Charles Fleming

Simple Graphs in R

Contents

Introduction

Manipulating vectors, matrices, and data frames, in order to analyze data, lies at the heart of R. The R project is a beneficiary of the great strides which have occurred in numerical computations ever since the advent of the digital electronic computer. Having easy access to computational resources like R, the author of statistical analyses is able to include into his report without much effort and expense pictures of data and graphs of functions. They are indispensable devices when writing a report for describing data and explaining theory. The graphical capabilities of R are versatile; graphs are easy to make. The painstaking etching of limestone slabs which required many hours of labor for lithographers to accomplish has been eliminated because graphs and figures with the highest quality for immediate publication can be made by authors with free statistical and mathematical typesetting computer programs. It is by virtue of the licensing of R under the General Public License (GPL) which has made it possible to obtain this free and reliable statistical computer program which enjoys active development from all corners of the world. Here, we will learn, by beginning with constructing simple graphs, how to create a complex graphical image like the one shown in Figure 1. Whenever a command is enclosed in a rectangle, a situation which occurs very often in these notes, the command is meant to be executed by the reader as if it belongs to a tutorial. The semi-colon is used to separate distinct commands when it is deemed convenient to write the more than one command on the same line, and the # symbol marks the beginning of a comment.

Washington, D.C.
February 2005

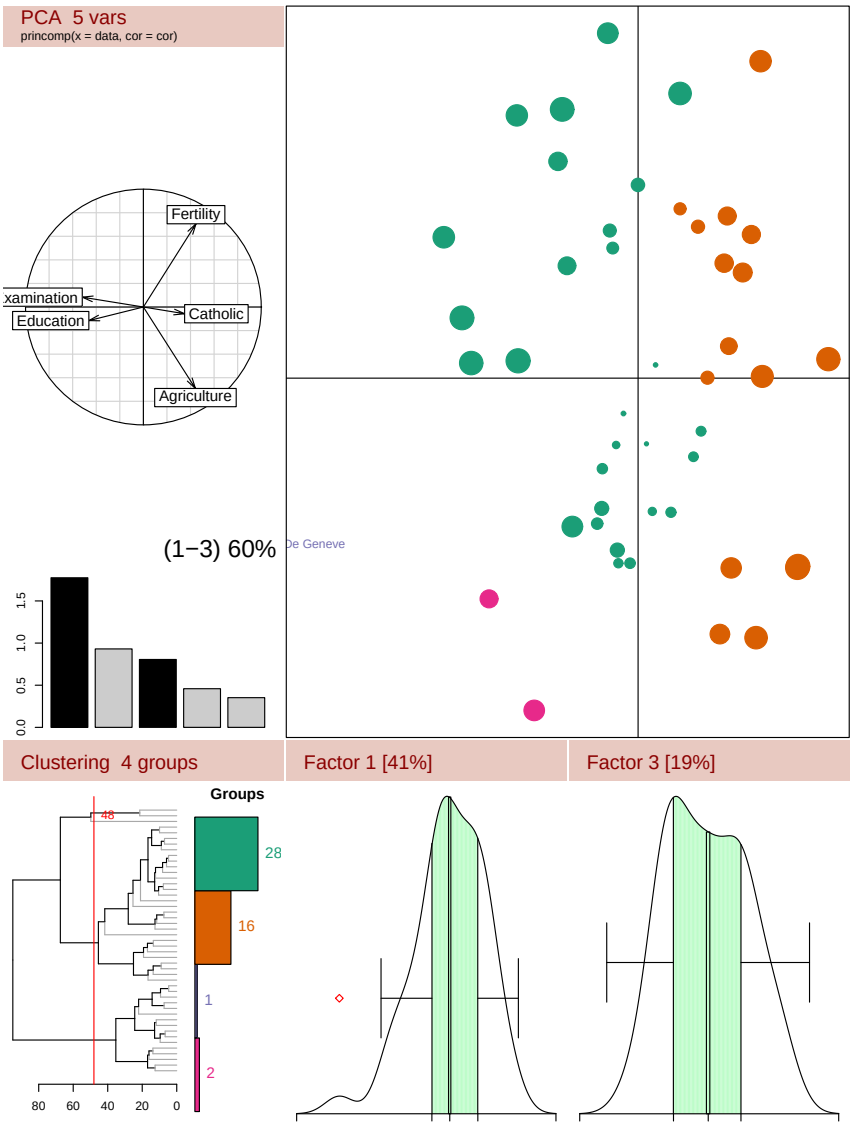


Figure 1: This image can be found at http:www.r-project.org and was created by Eric Lecoutre, Universite catholique de Louvain, Belgium

Chapter 1

Graphs of Functions

1.1 Curves

A graph of a mathematical function may be created in R. A simple graph to make is the one of the mathematical function `sin(x)`:

```
curve(sin(x),-2*pi,2*pi)
```

It is plotted from -2π to 2π. To embellish the picture with a title, labeled axes, and colored lines of various styles, options of `curve` are used.

```
curve(sin(x),-2*pi,2*pi,main="Sine and Cosine Curves",xlab="Time",
ylab="Amplitude",col="red",lty=5)
abline(h=0,col="blue",lty=2)
```

If `xlab=""` and `ylab=""`, then the axes are not labeled, and if `xaxt="n"` and `yaxt="n"`, then the axes are not printed. This was done in order to superimpose two graphs on each other. In the next set of instructions, the cosine curve is superimposed on the sine curve by using `par(new=TRUE)`. The command, `abline` with `h=0` adds a straight horizontal line.

```
curve(sin(x),-2*pi,2*pi,main="Sine and Cosine Curves",xlab="Time",
ylab="Amplitude",col="red",lty=5)
par(new=TRUE)
curve(cos(x),-2*pi,2*pi,xlab="",ylab="",yaxt="n",xaxt="n",
col="green",lty=5)
abline(h=0,col="blue",lty=2)
```

The syntax of `curve` allows for the specification of the domain of `sin(x)`. The color of the horizontal line is specified by `col="blue"` and the dotted style of the horizontal line is specified by `lty=2`.

A more complete description of the options for use in `curve` appears in `?curve`, `?plot`, and in `?par`. The command, `par`, does not produce statistics or a graph. It sets the graphical

parameters. Graphical parameters may be specified within a plotting function as was done in making a picture of the sine function with `curve`. The other way of setting a graphical parameter is by means of the command, `par`. When a graphical parameter is set by means of `par`, it is used henceforth for the duration of the current session of R, unless it is superseded by another use of `par`.

In the description of `par`, there is an option which specifies the seven styles of lines, `lty`:

Table 1.1: Styles of Lines

0	blank
1	solid
2	dashed
3	dotted
4	dot dash
5	long dash
6	two dash

The options, `xlab` and `ylab`, allows for the arbitrary use of labels for the x and y axes. The use of `xaxt="n"` specifies that the x-axis must not be plotted. In the example of drawing a picture of the cosine function, `xlab=""` and `xaxt="n"` cause no labelling of the x-axis and no use of a marked scale. Another option which is worth considering is the option `type="n"`. It suppresses the image of the graphs, so that, only the title and axes are visible Sometimes such a blank is useful on occasions of writing a examination in which the students are asked to plot data on a prescribed template.

The first instance of `curve` will produce the title, labeling of the axis, and the image of the first figure, while the second graph will be superimposed on the first. The superimposition does not occur automatically. Every time a plotting function like `plot`, `curve`, and `matplot` is used, R erases any previous vestige of a plot and starts with a fresh plot. In order to superimpose two images on the same plot, the command `par(new=TRUE)` must be inserted in between the two plotting functions as was done above for superimposing a cosine plot onto a sine plot.

Another and better way to superimpose two graphs is with the use of the option, `add=T`:

```
curve(sin(x),-2*pi,2*pi,main="Sine and Cosine Curves",xlab="Time",
ylab="Amplitude",col="red",lty=5)
curve(cos(x),-2*pi,2*pi,ylab="",yaxt="n",xaxt="n",
col="green",lty=5,add=TRUE)
abline(h=0,col="blue",lty=2)
```

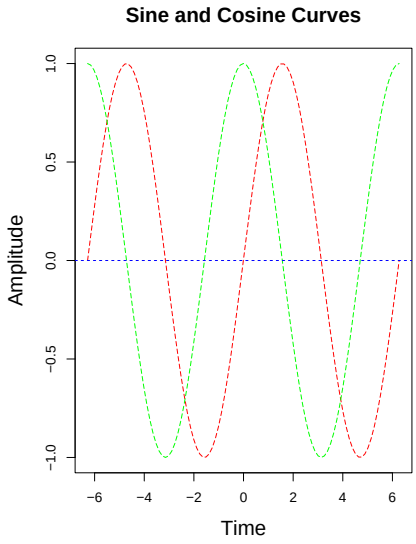


Figure 1.1:

1.2 Surfaces

Engineers might like to make sine and cosine curves, but a statistician would be more interested in drawing a response surface which he might use in a report or in a classroom. The response surface which appears in Figure 1.2 and is derived from the linear model, $y_i = \beta_0 + \beta_1 x_i + \epsilon_i$ where $\epsilon_i \sim N(0, \sigma^2)$ was drawn by the following set of commands.

