

# **Introduction to R**

**Benedikt Brors**

**Dept. Intelligent Bioinformatics Systems**

**German Cancer Research Center**

# What is R?

- R is a statistical computing environment with graphics capabilities
- It is fully scriptable and contains elements of programming languages
- A very high number of statistical procedures have been made available for R
- R is completely free, open source, and available for windows, Unix\*es (including Linux and MacOS X), and also for MacOS  $\geq$  8.6

# Relation to S and S-Plus

- Originally, a statistical language called **S** has been developed by Bell Labs (now AT&T)
- Several versions of this language have been issued, the latest being S4
- It has been licensed to Insightful Inc. The commercial software is called S-Plus
- Starting from 1997, the language has been reimplemented in a free software called **R**
- Though not fully compatible to S-Plus, much of the code runs also in R without alterations. Changes are mainly in graphics commands

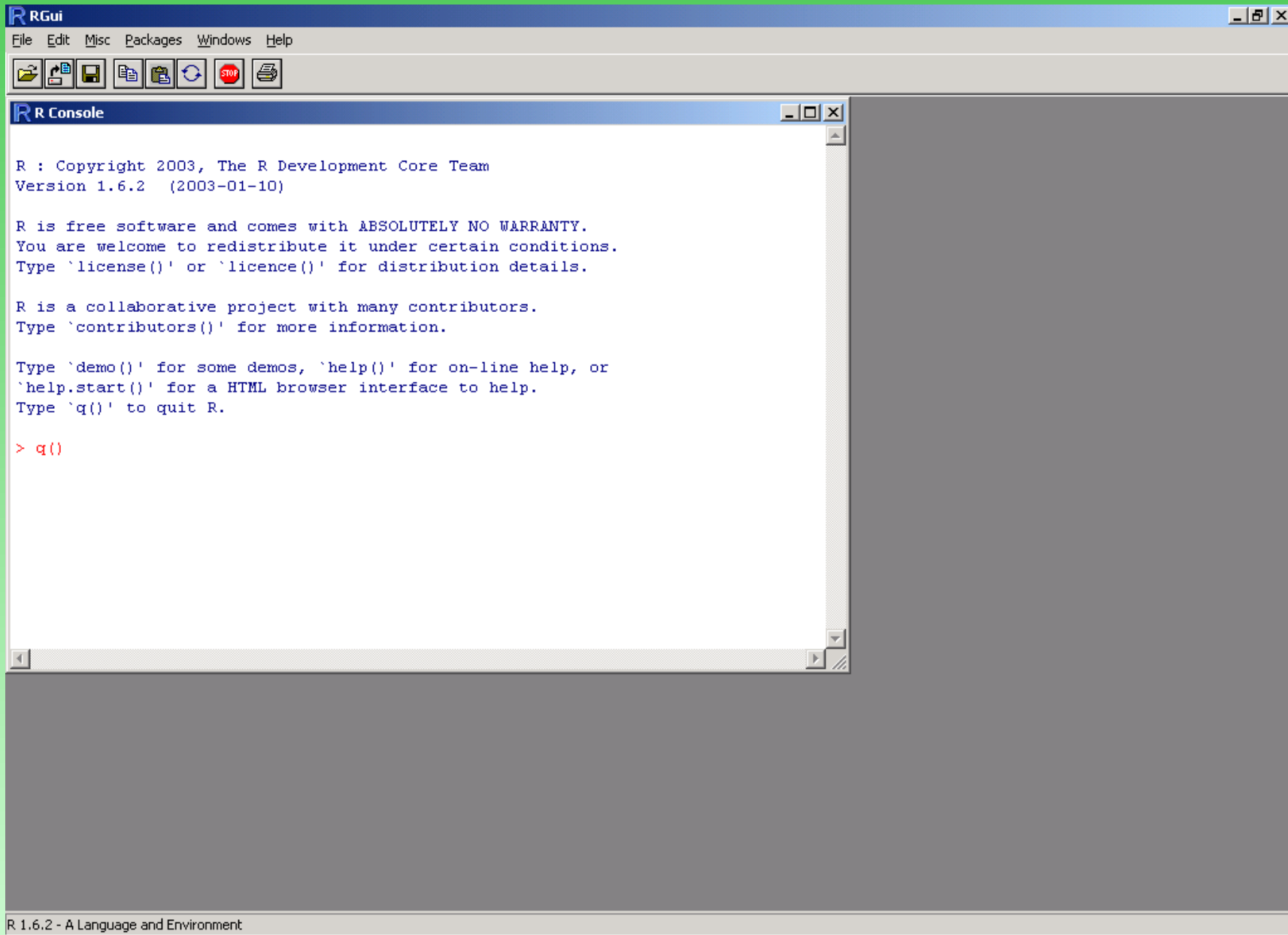
# Some of R's features

- effective data handling and storage facility
- suite of operators for calculations on arrays, in particular matrices
- large, coherent, integrated collection of intermediate tools for data analysis
- graphical facilities for data analysis and display either on-screen or on hardcopy
- well-developed, simple and effective programming language which includes conditionals, loops, user-defined recursive functions and input and output facilities.

