M3S8/M4S8: S-Plus worksheet

What is S-Plus ?

S-Plus is a very powerful (in my view) data analysis and graphics system.

S-Plus provides you with with a huge number of statistical techniques, from simple linear regression to non-linear modelling to time-series analysis.

How do I get help?

S-Plus comes with extensive on-line documentation, including a search facility for documentation on specific functions. The manuals are available on-line or alternatively can be found in the maths library.

How do I use it?

The opening main window of S-Plus brings up the default *Object Browser* window. You'll also see the Main menu (containing the options `File, Edit, View, Insert, Format...`), and directly underneath the Main menu, the Standard toolbar (containing icons which provide shortcuts to commonly used items – to see what the icon represents, just position the mouse over the icon, and a description will appear).

The first thing to do is to open the *Commands* window, as you'll soon see, most things in S-Plus can be achieved in several different ways, two ways of getting at the Commands window are

1. Through the Main menu: select `Window` → `Commands Window`.
2. Through the standard toolbar.

Once the Commands window is open, you'll see the `>` prompt – this is where we type S-Plus commands, so we're ready to go...

Arithmetic Operations

Use the operators `+` , `-` , `*`, and `/` for addition, subtraction, multiplication and division respectively. The operator `^` is used for exponentiation (including fractional). The operators to obtain integer quotients and remainders are `%/%` and `%%` respectively.
Examples:

```
> 9+7
[1] 16
> 3^2
[1] 9
> 9^(1/3)
[1] 2.080084
> 24.5 %/% 3.2
[1] 7
> 24.5 %% 3.2
[1] 2.1
```

In addition:
for the absolute value,

```
> abs(-4.5)
[1] 4.5
```

for the greatest integer less than or equal to x: $\lfloor x \rfloor$,

```
> floor(2.3)
[1] 2
```

for the smallest integer greater than x: $\lceil x \rceil$,

```
> ceiling(2.3)
[1] 3
```

Vectors and Matrices

A vector is an ordered set of values, and can be created using S-Plus in a number of ways;

```
> x <- c(3,1,7)
> x
[1] 3 1 7
> x[2]
[1] 1
> w <- 1:6
> w
[1] 1 2 3 4 5 6
> y <- seq(1,4,l=7)
> y
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0
> z <- seq(1,4,by=0.25)
> z
[1] 1.00 1.25 1.50 1.75 2.00 2.25 2.50 2.75 3.00 3.25 3.50 3.75 4.00
```

Matrices are constructed using the `matrix` command,

```
> A <- matrix(c(19,8,11,2,18,17,15,19,10),nrow=3)
> A
     [,1] [,2] [,3]
[1,]   19    2   15
[2,]    8   18   19
[3,]   11   17   10
```

Matrices can also be constructed by "binding" vectors together using `rbind` (assigns vectors to rows), or `cbind` (assigns vectors to columns):

```
> m <- c(14,13,10)
> n <- c(10,11,15)
> o <- c(19,3,15)
> B <- cbind(m,n,o)
```

```
> B
       m  n  o
[1,]  14 10 19
[2,]  13 11  3
[3,]  10 15 15
```

Most calculations on vectors or matrices are carried out *element by element*, so for example, if $X = \{x_{ij}\}$ and $Y = \{y_{ij}\}$ we have $X * Y = \{x_{ij}y_{ij}\}$:

```
> A*B
       m   n   o
[1,] 266  20 285
[2,] 104 198  57
[3,] 110 255 150
```

To perform matrix multiplication, use the operator `%*%`

```
> A %*% B
       m   n   o
[1,] 442 437 592
[2,] 536 563 491
[3,] 475 447 410
```

Elementary Functions

The following table lists functions included in S-Plus .

Name	Operation
`sqrt`	Square root
`abs`	Absolute value
`sin, cos, tan`	Trigonometric functions (radians)
`asin, acos, atan`	Inverse trigonometric functions (radians)
`sinh, cosh, tanh`	Hyperbolic trigonometric functions (radians)
`asinh, acosh, atanh`	Inverse hyperbolic trigonometric functions (radians)
`exp, log`	Exponential and natural logarithm
`log10`	Common logarithm
`gamma, lgamma`	Gamma function and its natural logarithm

