4

Trellis Graphics: the Lattice Package

Chapter preview

This chapter describes how to produce Trellis plots using R. There is a description of what Trellis plots are as well as a description of the functions used to produce them. Trellis plots are designed to be easy to interpret and at the same time provide some modern and sophisticated plotting styles, such as multipanel conditioning.

The grid graphics system provides no high-level plotting functions itself, so this chapter also describes the best way to produce a complete plot using the grid system. There are several advantages to producing a plot using the grid system, including greater flexibility in adding further output to the plot, and the ability to interactively edit the plot.

This chapter describes the lattice package, developed by Deepayan Sarkar[54]. Lattice is based on the grid graphics system, but can be used as a complete graphics system in itself and a great deal can be achieved without encountering any of the underlying grid concepts.* This chapter deals with lattice as a self-contained system consisting of functions for producing complete plots, functions for controlling the appearance of the plots, and functions for opening and closing devices. Section 5.8 and Section 6.7 describe some of the benefits that can be gained from viewing lattice plots as grid output and dealing directly with the grid concepts and objects that underly the lattice system.

*To give Deepayan proper credit, lattice uses grid only to *render* plots. Lattice performs a lot of work itself to deconstruct formulae, rearrange the data, and manage many user-settable options.

The graphics functions that make up the lattice graphics system are provided in an add-on package called `lattice`. The lattice system is loaded into R as follows.

```
> library(lattice)
```

The lattice package implements the Trellis Graphics system[6] with some novel extensions. The Trellis Graphics system has a large number of sophisticated features and many of these are described in this section, but more information, examples, and background are available from the Trellis Display web site:

`http://cm.bell-labs.com/cm/ms/departments/sia/project/trellis/index.html`

4.1 The lattice graphics model

In simple usage, lattice functions appear to work just like traditional graphics functions where the user calls a function and output is generated on the current device. The following example plots the locations of 1000 earthquakes that have occurred in the Pacific Ocean (near Fiji) since 1964 (see Figure 4.1).*

```
> xyplot(lat ~ long, data=quakes, pch=".")
```

It is perfectly valid to use lattice this way; however, lattice graphics functions do not produce graphical output directly. Instead they produce an object of class `"trellis"`, which contains a description of the plot. The `print()` method for objects of this class does the actual drawing of the plot. This can be demonstrated quite easily. For example, the following code creates a `trellis` object, but does not draw anything.

```
> tplot <- xyplot(lat ~ long, data=quakes, pch=".")
```

The result of the call to `xyplot()` is assigned to the variable `tplot` so it is not printed. The plot can be drawn by calling print on the `trellis` object (the result is exactly the same as Figure 4.1).

```
> print(tplot)
```

*The data are available as the data set `quakes` in the `datasets` package.

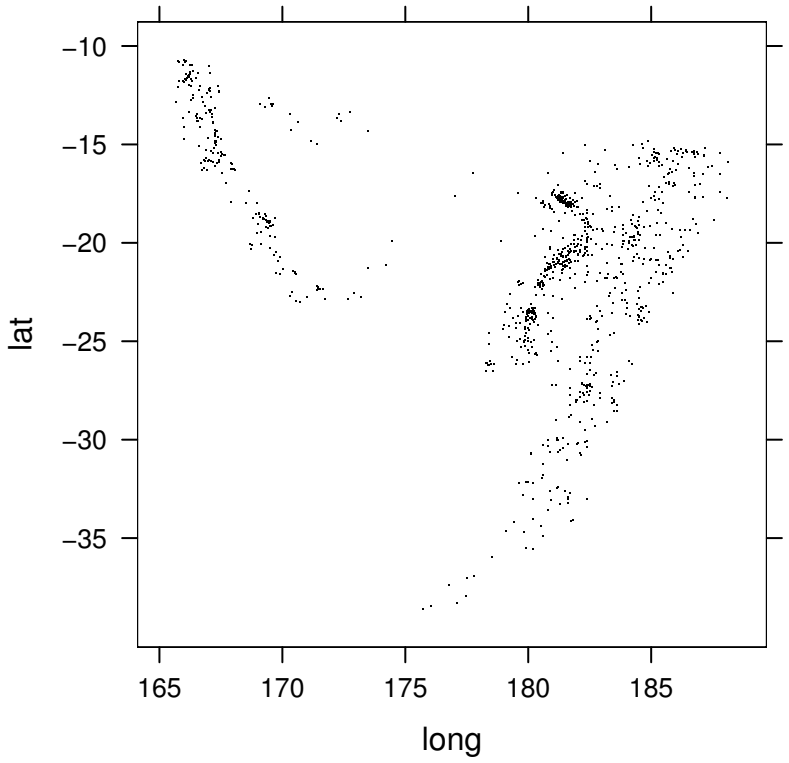

Figure 4.1
A scatterplot using lattice (showing the locations of earthquakes in the Pacific Ocean). A basic lattice plot has a very similar appearance to an analogous traditional plot.

