Lattice and Other Graphics in R

J H Maindonald

Centre for Mathematics and Its Applications
Australian National University.

June 9, 2008

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

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

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:

```
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 hand corner. If thus closed, enter `Commander()` to reopen it again later in the session.

If you installed from the Windows menu, you are likely to be missing some of the suggested packages, needed to use some of the R commander's features. If so, then upon starting the R commander, you will be asked whether you want to install them.

You may be asked, when you start the R commander, whether you want to install any missing packages, required if you are to use all of the R commander's features. (If you installed from the Windows menu, you are likely to be missing some of the suggested packages.)

From GUI to writing code: The R commander displays the code that it generates. Thus, users can take this code and modify it. Even if the R commander does not do quite what is wanted, it may be possible to use R commander to generate relevant code, which can then be modified.

The active data set: The R commander has the notion of an active data set. Here are alternative ways in which a data set can be made 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.
- Also possible is the loading of data from an R image (**.RData** or **.rda**) file; click on Load data set ...

Creating graphs: To draw graphs

Start by clicking 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.

Often, R commander can be used to give a rough approximation to the graph that is required. Modification of the code that R commander generates may then provide the required graph.

Statistics (& fitting models): Click on the Statistics drop down menu to get summary statistics and/or carry out variou statistical tests. Also, click here to fit models.

***Models:** Click here to extract information from model objects once they have been fitted. To fit the model in the first place, 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.

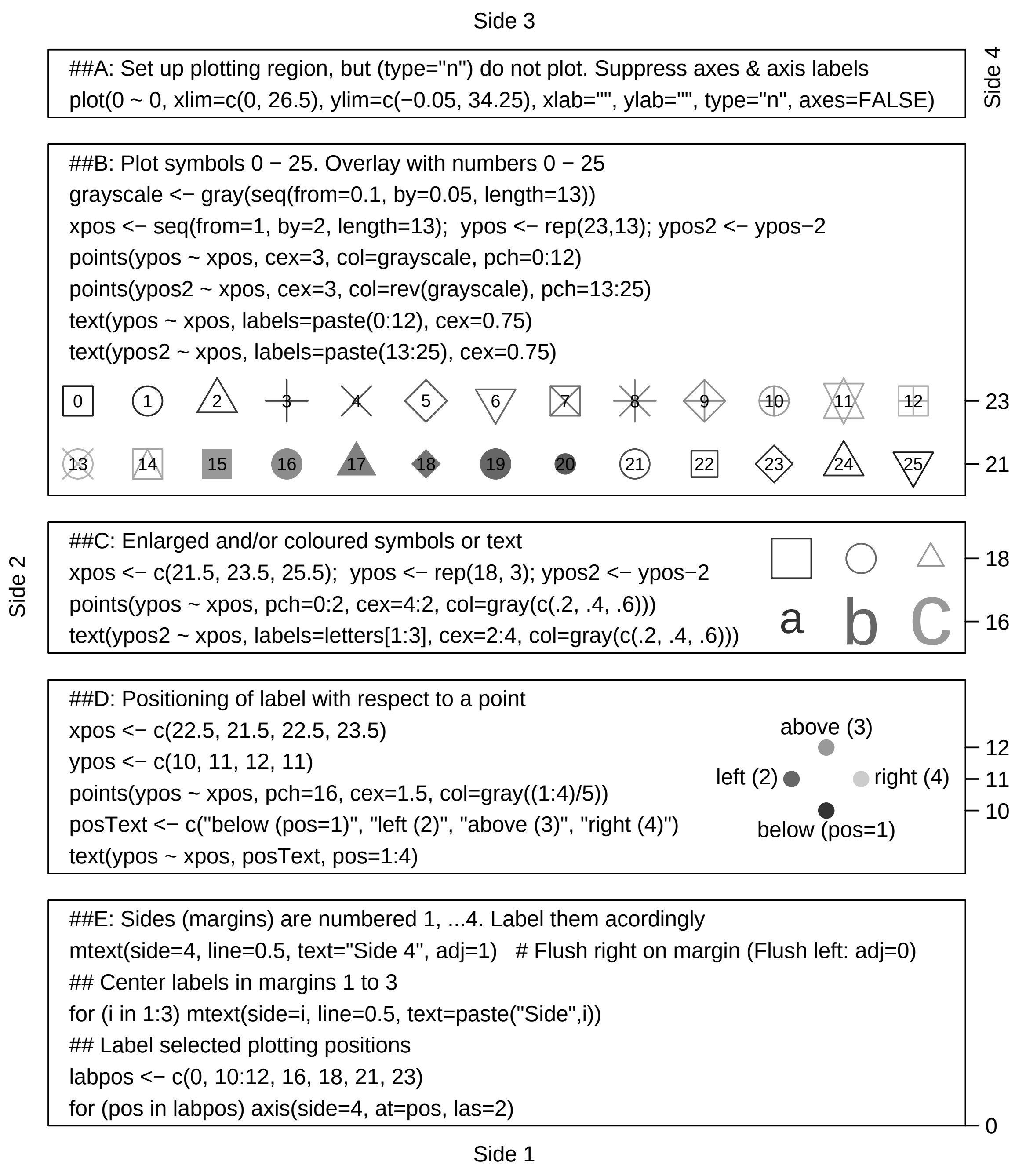
```
library(DAAG)
attach(elasticband)  # R can now find distance & stretch
plot(distance ~ stretch)
plot(ACT ~ year, data=austpop, type="l")
plot(ACT ~ year, data=austpop, type="b")
detach(elasticband)
```

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

Users who execute the code as it stands 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))
```