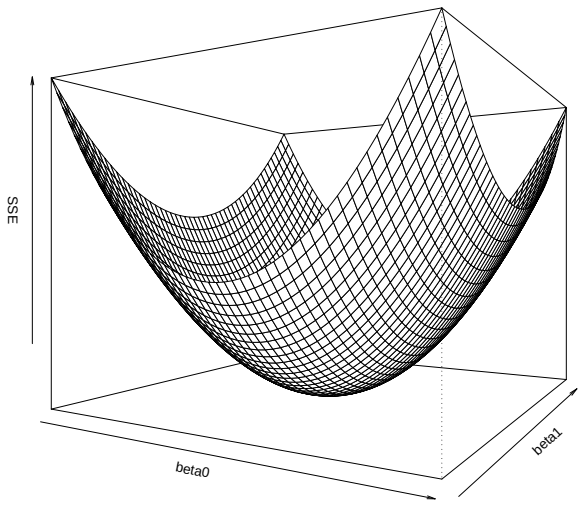


Figure 1.2:

```
sse<-function(beta0,beta1){
x1<- -1
y1<- -1
x2<-0
y2<-0
x3<-1
y3<-1
z<-(y1-beta0-beta1*x1)^2+(y2-beta0-beta1*x2)^2+(y3-beta0-beta1*x3)^2
return(z)
}
beta0<-seq(-1,1,length=50)
beta1<-seq(0,2,length=50)
z<-outer(beta0,beta1,sse)
par(cex.main=3,cex.lab=2,cex=1)
persp(beta0, beta1, z,d=1, theta = 30, phi = 0, expand = .8, col = "lightgray",
xlab="beta0",ylab="beta1",zlab="SSE",main="Simple Response Surface")
```

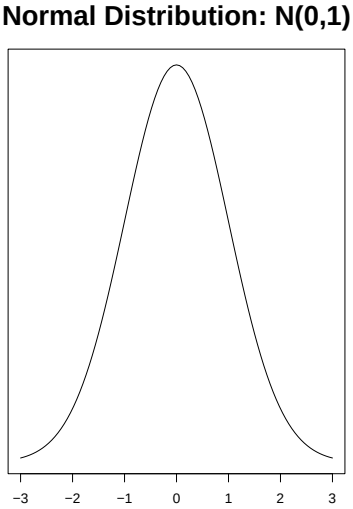


Figure 1.3:

The command which creates the surface is the command `persp()`. It has several interesting options like `expand` and the ones for specifying the angles of perspective, `theta` and `phi`. The response surface shows the case when the set of data consists of the three co-linear points, `(-1,-1)`,`(0,0)`, and `(1,1)`. The response surface shown in Figure 1.2 is the representation of the function, $SSE = \sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_i)^2$. The values of `x` and `y` of these co-linear points are written in the body of the function `sse()` as shown above. The variables of SSE are β_0 and β_1. As in the case of drawing a picture of the sine and cosine functions, numerical values of the variables must first be created. To that end, creating the `beta0` and `beta1` coordinates uses the command for creating sequences. By trial and error, the vectors `beta0` and `beta1` which span a two unit intervals using 50 sub-intervals were chosen because they produced a graph of the response surface that looked good, wherea, the values for the coordinate, `z`, are stored in a square matrix. It is understood that the horizontal aspect of the square matrix corresponds to the `x` coordinate and the vertical aspect corresponds to the `y` coordinate as if by analogy the square matrix were a multiplication table. The command, `outer(beta0,beta1,sse)`, performs the matrix multiplication of a $n \times 1$ vector by a $1 \times n$ vector to produce a $n \times n$ matrix having elements which produced by the evaluation of the function, `sse(beta0,beta1)`.

According to the manual page for `plotmath` which is given verbatim in Appendix I,

> "Expressions can also be used for titles, subtitles and x- and y-axis labels (but not for axis labels on `persp` plots)."

Consequently, Greek letters cannot be displayed in the axes of `persp`, even thought they can be displayed in its title.

1.3 Annotation

Another curve which a statistician is inclined to include in a report is the graph of a normal distribution like the one shown in Figure 1.3. It was produced by following the same logic which ordered the commands that produced the picture of the sine curve.

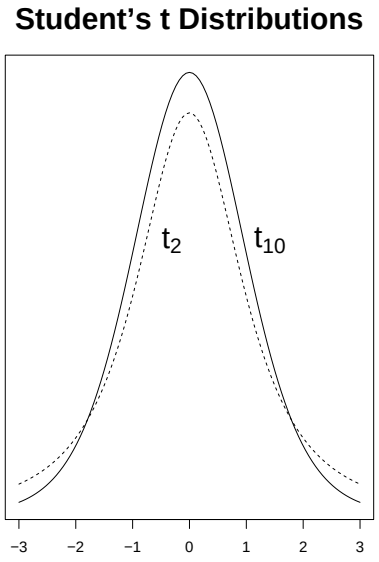


Figure 1.4:

```
par(cex.main=1.15,cex.lab=1.25,cex=1.5)
curve(dnorm(x,mean=0,sd=1),-3,3,yaxt="n",xlab=" ", ylab=" ",
main="Normal Distribution: N(0,1)")
```

A picture of Student's t distribution can be similarly created, but unlike any Normal distribution which can be transformed to the standard N(0,1), there are as many t distributions as there are degrees of freedom. In a picture of several superimposed t distributions, there would be a need to annotate the picture appropriately to identify a curve with its degree of freedom.

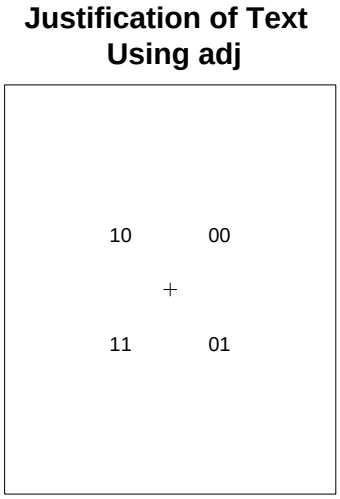


Figure 1.5:

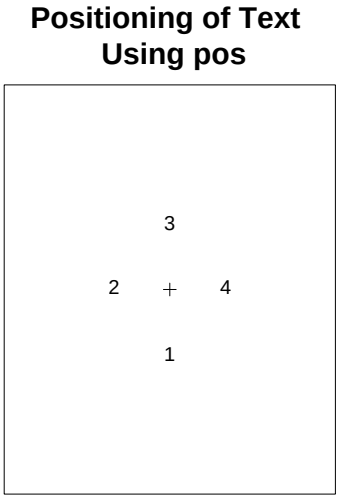


Figure 1.6:

```
par(cex.main=1.15,cex.lab=1.25,cex=1.5)
curve(dt(x,10),-3,3,yaxt="n",ylab=" ",xlab="",lty=1,
main="Student's t Distributions")
curve(dt(x,2),-3,3,yaxt="n",ylab=" ",lty=2,add=TRUE)
par(cex.main=1,cex.lab=1,cex=2)
t1.str<-expression(paste(t[10]))
text(1,dt(1,10),t1.str,adj=c(-.25,0))
t2.str<-expression(paste(t[2]))
text(-.75,dt(1,10),t2.str,adj=c(-.75,0))
```

In this example of annotating a graph as shown in Figure 1.4, a title appears on the graph, but there are different styles of lines and text has been placed at certain specified positions in the graph. The curve with the solid line corresponds to the Student's t distribution with ten degrees of freedom. To place t_{10} near the solid line, the `text` command is used in which the first entry is the x coordinate of the position, and the second entry is the y coordinate of the position. In the above example, `text(1,dt(1,10),t1.str,adj=c(0,0))`specifies that the text which is contained in the object, `t1.str`, shall be placed at the coordinates `(1,dt(1,10))` where `dt(1,10)` is the ordinate of the Student's t distribution with 10 degrees of freedom at $x = 1$. The option, `adj=c(0,0)`, specifies the way in which the text is justified relative to the coordinates of the text. A schematic depiction of the effects of `adj` on the justification of the text is shown in Figure 1.5 and described in Table 1.2. In Figure 1.5, the cross hairs mark the location of the x and y coordinates of the text, and the locations of `00`, `01`, `10`, and `11` show the effects of using `adj`.

0	0	North East
0	1	South East
1	0	North West
1	1	South West

Table 1.2: Justification of Text

1	Bottom
2	Left
3	Top
4	Right

Table 1.3: Position of Text

A similar option to `adj` for adjusting the justification of a text about its position is the option to position the text strictly in the vertical or horizontal direction. It is denoted by `pos`, and the effects of its options appear in Figure 1.6. This option will be used in the next section.

1.4 Polygons

Another indispensable embellishment of a picture is the one of shading the interior of a curve. The shading is accomplished by means of polygons.

```
par(cex.main=1,cex.lab=1,cex=2)
stitle<-expression(paste("Meaning of the F Quantile, ",
F[nu[1]][",][nu[2]][";][alpha]))
curve(df(x,df1=3,df2=5),0,5,yaxt="n",xlab=" ",ylab=" ",
xaxt="n",main=stitle)
axis(1,at=c(0), labels="0")
par(cex.main=1,cex.lab=1,cex=3)
axis(1,at=c(1.7), labels=expression(atop("|",
F[nu[1]][",][nu[2]][";][alpha])))
lines(cbind(1.7,1.7),cbind(0,df(1.7,df1=3,df2=5)),lty=5)
x0<-seq(1.7,5,.4)
y0<-df(x0,df1=3,df2=5)
xx<-c(x0,rev(x0))
yy<-c(y0,matrix(0,1,length(x0)))
polygon(xx,yy,col="gray", border = "red")
text(2.25,.02,expression(alpha),adj=c(0,0))
```

The set of commands which were used to create Figure 1.7 contains several new commands: `expression`, `axis`, `paste`, `lines`, `rev`, `polygon`, `atop`, and some Greek letters. The one which will be discussed here will be `polygon`.

By trial and error in making several plots of the F distribution, the under the curve for the interval `(1.7.5)`, produced a pleasing graph. In that interval, the area under the curve is supposed to be shaded gray in order to emphasize the meaning of a quantile. To the end of using the `polygon` command to do the shading, it is necessary to supply the x and y coordinates. In the

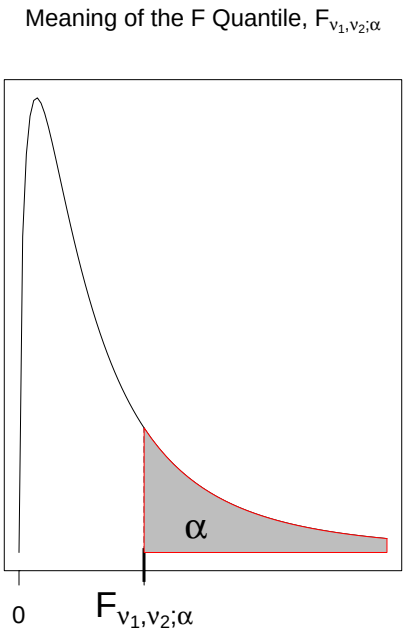


Figure 1.7:

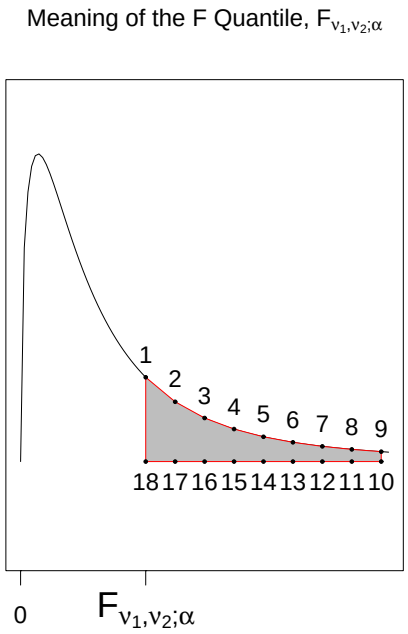


Figure 1.8:

example given above, the x and y coordinates for the `polygon` command must be chosen in such a way that the coordinates follow a path around the perimeter of the shaded area as depicted in Figure 1.8. The first element of `xx` and the first element of `yy` are the coordinates of the point labeled 1. The second element of `xx` and the second element of `yy` are the coordinates of point 2, and so on until a complete circuit about the shaded area has been made.

