Introduction to statistics with R

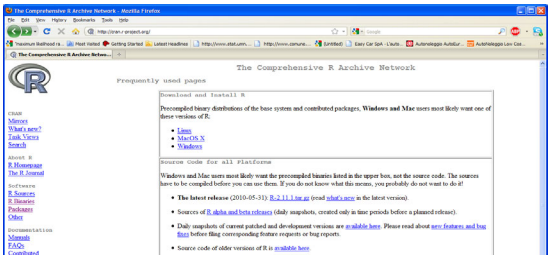
An overview of R

R overview

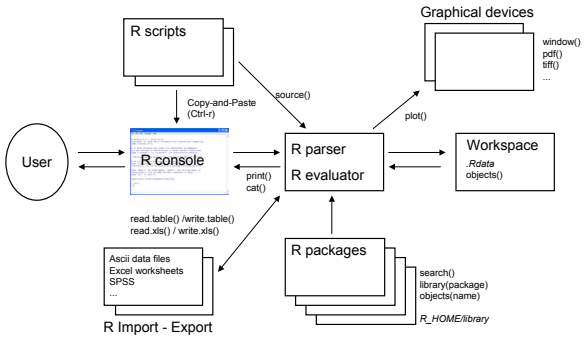
- The R language is a dialect of S which was designed in the 1980s by John M. Chambers at the Bell Laboratory. It has been in widespread use in the statistical community since.
- R is a high level language
  - Very rich set of data structures
  - Very large number of functions
- R is an interpreted language
- R is a vectorial language
  - Most operators/function can manipulate vectors
  - Good practice to avoid loops
- R is a functional language
- R includes object oriented programming concepts
- R is free

R Download

http://cran.r-project.org



R elements



R elements

- **The R console**

  The R-console is a simple interface to interact with the R-evaluator which interpret (or evaluate or execute) the typed-in command and prints the resulting value back in the R console.

- **The R evaluator**

  The R evaluator is core element of R: it reads the expressions written in the R-language and interprete (evaluate) them.

- **The workspace**

  The workspace represent the memory space that contains the user defined objects (e.g., variables, functions, data). It must be saved at the end of the session (.Rdata file).

- **The scripts (.R files)**

  The scripts contain the programs of the user (function definitions, analyses, etc.). They are simple text files and can be edited with any text processor. The expressions must be pasted in the R console to be executed.

R Console

- When a user types a *command* (or *statement* or *expression*) at the *prompt* (>), the R evaluator reads, parses (i.e., analyzes the syntax), executes the corresponding R expressions and returns the value of the expression.

- Autoprint: the value of top-level expression expression is automatically printed on the console

- The prompt is >, unless the command is syntactically incomplete, when the prompt changes to +.

- Commands can be separated by either the new line or semicolumn (;).

- The symbol # marks the rest of the line as a comment.

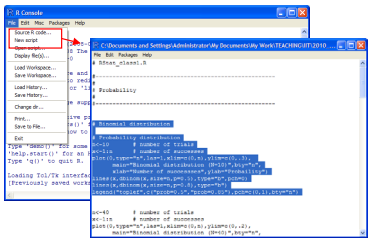
Examples

```
> # comments
> "A"
[1] "A"

> x<-c(3,1,2)
> x
[1] 3 1 2

> x<-c(3,1,2); x
[1] 3 1 2
```


R scripts



- The scripts typically contain the programs of the user (function definitions, analyses, etc.).
- It is conventional to give the extension .R for R scripts (.q, .ssc have also been used)
- R scripts are simple ASCII text files and can be edited with any text processor (e.g. notepad). The expressions must be copy/pasted in the R console to be executed.
- The R console comes with a simple editor which has a macro (Ctrl-R) to copy/paste in the selected text in the console.

