

Chapter 4

4. Graphical Procedures

Objectives

After completing this unit, students will be able to:

- draw graphs using the base graphics packages,
- understand plotting commands
- identify graphics parameters

Graphical facilities are an important and extremely versatile component of the R environment.

It is possible to use the facilities to display a wide variety of statistical graphs and also to build entirely new types of graph.

3.1. Plotting commands

Once the device driver is running, R plotting commands can be used to produce a variety of graphical displays and to create entirely new kinds of display.

Plotting commands are divided into three basic groups:

- **High-level** plotting functions create a new plot on the graphics device, possibly with axes, labels, titles and so on.
- **Low-level** plotting functions add more information to an existing plot, such as extra points, lines and labels.
- **Interactive graphics** functions allow you interactively add information to, or extract information from, an existing plot, using a pointing device such as a mouse.

In addition, R maintains a list of graphical parameters which can be manipulated to customize your plots.

i. High-level plotting commands

High-level plotting functions are designed to generate a complete plot of the data passed as arguments to the function. Where appropriate, axes, labels and titles are automatically generated (unless you request otherwise). High-level plotting commands always start a new plot, erasing the current plot if necessary.

The `plot()` function

One of the most frequently used plotting functions in R is the `plot()` function. This is a generic function: the type of plot produced is dependent on the type or class of the first argument.

- `plot(x, y)`: If **x** and **y** are vectors, `plot(x, y)` produces a scatterplot of **y** against **x**.
- `plot(x)`: If **x** is a time series, this produces a time-series plot. If **x** is a numeric vector, it produces a plot of the values in the vector against their index in the vector.
- `plot(f)`: If **f** is a factor object, `plot(f)` generates a bar plot of **f**
- `plot(f, y)`: If **f** is a factor object, **y** is a numeric vector, this produces boxplots of **y** for each level of **f**.
- If **df** is a data frame, **y** is any object, **expr** is a list of object names separated by '+' (e.g., `a + b + c`), then
 - `plot(df)`: It produces distributional plots of the variables in a data frame.
 - `plot(~ expr)`: It produces distributional plots of a number of named objects.
 - `plot(y ~ expr)`: It produces plots **y** against every object named in **expr**.

Displaying multivariate data

R provides two very useful functions for representing multivariate data.

i. `pairs(X)`

If **X** is a numeric matrix or data frame, the command `pairs(X)` produces a pairwise scatterplot matrix of the variables defined by the columns of **X**, that is, every column of **X** is plotted against every other column of **X** and the resulting $n(n-1)$ plots are arranged in a matrix with plot scales constant over the rows and columns of the matrix.

ii. `coplot(a ~ b | c)`

When three or four variables are involved a *coplot* may be more enlightening. If **a** and **b** are numeric vectors and **c** is a numeric vector or factor object (all of the same length), then the command `coplot(a ~ b | c)` produces a number of scatterplots of **a** against **b** for given values of **c**. If **c** is a factor, this simply means that **a** is plotted against **b** for every level of **c**.

Display graphics

Other high-level graphics functions produce different types of plots. Here are some examples:

Distribution-comparison plots

qqnorm(x) : plots the numeric vector **x** against the expected Normal order scores (a normal scores plot)

qqline(x) : It adds a straight line to such a plot by drawing a line through the distribution and data quartiles.

qqplot(x, y) : plots the quantiles of **x** against those of **y** to compare their respective

hist(x) : Produces a histogram of the numeric vector **x**. A sensible number of classes is usually chosen

hist(x, nclass=n) : Produces a histogram of the numeric vector **x** with the **nclass=** argument

hist(x, breaks=b, ...) : Produces a histogram of the numeric vector **x**, the breakpoints can be specified exactly with the **breaks=** argument.

Arguments to high-level plotting functions

There are a number of arguments which may be passed to high-level graphics functions, as follows:

The **type=** argument controls the type of plot produced, as follows:

type="p" Plot individual points (the default)

type="l" Plot lines

type="b" Plot points connected by lines (both)

type="o" Plot points overlaid by lines

type="h" Plot vertical lines from points to the zero axis (high-density)

type="n" No plotting at all. However axes are still drawn (by default) and the coordinate system is set up according to the data. Ideal for creating plots with subsequent low-level graphics functions.

xlab=string and **ylab=string** : Axis labels for the **x** and **y** axes. Use these arguments to change the default labels, usually the names of the objects used in the call to the high-level plotting function.

main=string : Figure title, placed at the top of the plot in a large font.

sub=string : Sub-title, placed just below the x-axis in a smaller font.

ii. Low-level plotting commands

Sometimes the high-level plotting functions don't produce exactly the kind of plot you desire. In this case, low-level plotting commands can be used to add extra information (such as points, lines or text) to the current plot.

