# An Introduction to S-PLUS

Aidan Palmer

January 29, 1997

## 1 Introduction

This document is intended as a basic introduction to S-PLUS. It covers starting and quitting S-PLUS, simple function calls, assignment statements, vectors, and an introdution to some simple functions.

## 2 Starting and Quitting S-PLUS

To start S-PLUS from a UNIX machine, type `Splus` at an xterm or hpterm prompt.

```
% Splus
```

After a short wait, you will see something like this:

```
S-PLUS : Copyright (c) 1988, 1995 MathSoft, Inc.
S : Copyright AT&T.
Version 3.3 Release 1 for HP Series 700, HP-UX 9.x : 1995
Working data will be in .Data.hp
>
```

S-PLUS will create a directory called `.Data.hp` or `.Data.sun4` (or something like that depending on what type of machine you are using; `_Data` on Windows machines) to store any functions or variables you define. This automatically saves your work from session to session.

The `>` is the prompt for entering commands. When you are done, type `q()` at this prompt and press return. That will quit S-PLUS.

## 3 Entering Commands

To have S-PLUS execute a command, enter the command at the prompt and press return. For instance, type `2+3` and then press return.

```
> 2 + 3
[1] 5
```

Arithmetic and order of operations work exactly in the ordinary way. Multiplication is `*`, division is `/` and exponentiation is `^`.

The format for calling a function is `functionname(`*arg1*, *arg2*, ...`)`, or `functionname()` for a function which takes zero arguments. Typing the function name by itself (with no parentheses) will return the definition of the function. S-PLUS is case-sensitive, so be sure to use the correct capitalization: `Functionname()` and `functionname()` are different things.

The square root and absolute value functions need just one argument each:

```
> sqrt(9)
[1] 3

> abs(-5)
[1] 5
```

The “[1]” which keeps showing up means that the first item on that line is the first item in the vector of output. This indexing is useful when you want to find a particular observation in a long list of output. In the output below, 1000 is the 36th observation.

```
 [1]    0.73    1.46    2.19    2.92    3.65    4.38    5.11    5.84    6.57
[10]    7.30    8.03    8.76    9.49   10.22   10.95   11.68   12.41   13.14
[19]   13.87   14.60   15.33   16.06   16.79   17.52   18.25   18.98   19.71
[28]   20.44   21.17   21.90   22.63   23.36   24.09   24.82   25.55 1000.00
[37]    1.04    1.30    1.56    1.82    2.08    2.34    2.60    2.86    3.12
[46]    3.38    3.64    3.90    4.16    4.42    4.68    4.94
```

If you enter an incomplete command (for instance, by forgetting a close-parenthesis), S-PLUS will give you another prompt, this time a “+”.

```
> sqrt(16
+
```

At this point, type the close-parenthesis and the command will execute normally.

```
> sqrt(16
+ )
[1] 4
```

# 4 Running S-PLUS in Batch Mode

Rather than typing in one line at a time, you may wish to prepare a list of commands and execute them all at once. This approach is preferable when you have a series of rote instructions, for instance for a simulation. It is not useful for exploratory analyses or anything else requiring interaction.

To run S-PLUS in batch mode, put all your commands in a file (note: end the file with several blank lines, or S-PLUS may not execute the final command correctly). Instead of plain `Splus`, run:

```
Splus BATCH yourfile outputfile
```

This takes the commands in `yourfile` and runs them through S-PLUS, storing the results in `outputfile`. For large tasks, you will want to do this in the background, by putting an ampersand `&` at the end of the call:

```
Splus BATCH yourfile outputfile &
```

# 5 Cutting and Pasting to a Report

The way S-PLUS handles cutting and pasting depends on where it is being run. If you are running S-PLUS in an hpterm or xterm window, you can copy with the left mouse button and paste with the center button. If you have emacs running, you can copy S-PLUS output to the emacs window in this fashion, then write the report in emacs. You could also do this in the other direction, by typing commands in the emacs window and then pasting them at the S-PLUS prompt.

