Graphical Layouts

The `par` Function

R graphics are controlled by use of the `par` function.

`par` makes it possible to control low-level graphics by querying and setting a large set of *graphical parameters*. Graphical parameters control many features such as:

- the layout of figures on the device
- the size of the margins around plots
- the colours, sizes and typefaces of text
- the colour and texture of lines
- the style of axis to be used
- the orientation of axis labels

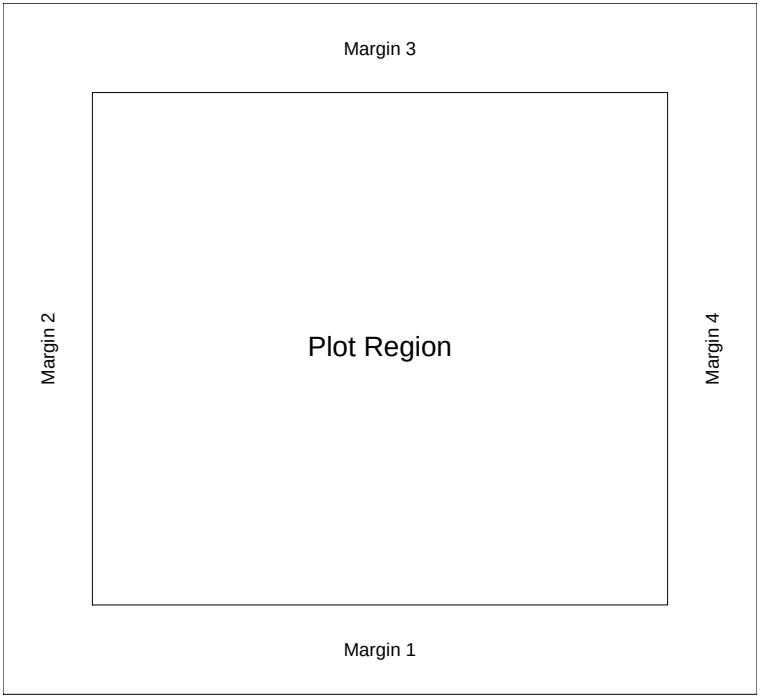
The Layout of Graphics

Graphs appear on R graphics devices as a series of rectangular graphical *figures*.

Each figure consists of a rectangular *plot region* surrounded by four *margins*.

Each element of a figure can be described as being in either the plot region or the margins.

The sides of a plot are numbered 1 through 4.


Margin 3
Margin 2
Plot Region
Margin 4
Margin 1

Controlling The Margins

The sizes of the margins can be changed by making a call to the function `par` before calling `plot.new`.

There are several possibilities:

1. Set the margin sizes in inches.

```
> par(mai=c(2, 2, 1, 1))
```

2. Set the margin sizes in lines of text.

```
> par(mar=c(4, 4, 2, 2))
```

3. Set the plot width and height in inches.

```
> par(pin=c(5, 4))
```

Querying the Margin Sizes

The par function can be used for querying the margin sizes.

The margin sizes in lines and inches can be obtained as follows:

```
> par("mar")
[1] 5.1 4.1 4.1 2.1

> par("mai")
[1] 1.02 0.82 0.82 0.42
```

and the the plot dimensions in inches as follows:

```
> par("pin")
[1] 9.952913 5.927717
```

Multifigure Layouts

`par` can be used to set up arrays of figures on the page. These arrays are then filled row-by-row or column-by-column.

The following example declares a 2×2 array to be filled by rows and then produces the plots.

```
> par(mfrow=c(2, 2))
> plot(rnorm(10), type = "p")
> plot(rnorm(10), type = "l")
> plot(rnorm(10), type = "b")
> plot(rnorm(10), type = "o")
```

