An Introduction to R

Bjørn-Helge Mevik
(based on original by Antonio Mora, IMBV)

Research Computing Services, USIT, UiO

23. November 2011

What is R?

- A 'language and environment for statistical computing and graphics' (implements a dialect of the language 'S')
- Syntax is C-like, but philosophy is functional
- Focus on matrices and vectors
- Free, open-source (GPL)
- Well documented
- Command line based, but there are GUIs
- Latest version: 2.14.0 (November 2011)
- URL: `http://www.r-project.org/`

R Features

- Very many analysis methods available
- Scriptable and extensible
- Can be used interactively, process batch jobs or run as a script
- Bindings to many other systems/languages, e.g., Python, Perl, Matlab, *SQL, Excel
- Active user community with thousands of contributed packages

The R Command Line

- Normal prompt: >
- R reads and executes one line at a time (as long as it is syntactically complete)
- Waiting for more input: +
- To abort: `Ctrl-c` (MSWin/Mac: `Esc`).

Help!

This is probably the most important slide!

- `?mean` - help for a function
- `help.search("regression")` or simply `??regression` - search in your installed R
- `RSiteSearch("logistic")` - search the R web site
- `demo()` - list/run demos
- `vignette()` - list package vignettes
- `help.start()` - start help centre

Elementary Data Types

Basic elements: *numbers*, *strings*, *logicals*

```
> 42
[1] 42
> "a string"
[1] "a string"
> TRUE
[1] TRUE

> 42 + 13.5
[1] 55.5
> 42 > 13.5
[1] TRUE
> substr("a string", 3, 5)
[1] "str"
> ! TRUE
[1] FALSE
```

Basic Calculations

- Arithmetic: `+`, `-`, `*`, `/`, `%%` (modulus), `%*%` (matrix multiplication), etc.
- Other mathematical: `sqrt()`, `exp()`, `log()`, `sin()`, `cos()`, etc.
- Logical: `>`, `<`, `>=`, `<=`, `==`, `!=`, `&&` (and), `||` (or), `!` (not)
- Strings: `substr()`, `paste()`, `strsplit()`, `grep()`, etc.

See `?Syntax` and its 'See also' section.

Conversion of Data types

Types are converted as needed (when possible)

```
> FALSE + 1            # logical to number
[1] 1
> ! 0                  # number to logical
[1] TRUE
> substr(42, 1, 1)     # number to string
[1] "4"
```

but not else

```
> "42" + 3
Error in "42" + 3 : non-numeric argument to binary operator
```

'Side step': Variables

- Values can be stored in *variables*:

```
> mynum <- 42
> mynum + 13.5
[1] 55.5
> adj <- "interesting"
> sentence <- paste("Very", adj)
> sentence
[1] "Very interesting"
```

- Tip: use descriptive names; avoid single-character names.
- List all variables: `ls()`
- Show value of variable: `mynum` or `print(mynum)`
- Remove a variable: `rm()`, e.g., `rm(adj)`

Compound Data Types: Vectors

- Basic elements collected in *vectors*, *lists*, *matrices* and *data frames*
- Collection of *equal* types of elements: *vector*:

```
> 1:5
[1] 1 2 3 4 5
> c(42, 1:5)   # "c" for "concatenate"
[1] 42  1  2  3  4  5
> c("three", "small", "things")
[1] "three"  "small"  "things"
```

- Indexing:

```
> nums <- c(42, 33, 58, 1, 3.2)
> nums[2]
[1] 33
> nums[2:3]
[1] 33 58
> nums[c(1,3)]
[1] 42 58
```

Compound Data Types: Lists

- Collection of *different* types of elements: *list*:

```
> c(42, "Mary")        # Probably not what you want
[1] "42"   "Mary"
> list(42, "Mary")     # That's more like it!
[[1]]
[1] 42

[[2]]
[1] "Mary"
```

