Introduction to S-plus

36-401 Howard Seltman 8/31/00

Getting started

First read the "Unix Summary" handout and perform the one-time tasks described there.

A typical homework session includes the following steps:

- Log on to a Solaris computer.
- Type "cd S401" to get to your class directory.
- Download any files you need as follows:
  - Start the Netscape web browser program by typing "netscape&". Enter the URL "http://www.stat.cmu.edu/~hseltman/401". Click on Bookmarks / Add Bookmark so that you can return here easily.
  - Download each file you need by using right click on the file name in the "Download" section of the web page, and choosing "Save Link As..." from the menu, then click "OK". For now download "concord1.dat", "ps.q", and "symplot.q". You may want to keep Netscape running, since S-plus uses it for its help system.
- Start a text document ("command file") to store your S-plus commands and results, e.g. "xemacs 0829.q&". By convention S-plus command files are named ".q".
- Start S-plus by typing "Splus -e". (If you forget the -e, you will not be able to edit any lines you are typing.) The ">" is the prompt for S-plus. Type "q()" if you want to quit S-plus. You cannot use the "&" when running S-plus.
- Use the left mouse drag to copy text from the S-plus or xemacs window and then use middle click to drop it in the other window. Save and retain this file as a record of your S-plus session.

For any homework assignment or project, you should use the work method described above. (See me for alternatives if you are highly computer sophisticated.)

The command file serves several purposes. First, it is your documentation of how you did something in S-plus and will be useful when you want to do something similar in the future. Second, it will allow you to make some small changes and redo your analysis quickly. (If you put the "#" comment character before all output, you will be able to re-run all commands by typing 'source("myfile.q")' in S-plus.) Believe me, *all* real projects have at least one point where you find a mistake in the data or analysis and need to re-run most or all of your commands! Third, you can cut-and-paste results from this file into your written report.

Some Basics of S-plus

• Everything that you type at the S-plus prompt is an expression that is evaluated. Under normal circumstances, the result of evaluation of the expression is displayed on your screen.

• Type in some mathematical expressions and observe the Splus result. You can use the standard math operators such as +, -, *, / and ∧ (to the power of). You can freely use parentheses, otherwise standard order of operations applies. Spaces may be used or omitted anywhere in an expression. You can also use standard functions such as log(), exp(), sqrt(), cos(), sin(), acos(), asin(), ceiling(), floor(), round(), etc.

Results near zero are expressed in scientific notation, where "e" means "times 10 to the power of", e.g. "log(1.00001)" is "9.99995e-06". An infinite result is NOT an error in S-plus, e.g. "3/0" has result "Inf". Impossible calculations generally return "NA" as the result along with a warning. Try "log(-1)".

Some functions are mainly used for their "side-effects" rather than for their result. One example is q(); we never see the result, just the side effect of quitting Splus. Another example is motif() which has the side effect of putting an empty graphics window on the screen.

• You can use control-c to get out of a long calculation or an erroneous command. The computer responds "Interrupt", then you press Return.

• All of the functions and data (collectively called "objects") in Splus are normally assigned to a named variable which can be as short as a single letter or more than 50 letters, numbers and periods long. Underscores are NOT allowed. Variable names are case sensitive, so T is different from t. You can see the contents of a variable by typing its name, because the expression is evaluated as its contents.

Variables live in different "frames". This means that there can be two variables with the same name at the same time. Normally this takes the form of a built-in function in one of the built-in frames along side of an object of the same name that you created in a user frame. It is best to avoid this situation. The easy way is as follows. If you want to create an object named "Howard", first type "Howard" at the S-plus prompt. If a "Howard" object already exists, its contents will be displayed; if you see a message like 'Error: Object "Howard" not found', the object name is unused and safe for you to use.

Some object names you may not use are break, for, function, if, in, next, repeat, return, and while. You should also avoid c, q, s, t, C, D, F, I, T, diff, mean, names, pi, range, rank, tree, and var.

To see if you have covered up an S-plus object with one of your own of the same name, use the command "masked()". If you have masked something, e.g. "c", you should type "rm(c)" to remove your version. (You can't accidentally erase S-plus's objects.)