This design makes it possible to work with the `trellis` object and modify it using the `update()` method for `trellis` objects, which is an alternative to modifying the original R expression used to create the `trellis` object. The following code demonstrates this idea by modifying the `trellis` object `tplot` to redefine the main title of the plot (it was empty). The resulting output is shown in Figure 4.2. A subtle change to look for is the fact that extra space has been introduced to allow room for adding the new main title text (the height of the plot region is slightly smaller compared to Figure 4.1).

```
> update(tplot,
         main="Earthquakes in the Pacific Ocean\n(since 1964)")
```

The side-effect of the code above is to produce new output that is a modification of the original plot, represented by `tplot`. However, it is important to remember that `tplot` has not been changed in any way (typing `tplot` again will produce output like Figure 4.1 again). In order to retain an R object representing the modified plot, the user must assign the value returned by the `update()` function, as in the following code.

```
> tplot2 <-
    update(tplot,
           main="Earthquakes in the Pacific Ocean (since 1964)")
```

4.1.1 Lattice devices

For each graphics device, lattice maintains its own set of graphical parameter settings that control the appearance of plots (colors of lines, fonts for text, and many more — see Section 4.3)*. The default settings depend on the type of device being opened (e.g., the settings are different for a PostScript device compared to a PDF device). In simple usage this causes no problems, because lattice automatically initializes these settings the first time that lattice output is produced on a device. If it is necessary to control the initial values for these settings the `trellis.device()` function can be used to explicitly open a device with specific lattice graphical parameter settings (or just to enforce specific lattice settings on an existing device). Section 4.3 describes more functions for manipulating the lattice graphical parameter settings.

*One of the features of Trellis Graphics is that carefully selected default settings are provided for colors, data symbols, and so on. These settings are selected to maximize the interpretability of plots and are based on principles of human perception[15].

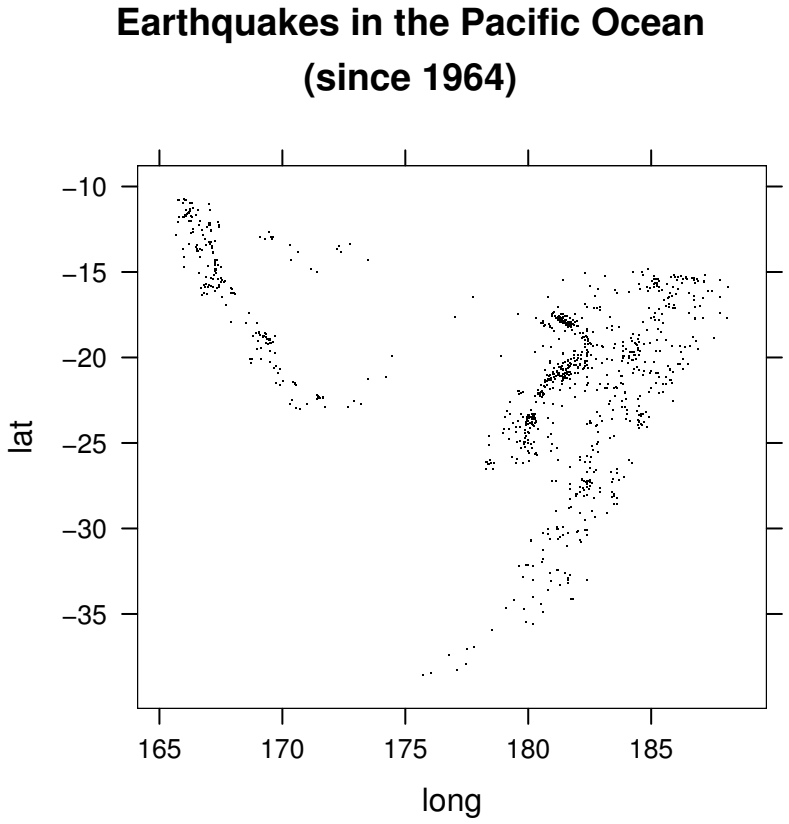

Figure 4.2
The result of modifying a lattice object. Lattice creates an object representing the plot. If this object is modified, the plot is redrawn. This figure shows the result of modifying the object representing the plot in Figure 4.1 to add a title to the plot.

4.2 Lattice plot types

Lattice provides functions to produce a number of standard plot types, plus some more modern and specialized plots. Table 4.1 describes the functions that are available and Figure 4.3 provides a basic idea of the sort of output that they produce.

There are a number of functions that produce output very similar to the output of functions in the traditional graphics system, but there are three possible reasons for using lattice functions instead of the traditional counterparts:

1. The default appearance of the lattice plots is superior in some areas. For example, the default colors and the default data symbols have been deliberately chosen to make it easy to distinguish between groups when more than one data series is plotted. There are also some subtle things such as the fact that tick labels on the y-axes are written horizontally by default, which makes them easier to read.

2. The lattice plot functions can be extended in several very powerful ways. For example, several data series can be plotted at once in a convenient manner and multiple panels of plots can be produced easily (see Section 4.2.1).

3. The output from lattice functions is grid output, so many powerful grid features are available for annotating, editing, and saving the graphics output. See Section 5.8 and Section 6.7 for examples of these features.

Most of the lattice plotting functions provide a very long list of arguments and produce a wide range of different types of output. Many of the arguments are shared by different functions and the on-line help for the `xyplot()` function provides an explanation of these standard arguments. The following sections address some of the important shared arguments, but for a full explanation of all arguments, the documentation for each specific function should be consulted. The next section discusses two important arguments, `formula` and `data`. The use of several other arguments is demonstrated in Section 4.2.2 in the context of a more complex example. Section 4.3 mentions the `par.settings` argument and Section 4.4 describes the `layout` argument. Section 4.5 describes the `panel` and `strip` arguments.

