CHAPTER 1

BASIC GRAPHICS

1.1 INTRODUCTION

This chapter first describes the procedures to start up R, produce R programs, and run them. This is followed by explanations of R programs used to construct simple graphs. New techniques are introduced by adding new methods to already-known materials. Hence, learning is followed by consulting, testing, and modifying the listed R programs sequentially. Alternatively, learners can find graphs that roughly suit their purpose and experiment with them. Previous articles may be referred to if unknown commands or functions are used in the programs. In addition, the last part of this chapter introduces techniques to share displayed graphs with other application software packages and to save graphs as digital files.

1.2 DOWNLOADING AND INSTALLATION OF R

The procedure below installs R on a PC loaded with a Windows OS.

2. Click "CRAN" under "Download, Packages" listed in the menu on the left side.

3. Choose one of the mirror sites on the "CRAN Mirrors" page. Most of the mirror sites are identical. For example, choose University of Tsukuba (http://cran.md.tsukuba.ac.jp/).

4. Click "Windows" under "Download and Install R" on "The Comprehensive R Archive Network" web page.

5. Click "base" on the same line as "base Binaries for base distribution (managed by Duncan Murdoch)" on the "R for Windows" page.

6. Click "Download R 2.11.0 for Windows (32 megabytes)" (the version may be different) on the "R-2.11.0 for Windows" page.

7. R-2.11.0-win32.exe (or a different version) is now available for installation. Construct a folder in the hard disk in your PC and download the file to the folder.

8. Double-click R-2.11.0-win32.exe to start the installation process of R.

9. Agree to the "GNU General public license" and specify the location of installation.

10. Select "Full installation" on the "Select components" page.

11. After a few further selections, installation starts.

Sets of programs called "packages" have been prepared for R. Packages can be added to the version of R installed on your PC. If a PC is connected to the Internet, select "Packages" in a menu after R is booted. Then, choose "Set CRAN mirror...". Many mirror sites appear. Then, choose "Japan (Tsukuba)", for example, and click "OK". Return to "Packages" in the menu. Then, select "Load package" to display the names of many packages. Choose the names of packages that you need and click "OK". The selected packages are installed in this way.

If packages are installed in this manner, the names of packages displayed are those that fit the version of R installed on a PC. However, some packages that fit an older version of R can be used in a newer version of R. For example, the "gtools" package that will be dealt with later fits version 2.10 of R but does not fit version 2.11 of R. Hence, when the "gtools" package is used, it is safe to employ version 2.10 of R. However, the use of the "gtools" package with version 2.11 of R can be attempted. For this purpose, the method described above cannot be used. A method for a PC that is not connected to the
Internet is useful for installing packages for older versions of R. This method is as follows:

1. In the fifth step of the installation procedure of R described above, click "contrib" on the same line of "contrib Binaries of contributed packages (managed by Uwe Ligges)".

2. The "Index of /bin/windows/contrib" page appears. Select a version of R that fits the packages that you will use (for example, 2.10).

3. The names of the files of packages compressed in zip format are displayed. Click the names of the files of packages needed for downloading them.

4. R is booted. Select "Packages" in the menu. Then, select "Install package(s) from local zip files...".

5. Select the files that were downloaded beforehand to install packages.

Using this method, even if R version 2.11 is installed on a PC, packages for R version 2.10 may be used.

1.3 START-UP OF R, AND CONSTRUCTION AND EXECUTION OF R PROGRAMS

We assume that the folder used for storing data files and the results of calculations is D:\GraphicsR. Save all files created with R in this folder.

![R Logo and .RData](image.png)

**Figure 1.1** Work image file.

The file for storing R programs is called the "work image file" (Fig. 1.1). Confirm the presence of this file in D:\GraphicsR. If the work image file is not located in D:\GraphicsR, search for a file named ".RData" (which will be somewhere in your hard disk if R has been installed correctly) and copy the file by saving it in D:\GraphicsR. If the drive where .RData is originally located is the "D" drive, hold the Ctrl key and copy the file. If the Ctrl key is not held, the file is not copied but moved. That is, the original .RData is deleted. When multiple .RData files are placed in the same PC, each .RData
file stores its own R programs. Even if the version of R is upgraded, the same .RData file can be used.

R is booted by double-clicking .RData in D:\GraphicsR. R can also be booted by double-clicking the shortcut button assigned to .RData on the desktop or other places. Upon booting R, the window shown in Fig. 1.2 appears.

The inner window in Fig. 1.2 is called the console window. In this window, R commands and R programs are executed. For example, when an R program named rpro1() is produced, type `fix(rpro1)` and click the return key in the console window (Fig. 1.3). Then, the display in Fig. 1.4 appears. The new
window is an editor. When R is installed in the standard manner, "notepad" is used as an editor. However, users can set up another editor for this purpose. An R program is a series of R commands listed between "function () {" and "}". Arguments can be specified in "()" of "function () {". For example, function (aa) {} is an R program that uses aa as an argument. By setting numerical values or text as aa, an R program with this argument is run.

![Figure 1.4 Editor.](image)

![Figure 1.5 R program.](image)

Let us produce an R program for multiplying 5 and 3 and displaying the result in the console window. Fig. 1.5 shows an example of an R program for this purpose. The editor is closed to execute this R program. For this, "x" (Fig. 1.6) located of the upper right of the editor is clicked. Then, a dialogue box asking for a selection is displayed (Fig. 1.7). "Yes" or the return key is clicked. Only the console window is then displayed (Fig. 1.8). rproc1() is
typed to execute the R program `rpro1()` (Fig. 1.9). Then, the return key is clicked to execute `rpro1()` and the result “15” appears (Fig. 1.10). This series of procedures summarizes the basic use of R: the start-up of “R”, the construction of R programs, and the execution of R programs.

![Figure 1.7 Dialogue box asking for selection.](image)

![Figure 1.8 Return to the console window.](image)

When graphics windows are displayed in the console window (Fig. 1.11), the execution of `graphics.off()` in the console window (Fig. 1.12) clears all graphics windows, and the console window remains (Fig. 1.13). If R is shut down and rebooted, only the console window appears. Even if only the
Figure 1.9 Execution of an R program.

Figure 1.10 Result of executing an R program.
Figure 1.11  Graphics windows are displayed in addition to the console window.

Figure 1.12  `graphics.off()` is executed in the console window.
console window is displayed, the configurations of previous graphs may affect new graphs. The shutdown and rebooting of R solves this problem.

R uses Notepad as the standard editor by default. Other editors, however, can be used. For example, “Programmer’s Notepad” (http://www.pnotepad.org/) is available as an editor: “Programmer’s Notepad” has various useful functions. For instance, the setting of “View” - “Change Scheme” - “C / C++” (Fig. 1.14) gives the edit screen such as Fig. 1.15. The structure of the parentheses is shown clearly.

The following R program (fixp()) is useful for employing “Programmer’s Notepad” as an editor:

Program (1 - 1)
function (x, ...)
{
  subx <- substitute(x)
  if (is.name(subx))
    subx <- deparse(subx)
  if (!is.character(subx) || length(subx) != 1)
    stop("'fix' requires a name")
  parent <- parent.frame()
  if (exists(subx, envir = parent, inherits = TRUE))
    x <- edit(get(subx, envir = parent), title = "temp",
               editor="C:\Program Files\Programmer's Notepad\pn.exe", ...)
  else {
    x <- edit(function() {
      }, title = subx, ...)
    environment(x) <- .GlobalEnv
}
Figure 1.14  Setting of "View" - "Change Scheme" - "C / C++" in "Programmer's Notepad".

Figure 1.15  Edit screen given by setting Fig. 1.14.
Figure 1.16 Construction of R program using "Programmer's Notepad".

\}
assign(subx, x, envir = .GlobalEnv)
\}

This R program is obtained by modifying fix(), which is implemented in R by default. When an R program is edited using fixp(), fixp() is carried out on a console window in the same manner as that of using fix()(Fig. 1.16).

1.4 COORDINATE AXES

Fig. 1.17 shows the result of Program (1 - 2).

Program (1 - 2)
function() {
  # (1)
  par(mai = c(1, 1, 1, 1), omi = c(0, 0, 0, 0))
  # (2)
  plot(x = c(0, 1), y = c(0, 1), xlab = "x", ylab = "y")
}

(1) par() sets the structure of R graphics and outputs the setting. In this example, a graphics area is set in a graphics window. The meanings of omi = and mai = are illustrated in Fig. 1.18. omi = sets the size of the outer margin surrounding the graphics area. The four values from left to right indicate the sizes of the lower, left, upper, and right margins of the graphics area (the outer rectangle in Fig. 1.18). mai = sets the size of the figure margin. The four values from left to right indicate the sizes of the lower, left, upper, and right margins of the figure area (the inner rectangle in Fig. 1.18). When values or
Figure 1.17  Coordinate axes and two data points given by Program (1 - 2).

Figure 1.18  Structure of a graphics window.
labels are written on the lower or left side of a line chart or scatter plot, the first two values of \texttt{mai} = are usually set to be larger than the last two values. The unit for \texttt{omi} = and \texttt{mai} = is inches. However, when graphs are drawn on a display, these values indicate relative lengths because the sizes of graphs can be changed at will.

(2) Fig. 1.17 shows that \texttt{plot()} illustrates a graph with two data points at (0,0) and (1,1). Furthermore, \texttt{xlab = "x"} adds the label “x” on the x-axis of the graph and \texttt{ylab = "y"} adds the label “y” on the y-axis of the graph.

![Graph with two data points](image)

**Figure 1.19** Coordinate axes and two data points drawn by Program (1 - 3).

Fig. 1.19 is identical to Fig. 1.17 and is constructed by Program (1 - 3).

Program (1 - 3)

```r
function() {
  # (1)
  par(mai = c(1, 1, 1, 1), omi = c(0, 0, 0, 0))
  # (2)
  plot(c(0, 1), c(0, 1), xlab = "x", ylab = "y")
}
```

In part (2), the first two arguments of \texttt{plot()} \((x = c(0, 1), y = c(0, 1))\) are replaced with \(c(0, 1), c(0, 1)\). In R commands, the names of arguments can usually be abbreviated if arguments are given in a preset order.

Program (1 - 4) produces Fig. 1.20 (left).

