R tutorial

13.02.07

Lorenza Bordoli

R commands are in written in red

Fundamentals of the R language

R as a calculator

- Calculator
  - +, -, /, *, ^, log, exp, …:

```
> (17*0.35)^(1/3)

> log(10)

> exp(1)

> 3^-1
```


Assigning Values to variables

- Variables are assigned using '<-':

```
> x<-12.6
> x
[1] 12.6
```

- Variables that contains many values (vectors), e.g. with the **concatenate** function:

```
> y<-c(3,7,9,11)
> y
[1]  3  7  9 11
```

Assigning Values to variables

- Type the numbers in at the keyboard using the `scan()` function:

```
> z<-scan()
1: 8
2: 4
3:
Read 3 items
> z
[1] 8 4
```

- Operator ' : ' means "a series of integers between":

```
> x<-1:6
> x
[1] 1 2 3 4 5 6
```

- Series in non-integer steps (e.g. 0.1) using the `seq()` function :

```
> b<-seq(0.5,0,-0.1)     : negative values for decreasing series
> b
[1] 0.5 0.4 0.3 0.2 0.1 0.0
```


Generating Repeats

- The `rep` function replicates the first argument by the number of times specified in the second argument:

```
> rep("A",10)
[1] "A" "A" "A" "A" "A" "A" "A" "A" "A" "A"
```

- Repeated series:

```
> rep(1:6,2)
[1] 1 2 3 4 5 6 1 2 3 4 5 6
```

- Elements of a series to be repeated:

```
> rep(1:6,rep(3,6))   : vector of the same length (second argument)
[1] 1 1 1 2 2 2 3 3 3 4 4 4 5 5 5 6 6 6
```

- To specify each repeat separately:

```
> rep(c(4,7,1,5),c(3,2,5,2))
[1] 4 4 4 7 7 1 1 1 1 1 5 5
```

Generating Factor Levels

- `gl` function for generating levels of factors ("up to" and "with repeats of"):

```
> gl(5,3)
[1] 1 1 1 2 2 2 3 3 3 4 4 4 5 5 5
Levels: 1 2 3 4 5
```

- To repeat the whole pattern, specify the total length:

```
> gl(5,3,30)
[1] 1 1 1 2 2 2 3 3 3 4 4 4 5 5 5 1 1 1 2 2 2 3 3 3 4 4
   4 5 5 5
Levels: 1 2 3 4 5
```


Reading Data from a File

GUI

- The `read.table` function reads data from a file: 

```
> dataframe<-read.table(("C:\\Documents and
  Settings\\plotdata.txt",header=T)
```

- header = T : row 1 of the file contains variable names

- To use the variables you need to use the `attach`:

```
> attach(dataframe)
```

- To see the names of the variables:

```
> names(dataframe)
[1] "xvalues" "yvalues"
```

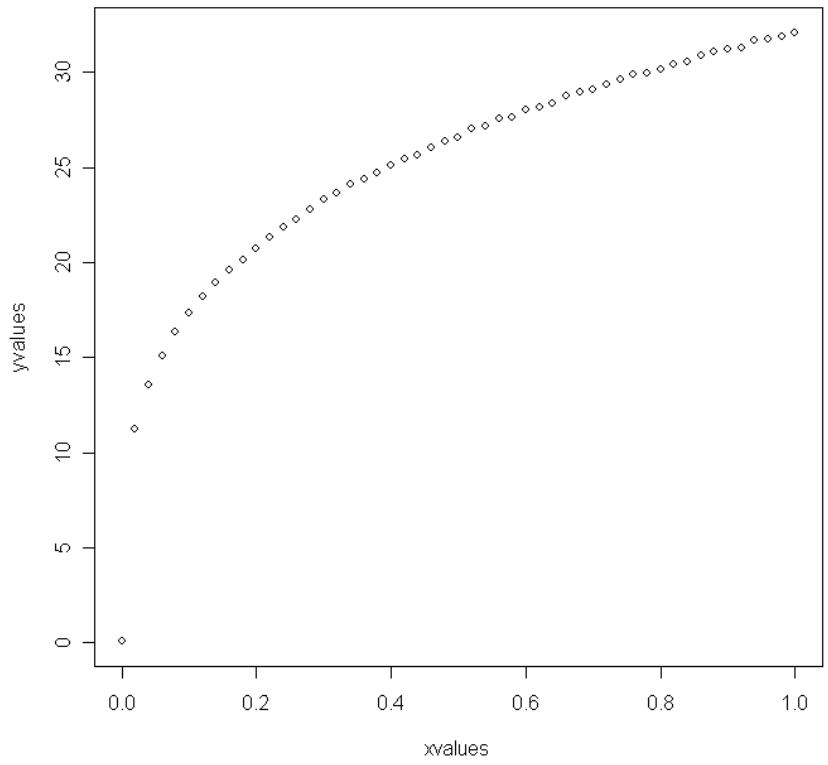
- Simplest plot:

```
> plot(xvalues,yvalues)
```