Each function acts *element by element* on its argument:

```
> A
     [,1] [,2] [,3]
[1,]   19    2   15
[2,]    8   18   19
[3,]   11   17   10
> sqrt(A)
         [,1]     [,2]     [,3]
[1,] 4.358899 1.414214 3.872983
[2,] 2.828427 4.242641 4.358899
[3,] 3.316625 4.123106 3.162278
```

```
> sin(A)
            [,1]       [,2]       [,3]
[1,]  0.1498772  0.9092974  0.6502878
[2,]  0.9893582 -0.7509872  0.1498772
[3,] -0.9999902 -0.9613975 -0.5440211
```

You can use `log` to compute logarithms of any base with the optional argument `base=`. For example, to compute $\log_2 7$:

```
> log(7,base=2)
[1] 2.807355
```

In order to see the vectors and matrices we have created we use the `ls` command:

```
> ls()
[1] "Last.value"     "A"     "B"    "m"
[5] "n"              "o"     "w"    "x"
[9] "y"              "z"
```

We can also track backwards and forwards through old commands by using the up and down arrows, this enables us to edit and re execute old commands without having to retype them.

Graphics

I'm only going to do a very brief introduction to S-Plus graphics here - the best way to learn is to experiment with S-Plus , using the plotting commands available.

In the commands window, type the following,

```
> x <- seq(0,4*pi,l=100)
> y <- sin(x) + rnorm(100,sd=0.2)
```

The function `rnorm` generates random variates from a normal distribution, the optional argument `sd=`, specifies the standard deviation of the normal distribution. We'll be meeting more of the S-Plus generators later in the course.

Let's imagine we want a scatter plot of x versus y, one way to do this is by using the 2-D plot icon from the Standard toolbar - this throws up a menu of different plots available, the very first of which is the 2-D scatter – once that is selected we have to tell S-Plus which variables we want in the x and y axes. To do this, we type `x` and `y` into the relevant columns, then hit OK. This should throw up the required scatter plot - which should look like a messy sine wave.

S-Plus gives us the power to fiddle endlessly with graphs – adding titles changing the axes etc., etc. Have a go...

It is often easier to plot directly from the command line, try the following:

```
> x <- seq(0,4*pi,l=100)
> y <- sin(x) + rnorm(100,sd=0.2)
> plot(x,y)
> plot(x,y,type="l")
> plot(x,y,xlab="x = 0 to 4 pi",ylab="sin(x) + noise")
```

Functions

Probably the most useful thing about S-Plus is it's power as a programming language, it actually has a similar (ish) structure to C, though it has to be said without the speed!

Here I'm going to outline how to write a very simple S-Plus function, which is no more than a list of S-Plus commands.

First things first – we need an editor, for this, we use the `Edit` function.

```
> Edit(my.lag.plot)
```

This will open up a Script file, ready for the commands, the Script file will contain the following:

```
"my.lag.plot" <-
function()
{
}
```

It is worthwhile documenting your code, to do this just precede any comments with a `#`. More often than not, you assume that you will remember what functions do, but take it from me (and from years of personal frustrating experience), you won't! So comment - at the very least have a line at the top of the function describing what it does.

Let's try a simple piece of code, take note of the structure as this remains the same as the code becomes more complicated. Amend the Script file to contain the following:

```
"my.lag.plot" <-
function(x)
{
# function to plot x(t) v. x(t+1)
# determine n, the number of elements in the vector x
n <- length(x)
# store x(1), x(2), ..., x(n-1) in x1
x1 <- x[1:(n-1)]
# store x(2), x(3), ..., x(n) in x2
x2 <- x[2:n]
# do a scatter plot of x1 v. x2
plot(x1,x2,xlab="x(t)", ylab="x(t+1)")
title("Lag one plot")
}
```

Take note of the `x` in the second line (`function(x)`), this tells the function to use the argument `x`.

We then need to run the script to make sure it's error free, to do this choose `Script` → `Run`. Or alternatively, hit `F10`.

Once we have run the script the function `my.lag.plot` is saved in our home directory for later use. Remember that any changes we make to the Script file will need to be re executed before they will be saved.

Note, again, that we were able to produce the plot using the `plot` command rather than using the 2-D plot menu.

To execute the function we need to go back to the command window and type the following,

```
> x <- arima.sim(128,model=list(ar=.9))
> my.lag.plot(x)
```

The `arima.sim` command produces (you've guessed it) variates from an arima model (in this case an AR(1) model with $\phi_{1,1} = 0.9$), so we should see positive correlation in a lag-one scatter plot.