Table 4.1
The plotting functions available in lattice

Lattice Function	**Description**	**Traditional Analogue**
`barchart()`	Barcharts	`barplot()`
`bwplot()`	Boxplots Box-and-whisker plots	`boxplot()`
`densityplot()`	Conditional kernel density plots Smoothed density estimate	*none*
`dotplot()`	Dotplots Continuous versus categorical	`dotchart()`
`histogram()`	Histograms	`hist()`
`qqmath()`	Quantile–quantile plots Data set versus theoretical distribution	`qqnorm()`
`stripplot()`	Stripplots One-dimensional scatterplot	`stripchart()`
`qq()`	Quantile–quantile plots Data set versus data set	`qqplot()`
`xyplot()`	Scatterplots	`plot()`
`levelplot()`	Level plots	`image()`
`contourplot()`	Contour plots	`contour()`
`cloud()`	3-dimensional scatterplot	*none*
`wireframe()`	3-dimensional surfaces	`persp()`
`splom()`	Scatterplot matrices	`pairs()`
`parallel()`	Parallel coordinate plots	*none*

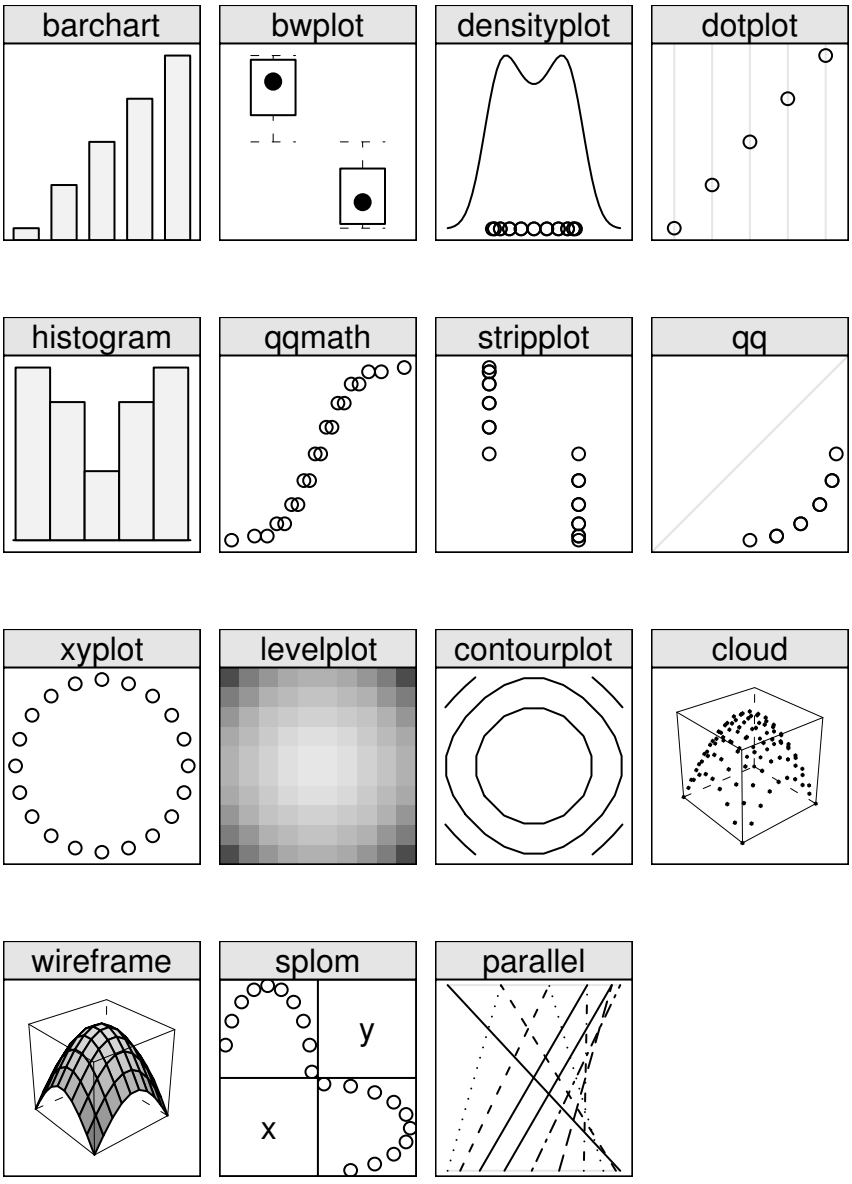

Figure 4.3
Plot types available in lattice. The name of the function used to produce the different plot types is shown in the strip above each plot.

4.2.1 The `formula` argument and multipanel conditioning

In most cases, the first argument to the lattice plotting functions is an R formula (see Section A.2.6) that describes which variables to plot. The simplest case has already been demonstrated. A formula of the form `y ~ x` plots variable `y` against variable `x`. There are some variations for plots of only one variable or plots of more than two variables. For example, for the `bwplot()` function, the formula can be of the form `~ x` and for the `cloud()` and `wireframe()` functions something of the form `z ~ x * y` is required to specify the three variables to plot. Another useful variation is the ability to specify multiple y-variables. Something of the form `y1 + y2 ~ x` produces a plot of both the `y1` variable and the `y2` variable against `x`. Multiple x-variables can be specified as well.

The second argument to a lattice plotting function is typically `data`, which allows the user to specify a data frame within which lattice can find the variables specified in the formula.

One of the very powerful features of Trellis Graphics is the ability to specify conditioning variables within the formula argument. Something of the form `y ~ x | g` indicates that several plots should be generated, showing the variable `y` against the variable `x` for each level of the variable `g`. In order to demonstrate this feature, the following code produces several scatterplots, with each scatterplot showing the locations of earthquakes that occurred within a particular depth range (see Figure 4.4). First of all, a new variable `depthgroup` is defined, which is a binning of the original `depth` variable in the `quakes` data set.

```
> depthgroup <- equal.count(quakes$depth, number=3, overlap=0)
```

Now this `depthgroup` variable can be used to produce a scatterplot for each depth range.

```
> xyplot(lat ~ long | depthgroup, data=quakes, pch=".")
```

In the Trellis terminology, the plot in Figure 4.4 consists of three *panels*. Each panel in this case contains a scatterplot and above each panel there is a *strip* that presents the level of the conditioning variable.

There can be more than one conditioning variable in the formula argument, in which case a panel is produced for each combination of the conditioning variables. An example of this is given in Section 4.2.2.