The vectors, `xx` and `yy`, were constructed so as to follow the path around the shaded area. A convenient command to manipulate a vector to reverse direction is the command `rev()`. It will reverse the order of the elements of a vector. By concatenating the elements of `x0` which run from 1.7 to 5 by .4 units with a vector of elements which runs in reverse order from 5 to 1.7 by .4, the following command was used: `c(x0,rev(x0))`. The top part of the shaded area when `x0` runs from 1.5 to 5 is the ordinate of the curve which is produced by the F distribution, i.e. `df(x0,df1=3,df2=5)`. The bottom side of the shaded area which corresponds to `x0` from 5 to 1.7 has ordinate 0; therefore, `yy` was created by concatenating `df(x0,df1=3,df2=5)` with a vector of all zeros. When the elements of `xx` and `yy` are combined, they correspond to the coordinates of the points which are depicted in Figure 1.8.

The set of commands which produced Figure 1.8 and which includes the commands written in italics is shown below. The commands which are given in italics print the points and write the

numbers which lie around the perimeter of the shaded area.

```
par(cex.main=1,cex.lab=1,cex=2)
stitle<-expression(paste("Meaning of the F Quantile, ",
F[nu[1]][",][nu[2]][";][alpha]))
curve(df(x,df1=3,df2=5),0,5,yaxt="n",xlab=" ",ylab=" ",
ylim=c(-.2,.8),xaxt="n",main=stitle)
axis(1,at=c(0), labels="0")
par(cex.main=1,cex.lab=1,cex=3)
axis(1,at=c(1.7), labels=expression(atop(" ",F[nu[1]][",][nu[2]][";][alpha])))
x0<-seq(1.7,5,.4)
y0<-df(x0,df1=3,df2=5)
xx<-c(x0,rev(x0))
yy<-c(y0,matrix(0,1,length(x0)))
polygon(xx,yy,col="gray", border = "red")
par(cex.main=1,cex.lab=1,cex=1)
points(xx,yy,pch=20)
par(cex=.8)
for (i in 1:(length(xx)/2)){
text(xx[i],yy[i],i,pos=3)
}
for (i in ((length(xx)+2)/2):length(xx)){
text(xx[i],yy[i],i,pos=1)
}
```

The coordinates for use in `polygon` must follow the right pattern. The vector, `xx`, was divided into two pieces, in order to write numbers directly above a point as specified by the option `pos=3` and directly below a point by `pos=1` according to the top and bottom of the shaded area. Note that `adj=c(.5,.5)` means dead center justification. The command `par(cex=.8)` adjusts the scaling of the written text, and the additional command, `points(xx,yy,pch=20)`, create the points with point symbol set to 20. An array of symbols with identification numbers is given in Figure 1.9.

All of the different symbols which are available in R for use in making graphs are shown in Figure 1.9, and the set of commands with produced Figure 1.9 is given below. The semi-colon is used to separate distinct commands when it is deemed convenient to write the more than one command on the same line.

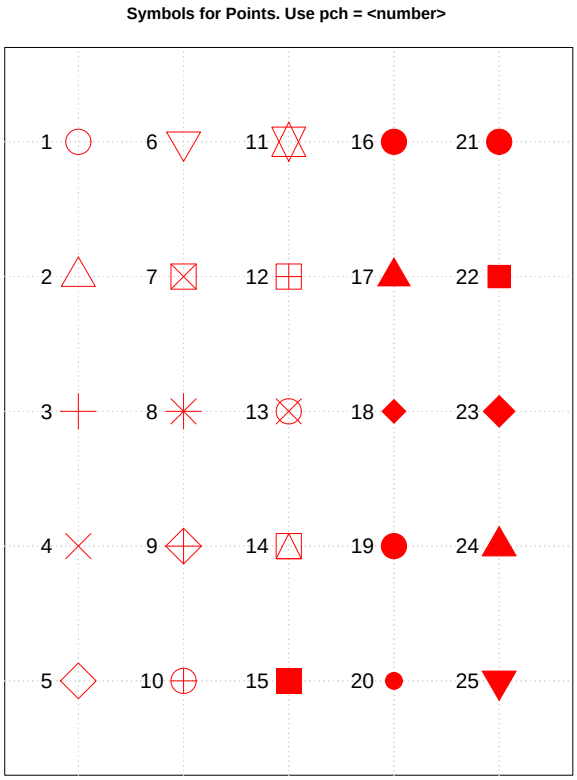


Figure 1.9:

```
ipch <- 1:25; dd <- c(-1,1)/2
rx <- dd+ range(ix <- (ipch-1) %/% 5)
ry <- dd+ range(iy <- 3 + 4-(ipch-1) %% 5)
plot(rx, ry, type="n", axes = F, xlab = "", ylab = "",xaxt="n",yaxt="n",
 main = "Symbols for Points. Use pch = <number> ")
abline(v=ix, h=iy, col = "lightgray", lty = "dotted")
for(i in ipch) { # red symbols with a yellow interior (where available)
 points(ix[i], iy[i], pch=i, col="red", bg="red", cex = 4)
 text (ix[i] - .3, iy[i], i, col="black", cex = 1.5)
 }
```

1.5 Greek Letters and Formulas

A striking feature of Figure 1.7 besides the inclusion of a shaded area is the presence of Greek letters and mathematical notation. The syntax for annotating a graph with mathematical formulae is given in Appendix I. The syntax is not the syntax which is used in LaTeX but it does bear a re-

semblance. For example, `F[nu[1]][",][nu[2]][";][alpha]` is the syntax for creating $F_{\nu_1,\nu_2;\alpha}$ which in LaTeX is written in mathematics mode as `F_{\nu_1,\nu_2;\alpha}`. Greek letters are spelled in English, and square brackets create subscripts. It is, however, necessary to use the command `expression()` when including Greek letters and mathematical symbols in a graph. The statement for creating the object, `stitle`, which contains the text of the title is given below:

```
   stitle<-expression(paste("Meaning of the F Quantile, ",
F[nu[1]][",][nu[2]][";][alpha]))
```

In order to put the mathematical expression, $F_{\nu_1,\nu_2;\alpha}$, into the title, it must be pasted to the English part of the title. An example of the syntax of `paste()` is `paste(a,b)` which will produce `ab`. In an object oriented programming language like R, there are such things as *objects*, *classes*, *methods* where tasks are organized by functions, objects are organized by classes, and functions and classes are brought together by methods. The class, *expression*, contains elements which are expected to be unevaluated. By applying the command, `expression()`, to the result of `paste()`, the English and mathematical formula is treated as an element of the class, expression which in turn is used by `plot()` to annotate the graph with a combination of English and mathematical notation in the title.

1.6 Legend and Identifying Points

In all of the examples discussed thus far, the plots have been of mathematical functions for which `curve` is used. Statisticians like to make pictures of data. By executing `apropos("plot")`, the result will prove that there are many commands with which to make pictures of data in R. Rather than make a graph of a mathematical function, the following examples will make plots of data. Before making pictures of data, we will describe the construction of a legend and the printing of a graph onto paper.

Incorporating a legend into the plot for identifying the two curves seems appropriate, but where should it be placed? The `locator` function will produce the coordinates at that place on the plot where the cursor is placed and the left key of the mouse clicked. Two points will be specified in `locator`. One point will coincide with the upper left corner of the legend, and the second point will coincide with the lower right corner of the legend. By means of the cursor, these two points will be used by R to place the legend of the right size in the right place.

```
legend(locator(n=2),legend=c("Summer","Winter"),col=c("red","green"),
lty="1")
```

.

To see what `locator` produces, execute `locator(n=2)` and click when the cursor is where

the upper left and lower right corners of the legend should be placed. Explicitly, `locator` produces coordinates. The coordinates which `locator` furnishes are automatically utilized by the legend command. Even though, it is convenient to incorporate `locator` directly in the `legend` command, putting the actual coordinates which `locator` provides into the legend manually makes it possible to reproduce the plot with the legend in exactly the same position.

```
legend(c(49.75750,75.37375),c(0.9460465,0.7325581),
legend=c("Sine","Cosine"),col=c("red","green"),lty="1")
```

A written report includes graphics, and unless a graph can be printed on paper it cannot be used in a report. A graph which is created by R can be saved as was done in the preceding example to a file in a Postscript format which is recognized by modern printers. The last command, `dev.off()`, in the last set of instructions terminates the use of the graphics device and causes the image to be sent to the file, `/tmp/CPI.ps`. Rather than create a Postscript file of the image, the graph can be saved in a PNG format by using the command:
`png(file="/tmp/CPI.png", bg="transparent", width=600, height=800)`
in place of the `postscript` command.

```
postscript(horizontal=F,file="/tmp/CPI.ps")
matplot(m,type="l",main="Sine and Cosine Curves",
col=c("red","green"),xlab="Time",ylab="Amplitude")
abline(h=0„col="blue",lty=2)
legend(c(49.75750,75.37375),c(0.9460465,0.7325581),legend=
c("Sine","Cosine"),col=c("red","green"),lty="1")
dev.off()
```

1.7 Plotting Data

One of the most popular forms of presenting data for a statistician is the histogram.

```
w<-c(83,85,74,70,92,64,72,87,88,75)
hist(w)
```

. There are various options in R for producing histograms with different styles. A histogram which displays the relative frequency is produced by: `hist(w,prob=T)` ; with absolute counts by: `hist(w,prob=F)` . The sizes of the bins may be specified by means of the `breaks` option as is done here:

```
br<-seq(40,100,5)
hist(w,breaks=br,prob=T,main="Exam Scores from Watching Videos",xlab="Scores")
```

It is often desired to superimpose a Normal distribution on a histogram by using:

```
curve(dnorm(x,mean=mean(w),sd=sd(w)),add=T)
```

.