Some of the more useful low-level plotting functions are:

points(x, y) and **lines(x, y)**

Adds points or connected lines to the current plot. `plot()`'s **type=** argument can also be passed to these functions (and defaults to "p" for `points()` and "l" for `lines()`.)

text(x, y, labels, ...)

Add text to a plot at points given by x, y. Normally labels is an integer or character vector in which case labels[i] is plotted at point (x[i], y[i]). The default is 1:length(x).

Note: This function is often used in the sequence

```
> plot(x, y, type="n"); text(x, y, names)
```

The graphics parameter `type="n"` suppresses the points but sets up the axes, and the `text()` function supplies special characters, as specified by the character vector names for the points.

abline(a, b), **abline(h=y)**, **abline(v=x)**, and **abline(lm.obj)**

Adds a line of slope b and intercept a to the current plot. `h=y` may be used to specify y-coordinates for the heights of horizontal lines to go across a plot, and `v=x` similarly for the x-coordinates for vertical lines. Also `lm.obj` may be list with a coefficients component of length 2 (such as the result of model-fitting functions,) which are taken as an intercept and slope, in that order.

legend(x, y, legend, ...)

Adds a legend to the current plot at the specified position. Plotting characters, line styles, colors etc., are identified with the labels in the character vector legend. At least one other argument v (a vector the same length as legend) with the corresponding values of the plotting unit must also be given, as follows:

legend(, fill=v): Colors for filled boxes

legend(, col=v): Colors in which points or lines will be drawn

legend(, lty=v): Line styles

legend(, lwd=v): Line widths

legend(, pch=v) : Plotting characters (character vector)

title(main, sub)

Adds a title main to the top of the current plot in a large font and (optionally) a sub-title sub at the bottom in a smaller font.

Examples:

Simple Base Graphics: Histogram

```
x <- rnorm(100)
hist(x)
library(datasets)
hist(airquality$Ozone) ## Draw a new plot
hist(airquality$Ozone, col = "green")
```

Simple Base Graphics: Scatterplot

```
with(airquality, plot(Wind, Ozone))
```

Scatter plot with colors

```
with(iris, plot(Sepal.Length, Sepal.Width, col = Species, pch=19))
```

Simple Base Graphics: Times series plot

```
plot(AirPassengers)
plot(AirPassengers, type="o")
plot(AirPassengers, type="o", col= "blue")
plot(AirPassengers, type="o", col= "blue", lwd =2)
```

Simple Base Graphics: Boxplot

```
airquality <- transform(airquality, Month = factor(Month))
boxplot(Ozone~Month, airquality, xlab = "Month", ylab = "Ozone (ppb)")
```

Basic Scatterplot Matrix

```
>pairs(airquality, main ="Air Quality Data")
>pairs(~mpg+disp+drat+wt, data=mtcars,
      main="Simple Scatterplot Matrix")
```

coplot

```
coplot(mpg ~ disp | as.factor(cyl), data = mtcars, rows = 1)
```

Simple Barplot

```
barplot(table(mtcars$gear), col="wheat", main = "Number of cars in each
Number of forward gears")
```

Multiple Barplot

```
barplot(VADeaths, beside = TRUE, col = c("blue", "lightblue", "gray",
"lightgray", "lightcyan"), legend=rownames(VADeaths), ylim=c(0,100))
title(main = "Death Rates in Virginia", font.main = 4)
```

4.2. Graphics parameters

When creating graphics, particularly for presentation or publication purposes, R's defaults do not always produce exactly that which is required. You can, however, customize almost every aspect of the display using graphics parameters. R maintains a list of a large number of graphics parameters which control things such as line style, colors, figure arrangement and text justification among many others. Every graphics parameter has a name (such as 'col', which controls colors,) and a value (a color number, for example.)

Graphics parameters can be set in two ways: either permanently, affecting all graphics functions which access the current device; or temporarily, affecting only a single graphics function call.

1. Permanent changes: The `par()` function

The `par()` function is used to access and modify the list of graphics parameters for the current graphics device.

- `par()` Without arguments, returns a list of all graphics parameters and their values for the current device.
- `par(c("col", "lty"))` With a character vector argument, returns only the named graphics parameters (again, as a list.)
- `par(col=4, lty=2)` With named arguments (or a single list argument), sets the values of the named graphics parameters, and returns the original values of the parameters as a list.

Setting graphics parameters with the `par()` function changes the value of the parameters permanently, in the sense that all future calls to graphics functions (on the current device) will be affected by the new value.

