

CSI606
Introduction to S-Plus

Jeff Solka

Outline

- **Running S-PLUS**
- **Data in S-PLUS**
- **S-PLUS Graphics**
- **Statistics**
- **Programming in S-PLUS**

Additional References

- *Modern Applied Statistics with S-Plus*, W. Venables and B. Ripley
- *The Basics of S and S-Plus*, A. Krause and M. Olson
- *A Handbook of Statistical Analysis using S-Plus*, B. Everitt

History

- The original development of the S language was funded under the auspices of the Office of Naval Research by Professor Ed Wegman
- S has evolved into S-PLUS and is marketed by StatSci a Division of MathSoft, Inc.
- S-PLUS runs on both UNIX (SUN, SGI...) and PC (Win 95 and Win 98)

Making it Go

- Under Unix Type

```
/package/Splus33/Splus (or the appropriate  
path on your machine)
```

On the science cluster just

```
Splus  
should work
```

- Under Windows

Double click on the S-PLUS icon

S-PLUS : Copyright 1988, 1995 MathSoft, Inc.

S : Copyright AT&T.

Version 3.3 Release 1 for MS Windows 3.1 :
1995

Working data will be in _Data

```
>
```

Making it Stop

- Type

```
> q()
```

- `q()` is a function execution

- Everything in S-PLUS is a function

- `q` merely returns a listing of the function

Syntax

- **Everything that we type in S-PUS is an expression**
- **We may have multiple expressions on each line separated by ;**
`2+3;4*5;6-9`
- **We use <- for making assignments**
`b<-5+9`
- **S-PLUS commands are case sensitive**
- **The result of any expression is an object**

Executing Unix/Dos/Win Commands

- `dos("dir")`
- `unix("ls")`
- `win3("notepad c:\\windows\\splus.ini")`
- **Note we can usually interrupt things with a Ctrl-C or Ctrl-Break**

Recalling Previous Commands

- **In WINDOWS one may use the arrow up key or the history command under the menus**
- **In UNIX one may use the `history()` command or `again(command,ed=T)`**

Getting Help

- **In both environments we may use**
`help(command name)`
`?command name`
- **We may also use**
`?methods(command name)`
- **For commands with multiple methods based on different object types**

Getting Function Information

- To view information on just the arguments to a function use the command `args`

```
> args(hist)
function(x, nclass, breaks, plot = TRUE,
  probability
    = FALSE, include.lowest = T, ..., xlab =
  deparse(substitute(x)))
NULL
```

- One may also use the `arg.dialog` function
 - **This allows one to call the function on the fly and to make assignments**

Assignments in S-PLUS

- Some Examples

```
> cat<-45
```

```
> dog_66
```

```
> cat
[1] 45
```

```
> dog
[1] 66
```

```
> 77 -> rat
```

```
> rat
[1] 77
```

- **Note = is used for specifying values in function calls**

Vectors

- **A vector example**

```
> a<-c(1,2,3,4)
```

```
> length(a)
[1] 4
```

```
> a
[1] 1 2 3 4
```

- **An example with character strings**

```
> name<-c("Jeff","Solka")
```

```
> name
[1] "Jeff" "Solka"
```

Matrices

- **A matrix example**

```
> b<-matrix(nrow=2,ncol=2)
```

```
> b
      [,1] [,2]
[1,]  NA  NA
[2,]  NA  NA
```

```
> b[,1]<-c(1,3)
> b[,2]<-c(2,4)
```

```
> b
      [,1] [,2]
[1,]   1   2
[2,]   3   4
```

Functions

- **We will discuss function at length later but for now I point out how to edit a function**

`fix(ftn name)` for new functions

`edit(ftn name)` for existing ones

- **It is possible to use other editors** (notepad, jot, vi ...)

Editing Data Sets

- **We may create and modify data sets on the command line**

```
> xx<-seq(from=1,to=5)
```

```
> xx  
[1] 1 2 3 4 5
```

```
> xx[xx>3]  
[1] 4 5
```

- **We may edit our data set in our editor**

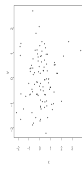
```
> mydata<-data.ed(mydata)
```

Graphics in S-PLUS

- `win.graph()` **or in UNIX** we say `motif()`
- `dev.list()` - **list currently opened graphics devices**
- `dev.cur()` - **list identifier for the current graphics device**

- **A simple plotting example**

```
> x<-rnorm(100)
> y<-rnorm(100)
> plot(x,y)
```



S-PLUS Search Path

```
> search()
[1] "_Data"
[2] "D:\\SPLUSWIN\\splus\\_Function"
[3] "D:\\SPLUSWIN\\stat\\_Function"
[4] "D:\\SPLUSWIN\\s\\_Function"
[5] "D:\\SPLUSWIN\\s\\_Dataset"
[6] "D:\\SPLUSWIN\\stat\\_Dataset"
[7] "D:\\SPLUSWIN\\splus\\_Dataset"
[8] "D:\\SPLUSWIN\\library\\trellis\\_Data"
```