- Indexing: `[]` and `[[]]`:

```
> lag <- list(c(3,5), "string", rep(TRUE, 5))
> lag
> lag[2:3]    # sub list
> lag[3]      # still sub list
> lag[[3]]    # element
```

Compound Data Types: Matrices

- Two-dimensional collections: *matrices* and *data frames*
- Same element type (usually number): *matrix*:

```
> A <- matrix(1:9, ncol = 3, nrow = 3)
> A
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> B <- matrix(6:1, ncol = 2, nrow = 3)
> A %*% B
     [,1] [,2]
[1,]   54   18
[2,]   69   24
[3,]   84   30
```

- Indexing: `A[2,1]`

Compound Data Types: Data Frames

- *Data frame*: Think 'data set with (possibly) different types of variables'
- Object = row, variable = column

```
> WorkData <- data.frame(y = 10:13, x = c(2.2, 1.3,
+  4, 5.1), z = c(TRUE, TRUE, FALSE, TRUE))
> WorkData
   y   x     z
1 10 2.2  TRUE
2 11 1.3  TRUE
3 12 4.0 FALSE
4 13 5.1  TRUE
> WorkData$x
[1] 2.2 1.3 4.0 5.1
> WorkData[2,"y"]   # or WorkData[2,1]
[1] 11
```

Summary of common data types

- Atomic types: number, string, logical
- Compound types:

type	1-dim	2-dim	> 2 dim
same	vector	matrix	array
mixed	list	data frame	

- (There are other types as well. . .)

Saving and loading data

- Entire workspace or single variables in binary format:

```
> save(x, y, WorkData, file = "vars.RData") # single vars
> save.image("allvars.RData")               # all vars

> load("vars.RData")
```

- Matrices or data frames in text files:

```
> write.table(WorkData, file = "mytable.txt")
> newdata <- read.table("mytable.txt")
> class(newdata) # read.table always creates a data frame
```

- Vectors in text files:

```
> write(x, file = "data.txt")
> x2 <- scan("data.txt")
```

Functions

- Statistical summaries:

```
> x <- rnorm(50, 1, 2)
> mean(x)
> var(x)
> sum(x)
> cumsum(x)
> summary(x)
```

- Modelling functions (linear regression):

```
> mymod <- lm(y ~ x, data = WorkData)
> summary(mymod)
> plot(mymod)
```

Plotting

- The main plot function is `plot()`:

```
> plot(1:10, (1:10)^2)
> plot(1:10, (1:10)^2, type="l", col="red", main="quad")
> plot(y ~ x, data = WorkData)
> plot(WorkData)
```

- Most analysis methods have their `plot` method (`plot(mymod)`)
- Many specialised plot functions, e.g., `boxplot()`, `hist()`, `contour()`, `coplot()`, `levelplot()`, `persp()`. Many of these try to be 'smart' when handed a matrix or data frame.
- Some functions add to plots, e.g. `points()`, `lines()`, `abline()`, `title()`
- See `demo(graphics)`, `demo(image)`, `demo(persp)`, `demo(plotmath)`

Analysis: linear regression

```
> data(ToothGrowth)
> ?ToothGrowth
> coplot(len ~ dose | supp, data = ToothGrowth,
+   panel = panel.smooth,
+   xlab = "length vs dose, given type of supplement")
> tooth <- lm(len ~ dose * supp, data = ToothGrowth)
> plot(tooth)
> anova(tooth)
> tooth2 <- lm(len ~ (dose + I(dose^2)) * supp,
+   data = ToothGrowth)
> plot(tooth2)
> anova(tooth2)
> anova(tooth2, tooth)
```

