Lattice and Other Graphics in R

J H Maindonald

Centre for Mathematics and Its Applications
Australian National University.

 April 4, 2009

Languages shape the way we think, and determine what we can think about.
[Benjamin Whorf.]

S has forever altered the way people analyze, visualize, and manipulate data... S is an elegant, widely accepted, and enduring software system, with conceptual integrity, thanks to the insight, taste, and effort of John Chambers.
[From the citation for the 1998 Association for Computing Machinery Software award.]

John H. Maindonald, Centre for Mathematics & Its Applications, Mathematical Sciences Institute, Australian National University, Canberra ACT 0200, Australia, john.maindonald@anu.edu.au

http://www.maths.anu.edu.au/~johnm

There will be occasional references to

DAAGUR: Maindonald, J. H. & Braun, J. B. 2007. Data Analysis & Graphics Using R. An Example-Based Approach. Cambridge University Press, Cambridge, UK, 2007.
http://www.maths.anu.edu.au/~johnm/r-book.html

Useful Web Sites for Australasian R Users:

CRAN (Comprehensive R Archive Network): http://cran.r-project.org

To obtain R and associated packages, use the nearest mirror.
http://mirror.aarnet.edu.au/pub/CRAN or http://cran.ms.unimelb.edu.au/.

Windows, Linux, Unix and MacOS X versions are available, at no cost.

R homepage: http://www.r-project.org/

Wikipedia: http://en.wikipedia.org/wiki/R_(programming_language)

R-downunder: http://www.stat.auckland.ac.nz/mailman/listinfo/r-downunder

For other useful web pages, click on the menu item R help, and look under Resources on the browser window that pops up.

Source of Information on R Graphics:

Helpful books on R graphics, with web sites that give code, are:

> Paul Murrell: *R Graphics.* Chapman and Hall/CRC 2006.
> (http://www.stat.auckland.ac.nz/~paul/RGraphics/rgraphics.html)
>
> Deepayan Sarkar: *Lattice. Multivariate Data Visualization with R.* Springer 2008.
> (http://lmdvr.r-forge.r-project.org).

The CRAN Graphics task view (http://cran.ms.unimelb.edu.au/web/views/Graphics.html) has summary information on a rich variety of R graphics packages.

Note also Hadley Wickham's forthcoming book on *ggplot2*. A draft is available from http://had.co.nz/ggplot2.

Contents

Chapter 1

Preliminaries

The "preliminaries" that are discussed here will extend to using the R Commander menu to draw graphs!

1.1 Installation of R and of R Packages

Installation of R First download and install R from a CRAN site, e.g.

`http://cran.ms.unimelb.edu.au/` or
`http://mirror.aarnet.edu.au/pub//CRAN/`

Windows an MacOS X users should download the relevant executable,
(e.g. **R-2.7.0-win32.exe** for Windows, or **R-2.7.0.dmg** for MacOS X),
and open the downloaded file (e.g., click on it) to start insallation

Installation of R Packages (Windows & MacOS X)

Start R (e.g., click on the R icon). Then use the relevant menu item
to install packages via an internet connection.
This is (usually) easier than downloading, then installing.

Locating packages The CRAN task views may be a good first place to go.

For installation, follow the instructions in the text box. For installing packages, Windows users will first need to specify a mirror. In Australia, specify the Australian mirror.

A fresh install is typically required to take advantage of new major releases (e.g. moving from a 2.6 series release to a 2.7 series release) when they appear. For working through these notes, version 2.7.0 or later should be installed.

1.1.1 Installation of packages from the command line

For packages where there are dependencies, installation from the command line may be an attractive way to go. First, start R, perhaps by clicking on an R icon. Make sure that you have a live internet connection.

For the R Commander, enter:

```
install.packages("Rcmdr", dependencies=TRUE)
```

Doing the installation this way ensures that other packages that R Commander may want get installed at the same time. One of those packages is the *rgl* 3D graphics package that I will describe briefly. Other graphics packages that this installs are *scatterplot3d*, *vcd* (visualization of categorical data) and *colorspace* (for generation of color palettes, etc).

A further package that will be discussed here, the *ggplot2* package, is not an R commander suggested package, and requires separate installation. Enter, at the command line:

```
install("ggplot2", dependencies=TRUE)
```

1.2 The R Commander

1.3 The R Commander GUI

The R commander gives a graphical user interface (GUI) to a wide range of abilities, in the base R system and in R packages. This includes graphical abilities, in the *lattice* and *rgl* packages as well as in base graphics.

To start the R commander, start up R and enter:[1]

```
library(Rcmdr)
```

This opens an R Commander script window, with the output window underneath. This window can be closed by clicking on the × in the top left corner. If thus closed, enter `Commander()` to reopen it again later in the session.

The R Commander GUI – a guide to getting started

Once the points that will now be noted are understood, use of the R Commander should for the most part be straightforward.

From GUI to writing code: The R commander displays the code that it generates. Users can take this code, modify it, and re-run it. The code can be run either from the R Commander script window or from the R console window (if open).

The active data set: The R commander has the notion of an active data set. Here are alternative ways to make a data set active. Start by clicking on the Data drop-down menu. Then

- Click on Active data set, and pick from among data sets, if any, in the workspace.
- Click on Import data, and follow instructions, to read in data from a file. The data set is read into the workspace, at the same time becoming the active data set.
- Click on New data set ..., then entering data via a spreadsheet-like interface.
- Click on Data in packages, click on Read Data from Package, then identify one of the attached packages and choose a data set from among those that are included with the package.
- A further possibility is to load data from an R image (**.RData**) file; click on Load data set ...

Creating graphs: To draw graphs, click on the Graphs drop-down menu. Then

- Click on Scatterplot ... to obtain a scatterplot. This uses the function `scatterplot()` from the *car* package, which is an option rich interface to functions that are in base graphics.
- Click on X Y conditioning plot ... for lattice scatterplots and panels of scatterplots.
- Click on 3D graph to obtain a 3D scatterplot, using the R Commander function `scatter3d()` that is an interface to functions in the *rgl* package.

[1] At startup, the R Commander checks whether all the suggested packages, needed to use all its features, are available. If some are missing, then upon starting up, the R commander offers to install them. For installing such packages, there must be a live internet connection.

Statistics (& fitting models): Click on the Statistics drop down menu to get submenus that give summary statistics and/or carry out various statistical tests. This includes (under Contingency tables) tables of counts and (under Means) One-way ANOVA. Also, click here to get access to the Fit models submenu.

***Models:** Click here to extract information from model objects once they have been fitted. (NB: To fit a model, go to the Statistics drop down menu, and click on Fit models).

Chapter 2

Base Graphics

Base Graphics implements a relatively "traditional" style of graphics:	
Plots go to one or more pages of a graphics device (screen, or hardcopy)	
`plot()`, etc.	Sets up figure region, with user region inside, usually starts the graph. Other functions that initiate a graph include `hist()` and `boxplot()`. Typically, it also creates the main part, or all, of the graph. Use `points()`, `lines()`, `text()`, `mtext()`, `axis()`, `rug()`, `identify()`, etc., to add to the graph.
Plot `y` vs `x`	`with(women, plot(height, weight)) # Older syntax` `plot(weight ~ height, data=women) # Newer syntax (graphics formula)`
Caveat	Some base graphics functions do not take a `data` parameter

To see some of the possibilities that traditional (or base) R graphics offers, enter

```
demo(graphics)
```

Press the Enter key to move to each new graph.

2.1 `plot()` and allied functions

Here are two examples.

```
library(DAAG)
attach(elasticband)  # R can now find distance & stretch
plot(distance ~ stretch)
plot(ACT ~ year, data=austpop, type="l")
plot(ACT ~ year, data=austpop, type="b")
detach(elasticband)
```

Figure 2.1 demonstrates some of the features of base graphics. Base graphics is highly flexible, but often requires a great deal of attention to detail. There are annoying inconsistencies.

