

## R QUICK REFERENCE CARD

Most frequently used R commands – Version v1.0 May 2014

The original Reference Cheat was created by Tom Short, EPRI PEAC, in 2004-10-21. I modified the original document so it fits my other reference cards, all of its content has been preserved!

### Help

Most R functions have online documentation.

`help(topic)` . documentation on `topic`

`?topic` . . . . . id.

`help.search("topic")`  
search the help system

`apropos("topic")`  
the names of all objects in the search  
list matching the regular expression "topic"

`help.start()` start the HTML version of help

### Fundamentals

`<-` . . . . . assign to an object, equivalent to `=`(?)

`<<-` . . . . . lexical assignment (\*NOT\* global assignment)

`getwd()` . . . . . get the working directory

`setwd()` . . . . . set the working directory

`system()` . . . . . call the operating system (shell)

`system.time()`  
time an evaluation

`Sys.sleep()` . pause

`str(a)` . . . . . display the internal \*structure\* of an R object `a`

`summary(a)` .. gives a "summary" of `a`, usually a statistical summary but it is *generic* meaning it has different operations for different classes of `a`

`ls()` . . . . . show objects in the search path; specify `pat="pat"` to search on a pattern

`ls.str()` . . . . . `str()` for each variable in the search path

`dir()` . . . . . show files in the current directory

`methods(a)` .. shows S3 methods of `a`

`methods(class=class(a))`  
lists all the methods to handle objects of class `a`

### Input and output

`load()` . . . . . load the datasets written with `save`

`data(x)` . . . . . loads specified data set

`library(x)` .. load add-on packages

`read.table(file)`  
reads a file in table format and creates a data frame from it; the default separator `sep=""` is any whitespace; use `header=T` to read the first line as a header of column names; use `as.is=T` to prevent character vectors from being converted to factors; use `comment.char=""` to prevent "#" from being interpreted as a comment; use `skip=n` to skip `n` lines before reading data; see the help for options on row naming, NA treatment, and others

`read.csv("filename",header=T)`  
id. but with defaults set for reading comma-delimited files

`read.delim("filename",header=T)`  
id. but with defaults set for reading tab-delimited files

`read.fwf(file,widths,header=F,sep="t",as.is=F)`  
read a table of fixed width formatted data into a 'data.frame'; `widths` is an integer vector, giving the widths of the fixed-width fields

`save(file,...)`  
saves the specified objects (...) in the XDR platform-independent binary format

`save.image(file)`  
saves all objects

`cat(..., file="", sep=" ")`  
prints the arguments after coercing to character; `sep` is the character separator between arguments

`print(a, ...)` prints its arguments; generic, meaning it can have different methods for different objects

`format(x,...)`  
format an R object for pretty printing

`write.table(x,file="",row.names=T,col.names=T, sep=" ")`  
prints `x` after converting to a data frame; if `quote` is `TRUE`, character or factor columns are surrounded by quotes ("); `sep` is the field separator; `eol` is the end-of-line separator; `na` is the string for missing values; use `col.names=NA` to add a blank column header to get the column headers aligned correctly for spreadsheet input

`sink(file)` .. [output to file, until `sink()`] Most of the I/O functions have a `file` argument. This can often be a character string naming a file or a connection. `file=""` means the standard input or output. Connections can include files, pipes, zipped files, and R variables.

On windows, the file connection can also be used with `description = "clipboard"`.

To read a table copied from Excel, use:

`x <- read.delim("clipboard")`

To write a table to the clipboard for Excel, use:

`write.table(x,"clipboard",sep="\t",col.names=NA)`

For database interaction, see packages `RODBC`, `DBI`, `RMySQL`, `RPostgreSQL`, and `ROracle`. See packages `XML`, `hdf5`, `netCDF` for reading other file formats.

