

using R: an introduction

Adrian Waddell

University of Waterloo
Department of Statistics and Actuarial Science

September 8, 2010

About these Slides

These slides were written on behalf of the Department of Statistics and Actuarial Science at the University of Waterloo, Canada.

At the time of writing, the current software versions are

- ▶ GNU Emacs 23.1.1
- ▶ Eclipse SDK Version: 3.5.2
- ▶ R version 2.11.1
- ▶ ESS 5.11

There are more slides like these on our [homepage](#).

About R

S is a statistical **high-level** and **interpreted** programming language developed at the Bell laboratories around 1975 by John Chambers (see **history of S**). The commercial implementation of S is called S-PLUS and appeared in 1988. R is an open-source implementation of S and was created in the early nineties by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand. These days, R is maintained by the **R core team**.

Since R and S-PLUS have their own flavor they are considered to be two different dialects of the S language.

R has become very popular particularly in academia but also in industry (see **NY Times article** and **follow up**). Much of R's success story is due to all the packages written for R by the R-community.

What you will Learn from these Slides

Due to the popularity of R, many good manuals, tutorials and books have been written on R. We will referee some of these resources and then focus on conceptual aspects of the language and parts we believe lack easily available documentation (such as the `tk` package).

R is a powerful data analysis tool. R distinguishes itself in my opinion from other statistic packages like SAS and SPSS by offering many of the features and flexibilities programming languages like Python, Ruby and Perl have.

You can of course use R just to run predefined statistical procedures like linear regression or analysis of variance, however due to the very nature of good data analysis practises, you probably will experience the need to analyze aspects of your data according to your own ideas. Therefore it is good to have a grasp of Rs programming capabilities so you can acquire the details once you need to know them.

More about these Slides

Because of R's flexibility in coding, it is easy to acquire a bad programming style. I therefore will try to give good practice hints.

My subjective good practice hints will be highlighted with this symbol:



Vocabulary

You might find some terms confusing in the slides that follow:

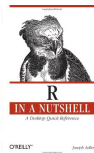
CRAN (short for *Comprehensive R Archive Network*) stands for an international network of servers that store the R software and all submitted packages (sometimes called CRAN packages). (see CRAN [website](#) and [server list](#)).

R prompt once R is started, the R prompt is indicated with a `>` at the beginning of a (command) line. The R prompt is where you can send code to R to be evaluated.

Where to Start

- ▶ **Download** and **install** R.
- ▶ Browse the **R website** to get an overview of the R project.
 - ▶ Especially the category *Documentation* in the navigation bar links to many great resources.
- ▶ At first, use R through the *R prompt* `>`. Once you understand some R basics, familiarize yourself with an IDE (like Emacs or Eclipse).
- ▶ Familiarize yourself with R using one of the **many** guides or tutorials. For example:
 - ▶ Official **R introduction**
 - ▶ **simpleR**, – Using R for Introductory Statistics by John Verzani
 - ▶ University of Waterloo **R tutorial**

Books



R in a Nutshell

Joseph Adler (O'Reilly)

This book is a good reference for R users and programmers. It covers the following four parts: R basics, the R language, working with data and statistics with R.



Modern Applied Statistics with S

W.N. Venables & B.D. Ripley (Springer)

Books: Springer Use R series

Springer started a book series called **Use R**. Each book in this series is aimed to cover the applied part of some statistical topic using R. These books can be downloaded for free by UW students [here](#).



At the time of writing this text, there were already 23 books in this series.

Before you continue reading these slides...

We will continue with the slides assuming that you understand the very basics of R. That is, you should understand

- ▶ how to start and end an R session
- ▶ how to assign a value to a variable: `a <- 2`
- ▶ check what variables you have in your *workspace* with `ls()`
- ▶ do simple math calculations: `sqrt(a^2)`
- ▶ create a vector with `c()`: `myVec <- c(3,4,5,6)`
- ▶ access elements of a vector: `myVec[3]` and `myVec[c(1,3)]`
- ▶ how to create a simple scatterplot with `plot()`: `plot(1:4,4:7)`
- ▶ import a data file with `read.table()`
- ▶ whats the difference between a *vector*, *matrix*, *data.frame* and a *list* data structure.

If these tasks seem alien to you, follow the instructions on the **this** slide. You can also get an **R book**. It is key that you know some source where you can look up the most common R tasks.

