Chapter No. 2
"Beyond the Basics: Adjusting Key Parameters"
In this package, you will find:
A Biography of the author of the book
A preview chapter from the book, Chapter NO.2 "Beyond the Basics: Adjusting Key Parameters"
A synopsis of the book’s content
Information on where to buy this book

About the Author

Hrishi V. Mittal has been working with R for a few years in different capacities. He was introduced to the exciting world of data analysis with R when he was working as a Senior Air Quality Scientist at King's College London, where he used R extensively to analyze large amounts of air pollution and traffic data to inform the Mayor of London's Air Quality Strategy. He has experience in various other programming languages, but prefers R for data analysis and visualization. He is actively involved in various R mailing lists, forums and the development of some R packages.

In early 2010, Hrishi started Pretty Graph Limited (www.prettygraph.com), a software company specializing in web-based data visualization products. The company's flagship product, Pretty Graph, uses R as the backend engine for helping researchers and businesses visualize and analyze data. The goal is to bring the power of R to a wider audience by providing a modern graphical user interface which can be accessed by anyone and from anywhere simply by using a web browser.

First and foremost, I am grateful to the creators of R, Ross Ihaka and Robert Gentleman, and the countless other contributors who have made one of the greatest open source software of all time.

I would like to thank my wife Louise for her patience and support throughout the writing of the book. Her feedback on the writing itself has also been very useful. Special thanks are also due to Clive and Jimmy, who have consistently provided silent, warm and furry stress relief.

I'm grateful to my parents and sister for their love and the pride they always take in my work, even when they are not quite sure what I'm doing.

It's been nice to have support from my friends Madhavi Bhargava, Rohit Menon and Aniruddha Kembavi, who have been very encouraging and at times more excited than me about the book.

I'd also like to thank the reviewers for pointing out some errors and suggesting valuable improvements. Last, but not least, I'd like to thank my editor Eleanor Duffy and the rest of the team at Packt, who have been very professional, understanding and helpful throughout the project.
R Graphs Cookbook

With more than two million users worldwide, R is one of the most popular open source projects. It is a free and robust statistical programming environment with very powerful graphical capabilities. Analyzing and visualizing data with R is a necessary skill for anyone doing any kind of statistical analysis, and this book will help you do just that in the easiest and most efficient way possible.

Unlike other books on R, this book takes a practical hands-on approach and will dive straight into creating graphs in R right from the very first page. If you wish to harness the power of this mighty open source programming language to visually present and analyze your data in the best way possible—this book is going to show you how.

The R Graphs Cookbook takes a practical approach to teaching how to create effective and useful graphs using R. It will demystify a lot of difficult and confusing R functions and parameters. It will enable you to construct and modify data graphics to suit your analysis, presentation, and publication needs.

This practical guide begins by teaching you how to make basic graphs in R and progresses through subsequent dedicated chapters about each graph type in depth. You will learn all about making graphics such as scatter plots, line graphs, bar charts, pie charts, dot plots, heat maps, histograms, and box plots. In addition, there are detailed recipes on making various combinations and advanced versions of these graphs. Dedicated chapters on polishing and finalizing graphs will enable you to produce professional quality graphs for presentation and publication. With the R Graphs Cookbook in hand, making graphs in R has never been easier.

What This Book Covers

Chapter 1, Basic Graph Functions introduces recipes for some basic types of graphs, useful in almost any kind of data analysis. We will go through all the steps to get you going from reading your data into R, making a first graph, tweaking it to suit your needs, and then saving and exporting it for use in presentations and publications.

Chapter 2, Beyond the Basics: Adjusting Key Parameters looks more closely at various arguments to graph functions and their values, highlighting common pitfalls and workarounds. The par() function is explained with some useful examples showing how to adjust colors, sizes, margins, and styles of various graph elements such as points, lines, bars, axes, and titles.

The subsequent chapters 3 to 9 cover the graph types introduced in the first two chapters in more detail.

Chapter 3, Creating Scatter Plots has over a dozen recipes covering scatter plots, which are some of the simplest and most commonly used type of graphs in data analysis. We will see how we can make more enhanced plots by adjusting various arguments and using some new functions.