# What R is NOT

- It is not a database, but it can connect to DBMS (Postgres, Oracle)
- No graphical user interface (interfaces to JAVA and Tcl/Tk exist)
- no spreadsheet view of data (connection to Excel possible)
- as an interpreted language, some procedures may be slow; however, compiled C, C++ or FORTRAN code can be called from within R

# Starting R



The screenshot shows the RGui application window. The title bar reads "RGui" and the menu bar includes "File", "Edit", "Misc", "Packages", "Windows", and "Help". Below the menu bar is a toolbar with icons for file operations and execution. The main window contains an "R Console" pane with the following text:

```
R : Copyright 2003, The R Development Core Team
Version 1.6.2 (2003-01-10)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for a HTML browser interface to help.
Type 'q()' to quit R.

> q()
```

At the bottom of the window, the status bar displays "R 1.6.2 - A Language and Environment".

# Objects

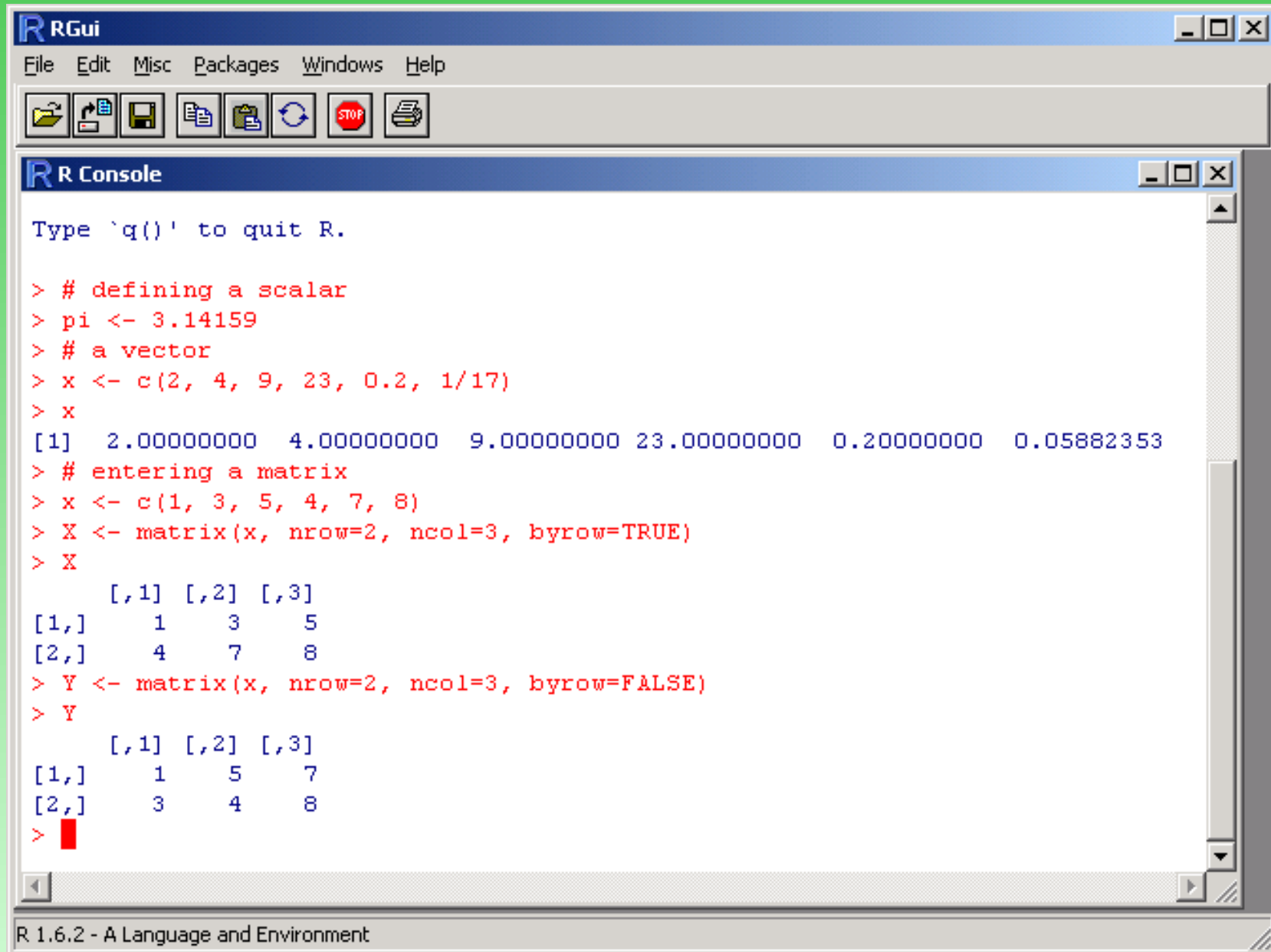
- Everything in R is an *object*. Objects mostly hold data, or function code
- All functions are written with round brackets, even if they don't take arguments, e.g. `q()` [quit], `ls()` [list workspace], or `dir()` [list working directory]
- if the name of an object is entered without assignment, or a function without brackets, the contents of this object is printed. Caution: For large objects (e.g. matrices), this will take a long time!
- Some of the functions are polymorphic, i.e. they will behave differently if they operate on different objects. The most important examples are `print`, `summary`, and `plot`