Working with R



There are many ways to interact with R. Some people just write all their code directly into the R prompt without saving their steps. More commonly however is to save your R code for later use or just as documentation of your work. It is generally a good idea to organize your code in some text file whether you finally choose to save the file or not. If you save your text file, you should name the file with an `.R` ending by convention. You can, however, save your code in any *plain text* file.

Any *plain text* - text editor such as, Notepad, Wordpad or gedit is sufficient to organize your code. Once you copy your code into the **clipboard**, you can either paste the code directly to the R prompt, or better enter

```
> source("clipboard")
```

because “Since the complete file is parsed before any of it is run, syntax errors result in none of the code being run.” [`?source`].

Finally, however, I suggest to use a good integrated development environment (IDE) due to reasons I already gave in earlier slides.

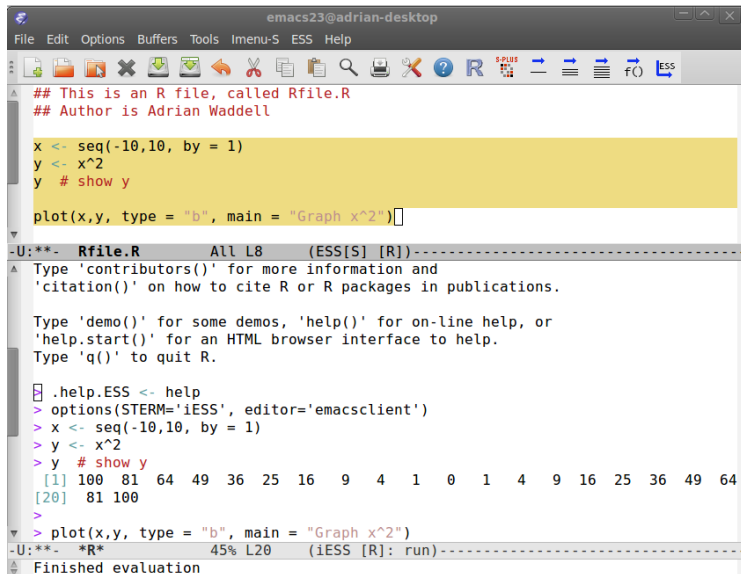
Emacs Speaks Statistic (ESS)

We assume that you read and understood the slides on IDEs in this series.

ESS is an Emacs add-on package that enhances Emacs in many ways to make your R experience better and more productive. See the [ESS project page](#) for more information. We will focus on showing how to perform the most common tasks.

- ▶ In Emacs, open or create some R source file with an `.R` ending. Your *major mode* should be `ESS[S]`.
- ▶ Split the window in two using Emacs's keystroke `C-x 2`.
- ▶ Switch to the lower window: `C-x o`. Start an R process: `M-x R RET`.
 - ▶ the *minibuffer* asks you for the *starting data directory* which is the path to your *working directory*.
- ▶ Write R code in the source *buffer*. Execute a selection with `C-c C-r`.

Emacs Speaks Statistic (ESS)



The screenshot shows the Emacs ESS interface. The top window displays the R source code for 'Rfile.R'. The code defines a sequence of values for x, calculates y as x squared, and plots the results. The bottom window shows the execution output, including help text, the R prompt, the execution of the code, the resulting data vectors, and the completion of the plot command.

```
emacs23@adrian-desktop
File Edit Options Buffers Tools Imenu-S ESS Help
[Icons] R S-PLUS f() ESS

## This is an R file, called Rfile.R
## Author is Adrian Waddell

x <- seq(-10,10, by = 1)
y <- x^2
y # show y

plot(x,y, type = "b", main = "Graph x^2")

-U:**- Rfile.R All L8 (ESS[S] [R])-----
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

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

[ ] .help.ESS <- help
> options(STERM='iESS', editor='emacsclient')
> x <- seq(-10,10, by = 1)
> y <- x^2
> y # show y
[1] 100 81 64 49 36 25 16 9 4 1 0 1 4 9 16 25 36 49 64
[20] 81 100
>
> plot(x,y, type = "b", main = "Graph x^2")
-U:**- *R* 45% L20 (iESS [R]: run)-----
Finished evaluation
```

Emacs Speaks Statistic (ESS)

Notice that the tool bar has special icons for R. Try the following Emacs keystrokes in the source file *buffer*:

C-c C-j send line to R

C-c C-n send line to R and move to next line

C-c C-r send selection to R

C-c M-j send line and return to the ESS process buffer

C-c C-f send the R function containing point to the ESS process

M-? OR **C-c TAB** list all possible completions of the object name at point

C-c C-v show help for some function

- inserts <-

For automatic code indentation use `TAB` for each individual line. For single line comments use `#`, you can indent the comment correctly with `M-;`.