Program (1 - 4)

```r
function() {
  # (1)
```
Figure 1.20  Coordinate axes and two data points drawn by Program (1 - 4) (left). Coordinate axes and two data points drawn by Program (1 - 5) (right).

```
par(mai = c(1, 1, 1, 1), omi = c(0, 0, 0, 0))
# (2)
plot(c(-2, 3), c(2, 9), xlab = "x", ylab = "y")

The first two arguments of part (2) are replaced with c(-2, 3), c(2, 9). This alteration changes the positions of the two points to (-2, 2) and (3, 9).

Program (1 - 5) constructs Fig. 1.20 (right).

Program (1 - 5)
function() {
  # (1)
  par(mai = c(1, 1, 1, 1), omi = c(0, 0, 0, 0))
  # (2)
  plot(c(-2, 3), c(2, 9), xlab = "x", ylab = "y", type = "n")
}

In part (2) in Program (1 - 4), type = "n" is added to the arguments of plot(). This removes the two data points so that only the coordinate axes remain. This technique draws the coordinate axes only. The ranges of the coordinate axes are set by the first two arguments.

1.5 POINTS AND STRAIGHT LINES

Program (1 - 6) produces Fig. 1.21 (left).

Program (1 - 6)
Figure 1.21  Straight line and data points drawn by Program (1 - 6) (left). Straight line and data points drawn by Program (1 - 7) (right).

```r
function() {
  # (1)
  par(mai = c(1, 1, 1, 1), omi = c(0, 0, 0, 0))
  # (2)
  plot(c(-2, 0, 3), c(2, 3, 9), xlab = "", ylab = "", type = "n", axes = F)
  # (3)
  lines(c(-2, 0, 3), c(2, 3, 9))
  # (4)
  points(c(-2, 0, 3), c(2, 3, 9))
}
```

(1) The graphics area is set.
(2) Three points, (−2, 2), (0, 3), and (3, 9), are set by the first two arguments in `plot()`. Similarly, the first two arguments in `plot()` can specify more than three points. The coordinate axes are set to position all the specified points appropriately in the graph. The remaining specifications of the arguments in `plot()` are set to not label the axes, the positions specified by the first two arguments, or the coordinate axes.
(3) Straight lines connecting the three points ((−2, 2), (0, 3), (3, 9)) are sequentially drawn.
(4) Small circles at (−2, 2), (0, 3), and (3, 9) are drawn.

Program (1 - 7) constructs Fig. 1.21 (right). The resultant graph is identical to that in Fig. 1.21 (left).

Program (1 - 7)

```r
function() {
```
#(1)
par(mai = c(1, 1, 1, 1), omi = c(0, 0, 0, 0))
# (2)
xx <- c(-2, 0, 3)
yy <- c(2, 3, 9)
# (3)
plot(xx, yy, xlab = "", ylab = "", type = "n", axes = F)
# (4)
lines(xx, yy)
# (5)
points(xx, yy)

This R program differs from Program (1 - 6) in that \((-2, 0, 3)\) is stored in xx in (2) and \((2, 3, 9)\) is stored in yy. xx and yy are variables (called "objects" in the terminology of R). Parts (3)(4)(5) use xx and yy as arguments. This R program produces the same graph as that in Fig. 1.21 (left) by specifying three data points simultaneously.

1.6 REUSE OF GRAPHS PRODUCED BY R

![Figure 1.22 Copy of a graph as a metafile.](image-url)
Figure 1.23  Copy of the metafile in a word processor document.

Figure 1.24  Completion of the copy of the metafile in a word processor document.
The graphs produced by R can be arranged using other programs of a personal computer or pasted in a document. For pasting graphs in a graphics window in a document of other programs, “Copy as metafile” is used. That is, a right mouse button is used click on the graphics window and “Copy as metafile” is selected with a left-click (Fig. 1.22). Then, “copy” is performed on a targeted document of the software (a word processor in this example (Fig. 1.23). Now, the paste of the graph constructed by R is completed (Fig. 1.24).

However, if a graph on the graphics window is copied directly, it is cumbersome to unify the lengths or ratios of the vertical axes to the horizontal axes, or the fonts. In this regard, it is useful to save the graphs produced by R and retrieve them as occasion arises. There are diverse formats of digital files available. Postscript files, jpeg (Joint Photographic Experts Group), and pdf (Portable Document Format) are typical examples.

![Graph](image)

**Figure 1.25** Display of a postscript file given by Program (1 - 8).

To construct digital files of graphs in postscript format, Program (1 - 8), for example, is useful. The digital file `ps1.ps` is displayed in Fig. 1.25.

```r
Program (1 - 8)
function() {
  # (1)
  postscript(file = "d:\GraphicsR\ps1.ps", horiz = F,
             width = 5, height = 5)
  # (2)
  par(mai = c(1, 1, 0.1, 0.1), omi = c(0.2, 0.2, 0.2, 0.2))
  # (3)
  plot(c(-2, 3), c(2, 9), xlab = "x", ylab = "y", type = "n")
  lines(c(-2, 3), c(2, 9))
```
# (4)
  graphics.off()
}

(1) The command `postscript()` describes that digital files in postscript format is produced. The file name is set as `sps1.ps`. The folder for this file is `d:\GraphicsR`. If `horiz = F` is not specified, resultant graphs are rotated by 90 degrees. Hence, this setting is requisite. The arguments `width = 5` and `height = 5` set the size of the image. The unit is inches.

(2) The graphics area is set.

(3) A graph is drawn.

(4) The task of producing a digital file terminates.

![Postscript Image]

**Figure 1.26** Display of a postscript file, given by Program (1 - 9).

When text is contained in a graph to be saved as a postscript file, the setting may be different from that for display in a graphics window. For example, Program (1 - 9) yields a postscript file shown in Fig. 1.26.

Program (1 - 9)

```r
function() {
  # (1)
  postscript(file = "d:\GraphicsR\ps2.ps", horiz = F)
  # (2)
  par(mai = c(1, 1, 1, 1), omi = c(6, 0, 0, 2))
  # (3)
  plot(c(0, 1), c(0, 1), xlab = "", ylab = "", type = "n",
       axes = F)
  # (4)
  text(0.5, 0.8, "font=1, abcddef", cex = 2, font = 1)
  text(0.5, 0.6, "font=2, abcddef", cex = 2, font = 2)
```
text(0.5, 0.4, "font=3, abcdef", cex = 2, font = 3)
text(0.5, 0.2, "font=4, abcdef", cex = 2, font = 4)
# (5)
graphics.off()
}

(1) The command postscript() describes the production of a digital file in postscript format.
(2) The graphics area is set.
(3) The coordinate axes are set.
(4) Text is written. Since the name of the font is not specified in (1), Helvetica is employed. However, another font may be used if the OS setting is different.
(5) The task of producing a digital file terminates.

![Postscript Image]

**Figure 1.27** Display of a postscript file, given by Program (1 - 10).

The name of the font can be specified when a postscript file is constructed. The digital file `ps2b.ps` (in postscript format) produced using this method realizes Fig. 1.27 given by Program (1 - 10).

Program (1 - 10)
function() {
# (1)
  postscript(file = "d:\GraphicsR\ps2b.ps", horiz = F,
              family = "Times")
# (2)
  par(mai = c(1, 1, 1, 1), omi = c(6, 0, 0, 2))
# (3)
  plot(c(0, 1), c(0, 1), xlab = "", ylab = "", type = "n",
       axes = F)
# (4)
text(0.5, 0.8, "font=1, abcdef", cex = 2, font = 1)
text(0.5, 0.6, "font=2, abcdef", cex = 2, font = 2)
text(0.5, 0.4, "font=3, abcdef", cex = 2, font = 3)
text(0.5, 0.2, "font=4, abcdef", cex = 2, font = 4)
# (5)
graphics.off()
}

(1) The command postscript() describes the production of a digital file in postscript format. The command family = "Times" specifies Times as the font here. The font Times is not available if the OS setting is different.
(2) The graphics area is set.
(3) The coordinate axes are set.
(4) Text is written. Since family = "Times" is set in (1), Times is employed as the font.
(5) The task of producing a digital file terminates.

Figure 1.28  Display of a jpeg file, given by Program (1 - 11).

As a format of a digital file for representing graphs, the jpeg format is also available. For example, Program (1 - 11) produces a jpeg file shown in Fig. 1.28.

Program (1 - 11)
function() {
  # (1)
  jpeg(file = "d:\GraphicsR\jpeg1.jpg", width = 400, height = 400)
  # (2)
par(mai = c(1, 1, 0.1, 0.1), omi = c(0.2, 0.2, 0.2, 0.2))
# (3)
plot(c(-2, 3), c(2, 9), xlab = "x", ylab = "y", type = "n")
# (3)
lines(c(-2, 3), c(2, 9))
# (4)
graphics.off()
}

(1) The command jpeg() describes the production of a digital file in jpeg format. The name of the file is set as jpeg1.ps. The name of the folder for saving the file is set as d:\\GraphicsR. The arguments width = 400 and height = 400 set the size of the image. The unit is pixel.
(2) The graphics area is set.
(3) A graph is drawn.
(4) The task of producing a digital file terminates.

Figure 1.29 Display of the pdf file given by Program (1 - 12).

As a digital file for representing graphs, the pdf format is also available. For example, Program (1 - 12) produces a pdf file shown in Fig. 1.29.

Program (1 - 12)
function() {
  # (1)
  pdf(file = "d:\\GraphicsR\\pdf1.pdf", width = 5, height = 5)
  # (2)
  par(mai = c(1, 1, 0.1, 0.1), omi = c(0.2, 0.2, 0.2, 0.2))
  # Drawing a graph
  # Plotting a line
  # Setting the graphics area
  # Ending the graphics file
}
# (3)
plot(c(-2, 3), c(2, 9), xlab = "x", ylab = "y", type = "n")
lines(c(-2, 3), c(2, 9))
# (4)
graphics.off()
}

(1) The command pdf() describes the production of a digital file in pdf format. The name of the file is set as pdf1.ps. The name of the folder for saving the file is set as d:\\GraphicsR. The arguments width = 5 and height = 5 set the size of the image. The unit is inches.
(2) The graphics area is set.
(3) A graph is drawn.
(4) The task of producing a digital file terminates.

![Graphs](image)

**Figure 1.30** Display of the pdf file constructed by Program (1 - 13).

One pdf file may consist of plural pages, each of which contains graphs. For example, Program (1 - 13) produces a pdf file shown in Fig. 1.30.

Program (1 - 13)
function() {
# (1)
pdf(file = "d:\\GraphicsR\\pdf2.pdf", width = 5, height = 5)
# (2)
par(mai = c(1, 1, 0.1, 0.1), omi = c(0.2, 0.2, 0.2, 0.2))
# (3)
plot(c(-2, 3), c(2, 9), xlab = "x", ylab = "y")
lines(c(-2, 3), c(2, 9))
# (4)
par(mai=c(1, 1, 0.1, 0.1), omi=c(0.2, 0.2, 0.2, 0.2))
# (5)
plot(c(1, 2, 3), c(1, 8, 2), xlab = "x", ylab = "y")
lines(c(1, 2, 3), c(1, 8, 2))

# (6)
graphics.off()

(1) The command pdf() describes the production of a digital file in pdf format.
(2) The graphics area for the first graph is set.
(3) The graph for the first page is drawn.
(4) The graphics area for the second graph is set.
(5) The graph for the second page is drawn.
(6) The task of producing a digital file terminates.

1.7 TEXT

![Images showing graphs and text]

Figure 1.31 Coordinate axes, data points, and text drawn by Program (1 - 14) (left). Coordinate axes and text arranged in a circular pattern drawn by Program (1 - 15) (right).

Text as well as straight lines and small circles can be written in the graphics area defined by plot(). Fig. 1.31 (left), produced by Program (1 - 14), shows an example of this.

Program (1 - 14)

function() {
  # (1)
  par(mai = c(1, 1, 1, 1), omi = c(0, 0, 0, 0))
  # (2)
  plot(c(0, 1), c(0, 1), xlab = "", ylab = "", type = "n")
  # (3)
points(0.5, 0.8)
text(0.5, 0.8, "World Peace", cex = 2)
# (4)
points(0.5, 0.6)
text(0.5, 0.6, "World Peace", cex = 2, adj = c(0.5, 0.5))
# (5)
points(0.5, 0.4)
text(0.5, 0.4, "World Peace", cex = 2, adj = c(1, 0))
# (6)
points(0.5, 0.2)
text(0.5, 0.2, "World Peace", cex = 2, adj = c(0, 1))
}

(1) The graphics area is set.
(2) The coordinate axes are set and drawn.
(3) A small circle is drawn at (0.5,0.8). “World Peace” is written at the same place. cex = 2 specifies the size of the letters. A comparison between the position of the small circle and that of the text shows that the coordinates in text() indicate the position of the text.
(4) A small circle is drawn at (0.5,0.6), and “World Peace” is written at the same place. adj = c(0.5, 0.5) is set in text(). However, adj = c(0.5, 0.5) does not affect the position of the text.
(5) A small circle is drawn at (0.5,0.4), and “World Peace” is written at the same place. This time, adj = c(1, 0) is set in text(). Here, the position set in text() is located in the lower right-hand corner of the text.
(6) A small circle is drawn at (0.5,0.2), and “World Peace” is written at the same place. In this case, adj = c(0, 1) is set in text(). The position specified in text() is placed in the lower left-hand corner of the text.

Text can be rotated as shown in Fig. 1.31 (right), which is produced by Program (1 - 15).

Program (1 - 15)
function() {
  # (1)
  par(mai = c(1, 1, 1, 1), omi = c(0, 0, 0, 0), cex = 1.5)
  # (2)
  plot(c(0, 1), c(0, 1), xlab = "", ylab = "", type = "n")
  # (3)
  points(0.5, 0.5)
  # (4)
  text(0.5, 0.5, "adj=-0.1", cex = 1, adj = -0.1, srt = 45)
  text(0.5, 0.5, "adj=-0.2", cex = 1, adj = -0.2, srt = 90)
  text(0.5, 0.5, "adj=-0.3", cex = 1, adj = -0.3, srt = 135)
  text(0.5, 0.5, "adj=-0.4", cex = 1, adj = -0.4, srt = 180)
  text(0.5, 0.5, "adj=-0.5", cex = 1, adj = -0.5, srt = 225)
  text(0.5, 0.5, "adj=-0.6", cex = 1, adj = -0.6, srt = 270)
(1) The graphics area is set.
(2) The coordinate axes are drawn.
(3) A small circle is drawn at (0.5, 0.5).
(4) The settings of adj = and srt = in text() are modified in various ways to rotate the text.

![Equation Image]

Figure 1.32 Equations written by Program (1-16).

The use of expression() enables mathematical equations to be written in the area set by plot() as shown in Fig. 1.32, which is constructed by Program (1-16)

Program (1-16)

function() {
  # (1)
  par(mai = c(1, 1, 1, 1), omi = c(0, 0, 0, 0))
  # (2)
  plot(c(0, 1), c(0, 1), xlab = "", ylab = "", type = "n")
  # (3)
  text(0.5, 0.9, expression(lim(g(x), x \to 0)), cex = 1.2)
  # (4)
  text(0.5, 0.7, expression(bar(\epsilon) = \sum_{i=1}^{n} \frac{p[i]}{M})), cex = 1.2)
  # (5)
  text(0.5, 0.5, expression(bold(y)^a = paste("\(\plain{\sin}, \\
"\plain{\phi}, \\
"\plain{\cos}\)"))), cex = 1.2)
# (6)