### Data creation

`c(...)` . . . . . generic function to combine arguments with the default forming a vector; with `recursive=T` descends through lists combining all elements into one vector

`from:to` ..... generates a sequence; “:” has operator priority; 1:4 + 1 is “2,3,4,5”

`seq(from,to)` generates a sequence `by=` specifies increment; `length=` specifies desired length

`seq(along=x)` generates 1, 2, ..., `length(along)`; useful for `for` loops

`rep(x,times)` replicate `x` `times`; use `each=` to repeat “each” element of `x` `each` times;

```
rep(c(1,2,3),2):  1 2 3 1 2 3
rep(c(1,2,3),each=2):  1 1 2 2 3 3
```

`data.frame(...)`  
create a data frame of the named or unnamed arguments

shorter vectors are recycled to the length of the longest:

```
data.frame(v=1:4,ch=c("a","B","c","d"),n=10)
```

`list(...)` ... create a list of the named or unnamed arguments

use: `list(a=c(1,2),b="hi",c=3i)`

`array(x,dim=)`  
array with data `x`; specify dimensions like `dim=c(3,4,2)`; elements of `x` recycle if `x` is not long enough

`matrix(x,nrow=,ncol=)`  
matrix; elements of `x` recycle

`factor(x,levels=)`  
encodes a vector `x` as a factor

`gl(n,k,length=n*k,labels=1:n)`  
generate levels (factors) by specifying the pattern of their levels; `k` is the number of levels, and `n` is the number of replications

`expand.grid()`  
a data frame from all combinations of the supplied vectors or factors

`rbind(...)` .. combine arguments by rows for matrices, data frames, and others

`cbind(...)` .. id. by columns

## Slicing and extracting data

### Indexing vectors

<code>x[n]</code>	$n^{th}$ element
<code>x[-n]</code>	all <i>but</i> the $n^{th}$ element
<code>x[1:n]</code>	first elements
<code>x[-(1:n)]</code>	elements from <code>n+1</code> to the end
<code>x[c(1,4,2)]</code>	specific elements
<code>x["name"]</code>	element named "name"
<code>x[x &gt; 3]</code>	all elements greater than 3
<code>x[x &gt; 3 &amp; x &lt; 5]</code>	all elements between 3 and 5
<code>x[x %in% c("a","and","the")]</code>	elements in the given set

### Indexing lists

<code>x[n]</code>	list with elements <code>n</code>
<code>x[[n]]</code>	$n^{th}$ element of the list
<code>x[["name"]]</code>	element of the list named "name"
<code>x\$name</code>	id.

### Indexing matrices

<code>x[i,j]</code>	element at row <code>i</code> , column <code>j</code>
<code>x[i,]</code>	row <code>i</code>
<code>x[,j]</code>	column <code>j</code>
<code>x[,c(1,3)]</code>	columns 1 and 3
<code>x["name",]</code>	row named "name"

### Indexing data frames

matrix indexing plus the following

<code>x[["name"]]</code>	column named "name"
<code>x\$name</code>	id.

### Variable information

`is.na(x)`, `is.null(x)`, `is.array(x)`, `is.data.frame(x)`, ...

`methods(is)` . all available lists of typetests

`methods(as)` . list of all variable conversions

`any(x)` ..... any TRUE elements of `x`?

`all(x)` ..... all TRUE elements of `x`?

`length(x)` ... number of elements in `x`

`dim(x)` ..... Retrieve or set the dimension of an object; `dim(x) <- c(3,2)`

`dimnames(x)` . Retrieve or set the dimension names of an object

`nrow(x)` ..... number of rows; `NROW(x)` is the same but treats a vector as a one-row matrix

`ncol(x)` ..... and

`NCOL(x)` ..... id. for columns

`class(x)` ..... get or set the class of `x`; `class(x) <- "myclass"`

`unclass(x)` .. remove the class attribute of `x`

`attr(x,which)`  
get or set the attribute `which` of `x`

`attributes(obj)`  
get or set the list of attributes of `obj`

### Data selection and manipulation

`which.max(x)` returns the index of the greatest element of `x`

`which.min(x)` returns the index of the smallest element of `x`

`rev(x)` ..... reverses the elements of `x`

`sort(x)` ..... sorts the elements of `x` in increasing order

`rev(sort(x))` to sort in decreasing order

`cut(x,breaks)`  
divides `x` into intervals (factors); `breaks` is the number of cut intervals or a vector of cut points

`x %in% y` ..... logical vector indicating if there is a match or not for its left operand

`match(x, y)` . returns a vector of the same length than `x` with the elements of `x` which are in `y` (NA otherwise)

**which(x == a)**  
returns a vector of the indices of **x** if the comparison operation is true (*T*), in this example the values of **i** for which **x[i] == a** (the argument of this function must be a variable of mode logical)