• Everything after the pound sign is a comment, i.e. ignored by S-plus.

```
\# Try out the sine function
sin(pi)  \# should be 0 if angles are in radians
```

• You can assign a number, logical value (T or F), or string value to a name using one of the assignment operators "_" or "<-", which many people read as "gets". Try the following and some of your own.

```
x<-T
x
x_3
x
y_"This is y"
y
y_sqrt(10)
x+y
```

Single assignments are treated by S-plus as vectors of length 1. That is why they print with a leading "[1]" indicating that we are at the beginning of a vector.

• **Line editing:** You can move backwards or forwards in the current line with control-b or control-f. Then you can insert characters by typing them, or delete characters with "Backspace" or control-d. Control-a takes you to the beginning of the line and control-e to the end. Control-k deletes the rest of the line. You can press "Return" at any time to execute the line. You can also use control-p and control-n to move to previous or next lines, e.g. to set y equal to the square root of 20, press control-p until you see the line with "sqrt", press control-b three times until the cursor is over the 1, type control-d to delete the 1, then type 2 and "Return" to assign the square root of 20 to y. A final line editing command is control-r. This recalls a previous command (even from a previous computer session). E.g., press control-r, type "This" and press "Return" to bring up the most recent line containing the word "This" for editing.

• Objects are persistent; they have the same contents when you leave S-plus and return. They can be deleted using "rm(x)" and listed with "ls()" or 'objects(pattern="∧x")' to see only objects starting with the letter x.

• To assign multiple elements of the same mode (numeric, logical or character) to the same object, thus creating a vector, use the concatenation function, c(). One alternative is the use of a colon for an integer sequence. Square brackets are used to get subsets of a vector.

```
x_1:5
y_c(1,5,2,4,7)
y[2]
y[-3]
y[c(1,4,5)]
z_c(T,F,T)
nams_c("Satish", "Alex", "Mona")  # names is built-in S function
x+y
x==y
```

Note that the last line creates a logical vector. You must use double equal signs for equality comparison.

• Two expressions can be written on the same line if separated by a semicolon. One expression can be written over several lines *as long as* a valid expression does not end a line. The line continuation prompt is "+". E.g.

```
> x_3; print(x^2)
> x_3+
+ 5+6
```

assigns 14 to x. But

```
> x_3+5
> +6
```

assigns 8 to x and then just evaluates the number 6 without performing any assignment.

• To create a "matrix", use the matrix() function as follows:

```
> bob_matrix(c(1, 2, 3, 4,
               5, 0.5, 2, 6,
               0, 1, 1, 0),   nrow=3, byrow=T)
> bob
     [,1] [,2] [,3] [,4]
[1,]    1  2.0    3    4
[2,]    5  0.5    2    6
[3,]    0  1.0    1    0
> dim(bob)
[1] 3 4
```

The dim() function shows the number of rows and columns. Matrices can be the arguments to functions, e.g. "t(bob)" evaluates to the transpose of the matrix. If you forget the "byrow=T" argument, the numbers are filled in to the matrix column-wise instead of row-wise, i.e. you get the transpose of the matrix you wanted.

The square brackets are used to refer to the rows and columns of a matrix, similar to the way they are used for vectors.

```
bob[1,3]
subbob_bob[2:3, 2:4]
bob[1,]
bob[3,4]_7
bob[,c(2,4)]
bob[-2,]
```

• **Function arguments:** The mode() of an object evaluates to "function", "null", "list" (see below), "numeric", "logical", or "character". If you type the name of a function, you get its definition (try "mean"). If you use parentheses, the function is evaluated on its arguments. E.g. "mean(c(3,4,5,6,10))" gives 5.6. Use "args(mean)" to get a short list of the arguments for the mean() function: "function(x, trim = 0, na.rm = F)". Arguments followed by an equal sign have default values and need not be specified. Note that "mean(3,4,5,6,10)" causes an error because the numbers to be averaged must all be in the first argument. Optional arguments can be added in the correct order with no missing arguments using commas, e.g. "mean(c(3,4,5,6,10), 0.2, T)" or by naming just those arguments that do not take on default values, e.g., "mean(c(3,4,5,6,10), na.rm=T)".