The most natural type of variable to use as a conditioning variable is a categorical variable (factor), but there is also support for using a continuous

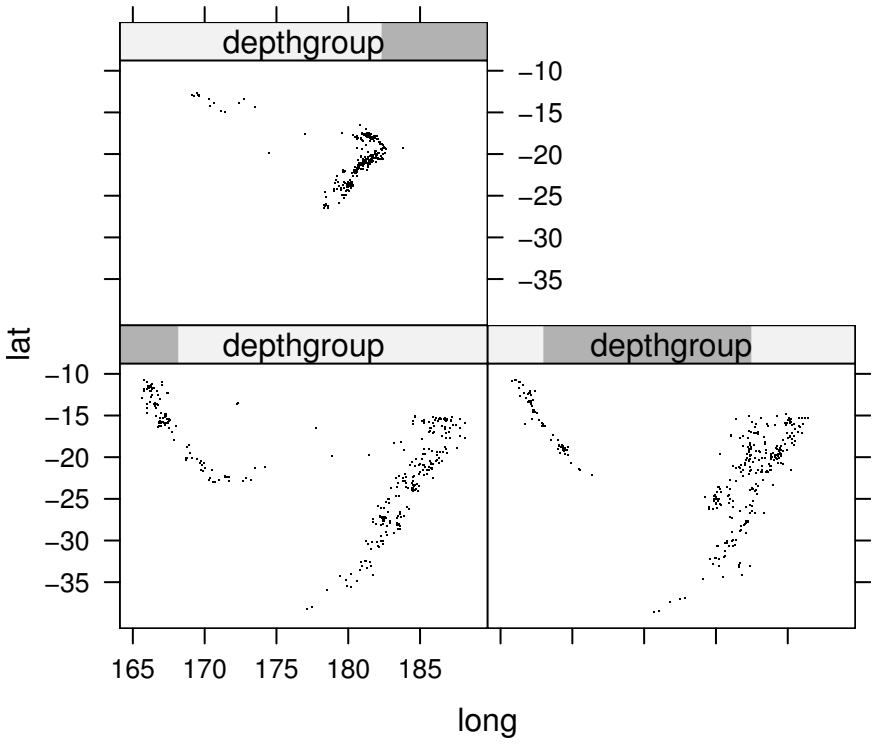

Figure 4.4
A lattice multipanel conditioning plot. A single function call produces several scatterplots of the locations of earthquakes for different earthquake depths.

(numeric) conditioning variable. For this purpose, Trellis Graphics introduces the concept of a *shingle*. This is a continuous variable with a number of ranges associated with it. The ranges are used to split the continuous values into (possibly overlapping) groups. The `shingle()` function can be used to explicitly control the ranges, or the `equal.count()` function can be used to generate ranges automatically given a number of groups (as was done to produce the `depthgroup` variable above).

4.2.2 A nontrivial example

This section describes an example that makes use of some of the common arguments to the lattice plotting functions to produce a more complex final result (see Figure 4.5). First of all, another grouping variable, `magnitude`, is defined, which is a shingle indicating whether an earthquake is big or small.

```
> magnitude <- equal.count(quakes$mag, number=2, overlap=0)
```

The plot is still produced from a single function call, but there are two conditioning variables, so there is a panel for each possible combination of depth and magnitude. A title and axis labels have been specified for the plot using the `main`, `xlab`, and `ylab` arguments. The `between` argument has been used to introduce a vertical gap between the top row of panels (big earthquakes) and the bottom row of panels (small earthquakes). The `par.strip.text` argument is used to control the size of text in the strips above each panel. The `scales` argument is used to control the drawing of axis labels; in this case the specification says that the x-axis labels should go at the bottom for both panels. This is to avoid the axis tick marks interfering with the main title. Finally, the `par.settings` argument is used to control the size of the tick labels on the axes.

```
> xyplot(lat ~ long | depthgroup * magnitude,
         data=quakes,
         main="Fiji Earthquakes",
         ylab="latitude", xlab="longitude",
         pch=".",
         scales=list(x=list(alternating=c(1, 1, 1))),
         between=list(y=1),
         par.strip.text=list(cex=0.7),
         par.settings=list(axis.text=list(cex=0.7)))
```

This example demonstrates that it is possible to have very fine control over many aspects of a lattice plot, given sufficient willingness to learn about all of the arguments that are available.