Chapter 4, Creating Line Graphs and Time Series Charts discusses some more intermediate to advanced recipes for customizing line graphs, improving and speeding up line graphs with multiple lines, processing dates to make time series charts, sparklines and stock charts.

Chapter 5, Creating Bar, Dot, and Pie Charts will show you how you can create many useful variations of bar graphs and dot plots by using only the base library functions. We will also look at a few recipes addressing common criticisms of pie charts with some ways to make them more readable.

Chapter 6, Creating Histograms enhances the basic histogram in R by changing the plotting mode and bins, in addition to style adjustments. We will also look at some advanced recipes combining histograms with other types of graphs.

Chapter 7, Creating Box and Whisker Plots looks into various stylistic and structural adjustments to box plots. We will start by looking at some basic arguments to change individual aspects of a box plot and slowly move to more advanced recipes involving the use of multiple function calls.

Chapter 8, Creating Heat Maps and Contour Plots discusses various types of heat maps for visualizing correlations, trends and multivariate data, and contour plots for showing topographical information in various two-dimensional and three-dimensional ways.

Chapter 9, Creating Maps builds on top of the introduction to visualizing data on geographical maps in the first chapter, covering recipes for plotting data from the World Bank, World Health Organization (WHO), Google Maps API, and some Geographical Information Systems (GIS).

Chapter 10, Finalizing Graphs for Publications and Presentations discusses some tricks and tips to add some polish to our graphs so that they can be used for publication and presentation. We will cover many important practical topics such as exported graph file formats, high resolution formats, vector formats such as PDF, SVG, and PS, mathematical and scientific notations, text descriptions, fonts, graph templates, and themes.

Beyond the Basics: Adjusting Key Parameters

In this chapter, we will cover:

- Setting colors of points, lines, and bars
- Setting plot background colors
- Setting colors for text elements: axis annotations, labels, plot titles, and legends
- Choosing color combinations and palettes
- Setting fonts for annotations and titles
- Choosing plotting point symbol styles and sizes
- Choosing line styles and width
- Choosing box styles
- Adjusting axis annotations and tick marks
- Formatting log axes
- Setting graph margins and dimensions

Introduction

In this chapter, we will learn about some of the simplest yet most important settings and parameters of graphs in R base graphics. Learning how to adjust colors, sizes, margins, and styles of various graph elements such as points, lines, bars, axes, and titles will give us the ability to improve upon the basic graph commands we learnt in Chapter 1.

Beyond the Basics: Adjusting Key Parameters

In the previous chapter, we got a glimpse of the different types of graphs that can be made in R using small snippets of code. Now, we will learn how to modify the fundamental building blocks of those graphs to better suit our needs.

The R base library has very powerful graphical capabilities. While you can produce pretty much any type of graph with a couple of lines of code, the default layout and look of the graph is often very basic. Sometimes, you may run into problems such as axis labels and titles getting chopped off at the edges or the legend size or position may mask part of your graph. Sometimes, the default color combinations may not be suitable for presentation or publication.

In this chapter we will go through the relevant names and accepted values of different arguments and arguments to graph functions. We will take a closer look at the `par()` function, which we briefly introduced in the previous chapter.

Reading and trying out all the recipes in this chapter is highly recommended as it will give you a very good hands-on grasp of certain aspects of graph manipulation, which you are likely to use a lot in any visual analysis in R.

Let's get started!

Setting colors of points, lines, and bars

In this recipe we will learn the simplest way to change the colors of points, lines, and bars in scatter plots, line plots, histograms, and bar plots.

Getting ready

All you need to try out this recipe is to run R and type the recipe at the command prompt. You can also choose to save the recipe as a script so that you can use it again later on.

How to do it...

The simplest way to change the color of any graph element is by using the `col` argument. For example, the `plot()` function takes the `col` argument:

```r
plot(rnorm(1000),
    col="red")
```

If we choose plot type as line, then the color is applied to the plotted line. Let’s use the dailysales.csv example dataset we used in Chapter 1. First, we need to load it:

```r
Sales <- read.csv("dailysales.csv", header=TRUE)

plot(sales$units~as.Date(sales$date, "%d/%m/%y"),
     type="l", #Specify type of plot as l for line
     col="blue")
```

Similarly, the `points()` and `lines()` functions apply the `col` argument’s value to the plotted points and lines respectively.