Example of a data file

Example file: plotdata.txt

xvalues	yvalues
0	0.062731954
0.02	11.24579655
0.04	13.58265422
0.06	15.10266693
0.08	16.32155649
0.1	17.34692739
0.12	18.21163282
0.14	18.95004852
0.16	19.57302982
0.18	20.14684122
0.2	20.73519153
0.22	21.34161004
0.24	21.86545585
0.26	22.26747557
0.28	22.80085017
0.3	23.31319749
0.32	23.66159626
0.34	24.10097538
0.36	24.4025692



Changing the Look of Graphics (I)

- The most likely change: orientation and size of labels of x and y axes:

```
> plot(xvalues,yvalues, ylab = "Label for y axis",
  xlab = "Label for x axis", las = 1, cex.lab = 1.5)
```

- `ylab`, `xlab` : changes the annotation of the axis labels;
- `las` : numeric in {0,1,2,3} change orientation of the axis labels;
- `cex.lab` : magnification to be used for x and y labels;

- To get full range of changes about graphical parameters:

```
>?par
```


Vector Functions in R

- Typical operations on vectors include summary statistics (`mean`, `var`, `range`, `max`,…):

```
> y<-c(5,7,7,8,2,5,6,6,7,5,8,3,4)
> z<-13:1
> mean(y)
[1] 5.615385
> var(z)
[1] 15.16667
```

- Arithmetic with entire vectors, e.g. * operator. In R if two vectors are not the same length, the shorter vector is repeated as necessary, up to the length of the longer vector:

```
> y*6
[1] 30 42 42 48 12 30 36 36 42 30 48 18 24
```

- Join together two vectors using the concatenate function `c`:

```
c(y,z)
```

Subscripts: Obtaining Parts of Vectors

- Elements of vectors by subscripts in []:

```
> y[3]
```

- The third to the seventh elements of y:

```
> y[3:7]
```

- The third, fifth, sixth and ninth elements:

```
> y[c(3,5,6,7)]
```

- To drop an element from the array, use negative subscripts:

```
> y[-1]
```

- To drop the last element of the array without knowing its length:

```
> y[-length(y)]
```


Subscripts as Logical Variables

- Logical condition to find a subset of the values in a vector:

```
> y[y>6]
```

- To know the values for `z` for wich `y>6:`

```
> z[y>6]
```

- Element of `y` not multiples of three:

```
> y[y%%3!=0]
```

Subscripts with Arrays (I)

- Three-dimensional array containing the numbers 1 to 30, with five rows and three columns in each two tables:

```
> A<-array(1:30,c(5,3,2))
> A
, , 1

     [,1] [,2] [,3]
[1,]    1    6   11
[2,]    2    7   12
[3,]    3    8   13
[4,]    4    9   14
[5,]    5   10   15

, , 2

     [,1] [,2] [,3]
[1,]   16   21   26
[2,]   17   22   27
[3,]   18   23   28
[4,]   19   24   29
[5,]   20   25   30
```

The numbers enter each table column-wise, from left to right (rows, then columns then tables)

Subscripts with Arrays (II)

- To select columns of A (e.g. second and third):

```
> A[,2:3,]
```

: Columns are the second (middle) subscript

```
, , 1

     [,1] [,2]
[1,]    6   11
[2,]    7   12
[3,]    8   13
[4,]    9   14
[5,]   10   15

, , 2

     [,1] [,2]
[1,]   21   26
[2,]   22   27
[3,]   23   28
[4,]   24   29
[5,]   25   30
```

Subscripts with Arrays (III)

- To select columns of A (e.g. second and third) and rows (e.g. two to four), of only the second table:

```
> A[2:4,2:3,2]
```

: rows are the first, columns are the second, and table are the third subscript

```
     [,1] [,2]
[1,]   22   27
[2,]   23   28
[3,]   24   29
```


Subscripts with Lists (I)