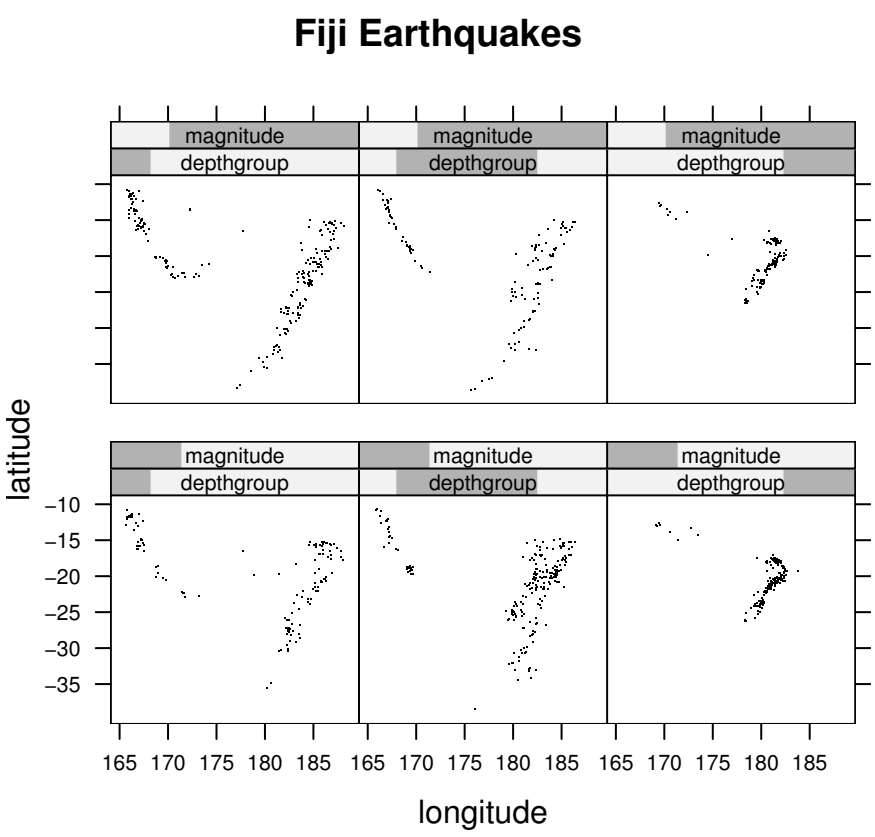

Figure 4.5
A complex lattice plot. There are a large number of arguments to lattice plotting functions to allow control over many details of a plot, such as the text to use for labels and titles, the size and placement of axis tick labels, and the size of the gaps between columns and rows of panels.

4.3 Controlling the appearance of lattice plots

An important feature of Trellis Graphics is the careful selection of default settings that are provided for many of the features of lattice plots. For example, the default data symbols and colors used to distinguish between different data series have been chosen so that it is easy to visually discriminate between them. Nevertheless, it is still sometimes desirable to be able to make alterations to the default settings for aspects like color and text size. It is also useful to be able to control the layout or arrangement of the components (panels and strips) of a lattice plot, but that is dealt with separately in Section 4.4. This section is only concerned with graphical parameters that control colors, line types, fonts and the like.

The lattice graphical parameter settings consist of a large list of parameter groups and each parameter group is a list of parameter settings. For example, there is a `plot.line` parameter group consisting of `col`, `lty`, and `lwd` settings to control the color, line type, and line width for lines drawn between data locations. There is a separate `plot.symbol` group consisting of `cex`, `col`, `font`, and `pch` settings to control the size, shape, and color of data symbols. The settings in each parameter group affect some aspect of a lattice plot: some have a "global" effect; for example, the `fontsize` settings affect all text in a plot; some are more specific; for example, the `strip.background` setting affects the background color of strips; and some only affect a certain aspect of a certain sort of plot; for example, the `box.dot` settings affect only the dot that is plotted at the median value in boxplots.

A separate list of graphical parameters is maintained for each graphics device. Changes to parameter settings (see below) only affect the current device.

The function `show.settings()` produces a picture representing some of the current graphical parameter settings. Figure 4.6 shows the settings for a black-and-white PostScript device.

The current value of graphical parameter settings can be obtained using the `trellis.par.get()` function. For a list of all current graphical parameter settings, type `trellis.par.get()`. If a name is specified as the argument to this function, then only the relevant settings are returned. The following code shows how to obtain only the `fontsize` group of settings (the output is on page 139).

```
> trellis.par.get("fontsize")
```

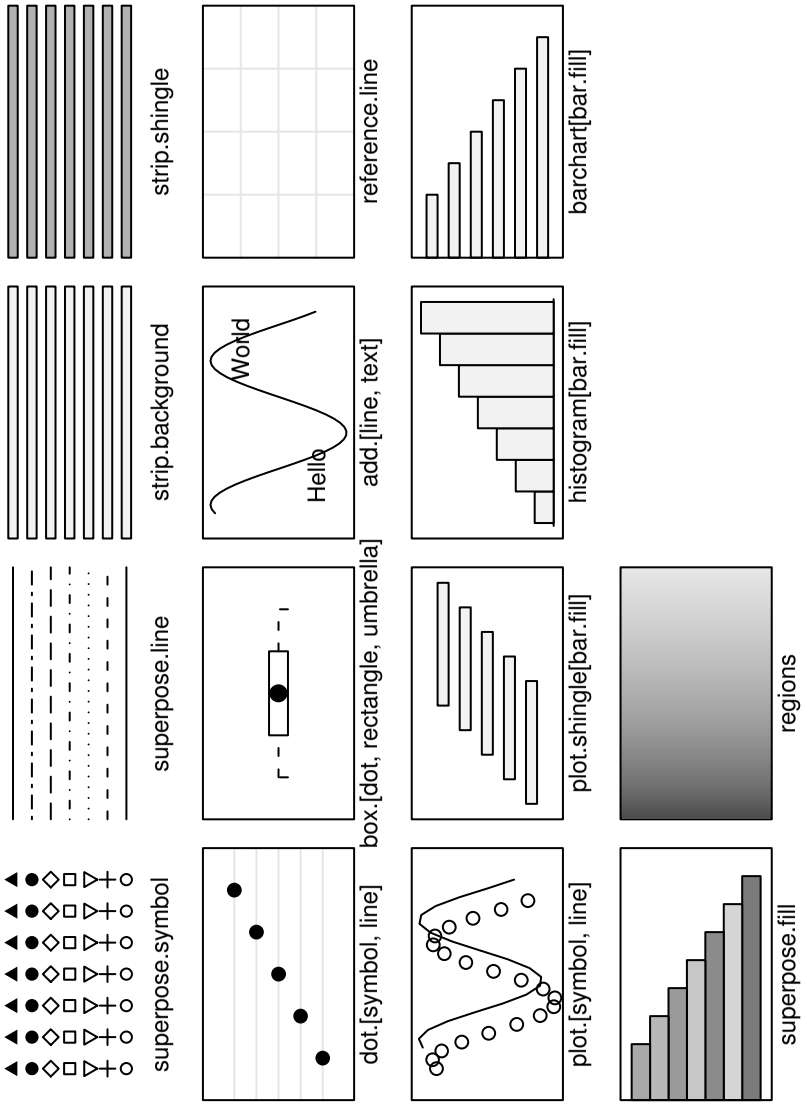

Figure 4.6
Some default lattice settings for a black-and-white PostScript device. This figure was produced by the lattice function `show.settings()`.

```
$text
[1] 9

$points
[1] 8
```

There are two ways to set new values for graphical parameters. The values can be set persistently (i.e., they will affect all subsequent plots until a new setting is specified) using the `trellis.par.set()` function, or they can be set temporarily for a single plot by specifying settings as an argument to a plotting function.