In fact, S-PLUS already provides us with a function to produce lag plots, try the following

```
> lag.plot(x, lags=1)
```

This should look similar to the plot created using our function `my.lag.plot`.

Printing

We can print either Graph sheets or Script files, first we must select what we want to print (click once on the item - let's say a graph), then choose `File` → `Print Graph Sheet...`, notice that if we haven't selected anything then there isn't a `Print` option under the `File` menu. Assuming that the printer is on line (and turned on!), this should do the trick.

Formatting data

There are several useful functions in order to format data. The `trunc` function acts like `floor` for elements greater than 0 and like `ceiling` for elements less than 0. The `round` function allows you to specify how many digits to include after the decimal point, while `signif` rounds to a specified number of significant digits.

In the following I generate 10 random uniform numbers on [-100,100], obviously the numbers you generate will be different to those below as the seed for the random number generator in S-PLUS changes. If you wish to reproduce results from the S-PLUS generators use the function `set.seed(i)` with `i` an integer between 0 and 1000.

The default bounds for `runif()` are [0,1], so to generate 10 random uniforms on [0,1] just type `runif(10)`.

```
> y <- runif(10, -100, 100)
> y
 [1] -11.721349  -9.683915  52.372354 -62.671819 -16.399008 -70.398967
 [7] -83.582508  21.236722 -25.531344 -57.007051
>  trunc(y)
 [1] -11  -9  52 -62 -16 -70 -83  21 -25 -57
> round(y, digits=3)
 [1] -11.721  -9.684  52.372 -62.672 -16.399 -70.399 -83.583  21.237
 [9] -25.531  -57.007
> signif(y, digits=3)
```

```
 [1] -11.70  -9.68  52.40 -62.70 -16.40 -70.40 -83.60  21.20 -25.50
[10] -57.00
```

Summary Functions

S-PLUS includes a number of functions for summarizing data. These functions take an input vector or matrix and return the appropriate summary statistics.

Name	Operation
`min, max`	Smallest and largest values
`range`	Returns a vector of length two, containing the min and max
`mean median`	Arithmetic mean and median
`quantile`	Returns sample quantiles for a given data set.
`var`	Returns the variance of a vector, or the variance-covariance of a matrix.
`cor`	Returns the correlation matrix.

For example:

```
> mean(y)
[1] -26.33869
> range(y)
[1] -83.58251  52.37235
> var(y)
[1] 1836.461
> x <- rnorm(1000)
> quantile(x, c(0.05, 0.5, 0.95))
[1] -1.624222  0.046012  1.623061
> median(x)
[1] 0.046012
```

Logical Operators

The S-PLUS operators for comparison and logic play an important role, both in terms of extracting elements of vectors and also in the efficient use of S-PLUS .

There are two types of logical operators for And and Or, which we refer to as "vectorized" or "control". The vectorized operators act element by element, while the control operators are used with `if` or `else` and return a single logical value. Control operators are only evaluated as far as necessary to return a correct value.

First, we include a table of the operators:

Operator	**Explanation**	**Operator**	**Explanation**
==	equal to	!=	not equal to
>	greater than	<	less than
>=	greater than or equal to	<=	less than or equal to
&	vectorized And	|	vectorized Or
&&	control And	||	control Or
!	not		

Here are some examples of vectorized operators, we shall return to control operators later:

```
> x <- rnorm(6)
> x
[1]  0.6261129 1.5461053 -0.8595883 -0.3363925 -0.3004342 -1.3122208
> x < 0
[1] F F T T T T
> x < -1 | x > 1
[1] F T F F F T
> x >= 0 & x <= 1
[1] T F F F F F
> x[x < 0]
[1] -0.8595883 -0.3363925 -0.3004342 -1.3122208
```

More on writing Functions

Programming Style

Here are some guidelines to writing S-PLUS functions:

- *Modularize your code.* Try to design smaller functions rather than writing one large function.
- *Comment.* This is for yourself, six months down the line, and also for any other users who may have to interpret your code.
- *Use existing functions.* Make use of the S-PLUS built in functions, have a look in Help to see if there are functions that may be useful.

Flow of Control

Expressions in S-PLUS are normally evaluated sequentially, this flow of control can be overridden with the looping constructions described in the Table 1.

