Introducing S-Plus

The purpose of these lectures is to introduce the student to some basic concepts regarding S-Plus. As we shall see, S-Plus is a language which has specially designed for explaratory data analysis and Statistics. It can be used either interactively or by programming. In these notes we will see how to program in S-Plus and by doing so, we will be able to build our own *functions*-a concept that will be described later in the lectures. The purpose of these notes is to introduce the student into

- General concepts of S-Plus.
- Using S-Plus in doing some elementary data analysis.
- Some programming concepts in S-Plus.

It would be most helpful initially to present a series of *functions* as an introductory session to the language.

An Introductory Session in S-Plus

The following session aims on introducing you to some features of S-Plus and show you how to use them. Most of the following commands might be confusing. However this will soon be eliminated as we are becoming more familiar with the language.

First Example

`x <- rnorm(50)`	Generate two standard normal vectors of x
`y <- rnorm(x)`	and y coordinates.
`hull <- chull(x,y)`	Find their convex hull in the plane
`plot(x,y)`	Plot the points in the plane,
`polygon(x[hull], y[hull],dens=15)`	and mark in their convex hull.
`objects()`	See which S-Plus objects are now in `_Data` directory.
`rm(x,y)`	Remove objects.

Second Example

`x <- 1:20`	Create $x = (1, 2, ..., 20)$.
`w <- 1+sqrt(x)/2`	Create a weight vector of standard deviations.
`dummy <- data.frame(x=x,`	Make a *data frame* of two columns
`y=x+rnorm(x)*w)`	x and y and examine it.
`dummy`	
`objects()`	See which S-Plus objects are now in `_Data` directory.
`fm <- lm(y~x, data=dummy)`	Fit a simple linear regression of y on x
`summary(fm)`	and look at the results.
`fm1 <- lm(y~x, data=dummy,`	Do a weighted regression.
`weight=1/w^2)`	
`lrf <- loess(y~x, data=dummy)`	Run a nonparametric local regression function.
`attach(dummy)`	The columns of the data frame are visible.
`plot(x,y)`	Plot of x versus y.
`lines(x, fitted(lrf))`	Add to the plot the fitted model.
`abline(0,1,lty=3)`	Add to the plot the true regression line.
`abline(coef(fm))`	The simple regression line.
`abline(coef(fm1), lty=4)`	The weighted regression line.
	At any time you you can produce a copy of the graphics by clicking on the `Graph` section and selecting the `Print` option.
`detach()`	Remove data frame from the list.
`plot(fitted(fm), resid(fm),`	Standard regression plot to check
`xlab`=Fitted Values",""	for heteroscedasticity.
`ylab`=Residuals", main=""	
Residuals vs Fitted")"	
`qqnorm(resid(fm), main=`	A normal QQ plot.
Residuals QQ Plot")"	
`rm(fm,fm1,lrf,x,dummy)`	

Third Example

	Graphical capabilities of S-Plus: contour and 3–dimensional plots.
`x <- seq(-pi,pi,length=50)`	x is a vector of 50 equally spaced values in $(-\pi, \pi)$.
`y <- x`	The same as x.
`f <- outer(x,y,`	Define a matrix f whose rows and columns
`function(x,y)`	are indexed by x, y respectively and correspond
`cos(y)/(1+x^2))`	to the values of $\cos(y)/(1+x^2)$.
`oldpar <- par()`	Save the plotting parameters.
`par(pty="s")`	Set the plotting region to *square*.
`contour(x,y,f)`	Make a contour map of f.
`contour(x,y,f,`	Add to the plot more lines for detail.
`nint=15, add=T)`	
`fa <- (f-t(f))/2`	fa is the asymmetric part of f.
`contour(x,y, fa, nint=15)`	Create a contour plot of fa.
`par(oldpar)`	Restore the old graphics parameters.
`persp(x,y,f)`	Create perspective and high
`persp(x,y,fa)`	images plots.
`image(x,y, f)`	
`image(x,y,fa)`	
`objects(); rm(x,y,f,fa)`	Clean up everything.
`q()`	Quit S-Plus.

Basic Concepts

S-Plus is an interpreted language in which individual language expressions are read and then immediately executed. By contrast, C and Fortran are compiled languages. This means that complete programs in the language are translated by a compiler into the appropriate machine language. The great advantage of interpreted languages is that they allow incremental development. In other words, you can write a function, run it, then write another function that call the previous one and so on and henceforth. However notice that compiled code runs faster and requires less memory than interpreted code.

Syntax

You interact with S-Plus by typing expressions, which the interpreter evaluates and executes. For instance

```
> sqrt
function(x)
x^0.5
```

or

```
log
function(x, base = 2.71828182845905)
{
    y <- .Internal(log(x), "do_math", T, 106)
    if(missing(base))
        y
    else y/.Internal(log(base), "do_math", T, 106)
}
```

Notice that S-Plus is case sensitive. This means that x and X are different objects. A function call is usually typed as a function name followed by an argument list. For example

```
> plot(corn.rain)
> mean(corn.rain)
[1] 10.78421
```

Infix operations are functions with two arguments that have special calling. For instance

```
> 2+5
[1] 7
> 3*6.8
[1] 20.4
> 12.6/6
[1] 2.1
```

One of the most frequently used infix operators is the assignment operator <- So

```
test <- 4
> test
[1] 4
```

Another common operator is the subscripting operator [, used to extract subsets of an S-Plus object. For example

```
letters
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n"
 "o" "p" "q" "r" "s" "t" "u" "v" "w" "x" "y" "z"
> letters[3]
[1] "c"
> letters[-3]
 [1] "a" "b" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n"
 "o" "p" "q" "r" "s" "t" "u" "v" "w" "x" "y" "z"
```

Logical values:

```
 j <- 1:26
> j < 5
 [1] T T T T F F F F F F F F F F F F F F F F F F F F F F
> letters[j < 5]
[1] "a" "b" "c" "d"
```

Subscripting is very important in making efficient use of S-Plus because it emphasizes treating data object as whole entities, rather than a collection of individual observations.

As a last note, we need to point out that every S-Plus expression is interpreted by the evaluator and returns a `data object`. Data objects have modes

- logical
- numeric
- complex
- character

The modes are listed from least informative to most informative. When you want to combine different modes together, S-Plus uses the most informative mode. The following example illustrates the concept:

```
> -3.6
[1] -3.6
> "Munich"
[1] "Munich"
> c(T, F, T)
[1] T F T
> c(-2, pi, 2)
[1] -2.000000  3.141593  2.000000
> c(T, pi, F)
[1] 1.000000 3.141593 0.000000
> c(T, pi, "Munich")
[1] "TRUE"             "3.14159265358979" "Munich"
> mode(c(T, pi, "Munich"))
[1] "character"
```