# Data types and data structures

- Some important data types are *numeric*, *character*, and *logical*
- In R, unlike in other programming languages, there is only one internal representation for numbers (double). There is no need to declare numbers explicitly as `int`, `longint`, `float` etc.
- Numbers can be single numbers (*constants*, *scalars*), vectors (ordered list of numbers), matrices (2-dimensional array of numbers), or arrays (any-dimensional array of numbers)



# How to enter scalars, vectors, or matrices



The screenshot shows the RGui window with the R Console. The console displays the following code and output:

```
Type `q()` to quit R.

> # defining a scalar
> pi <- 3.14159
> # a vector
> x <- c(2, 4, 9, 23, 0.2, 1/17)
> x
[1] 2.00000000 4.00000000 9.00000000 23.00000000 0.20000000 0.05882353
> # entering a matrix
> x <- c(1, 3, 5, 4, 7, 8)
> X <- matrix(x, nrow=2, ncol=3, byrow=TRUE)
> X
      [,1] [,2] [,3]
[1,]    1    3    5
[2,]    4    7    8
> Y <- matrix(x, nrow=2, ncol=3, byrow=FALSE)
> Y
      [,1] [,2] [,3]
[1,]    1    5    7
[2,]    3    4    8
> █
```

R 1.6.2 - A Language and Environment

# Operators and arithmetic functions

- Some important operators are:

| type       | examples                                   |
|------------|--|
| arithmetic | + - * / ^ %% (modulo) %*% (matrix product) |
| comparison | < > <= >= !=                               |
| logical    | & (AND)   (OR) ! (NOT)                     |
| assignment | <- (also = in R $\geq 1.5$ )               |

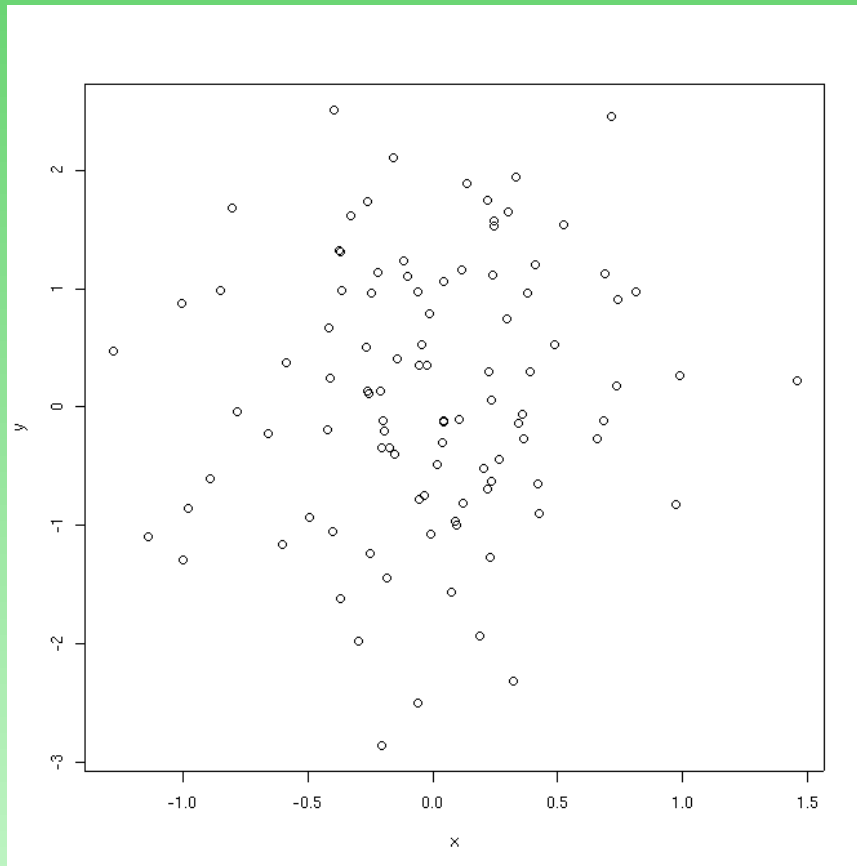
- 'usual' arithmetic functions exist (`sin()`, `cos()`, `exp()`, `log()`, `log10()`, `abs()`, `sqrt()`)
- unless for special operators (`%...%`), arithmetic operators and functions operate element-wise on vectors and matrices