Examples of Usage: `if`

```
> x
[1]  0.6261129  1.5461053 -0.8595883 -0.3363925 -0.3004342 -1.3122208
> if(any(x<0)) cat("elements of x must be non-negative\n")
elements of x must be non-negative
> if(x[1] > 0) x+10 else x
[1] 10.626113 11.546105  9.140412  9.663607  9.699566  8.687779
```

The following function returns a 1 for positive values of a vector and -1 for negative values.

```
"sgn2" <-
function(x)
{
for(i in 1:length(x)){
     if(x[i] > 0) x[i] <- 1
else if(x[i] < 0) x[i] <- -1
}
x
}
```

If you find yourself using `if` statements over each element of an array remember the `ifelse` statement, which is considerably faster, and copes with missing values:

```
"sgn" <-
function(x)
{
ifelse(x > 0, 1, ifelse(x < 0, -1 ,0))
}
```

Examples of running these functions:

```
> sgn2(x)
[1]  1  1 -1 -1 -1 -1
> sgn(x)
[1]  1  1 -1 -1 -1 -1
> sgn2(c(1,3,NA,-1))
Error in sgn2(c(1, 3, NA, -1)): Missing value where logical needed:
if(x[i] > 0) x[i] <- 1 else
if(x[i] < 0) x[i] <- -1
. . .
Dumped
> sgn(c(1,3,NA,-1))
[1]  1  1 NA -1
```

Construction	**Description**
`if` (*cond) expression*	Evaluates *cond*; if true, evaluates *expression*
`if` (*cond) expr* `else` *condexpr*	Evaluates *cond*; if true, evaluates *expr*. If false, evaluates *condexpr*.
`ifelse` (*cond, expr1, expr2*)	The `ifelse` functions is a vectorized version of the `if` statement. It evaluates *cond* and returns elements of *expr1* for `TRUE` elements, and elements of *expr2* for `FALSE` elements.
`switch` (*expr, ...*)	Evaluates *expr*, which is compared to the remaining arguments, if it matches one of these arguments exactly, the value of the evaluated argument is returned.
`break`	Terminates current loop and passes control out of the loop.
`next`	Terminates current iteration and immediately starts next iteration of the loop.
`return` (*expr*)	Terminates current function and immediately returns the value of *expr*.
`stop` (*message*)	Used to signals an error condition, terminates the current function and returns to the S-PLUS prompt printing the character string *message*.
`while` (*cond) expr*	Evaluates *cond*; if true, evaluates *expr*, then goes back to the top of the loop, evaluating *cond* again.
`repeat` *expr*	A simpler version of `while`, it simply repeats *expr* indefinitely, therefore, a repeat loop will need some `break` statements built in.
`for` (*name* `in` *expr1*) *expr2*	Evaluates *expr2* once for each *name* in *expr1*. Despite the (justified) popularity of `for` loops in most programming languages, they tend to be very inefficient in S-PLUS , use vectorized notation instead where possible.

Table 1: Looping and control constructions

Plotting

Given that I have rather Luddite tendencies, the following is based on plotting functions within S-PLUS which I use on a unix platform, the same results can probably be obtained just as well using other methods on the PC version, for example there is a great annotation icon on the tool bar which allows you to add all sorts of things to your already created plots.

High-Level Plotting Functions

Here, we introduce plotting functions which produce a complete figure by themselves, the most prominent of these is the function `plot()`, which, as its default, produces scatter plots.

```
plot(x,y, type="p")

hist(x, nclass=, breaks=, plot=T, probability=F)

barplot(x,horiz=F)

boxplot(...)
```

For the `hist()` and `barplot()` functions, there are many additional arguments to control shading patterns e.t.c. – have a look at the help files.

Most high-level plotting functions require two arguments, most commonly, two vectors giving the x- and y-coordinates for the points to be plotted. If, just one vector of length n is provided, it is assumed that these are the y-coordinates and the x-coordinates are assumed to be the vector $(1, 2, 3, \ldots, n)$.

Aside from the `x` and `y` arguments, there are other arguments which can be supplied to high-level plotting functions in order to customize the plot:

`type=` the type of plot:`"p"` (points), `"l"` (lines),`"b"` (both), `"o"` (overstruck), or +`"h"` (high-density). It is also possible to specify `type="n"`, to set up the axes for a plot that will later be filled in by low-level plotting functions.

`xlab=, ylab=` labels for the x- or y-axis, the defaults are the names of the vectors.

`main=` a main title in enlarged characters above the plot.

`axes=` logical flag. The specification `axes=F` omits drawing the axes (the default is `axes=T`), useful if you wish to customize the axis (see low-level plotting functions `box()` and `axis()`).