For a header comments use `##`.

For more information read section [7.4](#), [7.5](#) and [7.6](#) of the ESS [manual](#).

ESS: Using a Remote Connection

If you have ssh access to a remote server running R, you can use your local Emacs and ESS installation to interact with R on this remote server.

The official documentation to do this can be found [here](#). However I will summarize the steps.

1. You need to install `ssh.el`, download it from [here](#) and save it in some folder, e.g. (under UNIX) `/usr/share/emacs/site-lisp/ssh`
2. Start Emacs and add the line

```
(add-to-list 'load-path "/usr/share/emacs/site-lisp/ssh")
```

to the `.emacs` file where the path has to be adjusted. See the slides on Customizing Emacs.

3. Restart Emacs. Enter: `M-x load-library RET ssh RET`
4. Then enter `M-x ssh`, a login prompt will show up. (e.g. use `username@cpu119@math.uwaterloo.ca`)
5. Start R. Then enter `M-x ess-remote` and specify some name in the prompt like `r`.
6. Now open your R file. Use ESS as usual.

The StatET Plugin for Eclipse

- ▶ Read the instructions in the [StatET documentation](#).
- ▶ Execute R code (selection or single line) with the key combination `C-r C-r`.
- ▶ Code completion with `C-SPC`
- ▶ Repeatedly pressing `C-SPC` goes through different namespaces.
- ▶ Use the context menu (right click with mouse).
- ▶ Try the window maximization mode.
- ▶ Note the object browser.

Other IDEs

Other working environments and text editors exist which offer special functionality for R. A comprehensive list exists [here](#).

Rkward ([url](#)) is an IDE which runs on Linux systems (KDE native). A windows port exists.

JGR ([url](#)) is another popular R editor.

Rcmdr see `library(Rcmdr)`.

WindEdt ([url](#)) is a popular but commercial text editor for Windows. WinEdt also provides some latex functionality. See [Edt text editors](#).

So which IDE should you choose?



I suggest you should learn how to use Emacs in any case. Most importantly, Emacs really provides a productivity boost. Also Emacs's power will come in handy once you have to run some simulations on a UNIX server or you need to start R over an `ssh` connection.

Packages

Packages are a collection of functions, their documentation and data sets. The **CRAN servers** mirror a large list of packages made publicly available by the community.

You can *load* a package with the `library()` command. Check which packages get loaded automatically with `getOption("defaultPackages")`.

You can install packages from the CRAN mirrors using `install.packages("RnavGraph", dependencies=TRUE)`, if you want to install the `RnavGraph` package.

To get a list of all installed packages use `installed.packages()` or `.packages(all.available=TRUE)`. The currently loaded packages can be listed with `(.packages())`.

The `require()` function, like `library()`, also loads a package. However it is designed to be used within a function and will return `TRUE` or `FALSE` and a warning, rather than an error, if the package is not installed.

```
> require(tcltk) || stop("library 'tcltk' is not installed")
```

Other Package Repositories

Next to CRAN, there are some other R package repositories.

Bioconductor ([url](#)) focuses on genomic analysis.

Omegahat ([url](#)) mainly packages which define APIs to web services and programming languages.

R-Forge ([url](#)) is mainly intended to help developers to collaborate.

Workspace and Working Directory

Two terms you should not confuse

The *workspace* is the collection of R objects in your current R session. You can list all your objects using the `ls()` command and delete them using the `rm()` command.

```
> a <- 1; x <- seq(1, 10, by = 1); fit <- lm(x~I(x^2))
> ls()
[1] "a"    "fit"  "x"
> rm(x)
> ls()
[1] "a"    "fit"
> rm(list = ls())
> ls()
character(0)
```

R usually asks you whether you want to save your current *workspace* or not when you leave R with `q()`. You can save your *workspace* manually for later use with the `save()` or `save.image()` command. Load your saved *workspace* in a new R session with the `load()` command.

If needed, only save a few selected objects. Don't save your whole workspace for continuing working from it in your next R session: this will almost certainly cause problems. Better re-run your code.



Workspace and Working Directory

Two terms you should not confuse

The *working directory* is an **absolute path** to a directory. This directory can be used to access and save files from within R by specifying relative paths to your files of interest.

```
> getwd()
[1] "/home/adrian/Documents/R/"
> setwd("/home/adrian")
```

so if you want to save your current workspace, you could either use

```
> save.image(file = "test/myWorkspace.Rdata")
```

or

```
> save.image(file = "/home/adrian/test/myWorkspace.Rdata")
```

Both cases save the workspace to the same physical location on your hard drive, assuming the folder `test` exists.

