Graphics

Graphics

S has extensive graphics facilities. Some examples of high-level graphics functions are:

- Pie Charts, Bar Charts and Histograms
- Box-and-Whisker Plots
- Scatter plots
- Time Series Plots
- Surface Plots

The quality of the graphs produced by S is often cited as a major reason for using it in preference to other statistical software systems.

Low-Level Graphics

The high-level graphics facilities in S are built on a set of flexible low-level ones. This makes it possible to modify existing plot types or to implement completely new ones.

The low level facilities include:

- Page Layout
- Setup of Plotting Coordinates
- Drawing Points and Lines
- Drawing Polygons and Rectangles
- Color Management (R rather than S)

Devices and Device Drivers

In S all drawing operations are carried out by a *device driver*. This makes it possible to abstract the graphics system away from the capabilities of particular hardware devices and provides graphics functionality which works on all devices.

The most common "devices" are:

- The X Window System
- Microsoft Windows (Screen and Metafiles)
- Postscript Printers

For S to create a graph, it musr be told which kind of device the graph is intended for. This is done by calling a function which starts a device driver.

Devices and Device Drivers

Under Splus, an on-screen *graphics window*, use the the **`motif()`** function. Under R, use the **`X11()`** function.

Under either system, this is the device driver which is used automagically if you don't specify a device driver.

There are many optional arguments to device drivers. Look at the manual entries to find out what is approriate for each driver.

Postscript Graphics for LaTeX

To use S graphics in LaTeX, you will need to produce them with the **`postscript`** device driver.

The typical steps in producing a graph are:

```
> postscript(file="fig.ps",
+            width=6, height=5,
+            horizontal=FALSE)
```

graphics commands go here

```
> dev.off()
```

You can also adjust the size of font used to annotate the plots by using an optional **`\pointsize=`** argument.

Postscript Graphics and LaTeX

Once you have produced a graph in S, you can include it in a LaTeX document as follows.

```
\usepackage{graphicsx}   % in the preamble
\includegraphics{fig.ps} % insert the graph
```

You can modify the plot by using optional arguments to **`includegraphics`**. For example:

```
\includegraphics[width=\textwidth]{fig.ps}
\includegraphics[width=0.9\textwidth]{fig.ps}
\includegraphics[width=8cm]{fig.ps}
\includegraphics[width=8cm,angle=45]{fig.ps}
```

See the graphics bundle documentation for the full details.

Advice on Plot Sizes

In general you should aim to set the size of your graphs to exactly the size they are to be on the LaTeX page and use **`\includegraphics`** without any scaling options. This will make the font sizes and line thicknesses consistent across all the graphs in your documents.

An alternative strategy is to create the graphs twice as big as they need to be and to scale them down by 50% You can scale the graphs as follows:

```
\includegraphics[scale=.5]{fig.ps}
```

If you use this technique, remember to make the fonts twice as large too.

Example

Here is how a very simple scatterplot of some random numbers can be produced.

```
> x <- rnorm(100)
> y <- 1 + x/2 + rnorm(100)
> plot(x, y)
```

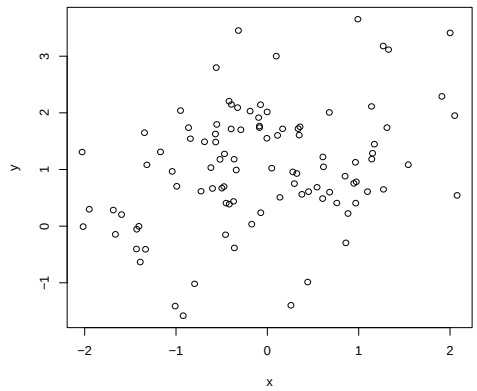
(This is a plot showing a sample from a bivariate normal distribution with correlated variables.)

Producing Graphics Interactively

The typical way of producing a graph using S is to get it right on the display screen first, and then to run the same instructions with the **`postscript`** driver running.

The easiest way to do this is to type the graphics commands into a text editor first, and then to copy-and-paste into the S interpreter. This makes it easy to fix bugs as well.

We will see that XEmacs provides additional support for this way of working.



XFig Graphics

R (but not Splus) has a device driver for the xfig graphics editor. The interface to the **`xfig`** driver is quite similar to the **`postscript`** one.

```
> xfig(file="plot.fig",
+      width=6, height=5,
+      horizontal=FALSE)
```

graphics commands go here

```
> dev.off()
```

You can now use xfig to edit the plot and add annotation, change colors etc.