```r
text(0.5, 0.3, expression(T == paste(prod(plain(Q), i),
    "(" , xi[i] , ")")), cex = 1.2)
```

# (7)

```r
text(0.5, 0.1, expression(Omega == paste(integral(sqrt(q(x) , 3),
    1, infinity), "dx")), cex = 1.2)
```

The execution of help("plotmath") in the console window of R shows the
list of mathematical expressions that can be used as arguments of expression().

### 1.8 VARIOUS POINTS AND STRAIGHT LINES

![Diagram of various line widths and types](image)

**Figure 1.33** Straight lines with various widths drawn by Program (1 - 17) (left). Various types of straight lines drawn by Program (1 - 18) (right).

The R programs presented so far have not specified the widths of straight lines in lines(). Hence, the thinnest line is selected. To illustrate how the width of a straight line is specified, Program (1 - 17) is executed to draw Fig. 1.33 (left).

Program (1 - 17)

```r
function() {
  # (1)
  par(mai = c(1, 1, 1, 1), omi = c(0, 0, 0, 0))
  # (2)
  plot(c(0, 1), c(0, 1), xlab = "", ylab = "", type = "n",
    axes = F)
  # (3)
  text(0.1, 1, "lwd=1")
  lines(c(0.3, 1), c(1, 1), lwd = 1)
```
# (4)
  text(0.1, 0.8, "lwd=2")
  lines(c(0.3, 1), c(0.8, 0.8), lwd = 2)
# (5)
  text(0.1, 0.6, "lwd=3")
  lines(c(0.3, 1), c(0.6, 0.6), lwd = 3)
# (6)
  text(0.1, 0.4, "lwd=4")
  lines(c(0.3, 1), c(0.4, 0.4), lwd = 4)
# (7)
  text(0.1, 0.2, "lwd=5")
  lines(c(0.3, 1), c(0.2, 0.2), lwd = 5)
# (8)
  text(0.1, 0.0, "lwd=6")
  lines(c(0.3, 1), c(0.0, 0.0), lwd = 6)
}

(1) The graphics area is set.
(2) The coordinate axes are set but not labeled.
(3) "lwd=1" is written at (0.1, 1). A straight line is drawn on the right side. lwd = 1 is specified as an argument of lines(). Even if lwd = is not set, the same straight line is obtained.
(4)(5)(6)(7)(8) Straight lines are drawn with various values of lwd = ranging from 2 to 6.

In addition, the type of straight line as well as its width can be specified as an argument of lines(). Program (1 - 18) illustrates this function as shown in Fig. 1.33 (right).

Program (1 - 18)
function() {
  # (1)
  par(mai = c(1, 1, 1, 1), omi = c(0, 0, 0, 0))
  # (2)
  plot(c(0,1), c(0,1), xlab = "", ylab = "", type = "n", axes = F)
  # (3)
  text(0.0, 1, 'lty=1, lty="solid"', adj = c(0, 0.5))
  lines(c(0.6, 1), c(1, 1), lwd = 2, lty = "solid")
  # (4)
  text(0.0, 0.8, 'lty=2, lty="dashed"', adj = c(0, 0.5))
  lines(c(0.6, 1), c(0.8, 0.8), lwd = 2, lty = "dashed")
  # (5)
  text(0.0, 0.6, 'lty=3, lty="dotted"', adj = c(0, 0.5))
  lines(c(0.6, 1), c(0.6, 0.6), lwd = 2, lty = "dotted")
  # (6)
  text(0.0, 0.4, 'lty=4, lty="dotted"', adj = c(0, 0.5))
lines(c(0.6, 1), c(0.4, 0.4), lwd = 2, lty = "dotdash")
# (7)
text(0.0, 0.2, 'lty=5, lty="longdash"', adj = c(0, 0.5))
lines(c(0.6, 1), c(0.2, 0.2), lwd = 2, lty = "longdash")
# (8)
text(0.0, 0.0, 'lty=6, lty="twodash"', adj = c(0, 0.5))
lines(c(0.6, 1), c(0.0, 0.0), lwd = 2, lty = "twodash")
}

(1) The graphics area is set.
(2) The coordinate axes are set but not labeled.
(3)(4)(5)(6)(7)(8) Straight lines are drawn with various values of lty = ranging from 1 to 6. These numbers in lines() can be replaced with words. The relationships between numbers and words are added.

```
col=1
```
```
col=2
```
```
col=3
```
```
col=4
```
```
col=5
```
```
col=6
```
```
col=7
```
```
col=8
```

**Figure 1.34** Straight lines of various colors given by Program (1 - 19).

Furthermore, the color of a straight line can be specified as an argument of lines() as demonstrated by Program (1 - 19), which constructs Fig. 1.34.

Program (1 - 19)
```
function() {
  # (1)
  par(mai = c(1, 1, 1, 1), omi = c(0, 0, 0, 0))
  # (2)
  plot(c(-0.4, 1), c(-0.4, 1), xlab = "", ylab = "", type = "n",
       axes = F)
  # (3)
  text(0.1, 1, "col=1")
  lines(c(0.3, 1), c(1, 1), lwd = 2, col = 1)
  # (4)
  text(0.1, 0.8, "col=2")
```
lines(c(0.3, 1), c(0.8, 0.8), lwd = 2, col = 2)
# (5)
text(0.1, 0.6, "col=3")
lines(c(0.3, 1), c(0.6, 0.6), lwd = 2, col = 3)
# (6)
text(0.1, 0.4, "col=4")
lines(c(0.3, 1), c(0.4, 0.4), lwd = 2, col = 4)
# (7)
text(0.1, 0.2, "col=5")
lines(c(0.3, 1), c(0.2, 0.2), lwd = 2, col = 5)
# (8)
text(0.1, 0.0, "col=6")
lines(c(0.3, 1), c(0.0, 0.0), lwd = 2, col = 6)
# (9)
text(0.1, -0.2, "col=7")
lines(c(0.3, 1), c(-0.2, -0.2), lwd = 2, col = 7)
# (10)
text(0.1, -0.4, "col=8")
lines(c(0.3, 1), c(-0.4, -0.4), lwd = 2, col = 8)
}

(1) The graphics area is set.
(2) The coordinate axes are set but not labeled.
(3)(4)(5)(6)(7)(8)(9)(10) Straight lines are drawn with various values of col ranging from 1 to 8.

The setting of colors in lines() can be done with words. The relationships between numbers and words are as follows.

<table>
<thead>
<tr>
<th>Numbers</th>
<th>Words</th>
<th>Numbers</th>
<th>Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>col = 1</td>
<td>&quot;black&quot;</td>
<td>col = 5</td>
<td>&quot;skyblue&quot;</td>
</tr>
<tr>
<td>col = 2</td>
<td>&quot;red&quot;</td>
<td>col = 6</td>
<td>&quot;magenta&quot;</td>
</tr>
<tr>
<td>col = 3</td>
<td>&quot;green&quot;</td>
<td>col = 7</td>
<td>&quot;yellow&quot;</td>
</tr>
<tr>
<td>col = 4</td>
<td>&quot;blue&quot;</td>
<td>col = 8</td>
<td>&quot;gray&quot;</td>
</tr>
</tbody>
</table>

Grays of various densities can be used to color straight lines given by lines() as shown in Program (1 - 20), which outputs Fig. 1.35.

Program (1 - 20)
function() {
# (1)
par(mai = c(1, 1, 1, 1), omi = c(0, 0, 0, 0))
# (2)
plot(c(0, 1), c(0, 1), xlab = "", ylab = "", type = "n",
     axes = F)
# (3)
text(0.0, 1, "col=gray(1)", adj = c(0, 0.5))
```
col=gray(1)
col=gray(0.8)

col=gray(0.8)

col=gray(0.4)

col=gray(0.2)

col=gray(0.0)
```

Figure 1.35  Gray straight lines of various densities, drawn by Program (1 - 20)