2.1.1 Fine control – parameter settings

Users who execute the code given above for Figure 2.1 will notice that the layout is different; there will be bigger margins, and the tick labels and the axis labels will be further out. To get the layout shown, there were some small changes to parameter settings:

```
## Invoke once device is open, and before starting the plot
oldpar <- par(mar = rep(2,4), xaxs="i", yaxs="i", mgp=c(1.5,0.75,0))
```

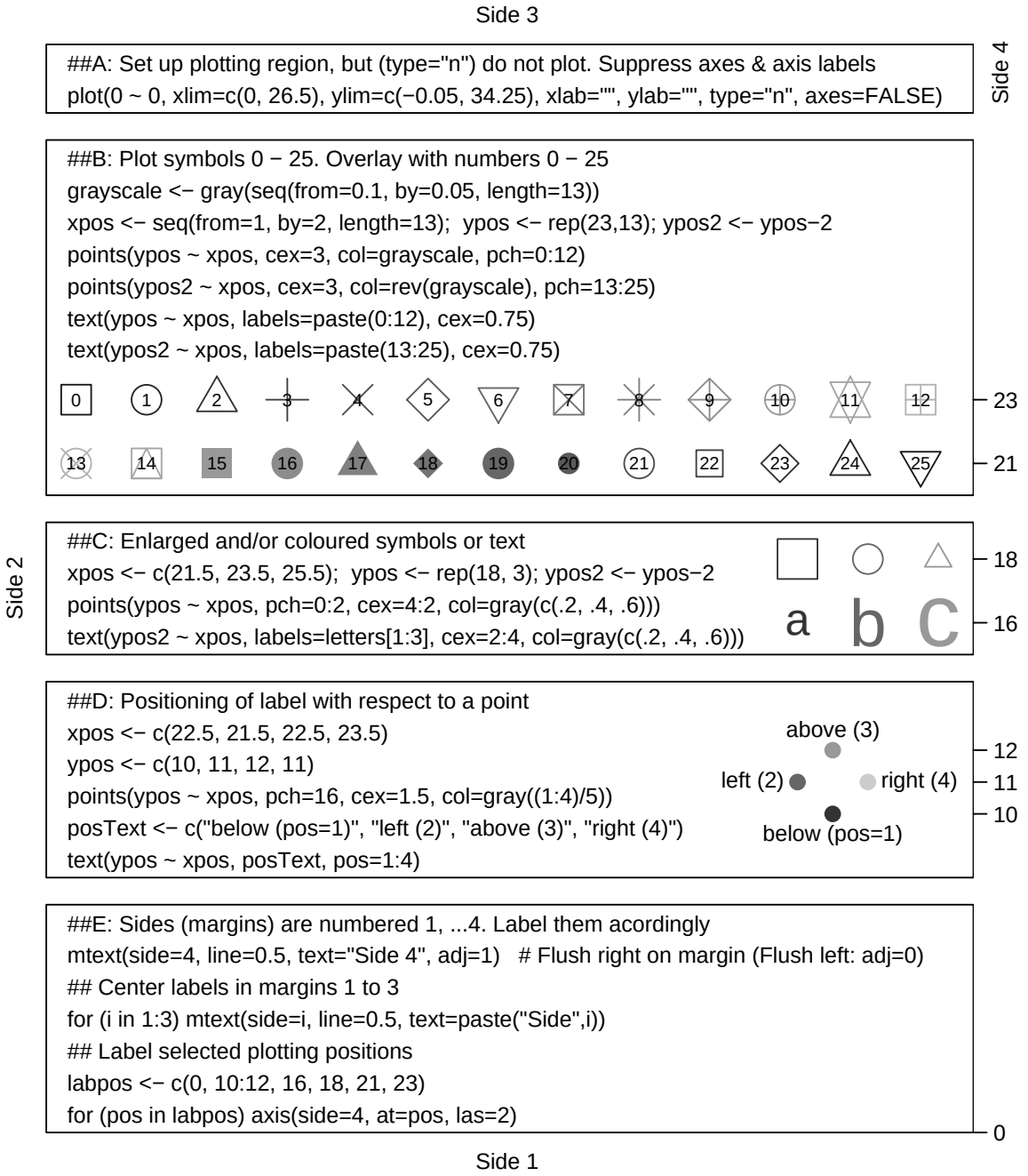

Figure 2.1: Here are illustrated a number of features of traditional graphics plots. The code reproduces the points, labels, ticks, tick labels and axis labels, but not the printing of the code in the figure region.

The existing parameter settings are stored in `oldpar`, so that they can be restored later. Margins are reduced in size (`mar = rep(2,4)`) so that each margin has room for two lines of text only. The figure area will take in the exact x- and y-limits (`xaxs="i", yaxs="i"`), rather than extending slightly beyond those limits. The margin parameters are set so that labels will be printed 1.5 lines out from the margin, tick labels 0.75 lines out from the margin, and ticks right on the margin.

Here are some of the more common settings:

- Plotting symbols: `pch` (choice of symbol); `cex` ("character expansion"); `col` (color). Thus `par(cex=1.2)` increases the plot symbol size 20% above the default.

- Lines: `lty` (line type); `lwd` (line width); `col` (color).

- Axis limits: `xlim`; `ylim`. (Assuming `xaxs="r"`, x-axis limits are by default extended by 4% relative to the data limits. Specify `xaxs="i"` to make the default an exact fit to the data limits. For the y-axis, replace `xaxs` by `yaxs` and x by y.)

- Axis annotation and labels: `cex.axis` (character expansion for axis annotation, independently of `cex`); `cex.labels` (size of the axis labels); `mgp` (margin line for the axis title, axis labels, and axis line; default is `mgp=c(3, 1, 0)`).

- Graph margins: `mar` (inner margins, clockwise from the bottom; the out of the box default is `mar=c(5.1, 4.1, 4.1, 2.1)`, in lines out from the axis); `oma` (outer margins, relevant when there are multiple graphs on the one graphics page).

- Plot shape: `pty="s"` gives a square plot (must be set using `par()`).

- Multiple graphs on the one graphics page: Specify `par(mfrow=c(m,n))` to get an m rows by n columns layout of graphs on a page.

Type `help(par)` to get a (very extensive) complete list.

In most (not all) instances, the change can be made either in a call to a plotting function (eg, `plot()`, `points()`), or using `par()`. If made in a call to a plotting function, the change applies only to that call. If made using `par()`, changes remain in place until changed again, or until a new device is opened.

2.1.2 Adding points, lines and text – examples

Here is a simple example (Figure 2.2) that uses the function `text()` to label the points on a plot.

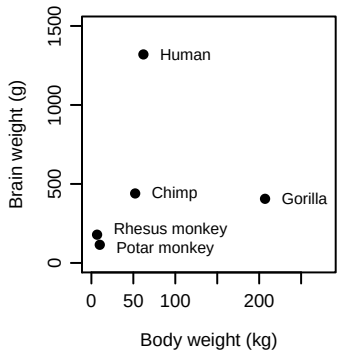

Figure 2.2: Plot of brain weight against body weight, for selected primates.

```
> ## Data used in plot
> primates   # DAAG package
              Bodywt Brainwt
Potar monkey    10.0     115
Gorilla        207.0     406
Human           62.0    1320
Rhesus monkey    6.8     179
Chimp           52.2     440
```

Code that gives the above plot is:

```
attach(primates)
plot(Bodywt, Brainwt, xlim=c(0, 250),
     xlab="Body weight (kg)", ylab="Brain weight (g)")
  # Specify xlim so that there is room for the labels
text(x=Bodywt, y=Brainwt, labels=rownames(primates), pos=4)
  # Alternatives are pos=1 (below), 2 (left), 3 (above)
detach(primates)
```

2.2 Plotting Mathematical Symbols

Lattice, as well as base graphics users, can take advantage of the features described here.

Both `text()` and `mtext()` will take an expression rather than a text string, as in the x-axis label of Figure 2.3. Observe that an arbitrary character string can appear as a variable in an expression. The operator '*' juxtaposes the separate elements of the "expression".