- It is sometime convenient to organize ones projects by duplicate S-PLUS icons and **separate** `_Data` files (`.Data` in UNIX)

Assessing Stored Objects

```
objects()  
  
> objects(pattern="coal*")  
[1] "coal.krige"  "coal.mat"    "coal.mp"  
[4] "coal.nll"   "coal.predict"  
    "coal.signal"  
[7] "coal.var1"  "coalsig.mat"
```

Removing Stored Objects

```
rm(x, y)  
  
remove(objects(patt="coal*"))
```

Customizing Your Environment

```
> options("editor")
options()
$editor:
[1] "notepad"

> options(editor="write")

> options("editor")
$editor:
[1] "write"

> options(editor="notepad")

> options("editor")
$editor:
[1] "notepad"
```

Customizing Startup and Shutdown

`.First` - A function executed from your `_Data` at S-PLUS startup

*** Hence one may have a different `.First` in multiple `_Data`; this may be over ridden by the `S_First` environment variable.**

`.Last` is executed on ones call to `q()`

Data Modes

- **logical** - Binary data mode, with values represented as T or F.
- **numeric** - Numeric data mode includes integer, single precision, and double precision representations of numeric values.
- **complex** - Complex numeric values (real and imaginary parts).
- **character** - Character values represented as strings.

Data Types

- **vector** - A set of elements in a specified order.
- **matrix** - A matrix is a two-dimensional array of elements of the same mode.
- **factor** - A factor is a vector of categorical data.
- **data frame** - A data frame is a two-dimensional array whose columns may represent data of different modes.
- **list** - A list is a set of components that can be any other object type.

Vector Creation Functions.

- **scan - Read values of any mode.**
`scan(), scan("mydata")`
- **c - Combine values of any mode.**
`c(1,2,3)`
- **rep - Repeat values of any mode.**
`rep(1,5)`
- **seq - Generate numeric sequences.**
`seq(1:10:1), 1:10`
- **vector, logical, numeric, complex, character - Initialize appropriate types.**
`vector('numeric',4),
logical(3), numeric(5)`

Matrix Creation Functions.

- **matrix - create matrix of values.**
`matrix(1:6,ncol=3,byrow=T)`

[,1]	[,2]	[,3]	
[1,]	1	2	3
[2,]	4	5	6
- **cbind - Bind together as columns.**
`c(1,2,3)`
`cbind(1:10,rep(c(1,2),c(5,5)))`
- **rbind - Bind together as rows.**
`rbind(sample(1:10,rep=T),rnorm(10))`
- **data.matrix - Covert data frame to matrix.**

Data Frames

- `read.table` - Reads in data from an external file.
- `data.frame` - Binds together S-PLUS objects of various kinds.

Lists

- The components of a list can be objects of any mode and type including other lists.
- Lists are useful for returning values from functions.

```
> x<-5
> list(original=x, square=x^2)
$original:
[1] 5

$square:
[1] 25
```

scan Function

- **This is very useful for reading in vectors or matrices.**

```
mat <-  
matrix(scan("mydata"),ncol=4,byrow  
=T)
```

read.table Function

- **Reads an ascii file and creates a data frame.**
- **Intended for data in tables of rows and columns.**
- **If first line in the file contains column labels and the first columns contain row labels then read.table will convert to a data frame naturally.**
- **Field separator is white space.**
- **Treats characters as factors.**

Importing Data

- dBase II, III, and IV
- FoxPro
- Clipper
- Xbase
- Lotus 1-2-3
- Lotus Symphony
- Excel
- Any ODBC Compliant Database
- Select Import-External Files from the S_PLUS for Windows File Menu.

data.dump and data.restore

- dump and restore
 - Used for S-PLUS Functions
 - Mostly Readable by Wetware
- data.dump and data.restore
 - Used for S-PLUS Functions and Objects
 - Understandable to data.restore only

Arithmetic Operators

- $*$ - Multiply
- $+$ - Add
- $-$ - Subtract
- $/$ - Divide
- $^$ - Exponentiate
- $\%\%$ - Modulus
- $\%/ \%$ - Integer Divide
- $\%*\%$ - Matrix Multiply

N.B. - These are all vectorized.