# 6 Getting Help

The S-PLUS help documentation provides specifications of functions, rather than a tutorial. You may find this frustrating at first, but it comes in handy when you are unsure as to what arguments a function needs, or how it works. For example, to see the documentation on the absolute value function, type `help(abs)`. If you know what you want to do, but not what that function is called in S-PLUS, type `help.start()` to open the interactive help window. This provides an interface which allows searches for all functions relevant to a topic.

## 7 A Few Useful Functions

S-PLUS stores every command you type in the file .Audit in your .Data directory. You can examine that file to review commands you have used to solve previous problems. You can also take advantage of this by typing `again()` to repeat the previous command. To repeat an old command, use the function `history()`. This function provides a list of the last ten commands, but by typing `history(max=100)` you can see the last 100 commands. On a UNIX machine, there is a better way to do this: run `Splus -e` instead of plain `Splus` (the `-e` means "in emacs mode"). In emacs mode, S-PLUS allows you to retrieve previous commands by pressing control-P. Because of this handy feature, you may wish to alias `Splus` to `Splus -e` in your .cshrc file. However, if you want to run Splus in batch mode after doing this, be sure to alias `Splus` back to normal, as `Splus -e BATCH in out` will not produce any output (emacs mode and batch mode do not mix).

## 8 Variables and Assignments

The command for assigning a value to a variable is `<-`. You can basically use any name for a variable, although it will cause problems if you give a variable the same name as a function, so be careful not to use `c` or `length` as a variable name. Although innocuous, these are both functions in S-PLUS. If you do accidentally give a variable the same name as a function, remove the variable from your .Data area (see below).

Variable names can be as long as you need, but should not contain blank spaces. If you have two words in a variable name, such as "Water Depth", you could write it as `WaterDepth`, `waterdepth`, or `water.depth`. You may not separate words with an underscore (that is a special character in S-PLUS) or a dash (that would be interpreted as a minus sign).

To assign the value `9` to the variable `x`, type:

```
> x <- 9
```

(Read this as "x gets 9".) Note that S-PLUS does not return any output after an assignment. If you type the variable name, S-PLUS will return its value (if you are ever unsure, this is a good way to confirm that an assignment was successful).

Now that `x` has a value, it can be manipulated like any other number.

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

> y <- (5 * (x + 2)) - 3
```

These manipulations do not affect the value of `x`, it is still `9`. To change it, you must assign a new value to `x`.

```
> sqrt(x)
[1] 3
> x
[1] 9
> x <- sqrt(x)
> x
[1] 3
```

Text values can also be assigned to a variable:

```
> y <- "hello"
> y
[1] "hello"
```

The variables `x` and `y` are now stored in your .Data directory. This means you will be able to use them for the remainder of this session and any time you run S-PLUS in the future. For a list of objects in your directory, use the `objects()` command. Unneeded objects may be removed with the `remove` or `rm` commands. The two work the same, only `remove` requires quotation marks around the argument, for instance `remove("x")` is the same as `rm(x)`.

> **Question:** How do you exchange the values of x and y?

# 9 Vectors

All of the preceding examples were concerned with single numbers, but most statistical analyses involve groups of numbers. In S-PLUS, groups of numbers can be manipulated most easily as vectors.

Suppose ten replications of an experiment had the following results:

```
2, 4.6, 1, 3.7, 5.9, 4.0, 6.7, 2.8, 1.4, 3.1
```

You can enter this data into S-PLUS as a vector, and store that vector as a variable (for instance, a variable called "observations").

```
> observations <- c(2, 4.6, 1, 3.7, 5.9, 4.0, 6.7, 2.8, 1.4, 3.1)
```

The function `c()` takes a list and returns it a vector (think of `c` as "column"). Check the value of the variable `observations` by entering it at the prompt.

```
> observations
[1] 2.0 4.6 1.0 3.7 5.9 4.0 6.7 2.8 1.4 3.1
```

Manipulations on vectors work the same way as manipulations on single numbers. Suppose these observations were in inches but need to be converted to centimeters:

```
> 2.54*observations
[1]  5.080 11.684  2.540  9.398 14.986 10.160 17.018  7.112  3.556  7.874
```

Note that the vector `observations` still contains the original data.