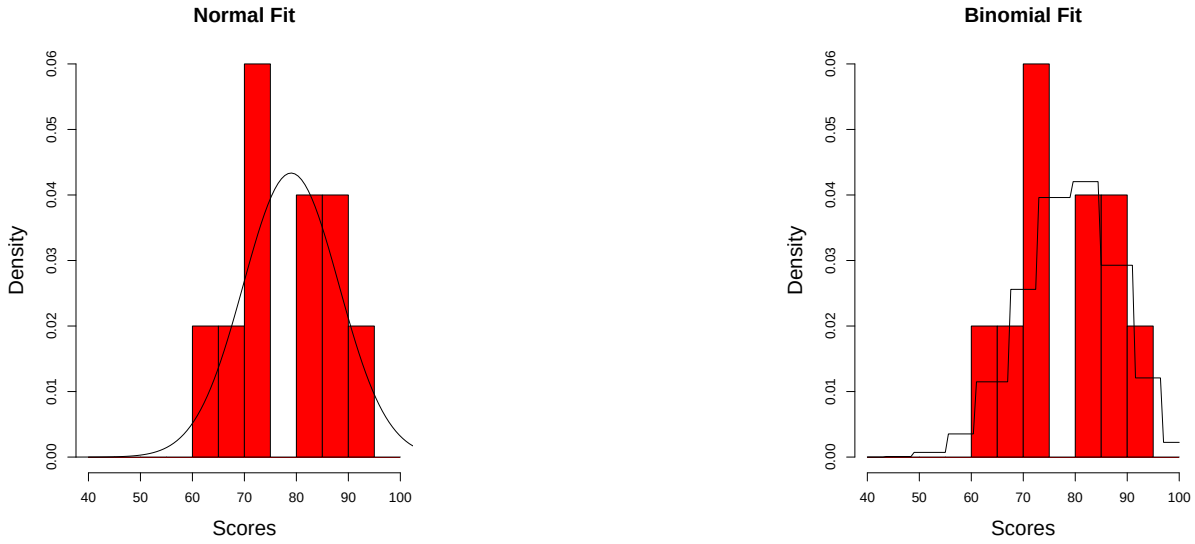


Figure 1.10:

1.8 Side-by-Side

Two graphs can be placed side-by-side by means of using `par(mfrow=c(1,2))`. For example,

```
w<-c(83,85,74,70,92,64,72,87,88,75)
br<-seq(40,100,5)
par(mfrow=c(1,2))
par(cex.lab=1.5, cex.main=1.5,cex=1.5)
hist(w,breaks=br,prob=T,main="Normal Fit", xlab="Scores",col="red")
curve(dnorm(x,mean=mean(w),sd=sd(w)),add=T)
hist(w,breaks=br,prob=T,main="Binomial Fit", xlab="Scores",col="red")
curve(dbinom(round((x-40)/60*length(w)),
length(w),mean((w-40)/60))/6,40,100,add=T)
```

.

The key command for putting two plots side-by-side on the same page is the parameter statement, `par(mfrow=c(1,2))`. To put four plots on the same page, `par(mfrow=c(2,2))` is used. Similarly, to put three columns in two rows of plots on the same page, `par(mfrow=c(2,3))` is used. To reset the frames so that only one plot appears on a page, use `par(mfrow=c(1,1))`

Suppose another set of data besides `w` was obtained and is assigned to the object, `x`.

```
x<-c(95,81,59,68,74,79,72,70,81,58)
```

The set of data contained in `w` and the set of data contained in `x` were obtained in a process

which makes w and x independent sets of data. The set of data in w are scores from an examination in understanding French from students who attend classroom lectures whereas x contains examination scores for proficiency in French from students who also listened to audio tapes of French. We wish to see the data of both:

`plot(w,x)` Some points lie far away from the rest of the data. The command `identify` will allow us to find which points in the data produced the points of interest in the plot.

`identify(w,x)` A more ambitious goal might be to place the names of the points on the plot as a result of identifying some of them as in the following example of identifying four points and saving the resulting image to a Postscript file:

```
w<-c(83,85,74,70,92,64,72,87,88,75)
x<-c(95,81,59,68,74,79,72,70,81,58)
names<-c("A","B","C","D","E","F","G","H","I","J")
par(cex.lab=1.5, cex.main=1.5,cex=1.5)
plot(w,x,main="Scores from Lectures Alone versus Lectures and Audio Tapes",
xlab="Only Lectures", ylab="Both Lectures and Audio Tapes")
identify(w,n=4,x,labels=names,plot=T)
dev.print( postscript, horizontal=FALSE, file="fig8.ps" )
```

After the points have been identified by means of using the cursor, the plot will be saved to `fig8.ps`.

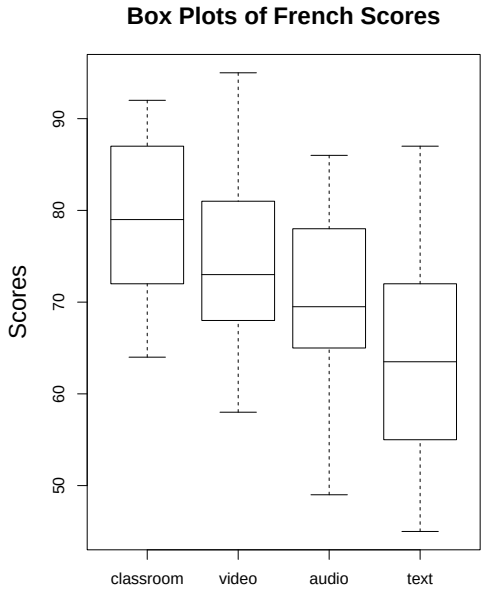
1.9 Box Plots

A single box plot is simple to make. Suppose `w<-c(83,85,74,70,92,64,72,87,88,75)`, then a box plot of this data can be made by: `boxplot(w)`.

A useful aspect of boxplots can been seen when a series of box plots are put side-by-side in the same plot. This arrangement of box plots offers a quick view of the relationship of the sets of data with each other. The following set of commands will create four box plots of the scores in French depending on classroom instruction only given in w, the use of only video tapes given in w, the use of only audio tapes given in y, and the use of only a textbook given in z.

```
w <-c(83,85,74,70,92,64,72,87,88,75)
x <-c(95,81,59,68,74,79,72,70,81,58)
y <-c(86,71,49,63,65,72,78,68,85,65)
z <-c(87,61,45,81,72,67,66,51,55,58)
p<-list(w,x,y,z)
boxplot(p,main="Box Plots of French Scores",
ylab="Scores",xlab="",xaxt="n",horizontal=FALSE)
axis(1,at=c(1,2,3,4), labels=c("classroom","video","audio","text"))
```

The use of the `list` allows the simultaneous plotting of the four box plots in one picture,



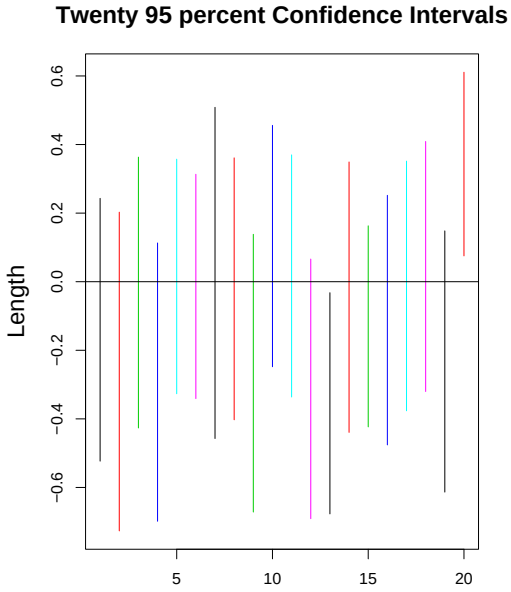
and the use of `axis` puts nice labels on the x-axis at positions 1, 2, 3, and 4, respectively.

1.10 Confidence Intervals

One of the most important concepts in statistics is the confidence interval. For a small enough population, it might be feasible to obtain all the desired information about it, like the mean and the variance. Almost always, there is limited time, and there are insufficient financial resources to examine the entire population. Instead, a sample of the population is usually drawn which, if it is done properly, will represent the population in which case the mean of the sample will be close to the mean of the population, and the variance of the sample will be close to the variance of the population. The statistics which are derived from a sample cannot except in extremely rare events be exactly the same as the corresponding statistics of the population. A good sample, nonetheless, does contain accurate information about the population.

By means of confidence intervals, it is possible to infer some characteristics of the population based on the set of experimental data which was obtained from a sampling of the population. The length of the confidence interval will indicate the precision of the data, and its location will indicate the likely region which contains the parameter of interest of the population. The importance of the confidence interval lies in its use to substantiate an inference about the population.

If a very large number of 95 percent confidence intervals are plotted, then, on the average, 95 percent of them will cover the true population mean. We will use R to produce a picture of twenty 95 percent confidence intervals to illustrate the meaning of confidence intervals.



The example begins by defining a function, `ci`. Every command after `{` and before `}` belongs to the function. A function in R is akin to a sub-routine in FORTRAN or to a module in SAS/IML. A vector of `30*n` random numbers is generated from a standard Normal distribution. The vector, `y`, is converted into a matrix consisting of 30 rows of n columns. The lower limit of the 95% confidence interval is

$$\bar{y} - t_{n-1,\frac{\alpha}{2}} \frac{s}{\sqrt{n}}$$

which will be translated in the R language as:
`mean(y)-qt(.975,length(y)-1)*sqrt(var(y)/length(y))`
The upper limit is the same except that a `+` symbol is used instead of the minus sign.

```
ci<-function(n=20){
y<-matrix(rnorm(30*n,0,1),nrow=30)
lower<-apply(y,2,function(y)(mean(y)-qt(.975,
length(y)-1)*sqrt(var(y)/length(y))))
upper<-apply(y,2,function(y)(mean(y)+qt(.975,
length(y)-1)*sqrt(var(y)/length(y))))
matplot(cbind(lower,upper),type="n",main=
"Twenty 95 percent Confidence Intervals",ylab="Length")
z1<-cbind(1:n,1:n)
z2<-cbind(lower,upper)
matlines(t(z1),t(z2),lty="solid")
abline(h=0)
}
ci()
```

The last command, `ci()`, will execute the function which will produce the 20 confidence intervals.