The concept of programming language

- Computers know how to execute only a limited number of simple instructions (e.g., add two number, copy a byte from one memory address to another) that forms the so-called ***language machine*** or ***binary code***.
- ***Low-level languages*** are closely related to the language machine of a computer.
  - Today, programms are written in high-level languages
- ***High-level languages*** (C, BASIC, FORTRAN, PASCAL, LISP, R) include a much larger number of instructions and fonctionalities. They are indepedent from the hardware.
  - All programming languages are *formally* defined. They therfore are unambiguous though their rules might be sometimes difficult to learn.
- High-level languages must be translated into binary code with an ***interpreter*** or a ***compiler***.
  - **Interpreter**: translate and execute (***evaluate***) expressions once at the time, often in an interactive manner with the programmer.
  - **compiler**: translate the whole program at once, which can be later executed

The concept of program

- The concept of program can refer to different things:

1. Binary code that can be executed
   - the *.exe or *.com files in Windows

2. A text file written in some programming language.
   - this text needs to be translated into binary code to be executed

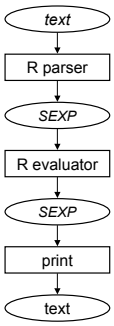
   .

3. A classical definition in computer science (N. Wirth, 1978):

   Programs = algorithms + data structures

   - Algorithms: methods to resolve the problem
     - Programming languages offer the means (e.g. control structures) to implement algorithms
   - Data structures: format/organisation of the information in the computer memory that is manipulated by the algorithms
     - Available data structures (or data types) depend on the programming language

R Evaluation




```
sqrt(7+2)
```

```
[1] 3
```

- The text (R expressions) is typically entered by the user in the console.
- R expressions are translated into an internal tree-like structure by the parser.
  Note that this structure is itself an R object that can be manipulated by R.
- The R evaluator will evaluates the internal structure created by the parser. Typically, this evaluation will yield a value (e.g., a number) that is returned
- The returned value is in general printed by the R console by an implicit call to the function print.

The workspace

- User-defined R objects are by default stored in the workspace (.GlobalEnv).
- The function `objects` returns the names of the objects in the workspace

```
> objects()
```

- All objects in the workspace can be deleted with

```
> remove(list=objects())
```

- A single object can be deleted with the function rm or remove

```
> remove("x")        # note that the object name must be quoted
> rm(x)              # the object name does not need to be quoted with rm()
```

- When an object is not found in the workspace, R will look for its presence in the packages (see next slide). If the object is not found at all, R returns an error message

```
> remove("x")
Error: object "xxx" not found
```

R objects and the assignement operator (<-)

- R objects (e.g., variables, functions, etc.) have a name and a value. The value of a R object is printed on the console when its name is given:

```
> pi          # predefined value of pi
[1] 3.141593
> "pi"        # Note that the name must not be quoted (otherwise it is a
              # string).
[1] "pi"
```

- New objects are created by associated a value to a name with the assignement operator "<-"

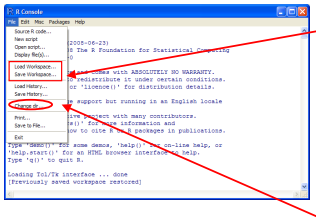
```
<object name> <- <expression>
```

the expression is evaluated and its value is associated to the name, which can be quoted.

```
> x <- (4+10)/2
> x
[1] 7
```

- A name in R must start with a letter. It can contain letters, numbers and the dot (e.g. "x", "v001", "home.address", etc.).

.Rdata, working directory



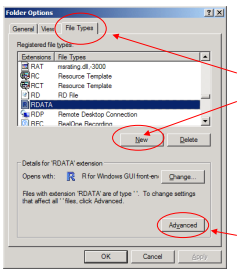
- An image of the workspace can be saved in a file (.Rdata)
- To save/load workspace from the console:

```
save.image()
save()
load(file)
```

- By default, all files input/output operation are done in the working directory

```
getwd()
setwd(dir)
```

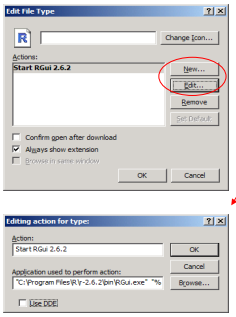

Associate R with .Rdata extension



- You can change file associations by going into Windows Explorer (hold down Windows key and press E) and choosing Tools | Options.
- Select the File Types tab.
- Now, select RDATA from the long list. If it does not exist, click on New to create it. Enter File extension .Rdata and click ok.