**choose(n, k)** computes the combinations of *k* events among *n* repetitions =  $n! / [(n - k)!k!]$

**combn(n, k)** . Generate All Combinations of *n* Elements, Taken *m* at a Time.

**na.omit(x)** .. suppresses the observations with missing data (NA) (suppresses the corresponding line if **x** is a matrix or a data frame)

**na.fail(x)** .. returns an error message if **x** contains at least one NA

**unique(x)** ... if **x** is a vector or a data frame, returns a similar object but with the duplicate elements suppressed

**table(x)** ..... returns a table with the numbers of the different values of **x** (typically for integers or factors)

**subset(x, ...)**  
returns a selection of **x** with respect to criteria (... , typically comparisons: **x\$V1 < 10**); if **x** is a data frame, the option **select** gives the variables to be kept or dropped using a minus sign

**sample(x, size)**  
resample randomly and without replacement **size** elements in the vector **x**, the option **replace = TRUE** allows to resample with replacement

**prop.table(x, margin =)**  
table entries as fraction of marginal table

## Characters (Strings)

**paste(...)** .. concatenate vectors after converting to character; **sep=** is the string to separate terms (a single space is the default); **collapse=** is an optional string to separate “collapsed” results

**substr(x,start,stop)**  
substrings in a character vector; can also assign, as  
**substr(x, start, stop) <- value**

**strsplit(x,split)**  
split **x** according to the substring **split**

**grep(pattern,x)**  
searches for matches to **pattern** within **x**; see **?regex**

**gsub(pattern,replacement,x)**  
replacement of matches determined by regular expression matching **sub()** is the same but only replaces the first occurrence.

**tolower(x)** .. convert to lowercase

**toupper(x)** .. convert to uppercase

**match(x,table)**  
a vector of the positions of first matches for the elements of **x** among **table**

**x %in% table**  
id. but returns a logical vector

**pmatch(x,table)**  
partial matches for the elements of **x** among **table**

**nchar(x)** ..... number of characters

**assign** ..... assign a value to a name

**get** ..... get a value from a name

**eval(parse(text='1+1'))**  
compute on the language!!

## Dates and Times

The class **Date** has dates without times. **POSIXct** has dates and times, including time zones. Comparisons (e.g. **>**), **seq()**, and **difftime()** are useful. **Date** also allows **+** and **-**. **?DateTimeClasses** gives more information. See also package **chron**.

**as.Date(s)** .. and

**as.POSIXct(s)**  
convert to the respective class; **format(dt)** converts to a string representation. The default string format is “2001-02-21”. These accept a second argument to specify a format for conversion. Some common formats are:

<b>%a, %A</b>	Abbreviated and full weekday name.
<b>%b, %B</b>	Abbreviated and full month name.
<b>%d</b>	Day of the month (01–31).
<b>%H</b>	Hours (00–23).
<b>%I</b>	Hours (01–12).
<b>%j</b>	Day of year (001–366).
<b>%m</b>	Month (01–12).
<b>%M</b>	Minute (00–59).
<b>%p</b>	AM/PM indicator.
<b>%S</b>	Second as decimal number (00–61).
<b>%U</b>	Week (00–53); the first Sunday as day 1 of week 1.
<b>%w</b>	Weekday (0–6, Sunday is 0).
<b>%W</b>	Week (00–53); the first Monday as day 1 of week 1.
<b>%y</b>	Year without century (00–99). Don’t use (!)
<b>%Y</b>	Year with century.
<b>%z</b>	(output only.) Offset from Greenwich; -0800 is 8 hours west of.
<b>%Z</b>	(output only.) Time zone as a character string (empty if not available).