```r
lines(c(0.4, 1), c(1, 1), lwd = 4, col = gray(1))
# (4)
text(0.0, 0.8, "col=gray(0.8)", adj = c(0, 0.5))
lines(c(0.4, 1), c(0.8, 0.8), lwd = 4, col = gray(0.8))
# (5)
text(0.0, 0.6, "col=gray(0.6)", adj = c(0, 0.5))
lines(c(0.4, 1), c(0.6, 0.6), lwd = 4, col = gray(0.6))
# (6)
text(0.0, 0.4, "col=gray(0.4)", adj = c(0, 0.5))
lines(c(0.4, 1), c(0.4, 0.4), lwd = 4, col = gray(0.4))
# (7)
text(0.0, 0.2, "col=gray(0.2)", adj = c(0, 0.5))
lines(c(0.4, 1), c(0.2, 0.2), lwd = 4, col = gray(0.2))
# (8)
text(0.0, 0.0, "col=gray(0.0)", adj = c(0, 0.5))
lines(c(0.4, 1), c(0.0, 0.0), lwd = 4, col = gray(0.0))
}
```

(1) The graphics area is set.
(2) The coordinate axes are set but not labeled.
(3)(4)(5)(6)(7)(8)(9)(10) Straight lines are drawn with various values of gray() ranging from 0 to 1. The color of a straight line is white if gray(1) is set. Hence, a straight line is not drawn if the background color is white.

points() not only draws small circles but also draws other symbols by specifying pch =. The R program below exemplifies this function and produces Fig. 1.36 (left).

Program (1 - 21)
```
function() {
```
Figure 1.36 Various symbols given by Program (1 - 21) (left). Line chart constructed by Program (1 - 22) (right).

```r
# (1)
par(mai = c(1, 1, 1, 1), omi = c(0, 0, 0, 0))

# (2)
plot(c(0, 1), c(0, 1.1), xlab = "", ylab = "", type = "n", axes = F)

# (3)
text(c(0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1), c(1.1, 1.0, 0.9, 0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1, 0), c("pch=0", "pch=1", "pch=2", "pch=3", "pch=4", "pch=5", "pch=6", "pch=7", "pch=8", "pch=9", "pch=10", "pch=11"), cex = 1)

# (4)
points(c(0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3), c(1.1, 1.0, 0.9, 0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1, 0), pch=c(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11), cex = 2)

# (5)
text(c(0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6), c(1.1, 1.0, 0.9, 0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1, 0), c("pch=12", "pch=13", "pch=14", "pch=15", "pch=16", "pch=17", "pch=18", "pch=19", "pch=20", "pch=21", "pch=22", "pch=23"), cex = 1)

# (6)
points(c(0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8, 0.8), c(1.1, 1.0, 0.9, 0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1, 0), pch=c(12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100), cex = 1)
```
19, 20, 21, 22, 23), cex = 2)
}

(1) The graphics area is set.
(2) The coordinate axes are set but not labeled.
(3)(4) Symbols given by various settings of pch = ranging from 0 to 11 are drawn alongside the settings.
(5)(6) Symbols given by various settings of pch = ranging from 12 to 23 are drawn alongside the settings.

A line chart such as Fig. 1.36 (right) can be illustrated. It is realized by combining the functions of lines(), which draws a straight line, and points(), which draws a symbol. The R program for this graph is Program (1 - 22).

Program (1 - 22)
function() {
  # (1)
  par(mai = c(1, 1, 1, 1), omi = c(0, 0, 0, 0))
  # (2)
  plot(c(0, 10), c(0, 1), xlab = "x", ylab = "y", type = "n")
  # (3)
  xx <- c(1, 3, 5, 7, 9)
  yy1 <- c(0.9, 0.8, 0.75, 0.72, 0.88)
  yy2 <- c(0.72, 0.66, 0.55, 0.59, 0.79)
  yy3 <- c(0.59, 0.2, 0.31, 0.29, 0.39)
  # (4)
  points(xx, yy1, pch = 16, cex = 1.5)
  lines(xx, yy1)
  # (5)
  points(xx, yy2, pch = letters[1:5], cex = 1.5)
  lines(xx, yy2)
  # (6)
  points(xx, yy3, pch = c("1", "2", "3", "4", "5"), cex = 1.5)
  lines(xx, yy3)
}

(1) The graphics area is set.
(2) The coordinate axes are set, drawn, and labeled.
(3) Values are stored in variables (objects) specifying the positions on a graph. xx stores values for positions on the x-axis. yy1, yy2, and yy3 store values for positions on the y-axis.
(4) Straight lines and small filled circles based on xx and yy1 are drawn.
(5) Straight lines and letters based on xx and yy2 are drawn. letters[1:5] functions in the same way as c("a", "b", "c", "d", "e"). The text specified by pch = must comprise one letter.
(6) Specific letters are set for pch = as an argument of points().
1.9 **FONTS**

Various fonts are available for displaying text in graphs. The type of font available depends on the setting of fonts in the OS installed on a PC. Additionally, the fonts shown on a display may be different from those in a digital file saved in postscript format. That is, the fonts shown on a display cannot be stored as they are in a digital file on some occasions.

\[
\begin{align*}
\text{font}=1, \text{abcdef} & \quad \text{font}=6, \text{abcdef} \\
\text{font}=2, \text{abcdef} & \quad \text{font}=7, \text{abcdef} \\
\text{font}=3, \text{abcdef} & \quad \text{font}=8, \text{abcdef} \\
\text{font}=4, \text{abcdef} & \quad \text{font}=9, \text{abcdef}
\end{align*}
\]

**Figure 1.37** Text in Helvetica plain, Helvetica bold, Helvetica italic, and Helvetica bold italic in descending order, given by Program (1 - 23) (left). Text in Times plain, Times bold, Times italic, and Times bold italic in descending order, given by Program (1 - 23) (right).

Fig. 1.37 (left) employs Helvetica fonts. Texts in Helvetica plain, Helvetica bold, Helvetica italic, and Helvetica bold italic in descending order, constructed by Program (1 - 23), are illustrated.

**Program (1 - 23)**

```r
function() {
  # (1)
  par(mai = c(1, 1, 1, 1), omi = c(0, 0, 0, 0))
  # (2)
  plot(c(0, 1), c(0, 1), xlab = "", ylab = "", type = "n",
       axes = F)
  # (3)
  text(0.5, 0.8, "font=1, abcdef", cex = 2, font = 1)
  # (4)
  text(0.5, 0.6, "font=2, abcdef", cex = 2, font = 2)
  # (5)
  text(0.5, 0.4, "font=3, abcdef", cex = 2, font = 3)
  # (6)
  text(0.5, 0.2, "font=4, abcdef", cex = 2, font = 4)
}
```

(1) The graphics area is set.
(2) The coordinate axes are set.
(3) The command `text()` draws "font=1, abcdef". The argument `font = 1` indicates Helvetica plain. However, the meaning of `font = 1` depends on the setting of fonts in the OS on a PC.
(4) "font=2, abcdef" is written. The argument font = 2 indicates Helvetica bold.
(5) "font=3, abcdef" is written. The argument font = 3 indicates Helvetica italic.
(6) "font=4, abcdef" is written. The argument font = 4 indicates Helvetica bold italic.

Fig. 1.37 (right), constructed by Program (1 - 24), employs the Times font.

Program (1 - 24)
function() {
    # (1)
    par(mai = c(1, 1, 1, 1), omi = c(0, 0, 0, 0))
    # (2)
    plot(c(0, 1), c(0, 1), xlab = "", ylab = "", type = "n",
         axes = F)
    # (3)
    text(0.5, 0.8, "font=6, abcdef", cex = 2, font = 6)
    # (4)
    text(0.5, 0.6, "font=7, abcdef", cex = 2, font = 7)
    # (5)
    text(0.5, 0.4, "font=8, abcdef", cex = 2, font = 8)
    # (6)
    text(0.5, 0.2, "font=9, abcdef", cex = 2, font = 9)
}

(1) The graphics area is set.
(2) The coordinate axes are set.
(3) "font=6, abcdef" is written. The argument font = 6 indicates Times plain. The meaning of font = 1 depends on the setting of the OS installed on a PC.
(4) "font=7, abcdef" is written. The argument font = 7 indicates Times bold.
(5) "font=8, abcdef" is written. The argument font = 8 indicates Times italic.
(6) "font=9, abcdef" is written. The argument font = 9 indicates Times bold italic.

1.10 FIGURES SUCH AS CIRCLES AND RECTANGLES

The R command symbols() draws circles, rectangles, stars, thermometers, and boxplots. Fig. 1.38 (left), constructed by Program (1 - 25), exemplifies figures given by symbols().

Program (1 - 25)
function() {

Figure 1.38 Various figures given by Program (1 - 25) (left). Rectangles given by Program (1 - 26) (right).