- Click on Advanced in the Folder Options Menu.

Associate R with .Rdata extension

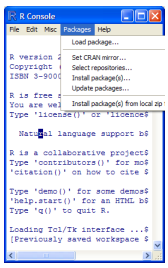


- Click New to create a new association or Edit to change an existng one
- Enter the path of the RGui.exe application in the box labelled Application:

  "C:\Program Files\R\R-2.6.2\bin\RGui.exe" "%1"

  and a some text (e.g., "Start R") in the box labelled Action.

R Packages



- R packages contains many R objects (typically functions).
- The function search() gives the list of currently attagched packages

```
> search()
[1] ".GlobalEnv"        "package:lattice"
    "package:stats"     "package:graphics"
    "package:grDevices"
[6] "package:utils"     "package:datasets"
    "package:methods"   "Autoloads"
    "package:base"
```

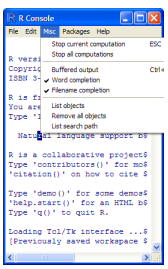
- The function library(<package>) attaches a new packages (or use the Packages/Load package menu option)

```
> library(lattice) # load the lattice package
```

- Packages can be downloaded from the R website using the Packages/Install packages menu option. By default, they are installed in the R_HOME/library subdirectory.

```
> Sys.getenv("R_HOME")
[1] "C:\\PROGRA~1\\R\\R-27~1.1"
```


R Misc



- ESC to stop current computation
- Buffered output: printout is displayed on the console only at the end of the computation
- List objects

```
> objects()
```

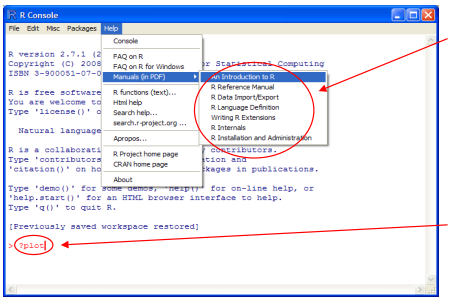
- Remove all objects

```
> remove(list=objects())
```

- list search path (packages)

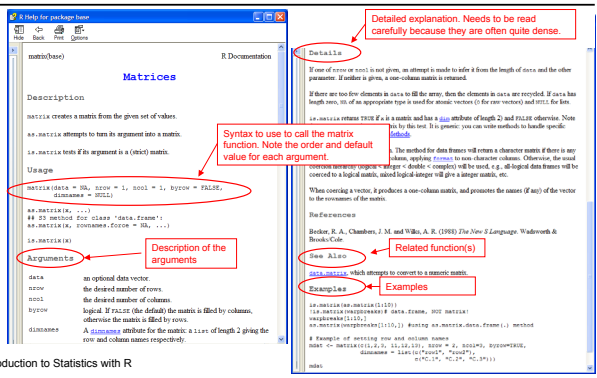
```
> search()
```

Help in R



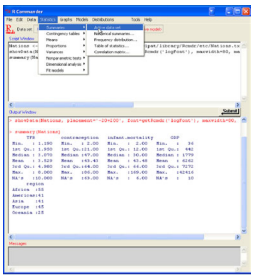
- Well done documentation is included
- Use ?<function> to get the help page for a function

Example of an R help page (?matrix)



R vs SPSS

R Commander;



http://socserv.mcmaster.ca/jfox/Misc/Rcmdr/

- **SPSS** is a stastical software with an sophisticated graphical user interface (GUI)
- Limited programming capacities (syntax)
- R is a programming language with good capacity to manipulate data and excellent graphics (like MATLAB).
- R is widely used in statistical departments
- By default, R has a simple command line interface (the R Console)
- **R Commander** developped by John Fox is a basic-statistics graphical user Interface to R