Where leading zeros are shown they will be used on output but are optional on input. See **?strftime**.  
**as.POSIXct( strptime( , format= ) )**  
**format()**

## Math

**sin, cos, tan, asin, acos, atan, atan2, log, log10, exp**

## Basic Math Operations

---

`%%, %/%` ..... modulo/quotient, remainder

`max(x)` ..... maximum of the elements of **x**

`min(x)` ..... minimum of the elements of **x**

`range(x)` ..... id. then `c(min(x), max(x))`

`sum(x)` ..... sum of the elements of **x**

`diff(x)` ..... lagged and iterated differences of vector **x**

`prod(x)` ..... product of the elements of **x**

`mean(x)` ..... mean of the elements of **x**

`median(x)` ... median of the elements of **x**

`quantile(x, probs=)`  
sample quantiles corresponding to given probabilities (default: 0,.25,.5,.75,1)

`weighted.mean(x, w)`  
mean of **x** with weights **w**

`rank(x)` ..... ranks of the elements of **x**

`var(x)` ..... or `cov(x)` variance of the elements of **x** (calculated on  $n-1$ ); if **x** is a matrix or a data frame, the variance-covariance matrix is calculated

`sd(x)` ..... standard deviation of **x**

`cor(x)` ..... correlation matrix of **x** if it is a matrix or a data frame (1 if **x** is a vector)

`var(x, y)` ... or `cov(x, y)` covariance between **x** and **y**, or between the columns of **x** and those of **y** if they are matrices or data frames

`cor(x, y)` ... linear correlation between **x** and **y**, or correlation matrix if they are matrices or data frames

`round(x, n)` . rounds the elements of **x** to **n** decimals

`log(x, base)` computes the logarithm of **x** with base **base**

`scale(x)` ..... if **x** is a matrix, centers and reduces the data; to center only use the option `center=F`, to reduce only `scale=F` (by default `center=T`, `scale=T`)

`pmin(x,y,...)` a vector which *i*th element is the minimum of **x[i]**, **y[i]**, ...

`pmax(x,y,...)` id. for the maximum

`cumsum(x)` ... a vector which *i*th element is the sum from **x[1]** to **x[i]**

`cumprod(x)` .. id. for the product

`cummin(x)` ... id. for the minimum

`cummax(x)` ... id. for the maximum

## Complex Numbers

---

`union(x,y)`, `intersect(x,y)`, `setdiff(x,y)`, `setequal(x,y)`

`is.element(el, set)`  
“set” functions

`Re(x)` ..... real part of a complex number

`Im(x)` ..... imaginary part

`Mod(x)` ..... modulus; `abs(x)` is the same

`Arg(x)` ..... angle in radians of the complex number

`Conj(x)` ..... complex conjugate

`convolve(x,y)`  
compute the several kinds of convolutions of two sequences

`fft(x)` ..... Fast Fourier Transform of an array

`mvfft(x)` ..... FFT of each column of a matrix

`filter(x, filter)`  
applies linear filtering to a univariate time series or to each series separately of a multivariate time series

Many math functions have a logical parameter `na.rm=F` to specify missing data (NA) removal.

## Matrices

---

`%o%`, `outer()` outer products on arrays

`kronecker` ... kronecker products on arrays

`t(x)` ..... transpose

`diag(x)` ..... diagonal

`%*%` ..... matrix multiplication

`solve(a,b)` .. solves  $a \%*\% x = b$  for **x**

`solve(a)` ..... matrix inverse of **a**

`rowsum(x)` ... sum of rows for a matrix-like object;

`rowSums(x)` .. is a faster version

`colsum(x)` ... sum of columns for a matrix-like object;

`colSums(x)` .. id. for columns

`rowMeans(x)` . fast version of row means

`colMeans(x)` . id. for columns

## Apply functions to elements

---

The base apply family of function is standardized and parallelized by the `plyr` package.

`apply(X, INDEX, FUN=)`  
a vector or array or list of values obtained by applying a function **FUN** to margins (**INDEX**) of **X**

`lapply(X, FUN)`  
apply **FUN** to each element of the list **X**

`tapply(X, INDEX, FUN=)`  
apply **FUN** to each cell of a ragged array given by **X** with indexes **INDEX**

`by(data, INDEX, FUN)`  
apply **FUN** to data frame **data** subsetted by **INDEX**

## The 6 common higher-order functions

---

`Reduce(f, x, init, right = F, accumulate = F)`

`Filter(f, x)`

`Find(f, x, right = F, nomatch = NULL)`

`Map(f, ...)`

`Negate(f)`

`Position(f,x,right = F,nomatch = NA_integer_)`

## Others

---

`optimise()` .. One Dimensional Optimization

**merge(a,b)** .. merge two data frames by common columns or row names

**xtabs(a b,data=x)**  
a contingency table from cross-classifying factors

**aggregate(x,by,FUN)**  
splits the data frame **x** into subsets, computes summary statistics for each, and returns the result in a convenient form; **by** is a list of grouping elements, each as long as the variables in **x**