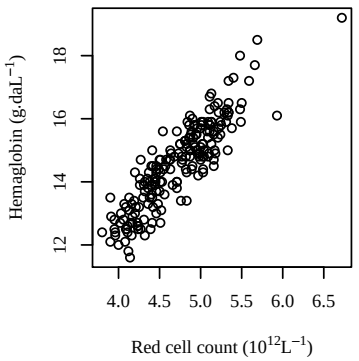

Figure 2.3: Hemaglobin concentration vs red cell count, for 202 Australian athletes. The SI symbol 'daL' is 'decaliters'.

```
par(family="Times")
plot(hg ~ rcc, data=ais,
     xlab=expression("Red cell count ("
                * 10^12 * italic(l)^{-1}
                * ")" ),
     ylab=expression("Hemaglobin ("
                * g*dot(" ")
                * daL^{-1} * ")" ))
```

Note that = must appear as ==, as in:

```
## Code used for plot:
r <- seq(0.1, 8.0, by=0.1)
plot(r, pi * r^2, xlab=expression(Radius == r),
     ylab=expression(Area == pi*r^2), type="l")
 # NB: Use ==, within an expression, to print =
```

2.3 Summary

The functions `plot()`, `points()`, `lines()`, `text()`, `mtext()`, `axis()`, `identify()` etc. form a suite that plots points, lines and text.

Note the alternatives `plot(x, y)`, `plot(y ~ x)`

2.4 Exercises

1. Check the distributions of head lengths (`hdlngth`) in the `possum` data set. Compare the following forms of display:

   a) a histogram (`hist(possum$hdlngth)`);

   b) a stem and leaf plot (`stem(qqnorm(possum$hdlngth))`;

   c) a normal probability plot (`qqnorm(possum$hdlngth)`); and

d) a density plot (`plot(density(possum$hdlngth))`.

What are the advantages and disadvantages of these different forms of display?

2. For the columns of the data frame `nihills`, examine the distribution using histograms, density plots and normal probability plots.

   Repeat the exercise with the logarithms of the data values.

3. Use `mfrow()` to set up the layout for a 3 by 4 array of plots. In the top 4 rows, show normal probability plots for four separate 'random' samples of size 10, all from a normal distribution. In the middle 4 rows, display plots for samples of size 100. In the bottom four rows, display plots for samples of size 1000. Comment on how the appearance of the plots changes as the sample size changes.

4. (a) The function `runif()` can be used to generate a sample from a uniform distribution, by default on the interval 0 to 1. Try `x <- runif(10)`, and print out the numbers you get. Then repeat exercise 6 above, but taking samples from a uniform distribution rather than from a normal distribution. What shape do the points follow?

   (b) Repeat exercise (a), but for other distributions such as chi-square (`rchisq()`) and t (`rt()`) (try, e.g., degrees of freedom 1, 5 and 40).

5. The data set `LakeHuron` (*datasets* package) has mean July average water surface elevations, in feet, IGLD (1955) for Harbor Beach, Michigan, on Lake Huron, Station 5014, for 1875-1972. Use the following to create a data frame that has the same information:

```
huron <- data.frame(year=as(time(LakeHuron), "vector"),
                    mean.height=LakeHuron)
```

   a) Plot `mean.height` against year.

   b) Use `identify()` to determine which years correspond to the lowest and highest mean levels. That is, type

```
identify(huron$year, huron$mean.height, labels=huron$year)
```

   and use the left mouse button to click on the lowest point and highest point on the plot. To quit, press both mouse buttons simultaneously.

   c) As in the case of many time series, the mean levels are correlated from year to year. To see how each year's mean level is related to the previous year's mean level, use

```
lag.plot(huron$mean.height)
```

   This plots the mean level at year i against the mean level at year i-1.

   d) *Now explain why the following code achieves the same effect:

```
plot(LakeHuron)
identify(LakeHuron, labels=time(LakeHuron))
```

   e) *Use the function `acf()` to plot the autocorrelation function. Compare with the result from the `pacf()` (partial autocorrelation). What are the graphs telling you? (For an explanation of the autocorrelation function, look up "Autocorrelation" on Wikepedia.)

Chapter 3

Lattice Graphics

3.1 Lattice Graphics

Lattice Graphics:	
Lattice	Lattice is a flavour of trellis graphics (the S-PLUS flavour was the original implementation)
Grid	*grid* is a low-level graphics system. It was used to build *lattice*. For *grid*, see Part II of Paul Murrell's *R Graphics*
Lattice vs base	Lattice is more structured, automated and stylized. Much is done automatically, without user intervention. Changes to the default style are harder than for base.
Lattice syntax	Lattice syntax is consistent and tightly regulated For lattice, graphics formulae are mandatory.

Lattice (trellis) graphics functions allow the use of the layout on the page to reflect meaningful aspects of data structure. Different levels of a factor may appear in different panels. Or they may appear in the same panel, distinguished by color and/or symbol. If lines or smooth curves are added, there is a different line or curve for each different group.

Similar considerations apply when columns of data are plotted in parallel. Different columns may appear in different panels. Or they may appear in the same panel, distinguished by color and/or symbol.

The lattice package comes already installed with all the binary distributions that are supplied from CRAN (Comprehensive R Archive Network: `http://mirror.aarnet.edu.au/pub//CRAN/`).

To see some of the possibilities that lattice graphics offers, enter

```
library(lattice)      # For use, the lattice package must be attached
demo(lattice)
```

The *lattice* package implements a *trellis* style of graphics, as in the S+ implementation of the S language.

Lattice graphics versus base graphics – `xyplot()` versus `plot()`

A `Brainwt` versus `Bodywt` scatterplot for the `primates` data, such as was given earlier, might alternatively have been obtained using the function the function `xyplot()` from the *lattice* package. The following, when typed on the command line, give a plot on the graphics device:

```
## Plot Brainwt vs Bodywt, data frame primates (DAAG)
plot(Brainwt ~ Bodywt, data=primates)       # base graphics
```

```
 # 'base' graphics use the abilities of the graphics package
library(lattice)
xyplot(Brainwt ~ Bodywt, data=primates)      # lattice
```

The mechanism that yields the plot is different in the two cases:

- `plot()` gives a graph as a side effect of the command.
- `xyplot()` generates a graphics object. As this is output to the command line, the object is "printed", i.e., a graph appears.

The following illustrates the difference between the two functions:

```
invisible(plot(Brainwt ~ Bodywt, data=primates))      # Graph appears
invisible(xyplot(Brainwt ~ Bodywt, data=primates))    # No graph
```

The function `invisible()` suppresses command line printing, so that `invisible(xyplot(...))` does not yield a graph.

Inside a function, `xyplot(...)` prints a graph only if it is the return value from the function, i.e. usually, is on the final line. In a file that is sourced (use `source()`, no graph will appear. Inside a function (except as mentioned), or in a file that is sourced, there must be an explicit `print()`, i.e.

```
print(xyplot(ACT ~ year, data=austpop))
```

Lattice functions create `trellis` objects. Trellis objects can be created even if no device is open. Such objects can be updated. Objects are plotted (by this time, a device must be open), either when output from a lattice function goes to the command line (thus implicitly invoking the `print()` command), or by the explicit use of `print()`.

By successively updating a trellis graphics object, it can be built up and/or modified in steps. Additionally, it is possible to add to a 'printed" or displayed graphics page. Subsection 3.1.5 will show how to do this.

Panels of scatterplots – the use of xyplot()

Graphics functions in the *lattice* package, are designed to allow row by column layouts of panels. Different panels are for different subsets of the data. Additionally, points can be distinguished, within panels, according to some further grouping within the data.

The `ais` dataset (*DAAG*) has data on elite Australian athletes who trained at the Australian Institute of Sport. Data were collected with a view to studying possible differences in blood characteristics, between athletes in endurance-related events and those in power-related events. The help page for `ais` has details of the measurements, including a variety of blood cell counts.

Here is a breakdown, by sex and sport, of numbers:

```
> with(ais, table(sex,sport))
   sport
sex B_Ball Field Gym Netball Row Swim T_400m T_Sprnt Tennis W_Polo
  f     13     7   4      23  22    9     11       4      7      0
  m     12    12   0       0  15   13     18      11      4     17
```

There are 202 athletes in total, all from the Australian Institute of Sport.