R vs MATLAB

	R	MATLAB
Row vectors $\begin{bmatrix}-1 & 4\end{bmatrix}$	`c(-1,4)` or `matrix(c(-1,4),1,2)`	`[-1 4]`
Column vectors $\begin{bmatrix}2\\3\end{bmatrix}$	`matrix(c(2,3),2,1)`	`[2 ; 3]`
Matrices $\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix}$	`matrix(c(1,4,2,5,3,5),2,3)` or `matrix(1:6,2,3,byrow=T)`	`[[1 2 3];[4 5 6]]` or `[[1;4],[2;5],[3;6]]` or `[[1 4]',[2 5]',[3 6]']`
Diagonal matrix $\begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & 1\end{bmatrix}$	`diagonal(3)`	`eye(3)`
Matrix of zeros $\begin{bmatrix}0 & 0 & 0\\0 & 0 & 0\end{bmatrix}$	`matrix(0,2,3)`	`zeros(2,3)`

R vs MATLAB

		R	MATLAB

Assignment		`<-`
		`x<-matrix(c(-1,4),1,2)`
		`A<-matrix(1:6,2,3,byrow=T)`
			`=`
			`x = [-1 4]`
			`A = [[1 2 3];[4 5 6]]`

Matrix dimensions	$\begin{bmatrix}2 & 3\end{bmatrix}$	`dim(A)`	`size(A)`
Extraction of subsets	$\begin{bmatrix}-1\end{bmatrix}$	`x[1]`	`x(1)`
	$\begin{bmatrix}1 & 2\\4 & 5\end{bmatrix}$	`A[,1:2]`	`A(:,1:2)`
Transpose	$\begin{bmatrix}-1\\4\end{bmatrix}$	`t(x)`	`x'`
Scalar Multiplication	$\begin{bmatrix}1 & 16\end{bmatrix}$	`x*x`	`x.*x`
Dot product	$\begin{bmatrix}17\end{bmatrix}$	`x%*%t(x)`	`x*x'`
Matrix multiplication	$\begin{bmatrix}15 & 18 & 21\end{bmatrix}$	`x%*%A[,1:2]`	`x*A(,1:2)`

R Import and Export

Outline

- Data Import and Export
  - Text tables, Excel, SPSS, SAS, etc.
- Exporting graphs
  - Postscript, PDF, tiff, jpeg, wmf, etc.
- Sounds, images, videos

Importing and exporting data in R

- Text files
  - `read.table`: read data in tabular format in a text file
  - `write.table`: write matrix or data frame in a text file
- Excel
- Statistical software (library `foreign`)
  - `read.spss`: SPSS
  - `read.systat`: SYSTAT
  - `read.xport, read.ssd`: SAS
  - `read.dta`: STATA
  - etc.

Reference: R Data Import/Export Manual

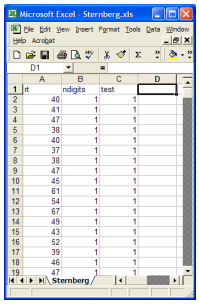
Importing from Excel

- Convert datasheet in tab or comma separated format
  - If you have access to Excel, export the data in tab-delimited (.txt, .dat) or comma separated (.csv) format from Excel and use `read.table` to import data in R.
  - If you don't have Excell Use an an open-source software to read excell data sheets (e.g., Gnmumeric, OpenOffice) and export them in a tab or comma separated format.
- Cut-and-paste method
  - You can select and Paste data in Excel and use read.table with file="clipboard" to import data in R. This method will also work with most other spreadshees.

Packages:

- xlsReadWrite:
  - read/write.xls 97-2003 Excel files.
- dataframe2xls:
  - requires Python
- WriteXLS:
  - requires Perl
- xlxs:
  - access to xlsx trhough Apache/Java project
- RExcelInstaller:
  - add-in for MS Excel, allows to transfer data between R and Excel, writing VBA macros using R as a library for Excel, and calling R functions as worksheet function in Excel.
- Package RODBC

Example. Sternberg dataset



Source: Howell (2002) Statistical Methods for Psychology.