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



```
##A: Set up plotting region, but (type="n") do not plot. Suppress axes & axis labels
plot(0 ~ 0, xlim=c(0, 26.5), ylim=c(−0.05, 34.25), xlab="", ylab="", type="n", axes=FALSE)
```

```
##B: Plot symbols 0 − 25. Overlay with numbers 0 − 25
grayscale <− gray(seq(from=0.1, by=0.05, length=13))
xpos <− seq(from=1, by=2, length=13);  ypos <− rep(23,13); ypos2 <− ypos−2
points(ypos ~ xpos, cex=3, col=grayscale, pch=0:12)
points(ypos2 ~ xpos, cex=3, col=rev(grayscale), pch=13:25)
text(ypos ~ xpos, labels=paste(0:12), cex=0.75)
text(ypos2 ~ xpos, labels=paste(13:25), cex=0.75)
```

```
##C: Enlarged and/or coloured symbols or text
xpos <− c(21.5, 23.5, 25.5);  ypos <− rep(18, 3); ypos2 <− ypos−2
points(ypos ~ xpos, pch=0:2, cex=4:2, col=gray(c(.2, .4, .6)))
text(ypos2 ~ xpos, labels=letters[1:3], cex=2:4, col=gray(c(.2, .4, .6)))
```

```
##D: Positioning of label with respect to a point
xpos <− c(22.5, 21.5, 22.5, 23.5)
ypos <− c(10, 11, 12, 11)
points(ypos ~ xpos, pch=16, cex=1.5, col=gray((1:4)/5))
posText <− c("below (pos=1)", "left (2)", "above (3)", "right (4)")
text(ypos ~ xpos, posText, pos=1:4)
```

```
##E: Sides (margins) are numbered 1, ...4. Label them acordingly
mtext(side=4, line=0.5, text="Side 4", adj=1)   # Flush right on margin (Flush left: adj=0)
## Center labels in margins 1 to 3
for (i in 1:3) mtext(side=i, line=0.5, text=paste("Side",i))
## Label selected plotting positions
labpos <− c(0, 10:12, 16, 18, 21, 23)
for (pos in labpos) axis(side=4, at=pos, las=2)
```

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.

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.

For information on possible settings for graphics parameters, see `help(par)`. Most parameters can be set either in a call to `par()` or in the function call. Some can only appear in a call to `par()` and others only in a function call.

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 2.2. Observe that an arbitrary character string can appear as a variable in an expression. The operator '*' juxtaposes the separate elements of the "expression".

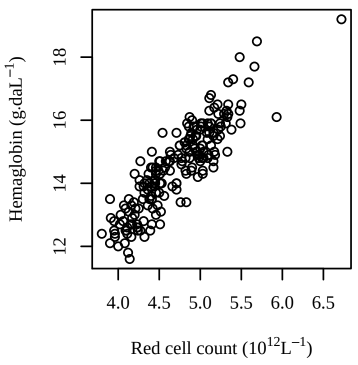


Figure 2.2: Hemaglobin concentration vs red cell count, for 202 Australian athletes. The SI symbol 'daL' is 'decaliters'.

```
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

Lattice Graphics:

Lattice	Lattice is a flavour of trellis graphics (the S-PLUS flavour was the original)
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.

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

```
library(lattice)
demo(lattice)
```

3.1 Lattice Graphics vs Base Graphics

Contrast the different ways that base and lattice graphics are designed to operate.

- In base graphics, a graphics page (possibly the first of a sequence of pages) opens when a device is opened. Plots then go to the page or pages. For a screen device, plots go to the screen. For a hardcopy device, plots usually go, in the first place, to a file.

- Lattice functions create trellis objects. 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()`.

The updating feature allows the graphics object to be built up in steps, or even modified. Additionally, abilities that were added relatively late in lattice's development make it possible to adds new features to the "printed" page, after a style of use that is common in base graphics.

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/`). For use in an R session, it must first be attached.

```
library(lattice)
```

Now compare:

```
plot(ACT ~ year, data=austpop)       # Base graphics
xyplot(ACT ~ year, data=austpop)     # Lattice graphics
```

In both cases, if these are typed from the command line, a graph is plotted. The reason 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 ouptut to the command line, the object is "printed", i.e., a graph appears.

The following makes this clear: The following makes clear the difference between the two functions:

```
invisible(plot(ACT ~ year, data=austpop))      # A graph is plotted
invisible(xyplot(ACT ~ year, data=austpop))    # Graph does appear
```

The function `invisible()` suppresses the command line printing. Hence `invisible(xyplot(...))` does not yield a graph.

Inside a function, `xyplot(...)` prints a graph only if it appears as the return value from the function, i.e. usually, as the final line. In a file that is sourced, 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))
```

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

- Different symbols and/or colors are used for different drinks, within the one panel. Different panels must then be used for different countries, as in Figure 3.1.[1] Or if different countries are shown in the same panel, then different panels must be used for the different drinks.
- Use a 3 drinks $\times$ 2 countries, or 2 countries $\times$ 3 drinks layout of panels.