Annotation

Axis labelling and an overall title can be added with additional arguments to the plot call.

```
> plot(x, y, main="Main Title",
+            xlab="X Label",
+            ylab="Y Label")
```

Alternatively, you can use blank row and column labels in the call to **`plot`** and use the **`title`** function to add them later.

```
> plot(x, y, xlab="", ylab="")
> title(main="Main Title", xlab="X Label",
+       ylab="Y Label")
```

Basic Graphics

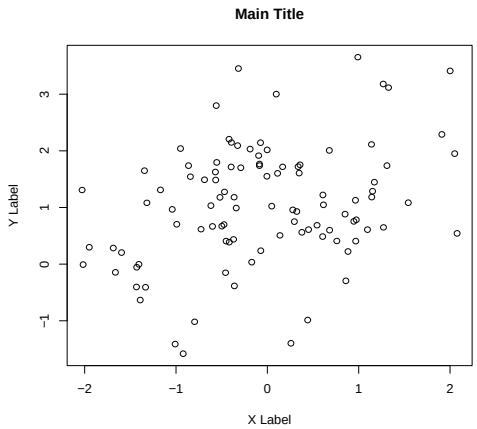
The principal graphics function is **`plot`**. The simplest call to it has the form:

```
> plot(x, y)
```

This produces a scatterplot with points at the coordinates passed as arguments. Extra control is possible through additional optional arguments. The call

```
> plot(x, y, xlim=xlims, ylim=ylims)
```

Changes the limits on the graph to those specified in the **`xlim`** and **`ylim`** arguments.



Plot Type

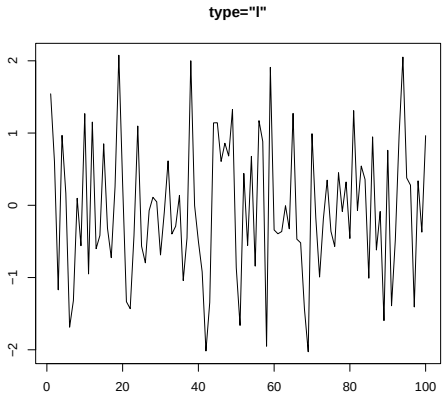
The **`type=`** argument to the **`plot`** function sets the plot type. Some possibilities are:

`"n"`	nothing
`"p"`	points
`"l"`	connected lines
`"h"`	high density “needles”
`"s"` & **`"S"`**	step functions

Adding to Plots

Once a plot has been created (with **`plot`**), you can add additional elements to it with a variety of functions.

`points`	add additional points
`lines`	add connected line segments
`segments`	add disconnected line segments
`arrows`	add arrows
`abline`	add a line in slope/intercept form
`polygon`	add polygons
`symbols`	add symbols to a plot



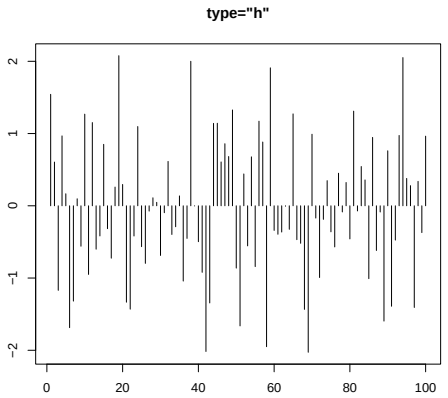
Example

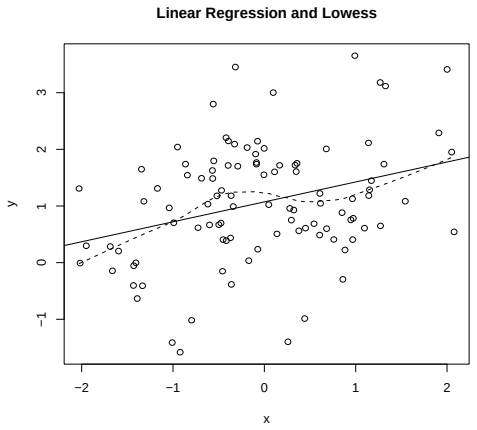
A typical example of a simple plot with added elements is a scatter plot with added regression lines (linear or smooth).

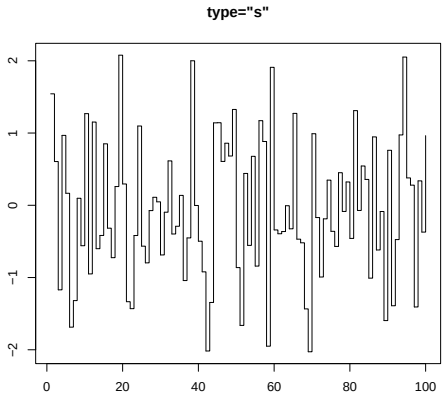
```
> plot(x, y)
> abline(lm(y~x))
> lines(lowess(x, y), lty=2)
> title("Linear Regression and Lowess")
```

Here, the call to **`abline`** adds the least-squares regression line to the plot. The **`lines`** call adds a lowess smooth to the plot.