- Lists are subscribed like this `[[3]]`: list called "cars", with three elements: "make", "capacity" and "color":

```
> cars<-
  list(c("Toyota","Nissan","Honda"),c(1500,1800,1750),c("b
  lue","red","black","silver"))
[[1]]
[1] "Toyota" "Nissan" "Honda"

[[2]]
[1] 1500 1800 1750

[[3]]
[1] "blue"   "red"    "black"  "silver"
```

- Difference between `cars[[3]]`:

```
[1] "blue"   "red"    "black"  "silver"
```

- And `cars[3]`:

```
[[1]]
[1] "blue"   "red"    "black"  "silver"
```

Subscripts with Lists (II)

- Lists are subscribed like this **[[3]]**: list called "cars", with three elements: "make", "capacity" and "color":

```
> cars<-
  list(c("Toyota","Nissan","Honda"),c(1500,1800,1750
  ),c("blue","red","black","silver"))
[[1]]
[1] "Toyota" "Nissan" "Honda"

[[2]]
[1] 1500 1800 1750

[[3]]
[1] "blue"   "red"    "black"  "silver"
```

- To extract one element of the sub-list:

```
> cars[[3]][2]
[1] "red"
```


Dataframes

Dataframes

- R handles data in objects known as dataframes;
  - rows: different observations;
  - columns: values of the different variables (numbers, text, calendar dates or logical variables (T or F);

Field Name	Area	Slope	Vegetation	Soil pH	Damp	Worm density
Nash's Field	3.6	11	Grassland	4.1	F	4
Silwood Bottom	5.1	2	Arable	5.2	F	7
Nursery Field	2.8	3	Grassland	4.3	F	2
Rush Meadow	2.4	5	Meadow	4.9	T	5
Gunness' Thicket	3.8	0	Scrub	4.2	F	6
Oak Mead	3.1	2	Grassland	3.9	F	2
Church Field	3.5	3	Grassland	4.2	F	3
Ashurst	2.1	0	Arable	4.8	F	4
The Orchard	1.9	0	Orchard	5.7	F	9
Rookery Slope	1.5	4	Grassland	5	T	7
Garden Wood	2.9	10	Scrub	5.2	F	8
North Gravel	3.3	1	Grassland	4.1	F	1
South Gravel	3.7	2	Grassland	4	F	2

Dataframes (II)

- All the values of the same explanatory variables must go in the same column!
- If you importing data from Excel, save the data in as tab-delimited text files
- The function **read.table** will fail if there are spaces in any of the variable names in the header (row 1) => replace " " by "."
- To read dataframes into R:
  - path: in double quotes;
  - **header = T** :the first row contains the variables names;
  - GUI: Used double back slash \\

```
> worms<-read.table("c:\\worms.txt",header=T,row.names=1)
```

Dataframes (III)

- Use **attach** to make the variables accessible by name:

```
> attach(worms)
```

- Use **names** to get a list of variable names:

```
> names(worms)
[1] "Area"         "Slope"        "Vegetation"
    "Soil.pH"      "Damp"
[6] "Worm.density"
```

- To see the content of the dataframe (object) just type ist name:

```
> worms
```


Dataframes (III)

- **Summary(worms)**

```
      Area           Slope          Vegetation    Soil.pH        Damp          Worm.density
 Min.   :0.800   Min.   : 0.00   Arable   :3   Min.   :3.500   Mode :logical   Min.   :0.00
 1st Qu.:2.175   1st Qu.: 0.75   Grassland:9   1st Qu.:4.100   FALSE:14        1st Qu.:2.00
 Median :3.000   Median : 2.00   Meadow   :3   Median :4.600   TRUE :6         Median :4.00
 Mean   :2.990   Mean   : 3.50   Orchard  :1   Mean   :4.555                   Mean   :4.35
 3rd Qu.:3.725   3rd Qu.: 5.25   Scrub    :4   3rd Qu.:5.000                   3rd Qu.:6.25
 Max.   :5.100   Max.   :11.00                 Max.   :5.700                   Max.   :9.00
```

- Values of the continuous variables:
  - arithmetic mean;
  - maximum, minimum, median, 25 and 75 percentiles (first and third quartile);
- Levels of categorical variables are counted

Selecting Parts of a Dataframe: Subscripts

- Subscripts within square brackets: to select part of a dataframe
- `[,` means "all the rows" and `,]` means "all the columns"
- To select the first three column of the dataframe `worms`:

```
> worms[,1:3]
                  Area Slope Vegetation
Nashs.Field        3.6    11  Grassland
Silwood.Bottom     5.1     2     Arable
Nursery.Field      2.8     3  Grassland
Rush.Meadow        2.4     5     Meadow
Gunness.Thicket    3.8     0      Scrub
(…)
```


