Chapter 6

Introduction to S

6.1 History and Terminology

Most major statistical packages are computer programs that have their own control language. The syntax goes with one computer program and just one. The SAS language controls the SAS software, and that's it. Minitab syntax controls Minitab, and that's it. S is a little different. Originally, S was both a program and a language; they were developed together at the former AT&T Bell Labs starting in the late 1970's. Like the unix operating system (also developed around the same time at Bell Labs, among other places), S was open-source and in the public domain. "Open-source" means that the actual program code (initially in Fortran, later in C) was public. It was free to anyone with the expertise to compile and install it.

Later, S was spun off into a private company that is now called Insightful Corporation. They incorporated both the S syntax and the core of the S software into a commercial product called S-Plus. S-Plus is *not* open-source. The "Plus" part of S-Plus is definitely proprietary. S-Plus uses the S language, but the S language is not owned by Insightful Corporation. It's in the public domain.

R also uses the S language. This is a unix joke. You know, like how the unix `less` command is an improved version of `more`. Get it? R is produced by a team of volunteer programmers and statisticians, under the auspices of the Free Software Foundation. It is an official GNU project. What is GNU? GNU stands for "GNU's Not Unix." The recursive nature of this answer is a unix joke. Get it?

The GNU project was started by a group of programmers (led by the great Richard Stallman, author of `emacs`) who believed that software should be open-source and free for anyone to use, copy or modify. They were irritated by the fact that corporations could take unix, enhance it in a few minor (or major) ways, and copyright the result. Solaris, the version of unix used on many Sun workstations, is an example. An even more extreme example is Macintosh OS X, which is just a very elaborate graphical shell running on top of Berkeley Standard Distribution unix.

The GNU operating system was to look and act like unix, but to be re-written from the ground up, and legally protected in such a way that it could not be incorporated into any piece of software that was proprietary. Anybody would be able to modify it and even sell the modified version – or the original. But any modified version, like the original, would have to be open-source, with no restrictions on copying or use of the software. The main GNU project has been successful; the result is called linux.

R is another successful GNU project. The R development team re-wrote the S software from scratch without using any of the original code. It runs under the unix, linux, MS Windows and Macintosh operating systems. It is free, and easy to install. Go to `http://www.R-project.org` to obtain a copy of the software or more information. There are also links on the course home page.

While they were re-doing S, the R development team quietly fixed an extremely serious problem. While the S *language* provides a beautiful environment for simulation and customized computer-intensive statistical methods, the S *software* did the job in a terribly inefficient way. The result was that big simulations ran very slowly, and long-running jobs often aborted or crashed the system unless special and very unpleasant measures were taken. S-Plus, because it is based on the original S code, inherits these problems. R is immune to them.

Anyway, S is a language, and R is a piece of software that is controlled by the S language. The discussion that follows will usually refer to S, but all the examples will use the R implementation of S — specifically, R version 1.4.0 running under unix on credit or tuzo (credit and tuzo are supposed to be 100% identical). Mostly, what we do here will also work in S-Plus. Why would you ever want to use S-Plus? Well, it does have some capabilities that R does not have (yet), particularly in the areas of survival analysis and spatial statistics.

6.2 S as a Calculator

To start R, type “R” and return at the unix prompt. Like this:

```
/res/jbrunner/442/S > R

R : Copyright 2001, The R Development Core Team
Version 1.4.0  (2001-12-19)

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.

>
```

S is built around *functions*. As you can see above, even asking for help and quitting are functions (with no arguments).

The primary mode of operation of S is line oriented and interactive. It is quite unix-like, with all the good and evil that implies. S gives you a prompt that looks like a ”greater than” sign. You type a command, press Return (Enter), and the program does what you say. Its default behaviour is to return the value of what you type, often a numerical value. In the first example, we receive the “>” prompt, type “1+1” and then press the Enter key. S tells us that the answer is 2. Then we obtain $2^3 = 8$.

```
> 1+1
[1] 2
> 2^3 # Two to the power 3
[1] 8
```

What is this `[1]` business? It’s clarified when we ask for the numbers from 1 to 30.

```
> 1:30
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
[26] 26 27 28 29 30
```

S will give you an array of numbers in compact form, using a number in brackets to indicate the ordinal position of the first item on each line. When it answered "1+1" with [1] 2, it was telling us that the first item in this array (of one item) was 2.