Where plots are superposed in the one panel and, e.g. regression lines or smooth curves are fitted, this will be done separately for each different set of points. The separate sets may be distinguished by colour, and/or they may be distinguished by different symbols and/or line styles.

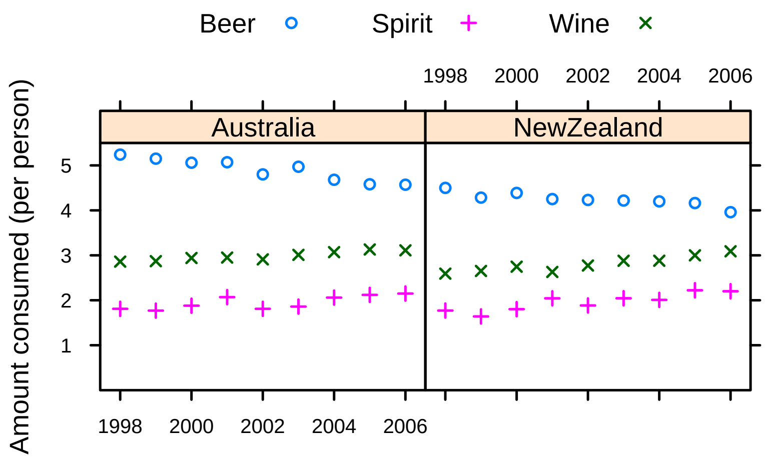


Figure 3.1: Australian and New Zealand apparent per person annual consumption (in liters) of the pure alcohol content of liquor products, for 1998 to 2006.

Simplified code for Figure 3.1 is:

```
xyplot(Beer+Spirit+Wine ~ Year | Country, data=grog, outer=FALSE,
       auto.key=list(columns=3))
```

To obtain various enhancements that give Figure 3.1, specify:

```
grogplot <-
  xyplot(Beer+Spirit+Wine ~ Year | Country, data=grog, outer=FALSE,
         auto.key=list(columns=3))
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`. With `outer=FALSE`, these appear in the same panel.
- Conditioning by country (`| Country`) gives separate panels for separate countries.
- The function `simpleTheme()` is convenient for setting or changing point and line settings.

[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 of `Beer`, `Wine` and `Spirit`. 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.

[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)
```

The plot superimposes the separate plots (two panels each):

```
xyplot(Beer ~ Year | Country, data=grog)     # Plot for Beer
xyplot(Spirit ~ Year | Country, data=grog)   # Plot for Spirit
xyplot(Wine ~ Year | Country, data=grog)     # Plot for Wine
```

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 of the syntax:

	Overplot (a single panel)	Separate panels (conditioning)
Levels of a factor	`Beer ~ Year, groups=Country`	`Beer ~ Year	Country`
	`Beer+Wine+Spirit ~ Year,`	
Columns in parallel	`outer=FALSE`	`outer=TRUE`

3.3 Lattice Parameters and Graphics Features

Lattice parameter settings

1. Changes to the defaults for points and lines are most easily made using the function `simpleTheme()` (in recent versions of *lattice*).
2. Axis, axis tick, tick label and axis label parameters are conveniently set using the parameter `scales` in the function call.
3. Lattice objects can be created, then updated to incorporate changes to parameter settings.
4. Note also the parameters `aspect` (aspect ratio) and `layout` (# rows × # columns × # pages).

3.3.1 Point, line and fill color settings

Lattice point, line & related settings
First use `simpleTheme()` to create a "theme" with the new settings:

```
miscSettings <- simpleTheme(pch = c(1,3,4), cex=1.25)
```

Alternatives are then:

(i) Supply the "theme" to `par.settings` in the function call.

```
xyplot(Beer+Spirit+Wine ~ Year | Country, outer=FALSE,
       auto.key=list(columns=3), data=grog,
       par.settings=miscSettings)
```

[This stores the settings with the object. These stored settings over-ride the global settings at the time of printing.]

(ii) Supply the "theme" to `trellis.par.set()`, prior to plotting:

```
trellis.par.set(miscSettings)
xyplot(Beer+Spirit+Wine ~ Year | Country, outer=FALSE,
       auto.key=list(columns=3), data=grog)
```

[Makes the change globally, until a new trellis device is opened]

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; or (iii) in the argument **`theme`** to `trellis.device()`.

Note the use of the function `trellis.device()` to open a new graphics device. By default, it has `retain=FALSE`, so that parameters are reset to their defaults for the relevant device.

