An Introduction to R

Phil Spector
Statistical Computing Facility
University of California, Berkeley

September 24, 2004

1 Background

The R language is a project designed to create a free, open source language which can be used as a replacement for the Splus language, originally developed as the S language at AT&T Bell Labs, and currently marketed by Insightful Corporation of Seattle, Washington. While R is not 100% compatible with Splus (it is often described as a language "which bears a passing resemblance to S"), many Splus programs will run under R with no alterations. Accordingly, much of the existing documentation for Splus can still be useful under R, and many authors of code for either language are careful to make sure that their code will be suitable for both languages.

2 Strengths and Weaknesses

2.1 Strengths

- free and open source, supported by a strong user community
- highly extensible and flexible
- implementation of modern statistical methods
- moderately flexible graphics with intelligent defaults

2.2 Weaknesses

- slow or impossible with large data sets
- non-standard programming paradigms

3 Basics

R is a highly functional language; virtually everything in R is done through functions. Arguments to functions can be named; these names should correspond to the names given in the help file or the function's definition. You can abbreviate the names of arguments if there are no other named arguments to the function which begin with the abbreviation you've used. If you don't provide a name for the arguments to functions, R will assume a one-to-one correspondence between the arguments in the function's definition and the arguments which you passed to the function.

To store the output of a function into an object, use either an equal sign (=) or the special assignment operator `<-`. For example, to save the value of the mean of a vector `x` in a scalar called `mx`, use

```
mx = mean(x)
```

In earlier versions of R, the `<-` operator could be abbreviated as an underscore (_), but in more recent versions of the language, the underscore symbol has no such special meaning, and the underscore symbol can be used as part of a variable name.

If you forget to save the output of a function, the answer will be temporarily available in an object with the name `.Last.value`.

Typing the name of any R object, including a function, will display a representation of that object. You can explicitly display an object with the functions `print` or `cat`. R provides online help through the `help` function, or by simply preceding a function's name with a question mark (`?`). If you are running R in an environment that includes a web browser, the statement

```
help.start()
```

will provide a web-based interface to the help system. In addition to plain text and HTML versions of the documentation, R also provides hardcopy documentation through LaTeX help files, available by using the `offline=T` argument to the `help` function. If you'd rather learn about a command by studying some examples, the `example` function will run one or more examples using the function passed as its argument. In addition, the `demo()` function will provide an easy-to-use interface to some R demonstration scripts. You should quickly get in the habit of consulting the help or example files while you use R, because many of the functions have a number of useful optional arguments which might not be apparent at first glance.

R supports a number of different data structures, accomodating virtually any type of data. At the simplest level, R supports vectors and matrices. These, however are just one- and two-dimensional examples of the more general concept of an array; R supports arrays of unlimited dimensions. In addition, R implements the more traditional "observations and variables" format of rectangular data sets through objects known as data frames, where character and numeric variables can be freely mixed. Finally, R has a very general data structure known as a list, which can hold any structure of data imaginable.

Besides numbers and character strings, the symbols `TRUE` and `FALSE`) are reserved in R to represent the logical values true and false, respectively.

The objects you create during an R session are temporarily stored in memory; each time you end an R session, you must decide whether you want to save the objects you've been using. (When you run R non-interactively, you must specify either the `--save` or `--no-save` option at invocation time.) If you choose to save the objects you've created, they are stored in a file called `.RData` in the current working directory. (Note that since this file's name begins with a period, you need to provide the UNIX `ls` command the `-a` option in order to display the file's name in a directory listing.) If you want to save your workspace at some other point in an R session, you can call the `save.image()` function with no arguments.

When you start up an R session, the program always looks in the current working directory to see if an `.RData` file exists (unless the `--no-restore` option is specified), and, if you decide to save your working environment at the end of an R session, an `.RData` file will be created. Thus, to keep data from different projects in separate places, it suffices to simply set a different working directory (with the UNIX `cd` command) for each of your projects.

In addition to the `.RData` file in your working directory, R searches for data and functions in a number of system directories. You can see the names of these directories by using the `search` function. Note that just typing the name `search` will display a print representation of the `search` function; to invoke the function with no arguments you must type

```
search()
```

You can see the contents of any directories in your search path through the `objects` function. With no arguments, it will list all the objects in your working directory. The optional argument `pos` allows you to specify other directories in your search path by providing the index of the directory as provided by the `search` command. The optional argument `pattern` will restrict the listing of objects to those whose names contain the string or regular expression specified by the `pattern` argument. Quoted strings in R can be surrounded by either single (`'`) or double (`"`) quotes. Thus, the command

```
objects(pat="dat")
```

will list those objects in your working directory whose names contain the string `dat`. (Notice that, when using named arguments to a function, you can abbreviate the name of the argument, provided that only one named argument agrees with the abbreviation.)

To use R objects which are stored in an `.RData` file other than the one in the current directory, pass a character string containing the full path name of the file to the `attach` command. In addition, a wide variety of functions are organized into sections known as libraries. To see which libraries are available on a given installation of R, type the command `library()`; once you've found a library of interest, you can make the functions and documentation in the library available with a command of the form `library(libraryname)`. The command `library(help=libraryname)` will list the functions which are available, and the `help` command can then be used in the usual way to get more information about individual functions in the library.

You can enter UNIX commands in an interactive or batch R program by passing a character string containing the command to the the `system` function.

You can interactively debug R functions by passing the unquoted name of the function you wish to debug to the `debug` function, and then running program which calls that function. See the help file for the `debug` function for details on how to use the debugger.

4 Specific Tasks

4.1 Entering and Exiting the Program

To start an interactive session with R, type

```
R
```

at the UNIX prompt. On most implementations or R, you can redisplay and edit previous commands using arrow keys, or `emacs`-style keystrokes. Inside of R, the interactive prompt is a single greater-than sign (`>`).

In addition, if you are familiar with the `emacs` editor, there is a special mode for running R which provides command recall, variable name completion, simplified access to help files and many other features. If it is installed, you can access it through typing `M-x R`, followed by the Return key.

R obeys standard UNIX redirection, so you can execute source files in the usual way, with a command like this one at the UNIX prompt:

```
R < infile >& outfile
```

The R commands to be executed would be in the file `infile`, and output would be sent to `outfile`. The `BATCH` command of R packages this capability in a slightly different fashion. Typing

```
R BATCH infile outfile
```

at the UNIX prompt would have a similar effect as the previous command.

To execute R statements from a file from within an interactive session, you can use the `source` command. Type

```
source("infile")
```

at the R prompt to execute the commands in the file `infile`.

To exit an interactive R session, type `q()`. Note that just typing the `q` without the parentheses will simply display a text representation of the `q` function.

End of ebook preview
Download the full PDF tutorial from the link below :

Click Here