- In a study on short-term memory (Sternberg, 1966), people were asked to identify whether a digit was present in a previously displayed set of digits (called the comparison set). The data set reports the reaction times (in 100s of a second, second column) for the correct responses of one subject as a function of the number of digits in the set (1, 3, or 5, 3rd column) and on whether the digit was present or not in the comparison set (1=included, 2= not included, 4th column).

```
> library(xlsReadWrite) # load package
> stern<-read.xls("sternberg.xls")
> dim(stern)
[1] 300   4
> head(stern)
  rt ndigits test
1 40       1    1
2 41       1    1
3 47       1    1
4 38       1    1
5 40       1    1
6 37       1    1
> names(stern)
[1] "rt"      "ndigits" "test"
```


read.table

```
read.table(file, header = FALSE, sep = "",  dec = ".", as.is = FALSE, skip = 0)
```

Description

Reads a file in table format and creates a data frame from it, with cases corresponding to lines and variables to fields in the file.

Main arguments

`file`	the name of the file which the data are to be read from.
`header`	a logical value indicating whether the file contains the names of the variables as its first line.
`sep`	the field separator character (e.g., `" "`: space, `"\t"`: tab, `","`: comma). Values on each line of the file are separated by this character. If `sep = ""` (the default), then one or more spaces, tabs, and newlines or carriage returns are interpreted as a separator.
`dec`	the character used in the file for decimal points (usually "." or ",")
`as.is`	if `as.is=TRUE`, character variables are converted to factors.
`skip`	the number of lines of the data file to skip before beginning to read data.
`col.names`	a vector of optional names for the variables. The default is to use `""V""` followed by the column number. Useful if the firs row of the file does not specify the column names.

Note

`read.csv, read.delim,...` are equivalent to read.table with different defaults

write.table

```
write.table(x, file = "", append = FALSE, quote = TRUE, sep = " ", dec = ".",
            row.names = TRUE, col.names = TRUE)
```

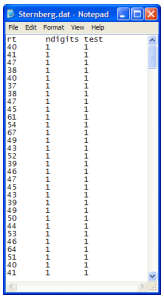
Description

'write.table' prints its required argument 'x' (a data frame or matrix) to a file.

Main arguments

`x`	the object to be written, preferably a matrix or data frame.
`file`	the name of the file which the data are to be written.
`col.names`	logical value indicating whether or not to write the column names
`row.names`	logical value indicating whether or not to write the row names
`sep`	the field separator string, e.g. `" "`: single space (the default) or `"\t"`: tab
`quote`	If `TRUE`, any character or factor columns will be surrounded by double quotes. If `FALSE`, nothing is quoted.
`dec`	the character used in the file for decimal points (usually "." or ",")
`append`	If `TRUE`, the output is appended to the file. If `FALSE`, any existing file of the name is destroyed.

Example. Sternberg dataset



- Write data in tab-separated text file

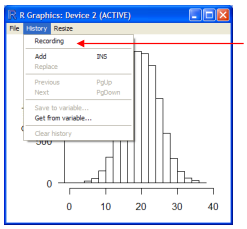
```
> write.table(stern,file="sternberg.dat",
          row.names=FALSE,quote=FALSE,sep="\t")
```

- Read data from text file

```
> stern<-read.table("sternberg.dat",header=TRUE)
> dim(stern)
[1] 300   4
> head(stern)
   rt ndigits test
1 40       1    1
2 41       1    1
3 47       1    1
4 38       1    1
5 40       1    1
6 37       1    1
> names(stern)
[1] "rt"      "ndigits" "test"
```


R Graphics

Windows graphical device

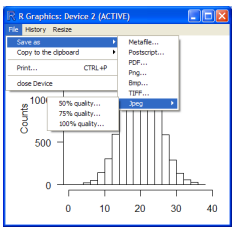


- A graphical window is automically created by a plotting functions (e.g `plot`, `hist`, ...)
- A device can record several plots. For the windows device, you need to activite recording. Use PgUp/PgDown to scroll between plots. You can clear recorded plots with Clear history. Don't forget to desactivate it!
- Several graphical devices or windows can coexist. New graphical window or device can be created from the console with `windows()`.
- Plotting occurs in the active or current device

```
> dev.cur()
windows
      2
```

Use `dev.list()` to get the list of all graphical devices.