Figure 3.1 demonstrates the use of `xyplot()`. The plot is restricted to rowers and swimmers. The two panels distinguish the two sports, while different plotting symbols (on a color device, different colors will be used) distinguish females from males. Here is suitable code:

```
trellis.device(color=FALSE)
xyplot(ht ~ wt | sport, groups=sex, pch=c(4,1), aspect=1, data=ais,
       auto.key=list(columns=2), subset=sport%in%c("Row","Swim"))
dev.off()             # Close device
trellis.device()      # Start new device, by default with color=TRUE
```

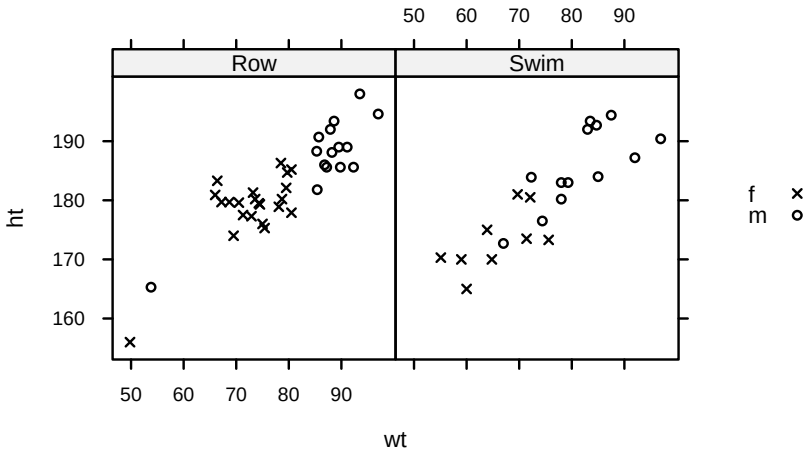

Figure 3.1: Height (`ht`) versus Weight (`wt`), for two categories of athlete. The different plotting symbols distinguish males from females.

In the graphics formula `ht ~ wt | sport`, the vertical bar indicates that what follows, in this case `sport`, is a conditioning variable or factor. The graphical information is broken down according to the factor levels or distinct values. The parameter `aspect` controls the ratio of dimensions in the y and x directions.

The setting `auto.key=list(columns=2)` generates a simple key, with the two key items side by side in two columns rather than one under another in a single column as happens with the default setting `columns=1`.

Selected lattice functions

```
dotplot(factor ~ numeric,..)    # 1-dim. Display
stripplot(factor ~ numeric,..) # 1-dim. Display
barchart(character ~ numeric,..)
histogram( ~ numeric,..)
densityplot( ~ numeric,..)      # Density plot
bwplot(factor ~ numeric,..)     # Box and whisker plot
qqmath(factor ~ numeric,..)     # normal probability plots
splom( ~ dataframe,..)          # Scatterplot matrix
parallel( ~ dataframe,..)       # Parallel coordinate plots
cloud(numeric ~ numeric * numeric, ...)       # 3D surface
wireframe(numeric ~ numeric * numeric, ...)  # 3D scatterplot
```

In each instance, users can add conditioning variables.

Further points to note about the *lattice* package are:

- Because the *lattice* package implements the trellis style of graphics, several of the functions that control stylistic features (color, plot characters, line type, etc.) have *trellis* (where *lattice* might have seemed more natural) as part of their name.
- Lattice graphics functions cannot be mixed (or not easily) with the graphics functions discussed earlier in Section 2.1.2. It is not possible to use `points()`, `lines()`, `text()`, etc., to add features to a plot that has been created using a *lattice* graphics function. Instead, it is necessary to use functions that are special to *lattice* – `lpoints()`, `llines()`, `ltext()`, `larrows()` and `lsegments()`

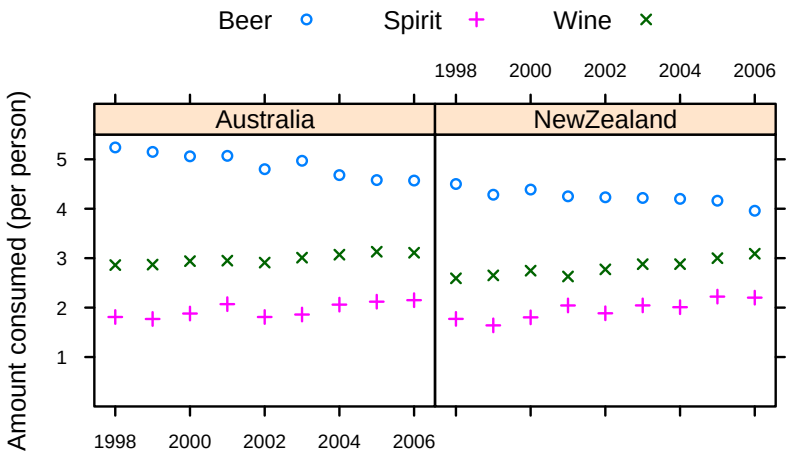

Figure 3.2: Australian and New Zealand apparent per person annual consumption (in liters) of the pure alcohol content of liquor products, for 1998 to 2006.

3.1.1 Groups within data, and/or columns in parallel

Here are selected lines from the data set `grog` (*DAAGxtras* package):

	Beer	Wine	Spirit	Country	Year
1	5.24	2.86	1.81	Australia	1998
2	5.15	2.87	1.77	Australia	1999
. . . .					
9	4.57	3.11	2.15	Australia	2006
10	4.50	2.59	1.77	NewZealand	1998
11	4.28	2.65	1.64	NewZealand	1999
. . . .					
18	3.96	3.09	2.20	NewZealand	2006

There are three liquor products (drinks), in different columns, and two countries, occupying different rows that are indexed by the factor `Country`. The function `xyplot()` can accommodate any of the following:

- Different symbols and/or colors, in the one panel, distinguish drinks. Different panels distinguish countries, as in Figure 3.2.[1] If different countries are in the same panel, then drinks must be separated across different panels.
- Use a 3 drinks × 2 countries, or 2 countries × 3 drinks layout of panels.

Where plots are superposed in the one panel and, e.g., regression lines or smooth curves are fitted, this is done separately for each different set of points. Different colors, and/or by different symbols and/or line styles, can be used to make the necessary distinctions.

Code for Figure 3.2 is:

```
## Simple version of plot
grogplot <- xyplot(Beer+Spirit+Wine ~ Year | Country, data=grog,
```

[1]The data (dataset `grog`, from *DAAGxtras*) are 1998 – 2006 Australian and New Zealand apparent per person annual consumption (in liters) of the pure alcohol content. Data, based on Australian Bureau of Statistics and Statistics New Zealand figures, are obtained by dividing estimates of total available alcohol by number of persons aged 15 or more.

```
                    outer=FALSE, auto.key=list(columns=3))
## Enhance, and print enhanced code
update(grogplot, ylim=c(0,5.5),
       xlab="", ylab="Amount consumed (per person)",
       par.settings=simpleTheme(pch=c(1,3,4)))
```

The footnote[2]has alternative code that updates the object, then uses an explicit `print()`.

Observe that:

- Use of `Beer+Spirit+Wine` gives plots for each of `Beer`, `Spirit` and `Wine`. The effect of `outer=FALSE` is that these appear in the same panel.
- Conditioning by country (| `Country`) gives separate panels for separate countries.
- The function `simpleTheme()` sets up a "theme" that can be used to control point and line settings.

For separate panels for the three liquor products (different levels of `Country` can now use the same panel), specify `outer=TRUE`:

```
xyplot(Beer+Spirit+Wine ~ Year, groups=Country, outer=TRUE,
       data=grog, auto.key=list(columns=2) )
```

Here is a summary:

Overplot (a single panel)	Separate panels
Break data down a/c to levels of the factor `Country`:	
`Beer` $\sim$ `Year, groups=Country`	`Beer` $\sim$ `Year` | `Country`
Plot columns in parallel, as in `Beer+Wine+Spirit` $\sim$ `Year`:	
`outer=FALSE`	`outer=TRUE`

3.1.2 Lattice Parameters and Graphics Features

Point, line and fill color settings, using `simpleTheme()`

The function `simpleTheme()` creates a "theme", i.e., a list of parameter settings, in a form that can be supplied: (i) in the argument `par.settings` in the graphics function call; or (ii) in the argument `theme` in a call to `trellis.par.set()`, prior to calling the graphics function. A further possibility is to include an argument `theme` when using `trellis.device()` to start a new device. This has the default `retain=FALSE`, with the result that unless otherwise specified, parameters are reset to their defaults for the relevant device.