`xlim=, ylim=` vectors giving the minimum and maximum values on the x- and y-axes, useful if you are adding data to the plot at a later stage.

`log=` character string giving the names of the axes to be logarithmic, e.g. `log="xy"`.

These are best illustrated by example:

```
> x <- seq(0,10,l=20)
> y <- x + rnorm(20)
> plot(x,y,type="b",ylab="x + random normal",main="noisy straight line")
> x <- rchisq(1000,8)
> hist(x,nclass=10)
> x1 <- rnorm(1000,0,1)
> x2 <- rnorm(1000,0,2)
> x3 <- rnorm(1000,0,3)
> boxplot(x1,x2,x3)
> barplot(c(max(x1),max(x2),max(x3)),names=c("N(0,1)","N(0,4)",
"N(0,9)"),xlab="max. of sampled normal data",horiz=T)
```

Low-Level Plotting Functions
There a several functions designed to add to existing plots, the most important are:

`lines(x, y, type="l", lty=)` add a line to the current plot.

`points(x, y, type="l", pch=)` add points to the current plot.

`axis(side, at=, labels=T, ticks=T, line=0, pos=, outer=F)` add an axis to the current plot.

- `side` a number representing the side of the plot for the axis (1 for bottom, 2 for left, 3 for top, and 4 for right).
- `at=` vector of positions for the ticks and tick labels.
- `labels=` vector of tick labels, with label[i] plotted at at[i].
- `line=` distance of axis line from the plot.
- `pos=` x- or y-coordinate position at which the axis line is plotted.

`title(main, sub=, xlab=, ylab=)` add a title to the current plot

- `main` character string for the main title.
- `sub=` character string for the sub-title plotted on the bottom below the x-axis title.
- `xlab=` character string to title the x-axis.
- `xlab=` character string to title the y-axis.

`legend(x, y, legend, lty=NULL, marks=NULL, pch=NULL)` add a legend to the current plot

- `x,y` location of the top left corner of the legend box.
- `legend` vector of character strings to be associated with line or plotting types.

`lty=` integer vector of line types.

`marks=` vector of plotting symbol numbers.

`pch=` character string of plotting characters.

`box(n=1,col=)` surrounds the current plot with a box of a specified colour of a given thickness.

`abline(a,b,coef,h=,v=)` add a line to a plot.

`a,b` intercept and slope of line.

`coef` vector containing intercept and slope.

`h=, v=` vector of y- or x-coordinates for horizontal or vertical lines.

For example (the function `lsfit()` fits a least squares regression):

```
> x <- seq(-5,5,l=20)
> y <- x + rnorm(20)
> plot(x,y,type="b",ylim=range(-10,10),xlim=range(-10,10),axes=F,
xlab="",ylab="")
> abline(lsfit(x,y),lty=2)
> abline(h=0,v=0)
> legend(-10,10,legend=c("exact","fitted"),lty=1:2)
> box()
> axis(1,at=c(-10,-5,0,5,10))
> axis(2,at=c(-10,-5,0,5,10))
> title("more noisy data")
```

S-Plus time series functions

S-Plus has many built-in time series functions, a few of which are listed below The calling names are:

`rts`	defines a regularly spaced time series.
`its`	defines an irregularly spaced time series.
`arima.sim`	simulates an arima time series.
`arima.mle`	fits an arima model.
`arima.forecast`	forecasts using an arima model.
`ar`	fits an ar model.
`acf`	estimates the acvs.
`spec.pgram`	estimates spectra using a smoothed periodogram.
`spec.ar`	estimates spectra using an ar model..

For example:

```
> x <- arima.sim(1024, model=list(ar=c(0.75,-0.5)))
> acf(x)
# raw periodogram:
```

```
> spec.pgram(x, demean=T, detrend=F, plot=T)
# fit an ar model to the data:
> ar.x_ar(x)
# calculate the true spectrum
> f_seq(0,.5,l=513)
> true.spec.ar.x_-10*log10(Mod(1-0.75*exp(-1i*2*pi*f)+0.5*exp(-1i*4*pi*f))^2)
# plot fitted ar spectrum with true spectrum
> spec.ar.x_spec.ar(ar.x,n.freq=513,plot=T)
> lines(f,true.spec.ar.x,lty=2)
```

All the figures used in the course were produced using S-Plus . I've stuck the functions and datasets on the course home page for anyone who's interested.