The following changes the plotting symbols to symbols 1, 3 and 4, as in Figure 3.1. It also increases the size of points by 25%:

```
miscSettings <- simpleTheme(pch = c(1,3,4), cex=1.25)
xyplot(Beer+Spirit+Wine ~ Year | Country, outer=FALSE,
       auto.key=list(columns=3), data=grog, ylim=c(0,5.5),
       par.settings=miscSettings,
       xlab="", ylab="Alcohol consumption (per person)")
```

Where there are a small number of points, it can be helpful to show them as large solid dots. The following affects all subsequent plots, until changed or until a new device is opened:

```
trellis.par.set(simpleTheme(pch = 16, cex=2))
```

When there are a large number of points, it may be helpful to set the background transparency `alpha` (c.f., also, `alpha.points` and `alpha.line`) to a value less than 1, so that regions where there are many overlapping points can be readily identified.

Where changes 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"
```

The settings that are of interest can then be inspected individually. Section 14.12 of DAAGUR has brief details. 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. Note that there are separate settings for text and symbols:

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

3.3.2 Parameters that affect axes, tick marks, and axis labels

Axis, tick, tick label and axis label settings – the `scales` argument

- Tick positions and tick labels:

```
jobplot <- xyplot(Ontario+BC ~ Date, data=jobs)
## Half-length ticks, each quarter, Label years, Add key
    tpos <- seq(from=95, by=0.25, to=97)
    tlabs <- rep(c("Jan95", "", "Jan96", "", "Jan97"),
                   c(1,3,1,3,1))
update(jobplot, auto.key=list(columns=2), xlab="",
       scales=list(tck=0.5, x=list(at=tpos, labels=tlabs)))
```

A logarithmic scale, and/or `relation="sliced"`

- Use a Logarithmic scale (here, natural logarithmic)

```
logplot <- xyplot(Ontario+BC ~ Date, data=jobs, outer=TRUE,
                  xlab="", scales=list(y=list(log="e")))
```

- Slice the scale

```
update(logplot, scales=list(y=list(relation="sliced")))
```

Scales may have `relation="fixed"`, or `relation="sliced"`, or `relation="free"`

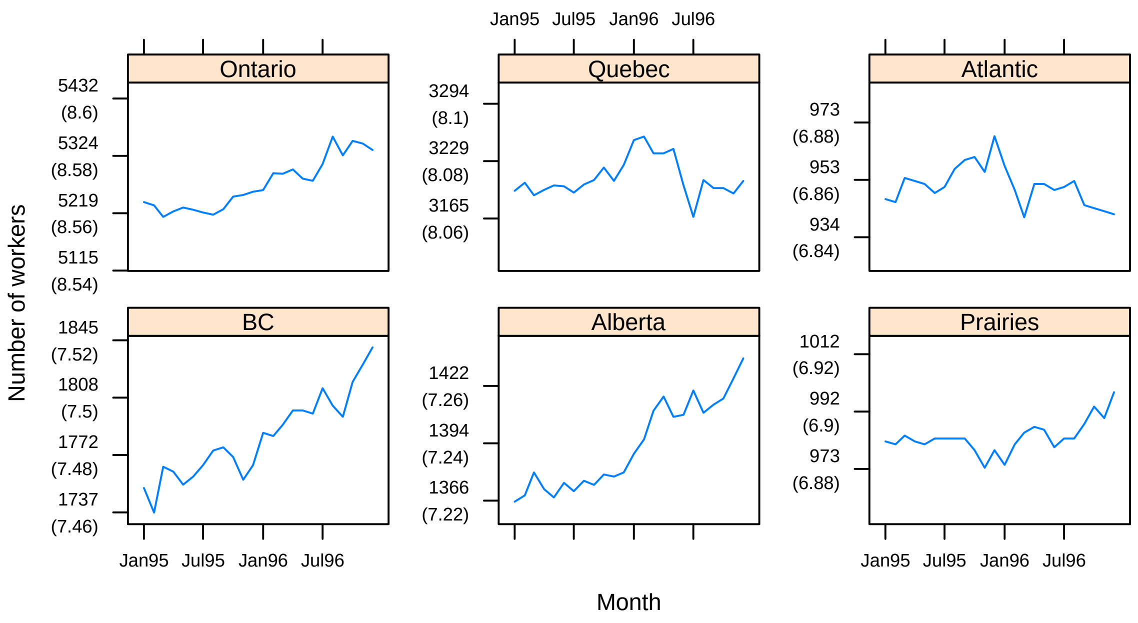
The data frame `jobs` (*DAAG*) has numbers in the Canadian labour force, for each of six different regions, by month over January 1995 to December 1996. The regions appear as columns of the data frame `jobs`. The following plots these in parallel, on the one panel:

```
xyplot(Ontario+Quebec+BC+Alberta+Prairies+Atlantic ~ Date, data=jobs,
       ylab="Number of jobs", type="b", outer=FALSE,
       auto.key=list(space="right", lines=TRUE))
```

To get a good visual indication of relative changes over time, however, a "sliced" logarithmic scale is needed. The following saves for later enhancement a simplified of the plot shown in Figure 3.2, giving it the name `jobs.xyplot`:

```
## Save the graphics object, for later updating
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)
```

Figure 3.2: Labor force numbers (1000s) for various regions of Canada. Labels on the vertical axis show both numbers and $\log_e$ of numbers. Distances between ticks are 0.02 on the $\log_e$ scale, i.e., a change of almost exactly 2%.



The sliced scale gives each panel the slice of the scale that is needed for points in that panel. A logarithmic scale makes equal relative changes equidistant.

The labeling can be greatly improved. In Figure 3.2 the y-axis label shows number of jobs, with the logarithms of the numbers given in parentheses. Additionally, dates of the form `Jan95` label the x-axis. In the following code, observe the use of `update()` to specify tick positions and tick labels for the graphics object `jobs.xyplot`.

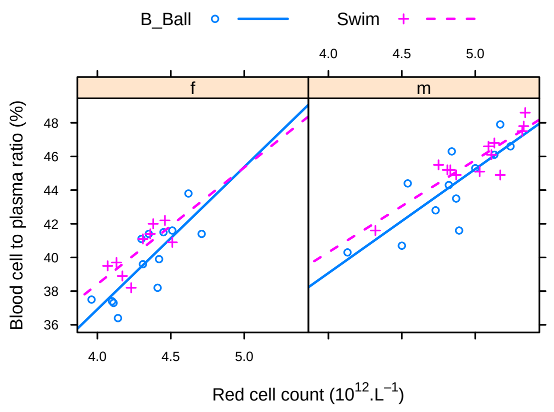
```
ylabpos <- exp(pretty(log(unlist(jobs[,-7])), 100))
ylabels <- paste(round(ylabpos),"\n(", log(ylabpos), ")", sep="")
## Create a date object 'startofmonth'; use this instead of 'Date'
atdates <- seq(from=95, by=0.5, length=5)
datelabs <- format(seq(from=as.Date("1Jan1995", format="%d%b%Y"),
                   by="6 month", length=5), "%b%y")
update(jobs.xyplot, xlab="", between=list(x=0.5, y=1),
       scales=list(x=list(at=atdates, labels=datelabs),
                   y=list(at=ylabpos, labels=ylabels), tck=0.6) )
```

Notice the use of `between=list(x=0.5, y=1)` to add horizontal and vertical space between the panels. The addition of extra vertical space ensures that the tick labels do not overlap. Specifying `tck=0.6` reduces the length of axis ticks to 60% of the default. This can be a vector of length 2.

3.3.3 A further example

The dataset `ais` (*DAAG*) has a number of physical and biological measurements on 202 athletes at the Australian Insitute of Sport. See `help(ais)` for details of the measurements, which include a variety of blood cell counts. Data are for ten different sports. Here is a breakdown of numbers:

```
> with(ais, table(sex,sport))
   sport
```



```
aisBS <- subset(ais,
  sport %in% c("B_Ball", "Swim"))
basic.xyplot <-
  xyplot(hg ~ rcc | sex,
         groups=sport[drop=TRUE],
         data=aisBS)
## Simplified Code
update(basic.xyplot,
       type=c("p","r"),
       auto.key=list(lines=TRUE,
       columns=2))
# For a smooth curve, specify
# type=c("p","smooth")
```

Figure 3.3: Blood cell to plasma ratio (`hc`) versus red cell count (`rcc`), by `sex` (different panels) and `sport` (distinguished within each panel). The argument `type=c("p", "r")` displays both points (`"p"`) and regression lines (`"r"`).

```
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
```

The data were collected with the aim of examining possible differences in blood characteristics, between athletes in endurance-related events and those in power-related events.

Figure 3.3 plots blood cell to plasma ratio (%) against red cell count, for two sports only. The two sports appear within the one panel, distinguished by different symbols and/or colours. Females and males appear in separate panels. Figure 3.3 shows a suitable plot, adding also regression lines. Again, note the creation of an initial basic plot, which is then updated. Code is:

```
basic1 <-  xyplot(hc ~ rcc | sex, groups=sport[drop=TRUE],
                  data=subset(ais, sport %in% c("B_Ball", "Swim")),
                  xlab="", ylab="Blood cell to plasma ratio (%)")
basic2 <- update(basic1, par.settings=simpleTheme(pch = c(1,3),
                 scales=list(tck=0.5), lty=1:2, lwd=1.5))
update(basic2, type=c("p", "r"), auto.key=list(columns=2, lines=TRUE),
       xlab=expression("Red cell count (10"^{12}*"."*L^{-1}*")"))
```

Again, there are details that require explanation:

- `type=c("p","r")` gives both points and fitted regression lines.
- In `groups=sport[drop=TRUE]`, the `drop=TRUE` is optional. If not included there will be one legend item for each of the 10 sports. Subsetting a factor leaves the levels attribute unchanged, even if some of the levels are no longer present in the data.
- As in base graphics, graphical annotation (tick labels, axis labels, labels on points, etc.) can be given using the function `expression()`. In this context,“expression” is broadly defined; thus expressions can have terms that are character strings. See `help(plotmath)`.

3.3.4 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 `text` is supplied as a parameter, `levels(groups)` provides the legends.

When updating, use `legend=NULL` to remove an existing key, prior to adding a different key.

3.4 Panel Functions and Interaction with Plots

Further flexibility is added, in the creation of plots, by the use of a user's own panel functions. A number of panel functions are provided that can be incorporated. A further possibility is interaction with the panels, or other graphics features, of a graph that has just been printed.

3.4.1 Panel functions

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()`.

The following gives a version of Figure 3.3 in which the lines for the two sports are parallel:

```
xyplot(hg ~ rcc | sex, groups=sport[drop=TRUE], data=aisBS,
    auto.key=list(lines=TRUE, columns=2), aspect=1,
    strip=strip.custom(factor.levels=c("Female","Male")),
      # In place of level names c("f", "m"), use c("Female", "Male")
    panel=function(x, y, groups, subscripts, ...){
         panel.superpose(x,y, groups=groups,
                      subscripts=subscripts, ...)
         b <- coef(lm(y ~ groups[subscripts] + x))
         lcol <- trellis.par.get()$superpose.line$col
         lty <- trellis.par.get()$superpose.line$lty
         panel.abline(b[1], b[3], col=lcol[1], lty=lty[1])
         panel.abline(b[1]+b[2], b[3], col=lcol[2],
                      lty=lty[2])
         })
```

When there are groups within panels, `panel.xyplot()` calls `panel.superpose()`. The customized panel function has the structure:

```
  panel=function(x, y, groups, subscripts, ...){
       panel.superpose(x,y, groups=groups,
                    subscripts=subscripts, ...)
```

```
        . . . .
        panel.abline(b[1], b[3], col=lcol[1], lty=lty[1])
        panel.abline(b[1]+b[2], b[3], col=lcol[2],
                     lty=lty[2])
        })
```

As the function `panel.superpose()` lacks an option to fit and display multiple parallel lines, the user must supply the needed code. The following calculates the regression slope estimates:

```
     b <- coef(lm(y ~ groups[subscripts] + x))
       # NB: groups (but not x and y) must be subscripted
```

The lines are then drawn one at a time, taking care that the line settings agree with those that will appear in the key.

A further enhancement (omitted from the above code) adds axis labels, using an expression for the x-axis label.

```
    xlab=expression("Red cell count (10"^{12}*"."*L^{-1}*")")
    ylab="Blood cell to plasma ratio (%)"
```

3.4.2 Interaction with Lattice Plots

Focusing and unfocusing:

- Following the plot, call `trellis.focus()`.
- In a multi-panel display, click on a panel to select it.
- Use functions such as `panel.points()`, `panel.text()`, `panel.abline()`, `panel.identify()`.
- Call `trellis.focus()`, as needed, to switch panels.
- When finished, call `trellis.unfocus()`.

For non-interactive use of `trellis.focus()`, turn off highlighting, i.e., the call becomes `trellis.focus(highlight=FALSE)`.

Use the call `trellis.panelArgs()` to identify the arguments that are available to panel functions following a call to `trellis.focus()`.

Viewports:
A lattice plot is made up of a number of "viewports". In the call to `trellis.focus()`, the default is `name="panel"`.

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

Here is an example of the interactive labeling of points:

```
xyplot(log(Time) ~ log(Distance), groups=roadORtrack,
       data=worldRecords)
trellis.focus()
## Now click (maybe twice) on a panel
panel.identify(labels=worldRecords$Place)
## Click near to points that should be labeled
## Right click to terminate
trellis.unfocus()
```

Lattice is built on top of the `grid` package. This implements `viewports`, which are arbitrary rectangular regions within which plotting takes place. In the course of plotting, the focus moves from one viewport to the next, as needed to build up the plot.

The function `trellis.focus()` can, once the printing is complete, be used to restore focus to a viewpoint within one of the current panels, or to the whole panel. Common choices for the parameter `name` are `"panel"` and `"strip"`, with `column` and `row` (by default, counting from the bottom up) identifying the column and row in the layout. For `name="legend"`; `side` must be indicated. The argument `name="toplevel"` gives access to the rectangular region within which the panels are placed.

For interactive use, the function `trellis.focus()` can be called without parameters. In a single panel display, this highlights the panel. In a multi-panel display, clicking on a panel will select it. The function `trellis.unfocus()` removes the highlighting and makes `"toplevel"` the current viewport.

Once the focus is on a panel, the user has access to the functions that were noted in the previous subsection, and to many others besides.

The functions `trellis.focus()` and `trellis.unfocus()` can be used in a non-interactive mode. The following prints the stripplot and a boxplot objects created in Subsection 3.5.1 one under the other on the same graphics page. Following the printing of each plot, the focus is placed on the top level viewport, and the function `grid.text()` (from the *grid* package) used to add a label:

```
library(DAAG)
library(grid)
## Positioning will be (xmin=0, ymin=0.46, xmax=1, ymax=1)
print(update(cuckoostrip, xlab=""), position=c(0, .46, 1 ,1))
trellis.focus("toplevel", highlight=FALSE)
grid.text("A", x=0.05, y=0.935, gp=gpar(cex=1.15))
trellis.unfocus()
print(cuckoobox, position=c(0, 0, 1, 0.54), newpage=FALSE)
trellis.focus("toplevel", highlight=FALSE)
grid.text("B", x=0.05, y=0.935, gp=gpar(cex=1.15))
trellis.unfocus()
```

Observe that the rectangular regions on which the objects are printed have been chosen so that they overlap somewhat, reducing the space between.

3.5 Displays of Distributions

3.5.1 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 a stripplot object and a boxplot object, for the `cuckoos` data (from *DAAG*):[3]