```r
# (1)
par(mai = c(1, 1, 1, 1), omi = c(0, 0, 0, 0))
# (2)
plot(c(0, 1), c(0, 4), xlab = "", ylab = "", type = "n")
# (3)
symbols(0.3, 3, add = T, inches = F, circles = 0.2,
  fg = "red", bg = "yellow", lwd = 5)
# (4)
symbols(0.75, 3, add = T, inches = F, squares = 0.3,
  fg = "green", bg = "skyblue", lwd = 5)
# (5)
symbols(0.3, 1.6, add = T, inches = F, rectangles =
  matrix(c(0.3, 0.5), nrow = 1), fg = "blue", bg = "magenta",
  lwd = 5)
# (6)
symbols(0.7, 1.6, add = T, inches = F, stars = matrix(
  c(0.2, 0.1, 0.13, 0.12, 0.1), nrow = 1), fg = "yellow",
  bg = "red", lwd = 5)
# (7)
th1 <- matrix(c(0.15, 0.1, 0.4, 0.5, 0.8, 0.3), ncol = 3)
symbols(c(0.15, 0.35), c(0.8, 0.3), add = T, inches = F,
  thermometers = th1, fg = "blue", bg = "yellow", lwd = 2)
# (8)
box1 <- matrix(c(0.25, 0.15, 0.4, 0.5, 0.1, 0.4, 0.1, 0.2,
  0.3, 0.8), ncol = 5)
symbols(c(0.65, 0.9), c(0.2, 0.6), add = T, inches = F,
  boxplots = box1, fg = "blue", bg = "yellow", lwd = 2)
```
(1) The graphics area is set.
(2) The coordinate axes are set.
(3) Circles are drawn using symbols(). add = T indicates adding a figure to an existing graph (the coordinate axes in this case). The setting of add = F or the default setting means drawing a new graph. The argument inches = F specifies the method of setting the size of a circle. The argument inches = F means that the radius of a circle is specified on the scale of the x-axis. The argument circles = 0.2 means the radius of a circle. The argument fg = "red" indicates that the circumference of a circle is drawn in red. The argument bg = "yellow" shows that the inner area of a circle is drawn in yellow.

(4) The command symbols() draws a square. The argument inches = F specifies the method of setting the size of a square. The argument inches = F means that the length of the side of a square is specified on the scale of x-axis. The argument squares = 0.3 sets the length of the side of a square.

(5) The command symbols() draws a rectangle. The argument inches = F specifies the method of setting the length of the sides of a rectangle. The argument inches = F means that the length of the sides of a rectangle are specified on the scales of the x-axis and y-axis. The argument rectangles = matrix(c(0.3, 0.5), nrow = 1) indicates that the horizontal width of the rectangle is 0.3 (on the scale of x-axis), and the vertical length is 0.5 (on the scale of the y-axis).

(6) The command symbols() draws a star. A star is given by specifying the position of the original point, dividing 360 degrees equally into a specific number of directions on the basis of the original point, drawing radial straight lines of specific lengths along these directions, and connecting the end points of the lines by straight lines. The argument inches = F specifies the method of setting the lengths of radial straight lines constituting a star. The first element indicates the length of the rightward straight line from the original point. Other elements specify the lengths of the straight lines in a counterclockwise direction. The argument inches = F indicates that the lengths of radial straight lines constituting a star are specified on the scale of the x-axis. Since stars = matrix(c(0.2, 0.1, 0.13, 0.12, 0.1), nrow = 1) gives a matrix with five elements, it indicates that the shape of a star with five apexes is illustrated; each element of the matrix specifies the length of the straight line in each direction.

(7) The command symbols() draws a thermometer. The argument inches = F specifies the method of setting the lengths of the sides of a thermometer and its shape. The argument inches = F indicates that the lengths of the sides of a thermometer are set on the scales of the x-axis and y-axis. The argument thermometers = th1 specifies the lengths of the sides of a thermometer and its shape. The elements of th1 in this example are as follows:

```
[,1] [,2] [,3]
```


[1,] 0.15 0.4 0.8  
[2,] 0.10 0.5 0.3

The element of 0.15 is the horizontal length of the left thermometer on the scale of the x-axis. The element of 0.4 is the vertical length of the left thermometer on the scale of the y-axis. The element of 0.8 shows the ratio of the lower part of the left thermometer. The element of 0.10 is the horizontal length of the right thermometer on the scale of the x-axis. The element of 0.5 is the vertical length of the right thermometer on the scale of the y-axis. The element of 0.3 indicates the ratio of the lower part of the right thermometer.

(8) The command symbols() draws a boxplot. The argument inches = F indicates that the lengths of the sides and whiskers of a boxplot is specified on the scales of the x-axis and y-axis. The argument boxplots = box1 specifies the lengths of the sides and whiskers of a boxplot. box1 in this example is as follows:

```
[1,] 0.25 0.4 0.05 0.7 0.3  
[2,] 0.15 0.5 0.45 0.2 0.8
```

The element of 0.25 is the horizontal width of the left boxplot on the scale of the x-axis. The element of 0.4 is the vertical width of the left boxplot on the scale of the y-axis. The element of 0.05 is the length of the lower whisker of the left boxplot. The element of 0.7 is the length of the upper whisker of the left boxplot. The element of 0.3 indicates the position of the middle line of the left boxplot: the ratio of the lower part of the box under the middle line. The element of 0.15 is the horizontal width of the right boxplot on the scale of the x-axis. The element of 0.5 is the vertical width of the right boxplot on the scale of the y-axis. The element of 0.45 is the length of the lower whisker of the right boxplot. The element of 0.2 is the length of the upper whisker of the right boxplot. The element of 0.8 indicates the position of the middle line of the right boxplot: the ratio of the lower part of the box under the middle line.

The command rect() allows drawing of various rectangles. For example, Program (1 - 26) gives Fig. 1.38 (right).

Program (1 - 26)
```
function(){
  # (1)
  par(mai = c(1, 1, 1, 1), omi = c(0, 0, 0, 0))
  # (2)
  plot(c(0, 1), c(0, 1), xlab = "", ylab = "", type = "n")
  # (3)
  rect(0.1, 0.7, 0.4, 0.9, border = "magenta", col = "yellow", lwd = 5)
  # (4)
```
rect(0.6, 0.65, 0.9, 0.9, border = "green", col = "blue",
   density = 10, ljoin = 1, lwd = 4)
# (5)
rect(0.1, 0.1, 0.35, 0.35, border = "red", col = "green",
   density = 5, angle = 120, ljoin = 1, lwd = 3)
# (6)
rect(0.6, 0.1, 0.95, 0.5, border = "blue", col = "skyblue",
   ljoin = 1, lwd = 2)
# (7)
rect(0.65, 0.35, 0.9, 0.45, border = "white", col = "white",
   ljoin = 1, lwd = 4)
# (8)
rect(0.65, 0.15, 0.9, 0.25, border = "transparent",
   col = "white", ljoin = 1, lwd = 4)

(1) The graphics area is set.
(2) The coordinate axes are drawn.
(3) The command rect() draws a rectangle. The argument border =
"magenta" indicates that the rectangle is bordered in magenta. Since ljoin =
is not specified, ljoin = 0 is assumed. If ljoin = 0 is set, the contact points
between two straight lines are round. The setting of ljoin = "round" gives
the same result.
(4) The command rect() draws a rectangle. If ljoin = 1 is set, the contact
points between two straight lines are acute. The setting of ljoin = "mitre"
gives the same result. Since density = 10 is set, the rectangular region is
striped with 10 diagonal lines per inch. Since angle = is not set, angle = 45
is assumed. Therefore, the inclination of the diagonal lines is 45 degrees
when the angle is measured in a counterclockwise direction from the rightward
straight line drawn from the original point.
(5) The command rect() draws a rectangle. Since density = 5 is set,
the rectangular region is striped with five diagonal lines per inch. Since angle = 120 is set, the inclination of the diagonal lines is 120 degrees when
the angle is measured in a counterclockwise direction from the straight line
drawn rightward from the original point.
(6)(7)(8) The command rect() draws a skyblue rectangle and two white
rectangles in it. The sizes of the two white rectangles are set to be the same.
However, since the upper rectangle is given by border = "white" and the
lower rectangle is given by border = "transparent", the upper rectangle is
slightly larger than the lower one.
polygon() draws polygons in a concise manner. For instance, Program
(1 - 27) draws Fig. 1.39.

Program (1 - 27)
function() {
# (1)
par(mai = c(1, 1, 1, 1), omi = c(0, 0, 0, 0))
# (2)
plot(c(0, 1), c(0, 1), xlab = "", ylab = "", type = "n")
# (3)
polygon(c(0.1, 0.4, 0.3, 0.1, 0.2), c(0.6, 0.5, 0.8, 0.7, 0.6), border = "magenta", col = "yellow", lwd = 5)
# (4)
polygon(c(0.6, 0.9, 0.5, 0.7, 0.9), c(0.9, 0.5, 0.8, 0.5, 0.8), border = "skyblue", col = "red", density = 8, lwd = 3)
# (5)
polygon(c(0.0, 0.5, 1, 1, 0.5, 0), c(0.1, 0.4, 0.3, 0, 0, 0), density = 0, col = "red", lwd = 2)
}

(1) The graphics area is set.
(2) The coordinate axes are drawn.
(3) The command polygon() draws a polygon. The given points are connected sequentially and a closed polygon is formed by connecting the first point and last point.
(4) The command polygon() draws a polygon. When lines connecting the given points cross, a figure the same as that in Fig. 1.39 is formed.
(5) The command polygon() draws a polygon. Since density = 0 is set, the inside of the rectangle is neither filled nor striped.
1.11 LEGENDS AND LOGARITHMIC PLOTS

A legend can be added in a graph. For example, the use of `legend()` in the following program yields Fig. 1.40 (left).

Program (1 - 28)