# Sequences and random numbers

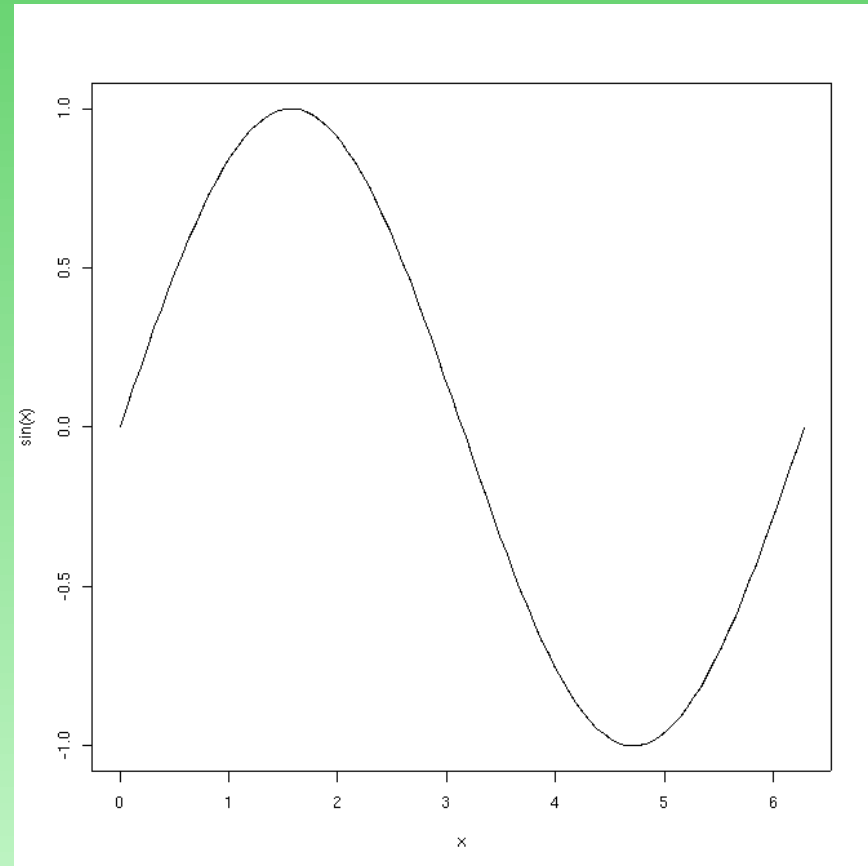
- Sequences are produced by `seq()`. Its arguments are `from`, `to`, and (optional) `by` (increment) or `length`. A shortcut for integer series of increment 1 is e.g. `1:100`.
- There are a number of functions for producing random numbers according to various distributions, e.g. `runif()` (uniform), `rnorm()` (normal), `rbeta()` (beta)
- The default random number generator or random seed are quite sensible, so there is no need to change this unless you're doing very special simulations

# Simple plotting

```
x <- rnorm(100, sd=0.5)
y <- rnorm(100, sd=1.2)
plot(x,y)
```



```
x <- seq(0,2*pi, length=100)
plot(x, sin(x), type="l")
```