```
cuckoostrip <- stripplot(species ~ length, aspect=0.5,
                         xlab="Cuckoo egg length (mm)", data=cuckoos)
cuckoobox <- 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.
The following demonstrates the use of `dotplot()`:

```
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"))
```

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 quite 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")
```

3.5.2 Lattice Style Density Plots

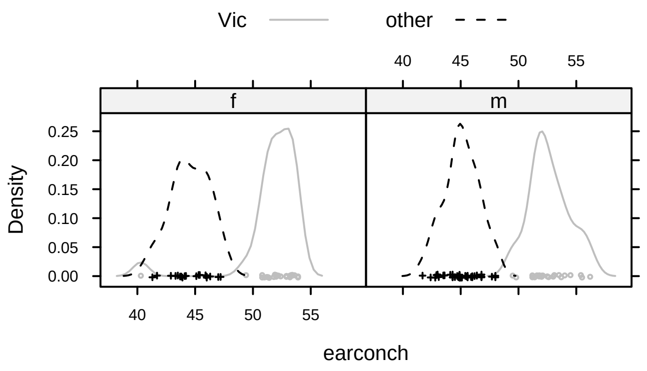


Figure 3.4: Lattice style density plot comparing possum earconch measurements, separately for males and females, between Victorian and other populations. Observe that the scatter of data values is shown along the horizontal axis.

`## For slightly improved labeling, precede with:`
[3]`levels(cuckoos$species) <- sub(".", " ", levels(cuckoos$species), fixed=T)`
[4]Data are from Nightingale, F. (1871): *Notes on Lying-in Institutions.*

Here is a density plot (Figure 3.4), 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. For histogram(), the groups argument is not available.

3.6 Lattice graphics functions – Further Points

3.6.1 Help on lattice functions

For an overview, type help(Lattice). For help on the graphical parameters used by lattice functions, see help(trellis.par.set) and help(simpleTheme). For other settings, see help(lattice.options).

Several of the help pages for lattice functions are shared between more than one function. For example, xyplot(), dotplot(), barchart(), stripplot() and bwplot() share the same help page. As a consequence. typing example(bwplot) has the same effect as typing example(xyplot).

3.6.2 Selected Lattice Functions

Functions that have already been demonstrated are xyplot(), stripplot(), dotplot(), densityplot() and bwplot(). Other ”high level” functions include:

```
barchart(character ~ numeric,..)
histogram( ~ numeric,..)       # NB: does not accept groups parameter
densityplot( ~ numeric,..)     # Density plot; does allow groups
bwplot(factor ~ numeric,..)    # Box and whisker plot
qqmath(factor ~ numeric,..)    # normal probability plots
## Bivariate
qq(factor ~ numeric, ...)      # comparing two empirical distributions
                               # (Two factor levels identify the 2 distns)
## Multivariate
cloud(numeric ~ numeric * numeric, ...)        # 3D surface
wireframe(numeric ~ numeric * numeric, ...)    # 3D scatterplot
levelplot(numeric ~ numeric * numeric, ...)    # cf image() in base graphics
contourplot(numeric ~ numeric * numeric, ...)  # contour plot
## Hypervariate
splom( ~ dataframe,..)         # Scatterplot matrix
parallel( ~ dataframe,..)      # Parallel coordinate plots
## Miscellaneous
rfs()               # Residual and fitted value plot (also see 'oneway')
tmd()               # Tukey Mean-Difference plot
```

In each instance, users can add conditioning variables.

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.

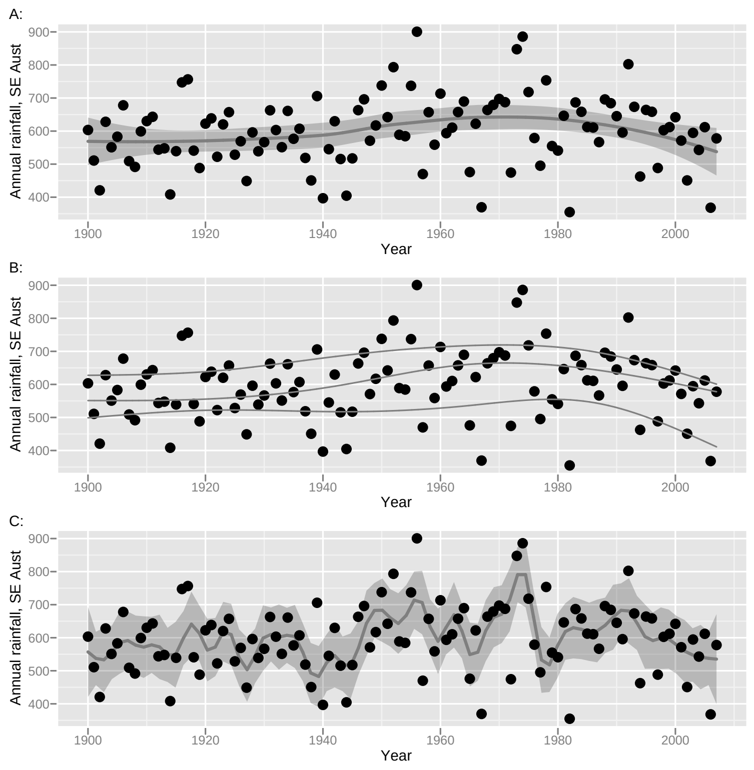


Figure 4.1: Annual rainfall, from 1901 to 2007, for South-East Australia. Curves are fitted thus: A: default loess smoother, with default smoothing parameter. B: 20%, 50% and 80% quantiles, based on 5 d.f. normal splines. C: robust 36 d.f. normal spline fit. Panel C is designed to show trends that are on the scale of an 11-year sunspot cycle.
The bands in (A) and (C) are designed to be pointwise one standard error bands. Because of likely sequential correlation, at least for the 5 d.f. curve, they should be regarded as rough indications only. To suppress the bands, specify `se=FALSE`.

Here is the code. Note however that, in Figure 4.1, the points for 2006 and 2007 have been added.