The following creates two "themes":

```
settings1 <- simpleTheme(pch = c(1,3,4), cex=1.25)
settings2 <- simpleTheme(pch = c(1,3,4), alpha=0.5)
```

Use of `settings1` may be appropriate when the number of points is small, while `settings2` may be appropriate when there are many points and there is extensive over-plotting. Here, `alpha` controls the background transparency (c.f., also, `alpha.points` and `alpha.line`). Use of a value less than 1 helps in showing the density of points in regions where there is extensive overlapping.

The following gives the symbols and size of symbol used in Figure 3.2:

[2]
```
## Update trellis object, then print
frillyplot <-
  update(grogplot, ylim=c(0,5.5), xlab="", ylab="Amount consumed (per person)",
         par.settings=simpleTheme(pch=c(1,3,4)))
print(frillyplot)
```

```
grogplot0 <- xyplot(Beer+Spirit+Wine ~ Year | Country, outer=FALSE,
                    data=grog, ylim=c(0,5.5))
grogplot <- update(grogplot0, par.settings=settings1)
print(grogplot)
```

The settings are stored as part of the graphics object `grogplot`.

Consider now the use of `trellis.par.set()` to change the settings globally, so that they remain in place until there is a further change or a new device is opened.

```
trellis.par.set(settings2)
```

Then `print(grogplot)` will use `settings1` which are stored as part of the object, while `print(grogplot0)` will use the global `settings2`.

3.1.3 Setting that are not available using `simpleTheme()`

For changes that go beyond what `simpleTheme()` allows, it is necessary to know the names under which settings are stored. To inspect these, type:

```
> names(trellis.par.get())
 [1] "fontsize"         "background"       "clip"
. . .
[28] "par.sub.text"
```

For a visual display that shows default settings for points, lines and fill colour, try the following:

```
trellis.device(color=FALSE)
show.settings()
trellis.device(color=TRUE)
show.settings()
```

The following sets the fontsize. Notice the separate settings for text and symbols:

```
trellis.par.set(list(fontsize = list(text = 7, points = 4)))
```

Parameters that affect axes, tick marks, and axis labels

These are readily manipulated by use of the `scales` argument to the lattice function. The following plots quarterly labor force numbers, in six regions of Canada, over 1995-1996. The code is:

```
## Create a simplified version of the graphics object
jobs.xyplot <-
  xyplot(Ontario+Quebec+BC+Alberta+Prairies+Atlantic ~ Date,
         data=jobs, type="b", layout=c(3,2), ylab="Number of jobs",
         scales=list(y=list(relation="sliced", log=TRUE)),
         outer=TRUE)
## Update jobs.xyplot, with various enhancements
ylabpos <- exp(pretty(log(unlist(jobs[,-7])), 100))
ylabels <- paste(round(ylabpos),"\n(", log(ylabpos), ")", sep="")
## Create a date object 'startofmonth'; use this instead of 'Date'
atdates <- seq(from=95, by=0.5, length=5)
datelabs <- format(seq(from=as.Date("1Jan1995", format="%d%b%Y"),
                    by="6 month", length=5), "%b%y")
update(jobs.xyplot, xlab="", between=list(x=0.5, y=0.5),
       scales=list(x=list(at=atdates, labels=datelabs),
                    y=list(at=ylabpos, labels=ylabels), tck=0.6) )
```

The enhancements are:

- The y-axis labels show number of jobs, with log(number) in parentheses underneath.
- Dates of the form `Jan95` label the x-axis.
- Tick marks are reduced in length (`tck=0.6`, i.e., 60% of the default).

Notice also the use of `between=list(x=0.5, y=0.5)` to add horizontal and vertical space between the panels, ensuring that the tick labels do not overlap.

3.1.4 Keys – `auto.key`, `key` & `legend`

The argument `auto.key=TRUE` gives a basic key that identifies colors, plotting symbols and names for the groups. For greater flexibility, `auto.key` can be a list. Settings that are often useful are:

- `points`, `lines`: in each case set to `TRUE` or `FALSE`.
- `columns`: number of columns of keys.
- `x` and `y`, which are coordinates with respect to the whole display area. Use these with `corner`, which is one of `c(0,0)` (bottom left corner of legend), `c(1,0)`, `c(1,1)` and `c(0,1)`.
- `space`: one of `"top"`, `"bottom"`, `"left"`, `"right"`.

Use of `auto.key` sets up the call `key=simpleKey()`. If not otherwise specified, colors, plotting symbols, and line type use the current trellis settings for the device. Unless the argument `text` is supplied, `levels(groups)` provides the legends. If necessary, use `legend=NULL` when updating, to remove an existing key and allow the addition of a different key.

3.1.5 Panel functions and interaction with plots

Each lattice command that creates a graph has its own panel function. Thus `xyplot()` has the panel function `panel.xyplot()`. The following are equivalent:

```
xyplot(ACT ~ year, data=austpop)
xyplot(ACT ~ year, data=austpop, panel=panel.xyplot)
```

The user's own function can be substituted for `panel.xyplot()`. Panel functions that may be used, either in combination with functions such as `panel.xyplot()` or separately, include:

- `panel.points()`, `panel.lines()` and a number of other such functions that are documented on the same help page as `panel.points()`;
- `panel.abline()`, `panel.curve()`, `panel.rug()`, `panel.average()` and a number of other functions that are documented on the same help page as `panel.abline()`.

Interaction with lattice plots – the *playwith* package

For using *playwith*, the GTK+ toolkit must be installed. For details, go to the website `http://playwith.googlecode.com/`.

For installing the *playwith* package type, from the command line:

```
install.packages("playwith", dependencies=TRUE)
```

Now type, for example

```
library(DAAGxtras)
library(playwith)
playwith(xyplot(age ~ distance, data=hotspots),
                labels=hotspots$name)
```

Plate 3.4 was then obtained as described in the figure caption.

An alternative is

```
gph <- xyplot(age ~ distance, data=hotspots)
playwith(update(gph), labels=hotspots$name)
```

The menu that appears to the left of the graph can be used to initiate single click identification, to add annotation or arrows, or to mark out a rectangle on the graph for zooming in or out. If labels are not specified, row names are used.

Note also the function `latticist()` in the *latticist* package. When called with a data frame as argument, this opens a window that has graphical summary information on the columns of the data frame. Additionally, the window gives a graphical user interface to the creation of further lattice plots from the data frame. As when `playwith()` is used as a wrapper for a call to a lattice function, various annotation features are available.

3.1.6 Interaction with lattice plots – focus, interact, unfocus

As described here, interaction starts with the use of `trellis.focus()` to focus down to the relevant "viewport", by default a panel. It may be called without arguments. If there is only one panel, it is then selected immediately. If there is more than one panel, the user chooses a panel by clicking on it.

Other choices of `name` include `"panel"`, `"strip"`, `name="legend"` and `"toplevel"`. For `name="legend"`; `side` should be indicated.

Use of `panel.text()` to label points

Following a call to `trellis.focus()`, panel functions can be used to supplement plots. Use `trellis.panelArgs()` to extract arguments that are available to panel functions following such a call. The following adds text labels:

```
xyplot(Brainwt ~ Bodywt, data=primates)
trellis.focus("panel", row=1, column=1, clip.off=TRUE,
              highlight=FALSE)  # Non-interactive use
xyetc <- trellis.panelArgs()
panel.text( x=xyetc$x, y=xyetc$y, labels=row.names(primates), pos=3)
trellis.unfocus()
```

For non-interactive use, be sure to call `trellis.focus()` with the argument `highlight=FALSE`.

Use of `panel.identify()` to label points interactively

Here is an example of interactive labeling.

```
## Use of xyplot(): data frame tinting (DAAG)
library(lattice)
xyplot(it ~ csoa | sex, data=tinting)
trellis.focus("panel", column=1, row=1)
panel.identify(labels=as.character(tinting$target))
## Now click near points as required.
## Terminate by right clicking inside the panel.
## Now interact with panel 2
trellis.focus("panel", row=1, column=2)
panel.identify(labels=as.character(tinting$target))
## Click, ..., and right click
```

By default, the `x` and `y` arguments to the function `panel.identify()` are taken to be those that were supplied to the lattice function, here `xyplot()`.