```r
function() {
  # (1)
  par(mai = c(1, 1, 1, 1), omi = c(0, 0, 0, 0))
  # (2)
  plot(c(-1.5, 3), c(2, 9), xlab = "x", ylab = "y", type = "n",
       main = "Legend")
  # (3)
  lines(c(-1, 0, 3), c(5, 3, 8), lwd = 2)
  points(c(-1, 0, 3), c(5, 3, 8), pch = 1, cex = 1.3)
  lines(c(-1, 0, 3), c(4, 2.5, 6), lwd = 2)
  points(c(-1, 0, 3), c(4, 2.5, 6), pch = 3, cex = 1.3)
  # (4)
  legend(-1, 8.5, legend = c("data-1", "data-2"), pch = c(1, 3),
         cex = 1.3)
}
```

(1) The graphics area is set.
(2) The coordinate axes are drawn. The argument `main = "Legend"` specifies the title of the graph.
(3) Straight lines, circles, and x-indications are drawn.
(4) A legend is put at \((-1, 8.5)\). The elements of `legend("data-1", "data-2")` give the text for respective lines. The elements of `pch = c(1, 3)` specify the symbols corresponding to respective text in the legend.

The logarithmic scale can be used as the scale of the axes in a graph. For example, Program (1 - 29) depicts Fig. 1.40 (right).

Program (1 - 29)
```r
function() {
  # (1)
  par(mai = c(1, 1, 1, 1), omi = c(0, 0, 0, 0))
  # (2)
  plot(c(1, 9), c(10, 1000), xlab = "x", ylab = "y", type = "n",
       log = "y")
  # (3)
  lines(c(2, 4, 6, 8), c(145, 500, 641, 192), lwd = 2)
  points(c(2, 4, 6, 8), c(145, 500, 641, 192), pch = 1,
         cex = 1.3)
  # (4)
  lines(c(2, 4, 6, 8), c(35, 223, 412, 21), lwd = 2)
  points(c(2, 4, 6, 8), c(35, 223, 412, 21), pch = 3, cex = 1.3)
}
```

(1) The graphics area is set.
(2) The coordinate axes are drawn. Since `log = "y"` is set in `plot()`, the scale of the y-axis is the logarithmic scale. The argument `log = "x"` makes the scale of the x-axis the logarithmic scale. The argument `log = "xy"` makes both scales the logarithmic scales.
(3) Straight lines and circles are drawn.
(4) Straight lines and x-indications are drawn.

### 1.12 BAR CHARTS

The command `barplot()` illustrates a bar plot. For example, Program (1 - 30) gives Fig. 1.41 (left).

Program (1 - 30)
```r
function() {
  # (1)
  par(mai = c(1, 1, 1, 1), omi = c(0, 0, 0, 0))
  # (2)
  yy <- c(225, 500, 641, 192)
  # (3)
  name1 <- c("data-a", "data-b", "data-c", "data-d")
  # (4)
  barplot(yy, ylab = "y", names.arg = name1)
```
Figure 1.41  Horizontal bar plot given by Program (1 - 30) (left). Horizontal bar plot given by Program (1 - 31) (right).

1. The graphics area is set.
2. Data for four bars are stored as yy.
3. The names of the four bars are set in name1.
4. The command barplot() illustrates a bar plot for representing the values of yy. The argument names.arg = name1 specifies the names of the four bars.

A bar plot illustrated by barplot() can be horizontal. For example, if (4) of Program (1 - 30) is replaced with the following command, Fig. 1.41 (left) is obtained.

# (4)
barplot(yy, xlab = "y", names.arg = name1, horiz = T, las = 2)

The command barplot() draws a bar plot for representing the values of yy. The argument horiz = T makes the graph horizontal. The argument las = 2 makes the names of the respective bars horizontal.

The bar plot given by barplot() can be a bar plot with partitioned bars. For example, Program (1 - 31) realizes Fig. 1.42 (left).
Program (1 - 31)
function() {
  # (1)
  par(mai = c(1, 1, 1, 1), omi = c(0, 0, 0, 0))
  # (2)
  yy1 <- c(145, 239, 640, 192)
  yy2 <- c(321, 233, 204, 218)
  # (3)
**Figure 1.42** Partitioned bar plot given by Program (1 - 31) (left). Grouped bar plot given by Program (1 - 32) (right).

\[
yy <- cbind(yy1, yy2)
\]

```r
# (4)
barplot(yy, xlab = "x", ylab = "y", names.arg = c("A-group", "B-group"))
```

(1) The graphics area is set.

(2) The values of the respective partitions of the first bar are given as the elements of \(yy1\). The values of the respective partitions of the second bar are given as the elements of \(yy2\).

(3) The matrix \(yy\) is constructed; the first column is \(yy1\) and the second column is \(yy2\). The elements of \(yy\) here are as follows:

\[
\begin{align*}
yy1 & \\
[1,] & 145 & 321 \\
[2,] & 239 & 233 \\
[3,] & 640 & 204 \\
[4,] & 192 & 218 \\
\end{align*}
\]

(4) The command `barplot()` illustrates a bar plot for representing the values of \(yy\). The argument `names.arg = c("A-group", "B-group")` specifies the names of respective bars.

If the data of a bar plot given by `barplot()` is divided into groups, a grouped bar plot is a good option. For example, by substituting (4) of Program (1 - 31) with the following command, Fig. 1.42 (right) is obtained.

Program (1 - 32)
# (4)  
\texttt{barplot(}yy, xlab = "x", ylab = "y", beside = T, names.arg =  \texttt{c(}"A\text{-group}", "B\text{-group}"\texttt{))}

The program \texttt{barplot()} illustrates a bar plot for depicting the values of \texttt{yy}. The argument \texttt{names.arg = c("A\text{-group}", "B\text{-group"}) sets the names of the respective bars. The argument \texttt{beside = T} specifies that the two bar plots are positioned horizontally.

\subsection*{1.13 PIE CHARTS}

\begin{figure}[h]
\centering
\includegraphics[width=0.8\textwidth]{pie_chart.png}
\caption{Pie chart given by Program (1 - 33) (left). Pie chart given by Program (1 - 34) (right).
}
\end{figure}

A pie chart is drawn by a \texttt{R} program. For example, Program (1 - 33) using \texttt{pie()} yields Fig. 1.43 (left).

Program (1 - 33)
\begin{verbatim}
function() {
# (1)
par(mai = c(1, 1, 1, 1), omi = c(0, 0, 0, 0))
# (2)
yy <- c(225, 500, 641, 192)
# (3)
nname1 <- c("data\text{-a}" , "data\text{-b}", "data\text{-c}" , "data\text{-d}" )
# (4)
pie(yy, labels = name1)
}
\end{verbatim}

(1) The graphics area is set.
(2) Data is stored as \texttt{yy}.
(3) The names of data is set as \texttt{name1}.
(4) The command \texttt{pie()} realizes a pie chart using \texttt{yy}.
The respective areas in Fig. 1.43 (right) are striped. This pie chart is realized by replacing (4) of Program (1 - 33) with the following command.

Program (1 - 34)

```r
# (4)
pie(yy, labels = name1, density = 10, angle = c(30, 60, 90, 120))
```

The argument `density =` in `pie()` sets the densities of the stripes. The argument `angle =` designates the angles of respective areas.

### 1.14 LAYOUT OF MULTIPLE GRAPHS

![Graph](image)

**Figure 1.44** Use of `par(mfrow = c(2, 1), mai = c(1, 1, 1, 1), omi = c(0, 0, 0, 0))` (Program (1 - 35)).

Plural graphs can be illustrated in a graphics window. For example, Program (1 - 35) constructs Fig. 1.44.

Program (1 - 35)

```r
function() {
    # (1)
    par(mfrow = c(2, 1), mai = c(1, 1, 1, 1),
        omi = c(0, 0, 0, 0))
    # (2)
```
Figure 1.45 Use of `par(mfrow = c(2, 1), mai = c(1.5, 1, 1, 1), omi = c(0, 0, 0, 0))` (left). Use of `par(mfrow = c(2, 1), mai = c(1.5, 2, 1, 1), omi = c(0, 0, 0, 0))` (right).

```r
plot(c(1, 3), c(2, 9), xlab = "x", ylab = "y", type = "n")
lines(c(1, 2, 3), c(2, 8, 9))
points(c(1, 2, 3), c(2, 8, 9), pch = 14)

# (3)
plot(c(1, 3), c(2, 9), xlab = "x", ylab = "y", type = "n")
lines(c(1, 2, 3), c(3, 8, 4))
points(c(1, 2, 3), c(3, 8, 4), pch = 15)
```

1. The argument `mfrow = c(2, 1)` in `par()` specifies that the graphics window is divided into two equal-spaced areas vertically for drawing a graph in each area. The graphics area for each area is set.
2. The first graph is drawn.
3. The second graph is drawn.

When graphs are drawn in divided areas, the implications of `mai =` and `omi =` in `par()` are clear. To understand the indication of the first element of `mai =`, (1) of Program (1 - 35) is replaced with the command below; Fig. 1.45 (left) is obtained.

Program (1 - 36)

```r
# (1)
par(mfrow = c(2, 1), mai = c(1.5, 1, 1, 1),
     omi = c(0, 0, 0, 0))
```
The argument \texttt{mai = c(1.5, 1, 1, 1)} is set in \texttt{par()}. Hence, when Fig. 1.44 is compared with Fig. 1.45 (left), the figure margins for describing the explanation of the x-axis are wider in the two graphs located vertically in Fig. 1.45 (left). Therefore, the figure areas are vertically narrower.

To understand the second element of \texttt{mai =, 1)} of Program (1 - 36) is replaced with the following command; Fig. 1.45 (right) is obtained.

\begin{verbatim}
Program (1 - 37)
# (1)
par(mfrow = c(2, 1), mai = c(1.5, 2, 1, 1),
     omi = c(0, 0, 0, 0))
\end{verbatim}

The graphics area is set. The argument \texttt{mai = c(1.5, 2, 1, 1)} is set in \texttt{par()}. Hence, when Fig. 1.45 (left) is compared with Fig. 1.45 (right), the figure margins for describing the explanation of the y-axis are wider in the two graphs located vertically in Fig. 1.45 (right). Therefore, the figure areas are horizontally narrower.

\begin{figure}
\centering
\includegraphics[width=0.4\textwidth]{figure1.png}
\caption{Use of \texttt{par(mfrow = c(2, 1), mai = c(1.5, 2, 0.5, 0.5),
omai = c(0.7, 0, 0, 0)) (left). Use of \texttt{omi = c(0.7, 0, 0, 0))
mai = c(1.5, 2, 0.5, 0.5), omi = c(0.7, 0.7, 0, 0)) (right).}
\end{figure}

Furthermore, if the setting of \texttt{omi = in par()} is altered, the implications of \texttt{omi = are clear. To understand the first element of \texttt{omi =, 1)} of Program (1 - 37) is replaced with the command below; Fig. 1.46 (left) is obtained.

\begin{verbatim}
Program (1 - 38)
# (1)
par(mfrow = c(2, 1), mai = c(1.5, 2, 0.5, 0.5),
     omi = c(0.7, 0, 0, 0))
\end{verbatim}

The argument \texttt{omi = c(0.7, 0, 0, 0)} is set in \texttt{par()}. Hence, the outer margin of the bottom area is wide in Fig. 1.46 (left). Therefore, the area for the two graphs is vertically narrower.
To understand the second element of omi = (1) of Program (1 - 38) is replaced with the command below; Fig. 1.46 (right) is obtained.

Program (1 - 30)

# (1)
par(mfrow = c(2, 1), mai = c(1.5, 2, 0.5, 0.5),
omi = c(0.7, 0.7, 0, 0))

The argument omi = c(0.7, 0.7, 0, 0) is set in par(). Hence, when Fig. 1.46 (left) is compared with Fig. 1.46 (right), the outer margin of the left area is wider in Fig. 1.46 (right). Therefore, the area for the two graphs is horizontally narrower.

**Figure 1.47** Three graphs given by Program (1 - 40) (left). Four graphs given by Program (1 - 41) (right).

After a graphics window is divided vertically, the bottom area can be divided horizontally. For example, Program (1 - 40) yields Fig. 1.47 (left).

Program (1 - 40)

function() {
# (1)
fun1 <- function() {
   plot(c(1, 3), c(2, 9), xlab = "x", ylab = "y", type = "n")
   lines(c(1, 2, 3), c(2, 8, 9))
   points(c(1, 2, 3), c(2, 8, 9), pch = 1)
}
# (2)
fun2 <- function() {
   plot(c(1, 3), c(2, 9), xlab = "x", ylab = "y", type = "n")
   lines(c(1, 2, 3), c(3, 8, 4))
}
points(c(1, 2, 3), c(3, 6, 4), pch = 5)
}

# (3)
fun3 <- function()
{
  plot(c(1, 3), c(2, 9), xlab = "x", ylab = "y", type = "n")
  lines(c(1, 2, 3), c(3, 3, 7))
  points(c(1, 2, 3), c(3, 3, 7), pch = 16)
}

# (4)
par(mfrow = c(1, 1), omi = c(0, 0, 0, 0))

# (5)
split.screen(figs = c(2, 1))
split.screen(figs = c(1, 2), screen = 2)

# (6)
screen(1)
par(mai = c(1.0, 1.0, 0.2, 0.2))
fun1()

# (7)
screen(3)
par(mai = c(1.0, 1.0, 0.2, 0.2))
fun2()

# (8)
screen(4)
par(mai = c(1.0, 1.0, 0.2, 0.2))
fun3()

(1) The function fun1() for drawing the first graph is defined.
(2) The function fun2() for drawing the second graph is defined.
(3) The function fun3() for drawing the third graph is defined.
(4) The graphics area is set. Although the argument omi = is set in par(),
mai = is not set. This is because mai = is specified for the three graphs.
(5) The command split.screen(figs = c(2, 1)) divides the graphics area
vertically. The command split.screen(figs = c(1, 2), screen = 2) di-
vides the second part of the dual-partitioned window horizontally. Although
figs = is set in split.screen(figs = c(2, 1)), screen = is not set. Be-
cause the graphics window is not separated at this stage, screen = 0 is set
automatically.
(6) The command screen(1) describes drawing the first graph. The argument
mai = specifies the figure margin in the first graph. Then, the first graph is
drawn.
(7) The command screen(3) describes drawing the second graph. It should
be noted that this is not screen(2). The argument mai = specifies the figure
margin in the second graph. Then, the second graph is drawn.
(8) The command `screen(4)` describes drawing the third graph. It should be noted that this is `screen(4)`, not `screen(3)`. The argument `mai =` specifies the figure margin in the third graph. Then, the third graph is drawn.

Four graphs can be placed on a grid. After the graphics window is divided into two equal-spaced areas vertically, each space is divided into two equal-spaced areas horizontally. For example, Program (1 - 41) realizes Fig. 1.47 (right).

Program (1 - 41)

