Stat 3341. Advanced Modern Statistical Computing I

Fall 2006

Department of Statistics, University of Pittsburgh

Hints for Using Splus/R

Taeyoung Park

This is designed to give a brief overview of Splus and some of the main commands you will need to know. The following are other sources for more in-depth information:

1. Online help: From the Splus prompt, type `help.start()`. A help screen will come up. If you know the command you need help with, you can also type `help(commandname)`. On Windows Splus, there is a help menu at the top of the screen with extensive documentation.

2. User's Guide: A User's Guide can be downloaded from the Splus web site:

   `http://www.insightful.com/support/documentation.asp`

3. Online quick overview:

   `www.utstat.toronto.edu/splus/contents.html.`

4. Another source of manuals and books:

   `www.umich.edu/$\sim$cscar/splus/splusintro.html.`

There are several ways to obtain Splus or R on the web:

1. You can download a student version of Splus:

   `http://elms03.e-academy.com/splus.`

2. R can be viewed as a free version of Splus, so it is very similar in appearance to Splus but has no tab menu. The implementation of a loop function in R is much faster than in Splus. R can be downloaded from the web site:

   `http://www.r-project.org.`

3. For those of you who prefer using a Linux/Unix machine, connect `bacchus.stat.pitt.edu`. Then type `R` at the prompt and a R command line (>) will come up. To quit R when running it on a Linux/Unix machine, type `q()` at the command line.

Now, a brief overview of some important Splus information.

1. How to run an Splus program

(a) You can type commands right at the command line, one at a time. The result will show on the screen automatically.

(b) For those of Linux/Unix users, you can write a series of commands as a text file and run it as a program (this method is highly recommended because most of time your programming is more than one line). There are two ways to do this. The first is to type `source("infile.txt")` at the command line, where `infile.txt` is the name of the file that contains the commands. The second option is to run the program as a batch job. This also enables you to leave it running even if you log out of the computer. To do this, type `Splus BATCH infile.txt outfile.txt` at the Unix prompt on either `nice` or `stat`. infile.txt should contain the program as a series of commands, and the output of the program will be printed to outfile.txt. When running programs in this way, you must specify what you want printed out by using the print command: `print("This is what I want to print")` or `print(objectname)`. Within a program, you can make comments by typing `#` at the beginning of each line.

2. Basics of Splus

(a) The main thing to know about Splus is that everything is done as vectors. So if x and y are both vectors of length n, if you type `x+y` the result will be a vector of length n, where each element is the sum of the corresponding elements of x and y. Realizing this is important as it helps minimize loops in Splus, which are very slow.

(b) There are 2 assignment methods. Either `z <- x+y` or `z_x+y` will assign the vector sum of x and y to a vector named z.

(c) Note: To do a dot product multiplication of 2 vectors, you must type `vector1%*%vector2`. Typing `vector1*vector2` will result in a vector of the same length as vector1 and vector2, where each element is the product of the corresponding elements of vector1 and vector2.

3. Reading in data

(a) To enter a vector of values manually:

```
data <- c(1,2,3,4,5)
```

(b) To read a text file that contains a data set:

```
data <- read.table("C:/datafile.txt",header=T).
```

`Header=T` is used if the `.txt` file contains a row at the top giving the variable names. Otherwise use `header=F`. Make sure the data file is in the right directory.

(c) The `.txt` file can be shown by typing the assign name (`data` in this case) at the prompt. When the `.txt` file is a matrix, you can call a column variable by typing either `data[,1]` or `data$firstcolumn` at the prompt.

4. Making new variables (Some variables need to be initialized in these ways before they can be referenced)

(a) Create a new vector of zeros of length `n`:

```
new.vector <- rep(0, n)
```

(b) Create a new matrix of zeros of size `n` by `m`:

```
new.matrix <- matrix(0, nrow=n, ncol=m)
```

5. Loops

(a) For loops have the following form:

```
for (i in 1:10){ commands here }
```

(b) If loops have the following form:

```
if(a==5){ commands here }
  else if(a<=3){ other commands here }
  else{ and other commands here }
```

(c) While loops have the following form:

```
while(a!=0){ commands here }.
```

The `!=` means "not equal". Be careful not to make infinite while loops!

6. Graphics

(a) Graphics are done from the command line. Graphics will not print to the output file when batch jobs are run.

(b) In the case of Linux/Unix users, a graphics window must be opened before making any graphics: `motif()`. To dismiss a graphics window: `dev.off()`. This is not necessary in Windows Splus, but if you want to have more than 1 graphics window at a time open in Windows Splus, you can use `win.graph()` to open a new graphics window.

(c) To make a histogram of my.vector:

```
hist(my.vector, main="Put main title here")
```

(d) To do a plot of a single vector: `plot(vector1)`

(e) To do a scatterplot of 2 vectors:

```
plot(vector1, vector2, xlab="first vector", ylab="second vector",
     main="Scatterplot", xlim=c(0,100), ylim=c(100,200))
```

Only the first two arguments are required. The other arguments are optional!