**stack(x, ...)** transform data available as separate columns in a data frame or list into a single column

**unstack(x, ...)**  
inverse of **stack()**

**reshape(x, ...)**  
reshapes a data frame between 'wide' format with repeated measurements in separate columns of the same record and 'long' format in separate records

use: (direction="wide") or (direction="long")

### Optimization and model fitting

**optim(par, fn, method = c("Nelder-Mead", "BFGS", ...))** general purpose optimization; **par** is initial values, **fn** is function to optimize (normally minimize)

**nlm(f,p).....** minimize function **f** using a Newton-type algorithm with starting values **p**

**lm(formula)** . fit linear models; **formula** is typically of the form **response termA + termB + ...**; use **I(x\*y)** + **I(x^2)** for terms made of nonlinear components

**glm(formula,family=)**  
fit generalized linear models, specified by giving a symbolic description of the linear predictor and a description of the error distribution

see ?family: **family** is a description of the error distribution and link function to be used in the model

**nls(formula)** nonlinear least-squares estimates of the nonlinear model parameters

**approx(x,y=)** linearly interpolate given data points; **x** can be an xy plotting structure

**spline(x,y=)** cubic spline interpolation

**loess(formula)**  
fit a polynomial surface using local fitting

Many of the formula-based modeling functions have several common arguments: **data=** the data frame for the formula variables, **subset=** a subset of variables used in the fit, **na.action=** action for missing values: "**na.fail**", "**na.omit**", or a function.

### Statistics

**help.search("test")** gives you a range of validity tests such as **t.test()**, **binom.test()**, **prop.test()**, **power.t.test()**, **pairwise.t.test()**, ...

### Model Analysis

The following generics often apply to model fitting functions

**predict(fit,...)**  
predictions from **fit** based on input data

**df.residual(fit)**  
returns the number of residual degrees of freedom

**coef(fit)** ... returns the estimated coefficients (sometimes with their standard-errors)

**residuals(fit)**  
returns the residuals

**deviance(fit)**  
returns the deviance

**fitted(fit)** . returns the fitted values

**logLik(fit)** . computes the logarithm of the likelihood and the number of parameters

**AIC(fit).....** computes the Akaike information criterion or AIC

**aov(formula)** analysis of variance model

**anova(fit,...)**  
analysis of variance (or deviance) tables for one or more fitted model objects

**density(x)** .. kernel density estimates of **x**

### Distributions

**rnorm(n, mean=0, sd=1)**  
Gaussian (normal)

**rexp(n, rate=1)**  
exponential

**rgamma(n, shape, scale=1)**  
gamma

**rpois(n, lambda)**  
Poisson

**rweibull(n, shape, scale=1)**  
Weibull

**rcauchy(n, location=0, scale=1)**  
Cauchy

**rbeta(n, shape1, shape2)**  
beta

**rt(n, df)** ... 'Student' (*t*)

**rf(n, df1, df2)**  
Fisher-Snedecor (*F*) ( $\chi^2$ )

**rchisq(n, df)**  
Pearson

**rbinom(n, size, prob)**  
binomial

**rgeom(n, prob)**  
geometric

**rhyper(nn, m, n, k)**  
hypergeometric

**rlogis(n, location=0, scale=1)**  
logistic

**rlnorm(n, meanlog=0, sdlog=1)**  
lognormal

**rnbinom(n, size, prob)**  
negative binomial

**runif(n, min=0, max=1)**  
uniform

```
rwilcox(nn, m, n)
      rsignrank(nn, n) Wilcoxon's statistics
```

All these functions can be used by replacing the letter **r** with **d**, **p** or **q** to get, respectively, the probability density (**dfunc**(**x**, ...)), the cumulative probability density (**pfunc**(**x**, ...)), and the value of quantile (**qfunc**(**p**, ...)), with  $0 < p < 1$ ).

