5 3 2
```

```
> data[data$count == 2,]
  count level
2     2     a
```

   - remove elements:

```
> x[-length(x)]      # remove last element
[1] 5 3 -8 2 -10 15
```

Indexing and sequences II

2 Sequences and replicates

- Simple sequences: *integer1*:*integer2*

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

- More complex: **seq(from, to, [by= , length=])**

  - Odd and even numbers:

```
> seq(1, 9, by=2); > seq(2, 10, by=2)
```

  - Fixed length:

```
> seq(10, 100, length=10)
[1] 10 20 30 40 50 60 70 80 90 100
```

  → *Useful to extract subsets of vectors or tables*

- Replicate numbers: **rep(x, times, each, . . .)**

```
> rep(1:5, times = 2)          > rep(1:5, each = 2)
  [1] 1 2 3 4 5 1 2 3 4 5        [1] 1 1 2 2 3 3 4 4 5 5
```

  → *Useful to construct factors*

More useful commands

1. Manipulate strings
   - Find strings: `grep; match`
   - Extract and/or replace: `sub; substr`
   - Concatenate (vectors transformed in) strings: `paste(..., sep = " ", collapse = NULL)`
2. Manipulate booleans (conditions)
   - Test elements: `%in%; any; all`
   - Locate elements: `which`
3. General
   - `summary(`*`object`*`)`: display information depending on the class of *`object`*
   - `print(`*`object`*`)`: print *`object`* on the console
   - `unique; sort; order; rank; rev`

Import and export data I

1. Make data frames from files (delimited *.txt, .csv, .xls, .dat,...*)

```
> read.table(file, header=FALSE, sep = "", dec = ".",
na.strings = "NA", nrows = -1,...)
```

   - *file*: name of the file to be read,
   - *header*: does the file contain variable names as its first line?
   - *sep*: character separating fields in the table (default "" = white space);
   - *dec*: characeter used for decimal points;
   - *na.strings*: vector of strings interpreted as "NA" (non arithmetic values, missing data);
   - *nrows*: number of rows to be read;

See the help manual for additional parameters: `?read.table`
See also the command **scan** for reading vectors

Import and export data II

2. Write data in files

```
> write.table(x, file = "", append = FALSE, sep = " ",
na = "NA", row.names = TRUE, col.names = TRUE,...)
```

- *x*: object to be written in *file*, converted in data frame,
- *append*: *file* overwritten (default) or output at the end of *file*,
- *sep*: string to separate fields,
- *na*: string to be written in place of NA values,
- *row.names*: logical value (if TRUE, row names are be written along *x*), or vector of strings written as names for rows,
- *col.names*: similar to *row.names* for columns

See also the command `cat`:

```
cat(...  , file = "", sep = " ", fill = FALSE, labels =
NULL, append = FALSE)
```

References

This presentation is largely inspired by the manual:

An Introduction to R.
Notes on R: A Programming Environment for Data Analysis and Graphics
Version 2.4.0, 2006-10-03
by W. N. Venables, D. M. Smith and the R Development Core Team