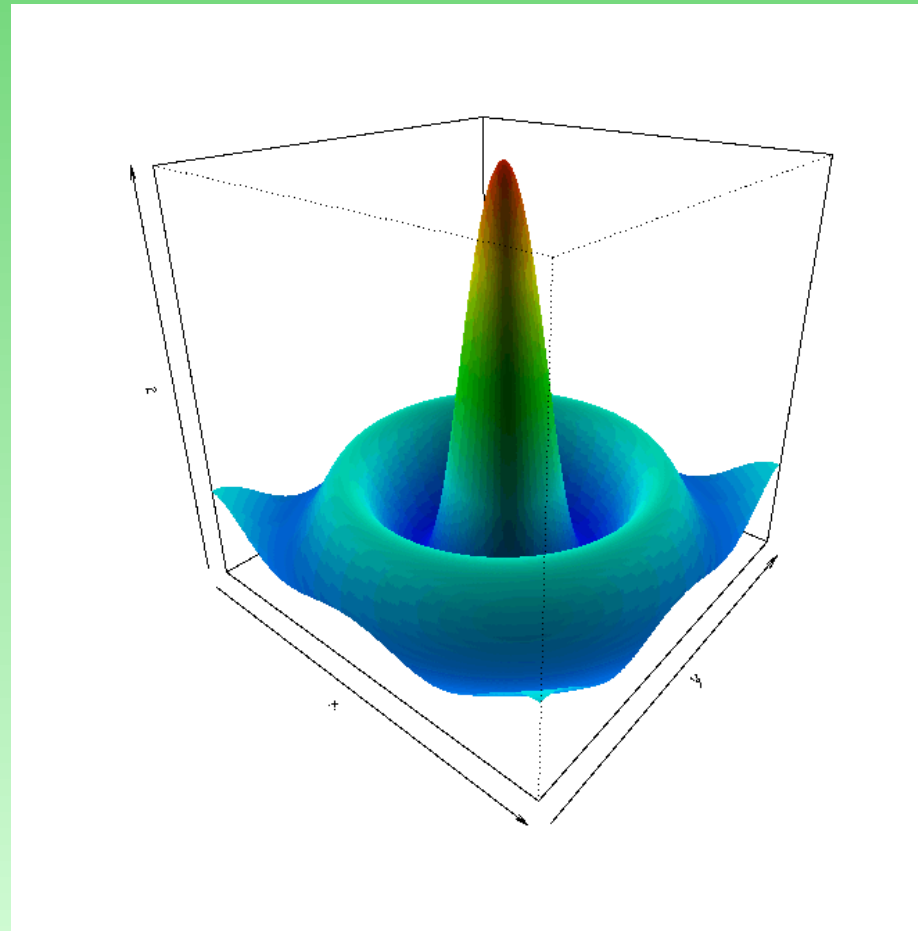


# Plotting capabilities

- Overview statistics: histogram, boxplot, quantile-quantile plot, scatterplot matrices
- 2D and 3D-scatterplots, barplots, line plots, 3D wireframe plots
- contour plots, levelplots, image plots
- large number of graphics parameters changeable

# Example of complex plotting

```
x <- seq(-10,10, by=0.04); y <- seq(-10,10, by=0.04)
f <- function(x,y) {
  r <- sqrt(x^2 + y^2) + 1e-10
  return( sin(r)/r ) }
z <- outer(x,y,f)
persp(x,y,z, theta=40, phi=25, col=fill, border=NA, shad=0.3)
```



# Missing values and special numbers

- As missing values are quite frequent in statistics, a special symbolic value has been introduced: `NA`. Please note:
  - ★ `NA` is NOT the same as `0`
  - ★ `NA` is NOT the same as `" "`
  - ★ `NA` is NOT the same as `FALSE`
- Operations on objects that contain an `NA` usually return `NA`, e.g.

```
> x <- c(2, 5, 7, NA, 3)
> sum(x)
[1] NA
```

- Often an argument `na.rm` or `na.action` can be supplied to drop NA values:

```
> x <- c(2, 5, 7, NA, 3)
> sum(x, na.rm=TRUE)
[1] 17
```

- For infinity and undefined numbers ( $0/0$ ) exist special values, too:

`Inf`, `-Inf`, `NaN`



# Lists and factors

- vectors and matrices only hold data of the same type (numerical, character, logical)
- Lists can hold data of different types:

```
doe <- list(name="john", age=28, married=FALSE)
doe$name
[1] "john"
```

- Lists are usually accessed by names, vectors and matrices by indices (see below)
- In statistics, character vectors exist that can take only one of several values, e.g. "M" or "F" for sex, or "low", "medium" or "high" for blood pressure. This is encoded in a data structure called *factor*. The different possible values are called *levels*.

```
bp <- factor(c(0,0,1,2,2,1,2,0))
levels(bp) <- c("low", "medium", "high")
> bp
 [1] low    low    medium high   high   medium high   low
Levels: low medium high
```

# Data frames

- In statistics, data come often as a table containing numbers and descriptive text. This is read into a *data frame*, a matrix containing numerical as well as character columns or rows. While reading in data from files, data are converted to appropriate types. If no column or row headers are supplied, these are filled in by defaults.
- Data frames can be explicitly casted to other data types. For example, `as.numeric()` will force numeric interpretation of values. You shouldn't apply this to strings/characters, of course!