Comparison Operators

- \neq - Not Equal To
- $<$ - Less Than
- \leq - Less Than or Equal to
- \equiv - Equal
- $>$ - Greater Than
- \geq - Greater Than or Equal to

Logical Operators

- **!** - Not
- **|** - Or (For Calculating Vectors and Arrays of Logicals)
- **||** - Sequential or (for Evaluating Conditionals)
- **&** - And (For Calculating Vecros and Arrays of Logicals)
- **&&** - Sequential And (For Evaluating Conditionals)

Mathematical Functions

- **abs** - Absolute Value
- **acos, asin, atan** - Inverse Trig.
- **acosh, asinh, atanh** - Inverse Hyper. Trig.
- **ceiling** - Next Larger Integer
- **floor** - Next Smallest Int.
- **cos, sin, tan** - Trig. Functions
- **exp** - e^x
- **log** - Natural Logarithm
- **log10** - Log Base 10.
- **max** - Maximum
- **min** - Minimum
- **sqrt** - Square Root

Statistical Summary Functions

- `all`- Logical Product
- `any`- Logical Sum
- `length`- Length of Object
- `max`- Maximum Value
- `mean`- Arithmetic Mean
- `median`- Median
- `min`- Minimum Value mode
- Data Mode
- `prod`- Product of Values
- `quantile`- Empirical Quantiles
- `sum`- Sum
- `var`- Square Root
- `cor`- Correlation Between
Matrices or Vectors

Sorting Functions

- `rev`- Put Values of Vectors in Reverse Order
- `sort`- Sort Values of Vector
- `order`- Permutation of Elements to Produce Sorted Order
- `rank`- Ranks of Values in Vector
- `match`- Detect Occurences in a Vector
- `cumsum`- Cummulative Sums of Values in Vector
- `cumprod`- Cumulative Products

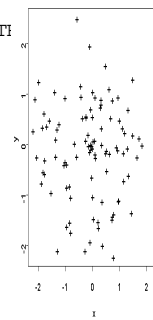
Writing Free-format Files

- `write`
 - Allows one to specify the number of columns
 - Don't forget to use `t = transpose` function and specify number of columns consistent with your original data (default is to write column by column)
- `cat`
 - Less useful than `write`
- `write.table`
 - data exporting utilities under the windows file structure
- `data.dump` (preferable method), `dump`

High-Level Graphics Functions

- `win.graph()`, `motif()`,
`trellis.device(motif)`
- All Examples of Calls to Launch Graphics Window
- Trellis Call Needs to Be Made Aft

```
library(trellis, first=T)  
> win.graph()  
> x<-rnorm(100)  
> y<-rnorm(100)  
> plot(x,y,pch="+")
```



Univariate Plotting Functions

- `barplot`- Creates a Bar Plot
- `boxplot`- Creates Side-by-Side Boxplots
- `hist`- Creates a Histogram
- `dotchart`- Creates a Dot Chart
- `pie`- Creates a Pie Chart
- **Note** - These commands along with the commands on the next several slides are all high-level graphics calls.

Bivariate Plotting Functions

- `plot`- Creates a Scatter
- `barplot`- Creates a simple barplot
- `boxplot`- Creates side-by-side box plots
- `qqnorm`- Plot quantile-quantile plot for one sample against standard normal
- `qqplot`- Plot quantile-quantile plot for two samples

Three-Dimensional Plotting Function

- `contour`- Creates a contour plot
- `persp`- Creates a perspective or mesh plot
- `image`- Creates an image plot

Multivariate Plotting Function

- `faces`- Creates a contour plot
- `matplot`- Creates a perspective or mesh plot
- `pairs`- Creates an image plot
- `stars`- Starplots.
- `symbols`- Plot symbols.

Dynamic Graphics

- `brush`- Interactive scatter plot matrix with the ability to interact with data
- `spin`- Creates three-dimensional point cloud with ability to spin data

The `par` function

- `par`
 - Returns current setting on the graphics parameters
- To save the current graphics settings
`oldsettings<-par()`
- 4 categories of graphics parameters
 - High-level graphics parameters
 - Control appearance of the plot region
 - Only used as arguments to high-level plotting functions

Graphics Parameter Categories

- **High-level graphics parameters**
 - Control appearance of the plot region
 - Only used as arguments to high-level plotting functions
- **Layout graphics parameters**
 - Control the page layout
 - Only set with the par function
- **General graphics parameters**
 - Set with either call to par or to plotting function
 - When set with par they are set for the current graphics device
- **Information graphics parameters**
 - Can't be set by user, but can be queried by par

Multiple Plots Per Page

- `par(mfrow=c(2,2))`
 - **This specifies two rows and two columns of plots**
- `par(mfrow=c(1,1))`
 - **Back to the normal arrangement**
- `plot(x,y,pch="+")`
 - **Override the default plotting symbol**

Adding to Plots

- **You can continue to add to plots until you call another high-level plotting function or `frame()`**
- **We may use low level plot functions to add things to plots**
 - `lines`
 - `points`
- **Here is a useful trick**
`plot(rx,ry,type="n")`