`barplot()` and `hist()` also take the `col` argument and apply them to the bars. So the following code would produce a bar plot with blue bars:

```r
barplot(sales$ProductA~sales$City,
        col="blue")
```

The `col` argument for `boxplot()` is applied to the color of the boxes plotted.

**How it works...**

The `col` argument automatically applies the specified color to the elements being plotted, based on the plot type. So, if we do not specify a plot type or choose points, then the color is applied to points. Similarly, if we choose plot type as line then the color is applied to the plotted line and if we use the `col` argument in the `barplot()` or `histogram()` commands, then the color is applied to the bars.

`col` accepts names of colors such as `red`, `blue`, and `black`. The `colors()` function lists all the built-in colors (more than 650) available in R. We can also specify colors as hexadecimal codes such as `#FF0000` (for red), `#0000FF` (for blue), and `#000000` (for black). If you have ever made any web pages, you would know that these hex codes are used in HTML to represent colors.

`col` can also take numeric values. When it is set to a numeric value, the color corresponding to that index in the current color palette is used. For example, in the default color palette the first color is black and the second color is red. So `col=1` and `col=2` refers to black and red respectively. Index 0 corresponds to the background color.

**There’s more...**

In many settings, `col` can also take a vector of multiple colors, instead of a single color. This is useful if you wish to use more than one color in a graph. For example, in Chapter 1 we made a bar plot of sales data for three products across five cities. In that example, we did use a vector of five colors to represent each of the five cities with the help of the `heat.colors()` function. The `heat.colors()` function takes a number as an argument and returns a vector of those many colors. So `heat.colors(5)` produces a vector of five colors.

Type the following at the R prompt:

```r
heat.colors(5)
```

You should get the following output:

```
[1] "#FF0000FF" "#FF5500FF" "#FFAA00FF" "#FFFF00FF" "#FFFF80FF"
```

Those are five colors in the hexadecimal format.

Another way of specifying a vector of colors is to construct one:

```r
barplot(as.matrix(sales[,2:4]), beside=T,
legend=sales$City,
col=c("red","blue","green","orange","pink"),
border="white")
```

In the example, we set the value of `col` to `c("red","blue","green","orange","pink")`, which is a vector of five colors.

We have to take care to make a vector matching the length of the number of elements, in this case bars we are plotting. If the two numbers don't match, R will 'recycle' values by repeating colors from the beginning of the vector. For example, if we had fewer colors in the vector than the number of elements, say if we had four colors in the previous plot, then R would apply the four colors to the first four bars and then apply the first color to the fifth bar. This is called recycling in R:

```r
barplot(as.matrix(sales[,2:4]), beside=T,
legend=sales$City,
col=c("red","blue","green","orange"),
border="white")
```

In the example, both the bars for the first and last data rows (Seattle and Mumbai) would be of the same color (red), making it difficult to distinguish one from the other.

One good way to ensure that you always have the correct number of colors is to find out the length of the number of elements first and pass that as an argument to one of the color palette functions. For example, if we did not know the number of cities in the example we have just seen; we could do the following to make sure the number of colors matches the number of bars plotted:

```r
barplot(as.matrix(sales[,2:4]), beside=T,
legend=sales$City,
col=heat.colors(length(sales$City)),
border="white")
```

We used the `length()` function to find out the length or the number of elements in the vector `sales$City` and passed that as the argument to `heat.colors()`. So, regardless of the number of cities we will always have the right number of colors.

**See also**

In the next four recipes, we will see how to change the colors of other elements. The fourth recipe is especially useful where we look at color combinations and palettes.

Setting plot background colors

The default background color of all plots in R is white, which is usually the best choice as it is least distracting for data analysis. However, sometimes we may wish to use another color. We will see how to set background colors in this recipe.