## Programming

Use curly braces **{}** around statements

```
function( arglist ) expr # function definition
return(value) if(cond) expr
if(cond) cons.expr else alt.expr
for(var in seq) expr
while(cond) expr
repeat expr
break
next
```

```
ifelse(test, yes, no)
      a value with the same shape as test
      filled with elements from either yes or no
do.call(funname, args)
      executes a function call from the name
      of the function and a list of arguments
      to be passed to it
```

## Plotting

```
plot(x) ..... plot of the values of x (on the y-axis)
                  ordered on the x-axis
plot(x, y) .. bivariate plot of x (on the x-axis) and
               y (on the y-axis)
hist(x) ..... histogram of the frequencies of x
barplot(x) .. histogram of the values of x; use horiz=F
               for horizontal bars
dotchart(x) . if x is a data frame, plots a Cleveland
               dot plot (stacked plots line-by-line and
               column-by-column)
pie(x) ..... circular pie-chart
boxplot(x) .. “box-and-whiskers” plot
```

```
sunflowerplot(x, y)
      id. than plot() but the points with
      similar coordinates are drawn as flow-
      ers which petal number represents the
      number of points
```

```
stripplot(x) plot of the values of x on a line (an al-
               ternative to boxplot() for small sam-
               ple sizes)
```

```
coplot(x~| z)
      bivariate plot of x and y for each value
      or interval of values of z
```

```
interaction.plot (f1, f2, y)
      if f1 and f2 are factors, plots the means
      of y (on the y-axis) with respect to the
      values of f1 (on the x-axis) and of f2
      (different curves); the option fun al-
      lows to choose the summary statistic
      of y (by default fun=mean)
```

```
matplot(x,y) bivariate plot of the first column of x
              vs. the first one of y, the second one of
              x vs. the second one of y, etc.
```

```
fourfoldplot(x)
      visualizes, with quarters of circles, the
      association between two dichotomous
      variables for different populations (x
      must be an array with dim=c(2, 2,
      k), or a matrix with dim=c(2, 2) if
      k = 1)
```

```
assocplot(x) Cohen–Friendly graph showing the de-
               viations from independence of rows and
               columns in a two dimensional contin-
               gency table
```

```
mosaicplot(x)
      ‘mosaic’ graph of the residuals from a
      log-linear regression of a contingency
      table
```

```
pairs(x) ..... if x is a matrix or a data frame, draws
                  all possible bivariate plots between the
                  columns of x
```

```
plot.ts(x) .. if x is an object of class "ts", plot of
               x with respect to time, x may be mul-
               tivariate but the series must have the
               same frequency and dates
```

```
ts.plot(x) .. id. but if x is multivariate the series
               may have different dates and must have
               the same frequency
```

```
qqnorm(x) ... quantiles of x with respect to the values
               expected under a normal law
```

```
qqplot(x, y) quantiles of y with respect to the quan-
               tiles of x
```

```
contour(x, y, z)
      contour plot (data are interpolated to
      draw the curves), x and y must be vec-
      tors and z must be a matrix so that
      dim(z)=c(length(x), length(y)) (x
      and y may be omitted)
```

```
filled.contour(x, y, z)
      id. but the areas between the contours
      are coloured, and a legend of the colours
      is drawn as well
```

```
image(x, y, z)
      id. but with colours (actual data are
      plotted)
```

```
persp(x, y, z)
      id. but in perspective (actual data are
      plotted)
```

```
stars(x) ..... if x is a matrix or a data frame, draws
                  a graph with segments or a star where
                  each row of x is represented by a star
                  and the columns are the lengths of the
                  segments
```

```
symbols(x, y, ...)
      draws, at the coordinates given by x
      and y, symbols (circles, squares, rect-
      angles, stars, thermometres or “box-
      plots”) which sizes, colours ... are spec-
      ified by supplementary arguments
```

```
termplot(mod.obj)
      plot of the (partial) effects of a regres-
      sion model (mod.obj)
```

## Plot Modifiers

The following parameters are common to many plotting functions

```
add=F      if TRUE superposes the plot on
            the previous one (if it exists)
```

<code>axes=T</code>	if <b>FALSE</b> does not draw the axes and the box
<code>type="p"</code>	specifies the type of plot, " <b>p</b> ": points, " <b>l</b> ": lines, " <b>b</b> ": points connected by lines, " <b>o</b> ": id. but the lines are over the points, " <b>h</b> ": vertical lines, " <b>s</b> ": steps, the data are represented by the top of the vertical lines, " <b>S</b> ": id. but the data are represented by the bottom of the vertical lines
<code>xlim=, ylim=</code>	specifies the lower and upper limits of the axes, for example with <code>xlim=c(1, 10)</code> or <code>xlim=range(x)</code>
<code>xlab=, ylab=</code>	annotates the axes, must be variables of mode character
<code>main=</code>	main title, must be a variable of mode character
<code>sub=</code>	sub-title (written in a smaller font)