```
> mean(observations)
[1] 3.52
> var(observations)
[1] 3.450667
```

The above functions return the mean and variance, respectively. Note that these functions will not work correctly unless they are given vectors as arguments. This first item below is an attempt to give `mean` three arguments, but it only expects one. The second item is the correct way: give `mean` a single argument, which happens in this case to be a vector with three elements.

```
> mean(1,3,5)
[1] 1
> mean(c(1,3,5))
[1] 3
```

> **Question:** How would you normalize the vector `observations` to have mean 0 and variance 1?

## 10 Selecting from Vectors

Subsets of a vector can be selected by using square brackets.

```
> observations[3]
[1] 1
> observations[5:7]
[1] 5.9 4.0 6.7
> observations[c(1,2,7,1)]
[1] 2.0 4.6 6.7 2.0
```

The above functions return the third observation, the fifth through seventh observations, and the first, second, seventh and first (again) observations, respectively.

To select everything *except* some particular element or set of elements, use a minus sign. The following command would return the entire vector except the first element.

```
> observations[-1]
[1] 4.6 1.0 3.7 5.9 4.0 6.7 2.8 1.4 3.1
```

Vectors can also be searched using logic statements. To pick out all observations which have a value larger than four, type:

```
> observations[observations>4]
[1] 4.6 5.9 6.7
```

## 11 An Example in S-PLUS

In the previous example we had to type the data in by hand. S-PLUS can read data from a file, as will be described in a future handout. For now, consider a data set that comes pre-loaded into S-PLUS.

These data concerns cars purchased in 1974. There are three vectors of data: `car.time`, the number of days since the car was purchased; `car.miles`, the number of miles driven betwen this fill-up and the previous fill-up; and `car.gals`, the number of gallons required to fill the tank. These vectors are already loaded into S-PLUS, so you should be able to access them by typing their names.

```
> car.time
  [1]   1.0   7.0  17.0  28.0  42.0  56.0  65.0  74.0  76.0  88.0  92.0 100.0
 [13] 107.0 119.0 126.0 135.0 139.0 146.0 151.0 156.0 161.0 170.0 177.0 192.0
 [25] 200.0 208.0 215.0 219.0 226.0 234.0 239.0 245.0 251.0 257.0 262.0 270.0
 [37] 277.0 282.0 288.0 289.0 292.0 292.5 294.0 295.0 300.0 305.0 310.0 314.0
 [49] 321.0 323.0 330.0 338.0 347.0 355.0 360.0 364.0 368.0 370.0 370.5 371.0
 [61] 377.0 382.0 389.0 395.0 405.0 409.0 413.0 416.0 421.0 423.0 427.0 433.0
 [73] 438.0 447.0 453.0 461.0 466.0 475.0 482.0 487.0 491.0 501.0 516.0 524.0
 [85] 534.0 543.0 548.0 558.0 565.0 572.0 576.0 583.0 595.0 603.0 607.0 607.5
 [97] 609.0 610.0 610.5 614.0 622.0 635.0 645.0 652.0 660.0 667.0 686.0 702.0
[109] 717.0 728.0 737.0 747.0 753.0 764.0 771.0 777.0 783.0 786.0
> car.miles
  [1] 210.0 199.0 182.0 208.4 217.0 379.8 204.0 180.6 157.5 226.8 181.0 194.5
 [13] 194.5 208.3 188.6 193.0 212.0 207.5 216.2 203.1 202.5 193.5 116.0 221.6
 [25] 194.2 213.6 217.3 211.5 200.0 191.0 227.0 207.4 209.8 212.5 209.0 213.1
 [37] 215.4 215.8 204.1 165.0 176.3 243.0 142.0 293.0 118.0 224.0 221.9 199.0
 [49] 242.0 222.0 219.0 202.6 221.0 226.0 207.0 218.0 228.0  97.0 266.0 253.0
 [61] 191.0 228.0 212.7 217.0 190.0 213.0 210.6 217.0 176.7 206.0 208.3 190.0
 [73] 212.0 313.0 197.0 189.0 188.0 334.0 203.0 209.0 236.0 206.0 163.0 195.0
```