Exporting graphs



- Available graphical formats are
  - Windows Metafile (vectorial)
  - Postscript (vectorial)
  - PDF (vectorial)
  - TIFF (raster)
  - PNG (raster)
  - BMG (raster)
  - JPEG (raster)

  Vectorial file formats (e.g., wiallow one to resize the file without loosing quality. Postscript file format can be read by the largest number of programs (Adobe Illustrator, Corel Draw, etc.)
- To quickly transfer a graph in another document (e.g. Word, PowerPoint), use the "File/Copy to Clipboard/as Metafile" menu option and paste in the desired document.
- To save the graph in a file Use the "File/Save As" menu option.
- Plot directly into a file by create a graphical device of the desired type with the functions `postscript`, `pdf`, `bmp`, `tiff` or `jpeg`. This methods offers more options than using the "File/Save As menu" option.

Graphic devices

- `windows`	The graphics device for Windows (on screen, to printer and to Windows metafile).
- `postscript`	Writes PostScript graphics commands to a file
- `pdf`	Write PDF graphics commands to a file
- `pictex`	Writes LaTeX/PicTeX graphics commands to a file
- `png`	PNG bitmap device
- `jpeg`	JPEG bitmap device
- `bmp`	BMP bitmap device
- `tiff`	TIFF bitmap device
- `xfig`	Device for XFIG graphics file format
- `bitmap`	bitmap pseudo-device via GhostScript (if available).

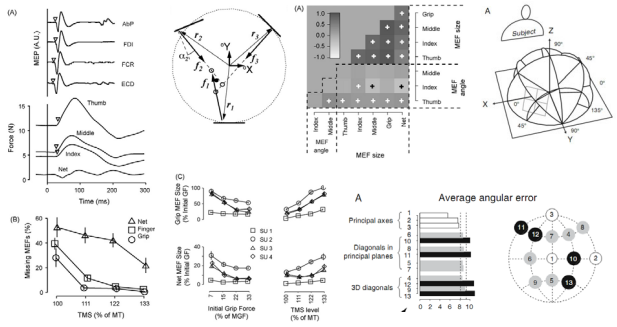
- Example. Making a plot in a pdf file:

```
># define pdf graphic device
>pdf(file="hist.pdf",paper="a4")
># plotting instructions
> hist(rnorm(1000,20,5),xlim=c(0,40),
   breaks=20,xlab="",ylab="Counts",
   main="Normal distribution")
># list graphic devices
> dev.list()
windows     pdf
      2       3
># current graphic device
> dev.cur()
 pdf
   3
># delete current graphic device
> dev.off()
windows
      2
```

Graphical Procedures in R

- R default graphical functions (graphics package)
  - High-level plotting functions
    High-level plotting functions are designed to generate a complete plot of the data passed as arguments to the function. Where appropriate, axes, labels and titles are automatically generated (unless you request otherwise.) High-level plotting commands always start a new plot, erasing the current plot if necessary.
  - Low-level plotting commands.
    Sometimes the high-level plotting functions don't produce exactly the kind of plot you desire. In this case, low-level plotting commands can be used to add extra information (such as points, lines or text) to the current plot.
  - see plotmath for mathematical expressions and greek letters on the plots
- Exploratory data analysis: the **lattice** package

Examples of plots (graphics package)



High-level plotting functions

Basic plots:

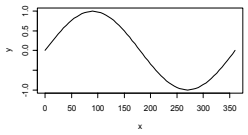
`plot(x,y,…)`	if x and y are vectors, produces a line plot or scatterplot of y against x. Note that plot is a generic function and its behavior might change according to the type of its first argument.
`matplot(x,y,…)`	plot the columns of one matrix against the columns of another (x and y are assumed to be two matrices).
`hist(x,…)`	produces a histogram of the numeric vector x.
`boxplot(x,…)`	produce box-and-whisker plot(s) of the given (grouped) values
`barplot(height,…)`	creates a bar plot with vertical or horizontal bars.