## Low-level plotting commands

<code>dev.new()</code> ...	open a new graphics device (typically a window). see similar in help.
<code>points(x, y)</code>	adds points (the option <code>type=</code> can be used)
<code>lines(x, y)</code>	id. but with lines
<code>text(x, y, labels...)</code>	adds text given by <code>labels</code> at coordinates (x,y); a typical use is: <code>plot(x, y, type="n"); text(x, y, names)</code>
<code>mtext(text, side=3, line=0, ...)</code>	adds text given by <code>text</code> in the margin specified by <code>side</code> (see <code>axis()</code> below); <code>line</code> specifies the line from the plotting area
<code>segments(x0, y0, x1, y1)</code>	draws lines from points (x0,y0) to points (x1,y1)

<code>arrows(x0, y0, x1, y1, angle= 30, code=2)</code>	id. with arrows at points (x0,y0) if <code>code=2</code> , at points (x1,y1) if <code>code=1</code> , or both if <code>code=3</code> ; <code>angle</code> controls the angle from the shaft of the arrow to the edge of the arrow head
<code>abline(a,b)</code>	draws a line of slope <code>b</code> and intercept <code>a</code>
<code>abline(h=y)</code>	draws a horizontal line at ordinate <code>y</code>
<code>abline(v=x)</code>	draws a vertical line at abscissa <code>x</code>
<code>abline(lm.obj)</code>	draws regression line given by <code>lm.obj</code>
<code>rect(x1, y1, x2, y2)</code>	draws a rectangle which left, right, bottom, and top limits are <code>x1</code> , <code>x2</code> , <code>y1</code> , and <code>y2</code> , respectively
<code>polygon(x, y)</code>	draws a polygon linking the points with coordinates given by <code>x</code> and <code>y</code>
<code>legend(x, y, legend)</code>	adds the legend at the point (x,y) with the symbols given by <code>legend</code>
<code>title()</code> .....	adds a title and optionally a sub-title
<code>axis(side, vect)</code>	adds an axis at the bottom ( <code>side=1</code> ), on the left ( <code>2</code> ), at the top ( <code>3</code> ), or on the right ( <code>4</code> ); <code>vect</code> (optional) gives the abscissa (or ordinates) where tick-marks are drawn
<code>rug(x)</code> .....	draws the data <code>x</code> on the <i>x</i> -axis as small vertical lines
<code>locator(n, type="n", ...)</code>	returns the coordinates ( <i>x</i> , <i>y</i> ) after the user has clicked <code>n</code> times on the plot with the mouse; also draws symbols ( <code>type="p"</code> ) or lines ( <code>type="l"</code> ) with respect to optional graphic parameters (...)
by default nothing is drawn: <code>type="n"</code>	

## Graphical parameters

These can be set globally with `par(...)`; many can be passed as parameters to plotting commands.