```
 [85] 201.0 213.0 209.5 197.4 199.6 167.0 208.0 204.0 188.5 212.3  88.7 200.0
 [97] 179.0 153.0 207.0 234.0 193.0 210.0 196.0 217.0 228.0 201.0 216.0 189.4
[109] 184.7 189.0 197.0 168.0 209.0 202.0 183.0 204.0 197.0 204.0
> car.gals
  [1] 13.3 12.2 11.5 13.5 14.3 25.7 13.3 12.7  8.9 14.2 12.0 13.2 12.8 13.6 12.3
 [16] 13.2 13.1 13.6 13.6 13.0 12.5 11.6  7.5 14.2 11.9 13.1 13.6 14.2 12.1 12.8
 [31] 13.9 13.0 12.5 13.6 12.8 14.2 13.0 13.8 12.3  8.9 10.3 11.6  7.8 14.5  7.1
 [46] 12.0 12.3 12.5 14.1 13.0 12.3 13.2 13.7 13.4 12.5 12.9 13.3  6.2 13.2 12.8
 [61] 11.2 13.5 13.2 14.1 13.6 13.0 12.8 13.2 12.9 13.0 13.0 12.0 14.5 21.2 13.2
 [76] 12.9 12.5 22.0 13.1 12.3 13.2 13.7 12.5 13.2 13.5 13.9 12.5 12.4 12.3 10.5
 [91] 11.5 12.8 12.0 13.0  5.8  9.5  9.0  7.7 10.1 11.9 11.4 12.6 12.5 12.7 13.6
[106] 11.7 13.5 12.3 13.0 13.2 13.2 11.6 13.8 13.5 13.9 14.2 13.0 13.0
```

Suppose we are interested in how many miles per gallon these cars got. Let's create a "miles per gallon" vector by dividing the `car.miles` vector by the `car.gals` vector. S-PLUS will divide each element of `car.miles` by the corresponding element of `car.gals`, which is exactly what we want.

```
> car.mpg <- car.miles/car.gals
```

Now we can analyze the miles per gallon data. Some summary statistics might be a good place to start:

```
> summary(car.mpg)
  Min. 1st Qu. Median  Mean 3rd Qu.  Max.
 13.04   15.19  15.93 16.21   16.78 21.05
> var(car.mpg)
[1] 2.438916
```

A stem-and-leaf plot might also prove interesting:

```
> stem(car.mpg)

N = 118   Median = 15.92875
Quartiles = 15.18182, 16.784

Decimal point is at the colon

   13 : 02
   13 : 7
   14 : 02234
   14 : 5667788899999
   15 : 00001122223333333444
   15 : 5566666777888899999
   16 : 000001122233333344
   16 : 556666778889999
   17 : 01111122
   17 : 789
   18 : 012
   18 : 57
   19 :
   19 : 7899

High: 20.15152 20.20690 20.49505 20.94828 21.05263
```

This shows that most of these cars got between 15 and 17 miles per gallon, although there are a few cars which got 19.5 MPG or more. This might be related to how long the car has been owned. We can summarize the `car.time` vector and then select the cars with the good MPG numbers to see how long they have been owned and compare them to the rest.