Getting ready

All you need to try out this recipe is to run R and type the recipe at the command prompt. You can also choose to save the recipe as a script so that you can use it again later on.

How to do it...

To set the plot background color to gray we use the `bg` argument in the `par()` command:

```r
par(bg="gray")
plot(rnorm(100))
```

How it works...

The `par()` command's `bg` argument sets the background color for the entire plotting area including the margins for any subsequent plots on the same device. Until the plotting device is closed or a new device is initiated, the background color stays the same.

It is more likely that we want to set the background color only for the plot region (within the axes) but there is no straightforward way to do this in R. We must draw a rectangle of the desired color in the background and then make our graph on top of it:

```r
plot(rnorm(1000), type="n")
x<-par("usr")
rect(x[1], x[3], x[2], x[4], col="lightgray ")
points(rnorm(1000))
```

First we draw the plot with `type` set to "n" so that the plotted elements are invisible. This does not show the graph points or lines but sets the axes up, which we need for the next step.

`par("usr")` gets us the co-ordinates of the plot region in a vector of form `c(xleft, xright, ybottom, ytop)`. We then use the `rect()` function to draw a rectangle with a fill color that we wish to use for the plot background. Note that `rect()` takes a set of arguments representing the `xleft, ybottom, xright, ytop` co-ordinates. So we must pass the values we obtained from `par("usr")` in the correct order. Then, finally we redraw the graph with the correct type (points or lines).
Setting colors for text elements: axis annotations, labels, plot titles, and legends

Axis annotations are the numerical or text values placed beside tick marks on an axis. Axis labels are the names or titles of axes, which tell the reader what the values on a particular axis represent. In this recipe, we will learn how to set the colors for these elements and legends.

Getting ready

All you need to try out this recipe is to run R and type the recipe at the command prompt. You can also choose to save the recipe as a script so that you can use it again later on.

How to do it...

Let's say we want to make the axis value annotations black, the labels of the axes gray, and the plot title dark blue, you should do the following:

```r
plot(rnorm(100),
     main="Plot Title",
     col.axis="blue",
     col.lab="red",
     col.main="darkblue")
```

How it works...

Colors for axis annotations, labels, and plot titles can be set either using the `par()` command before making the graph or in the graph command such as `plot()` itself. The arguments for setting the colors for axis annotations, labels, and plot titles are `col.axis`, `col.lab`, and `col.main` respectively.

They are similar to the `col` argument and take names of colors or hex codes as values, but do not take a vector of more than one color.

There's more...

If we use the `par()` command, the difference is that `par()` will apply these settings to every subsequent graph, until it is reset either by specifying the settings again or starting a new graphics device:

```r
par(col.axis="black",
    col.lab="#444444",
    col.main="darkblue")

plot(rnorm(100),main="plot")
```

The `col.axis` argument can also be passed to the `axis()` function, which is useful for making a custom axis if you do not want to use the default axis. The `col.lab` argument does not work with `axis()` and must be specified in `par()` or the main graph function such as `plot()` or `barplot()`.

The `col.main` argument can also be passed to the `title()` function, which is useful for adding a custom plot title if you do not want to use the default title:

```r
title("Sales Figures for 2010", col.main="blue")
```

Axis labels can also be specified with `title()`:

```r
title(xlab="Month",ylab="Sales",col.lab="red")
```

This is handy because you can specify two different colors for the X and Y axes:

```r
title(xlab="X axis",col.lab="red")
title(ylab="Y axis",col.lab="blue")
```

When setting the axis titles with the `title()` command, we must set `xlab` and `ylab` to empty strings `""` in the original plot command to avoid overlapping titles.
Choosing color combinations and palettes

We often need more than one color to represent various elements in graphs. Palettes are combinations of colors which are a convenient way to use multiple colors without choosing individual colors separately. R provides inbuilt color palettes as well as the ability to make our own custom palettes. Using palettes is a good way to avoid repeatedly choosing or setting colors in multiple locations, which can be a source of error and confusion. It helps in separating the presentation settings of a graph from the construction.

Getting ready

All you need to try out this recipe is to run R and type the recipe at the command prompt. You can also choose to save the recipe as a script so that you can use it again later on. One new library needs to be installed, which is also explained.