Absolute paths in Windows differ from those of the UNIX, Linux and OSX world. You will have to use something like

```
> setwd("C:/Users/adrian/")
```

The Search Path

The search path is where and the order of how R “looks” for variables and functions when you type their names at the prompt. You can display your search path with

```
> search()
```

Notice that most of the entries are packages. Every time you use the `library()` command, your *search path* gets expanded by one element.

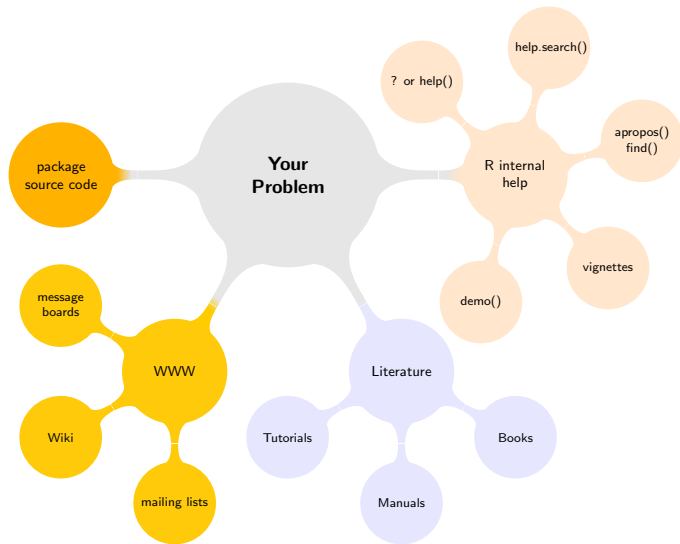
<code>.GlobalEnv</code>	<code>package:rggobi</code>	<code>...</code>	<code>package:graph</code>
<code>a</code>	<code>...</code>		<code>...</code>
<code>b</code>	<code>edges()</code>		<code>edges()</code>
<code>foo()</code>	<code>...</code>		<code>...</code>

For the above situation, the `edges()` function defined in the `graph` package gets *masked* by the `edges()` function defined in the `rggobi` package. You can still access the `edges()` function in the `graph` package by using

```
> graph::edges(G)
```

You can remove an entry from the *search path* using `detach()`.

Help



Help

The R internal help system

- ▶ All packages in R come with documentation.
- ▶ You can access the documentation for `plot()` with `help(plot)`, `?plot` or in Emacs with `C-c C-v plot RET`.
- ▶ If you need the documentation on a Operator, use `help('&&')` or `? '&&'`.
- ▶ Choose between the output format:
 - `plain text` `options(help_type = 'text')`
 - `html` `options(help_type = 'html')`, for some systems you might have to add the argument `browser='firefox'`.
 - `pdf` `options(help_type = 'pdf')`
- ▶ You get to the index of all documented R functions from a certain package with `library(help="MASS")` or by using `help.start()` and then selecting packages and your package name.

More on the Internal R Help

- ▶ Explore `help.start()`, read its documentation with `?help.start`
- ▶ If don't know exactly what you are looking for, try `help.search()` which searches for a particular pattern using fuzzy matching or regular expressions. Its short notation is `??`. For example enter

```
> help.search("quantile")
```

or

```
> ??"quantile"
```

if you are looking for a *cumulative quantile plot* function.

R Help on the WWW

- ▶ R has a very nice built in web search function, e.g. if you are looking for R and multithreading, enter into R

```
> RSiteSearch("multithreading")
```

- ▶ R has a very active user community which communicates mainly through the **R mailing lists**. It's very likely that someone else posted a problem which is similar or equivalent to yours.
- ▶ Another way than `RSiteSearch` to search through the mailing list archives is to use **google**. Enter `site:stat.ethz.ch` and then the topic you are looking for:

Running `demo()`

Sometimes, the package developer writes demo files for his/her package. They usually consist of a working example using the specific package.

- ▶ To get a list of all *demos* accessible from the current search path use

```
> demo()
```

- ▶ To get a list of *demos* for a specific package, e.g. the `tcltk` package, use

```
> demo(package = 'tcltk')
```

- ▶ To run a demo use

```
> demo(tkcanvas)
```

- ▶ To find the file path of the demo source code, use

```
> system.file("demo", "tkcanvas.R", package="tcltk")
```