Functions that may be called include `panel.lines()` and related functions, and `panel.abline()` and related functions, as described earlier.

3.1.7 Arbitrary placement of labels

The following uses the focus/unfocus mechanism to add to an existing graph:

```
stripplot(species ~ length, xlab="", data=cuckoos)
trellis.focus(name="toplevel", highlight=FALSE)
panel.text("Stripplot of cuckoo data", x=0.05, y=0.95)
  # Here, x=0.05 translates to x=unit(0.05,"npc"); the range is (0,1)
trellis.unfocus()
```

The following uses the *grid* function `textGrob()` to create a text object which is then supplied to the lattice function:

```
library(grid)
stripplot(species ~ length, xlab="", data=cuckoos,
          legend=list(top=list(fun=textGrob,
                               args=list(label="Stripplot", x=0))))
  # Here, x=0 is equivalent to x=unit(0,"npc"); the range is (0,1)
```

3.1.8 Multiple graphs on a single graphics page

Note first, for comparison, the use of the base graphics parameter `fig` to mark out a rectangular region where the graph will appear. For example:

```
par(fig = c(0, 1, 0.38, 1))  # xleft, xright, ybottom, ytop
## Plot graph A
par(fig = c(0, 1, 0, 0.38), new=TRUE)
## Plot graph B
par(fig = c(0, 1, 0, 1)      # Reset to default
```

The initial `par(fig = c(0, 1, 0.38, 1))` marks out a plot region that occupied the total width of the graphics page, started 38% of the way up, and extended to the top of the page. The subsequent `par(fig=c(0, 1, 0, 0.38), new=TRUE)` marked out the lower 38% of the page . The effect of `new=TRUE` is, counter-intuitively, "assume a new page is already open; do not open a new page".

For lattice graphs, the location of the graph can be determined by the argument `position`, when `print()` is called. The following demonstrates its use:

```
cuckoos.strip <- stripplot(species ~ length, xlab="", data=cuckoos)
print(cuckoos.strip, position=c(0,0.5,1,1))
                           # xleft, ybottom, xright, ytop
cuckoos.bw <- bwplot(species ~ length, xlab="", data=cuckoos)
print(cuckoos.bw, position=c(0,0,1,0.5), newpage=FALSE)
```

Note the use of `newpage=FALSE` for the second plot.

Base and trellis plots on the same graphics page

The following uses the base graphics command `mtext()` to label a lattice plot:

```
plot(0:1, 0:1, type="n", bty="n", axes=FALSE, xlab="", ylab="")
mtext(side=3, line=3, "Lattice bwplot (i.e., boxplot)")
cuckoos.bw <- bwplot(species~length, data=cuckoos)
print(cuckoos.bw, newpage=FALSE)
```

3.1.9 Plots that Show Distributions

Stripplots, dotplots and boxplots

Because the syntax for `stripplot()` and `boxplot()` are very similar, we demonstrate suitable code side by side. (The function `dotplot()` is very similar to `stripplot()`, with differences that are mainly cosmetic.) The following code creates plots using the `cuckoos` data (from *DAAG*):[3]

```
stripplot(species ~ length, aspect=0.5, data=cuckoos,
          xlab="Cuckoo egg length (mm)")
bwplot(species ~ length, aspect=0.5, data=cuckoos,
       xlab="Cuckoo egg length (mm)")
```

The `aspect` argument determines the ratio of distance in the y-direction to distance in the x-direction.

Lattice Style Density Plots

Here is a density plot (Figure 3.3), for data from the `possum` data set (*DAAG*), that compares `sexes` and `Vic/other` populations.

```
densityplot(~ earconch | sex, groups=Pop, data=possum,
            par.settings=simpleTheme(col=c("gray","black"),
            auto.key=list(columns=2))
```

Where `densityplot()` (and `histogram()`) have a formula as argument, a name is not allowed on the left of the ∼ symbol.

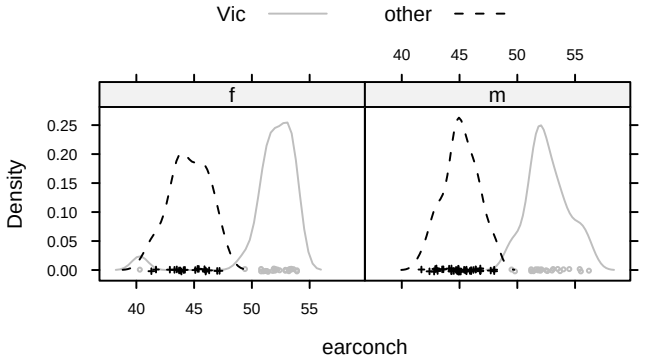

Figure 3.3: Lattice style density plot comparing possum earconch measurements, separately for males and females, between Victorian and other populations. Observe that the scatter of data values is shown along the horizontal axis.

Where `densityplot()` (and `histogram()`) have a formula as argument, a name is not allowed on the left of the ∼ symbol. For `histogram()`, which is otherwise similar to `densityplot()`, the `groups` argument is not available.

Further comments on Stripplots and Dotplots

Unless care is taken with the dimensions of the graphics page and/or the font size, labels for the levels of `yield` in the following plot will overlap.

```
dotplot(variety ~ yield | site, data = barley, groups = year,
    xlab = "Barley Yield (bushels/acre) ", ylab = NULL,
    layout = c(1, 6), aspect = 0.5,
    auto.key=list(labels=levels(barley$year), space = "right"))
```

[3] `## For slightly improved labeling, precede with:`
`levels(cuckoos$species) <- sub(".", " ", levels(cuckoos$species), fixed=T)`

Try stretching the plot vertically so that the labels do not overlap.

The argument `type="h")` gives a line from the origin to the point. Both a line and a point may be given. This can be used to striking effect, as in the following:[4]

```
deathrate <- c(40.7, 36,27,30.5,27.6,83.5)
hosp <- c("Cliniques of Vienna (1834-63)\n(> 2000 cases pa)",
     "Enfans Trouves at Petersburg\n(1845-59, 1000-2000 cases pa)",
     "Pesth (500-1000 cases pa)",
     "Edinburgh (200-500 cases pa)",
     "Frankfort (100-200 cases pa)", "Lund (< 100 cases pa)")
hosp <- factor(hosp, levels=hosp[order(deathrate)])
dotplot(hosp ~ deathrate, xlim=c(0,110), cex=1.5,
        scale=list(cex=1.25), type=c("h","p"),
        xlab=list("Death rate per 1000 ", cex=1.5),
        sub="From Nightingale (1871) - data from Dr Le Fort")
```