The trick which R provides is given by the command `apply`. It means that a function is to be applied to each record of a column. That is, `apply(y,2,function(y){...})` will apply the function to every column of `y`. The command, `apply(y,1,function(y){...})`, will cause the function to be applied to every row of `y`. `apply` is a peculiar though very handy command which R inherited from S. There is no corresponding command in FORTRAN or in SAS/IML, like `apply`.

The procedure uses `matplot` to plot the end points of the twenty confidence intervals on the plot. Two vectors, `z1` and `z2`, are created which contain the end points of the twenty confidence intervals, but the end points are made invisible by the option, `type="n"`. The x coordinates of the lower and upper limits are contained in `z1` and the y coordinates for the lower and upper limits are contained in `z2`. The lower and upper limits are connected with a solid line by means of `matlines`. The true population mean is denoted by the horizontal line created by `abline(h=0)`. That 18 out of 20 confidence intervals appear to cover the population mean substantiates the theory that, on the average, 95% of the confidence intervals will contain the population mean.

Chapter 2

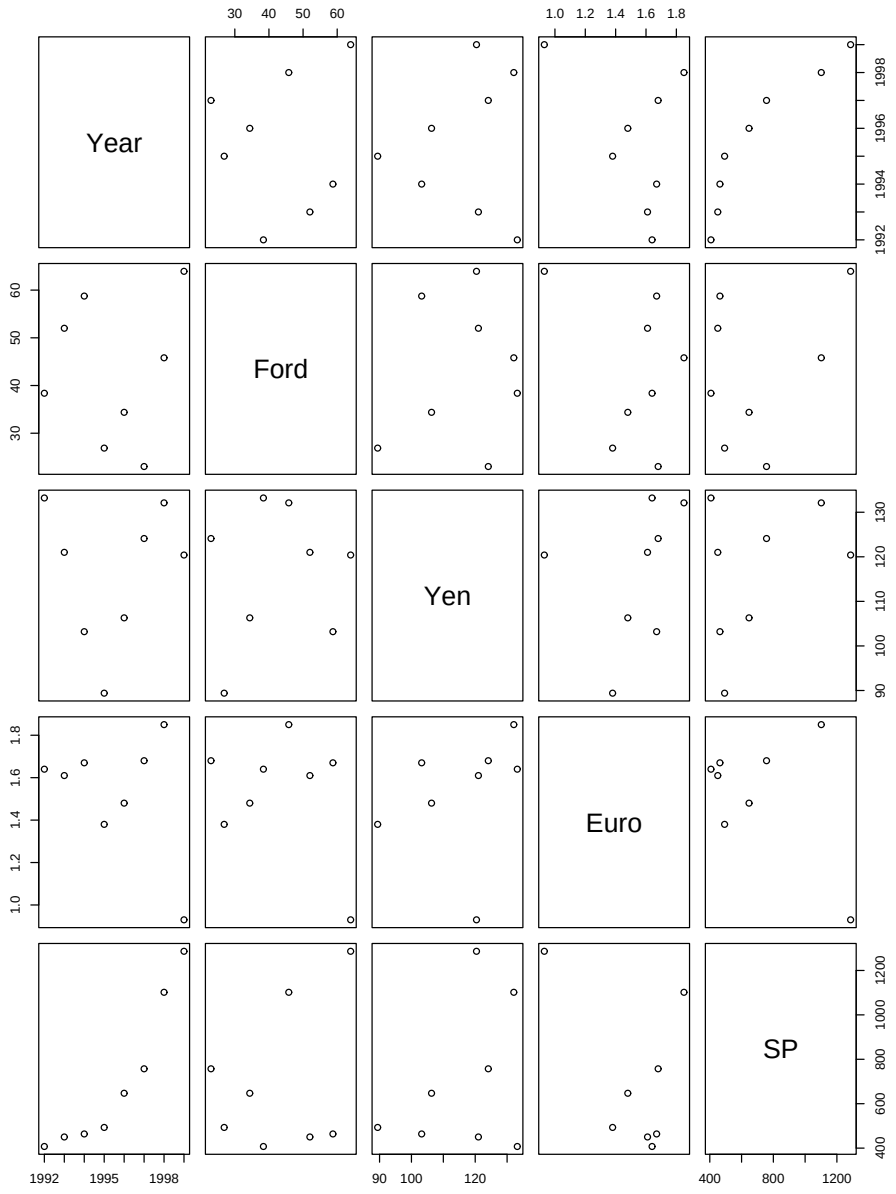
Data Frames and Levels

A data frame is a collection of variables which have the same length. Its structure is like an array in which the elements of a column correspond to the elements of a variable. Some permissible manipulations of a data frame are like those of an array or matrix, but they cannot be fully extended to matrix algebra. In order to apply the operations of matrix algebra on data frames, a data frame must be converted to a matrix. The name `dd` will be given to the object which will be the data frame of the following examples. The data frame can be initialized by the command: `>dd<-data.frame()` and `dd` will consist of the five variables:

```
>year<-c(1992,1993,1994,1995,1996,1997,1998,1999)
>ford<-c(38.38,52,58.75,26.88,34.38,23.02,45.81,63.94)
>yen<-c(133.2,121,103.2,89.4,106.3,124.1,132.1,120.4)
>eu<-c(1.64,1.61,1.67,1.38,1.48,1.68,1.85,.93)
>poors<-c(407.36,450.16,463.81,493.15,647.07,757.12,1101.75,1286.37)
>dd<-data.frame(year,ford,yen,eu,poors)
>names(dd)<-c("Year","Ford","Yen","EU","SP")
```

To assemble these five variables into the data frame, the following command is executed: `>dd<-data.frame(year,ford,yen,eu,poors)`. The resulting structure of `dd` can be displayed by `>str(dd)`. The command `str()` is like the `proc contents` procedure of SAS.

The first order of business in analyzing a set of data is to make a picture of the data. Let us examine the price of Ford common stock per share as a function of the exchange rate for Japanese Yen, Euro, and the Standard and Poors (S&P), index as of the beginning of the year. According to theory, the higher the exchange rate of yen per dollar or euro per dollar rises, the more affordable Ford automobiles become relative to Japanese and German imported cars and therefore the greater the demand for Ford common stock. If the theory is correct, there should appear discernible patterns between the variables and the price of Ford stock. `>plot(dd)`



No simple functional relationship is apparent between the price of Ford stock and the other variables of interest upon inspecting the plot of the data. As a result, a statistical analysis of the data will probably produce no useful information.

2.1 Factors and Levels

There is a structure in R which is associated with a class called factor. A factor is a special type of vector which is designed to contain non-numeric data known as categorical data. The elements of a factor are strings and they are assigned to a level. The following example illustrates the creation of a factor based on a set of data which was obtained from a survey of 51 people. They were asked to rate their satisfaction with their jobs according to seven categories: `don't know`, `very dissatisfied`, `dissatisfied`, `so so`, `satisfied`, and `very satisfied`. Some one who did not respond was give a NA for an answer. Notice that in the vector `job`, there are nine occurrences of the response: `so so`. The vector was created by the `factor()` command. Its basic syntax is: `factor(c(...), levels=c(...))`. The entries of the data and the names of the levels must agree.

```
job<-factor(c("so so","satisfied","very satisfied","satisfied",
"very satisfied","very satisfied","so so","very satisfied","NA",
"so so","satisfied","satisfied","so so","so so","satisfied",
"satisfied","so so","very satisfied","very satisfied","satisfied",
"don't know","so so","very dissatisfied","NA","very satisfied",
"very satisfied","so so","very satisfied","very satisfied",
"very dissatisfied","satisfied","satisfied","so so","very satisfied",
"NA","satisfied","satisfied","satisfied","very satisfied","satisfied",
"dissatisfied","very satisfied","satisfied","dissatisfied",
"satisfied","NA","very satisfied","satisfied","satisfied",
"satisfied","satisfied"),
levels=c("NA","don't know","very dissatisfied","dissatisfied",
"so so","satisfied","very satisfied"))

edu<-factor(c("high school","high school","masters","college",
"college","masters","masters","college","high school","high school",
"PhD","masters","college","high school","college","high school",
"PhD","college","high school","masters","PhD","masters","masters",
"PhD","PhD","college","masters","college","college","masters",
"masters","masters","masters","masters","high school","college",
"masters","masters","masters","masters","grade school","PhD",
"masters","high school","college","college","masters","PhD",
"college","college","masters"),levels<-c("other","PhD",
"masters","college","high school","grade school"))

par(cex.main=2,cex.lab=2,cex=1)
plot(edu,job,main="Job Satisfaction versus Education",
col=c("red","blue","yellow","green","brown","pink","cyan"))
```

Likewise, the vector, `edu`, was created by the command, `factor`. Both vectors, `job` and `edu`, have the same length. The advantage of creating these vectors of categorical data is evident in the graph of `edu` versus `job`. Within each column for a level of education, there are colored regions having relative areas corresponding to the frequency of each level. The colors are stip-

ulated in the `plot()` command. The plotting routine will cycle through the list of specified colors with each occurrence of a level as illustrated in Figure 2.1.