• Use help(mean, win=T) to get the complete description of a function. Use help.start() to get searchable help.

• Lists contain unlike data. E.g. try

```
> who_list(name="Howard", age=45, married=T)
```

Then "who" prints the entire list, and e.g. who$age prints a component of the list. Also "names(who)" lists all of the components of the list. Lists are commonly used by statistical procedures to return a variety of information in a single object.

Enter

```
x1_rnorm(10); x2_rnorm(10); y_rnorm(10)
rslt_lm(y~x1+x2)
```

then check the names in rslt, print a few individual components, and print the entire list.

• A data.frame is a very special kind of list. It consists of many rows or records all containing the same type of information in several columns. The columns may or may not be of the same modes. Data.frames are commonly use to store the data we will analyze, and many statistical functions expect input in the form of a data.frame. Data.frames are created using either of two methods:

```
> name_c("Howard", "Kathy", "Rachel", "Madeline")
> age_c(45,46,17,15)
> married_c(T,T,F,F)
> myfamily_data.frame(name, age, married)

> conc_read.table("concord1.dat", header=T)
```

The second method reads the data from a file. The header=T argument indicates that the first line of the file contains the column names. The default is header=F (how can you find this out?). The file may contain "NA" to indicate missing data. (The "na.strings" argument can be used to tell read.table() that some other code(s) are used in the file to indicate missing data.)

Use "dim(conc)" and "names(conc)" to see what is in the data.frame. Use "conc[1:5,]" to view the first few lines.

Try these and your own similar commands:

```
nrow(conc)
ncol(conc)
conc$water80[1:8]
conc[1:8, 3]
conc[1:8, "water80"]
median(conc$water80)
boxplot(conc$water80-conc$water81)
boxplot(conc$water79, conc$water80, conc$water81)
median(conc$water79)
```

Something is wrong with water79! Examine that variable. Note the periods, which shouldn't be there. Technically, read.table() saw a mix of numbers and non-numeric items (periods) in that column of the file and created a categorical "factor" with 82 levels corresponding to the 82 different values in the column. That is why the boxplot only goes up to 82. We will learn about factors later in the course. For now, we can re-read the file, instructing S-plus to treat periods as missing data.

```
conc_read.table("concord1.dat", header=T, na.strings=".")
conc$water79
boxplot(conc$water79, conc$water80, conc$water81)
boxplot(conc$water79, conc$water80, conc$water81, names=c("1979","1980","1981"),
  ylab="Cubic Feet of Water", main="Concord Water Study")
```

```
median(conc$water79)
median(conc$water79, na.rm=T)
```

Several statistics commands have the optional argument "na.rm" to remove the NA's when calculating their result.

• A for loop is a statement of the form "for (argument in range) {}". It causes the following statement (or statements if the curly braces are used) to be repeated with the argument taking on each of the values in the range.

```
print(myfamily$age)
for (i in 1:3) print(myfamily$age[i])
for (i in 2:4) print(median(conc[,i], na.rm=T))
for (n in 1:10) print(c(n, n^2, n^3))
# This next one is hard if you haven't studied programming--ask for help
for (J in 1:ncol(conc)) {
  if (is.numeric(conc[,J])) {
    cat(names(conc)[J], ":")
    cat("  Median =", median(conc[,J]))
    cat("  Mean =", round(mean(conc[,J]),3), "\n")
  }
}
```

Try some variations on for loops.

• An alternative to for loops for data.frames is lapply(). (There is also apply() for matrices.) It is much quicker than a for loop, so consider using it when you have a slow for loop.

```
lapply(conc, is.numeric)
unlist(lapply(conc, is.numeric))
unlist(lapply(conc[, c(2:6, 8:10)], median))
unlist(lapply(conc[, c(2:3, 5:6, 8:10)], median))
unlist(lapply(conc[, c(2:6, 8:10)], median, na.rm=T))
round(unlist(lapply(conc[, c(2:6, 8:10)], median, na.rm=T)),2)
```

You will learn to carefully read commands like the last one "inside out". They may appear overwhelming at first, but you can learn to read them. It may help to actually enter them starting from the inside, e.g. "conc[, c(2:6, 8:10)]", then "lapply(conc[, c(2:6, 8:10)], median, na.rm=T)", etc.