The `trellis.par.set()` function can be used in several ways. For back-compatibility with the original implementation of Trellis, it is possible to provide a name as the first argument and a list of settings as the second argument. This will modify the values for one parameter group.

A new approach is to provide a list of lists that can be used to modify multiple parameter groups at once. Lattice also introduces the concept of *themes*, which is a comprehensive and coherent set of graphical parameter values. It is possible to specify such a theme and enforce a new "look and feel" for a plot in one function call. Lattice currently provides one such theme via the `col.whitebg()` function. It is also possible to obtain the default theme for a particular device using the `canonical.theme()` function.

The following code demonstrates how to use `trellis.par.set()` in either the backwards-compatible, one-parameter-group-at-a-time way, or the new list-of-lists way, to specify `fontsize` settings.

```
> trellis.par.set("fontsize", list(text=14, points=10))
> trellis.par.set(list(fontsize=list(text=14, points=10)))
```

The theme approach is usually more convenient, especially when setting only one value within a parameter group. For example, the following code demonstrates the difference between the two approaches for modifying just the `text` setting within the `fontsize` parameter group (old way first, new way second).

```
> fontsize <- trellis.par.get("fontsize")
> fontsize$text <- 20
> trellis.par.set("fontsize", fontsize)

> trellis.par.set(list(fontsize=list(text=20)))
```

The concept of themes is an example of a lattice-specific extension to the original Trellis Graphics system.

The other way to modify lattice graphical parameter settings is on a per-plot basis, by specifying a `par.settings` argument in the call to a plotting function. The value for this argument should be a theme (a list of lists). Such a setting will only be enforced for the relevant plot and will not affect any subsequent plots. The following code demonstrates how to modify the `fontsize` settings just for a single plot.

```
> xyplot(lat ~ long, data=quakes,
         par.settings=list(fontsize=list(text=14, points=10)))
```

4.4 Arranging lattice plots

There are two types of arrangements to consider when dealing with lattice plots: the arrangement of panels and strips within a single lattice plot; and the arrangement of several complete lattice plots together on a single page.

In the first case (the arrangement of panels and strips within a single plot) there are two useful arguments that can be specified in a call to a lattice plotting function: the `layout` argument and the `aspect` argument.

The `layout` argument consists of up to three values. The first two indicate the number of columns and rows of panels on each page and the third value indicates the number of pages. It is not necessary to specify all three values, as lattice provides sensible default values for any unspecified values. The following code produces a variation on Figure 4.4 by explicitly specifying that there should be a single column of three panels via the `layout` argument, and that each panel must be "square" via the `aspect` argument. The `index.cond` argument has also been used to specify that the panels should be ordered from top to bottom (see Figure 4.7).

```
> xyplot(lat ~ long | depthgroup, data=quakes, pch=".",
         layout=c(1, 3), aspect=1, index.cond=list(3:1))
```

The `aspect` argument specifies the aspect ratio (height divided by width) for the panels. The default value is `"fill"`, which means that panels expand to occupy as much space as possible. In the example above, the panels were all forced to be square by specifying `aspect=1`. This argument will also accept the special value `"xy"`, which means that the aspect ratio is calculated to satisfy the "banking to 45 degrees" rule proposed by Bill Cleveland[13].

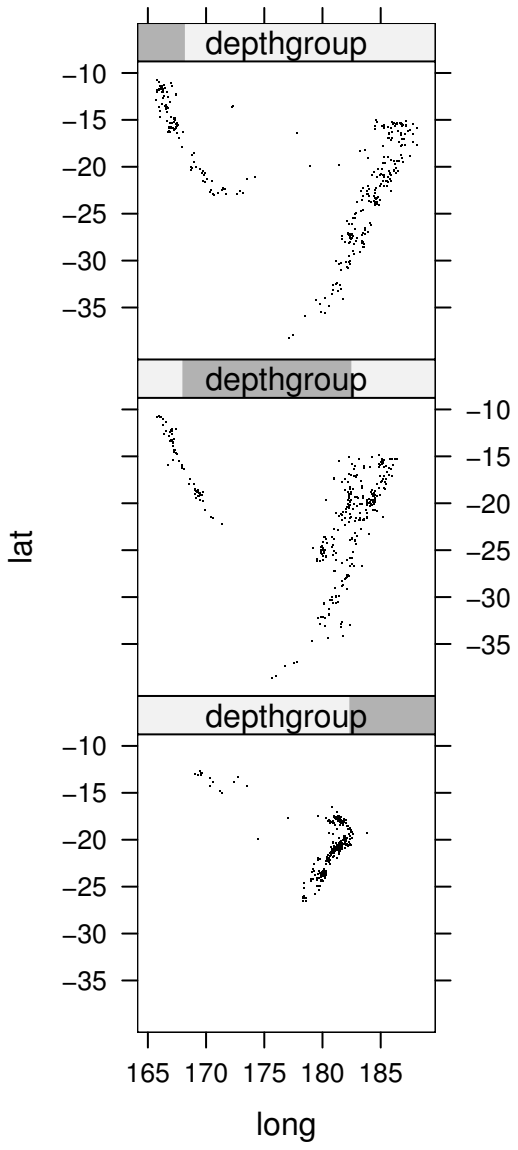

Figure 4.7
Controlling the layout of lattice panels. Lattice arranges panels in a sensible way by default, but there are several ways to force the panels to be arranged in a particular layout. This figure shows a custom arrangement of the panels in the plot from Figure 4.4.