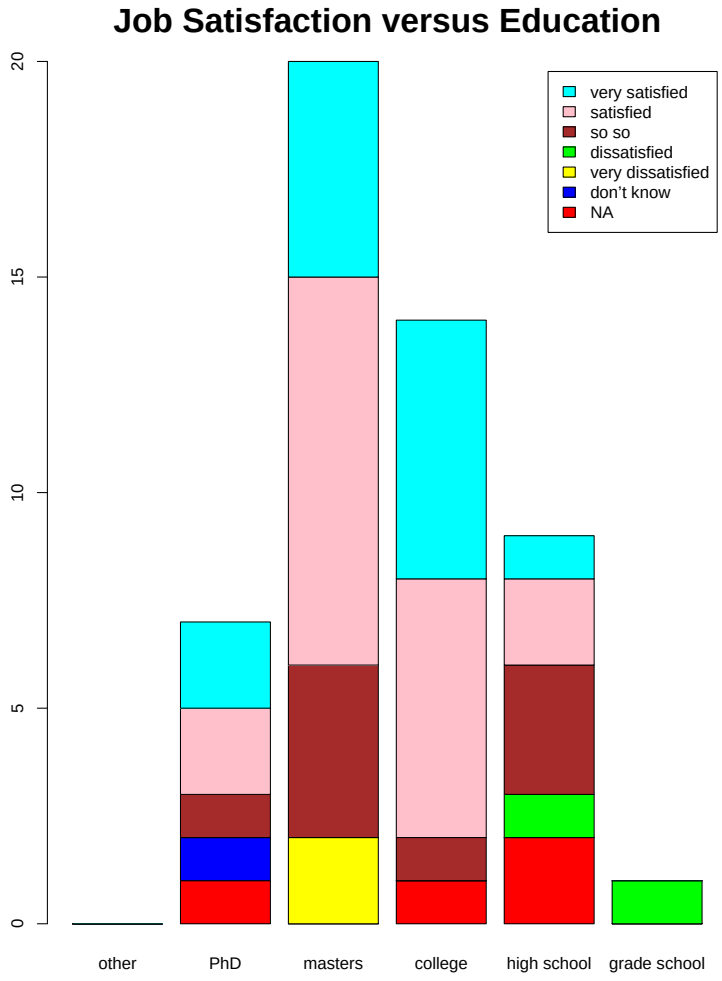


Figure 2.1:

Chapter 3

Least Squares and Data Frames

```
w<-c(83,85,74,70,92,64,72,87,88,75)
x<-c(95,81,59,68,74,79,72,70,81,58)
y<-c(86,71,49,63,65,72,78,68,85,65)
z<-c(87,61,45,81,72,67,66,51,55,58)
```

To assemble these four variables into the data frame, the following command is executed: `dd<-data.frame(w,x,y,z)` .

To illustrate the use of `lm` in making pictures which may be appropriate in the study of linear model, the next example will focus on the problem of fitting a linear model $classroom = \beta_0 + \beta_1 video + \beta_2 audio + \beta_3 text + \epsilon$ where $\epsilon \sim N(0, \sigma^2)$ to the data contained in `dd`. Suppose that the set of data already exists in the data frame, `dd`, so that `lm()` may immediately be applied to it. `lm(dd)` Under the heading of `Coefficients`, the estimates $\beta_0 = 64.39754$, $\beta_1 = 0.50043$, $\beta_2 = -0.05749$, and $\beta_3 = -0.28372$ appear. The same results are produced in the following equivalent formulation. `lm(w~x+y+z)` .

The syntax which represents the model has the form: `w~x+y+z`. All the necessary information for performing an analysis of variance is contained in the output of the `lm` and can be passed to a subsequent procedure like `anova()`: `anova(lm(w~x+y+z))` . Rather than type the command, `lm(w~x+y+z)`, many times over again, the `lm` procedure can be assigned to an object such as: `w.lm<-lm(w~x+y+z)` While expressed as an object, the output of the `lm` procedure can be easily analyzed by means of applying various utilities to it, like: `anova(w.lm)` . In the case of `fitted(w.lm)` , this procedure produces the fitted values of the linear model while the `resid` procedure will produce the residuals of the linear model: `resid(w.lm)` . These two procedures make it easy to produce the very important diagnostic plot of residuals versus predicted values to help determine whether or not the model is a good model.

```
plot(fitted(w.lm),resid(w.lm),main="Residuals versus Predicted Values",
xlab="Predicted Values", ylab="Residuals")
abline(h=0)
```

To make a picture of the data is one of the very first steps in the analysis of data. In this example, the line which best fits the data is drawn and it is annotated with its equation.

```
x<-c(95,81,59,68,74,79,72,70,81,58)
y<-c(86,71,49,63,65,72,78,68,85,65)
plot(x,y,main="Semester Course Grade vs Quiz Grade",
xlab='Quiz Average',ylab='Semester Score')
par(lty=1)
coef<-lsfit(x,y)$coef
abline(coef=coef,lty=2)
B.str<-paste("y=",round(coef[1],2),"+",round(coef[2],2),"x")
text(median(x),coef[1]+coef[2]*median(x) - 1,B.str,pos=4)
boty<-y[order(y)][1:2]
topy<-y[order(-y)][1:2]
botx<-x[order(y)][1:2]
topx<-x[order(-y)][1:2]
for (j in 1:5){
points(botx[j],boty[j],col=2,pch=20)
points(topx[j],topy[j],col=4,pch=20)
}
```

Instead of using the combination:

```
coef<-lsfit(x,y)$coef
abline(coef=coef,lty=2)
```

the simpler command can used instead:

```
abline(lsfit(x,y),lty=2)
```

3.1 Logistic Regression

As soon as a random variable is defined in the context of a phenomenon, an associated probability distribution is immediately induced. Some probability distributions occur so often that they are given names. If a random variable has only two possible values which correspond to two and only two outcomes then it is called a Bernoulli random variable, and it is denoted by $X \sim b(1, p)$ where 1 stands for one trial and p is the probability of success that the event will occur. A Bernoulli random variable is characterized by having one trial with two possible outcomes: success-failure, on-off, 0-1, accept-reject, man-woman, etc.. Sometimes the Bernoulli random variable is referred to as being dichotomous.

Usually, when formulating a model particularly in the social and biological sciences, the sex

of a person or the sex of an animal is used to help explain on outcome. For example, the variables sex, age, years of education, religion, occupation, and the number of dependent children might be used to predict a person's annual income. On the other hand, it is conceivable that age, education, religion, number of dependent children, occupation, and personal income could be used to predict a person's sex. Similarly, temperature, moisture of soil, and pH might be used to predict whether a seed will germinate or die. Or a credit card company might use age, sex, personal income, and personal debt to determine if an applicant for a credit card is a good or bad credit risk,

In these last examples, the response variable is a Bernoulli random variable because there are only two possible outcomes. Because the response can only assume two possible values, such a model cannot be analyzed in the same manner as a regular linear model.

Consider the example of using a person's examination score to predict whether he will pass or fail a course. From a previous class, the following table of scores and outcomes were observed.

Table 3.1: Examination Scores and Pass-Fail in a Course

Score	Pass=1, Fail=0
1	0
2	0
3	0
4	0
5	0
6	0
5	1
6	1
7	1
8	1
9	1
10	1

A plot of the data appears in Figure 3.1. It shows two levels: one at 0 and another at 1.

An analyst would like to predict the probability of a student passing the course based on

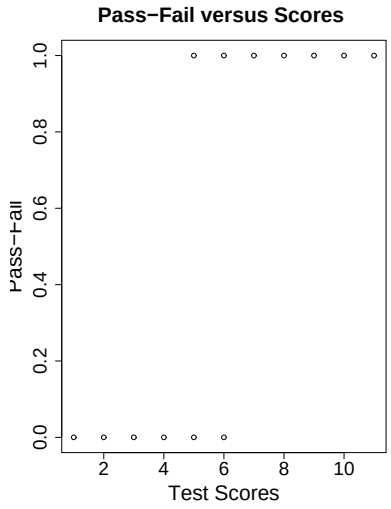


Figure 3.1

his exam score. In order to accommodate the analyst's goal, the method of logistic regression is available. Unlike the method of least squares from which easily derived and compact formulas exist, the method of logistic regression uses least squares indirectly in a rather obscure procedure which can only be processed by means of a computer. There are no equations with which some one can use to produce a fitted logistic curve to the data by hand. As a result, the discussion of logistic regression tends to be either heuristic or highly technical.

A smooth function which approximates a step function lies at the basis of logistic regression. In the picture of the data taken from Table 3.1 and shown in Figure 3.1, two distinct levels exist: one at y=0 and the other at y=1. The logistic function approximates a smooth connection of the two levels as shown in Figure 3.2 where the logistic function is shown superimposed on the plot of the data.

The equation of the logistic function has the form: $f(x) = \frac{e^x}{1+e^x}$. In logistic regression, the probability denoted by p that a trial will lead to a success is assumed to follow the logistic function. In the most simple case, it is assumed that

$$p = \frac{e^{\alpha_0+\alpha_1 x}}{1 + e^{\alpha_0+\alpha_1 x}}$$

This equation can be rearranged through a series of algebraic steps into: $log(\frac{p}{1-p}) = \alpha_0 + \alpha_1 x$. The right hand side of the equation bears a resemblance to a linear model in two parameters, hence, the association of logistic with regression. Given the data consisting of 1's for success and 0's for failure and their respective values for x, a computer statistics package will compute $\widehat{\alpha_0}$ and $\widehat{\alpha_1}$ from a complex algorithm.

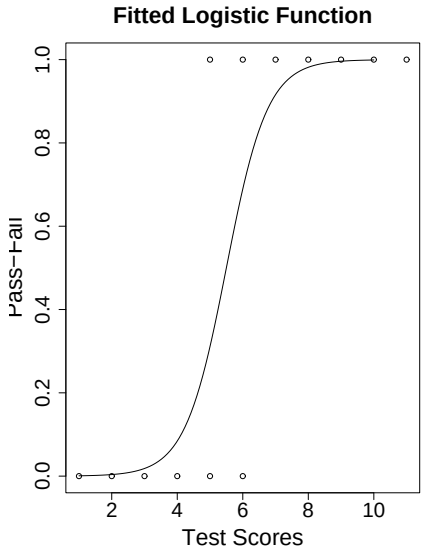


Figure 3.2:

```
library("brlr")
x<-c(1, 2, 3, 4, 5, 6, 5, 6, 7, 8, 9,10,11)
y<-c(0,0,0,0,0,0,1,1,1,1,1,1,1)
coef<-brlr(y x,br=FALSE)$coef
par(cex.lab=2, cex.main=2,cex=1.25,cex.axis=1.8)
plot(x,y,xlab="Test Scores", ylab="Pass-Fail",
main="Fitted Logistic Function")
curve(exp(coef[1]+coef[2]*x)/(1+exp(coef[1]+coef[2]*x)),1,10,add=TRUE)
```

The results of having applied a statistical computer package to the set of data shown in Table 3.1 show that the logistic function which best fits the graph of the data is the one with $\widehat{\alpha_0} = -8.75$ and $\widehat{\alpha_1} = 1.59$. The graph of this particular logistic function appears in Figure 3.2.

Suppose a new student received a score of 5 on the examination, then the probability that he will pass the course is:

$$\begin{aligned} \widehat{p} &= \frac{e^{\widehat{\alpha_0}+\widehat{\alpha_1}x}}{1+e^{\widehat{\alpha_0}+\widehat{\alpha_1}x}} \\ \widehat{p} &= \frac{e^{-8.75+1.59(5)}}{1+e^{-8.75+1.59(5)}} \\ \widehat{p} &= .31 \end{aligned}$$

The probability that the student will pass the course with a score of 5 on the examination is .31.