```
> summary(car.time)
 Min. 1st Qu. Median  Mean 3rd Qu. Max.
    1   235.2  370.8 388.2   563.2  786
> car.time[car.mpg > 19.5]
[1] 292.5 295.0 370.5 371.0 607.5 609.0 610.0 610.5 614.0
```

There does not appear to be any pattern to this: some of the cars are relatively old, some are rather new.

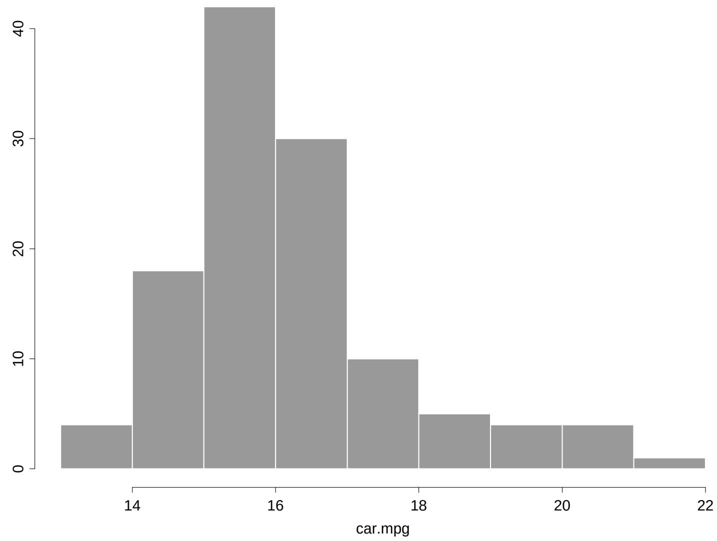
> **Question:** How would you do a stem-and-leaf plot of miles per gallon for all cars owned less than 600 days?

## 12 Graphics in S-PLUS

Perhaps a graphical approach to this data would be more enlightening. The first step is to open a window for graphics, by typing `motif()` or `X11()` if you are using a UNIX workstation or `win.graph()` if you are running S-PLUS on Windows. Either function should cause a window titled “S-PLUS” to appear. Plot commands entered at the regular S-PLUS prompt will cause pictures to appear in the S-PLUS window.

Let’s start with a histogram of the miles per gallon data (after opening the graphics window).

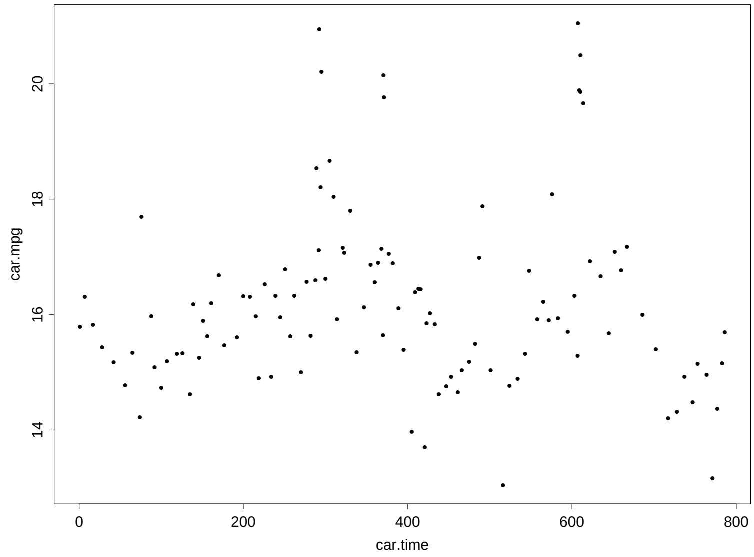
```
> motif()
> hist(car.mpg)
```




If you try this, your plot will probably be in a different color scheme (probably white boxes on a black background). The color scheme is controlled by a menu in the graphics window, and you may want to change it to “Black on White” or whatever you find easiest to read.

To find out the relationship between how long a car has been owned and how many miles per gallon it gets, plot `car.time` on the x-axis and `car.mpg` on the y-axis.

```
> plot(car.time, car.mpg)
```




There doesn't appear to be any obvious relationship between how long a car has been owned and its mileage per gallon.

> **Question:** Is there a relationship between `car.time` and `car.miles`, or between `car.time` and `car.gals`? Do some plots and find out.

If you give `plot` just one argument then it will plot that vector on the y-axis against an index (this is useful when you want to plot residuals in the order they were observed, to make sure that the variance was constant).

When you are done, use the function `q()` to quit S-PLUS. You do not need to close the graphics window, it will close on its own.