```
library(DAAG)
library(ggplot2)
## A: Default loess smooth
qplot(Year, seRain, data=bomsoi, geom=c("point","smooth"))
## B: 20%, 50% & 80% quantiles
##    5 d.f. normal splines
qplot(Year, seRain, data=bomsoi, geom=c("point", "quantile"),
      formula = y ~ ns(x,5), quantiles=c(0.2,0.5,0.8) )
## C: Robust fit using rlm()
##    15 d.f. normal splines
qplot(Year, seRain, data=bomsoi, geom=c("point", "smooth"),
      formula = y ~ ns(x,15), method="rlm")
```

Physical measurements of Australian athletes

Figure 4.2A plots height against weight, for the `ais` data. Two-dimensional density contour estimates have been added. Figure 4.2B shows boxplots (`geom="boxplot"`), by `sport` (given as the x-variable) and `sex` (separate panels):

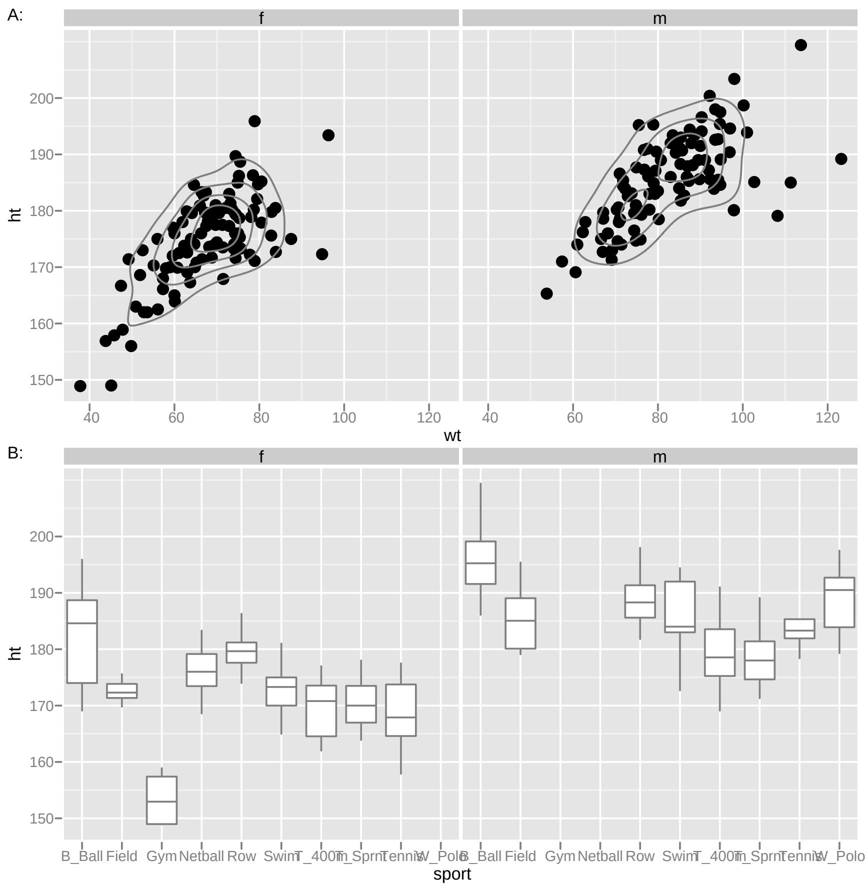


Figure 4.2: A: Height versus weight, for Australian athletes in the `ais` data set. Two-dimensional density contours have been added. B: Boxplots of heights of athletes, by `sport` and (in separate panels) by `sex`.

```
## A
qplot(wt, ht,
      data=ais,
      geom=c("point",
             "density2d"),
      facets = sex~.)
## B
qplot(sport, ht,
      data=ais,
      geom="boxplot",
      facets = sex~.)
```

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`.

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

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