Printing Graphics

- `File-Print Menu`
- **Starting Printing Graphics Device**
 - `win.printer`
 - `postscript`
 - `hpgl`
- **In UNIX one may print to a dummy printer and then use the file that the printer prints to**

Capturing Graphics to the Windows Clipboard

```
win.printer(format = "metafile+,  
file="clipboard")  
  
plot(x,y,pch="*")  
  
dev.off()
```

Probability Distribution Functions

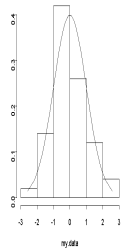
- `ddistname(x)` - Computes density values
- `pdistname(q)` - Computes probabilities from the cdf
- `qdistname(p)` - Computes quantiles from the inverse cdf
- `rdistname(n)` - Generates random numbers

Some Common Probability Distributions

- binom
 - size, prob
- norm
 - mean, sd
- unif
 - min, max

Example Probability Plot

```
> win.graph()  
> my.data<-rnorm(100)  
> hist(my.data,den=-1,prob=T)  
> p<-ppoints(100)  
>  
> lines(my.data,denorm(qnorm(p))),dnorm(qnorm(p)))
```



Statistical Summary Functions

- `mean`- Arithmetic mean
- `median`- Median
- `var`- Variance of vector, covariance matrix of matrix
- `cor`- Correlation between matrices or vectors
- `quantile`- Empirical quantiles
- `summary`- Summary statistics of vector or other object

Statistical Tests

```
> set.seed(19)
> x<-rnorm(10)
> y<-rnorm(5,mean=1)
> t.test(x,y)

Standard Two-Sample t-Test

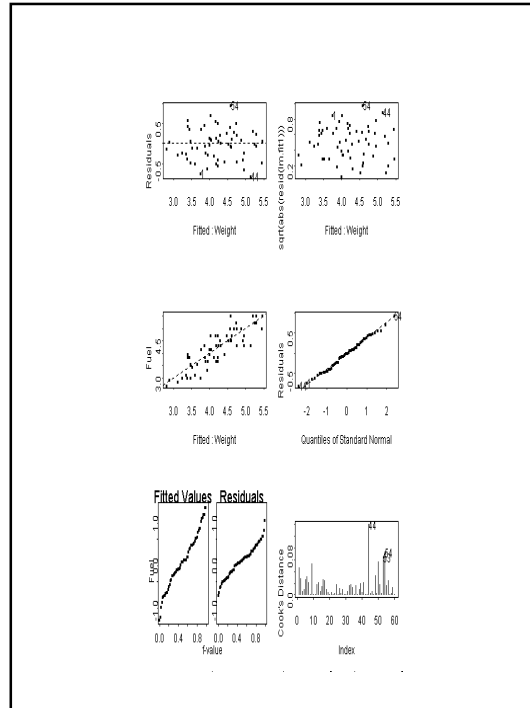
data:  x and y
t = -1.4312, df = 13, p-value = 0.176
alternative hypothesis: true
difference in means is not e
qual to 0
95 percent confidence interval:
-1.7254080  0.3502894
sample estimates:
mean of x mean of y
-0.4269014  0.2606579
```

Regression

- `lm`- Regression Command
- Some examples of formula specification
 - `Yield ~ Temp + Conc`
 - Yield is modeled as depending on Temp and Conc
 - `Yield ~ Temp + Conc + Temp:Conc`

Example Regression

```
> lm.fit1<-lm(Fuel ~ Weight)
> par(mfrow=c(3,2))
> plot(lm.fit1)
```



Writing S-Plus Functions

- You can write your own functions or modifying existing functions

```
> name<-function(arguments){body}
```

- Default values may be specified with an equal sign and a value
- Objects defined within the body are local to the function

Example Function

- `cube <- function(x) {return (x^3)}`
- `cube(3)`
- The function may be defined in a separate file which is then sourced
- Functions may have multiple inputs and return multiple outputs via the list construct.
- Insert a `browser()` command in your function to help examine the contents of the function
 - type a 0 to exit the browser

Iteration and Flow of Control

- **Conditional Statements**
`if (cond) {body}`
- **for and while loops allowed (**but to be avoided if possible**)**
`for(name in vlaues) {body}`

Apply and Outer

- **To perform calculations on each row or column of a matrix use `apply`**

```
apply(mymatrix,2,means)
# Computes column means or mymatrix
```

- **To perform the outer product of two vectors (or matrices)**

- **Useful for computing a function over a grid of values**

```
> surf <- function(x,y) {cos(x) +
  sin(y)}
> x<-seq(-2*pi, 2*pi,len=40)
> y<- x
> z<-outer(x,y,surf)
> persp(x,y,z)
```