How to do it...

We can change the current palette by passing a character vector of colors to the `palette()` function. For example:

```r
palette(c("red","blue","green","orange"))
```

To use the colors in the current palette, we can refer to them by the index number. For example, `palette()[1]` would be red.

How it works...

R has a default palette of colors which can be accessed by calling the `palette()` function. If we run the `palette()` command just after starting R, we get the default palette:

```r
palette()
[1] "black"  "red"  "green3"  "blue"  "cyan"  "magenta"
   "yellow"  "gray"
```

To revert back to the default palette type:

```r
palette("default")
```

When a vector of color names is passed to the `palette()` function, it sets the current palette to those colors. We must enter valid color names otherwise we will get an invalid color name error.
There's more...

Besides the default palette provided by the `palette()` function, R has many more built-in palettes and additional palette libraries. One of the most commonly used palettes is the `heat.colors()` palette, which provides a range of colors from red through yellow to white, based on the number of colors specified by the argument `n`. For example, `heat.colors(10)` produces a palette of 10 warm colors from red to white.

Other palettes are `rainbow()`, `terrain.colors()`, `cm.colors()`, and `topo.colors` which take the number of colors as an argument.

`RColorBrewer` is a very good color palette package that creates nice looking color palettes especially for thematic maps. It is an R implementation of the RColorBrewer palettes, which provides three types of palettes: sequential, diverging, and qualitative. More information is available at [http://www.colorbrewer.org](http://www.colorbrewer.org).

To use `RColorBrewer`, we need to install and load it:

```r
install.packages("RColorBrewer")
library(RColorBrewer)
```

To see all the `RColorBrewer` palettes run the following command at the R prompt:

```r
display.brewer.all()
```
Beyond the Basics: Adjusting Key Parameters

The names of the palettes are displayed in the left-hand margin and the colors in each palette are displayed in each row running to the right.

To use one of the palettes, let's say YlOrRd (which as the names suggest is a combination of yellows and reds), we can use the brewer.pal() function:

```r
brewer.pal(7,"YlOrRd")
[1] "#FFFFB2" "#FED976" "#FEB24C" "#FD8D3C" "#FC4E2A" "#E31A1C" "#B10026"
```

The `brewer.pal` function takes two arguments: the number of colors we wish to choose and the name of the palette. The minimum number of colors is three but the maximum varies from palette to palette.

We can view the colors of an individual palette by using the `display.brewer.pal()` command:

```r
display.brewer.pal(7,"YlOrRd")
```

To use a specific color of the palette we can refer to it by its index number. So the first color in the palette is `brewer.pal(7,"YlOrRd")[1]`, the second is `brewer.pal(7,"YlOrRd")[2]`, and so on.

We can set the current palette to the previous one by using the `palette()` function:

```r
palette(brewer.pal(7,"YlOrRd"))
```

Now we can refer to the individual colors as `palette()[1]`, `palette()[2]`, and so on. We can also store the palette as a vector:

```r
pal1<- brewer.pal(7,"YlOrRd")
```

See also

We will see the use of a lot of color palettes throughout the recipes in this book starting from Chapter 3, Creating Scatter Plots.

Setting fonts for annotations and titles

For most data analysis we can just use the default fonts for titles. However, sometimes we may want to choose different fonts for presentation and publication purposes. Selecting fonts can be tricky as it depends on the operating system and the graphics device. We will see some simple ways to choose fonts in this recipe.

Getting ready

All you need to try out this recipe is to run R and type the recipe at the command prompt. You can also choose to save the recipe as a script so that you can use it again later on.

How to do it...

The font family and face can be set with the `par()` command:

```
par(family="serif",font=2)
```

How it works...

A font is specified in two parts: a font family (such as Helvetica or Arial) and a font face within that family (such as bold or italic).

The available font families vary by operating system and graphics devices. So R provides some proxy values which are mapped on to the relevant available fonts irrespective of the system. Standard values for family are "serif", "sans", and "mono".

The font argument takes numerical values: 1 corresponds to plain text (the default), 2 to bold face, 3 to italic, and 4 to bold italic.