Selecting Parts of a Dataframe: Subscripts (II)

- To select certain rows based on logical tests on the values of one or more variables:

```
> worms[Area>3&Slope<3,]

                Area Slope Vegetation Soil.pH  Damp Worm.density
Silwood.Bottom   5.1     2     Arable     5.2 FALSE            7
Gunness.Thicket  3.8     0      Scrub     4.2 FALSE            6
Oak.Mead         3.1     2  Grassland     3.9 FALSE            2
North.Gravel     3.3     1  Grassland     4.1 FALSE            1
South.Gravel     3.7     2  Grassland     4.0 FALSE            2
Pond.Field       4.1     0     Meadow     5.0  TRUE            6
Water.Meadow     3.9     0     Meadow     4.9  TRUE            8
Pound.Hill       4.4     2     Arable     4.5 FALSE            5
```

Sorting

- You can sort the rows or the columns in any way you choose but you need to state which column you want to be sorted (i.e. all of them for **worms** 1:6)
- e.g. the rows of the whole dataframe sorted by Area (this is the variable in column number one **[,1]**):

```
>worms[order(worms[,1]),1:6]

                  Area Slope Vegetation Soil.pH  Damp Worm.density
Farm.Wood          0.8    10      Scrub     5.1  TRUE            3
Rookery.Slope      1.5     4  Grassland     5.0  TRUE            7
Observatory.Ridge  1.8     6  Grassland     3.8 FALSE            0
The.Orchard        1.9     0    Orchard     5.7 FALSE            9
Ashurst            2.1     0     Arable     4.8 FALSE            4
Cheapside          2.2     8      Scrub     4.7  TRUE            4
Rush.Meadow        2.4     5     Meadow     4.9  TRUE            5
Nursery.Field      2.8     3  Grassland     4.3 FALSE            2
(…)
```


Sorting (II)

- Alternatively the dataframe can be sorted in descending order by Soil pH, with only Soil pH and Worm density as output:

```
>worms[rev(order(worms[,4]),c(4,6)]

                  Soil.pH Worm.density
The.Orchard           5.7            9
Garden.Wood           5.2            8
Silwood.Bottom        5.2            7
Farm.Wood             5.1            3
Pond.Field            5.0            6
Rookery.Slope         5.0            7
Water.Meadow          4.9            8
Rush.Meadow           4.9            5

(…)
```

Sorting and ordering

- Sorting != ordering
- It is dangerous in a dataframe to sort any of the variables on its own, because it becomes uncoupled from its associated explanatory variables
- => never use **`sort`** on variable that are part of a dataframe but **`order`**

Saving your work

- Save your Graphs (GUI:File-> Save as)
- To review the command lines entered during the sessions: **`history(Inf)`**
- Save the history of command lines to a text file: **`savehistory("c:\\tmp\\today.txt")`**
- And read it back into R with **`loadhistory("c:\\tmp\\today.txt")`**
- The session as a whole can be saved as a binary file with: **`save(list=ls(),file="c:\\tmp\\all.Rdata")`** and retrieved using **`load("c:\\temp\\ all.Rdata")`**

Save Function

- **save** writes an external representation of R objects to the specified file. The objects can be read back from the file at a later date by using the function **load**
- In the exercises: object called `lysozyme.data_frame.dat`

```
> load("lysozyme.data_frame.dat")
> names(lysozyme.data)

[1] "phen.bool"            "from.res"              "to.res"
[4] "sasa"                 "norm.bfactor"          "buried.charge"
[7] "buried.charge.change" "exposed.hydrophobic"   "delta.mass"
[10] "delta.surface"        "delta.volume"          "dssp.ss"
[13] "helix.breaker"        "turn.breaker"          "conservation"
[16] "pssm.score"

>lysozyme.data$norm.bfactor[lysozyme.data$phen.bool
   == T]
```


Tidying up

- Good practice to remove **rm(x,y,z)** any variables names
- and to detach any dataframes: **detach(worms)**. The dataframe do not disappear, but the variables within **worms** are no longer accessible directly by name
- To get rid of everything: **rm(list=ls())**

References

- M. Crawley, Statistics An Introduction using R, Wiley
- R web site: http://www.r-project.org/