As with the choice of colors and data symbols, a lot of work is done to select sensible default values for the arrangement of panels, so in many cases nothing special needs to be specified.

Another issue in the arrangement of a single lattice plot is the placement and structure of the key or legend. This can be controlled using the `auto.key` or `key` argument to plotting functions, which will accept complex specifications of the contents, layout, and positioning of the key.

The problem of arranging multiple lattice plots on a page requires a different approach. A `trellis` object must be created (but not plotted) for each lattice plot, then the `print()` function is called, supplying arguments to specify the position of each plot. The following code provides a simple demonstration using the average yearly number of sunspots from 1749 to 1983, available as the `sunspots` data set in the `datasets` package (see Figure 4.8). Two lattice plots are produced and then positioned one above the other on a page. The `position` argument is used to specify their location, `(left, bottom, right, top)`, as a proportion of the total page, and the `more` argument is used in the first `print()` call to ensure that the second `print()` call draws on the same page. The `scales` argument is also used to draw the x-axis at the top of the top plot.

```
> spots <- by(sunspots, gl(235, 12, lab=1749:1983), mean)
> plot1 <- xyplot(spots ~ 1749:1983, xlab="", type="l",
                  main="Average Yearly Sunspots",
                  scales=list(x=list(alternating=2)))
> plot2 <- xyplot(spots ~ 1749:1983, xlab="Year", type="l")
> print(plot1, position=c(0, 0.2, 1, 1), more=TRUE)
> print(plot2, position=c(0, 0, 1, 0.33))
```

Section 5.8 describes additional options for controlling the arrangements of panels within a lattice plot, and more flexible options for arranging multiple lattice plots, using the concepts and facilities of the grid system.

4.5 Annotating lattice plots

In the original Trellis Graphics system, plots are completely self-contained. There is no real concept of adding output to a plot once the plot has been drawn. This constraint has been lifted in lattice, though the traditional approach is still supported.

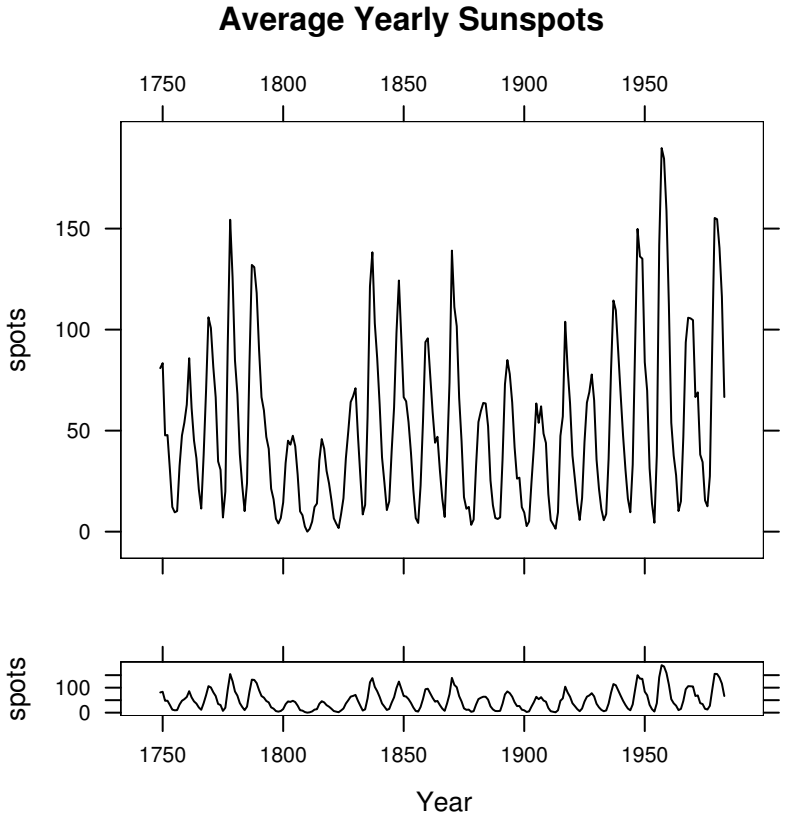

Figure 4.8
Arranging multiple lattice plots. This shows two separate lattice plots arranged together on a single page.

4.5.1 Panel functions and strip functions

The `trellis` object that is produced by a lattice plotting function is a complete description of a plot. The usual way to add extra output to a plot (e.g., add text labels to data symbols), is to add extra information to the `trellis` object. This is achieved by specifying a *panel function* via the `panel` argument of lattice plotting functions.

The panel function is called for each panel in a lattice plot. All lattice plotting functions have a default panel function, which is usually the name of the function with a "`panel.`" prefix. For example, the default panel function for the `xyplot()` function is `panel.xyplot()`. The default panel function draws the default contents for a panel so it is typical to call this default as part of a custom panel function.

The arguments available to the panel function differ depending on the plotting function. The documentation for individual panel functions should be consulted for full details, but some common arguments to expect are `x` and `y` (and possibly `z`), giving locations at which to plot data symbols, and `subscripts`, which provides the indices used to obtain the subset of the data for each panel.

In addition to the panel function, it is possible to specify a *prepanel function* for controlling the scaling and size of panels and a *strip function* for controlling what gets drawn in the strips of a lattice plot.