For example, `par(family="serif",font=2)` sets the font to a bold Times New Roman on Windows. You can check the other font mappings by running the `windowsFonts()` command at the R prompt.

The fonts for axis annotations, labels, and plot main title can be set separately using the `font.axis`, `font.lab`, and `font.main` arguments respectively.

There's more...

The choice of fonts is very limited if we just use the proxy family names. However, we can use a wide range of fonts if we are exporting our graphs in the PostScript or PDF formats. The `postscriptFonts()` and `pdfFonts()` functions show all the available fonts for those devices. To see the PDF fonts, run the following command:

```
names(pdfFonts())
```

```
[1]  "serif"      "sans"      "mono"
[7] "Helvetica"   "Helvetica-Narrow" "NewCenturySchoolbook"
[10]"Palatino"    "Times"      "URWGothic"
[13]"URWBookman" "NimbusMon" "NimbusSan"
```

Beyond the Basics: Adjusting Key Parameters

To use one of these font families in a PDF, we can pass the `family` argument to the `pdf()` function:

```r
df(family="AvantGarde")  
df(paste(family="AvantGarde"))
```

See also

In Chapter 10, Finalizing Graphs, we will see some more practical recipes on setting fonts for publications and presentations.

Choosing plotting point symbol styles and sizes

In this recipe, we will see how we can adjust the styling of plotting symbols, which is useful and necessary when we plot more than one set of points representing different groups of data on the same graph.

Getting ready

All you need to try out this recipe is to run R and type the recipe at the command prompt. You can also choose to save the recipe as a script so that you can use it again later on. We will also use the `cityrain.csv` example data file that we used in the first chapter. Please read the file into R as follows:

```r
rain<-read.csv("cityrain.csv")
```

How to do it...

The plotting symbol and size can be set using the `pch` and `cex` arguments:

```r
plot(rnorm(100),pch=19,cex=2)
```
How it works...

The pch argument stands for plotting character (symbol). It can take numerical values (usually between 0 and 25) as well as single character values. Each numerical value represents a different symbol. For example, 1 represents circles, 2 represents triangles, 3 represents plus signs, and so on. If we set the value of pch to a character such as "*" or "£" in inverted commas, then the data points are drawn as that character instead of the default circles.

The size of the plotting symbol is controlled by the cex argument, which takes numerical values starting at 0 giving the amount by which plotting symbols should be magnified relative to the default. Note that cex takes relative values (the default is 1). So, the absolute size may vary depending on the defaults of the graphic device in use. For example, the size of plotting symbols with the same cex value may be different for a graph saved as a PNG file versus a graph saved as a PDF.

There's more...

The most common use of pch and cex is when we don't want to use color to distinguish between different groups of data points. This is often the case in scientific journals which do not accept color images. For example, let's plot the city rainfall data we looked at in Chapter 1 as a set of points instead of lines:

```r
plot(rain$Tokyo,
     ylim=c(0,250),
     main="Monthly Rainfall in major cities",
     xlab="Month of Year",
```

Beyond the Basics: Adjusting Key Parameters

```r
ylab="Rainfall (mm)",
pch=1

points(rain$NewYork,pch=2)
points(rain$London,pch=3)
points(rain$Berlin,pch=4)

legend("top",
legend=c("Tokyo","New York","London","Berlin"),
ncol=4,
cex=0.8,
bty="n",
pch=1:4)
```

**See also**

We will see more examples of symbol settings later in the book, especially in the next chapter on scatter plots.

**Choosing line styles and width**

Similar to plotting point symbols, R provides simple ways to adjust the style of lines in graphs.

Getting ready

All you need to try out this recipe is to run R and type the recipe at the command prompt. You can also choose to save the recipe as a script so that you can use it again later on. We will again use the cityrain.csv data file that we read in the last recipe.

How to do it...