# Data frames (continued)

- Example: diagnostic test on leukemia:

```
>      wbc    test  survivaltime
1     2300 present      65
2       750 present     156
3     4300 present     100
4     2600 present     134
5     6000 present      16
6    10500 present     108
7    10000 present     121
8    17000 present       4
9     5400 present      39
10    7000 present     143
11    9400 present      56
12   32000 present      26
13   35000 present      22
      (...)
```

# Indexing

- Vectors and matrices are indexed numerically. Most unlike C, JAVA and PERL, indices start at 1 (NOT at 0).

```
> x[2]
```

```
[1] 5
```

```
> x[2,7]
```

```
[1] 10
```

- Ranges may be given, as well as any combination of row and column indices; if all indices are wanted, nothing may be specified for this dimension. `x[1:10, ]` will give the first 10 rows and all columns of `x`, `x[c(1,3,5), 1:2]` the 1st, 3rd and 5th row and the first two columns

- Indices may be dropped by adding a minus sign:  $x[-(1:2), ]$  will give all except the first two rows (and all columns)
- Logical indices may be given as well:  $x[x>0]$  will only return the positive values of  $x$

# Importing and exporting data

- Data come usually as tab-delimited text files, or can be converted to such files
- The functions `read.delim()`, `read.table()` and `read.csv()` read these files and store data in a data frame. To get data types correct, additional arguments (`as.is`, `quote`) may be needed – read the help files. But getting your data into R can be as simple as

```
my.data.frame <- read.delim("filename.txt")
```

- You can write tab-delimited files by using `write.table()`

- All variables and functions in the current workspace (i.e. computer RAM memory) can be saved by `save()`. It will be then written in some binary format called XDR.
- If you save your data while exiting, it is always stored in the file “.RData”. Thus, if you work in the same directory on several occasions, this file is frequently overwritten. This may not be what you want (because old data are lost)



# Functions

- A lot of predefined functions exist. However, if you want to write your own function, which may also facilitate repetitive tasks, this is quite easy.
- The general syntax is

```
my.function <- function(a,b,c,...) {  
  < definition of function >  
  return( result ) }
```

In round brackets are *arguments* of the function. Some default values may be defined by e.g. `function(a, b=50, c=TRUE)`. The return value doesn't need to be explicitly defined, the result of the last calculation is returned by default

# Conditional expressions

- if-then-else statements can be made both interactively and (more frequently) while writing functions:

```
if (logical expression) {  
    statements  
} else {  
    alternative statements  
}
```

The `else` branch is optional

# Loops

- Loops (repetitive tasks) can be defined with

```
fact <- 1
for (i in 1:10) {
  fact <- fact*i # calculates factorial 10!
}
```

```
i <- 2
while (i < 10) {
  i <- i*sqrt(i)
  print(paste("i is now", i, "\n"))
}
```

# lapply, sapply, apply

- If loops contain complex functions and are called a large number of times, they become very slow. Using one of `lapply()`, `sapply()` or `apply()` will be much faster
- `lapply()` operates on lists, and returns a list with the same names, containing the results of this operation.

```
> li = list("klaus", "martin", "georg")
> lapply( li, toupper)
[[1]]
 [1] "KLAUS"
[[2]]
 [1] "MARTIN"
[[3]]
 [1] "GEORG"
```

- `sapply()` works similar, but casts the results to a vector.

# apply

- `apply()` works on either columns or rows of a matrix. Its syntax is

```
apply(matrix, margin, function, args)
```

where `margin=1` works on rows, `margin=2` on columns, and `args` are additional arguments to `function`, which must operate on vectors

- Example:

```
> x
      [,1] [,2] [,3]
[1,]    5    7    0
[2,]    7    9    8
[3,]    4    6    7
[4,]    6    3    5
```