The following code provides a simple demonstration of the use of panel, prepanel and strip functions. The plot is a lattice multi-panel scatterplot with text labels added to the data points and a custom strip showing both levels of the conditioning variable with the relevant level bold and the other level grey (see Figure 4.9).

The panel function calls the default `panel.xyplot()` to draw data symbols, then calls `ltext()` to draw the labels. Because lattice is based on grid, traditional graphics functions will not work in a panel function (though see Appendix B for a way around this constraint). However, there are several lattice functions that correspond to traditional functions and can be used in much the same way as the corresponding traditional functions. The names of the lattice analogues are the traditional function names with an "`l`" prefix added. In this case, the code draws letters as the labels, using the `subscripts` argument to select an appropriate subset. The labels are drawn slightly to the left of and above the data symbols by subtracting 1 from the `x` values and adding 1 to the `y` values.

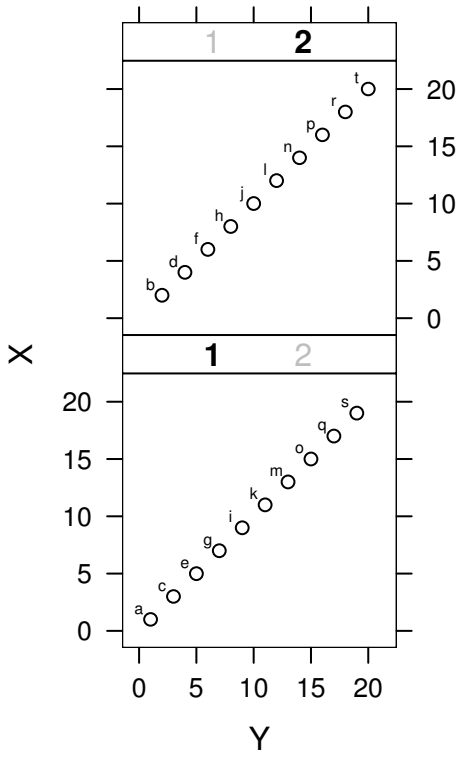

Figure 4.9
Annotating a lattice plot using panel and strip functions. The text labels have been added beside the data symbols using a custom panel function and the bold and grey numerals in the strips have been produced using a custom strip function.

```
> myPanel <- function(x, y, subscripts, ...) {
    panel.xyplot(x, y, ...)
    ltext(x - 1, y + 1, letters[subscripts], cex=0.5)
  }
```

The strip function also uses `ltext()`. Locations within the strip are based on a "normalized" coordinate system with the location `(0, 0)` at the bottom-left corner and `(1, 1)` at the top-right corner. The font face and color for the text is calculated using the `which.panel` argument. This supplies the current level for each conditioning variable in the panel.

```
> myStrip <- function(which.panel, ...) {
    font <- rep(1, 2)
    font[which.panel] <- 2
    col=rep("grey", 2)
    col[which.panel] <- "black"
    llines(c(0, 1, 1, 0, 0), c(0, 0, 1, 1, 0))
    ltext(c(0.33, 0.66), rep(0.5, 2), 1:2,
          font=font, col=col)
  }
```

The prepanel function calculates the limits of the scales for each panel by extending the range of data by 1 unit (this allows room for the text labels that are added in the panel function).

```
> myPrePanel <- function(x, y, ...) {
    list(xlim=c(min(x) - 1, max(x) + 1),
         ylim=c(min(y) - 1, max(y) + 1))
  }
```

We now generate some data to plot and create the plot using `xyplot()`, with the special panel functions provided as arguments. The final result is shown in Figure 4.9.

```
> X <- 1:20
> Y <- 1:20
> G <- factor(rep(1:2, 10))
```

```
> xyplot(X ~ Y | G, aspect=1, layout=c(1, 2),
         panel=myPanel, strip=myStrip,
         prepanel=myPrePanel)
```

A great deal more can be done with panel functions using grid concepts and functions. See Sections 5.8 and 6.7 for some examples.

4.5.2 Adding output to a lattice plot

Unlike in the original Trellis implementation, it is also possible to add output to a complete lattice plot (i.e., without using a panel function). The function `trellis.focus()` can be used to return to a particular panel or strip of the current lattice plot in order to add further output using, for example, `llines()` or `lpoints()`. The function `trellis.panelArgs()` may be useful for retrieving the arguments (including the data) used to originally draw the panel. Also, the `trellis.identify()` function provides basic mouse interaction for labelling data points within a panel. Again, Sections 5.8 and 6.7 show how grid provides more flexibility for navigating to different parts of a lattice plot and for adding further output.

4.6 Creating new lattice plots

The lattice plotting functions have many arguments and are very flexible in the variety of output that they can produce. However, lattice is not designed to be the best environment for developing new types of graphical display. For example, there is no mechanism for adding new graphical parameters to the list of values that control the appearance of plots (see Section 4.3).

Nevertheless, a lot can be done by defining a panel function that does not just add extra output to the default output, but replaces the default output with some sort of completely different display. For example, the lattice `dotplot()` function is really only a call to the `bwplot()` function with a different panel function supplied.

Users wanting to develop a new lattice plotting function along these lines are advised to read Chapter 5 to gain an understanding of the grid system that is used in the production of lattice output.

Chapter summary

The lattice package implements and extends the Trellis graphics system for producing complete statistical plots. This system provides most standard plot types and a number of modern plot types with several important extensions. For a start, the layout and appearance of the plots is designed to maximize readability and comprehension of the information represented in the plot. Also, the system provides a feature called multipanel conditioning, which produces multiple panels of plots from a single data set, where each panel contains a different subset of the data. The lattice functions provide an extensive set of arguments for customizing the detailed appearance of a plot and there are functions that allow the user to add further output to a plot.