Note how one bad column (missing values in water79) causes an error for the entire lapply(). Adding na.rm=T works for median, but var() does not have that option. One solution is to use na.omit().

```
dim(conc)
conc0_na.omit(conc)
dim(conc0)
unlist(lapply(conc0[, c(2:6, 8:10)], var))
thevars_.Last.value
sqrt(thevars)
```

Note how the special variable ".Last.value" can be used to save the last result if you forgot to assign it to a variable.

• **Reading in commands or functions** To read in commands or functions from a file rather than typing them in, use source(). Try "source('symplot')" to read in the function for symmetry plots. Then "symplot(rnorm(40))" will show a symmetry plot of symmetric data, while "symplot(rchisq(40,5))" shows asymmetric data.

You can type "symplot" to see the commands in symplot(). Or look at the file symplot.q using xemacs or a Unix command such as "cat" or "more"; this method may show additional comments about the command.

You can check the arguments for any function with "args(symplot)".

Exercises

1. Check the arguments for t.test(). Check the help for t.test(). Perform a 2 sample t-test for d1_c(1,3,4,5,4,3,1), d2_2:8. Repeat, dropping the assumption that the variances are equal.

2. Create a new function to find the standard deviation by typing:

   ```
   sd_function(x) {
     return(sqrt(var(x)))
   }
   ```

   Show that the standard deviation of 4, 5, 8 and 10 is 2.753785.

3. Make a plot of x=1,2,3,4,5 vs. y=1.2,2.3,3.1,4.8,6.2 using plot(x,y). Pull up the searchable help system using help.start(). Redo the plot with a title, and some x and y labels. Use lines() to overlay connecting lines between the symbols. Add your name as a subtitle.

   You could make a print-out of your graph by clicking the Print button, but that causes S-plus to both print the graph and create a postscript version of the graph in your Unix file space under a computer generated name. A better solution is to source("ps.q") once to load the function ps(). Then

   ```
   > ps("myfile")
   ```

creates a file named myfile.ps with the current graph. You can also use

```
> ps("myfile", T)
```

to save and print the graph.

Now myfile.ps sits around in your class directory, to be reprinted from Unix or included in a report if needed. You can use e.g. "!lpr myfile.ps" to issue Unix commands from inside S-plus.

4. Generate 100 random numbers from a Gaussian distibution with mean=10 and sd=2 using "rn_rnorm(100,10,2)". Plot a histogram of them. Try changing the number of bars in the histogram. Use par(mfrow=c(3,2), oma=c(1,1,3,1)) to divide your plot area up into 3 rows of 2 plots each and save space for an overall title on top. Plot 6 different sets of 100 random N(10,4) points using symplot() to get a feel for how far true normal data of size 100 varies from completely symmetrical. Use "mtext("The overall title", cex=1.2, outer=T)" to get an overall title (cex adjusts the character size).

   (par(mrow=c(1,1), oma=rep(0,4)) returns the plotting to full screen when you are done.)

   Note that any plot command automatically opens a graph window. But if you want to use par() before plotting your first graph, you must open the graph window manually using "motif()".

5. Using the variable y from exercise 3, compute the normalized version of y, say "yn", obtained by subtracting the mean and dividing by the standard deviation.

6. Write you own function to describe a vector. It should print the mean, median, sd, min, and max. You can use commands of the form: cat("The mean is", mean(x), "\n") where the "\n" means "new line".

   Add an optional second argument "plotit=F", which will make a histogram if you use the form "describe(conc$educat, T)". This involves a statement in your function such as

   ```
   if (plotit==T) hist(x)
   ```

   Use your function to describe conc$educat.