The ratio $\frac{p}{1-p}$ is called the odds. For example, if the probability that a horse will win a race is 10/11 then the odds are: $\frac{10/11}{1/11} = \frac{10}{1} = 10:1$ that it will win. Or conversely, if the odds are 5:4 that a horse will win, the probability is: $p = \frac{o}{1+o} = \frac{5/4}{9/4} = 5/9$. The *odds ratio* is defined to be:

$$\psi = \frac{\frac{p_1}{1-p_1}}{\frac{p_0}{1-p_0}} = e^{\alpha_1}$$

and $log(\psi) = \alpha_1$ is called the *log ratio*. For example, if x=1,then $p_1 = \frac{e^{\alpha_0+\alpha_1}}{1+e^{\alpha_0+\alpha_1}}$ and if x=0, then $p_0 = \frac{e^{\alpha_0}}{1+e^{\alpha_0}}$. From the previous example, $\alpha_0 = -8.75$ and $\alpha_1 = 1.59$ so that $\psi = e^{1.59} = 4.9$. The odds ratio is $\frac{odds\ of\ passing}{odds\ of\ failing} = \frac{4.9}{1}$. The interpretation that would be given is that for an increase in grade by one unit the odds of passing increases by a factor of 4.9.

Chapter 4

acpclust.R

```
### Code by Eric Lecoutre, Universite catholique de Louvain, Belgium
### Winner of the R Homepage graphics competition 2004

### Works in R 1.8.1 ...

require(ade4)
require(mva)
require(RColorBrewer)
require(pixmap)

# ------------------------------------------------------------------------------------------

postscript(horizontal=FALSE, file="acpclust.ps")
ltitle=function(x,backcolor="#e8c9c1",forecolor="darkred",cex=2,ypos=0.4){

plot(x=c(-1,1),y=c(0,1),xlim=c(0,1),ylim=c(0,1),type="n",axes=FALSE)
polygon(x=c(-2,-2,2,2),y=c(-2,2,2,-2),col=backcolor,border=NA)
text(x=0,y=ypos,pos=4,cex=cex,labels=x,col=forecolor)

}

# plotacpclust(USArrests)

plotacpclust = function(data,xax=1,yax=2,hcut,cor=TRUE,clustermethod="ave",
colbacktitle="#e8c9c1",wcos=3,Rpowered=FALSE,...){
# data: data.frame to analyze
# xax, yax: Factors to select for graphs

# Parameters for hclust
#   hcut
```

```
#   clustermethod

require(ade4)

pcr=princomp(data,cor=cor)

datac=t((t(data)-pcr$center )/pcr$scale)

hc=hclust(dist(data),method=clustermethod)
if (missing(hcut)) hcut=quantile(hc$height,c(0.97))

def.par <- par(no.readonly = TRUE)
on.exit(par(def.par))

mylayout=layout(matrix(c(1,2,3,4,5,1,2,3,4,6,7,7,7,8,9,7,7,7,10,11),ncol=4),
widths=c(4/18,2/18,6/18,6/18),heights=c(lcm(1),3/6,1/6,lcm(1),1/3))

     par(mar = c(0.1, 0.1, 0.1, 0.1))
        par(oma = rep(1,4))
ltitle(paste("PCA ",dim(unclass(pcr$loadings))[2], "vars"),cex=1.6,ypos=0.7)
text(x=0,y=0.2,pos=4,cex=1,labels=deparse(pcr$call),col="black")
pcl=unclass(pcr$loadings)
pclperc=100*(pcr$sdev)/sum(pcr$sdev)
s.corcircle(pcl[,c(xax,yax)],1,2,sub=paste("(",xax,"-",yax,") ",
round(sum(pclperc[c(xax,yax)]),0),"%",sep=""),possub="bottomright",
csub=3,clabel=2)
wsel=c(xax,yax)
scatterutil.eigen(pcr$sdev,wsel=wsel,sub="")

dend=hc
dend$labels=rep("",length(dend$labels))
dend=as.dendrogram(dend)

ngrp=length(cut(dend,hcut)$lower)

ltitle(paste("Clustering ",ngrp, "groups"),cex=1.6,ypos=0.4)

par(mar = c(3, 0.3, 1, 0.5))

# Dendrogram
attr(dend,"edgetext") = round(max(hc$height),1)
plot(dend, edgePar = list(lty=1, col=c("black","darkgrey")),
edge.root=FALSE,horiz=TRUE,axes=TRUE)

abline(v=hcut,col="red")
text(x=hcut,y=length(hc$height),labels=as.character(round(hcut,1)),
col="red",pos=4)
```

```
colorsnames= brewer.pal(ngrp,"Dark2")
groupes=cutree(hc,h=hcut)
ttab=table(groupes)

# Groups
par(mar = c(0.3, 0.3, 1.6, 0.3))
#names.arg=paste("g",ngrp:1,sep="")
mp=barplot(as.vector(rev(ttab)),horiz=TRUE,space=0,col=rev(colorsnames),
xlim=c(0,max(ttab)+10),axes=FALSE,main="Groups",axisnames=FALSE)
text(rev(ttab),mp,as.character(rev(ttab)),col=rev(colorsnames),cex=1.2,pos=4)

# Main ACP scatterplot

par(mar = c(0.1,0.1, 0.1,0.1))
selscores=pcr$scores[,c(xax,yax)]

zi=apply(datac,1,FUN=function(vec)return(sum(vec^2)))
cosinus= cbind(selscores[,1]^2 / zi,selscores[,2]^2 / zi)
cosinus= cbind(cosinus,apply(cosinus,1,sum))
ww= (cosinus[,wcos])*4 +0.5

# Outliers? Test with median+1.5*IQ

# Factor #1
out <- selscores[,1] < median(selscores[,1]) -
1.5 * diff(quantile(selscores[,1],c(0.25,0.75)))
out = out | selscores[,1] > median(selscores[,1]) +
1.5 * diff(quantile(selscores[,1],c(0.25,0.75)))
# factor #2
out = out | selscores[,2] < median(selscores[,2]) -
1.5 * diff(quantile(selscores[,2],c(0.25,0.75)))
out = out | selscores[,2] > median(selscores[,2]) +
1.5 * diff(quantile(selscores[,2],c(0.25,0.75)))

plot(selscores,axes=FALSE,main="",xlab="",ylab="",type="n")
abline(h=0,col="black")
abline(v=0,col="black")

points(selscores[!out,1:2],col=(colorsnames[groupes])[!out],cex=ww,pch=16)
text(x=selscores[out,1],y=selscores[out,2],
labels=dimnames(selscores)[[1]][out],col=(colorsnames[groupes])[out])
box()

# Factor 1
par(mar = c(0.1, 0.1, 0.1, 0.1))
ltitle(paste("Factor ",xax, " [",round(pclperc[xax],0),"%]",sep="" ),
cex=1.6,ypos=0.4)
plotdens(pcr$scores[,c(xax)])
```

```
# Factor 2
par(mar = c(0.1, 0.1, 0.1, 0.1))
ltitle(paste("Factor ",yax," [",round(pclperc[yax],0),"%]",sep=""),
cex=1.6,ypos=0.4)
plotdens(pcr$scores[,c(yax)])

# R logo
#plot(0,0,type="n",xlim=c(0,100),ylim=c(0,15),axes=FALSE)
#if (Rpowered){
# logo <- read.pnm(system.file("pictures/logo.ppm", package="pixmap")[1])
# addlogo(logo, px=c(100- (101/77)*11,100), py=c(0, 11), asp=1)
#}
#text(x=100-15,y=c(2,5),pos=2,labels=c("Powered by R <www.r-project.org>",date()))

#box()

}

 confshade2 =  function(y, xlo, xhi, col = 8.)
{
n <- length(y)
for(i in 1.:(n - 1.)) {
polygon(c(xlo[i], xlo[i + 1.], xhi[i + 1.], xhi[i]), c(y[i], y[
i + 1.], y[i + 1.], y[i]), col = col, border = FALSE)
}
}

confshade=function(x, ylo, yhi, col = 8.)
{
n <- length(x)
for(i in 1.:(n - 1.)) {
polygon(c(x[i], x[i + 1.], x[i + 1.], x[i]), c(ylo[i], ylo[i + 1.],
yhi[i + 1.], yhi[i]), col = col, border = FALSE)
}
}

plotdens=function(X, npts = 200, range = 1.5, xlab = "",
ylab = "", main = "", ...)
{
dens <- density(X, n = npts)
qu <- quantile(X, c(0., 0.25, 0.5, 0.75, 1.))
x <- dens$x
y <- dens$y
fqux <- x[abs(x - qu[2.]) == min(abs(x - qu[2.]))]
fquy <- y[x == fqux]
fquX <- as.numeric(qu[2.])
tqux <- x[abs(x - qu[4.]) == min(abs(x - qu[4.]))]
```

```
tquy <- y[x == tqux]
tquX <- as.numeric(qu[4.])
medx <- x[abs(x - qu[3.]) == min(abs(x - qu[3.]))]
medy <- y[x == medx]
# Prepare les donnees a dessiner
#
medX <- as.numeric(qu[3.])
dx <- dens$x

dy <- dens$y
dx2 <- c(dx[dx <= fquX], fquX, dx[(dx > fquX) & (dx <= medX)], medX,
dx[ (dx > medX) & (dx <= tquX)], tquX, dx[dx > tquX])

dy2 <-  c(dy[dx <= fquX], fquy, dy[(dx > fquX) & (dx <= medX)],
medy, dy[(dx > medX) & (dx <= tquX)], tquy, dy[dx > tquX])
IQX <- dx2[(dx2 >= fquX) & (dx2 <= tquX)]
#
#
# Initialise le graphique
#
# axes(axes = F, xlim = c(min(dx2), max(dx2)), ylim = c(min(dy2), max(d
#
# Dessine la densite
IQy <- dy2[(dx2 >= fquX) & (dx2 <= tquX)]
# Trace densit sous IQ
#
plot(0., 0., xlim = c(min(dx2), max(dx2)), ylim = c(min(dy2), max(dy2)),
axes = F, xlab = xlab, ylab = ylab, main = main,type="n", ...)
# Ajoute mediane
#
#
confshade(IQX, rep(0., length(IQX)), IQy, col = "#bdfcc9")
bdw <- (tquX - fquX)/20.
x1 <- c(medX - bdw/2., medX - bdw/2.)
x2 <- c(medX + bdw/2., medX + bdw/2.)
y1 <- c(0., medy)
# Ajoute lignes wiskers
#
#
polygon(c(x1, rev(x2)), c(y1, rev(y1)), col = 0.)
lines(x = c(fquX, fquX), y = c(0., fquy))
# Ajoute wiskers
#
#
lines(x = c(tquX, tquX), y = c(0., tquy))
meany <- mean(dy2)
IQrange <- tquX - fquX
lines(x = c(medX - range * IQrange, fquX), y = c(meany, meany))
```

```
lines(x = c(tquX, medX + range * IQrange), y = c(meany, meany))
lines(x = c(medX - range * IQrange, medX - range * IQrange), y = c(meany -
(max(dy2) - min(dy2))/8., meany + (max(dy2) - min(dy2))/8.))
#
# Ajoute outliers
#
#
lines(x = c(medX + range * IQrange, medX + range * IQrange), y = c(meany -
(max(dy2) - min(dy2))/8., meany + (max(dy2) - min(dy2))/8.))
out <- c(X[X < medX - range * IQrange], X[X > medX + range * IQrange])
#
# Ajoute les points...
#
# Ajoute l'axe
points(out, rep(meany, length(out)), pch = 5., col = 2.)
# Ajoute l'axe
#
#
points(dx2, dy2, pch = ".", type = "l")
#return(x = dessinx2, y = dessiny2)
axis(1., at = round(c(min(x), fquX, medX, tquX, max(x)), 2.), labels = F,
pos = 0.)
invisible(list(x = dx2, y = dy2))
}

BoxDens=function(data, npts = 200., x = c(0., 100.), y = c(0., 50.),
orientation = "paysage",
add = TRUE, col = 11., border=FALSE,colline = 1., Fill = TRUE)
{

dens <- density(data, n = npts)
dx <- dens$x
dy <- dens$y
if(add == FALSE)
plot(0., 0., axes = F, main = "", xlim = x, ylim = y, xlab = "",
ylab = "")
if(orientation == "paysage") {
dx2 <- (dx - min(dx))/(max(dx) - min(dx)) * (x[2.] - x[1.]) * 0.98 +
x[1.]
dy2 <- (dy - min(dy))/(max(dy) - min(dy)) * (y[2.] - y[1.]) * 0.98 +
y[1.]
seqbelow <- rep(y[1.], length(dx))
if(Fill == T)
confshade(dx2, seqbelow, dy2, col = col)
if (border==TRUE) points(dx2, dy2, type = "l", col = colline)
}
else {
dy2 <- (dx - min(dx))/(max(dx) - min(dx)) * (y[2.] - y[1.]) * 0.98 +
```

```
y[1.]
dx2 <- (dy - min(dy))/(max(dy) - min(dy)) * (x[2.] - x[1.]) * 0.98 +
x[1.]
seqleft <- rep(x[1.], length(dy))
if(Fill == T)
confshade2(dy2, seqleft, dx2, col = col)
if (border==TRUE) points(dx2, dy2, type = "l", col = colline)
}
polygon(x = c(x[1.], x[2.], x[2.], x[1.]), y = c(y[2.], y[2.], y[1.],
y[1.]), density = 0.)
}
data(swiss)
# png(file="swiss.png", width=600,height=400)
plotacpclust(swiss[,1:5], 1, 3, hcut=48)
 dev.off()
```