The additional argument **`lty=2`** ensures that the second line is drawn with a different line texture.



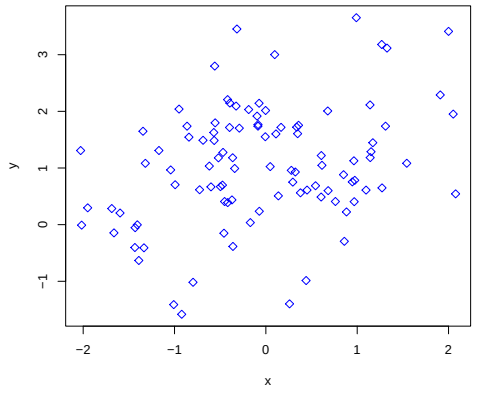




Control of Plotting Symbols and Color

- The arguments **`pch=`** and **`col=`** can be used to control the choice of symbol and colors for points plotted in graphs.
- In both Splus and R small integer values will produce different symbols and colors for the points plotted in a graph.
- Splus ignores all but the first values in the arguments, but R allows vectors of symbols and colors to be used.

```
> plot(x, y, pch=5, col=4)
```



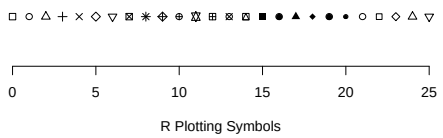
Color Control In R

R has a rather more sophisticated color model than S. For campatility with S, colors can be specified by small integer values, but they can also be specified by name, or by specifying the amount of Red, Green and Blue which are mixed to produce the color.

Some valid color specifications in R are:

`col=2`	this is Red
`col="purple"`	this Purple
`col="#FFFF00"`	this is Yellow
`col=rgb(1,0,1)`	this is Magenta
`col=hsv(1/12,1,1)`	this is Orange(ish)

Better ways of specifying color are a current research topic.



Line Texture Control In R

- In R, the linetype values are as follows: 0=blank, 1=solid, 2=dashed, 3=dotted, 4=dotdash, 5=longdash, 6=twodash.
- As well as specifying line textures by index, they can also be specified by name – one of the strings **`"blank"`**, **`"solid"`**, **`"dashed"`**, **`"dotted"`**, **`"dotdash"`**, **`"longdash"`** and **`"twodash"`**).
- Note that in S, linetypes are device dependent.

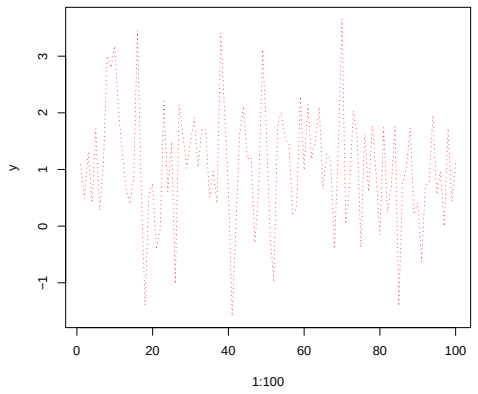
Control of Line Texture and Color

- The arguments **`lty=`** and **`col=`** can be used to control the choice of symbol and colors for lines plotted in graphs.
- In both Splus and R small integer values will produce different textures and colors for the lines plotted in a graph.
- Again, Splus ignores all but the first values in the arguments, but R allows vectors of linetypes and colors (where this makes sense).

```
> plot(1:100, y, type="l", lty=3, col=2)
```

Direct Line Texture Control

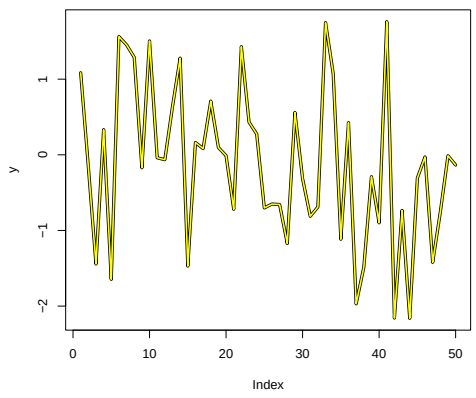
- The texture can also be specified in terms of dash length. For example, the texture **`"3212"`** represents "on" for 3 points, "off" for 2 points, "on" for 1 point and "off" for 2.
- "Points" are a typesetting measure — there are 72 points to the inch.
- The texture string can be up to 8 characters long, each character being one of the hexadecimal digits **`1`**,. . . ,**`F`**.



Line Width Control

- The widths of lines drawn by R and S can be controled with a parameter called **`lwd`**.
- This parameter tends not to be used much, but it can provide some very useful effects.

```
> y <- rnorm(50)
> plot(y, type="l", lwd=4)
> lines(y, lwd=2, col="yellow")
```


y
1
0
−1
−2
0
10
20
30
40
50
Index