<code>adj</code> .....	controls text justification (0 left-justified, 0.5 centred, 1 right-justified)
<code>bg</code> .....	specifies the colour of the background (ex. : <code>bg="red"</code> , <code>bg="blue"</code> , ... the list of the 657 available colours is displayed with <code>colors()</code> )
<code>bty</code> .....	controls the type of box drawn around the plot, allowed values are: " <code>o</code> ", " <code>l</code> ", " <code>7</code> ", " <code>c</code> ", " <code>u</code> " or " <code>J</code> " (the box looks like the corresponding character) if <code>bty="n"</code> : the box is not drawn
<code>cex</code> .....	a value controlling the size of texts and symbols with respect to the default; the following parameters have the same control for numbers on the axes, <code>cex.axis</code> , the axis labels, <code>cex.lab</code> , the title, <code>cex.main</code> , and the sub-title, <code>cex.sub</code>
<code>col</code> .....	controls the color of symbols and lines; use color names e.g. " <code>red</code> ", " <code>blue</code> " or as " <code>#RRGGBB</code> " see: see <code>colors()</code> , <code>rgb()</code> , <code>hsv()</code> , <code>gray()</code> and <code>rainbow()</code> as for <code>cex</code> there are: <code>col.axis</code> , <code>col.lab</code> , <code>col.main</code> , <code>col.sub</code>
<code>font</code> .....	an integer which controls the style of text (1: normal, 2: italics, 3: bold, 4: bold italics) as for <code>cex</code> there are: <code>font.axis</code> , <code>font.lab</code> , <code>font.main</code> , <code>font.sub</code>
<code>las</code> .....	an integer which controls the orientation of the axis labels (0: parallel to the axes, 1: horizontal, 2: perpendicular to the axes, 3: vertical)
<code>lty</code> .....	controls the type of lines, can be an integer or string (1: " <code>solid</code> ", 2: " <code>dashed</code> ", 3: " <code>dotted</code> ", 4: " <code>dotdash</code> ", 5: " <code>longdash</code> ", 6: " <code>twodash</code> ", or a string of up to eight characters (between " <code>0</code> " and " <code>9</code> ") which specifies alternatively the length, in points or pixels, of the drawn elements and the blanks, for example <code>lty="44"</code> will have the same effect than <code>lty=2</code>

**lwd**..... a numeric which controls the width of lines, default 1

**mar**..... a vector of 4 numeric values which control the space between the axes and the border of the graph of the form `c(bottom, left, top, right)`, the default values are `c(5.1, 4.1, 4.1, 2.1)`

**mfcol**..... a vector of the form `c(nr,nc)` which partitions the graphic window as a matrix of **nr** lines and **nc** columns, the plots are then drawn in columns

**mfrow**..... id. but the plots are drawn by row

**pch**..... controls the type of symbol, either an integer between 1 and 25, or any single character within ""

**ps**..... an integer which controls the size in points of texts and symbols

**pty**..... a character which specifies the type of the plotting region, "s": square, "m": maximal

**tck**..... a value which specifies the length of tick-marks on the axes as a fraction of the smallest of the width or height of the plot; if `tck=1` a grid is drawn

**tcl**..... a value which specifies the length of tick-marks on the axes as a fraction of the height of a line of text (by default `tcl=-0.5`)

**xaxt**..... if `xaxt="n"` the *x*-axis is set but not drawn (useful in conjunction with `axis(side=1, ...)`)

**yaxt**..... if `yaxt="n"` the *y*-axis is set but not drawn (useful in conjunction with `axis(side=2, ...)`)

## Lattice (Trellis) graphics

Use `panel=` to define a custom panel function (see `apropos("panel")` and `?llines`). Lattice functions return an object of class `trellis` and have to be **printed** to produce the graph. Use `print(xyplot(...))` inside functions where automatic printing doesn't work. Use `lattice.theme` and `lset` to change Lattice defaults.

`xyplot(y~x)` . bivariate plots (with many functionalities)

`barchart(y~x)`  
histogram of the values of *y* with respect to those of *x*

`dotplot(y~x)` Cleveland dot plot (stacked plots line-by-line and column-by-column)

`densityplot(~x)`  
density functions plot

`histogram(~x)`  
histogram of the frequencies of *x*

`bwplot(y~x)` . "box-and-whiskers" plot

`qqmath(~x)` .. quantiles of *x* with respect to the values expected under a theoretical distribution

`stripplot(y~x)`  
single dimension plot, *x* must be numeric, *y* may be a factor

`qq(y~x)` ..... quantiles to compare two distributions, *x* must be numeric, *y* may be numeric, character, or factor but must have two 'levels'

`splom(~x)` ... matrix of bivariate plots

`parallel(~x)` parallel coordinates plot

`levelplot(z~x*y|g1*g2)`  
coloured plot of the values of *z* at the coordinates given by *x* and *y* (*x*, *y* and *z* are all of the same length)

`wireframe(z~x*y|g1*g2)`  
3d surface plot

`cloud(z~x*y|g1*g2)`  
3d scatter plot

In the normal Lattice formula, `y x|g1*g2` has combinations of optional conditioning variables *g1* and *g2* plotted on separate panels. Lattice functions take many of the same arguments as base graphics plus also `data=` the data frame for the formula variables and `subset=` for subsetting.

This TeXfile is based on Gabriel B. Burcas © `git-qrc.tex` and has then been modified to my own use.