```r
function() {
# (1)
  fun1 <- function(){
    plot(c(1, 3), c(2, 9), xlab = "x", ylab = "y", type = "n")
    lines(c(1, 2, 3), c(2, 8, 9))
    points(c(1, 2, 3), c(2, 8, 9), pch = 1)
  }
# (2)
  fun2 <- function(){
    plot(c(1, 3), c(2, 9), xlab = "x", ylab = "y", type = "n")
    lines(c(1, 2, 3), c(3, 8, 4))
    points(c(1, 2, 3), c(3, 8, 4), pch = 5)
  }
# (3)
  fun3 <- function(){
    plot(c(1, 3), c(2, 9), xlab = "x", ylab = "y", type = "n")
    lines(c(1, 2, 3), c(3, 3, 7))
    points(c(1, 2, 3), c(3, 3, 7), pch = 16)
  }
# (4)
  fun4 <- function(){
    plot(c(1, 3), c(2, 9), xlab = "x", ylab = "y", type = "n")
    lines(c(1, 2, 3), c(7, 3, 8))
    points(c(1, 2, 3), c(7, 3, 8), pch = 18)
  }
# (5)
  par(mfrow = c(2, 2), mai = c(1, 1, 0, 0),
    omi = c(0.5, 0.5, 0.5, 0.5))
# (6)
  fun1()
# (7)
  fun2()
# (8)
  fun3()
# (9)
  fun4()
```
(1) The function `fun1()` for drawing the first graph is defined.
(2) The function `fun2()` for drawing the second graph is defined.
(3) The function `fun3()` for drawing the third graph is defined.
(4) The function `fun4()` for drawing the fourth graph is defined.
(5) The graphics area is set. Since the argument `mfrow=c(2, 2)` is set in `par()`, the whole area of the graphics window is divided into two equal-spaced areas vertically and each space is divided into two equal-spaced areas horizontally.
(6) The first graph is drawn.
(7) The second graph is drawn.
(8) The third graph is drawn.
(9) The fourth graph is drawn.

![Graphs](image)

**Figure 1.48** Two graphs given by Program (1 - 42).

The graphics windows can be divided into unequal-spaced areas as well as into equal-spaced areas. Program (1 - 42), which yields Fig. 1.48, exemplifies it.

Program (1 - 42)

```r
function() {
  # (1)
  fun1 <- function()
  plot(c(1, 3), c(2, 9), xlab = "x", ylab = "y", type = "n")
  lines(c(1, 2, 3), c(2, 8, 9))
```
points(c(1, 2, 3), c(2, 8, 9), pch = 14)
}
# (2)
fun2 <- function(){
  plot(c(1, 3), c(2, 9), xlab = "x", ylab = "y", type = "n"
  lines(c(1, 2, 3), c(3, 8, 4))
  points(c(1, 2, 3), c(3, 8, 4), pch = 15)
}
# (3)
par(mfrow = c(1, 1), omi = c(0, 0, 0, 0))
# (4)
figs1 <- matrix(c(0.1, 0.9, 0.6, 1.0, 0.1, 0.9, 0.0, 0.7),
  nrow = 2, byrow = T)
print(figs1)
split.screen(figs = figs1, erase = T)
# (5)
screen(1)
par(mai = c(1, 1.2, 0.1, 0.1))
fun1()
# (6)
screen(2)
par(mai = c(1, 0.7, 0.1, 0.1))
fun2()

(1) The function fun1() for drawing the first graph is defined.
(2) The function fun2() for drawing the second graph is defined.
(3) The graphics area is set. The argument omi = is set in par(). However, mai = is not set.
(4) The matrix figs1 for specifying the ratios of the division areas is defined. The matrix figs1 in this example is:

  [1,]  0.1  0.9  0.6  1.0
  [2,]  0.1  0.9  0.0  0.7

The first row represents the area of the first graph. The second row represents the area of the second graph. The first two elements of each row stand for the range of the x-axis. The last two elements of each row stand for the range of the y-axis. Hence, two graphs positioned vertically are obtained. The whole ranges of both the x-axis and the y-axis are between 0 and 1.
(5) The first graph is drawn.
(6) The second graph is drawn.
Figure 1.49  Three graphs given by Program (1 - 43) (left). Two graphs given by Program (1 - 44) (right).

When the graphics area is divided into two areas horizontally and vertically to locate the four graphs on a grid, the ratio of the division can be flexible. Fig. 1.49 (left) given by Program (1 - 43) is obtained.

Program (1 - 43)

```r
function() {
# (1)
  fun1 <- function()
  plot(c(1, 3), c(2, 9), xlab = "x", ylab = "y", type = "n",
       main = "Area-1")
  lines(c(1, 2, 3), c(2, 8, 9))
  points(c(1, 2, 3), c(2, 8, 9), pch = 4)
}

# (2)
fun2 <- function()
  plot(c(1, 3), c(2, 9), xlab = "x", ylab = "y", type = "n",
       main = "Area-2")
  lines(c(1, 2, 3), c(3, 8, 4))
  points(c(1, 2, 3), c(3, 8, 4), pch = 15)

# (3)
fun3 <- function()
  plot(c(2, 5), c(-2, 4), xlab = "x", ylab = "y", type = "n",
       main = "Area-3")
  lines(c(2, 4, 5), c(3, -1, 4))
}
```
points(c(2, 4, 5), c(3, -1, 4), pch = 6)
}
# (4)
par(omi = c(0, 0, 0, 0))
# (5)
layout(matrix(c(0, 1, 3, 2), ncol = 2, byrow = T),
       widths = c(1.6, 1.4), heights = c(5.2, 4.1))
# (6)
par(mai = c(0.9, 0.7, 0.2, 0.1))
fun1()
# (4)
par(mai = c(0.9, 0.7, 0.2, 0.1))
fun2()
# (5)
par(mai = c(0.9, 0.7, 0.2, 0.1))
fun3()
}

(1) The function fun1() for drawing the first graph is obtained.
(2) The function fun2() for drawing the second graph is obtained.
(3) The function fun3() for drawing the third graph is obtained.
(4) The graphics area is set.
(5) The command layout() divides the area into areas on a grid. The matrix
(matrix(c(0, 1, 3, 2), ncol = 2, byrow = T) divides the area into areas
on the 2 x 2 grid. Nothing is drawn in the upper left area. The first graph
is drawn in the upper right area. The third graph is drawn in the bottom
left area. The second graph is drawn in the bottom right area. The argument
widths = c(1.6, 1.4) specifies 1.6:1.4 ratio horizontal division. The argument
heights = c(5.2, 4.1) sets a 5.2:4.1 ratio vertical division.
(6) After the first area (upper right) is set, the graph is drawn in this area.
(7) After the second area (bottom right) is set, the graph is drawn in this area.
(8) After the third area (bottom left) is set, the graph is drawn in this area.

When the graphics area is divided into two areas horizontally and vertically
to locate the four graphs on a grid, adjacent areas can be united. Fig. 1.49
(right) given by Program (1 - 44) is realized.

Program (1 - 44)
function() {
  # (1)
  fun1 <- function(){
    plot(c(1, 3), c(2, 9), xlab = "x", ylab = "y", type = "n",
         main = "Area-1")
    lines(c(1, 2, 3), c(2, 8, 9))
    points(c(1, 2, 3), c(2, 8, 9), pch = 4)
  }
}
# (2)  
fun2 <- function()
  plot(c(1, 3), c(2, 9), xlab = "x", ylab = "y", type = "n",  
      main = "Area-2")  
  lines(c(1, 2, 3), c(3, 8, 4))  
  points(c(1, 2, 3), c(3, 8, 4), pch = 15)
}

# (3)  
par(omi = c(0, 0, 0, 0))

# (4)  
layout(matrix(c(0, 1, 2, 2), ncol = 2, byrow = T),  
        widths = c(1.6, 1.4), heights = c(5.2, 4.1))

# (5)  
par(mai = c(0.9, 0.7, 0.2, 0.1))
fun1()

# (6)  
par(mai = c(0.9, 0.7, 0.2, 0.1))
fun2()

(1) The function fun1() for drawing the first graph is obtained.
(2) The function fun2() for drawing the second graph is obtained.
(3) The graphics area is set.
(4) The command layout() divides the area into areas on a grid. The matrix
    matrix(c(0, 1, 2, 2), ncol = 2, byrow = T) divides the area into 2×2
    areas. Nothing is drawn in the upper left area. The first area is drawn in the
    upper right area. The second graph is drawn in the area given by uniting the
    two areas in the bottom.
(5) After the first area (upper right) is set, the graph is drawn in the area.
(6) After the second area (bottom) is set, the graph is drawn in the area.

The use of par(new = T) is another option for positioning plural graphs
at intended positions in a graphics window. For example, Program (1 -
45) yields Fig. 1.50.

Program (1 - 45)

function()
  {  
# (1)  
  fun1 <- function()
    {  
      plot(c(1, 3), c(2, 9), xlab = "x", ylab = "y", type = "n")  
      lines(c(1, 2, 3), c(2, 8, 9))  
      points(c(1, 2, 3), c(2, 8, 9), pch = 14)
    }

# (2)  
  fun2 <- function()
    {  
      plot(c(1, 3), c(2, 9), xlab = "x", ylab = "y", type = "n")  
      lines(c(1, 2, 3), c(3, 8, 4))  
    }
points(c(1, 2, 3), c(3, 8, 4), pch = 15)
}

# (3)
par(mfrow = c(1, 1), omi = c(0, 0, 0, 0))

# (4)
par(fig = c(0.0, 0.6, 0.0, 0.4), mai = c(0.8, 0.8, 0.1, 0.1))

# (5)
fun1()

# (6)
par(new = T)

# (7)
par(fig = c(0.3, 1, 0.4, 1), mai = c(0.8, 0.8, 0.3, 0.3))

# (8)
fun2()

(1) The function fun1() for drawing the first graph is obtained.
(2) The function fun2() for drawing the second graph is obtained.
(3) The graphics area is set. The argument omi is set in par(). However, mai is not set.
(4) Specification of fig = c(0.0, 0.6, 0.0, 0.4) in par() sets the first area. The argument mai = c(0.8, 0.8, 0.1, 0.1) specifies the size of the figure margins.
(5) The command par(new = T) describes the superimposition of a new graph on the current graph.
(6) The first graph is drawn.
(7) The specification of `fig = c(0.3, 1, 0.4, 1)` in `par()` sets the area of
the second graph. The argument `mai = c(0.8, 0.8, 0.3, 0.3)` sets the
size of the figure margin in the graph.
(8) The second graph is drawn.

![Bar plot with an image](image.png)

**Figure 1.51** Bar plot with an image, given by Program (1 - 46).

Since `par(new = T)` enables the superimposition of graphs, a graph can
be superimposed on an image displayed by `image()` in black and white; the
image is obtained by a digital camera.

For example, Program (1 - 46) yields Fig. 1.51.

Program (1 - 46)