Analysis: other types of analyses

ACE, ACF, Adaboot, adaptive rejection sampling, AIC, ANCOVA, Anderson-Darling K-sample test, Ansari-Bradley test, APT, ARFIMA, ARIMA, ARMA, Aster models, Asymptotic regression, asymptotic statistics, AVAS, AWS, BACCO, bagging, Bartletts test, Bayesian statistics, binomial test, boosting, Bootstrap, Box test, canonical correlations, contingency tables, factor analysis Fifty-fifty MANOVA, Fisher's exact test, Friedmans rank sum test, fuzzy clustering, Gibbs sampling, GLM, hierarchic clustering, imputation, Kalman filtering, K-means clustering, k-NN, Kolmogorov-Smirnov test, Kruskal-Wallis rank sum test, LDA, LOESS, logistic regression, logit models, LOWESS, MAD, Mahalanobis distance, MANCOVA, MANOVA, mixed models, multidimensional scaling, multiple comparison tests, non-linear optimization, non-linear regression, path modelling, PCA, PCR, Phillips-Perron test, PLSR, QDA, random effect models, running median smoothing, Shapiro-Wilk normality test, splines, time series, t test, varimax rotation, Wilcoxon signed rank test, ...

Control structures 1: if

R has several types of control structures: *if statements*, *loops*, *switch statements*.
If statements:

```
if (a > 1) { print("hello") }

if (length(x) > 5) {
    print("long")
} else {
    print("short")
}
```

Control structures 2: loops

For loops:

```
for (i in 1:10) { print(i) }
for (i in list("a", "b", TRUE)) { print(i) }
```

While loops:

```
num <- 1
while (num < 10) {
   print(num)
   num <- num * 2
}
```

Functions

- Container of a group of statements, returns a value
- R is a functional language => 'everything' is a function
- Arguments can be specified by name, skipped, and have default values
- See argument list: `args(rnorm)`
- See function definition: Just type its name, e.g. `ls`

Example:

```
> args(rnorm)
> rnorm(10, sd = 2) # versus
> rnorm(10, 0, 2)

mysum <- function(a, b) {
    a + b
}
mysum(1, 2)
```

Scripts and batch jobs

It is boring to type in commands over and over again, so we store them in plain text files (R *scripts*).
R scripts can be run in several ways:

- From inside R: `source("script.R")`
- From command line: `R CMD BATCH script.R`
- *As* a command: Put `#!/usr/bin/env Rscript` in the first line, and make the file executable (`chmod a+x script.R`).

For instance, `script.R`:

```
#!/usr/bin/env Rscript
print("Hello, world!")
```

R packages

- Collection of functions (and data) for a specific purpose
- List installed: `library()`
- Help about installed package: `library(help = package)`
- Use package: `library(package)`
- Thousands of contributed packages on `cran.r-project.org`
  - To simplify: Collected in 'task views'
  - Install with `install.packages("package")`

BioConductor

- Large collection of R packages for molecular biology, bioinformatics and systems biology
- Main categories:
  - *Annotation*: GO, Pathways, etc. . .
  - *AssayDomains*: CellBasedAssays, ChIPchip, CopyNumberVariants, CpGIsland, DNACopyNumber, DNAMethylation, ExonArray, GeneExpression, GeneticVariability, SNP, Transcription. . .
  - *AssayTechnologies*: Microarray, MassSpectrometry, SAGE, FlowCytometry, Sequencing, HighThroughputSequencing. . .
  - *BiologicalDomains*: CellBiology, Genetics, Proteomics
  - *Infrastructure*: DataImport, DataRepresentation, GraphsAndNetworks, GUI, Visualization
  - *Bioinformatics*: Clustering, Classification, MultipleComparisons, QualityControl. . .

Installing BioConductor

- Home page: `www.bioconductor.org`
- To get the standard distribution:

```
source("http://bioconductor.org/biocLite.R")
biocLite()
```

- To get additional packages:

```
source("http://bioconductor.org/biocLite.R")
biocLite(c("GenomicFeatures", "AnnotationDbi"))
```

Help!

This is probably the most important slide!

- `?mean` - help for a function
- `help.search("regression")` or simply `??regression` - search in your installed R
- `RSiteSearch("logistic")` - search the R web site
- `demo()` - list/run demos
- `vignette()` - list package vignettes
- `help.start()` - start help centre