Plots of three variables (or more variables):

`image(x,y,z,...)`	draws different colors to represent the value of z
`contour(x,y,z,...)`	draws contour lines to represent the value of z
`persp(x,y,z,...)`	draws a 3D surface.
`pairs(x)`	produces a pairwise scatterplot matrix of the variables defined by the columns of X.
`coplot(a ~ b | c)`	If a and b are numeric vectors and c is a numeric vector or factor object (all of the same length), then the produces a number of scatterplots of a against b for given values of c.

Many graphical parameters for these functions are set by the function `par()`

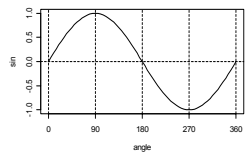
Example: simple line plot.



Simple plot of y=sin(x)

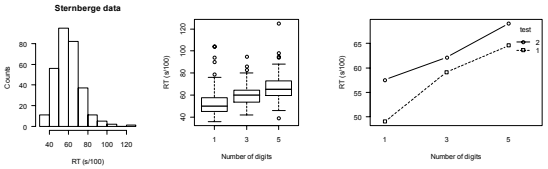
```
> x<-seq(0,360,5)
> y<-sin(x*pi/180)
> plot(x,y,type="l")
```

Plot with more detailed control




```
> # setup the plot (except x-axis)
+   without plotting anything inside
> plot(0,type="n",xlim=c(0,360),ylim=c(-1,1),
+           xlab="angle",ylab="sin",xaxt="n")
> # plot x-axis
> axis(1,at=seq(0,360,90))
> # plot thick curve
> lines(x,y,lwd=2)
> # plot dotted vert and horiz lines
> abline(h=0,v=seq(0,360,90),lty=2)
```


Example of plots




```
hist(stern$rt,las=1,xlab="RT (s/100) ",ylab="Counts",main="Sternberg data")

boxplot( rt ~ ndigits,stern,xlab="Number of digits",ylab="RT (s/100)")

interaction.plot(stern$ndigits,stern$test,stern$rt,type="b",pch=0:1,las=1,
          xlab="Number of digits",ylab="RT (s/100)",trace.label="test")
```


Arguments to high level functions

There are a number of arguments which may be passed to high-level graphics functions as follows:

`type=char`	character indicating the type of the plot (e.g, `type="p"`): - "p" for points (default), - "l" for lines , - "n" for setting up the plot without plotting anything.
`xlim=c(x0,x1)`	range of values for the x-axis (e.g., `xlim=c(-1,1)`)
`ylim=c(y0,y1)`	range of values for the y-axis
`xlab=string`	labels for the x axis (e.g., `xlab="time [s]"`)
`ylab=string`	labels for the y axis
`main=string`	title of the plot
`box=FALSE`	suppresses the box around the plot
`axes=FALSE`	suppresses generation of axes—useful for adding your own custom axes with the `axis()` function
`xaxt, yaxt`	a character which specifies the axis type. Specifying "'n'" causes an axis to be set up, but not plotted.

Example

```
> x<-seq(0,360,5)
> y<-sin(2*pi/180)
> plot(x,y,type="l",xlim=c(0,360),ylim=c(-1,1),xlab="angle",ylab="sin")
```

Low-level plotting commands

Sometimes the high-level plotting functions don't produce exactly the kind of plot you desire. In this case, low-level plotting commands can be used to add extra information (such as points, lines or text) to the current plot.