Line styles can be set by using the lty and lwd arguments (for line type and width respectively) in the plot(), lines(), and par() commands. Let’s take our rainfall example and apply different line styles keeping the color the same:

```r
plot(rain$Tokyo,
     ylim=c(0,250),
     main="Monthly Rainfall in major cities",
     xlab="Month of Year",
     ylab="Rainfall (mm)",
     type="l",
     lty=1,
     lwd=2)
lines(rain$NewYork,lty=2,lwd=2)
lines(rain$London,lty=3,lwd=2)
lines(rain$Berlin,lty=4,lwd=2)
legend("top",
       legend=c("Tokyo","New York","London","Berlin"),
       ncol=4,
       cex=0.8,
       bty="n",
       lty=1:4,
       lwd=2)
```

Beyond the Basics: Adjusting Key Parameters

How it works...

Both line type and width can be set with numerical values as shown in the previous example. Line type number values correspond to types of lines:

- 0: blank
- 1: solid (default)
- 2: dashed
- 3: dotted
- 4: dotdash
- 5: longdash
- 6: twodash

We can also use the character strings instead of numbers, for example, lty="dashed" instead of lty=2.

The line width argument lwd takes positive numerical values. The default value is 1. In the example we used a value of 2, thus making the lines thicker than default.

See also

We will explore more examples of line styles in subsequent chapters, especially Chapter 4, Creating Line Graphs and Time Series Charts in which we will see some advanced line graph recipes.

Choosing box styles

The styles of various boxes drawn in a graph such as the one around the plotting region and the legend can be adjusted in a similar way to the line styles we saw in the last recipe.

Getting ready

All you need to try out this recipe is to run R and type the recipe at the command prompt. You can also choose to save the recipe as a script so that you can use it again later on.

How to do it...

Let's say we want to make an L-shaped box around a graph, such that the default top and right borders are not drawn. We can do so using the `bty` argument in the `par()` command:

```r
par(bty="l")
plot(rnorm(100))
```

How it works...

The `bty` argument stands for box type and takes single characters in inverted commas as values. The resulting box resembles the corresponding upper case letter. For example, the default value is `o`, thus giving a box with all four edges. Other possible values are `l`, `7`, `c`, `u`, and `]`. If we do not wish to draw a box at all, we can set `bty` to `n`.

Note that setting `bty` to `n` doesn't suppress the drawing of axes. If we wish to suppress those too then we would also have to set `xaxt` and `yaxt` to `n`. Alternatively, we can simply set the `axes` argument to `FALSE` in the `plot()` function call.

Beyond the Basics: Adjusting Key Parameters

There’s more...

Box styles can be controlled in a finer way using the `box()` command. In addition to the `lty` and `lwd` arguments, we can also specify where the box should be drawn using the `which` parameter, which can take values of `plot`, `figure`, `inner`, and `outer`.

Let’s say we want to draw a graph with an L-shaped box for the plot area and a full box around the figure including the axis annotations and titles, then we can do:

```r
par(oma=c(1,1,1,1))
plot(rnorm(100), bty="l")
box(which="figure")
```

Note that we had to first set the outer margins by setting the `oma` argument with the `par()` function. We will learn more about this argument later in this chapter. If we did not set the outer margins, the box around the figure would be right at the edge of the plot and get cut off because the default margins are set to zero.

Adjusting axis annotations and tick marks

The default axis settings are often not adequate to deal with all kinds of data. For example, we may wish to change the number of tick marks along an axis or change the orientation of the annotations if they are too long to fit horizontally. In this recipe we will cover some settings which can be used to customize axes as per our requirements.

Getting ready

All you need to try out this recipe is to run R and type the recipe at the command prompt. You can also choose to save the recipe as a script so that you can use it again later on.

How to do it...

We can set the \texttt{xaxp} and \texttt{yaxp} arguments with the \texttt{par()} command to specify co-ordinates of the extreme tick marks and the number of intervals between tick marks in the form \texttt{c(min, max, n)}.

\begin{verbatim}
plot(rnorm(100), xaxp=c(0, 100, 10))
\end{verbatim}
How it works...

When xaxp or yaxp is not specified, R automatically calculates the number of tick marks and their values. By default, R extends the axis limits by adding 4% at each end and then draws an axis which fits within the extended range. This means that even if we set the axis limits using xlim or ylim, the graph corners don't exactly correspond with those values. To make sure they do, we need to change the axis style using the xaxs argument, which takes one of two possible values: r (regular or default) and i (internal). We need to set xaxs to i.