Chapter 5

Appendix 1

`plotmath` *Mathematical Annotation in R*

Description

If the `text` argument to one of the text-drawing functions (`text`, `mtext`, `axis`) in R is an expression, the argument is interpreted as a mathematical expression and the output will be formatted according to TeX-like rules. Expressions can also be used for titles, subtitles and x- and y-axis labels (but not for axis labels on `persp` plots).

Details

A mathematical expression must obey the normal rules of syntax for any R expression, but it is interpreted according to very different rules than for normal R expressions.

It is possible to produce many different mathematical symbols, generate sub- or superscripts, produce fractions, etc.

The output from `demo(plotmath)` includes several tables which show the available features. In these tables, the columns of grey text show sample R expressions, and the columns of black text show the resulting output.

The available features are also described in the tables below:

Syntax	**Meaning**
`x + y`	x plus y
`x - y`	x minus y
`x*y`	juxtapose x and y
`x/y`	x forwardslash y
`x %+-% y`	x plus or minus y
`x %/% y`	x divided by y

`x %*% y`	x times y
`x[i]`	x subscript i
`x^2`	x superscript 2
`paste(x, y, z)`	juxtapose x, y, and z
`sqrt(x)`	square root of x
`sqrt(x, y)`	yth root of x
`x == y`	x equals y
`x != y`	x is not equal to y
`x < y`	x is less than y
`x <= y`	x is less than or equal to y
`x > y`	x is greater than y
`x >= y`	x is greater than or equal to y
`x %~~% y`	x is approximately equal to y
`x %=~% y`	x and y are congruent
`x %==% y`	x is defined as y
`x %prop% y`	x is proportional to y
`plain(x)`	draw x in normal font
`bold(x)`	draw x in bold font
`italic(x)`	draw x in italic font
`bolditalic(x)`	draw x in bolditalic font
`list(x, y, z)`	comma-separated list
`...`	ellipsis (height varies)
`cdots`	ellipsis (vertically centred)
`ldots`	ellipsis (at baseline)
`x %subset% y`	x is a proper subset of y
`x %subseteq% y`	x is a subset of y
`x %notsubset% y`	x is not a subset of y
`x %supset% y`	x is a proper superset of y
`x %supseteq% y`	x is a superset of y
`x %in% y`	x is an element of y
`x %notin% y`	x is not an element of y
`hat(x)`	x with a circumflex
`tilde(x)`	x with a tilde
`dot(x)`	x with a dot
`ring(x)`	x with a ring
`bar(xy)`	xy with bar
`widehat(xy)`	xy with a wide circumflex
`widetilde(xy)`	xy with a wide tilde
`x %<->% y`	x double-arrow y
`x %->% y`	x right-arrow y
`x %<-% y`	x left-arrow y
`x %up% y`	x up-arrow y
`x %down% y`	x down-arrow y
`x %<=>% y`	x is equivalent to y
`x %=>% y`	x implies y
`x %<=% y`	y implies x
`x %dblup% y`	x double-up-arrow y
`x %dbldown% y`	x double-down-arrow y

`alpha` – `omega`	Greek symbols
`Alpha` – `Omega`	uppercase Greek symbols
`infinity`	infinity symbol
`partialdiff`	partial differential symbol
`32*degree`	32 degrees
`60*minute`	60 minutes of angle
`30*second`	30 seconds of angle
`displaystyle(x)`	draw x in normal size (extra spacing)
`textstyle(x)`	draw x in normal size
`scriptstyle(x)`	draw x in small size
`scriptscriptstyle(x)`	draw x in very small size
`x ~~ y`	put extra space between x and y
`x + phantom(0) + y`	leave gap for "0", but don't draw it
`x + over(1, phantom(0))`	leave vertical gap for "0" (don't draw)
`frac(x, y)`	x over y
`over(x, y)`	x over y
`atop(x, y)`	x over y (no horizontal bar)
`sum(x[i], i==1, n)`	sum x[i] for i equals 1 to n
`prod(plain(P)(X==x), x)`	product of P(X=x) for all values of x
`integral(f(x)*dx, a, b)`	definite integral of f(x) wrt x
`union(A[i], i==1, n)`	union of A[i] for i equals 1 to n
`intersect(A[i], i==1, n)`	intersection of A[i]
`lim(f(x), x %->% 0)`	limit of f(x) as x tends to 0
`min(g(x), x > 0)`	minimum of g(x) for x greater than 0
`inf(S)`	infimum of S
`sup(S)`	supremum of S
`x^y + z`	normal operator precedence
`x^(y + z)`	visible grouping of operands
`x^{y + z}`	invisible grouping of operands
`group("(",list(a, b),"]")`	specify left and right delimiters
`bgroup("(",atop(x,y),")")`	use scalable delimiters
`group(lceil, x, rceil)`	special delimiters

References

Murrell, P. and Ihaka, R. (2000) An approach to providing mathematical annotation in plots. *Journal of Computational and Graphical Statistics*, **9**, 582–599.

See Also

`demo(plotmath)`, `axis`, `mtext`, `text`, `title`

Examples

```
x <- seq(-4, 4, len = 101)
y <- cbind(sin(x), cos(x))
```

```
matplot(x, y, type = "l", xaxt = "n",
        main = expression(paste(plain(sin) * phi, "  and  ",
                                plain(cos) * phi)),
        ylab = expression("sin" * phi, "cos" * phi), # only 1st is taken
        xlab = expression(paste("Phase Angle ", phi)),
        col.main = "blue")
axis(1, at = c(-pi, -pi/2, 0, pi/2, pi),
     lab = expression(-pi, -pi/2, 0, pi/2, pi))

## How to combine "math" and numeric variables :
plot(1:10, type="n", xlab="", ylab="", main = "plot math & numbers")
tt <- 1.23 ; mtext(substitute(hat(theta) == that, list(that= tt)))
for(i in 2:9)
    text(i,i+1, substitute(list(xi,eta) == group("(",list(x,y),")"),
                           list(x=i, y=i+1)))

plot(1:10, 1:10)
text(4, 9, expression(hat(beta) == (X^t * X)^{-1} * X^t * y))
text(4, 8.4, "expression(hat(beta) == (X^t * X)^{-1} * X^t * y)",
     cex = .8)
text(4, 7, expression(bar(x) == sum(frac(x[i], n), i==1, n)))
text(4, 6.4, "expression(bar(x) == sum(frac(x[i], n), i==1, n))",
     cex = .8)
text(8, 5, expression(paste(frac(1, sigma*sqrt(2*pi)), " ",
                            plain(e)^{frac(-(x-mu)^2, 2*sigma^2)})),
     cex = 1.2)
```

Bibliography

John M. Chambers. *Programming with Data A Guide to the S Language*. Springer-Verlag, New York, New York, 1998.

Peter Dalgaard. *Introductory Statistics with R*. Springer-Verlag, New York, New York, 2002.

Charles Fleming. *R Notes*. SIGSTAT, Washington, D.C., 2004.

W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S-PLUS*. Springer-Verlag, New York, New York, 1999.