You can restore the initial values by saving the result of `par()` when making changes, and restoring the initial values when plotting is complete.

```
> oldpar <- par(col=4, lty=2)
. . . plotting commands . . .
```

```
> par(oldpar)
```

Example—Set a graphical parameter using `par()`

```
par()                # view current settings
opar <- par()        # make a copy of current settings
par(col.lab="red")   # red x and y labels
hist(mtcars$mpg)     # create a plot with these new settings
par(opar)            # restore original settings
```

2. Temporary changes: Arguments to graphics functions

Graphics parameters may also be passed to (almost) any graphics function as named arguments.

This has the same effect as passing the arguments to the `par()` function, except that the changes only last for the duration of the function call. For example:

```
> plot(x, y, pch="+")
```

produces a scatterplot using a plus sign as the plotting character, without changing the default plotting character for future plots.

Some Important Base Graphics Parameters

Many base plotting functions share a set of parameters. Here are a few key ones:

- **pch**: the plotting symbol (default is open circle)
- **lty**: the line type (default is solid line), can be dashed, dotted, etc.
- **lwd**: the line width, specified as an integer multiple
- **col**: the plotting color, specified as a number, string, or hex code; the `colors()` function gives you a vector of colors by name
- **xlab**: character string for the x-axis label
- **ylab**: character string for the y-axis label

The `par()` function is used to specify global graphics parameters that affect all plots in an R session. These parameters can be overridden when specified as arguments to specific plotting functions.

- **las**: the orientation of the axis labels on the plot
- **bg**: the background color
- **mar**: the margin size

- **oma**: the outer margin size (default is 0 for all sides)
- **mfrow**: number of plots per row, column (plots are filled row-wise)
- **mfcol**: number of plots per row, column (plots are filled column-wise)

Default values for global graphics parameters

- `par("lty")`
[1] "solid"
- `par("col")`
[1] "black"
- `par("pch")`
[1] 1·
- `par("bg")`
[1] "transparent"
- `par("mar")`
[1] 5.1 4.1 4.1 2.1
- `par("mfrow")`
[1] 1 1

• Base Plotting Functions

- **plot**: make a scatterplot, or other type of plot depending on the class of the object being plotted
- **lines**: add lines to a plot, given a vector x values and a corresponding vector of y values (or a 2-column matrix); this function just connects the dots
- **points**: add points to a plot
- **text**: add text labels to a plot using specified x, y coordinates
- **title**: add annotations to x, y axis labels, title, subtitle, outer margin
- **mtext**: add arbitrary text to the margins (inner or outer) of the plot
- **axis**: adding axis ticks/labels

Examples - Base Plot with Annotation

```
1. library(datasets)
   with(airquality, plot(Wind, Ozone))
   title(main = "Ozone and Wind in New York City") ## Add a title
```

```

2. with(airquality, plot(Wind, Ozone,
    main = "Ozone and Wind in New York City"))
    with(subset(airquality, Month==5), points(Wind, Ozone, col="blue"))
3. with(airquality, plot(Wind, Ozone, main = "Ozone and Wind in New
    York City", type = "n"))
    with(subset(airquality, Month==5), points(Wind, Ozone, col="blue",
    pch=19))
    with(subset(airquality, Month!= 5), points(Wind, Ozone, col = "red",
    pch=19))
    legend("topright", pch = 19, col = c("blue", "red"), legend =
    c("May", "Other Months"))

```

Example - Base Plot with Regression Line

```

with(airquality, plot(Wind, Ozone, main="Ozone and Wind in New York
City", pch = 20))
model <- lm(Ozone ~ Wind, airquality)
abline(model, lwd = 2)

```

Combining Plots

R makes it easy to combine multiple plots into one overall graph, using either the `par()` or `layout()` function. With the `par()` function, you can include the option `mfrow=c(nrows, ncols)` to create a matrix of $nrows \times ncols$ plots that are filled in by row. `mfcow=c(nrows, ncols)` fills in the matrix by columns.

Example - 4 figures arranged in 2 rows and 2 columns

```

attach(mtcars)
par(mfrow=c(2,2))
plot(wt, mpg, main="Scatterplot of wt vs. mpg")
plot(wt, disp, main="Scatterplot of wt vs disp")
hist(wt, main="Histogram of wt")
boxplot(wt, main="Boxplot of wt")

```

Example

```

par(mfrow = c(1, 2))
with(airquality, {
plot(Wind, Ozone, main = "Ozone and Wind")

```

```
plot(Solar.R, Ozone, main = "Ozone and Solar Radiation")
})
```

Example

```
par(mfrow = c(1, 2))
with(airquality, {
  plot(Wind, Ozone, main = "Ozone and Wind")
  plot(Solar.R, Ozone, main = "Ozone and Solar Radiation")
})
```

Viewing Several Graphs

Creating a new graph by issuing a high level plotting command (plot, hist, boxplot, etc.) will typically **overwrite** a previous graph. To avoid this, open a new graph window before creating a new graph.

To open a new graph window use **dev.new()** or **windows()**