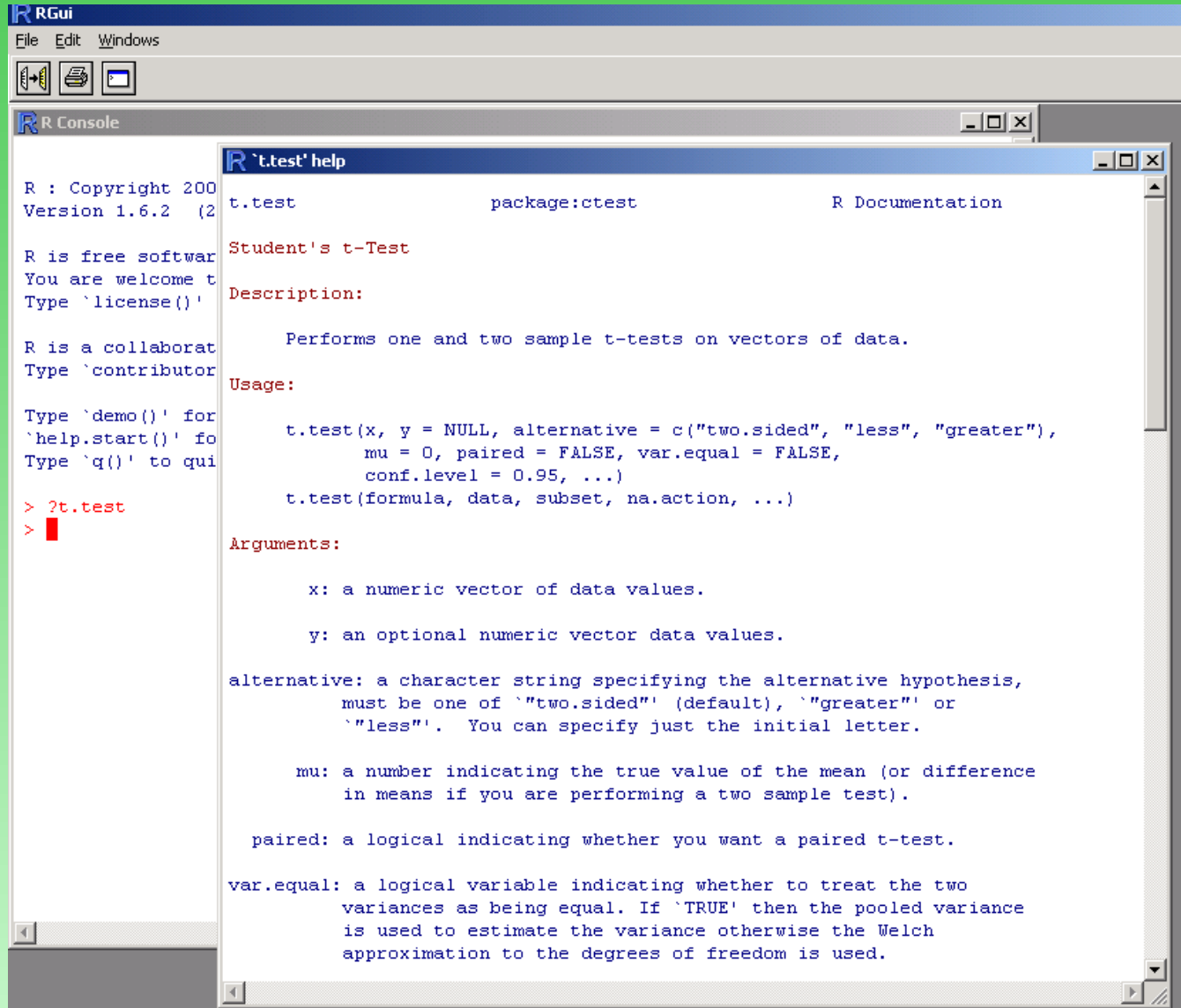
```
> apply( x, 1, sum)
[1] 12 24 17 14
```

```
> apply( x, 2, sum)
[1] 22 25 20
```

# Getting help

- A number of introductory documents come together with R, as well as a reference manual
- However, for daily work the online help is most useful: type `?t.test` or `help(t.test)` if you want help on a specific function
- `help.search()` allows for keyword search, `help.start()` opens an HTML help page.
- For very special questions, you may want to consult the R mailing list at <http://www.r-project.org> (searchable versions exist)

# R help window



The screenshot shows the R GUI interface. The main window is titled 'R Console' and contains the following text:

```
R : Copyright 200
Version 1.6.2 (2

R is free softwar
You are welcome t
Type `license()'

R is a collaborat
Type `contributor

Type `demo()' for
`help.start()' fo
Type `q()' to qui

> ?t.test
> █
```