(f) To draw a line of 2 vectors:

```
plot(vector1, vector2, type="l", xlab="first vector", ylab="second vector",
     main="Draw a line", xlim=c(0,100), ylim=c(100,200))
```

The option `type="l"` means a line; the option `type="p"` means a point.

(g) To add a line/point to the plot:

```
lines(vector1,vector2) : draws a line connecting two vectors
abline(a,b)            : a line with the intercept a and slope b
abline(h=a)            : a horizontal line with the y-value a
abline(v=b)            : a vertical line with the x-value b
points(vector1,vector2): draws each point of two vectors
```

(h) Other plot commands:

```
qqplot           : quantile plots
qqnorm and qqline: normal quantile plots
boxplot          : box plots
contour          : contour plots
persp            : perspective plots
```

(i) Several plots can be displayed on one page. To get a 3 by 2 grid of plots, type

`par(mfrow=c(3,2)).`

Type `par(mfrow=c(1,1))` to return to the default.

7. Generating random draws from various distributions

(a) Splus has a number of built in functions for generating draws, calculating the density, and calculating tail probabilities for standard distributions.

(b) Normal: `dnorm(x, mean=0, sd=1)` would give the density of the value x (or vector of values x), from a normal with mean 0 and standard deviation 1. `rnorm(n, mean=0, sd=1)` will give `n` draws from a normal distribution with mean 0 and standard deviation 1. Type `help(rnorm)` for more information on these.

(c) The names for various distributions are as follows:

```
norm  :  normal
beta  :  beta
chisq :  chi squared
pois  :  Poisson
gamma :  gamma
binom :  binomial
nbinom:  negative binomial
t     :  t
f     :  F
```

Then you can add one of four letters to each distribution to create a command:

```
r: random draws
d: density functions (pdf or pmf)
q: quantiles
p: cdf (cumulative probabilities)
```

(d) To sample from a given set of values:

`sample(x, 1000, replace=T, prob=NULL).`

This will generate 1000 values from the values in `x`, with replacement, where each value in `x` has the same probability of being sampled. You can also specify these probabilities if they are unequal. See `help(sample)` for more information.

8. Other useful statistics functions

   (a) This is not an exhaustive list of the functions you will need to use, but here are some useful ones. Use the Splus help options to search for others you need.

   (b) Some usual summary functions:

   `mean(vector), var(vector), sd(vector), sqrt(object).`

   (c) Logarithms: `log(value), log10(value)`.

   (d) Cholesky decomposition of a matrix: `chol(matrix)`.

   (e) Equally spaced points between `a` and `b` containing the margins: `seq(a,b,length=100)`.

   (f) Equally spaced points between `a` and `b` without containing the margins:

   `ppoints(100)*(b-a)+a,`

   where `ppoints(length)` is a function that generates the sequence of points between 0 and 1, excluding margins.

   (g) T-tests: `t.test(vector1, vector2, paired=F)` will run an unpaired t-test of the mean of vector1 and the mean of vector2. You can also do a test of whether the mean of a vector is equal to a certain value. Type `help(t.test)` for more information.

   (h) F-tests: `var.test(vector1, vector2)`

   (i) Simple linear regression: `reg <- lm(z∼x+y, data=mydata)` runs a simple linear regression of `z` on `x` and `y`, using the data frame mydata. The output is sent to the object `reg`. To look at a summary of the output, type `summary(reg)`. You can also access objects like the coefficients directly by using `reg$coef`. `reg$fit` will give the fitted values of the regression. `names(reg)` will show a list of the regression objects you can access directly. Again, look at `help(lm)` for more information on that.

   (j) Logistic regression: `logitreg <- glm(z∼x+y, family=binomial, data=mydata)` will run a logistic regression (binomial family with logit link) of `z` on `x` and `y`. The output and other commands are similar to `lm`. See `help(glm)` for more information. Information on running other general linear models is also given there.

9. Writing functions in Splus

   (a) You may find it useful to occasionally write your own functions in Splus. Here is the syntax for that:

```
myfunction <- function(arguments){
  commands here, ending with return(output) }.
```

The arguments are whatever you need to send to the function (there can be multiple arguments, separated by commas, and arguments could be vectors or any other objects). The line `return(output)` will return whatever output is. You should put whatever you want the output of the function to be. For example, try the following commands where $g(x) = e^{-x^2}/x + \sin(x) - 1/x$:

```
g <- function(x){
  output <- exp(-x^2)/x+sin(x)-1/x
  return(output)
}
x <- ppoints(100)*10-5
plot(x,g(x),type="l")
```

(b) Exercise. Write a function that implements the bisection algorithm:

`bisect(a,b,max.error=1e-5).`

– Choose two points, a and b, such that $g(a)g(b) < 0$.

– Repeat

$c \leftarrow \dfrac{a+b}{2}$

if $g(c) = 0$ then stop.

else

if $g(a)g(c) < 0$ then $b \leftarrow c$.

else then $a \leftarrow c$.

until $|a - b| < \epsilon$.