[4]Data are from Nightingale, F. (1871): *Notes on Lying-in Institutions.*

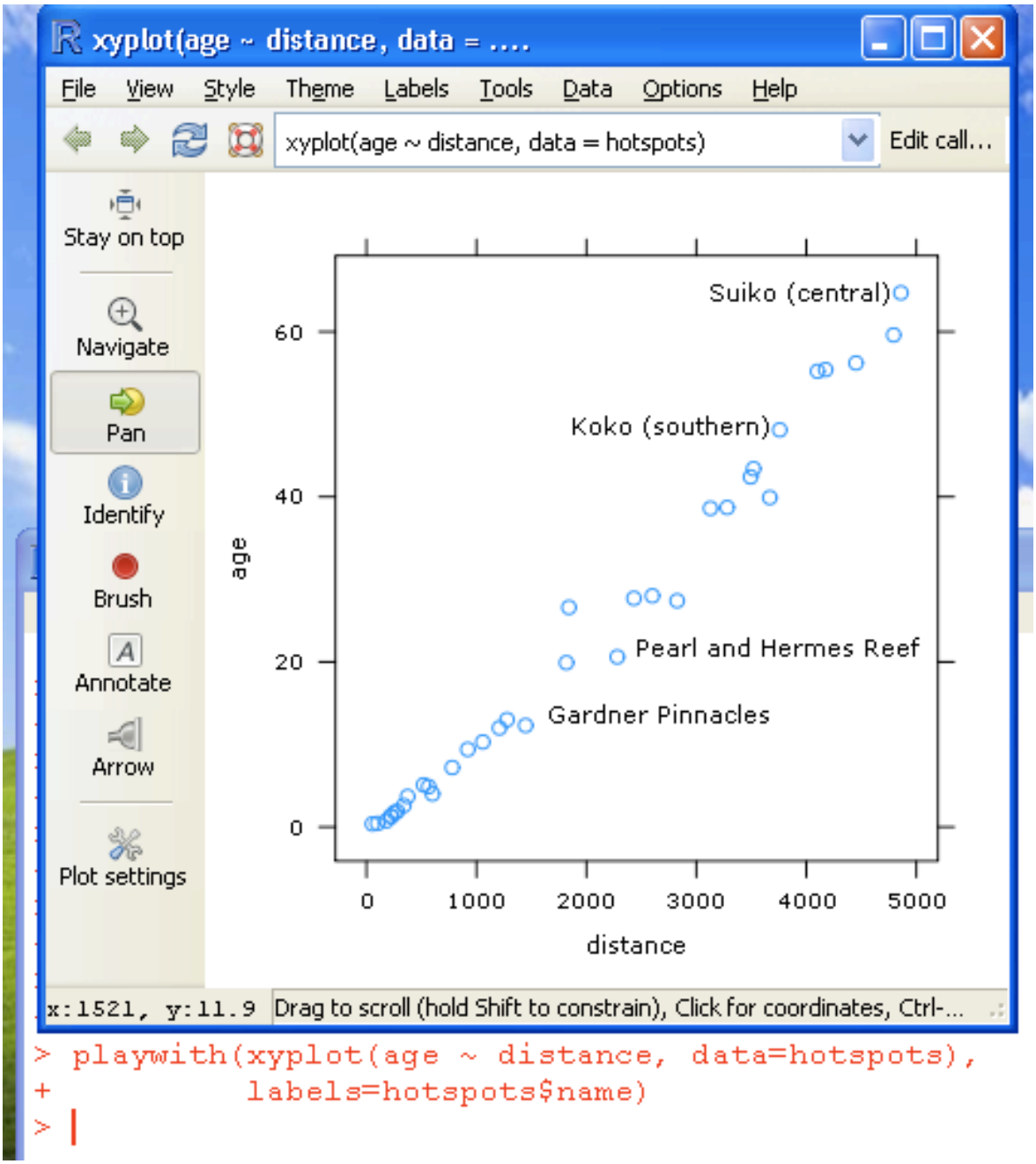

Figure 3.4: This playwith GUI window was generated by wrapping the call to `xyplot()` in the function `playwith()`, then clicking on Identify. Click near to a point to see its label. A second click adds the label to the graph.

Chapter 4

The *ggplot2* Package

This package, by Hadley Wickham, implements the graphics language that is described in Wilkinson's *Grammar of Graphics*. A draft of Hadley Wickham's book that describes the package is available from `http://had.co.nz/ggplot2/`. In contrast to base graphics, the syntax is consistent. It is much less stylized than *lattice*, and accordingly easier to adapt.

The examples that are given here will use the wrapper function `qplot()` (quickplot) that is designed for creating simple ggplot graphics objects. It has remarkably wide-ranging abilities.

4.1 Examples

Australian rain data

Figure 4.1 plots annual rainfall for South-East Australia.

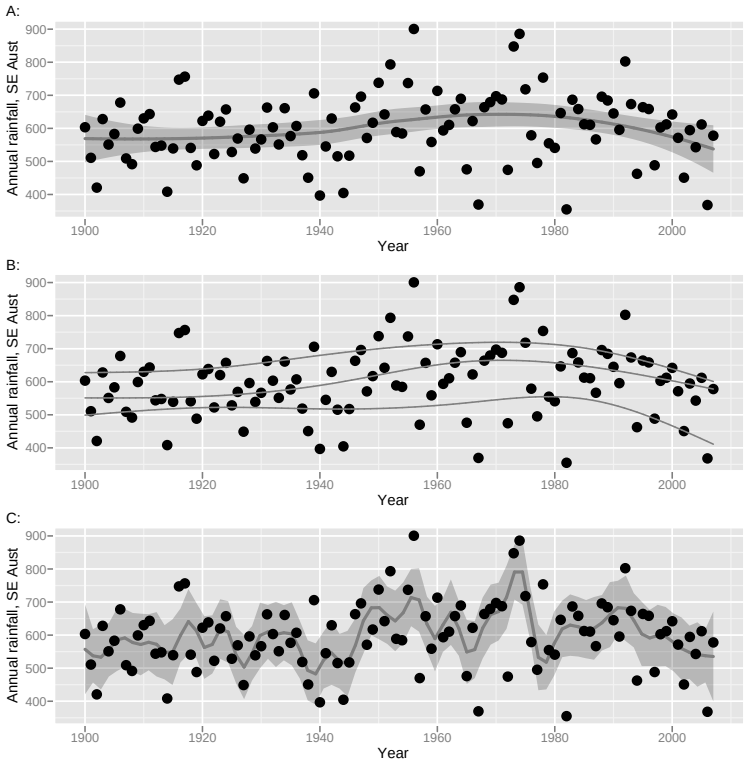

Figure 4.1: Annual rainfall, from 1901 to 2008, for the Murray-Darling basin region of Australia. The curve is fitted using the default loess smoother. The pointwise standard error bands assume that errors about the curve are independent; this is unlikely to be strictly true. To suppress the bands, specify `se=FALSE`.

Here is the code.

```
library(DAAGxtras)
library(ggplot2)
## Default loess smooth, with SE bands added.
quickplot(Year, seRain, data=bomregions, geom=c("point","smooth"),
          span=0.1, xlab="", ylab="Av. rainfall, M-D basin")
```

Try also the following:

```
library(splines)
quickplot(Year, seRain, data=bomregions, geom=c("point", "quantile"),
          formula = y ~ ns(x,5), quantiles=c(0.2,0.5,0.8) )
```

The normal spline basis `ns(x,5)` is supplied to the function that fits the quantiles, so that 5 d.f. spline curves are fitted at the 20%, 50% and 80% quantiles.

Physical measurements of Australian athletes

Figure 4.2 plots height against weight, for the `ais` data. Boxplots that show the distributions of heights, and two-dimensional density contour estimates have been added. Figure 4.2 shows boxplots (`geom="boxplot"`), by `sport` (given as the x-variable) and `sex` (separate panels): The code is:

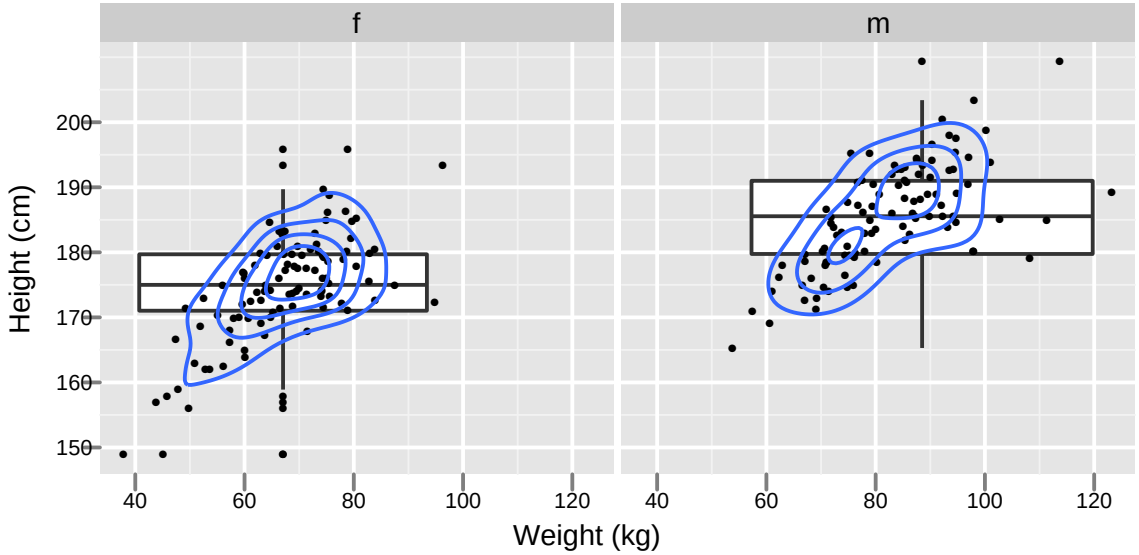

Figure 4.2: Height versus weight, by sex, for Australian athletes in the `ais` data set. Boxplots that show the distributions of heights, and two-dimensional density contours have been added.

```
## Overlay scatterplots with boxplots and with density contours
quickplot(wt, ht, xlab="Weight (kg)", ylab="Height (cm)", data=ais,
          geom=c("boxplot", "point", "density2d"),
          facets = . ~ sex)
```

The `facets` argument takes the form `row.var col.var`, where `row.var` indexes the rows of panels, `col.var` indexes columns. and `.` is used as a placeholder when there is one row or one column only.