A smaller window titled 'R `t.test` help' is open in the foreground, displaying the help text for the `t.test` function:

```
t.test                package:cctest                R Documentation

Student's t-Test

Description:
  Performs one and two sample t-tests on vectors of data.

Usage:
  t.test(x, y = NULL, alternative = c("two.sided", "less", "greater"),
         mu = 0, paired = FALSE, var.equal = FALSE,
         conf.level = 0.95, ...)
  t.test(formula, data, subset, na.action, ...)

Arguments:
  x: a numeric vector of data values.

  y: an optional numeric vector data values.

alternative: a character string specifying the alternative hypothesis,
  must be one of "two.sided" (default), "greater" or
  "less". You can specify just the initial letter.

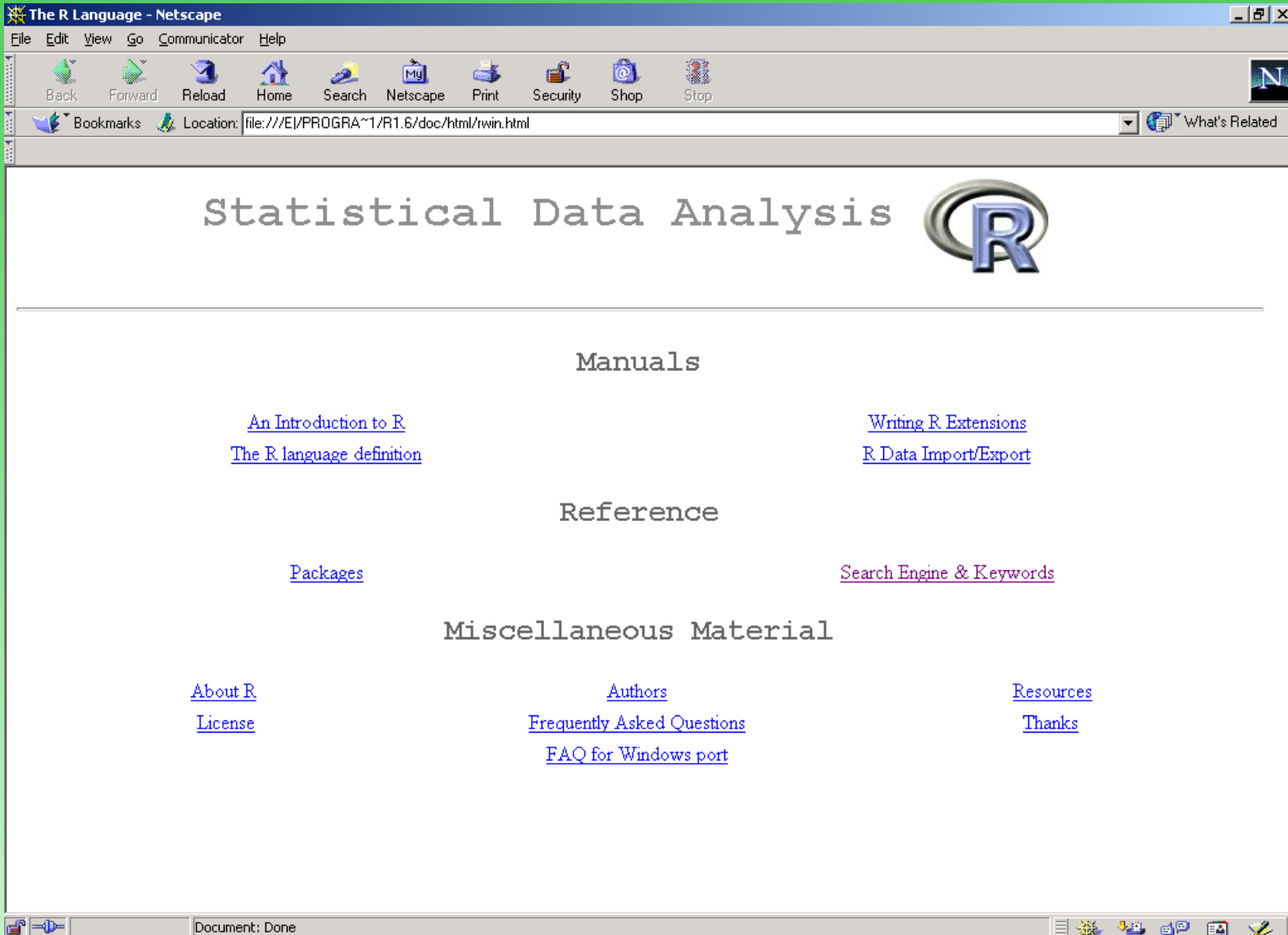
mu: a number indicating the true value of the mean (or difference
  in means if you are performing a two sample test).

paired: a logical indicating whether you want a paired t-test.

var.equal: a logical variable indicating whether to treat the two
  variances as being equal. If TRUE then the pooled variance
  is used to estimate the variance otherwise the Welch
  approximation to the degrees of freedom is used.
```



# R HTML help pages



# Web resources

- The R home page is at <http://www.r-project.org> (TU Vienna, Austria).
- The R program as well as packages (additional statistical program libraries) are available from CRAN (pronounce siih-ran), the Comprehensive R-Archive Network: <http://cran.r-project.org>. Numerous mirror sites exist.