

# R

R (and S, and S-Plus, another packaged based on S) is an interactive, interpretive, function language.

Available on Unix and MS Windows systems. Documentation exists in several volumes, and in an on-line help system.

Most statements are of the form

*variable* <- *function(...)*

or

*function(...)*

R is case sensitive.

# R Fundamentals

R objects:

variables,

vectors,

lists,

matrices,

arrays,

formulas,

factors (for statistical applications),

data frames (for statistical applications),

fits (for statistical applications)

R has an extensive set of functions, i.e. verbs. The specific meaning depends on the class of the object to which it is applied.

Objects are built by constructor functions:

```
> A <- matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9), nrow=3)
```

This object is a matrix and can participate in matrix multiplication and so on. Typing A results in

```
A =  
  1 4 7  
  2 5 8  
  3 6 9
```

# R Syntax, Functions

All actions are “functions”. A function name is followed by a set of parentheses to enclose arguments.

`help()` or `help(plot)`, e.g. or `?plot`

`q()`

The syntax is somewhat similar to C++, except that there are no special delimiters for statements; statements continue as necessary to complete the function.

Statements may be grouped by { and }.

Blanks are significant only when it makes sense for them to be.

The language is not strongly typed.

Numbers are generally single precision floating point numbers.

# Examples

```
> x <- c(2, 5, 3)
> x
> z1 <- 3 + 4*i
> z2 <- complex(real=3, imaginary=4)
> z <- scan(file="example.dat")
> y <- scan()
1: 21. 23. 26. 27. 27. 26.
7: 23. 23. 28. 28. 27. 26.
13: 24. 22. 27. 26. 28. 28.
19: 25. 23. 29. 28. 28. 26.
25:
> a <- 0:10
> b <- 0:10/10
> c <- seq(from=0, t=1, by=.1)
> plot(y,type="l")
```

# General Design

R deals with *functions*.

This dictates the syntax – no statement delimiters (but does use { and }).

Comments begin with #.

No fixed naming conventions; the wise user, however, adopts mnemonic conventions. Use periods to represent components.

Various ways of extracting components. \$ component operator:  
object\$member

See objects with objects().

# R Matrices

```
A <- matrix(c(1, 3, 5, 2, 4, 6), nrow=3)
x <- c(3, 4)
z <- A[2, ]
w <- A[2, 1:2]
v <- c(1, 2, 1)
b <- A%*%x;
cc <- v%*%A
D <- cbind(A, v)
E <- solve(D)
f <- solve(D, v)
H <- D%*%A
L <- D*D
M <- D^2
objects()
rm(...)
```

# Other Operators and Functions

< , <=, >, >=, ==, !=

&, |, !

sin, cos, ...

abs, Arg, sqrt, Re, Im, Conj,  
round, trunc, floor, ceiling, sign,  
%%,  
exp, log, log10

t, crossprod, solve,

sink("filename")

sink()

Plus lots of functions for statistical analyses.

# Programming

Conditionals:

```
if(x >= 3)
{
  y <- 2
  z <- 4
} else if (x <= 1)
  y <- 1
else
  y <- 5
```

# Programming

Loops:

```
for (i in seq(1,10,by=2)) {  
  for (j in 2:5) A[i,j] <- 0  
  x[i] <- 3  
}
```

```
i <- 1  
while (i < 5) {  
  A[i] <- i+1  
  i <- i+1  
}
```

# Programming

Loops are inefficient in R. Try to implement as statements involving vectors.

```
aa <- c(1:10)
bb <- c(rep(0,10))
x  <- ifelse(y>0,1,0)
```

Use `apply`.

# R Functions for Graphics

plot  
plot.factor  
pairs  
brush  
hist  
stem  
barplot  
persp  
faces  
stars  
matplot

And others. The actual appearance of the graph depends on the class of the object.

Arguments for functions may be required or optional. Most required ones are positional, many optional ones are keyword.

# R Functions for Standard Distributions

Some distributions:

- beta
- f
- gamma
- norm
- t
- unif

# R Functions for Standard Distributions

## Functions

- d – density
- p – cumulative probability
- q – quantile
- r – random number generation

Examples: `rnorm(25, 100, 8)` generates 25  $N(100,64)$  numbers  
`qf(.95, 5, 10)` the .95 quantile of an F with 5 and 10 degrees of freedom.

# The Object Orientation of R

Most of the functions of R are overloaded and are data-driven.

A new class can be defined by defining a constructor function that gives an identifier to the class:

```
factor <- function(x, levels = sort(unique(x)),
  labels = as.character(levels)) {
  y <- match(x, levels)
  names(y) <- names(x)
  levels(y) <- labels
  class(y) <- "factor"
  y
}
```

The function `factor` will construct an object of class “factor”.

# Interfacing R with C or C++

Reasons to do so:

- Availability of code
- Speed of compiled code

Reasons not to:

- Can pass only the simplest of arguments (objects lose their characteristics – must reconstruct more complicated objects)
- Increased complexity

# Interfacing R with C or C++

Strategy:

- Do most error checking and data management in R
- Do computationally intensive tasks in C
- I/O ... depends ...

These are the general considerations for linking systems.