Try also:

```
## Two panels (columns): sexes have different colors and symbols
aisBS <- subset(ais, sport %in% c("Row","Swim"))
```

```
aisBS$sport <- factor(aisBS$sport)
quickplot(wt, ht, data=aisBS, type="point",
          colour=sex, shape=sex,
          facets = . ~ sport)
## Single panel: distinguish sexes by colors; sports by symbols
quickplot(wt, ht, data=aisBS, type="point",
          colour=sex, shape=sport)
```

To get different colours for different levels of `sport`, specify `colour=sport`, For different plotting symbols, specify `shape=sport`. For different sizes of symbol, specify `size=sport`. Two of these may appear together. For example:

```
quickplot(wt, ht, data=aisBS, type="point",
          colour=sex, shape=sex, size=I(2.5),
          facets = . ~ sport)
## Single panel: distinguish sexes by colors; sports by symbols
quickplot(wt, ht, data=aisBS, type="point",
          colour=sex, shape=sport, size=I(2.5))
## Single panel: distinguish sexes by colors; sports by symbol size
quickplot(wt, ht, data=aisBS, type="point",
          colour=sex, size=sport)
```

Possible choices of `geom`, additional to those already demonstrated, are `"path"` (join points), `"line"` (join points), `"histogram"`, and `"density"`.

Note the difference between:

`size=I(2.5)`,used to make points somewhat larger than otherwise.

`size=2.5`, which specifies a mapping from the vector with the single element 2.5 to all points in the data. This does change the point size, but it adds adds an extraneous key. (This is analagous to the use of `size=sport` to specify a mapping from `sport` to `size`.)

Thus also, to make all points red, specify `color=I("red")`, not `color="red"`

Note also the following:

```
## Change the base pointsize for text to 8
theme_set(theme_gray(base_size=8))  # Gray theme
theme_set(theme_bw(base_size=8))    # Black and white theme
## Modify the pointsize
update_geom_defaults("point", aes(cex=1.25))
```

The function `aes()` generates aesthetic mappings. These map variables in the data to visual properties (aesthetics) of geoms.

Consult the web page `http://had.co.nz/ggplot/` for up to date information on *ggplot2*.

4.2 Dynamic Graphics – the *rgl* package

This provides three-dimensional dynamic graphics. Try the following code. It uses functions in the *Rcmdr* package – `scatter3d()` and `identify3d()`. These may be more convenient for novices than the *rgl* functions that they call.

```
## The Rcmdr and rgl packages must be installed
library(Rcmdr)        # This makes scatter3d() available
## The call to open3d() is optional, but see below
open3d()
par3d(cex=0.6)        # Optional.  Requires an rgl device to be open
with(nihills, scatter3d(x=log(dist), y=log(climb), z=log(time),
```

```
                          grid=FALSE, surface.col="gray",
                          point.col="black", axis.scales=FALSE))
with(nihills, identify3d(x=log(dist), y=log(climb), z=log(time),
                         labels=row.names(nihills), col="gray40"))
## NB: Use the middle or right mouse button to drag a rectangle
## around any pony that is to be labeled.
```

Following the call to `identify3d()`, use the middle (or maybe right) mouse button to drag a rectangle around any point that is to be labeled. To cease identifying points, make a middle (or right) click on an empty region of the plot. Use `rgl.snapshot()` to save the current plot into a file.

Chapter 5

References and Bibliography

5.1 Books and Papers on R

Crawley, M.J. 2005. Statistics – An Introduction with R. Wiley.

Crawley, M.J. 2007. The R Book. Wiley.

Dalgaard, P. 2002. Introductory Statistics with R. Springer-Verlag, New York.
[An excellent R-based introductory statistics text]

Fox, J. 2002. An R and S-PLUS Companion to Applied Regression. Sage Books.
(web page `http://socserv.socsci.mcmaster.ca/jfox/Books/Companion/index.html`)
[This is particularly aimed at classical types of regression calculations.]

Kuhnert, P. and Venables, W. 2005. An Introduction to R: Software for Statistical Modelling & Computing. CSIRO Australia. Available from
`http://cran.r-project.org/doc/contrib/Kuhnert+Venables-R_Course_Notes.zip`

Ihaka, R. & Gentleman, R. 1996. R: A language for data analysis and graphics. Journal of Computational and Graphical Statistics 5: 299-314.

Maindonald, J. H. & Braun, J. B. 2007. Data Analysis & Graphics Using R. An Example-Based Approach. Cambridge University Press, Cambridge, UK, 2007.
(web page `http://www.maths.anu.edu.au/~johnm/r-book.html`
[This is aimed at researchers who have had some previous exposure to statistics, and at applied statisticians.]

Venables, W.N. and Ripley, B.D. 2000. S Programming. Springer-Verlag, New York.
[This treats both R and S-PLUS.]

Venables, W.N. and Ripley, B.D., 4^{th} edn 2002. Modern Applied Statistics with S. Springer.
[This demands a relatively high level of sophistication. This treats both R and S-PLUS.]

5.2 Web-Based Information

See Documentation on the web page `http://www.r-project.org`

Note the R Wiki (`http://wiki.r-project.org/rwiki/doku.php`) and the help information listed under Other (`http://www.r-project.org/other-docs.html`).

For examples of R graphs, see `http://addictedtor.free.fr/graphiques/`.

R News: Successive issues of *R News* contain much useful information. These can be copied down from one of the CRAN sites.

Contributed Documentation: There is an extensive collection of user-written documents on R that can be accessed by going to this same mirror site, and clicking (under Documentation) on **Contributed**. See also the links that John Fox gives on the web page for his book that is noted under the reference for his book.

Books: See `http://www.R-project.org/doc/bib/R.bib` for a list that is updated regularly.

5.3 Graphics

Cleveland, W. S. 1985. The Elements of Graphing Data. Wadsworth, Monterey, California.

Chen, C., Hrdle, W. and Unwin A. 2008. Handbook of Data Visualization. Springer, in press.

Maindonald J H 1992. Statistical design, analysis and presentation issues. New Zealand Journal of Agricultural Research 35: 121-141.

Murrell, P. 2005. R Graphics. Chapman & Hall/CRC.
`http://www.stat.auckland.ac.nz/~paul/RGraphics/rgraphics.html.`
[This is a detailed exposition of the R graphics systems, with examples of their use.]

Tufte, E. R. 1983. The Visual Display of Quantitative Information. Graphics Press, Cheshire, Connecticut, U.S.A.

Tufte, E. R. 1990. Envisioning Information. Graphics Press, Cheshire, Connecticut, U.S.A.

Tufte, E. R. 1997. Visual Explanations. Graphics Press, Cheshire, Connecticut, U.S.A.

Wainer, H. 1997. Visual Revelations. Springer-Verlag, New York

Large and Possibly Sparse Data

Go to the website `http://user2007.org/program/`. Scroll down to "Large data and Programming Competition Winners".

Unwin, A., Theus, M. and Hofmann, H. 2006. Graphics of Large Datasets. Springer, NY 2006

Literature on trellis (lattice) graphics

Cleveland, W. S. 1993. Visualizing Data. Hobart Press, Summit, New Jersey.

Sarkar, D. 2008. Lattice. Multivariate Data Visialization with R. Springer.
[This is the definitive reference on Lattice graphics.]

The grammar of graphics in R (`ggplot2`)

Wilkinson, L. 2005. The Grammar of Graphics. Springer, 2005.

Index of Functions