S has an amazing variety of mathematical and statistical functions. For example, the gamma function is defined by $\Gamma(a) = \int_0^\infty e^{-t} t^{a-1}\, dt$, and with enough effort you can prove that $\Gamma(\frac{1}{2}) = \sqrt{\pi}$. Note that everything to the left of a # is a comment.

```
> gamma(.5)^2      # Gamma(1/2) = Sqrt(Pi)
[1] 3.141593
```

Assignment of values is carried out by a "less than" sign followed *immediately* by a minus sign; it looks like an arrow pointing to the left. The command `x <- 1` would be read "x gets 1."

```
> x <- 1           # Assigns the value 1 to x
> y <- 2
> x+y
[1] 3
> z <- x+y
> z
[1] 3
> x <- c(1,2,3,4,5,6)    #  Collect these numbers; x is now a vector
```

Originally, x was a single number. Now it's a vector (array) of 6 numbers. S operates naturally on vectors.

```
> y <- 1 + 2*x
> cbind(x,y)
     x  y
[1,] 1  3
[2,] 2  5
[3,] 3  7
[4,] 4  9
[5,] 5 11
[6,] 6 13
```

The `cbind` command `bind`s the vectors x and y into `c`olumns. The result is a matrix whose value is returned (displayed on the screen), since it is not assigned to anything.

The bracket (subscript) notation for selecting elements of an array is very powerful. The following is just a simple example.

```
> z <- y[x>4]           # z gets y such that x > 4
> z
[1] 11 13
```

If you put an array of integers inside the brackets, you get those elements, in the order indicated.

```
> y[c(6,5,4,3,2,1)] # y in opposite order
[1] 13 11  9  7  5  3
> y[c(2,2,2,3,4)] # Repeats are okay
[1] 5 5 5 7 9
> y[7] # There is no seventh element.  NA is the missing value code
[1] NA
```

Most operations on arrays are performed element by element. If you take a function of an array, S applies the function to each element of the array and returns an array of function values.

```
> z <- x/y       # Most operations are performed element by element
> cbind(x,y,z)
     x  y         z
[1,] 1  3 0.3333333
[2,] 2  5 0.4000000
[3,] 3  7 0.4285714
[4,] 4  9 0.4444444
[5,] 5 11 0.4545455
[6,] 6 13 0.4615385
> x <- seq(from=0,to=3,by=.1)   # A sequence of numbers
> y <- sqrt(x)
```

S is a great environment for producing high-quality graphics, though we won't use it much for that. Here's just one example. We activate the `pdf` graphics device, so that all subsequent graphics in the session are written to a file that can be viewed with Adobe's *Acrobat Reader*. We then make a line plot of the function $y = \sqrt{x}$, and quit.

```
> pdf("testor.pdf")
> plot(x,y,type='l')       # That's a lower case L
> q()
```

Actually, graphics are a good reason to download and install R on your desktop or laptop computer. By default, you'll see nice graphics output on your screen. Under unix, it's a bit of a pain unless you're in an X-window environment (and we're assuming that you are not). You have to transfer that `pdf` file somewhere and view it with *Acrobat* or *Acrobat Reader*.

Continuing the session, a couple of interesting things happen when we quit. First, we are asked if we want to save the "workspace image." The responses are Yes, No and Cancel (don't quit yet). If you say Yes, R will write a file containing all the objects (x, y and z in the present case) that have been created in the session. Next time you start R, your work will be "restored" to where it was when you quit.

```
Save workspace image? [y/n/c]: y
credit.erin > ls
testor.pdf
```

Notice that when we type `ls`, to list the files, we see only `testor.pdf`, the pdf file containing the plot of $y = \sqrt{x}$. Where is the workspace image? It's an invisible file; type `ls -a` to see *all* the files.

```
credit.erin > ls -a
./                ../                .RData          testor.pdf
```

There it is: `.RData`. Files beginning with a period don't show up in output to the `ls` command unless you use the `-a` option. R puts `.RData` *in the (sub)directory from which R was invoked*. This means that if if you have a separate subdirectory for each project or assignment (not a bad way to organize your work), R will save the workspace from each job in a separate place, so that you can have variables with names like x in more than one place, containing different numbers. When we return to R,

```
credit.erin > R

R : Copyright 2001, The R Development Core Team
Version 1.4.0  (2001-12-19)
```

```
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.

[Previously saved workspace restored]

> ls()
[1] "x" "y" "z"

> max(x)
[1] 3
```

All the examples so far (and many of the examples to follow) are interactive, but for serious work, it's better to work with a command file. Put your commands in a file and execute them all at once. Suppose your commands are in a file called `commands.R`. At the S prompt, you'd execute them with `source("commands.R")`. From the unix prompt, you'd do it like this. The `--vanilla` option invokes a "plain vanilla" mode of operation suitable for this situation.

```
credit.erin > R --vanilla < commands.R > homework.out
```

For really big simulations, you may want to run the job in the background at a lower priority. The `&` suffix means run it in the background. `nohup` means don't hang up on me when I log out. `nice` means be nice to other users, and run it at a lower priority.

```
credit.erin > nohup nice R --vanilla < bvnorm.R > bvnorm.out &
```