A vector of the form c(x1, x2, n) giving the co-ordinates of the extreme tick marks and the number of intervals between tick marks

There's more...

To change the orientation of axis value annotations, we need to set the las argument of the par() command. It takes one of four possible numeric values:

- 0: always parallel to the axis (default)
- 1: always horizontal
- 2: always perpendicular to the axis
- 3: always vertical

We can also use the axis() command to make a custom axis by specifying a number of arguments. The basic arguments are:

- side which takes numeric values (1=below, 2=left, 3=above and 4=right)
- at which takes a vector of co-ordinates where tick marks are to be drawn
- labels which takes a vector of tick mark annotations

We can separately set the line width for the axis lines and the tick marks by passing the lwd and lwd.ticks arguments respectively. Similarly colors can be set using the col and col.ticks arguments.

See also

We will come across various examples of custom axes in the following chapters as we explore more advanced recipes.

Formatting log axes

In scientific analysis, we often need to represent data on a logarithmic scale. In this recipe, we will see how we can do this easily in R.

Getting ready

All you need to try out this recipe is to run R and type the recipe at the command prompt. You can also choose to save the recipe as a script so that you can use it again later on.

How to do it...

The simplest way to make an axes logarithmic is to use the log argument in the `plot()` command:

```
plot(10^c(1:5), log="y", type="b")
```

How it works...

The `log` argument takes character values specifying which axes should be logarithmic: `x` for X axis only, `y` for Y axis only, and `xy` or `yx` for both axes.
Beyond the Basics: Adjusting Key Parameters

There's more...

We can also set scales to be logarithmic by setting the xlog and ylog arguments to TRUE with the par() command. This can be handy if we wish to have the same setting for multiple plots as par() applies the settings to all subsequent plots on the same device.

Note that R will not create the plot if our data contains zero or negative values.

Setting graph margins and dimensions

In this recipe we will learn how to adjust graph margins and dimensions.

Getting ready

All you need to try out this recipe is to run R and type the recipe at the command prompt. You can also choose to save the recipe as a script so that you can use it again later on.

How to do it...

We can use the fin and pin arguments of the par() command to set the figure region and plot dimensions:

```r
par(fin=c(6,6),
     pin=c(4,4))
```

We can use the mai and omi arguments to adjust the inner and outer margins respectively:

```r
par(mai=c(1,1,1,1),
     omi=c(0.1,0.1,0.1,0.1))
```

How it works...

All the previous arguments accept values in inches as a pair of width and height values. The default values for fin and pin are approximately 7x7 and 5.75x5.15. We have to be careful not to specify bigger values for pin than fin or we would get an error.

Adjusting fin and pin is one way of setting the figure margins containing the axis annotations and labels. Another way is to use the mai or mar arguments. In the example, we used mai which takes a vector value in inches, whereas mar takes a vector of numerical values in terms of number of lines of margins. It is better to use mar or mai because they adjust the figure margins irrespective of the figure or plot size.

We can also set an outer margin which is set to zero by default. This margin is useful if we wish to contain the entire graph including axis labels within a box as we saw in an earlier recipe. Like figure margins, outer margins can be set in inches with \texttt{omi} or in number of lines of text using \texttt{oma}.

\textbf{R Graphics} by Paul Murrell is an excellent reference with visual explanations of how margins work in R. See the book homepage for more details: \url{http://www.stat.auckland.ac.nz/~paul/RGraphics/rgraphics.html}.

This talk by Paul Murrell also contains figures from the book explaining the same concepts: \url{http://www.stat.auckland.ac.nz/~paul/Talks/Rgraphics.pdf}.

\section*{See also}

We will come across examples of figure margin settings in some of the recipes in the following chapters.

\section*{For More Information: \url{www.packtpub.com/r-graph-cookbook/book}}
Where to buy this book

You can buy R Graphs Cookbook from the Packt Publishing website:

Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our shipping policy.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.