```r
function() {
  # (1)
  image1 <- read.csv(file = "d:\GraphicsR\image1.txt",
                     header=F)
  # (2)
  par(mai = c(1, 2, 1, 1), omi = c(0, 0, 0, 0))
  # (3)
  image1 <- matrix(unlist(image1), nrow = 323)
  # (4)
  image(image1, col = gray(seq(from = 0.1, to = 1,
                              length = 100)), axes = F)
  # (5)
  par(new = T)
  # (6)
  yy <- c(225, 500, 641, 192)
```
name1 <- c("data-a", "data-b", "data-c", "data-d")
x <- barplot(yy, ylab = "y", names.arg = name1,
ylim = c(0, 1000), density = 80, angle = c(30, 60, 90, 120))

(1) The brightness data file image1.txt is retrieved and named image1. For the origin of image1.txt, refer to the Appendix.
(2) The graphics area is set.
(3) image1 is converted to a matrix of 323 x 389 size. The command unlist() transforms the elements of the matrix into numerical form.
(4) image1 is shown as an image on the display. The argument col=gray(seq(from=0.1, to=1, length=100)) makes the image a black-and-white image represented by stepwise thickness. Since axes = F is set, no coordinate axes are drawn.
(5) The command par(new = T) describes the superimposition of a new graph on the current graph.
(6) A bar plot is illustrated.

![Graphs in R](image)

**Figure 1.52** Display of graphs in plural graphics window, given by Program (1 - 47).

Plural graphics window for respective graphs can be produced. The command dev.set(1) realizes it. Program (1 - 47) achieves Fig. 1.52.

**Program (1 - 47)**

```r
function() {
  # (1)
  par(mfrow = c(1, 1), mai = c(1, 1, 1, 1), omi = c(0, 0, 0, 0))
  # (2)
  plot(c(1, 3), c(2, 9), xlab = "x", ylab = "y", type = "n")
  lines(c(1, 2, 3), c(2, 8, 9))
  points(c(1, 2, 3), c(2, 8, 9), pch = 14)
  # (3)
```
dev.set(1)
#
# (4)
par(mfrow = c(1, 1), mai = c(1, 1, 1, 1), omi = c(0, 0, 0, 0))
plot(c(1, 3), c(2, 9), xlab = "x", ylab = "y", type = "n")
lines(c(1, 2, 3), c(3, 8, 4))
points(c(1, 2, 3), c(3, 8, 4), pch = 15)
#
(1) The graphics area is set.
(2) A graph is drawn.
(3) The command dev.set(1) produces a new graphics window.
(4) The graphics area is set.
(5) A graph is drawn.

1.15 SUMMARY

1. The "work image file" ".RData" starts up R.
   R commands and R programs are executed in a console window.

2. An editor is used for editing

3. The shutdown and rebooting of R reset the settings of graphics.

4. The command par() specifies the graphics settings or displays the content of the settings.

5. The command plot() sets the coordinate axes and/or draws the coordinate axes.

6. When the argument xlab = is specified in plot(), the explanation is written on the x-axis. When ylab = is specified, the explanation is written on the y-axis.

7. The first two arguments in plot() are the coordinates of the x-axis and y-axis, respectively.

8. If type = "n" is set in plot(), no data points are drawn.

9. The command lines() draws straight lines.

10. The command points() draws marks.

11. If axes = F is set in plot(), no coordinate axes are drawn.

12. Data are stored in variables (a variable is called an object in exact R terminology).

13. The command text() writes letters.
14. Mathematical expressions are written using expression().

15. The argument lwd = in lines() sets the thickness of straight lines.

16. The argument lty = set in lines() specifies the type of straight lines.

17. The argument col = set in lines() specifies the color of straight lines.

18. The value of gray() specified by col = in lines() changes the thickness of gray.

19. The value of pch = set in points() specifies the type of marks.

20. The argument font = set in text() gives the type of font.

21. The command symbols() draws circles, rectangles, stars, thermometers, and boxplots.

22. The command rect() draws rectangles.

23. The command polygon() draws polygons.

24. legend() adds a legend.

25. The argument log = "y" set in plot() makes the axis the logarithmic scale.

26. The command barplot() illustrates a bar plot.

27. The command pie() draws a pie chart.

28. The argument mfrow = set in par() divides a graphics windows into equal-spaced areas.

29. The arguments mai = and omi = set in par() specify the sizes and locations of graphs.

30. The command layout() divides the area into areas on a grid.

31. The argument fig = set in par() specifies the size and location of each graph.

32. The arguments widths = and heights = set in layout() specify the ratio of the division of a graphics window.

33. The command par(new = T) describes the superimposition of a new graphs on the current graph.

34. The command image() displays a black-and-white image.

35. The dev.set(1) produces a new graphics window.
36. The selection of "Copy as metafile" enables copying of a graph to a document of another sprogram.

37. The command `postscript()` constructs a digital file in postscript format.

38. The argument `family` in `postscript()` specifies the type of font.

39. The command `jpeg()` constructs a digital file in jpeg format.

40. The command `pdf()` constructs a digital file in pdf format.

EXERCISES

1.1 Replace part (1) of Program (1 - 2)(page 11) with the list below. The position and size of the graph will be altered.

```r
par(mai = c(2, 2, 1, 1), omi = c(0.5, 0.5, 0.5, 0.5))
```

1.2 Replace part (1) in Program (1 - 3)(page 13) with the list below. That is, `las = 1` is added to the arguments of `par()`. The values along the y-axis will be positioned horizontally.

```r
par(mai = c(1, 1, 1, 1), omi = c(0, 0, 0, 0), las = 1)
```

1.3 Replace part (2) of Program (1 - 4)(page 13) with the list below. The values along the axes will be altered.

```r
plot(c(-20, 30), c(25, 95), xlab = "x", ylab = "y")
```

1.4 In part (2) of Program (1 - 5)(page 14), add `main = "Graph-ABC"` to the arguments of `plot()`. That is, replace part (2) with the list below. The title of the graph will appear.

```r
plot(c(-2, 3), c(2, 9), xlab = "x", ylab = "y", type = "n",
     main = "Graph-ABC")
```

1.5 Replace parts (3) and (4) in Program (1 - 6)(page 14) with the list below. The positions of the straight lines and small circles will be altered.

```r
lines(c(-1, 0.5, 2.5), c(4, 7, 2.5))
p0ints(c(-1, 0.5, 2.5), c(4, 7, 2.5))
```

1.6 Replace part (2) in Program (1 - 7)(page 15) with the list below. The positions of the straight lines and data points will be altered.
xx <- c(-1, 0.5, 2.5)
yy <- c(4, 7, 2.5)

1.7 Replace part (3) in Program (1 - 14)(page 24) with the list below. The size and position of the text will be changed.

points(0.7, 0.8)
text(0.7, 0.8, "World Peace", cex = 1)

1.8 The part of (3) of Program (1 - 19)(page 29) is replaced with the list below. rgb() constructs colors. max = 255 indicates that the values of red =, green =, blue =, alpha = ranges from 0 to 255. alpha = specifies transparency. alpha = 255 means perfect opacity. However, some R commands gives opaque colors even if translucent colors are specified. Furthermore, even if translucent colors are shown in a display, they may be transformed into perfect opaque colors in digital files.

# (3),
rgb1 <- rgb(red = 200, green = 50, blue = 100, alpha = 255,
max = 255)
text(0, 1, 'col="rgb1"')
lines(c(0.3, 1), c(1, 1), lwd = 2, col = rgb1)

1.9 To make the scale of the x-axis the logarithmic scale, (2) of Program (1 - 29)(page 42) is replaced with the following R program.

plot(c(1, 9), c(10, 1000), xlab = "x", ylab = "y", type = "n",
log = "x")

1.10 To make the bars colorful, replace (4) of Program (1 -30)(page 42) with the following command.

barplot(yy, ylab = "y", names.arg = name1, col = rainbow(4))

1.11 To add a legend, replace (4) of Program (1 -31)(page 43) with the command below.

barplot(yy, xlab = "x", ylab = "y",
names.arg = c("A-group", "B-group"),
legend = c("q1", "q2", "q3", "q4"))

1.12 To make the pie chart colorful, replace (4) of Program (1 -33)(page 45) with the following command.

pie(yy, labels = name1, col = topo.colors(4))
1.13 Set `col` as one of the following specifications; `col` is the argument in `image()` in (4) of Program (1-46) (page 58).

```r
col = rainbow(50)
col = rainbow(30, start = 0.7, end = 0.1)
col = rainbow(2, start = 0.3, end = 0.2)
col = topo.colors(10)
col = terrain.colors(20)
col = heat.colors(3)
col = cm.colors(10)
```