`lines(x,y,…)`	adds a line to the current plot
`points(x,y,…)`	adds a points to the current plot
`abline(h=y,v=x,…)`	
`abline(a,b,…)`	adds one or more straight lines through the current plot. `a` and `b` correspond to the intercept and slope. `h=y` may be used to specify y-coordinates for the heights of horizontal lines to go across a plot, and `v=x` similarly for the x-coordinates for vertical lines.
`title()`	adds a title to the current plot
`axis()`	adds an axis to the current plot
`legend()`	adds a legend to the current plot
`text()`	adds text to the current plot
`arrows()`	adds arrows to the current plot
`polygon()`	adds a polygon to the current plot

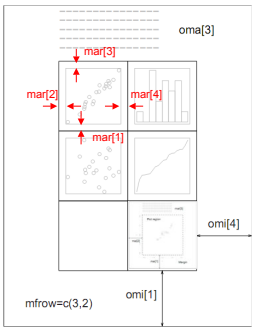
See corresponding help pages for mode details.

Graphical parameters

These graphical parameters can also be used with most plotting functions such as plot, lines, points, abline, etc.

`col`	a number or string indicating the color (1: black, 2: red, 3: green, 4: blue, etc.).
`lwd`	a number indicating the width of the line.
`lty`	The line type. Line types can either be specified as an integer or the corresponding character string (0=blank, 1=solid, 2=dashed, 3=dotted, 4=dotdash, 5=longdash, 6=twodash) .
`pch`	symbol to use as a point: can be a character (e.g. pch='+') or an integer code between 0 and 18 for a graphics symbol (e.g., 0=empty square, 1=empty circle, etc.).

`par` - controlling the layout



The function `par` can be used to control the layout.

`mar`	specify the size od the margins (default: `mar=c(5.1,4.1,4.1,2.1)`)
`oma`	specify the size of the outer margin (default: `oma=c(0,0,0,0)`)
`mfrow,mfcol`	divide the plotting area in several sub-plot areas arranged in m rows and n column (default: `mfrow=c(1,1)`).
Example.	The following instruction will divide the page in 6 plot regions arranged in a 3x2 layout (mfrow), make a outer margin (oma) and change the value of the plot region margins (mar).

```
par(mfrow=c(3,2),oma=c(2,2,2,2),mar=c(1,1,1,1))
```


The lattice library

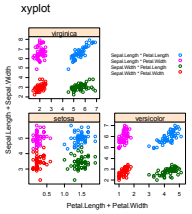
- The lattic library is an implementation of the Trellis Graphics for data visualization. All these functions allow to draw various plots (scatter plots, etc.) conditioned on some additional variables. The most useful function is `xyplot`.

- Univariate data:
  - `barchart` bar plots
  - `bwplot` box and whisker plots
  - `densityplot` kernel density plots
  - `dotplot` dot plots
  - `histogram` histograms
  - `qqmath` quantile plots against mathematical distributions
  - `stripplot` 1-dimensional scatterplot
- Bivariate data:
  - `xyplot` scatter plot (and possibly a lot more)
  - `qq` q-q plot for comparing two distributions
- Trivariate:
  - `levelplot` level plots (similar to image plots in R)
  - `contourplot` contour plots
  - `cloud` 3-D scatter plots
  - `wireframe` 3-D surfaces (similar to persp plots in R)
- Hypervariate:
  - `splom` scatterplot matrix (similar to pairs in R)
  - `parallel` parallel coordinate plots

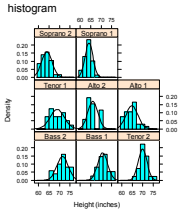
See corresponding help pages for mode details

Examples. Trellis functions

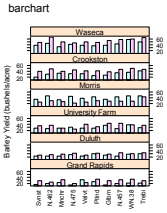
xyplot



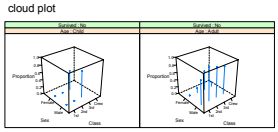
histogram



barchart



cloud plot



wireframe plot