A 2×2 array to be filled by columns would be declared with

```
> par(mfcol = c(2, 2))
```

rnorm(10)
Index
rnorm(10)
Index
rnorm(10)
Index
rnorm(10)
Index

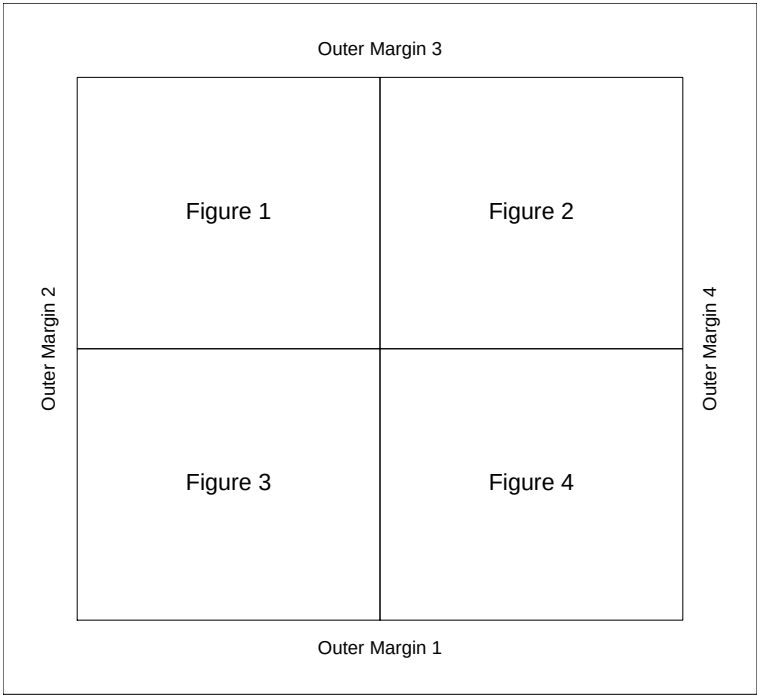
Outer Margins

Arrays of figures can also be created with a set of outer margins.

The size of outer margins can be specified with the `oma` or `omi` arguments to `par`.

Here is an example which shows a 2×2 array of figures surrounded by an outer margin with space for 4 lines of text on all sides.

```
> par(oma = c(4, 4, 4, 4),
      mfrow = c(2, 2))
```

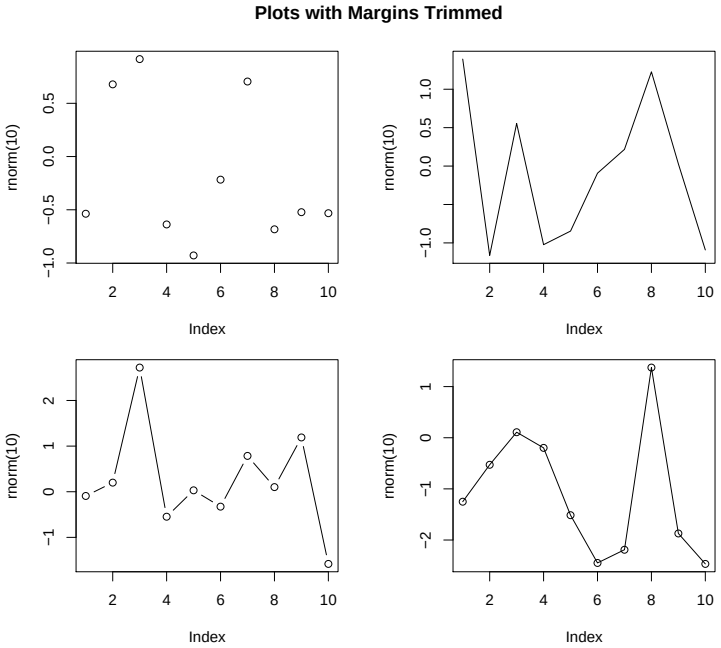

Outer Margin 3
Figure 1
Figure 2
Outer Margin 2
Outer Margin 4
Figure 3
Figure 4
Outer Margin 1

Eliminating Waste Margin Space

It can be useful to eliminate redundant space from multi-way arrays by trimming the margins a little.

```
> par(mfrow=c(2, 2))
> par(mar = c(5.1, 4.1, 0.1, 2.1))
> par(oma = c(0, 0, 4, 0))
> plot(rnorm(10), type = "p")
> plot(rnorm(10), type = "l")
> plot(rnorm(10), type = "b")
> plot(rnorm(10), type = "o")
> title(main = "Plots with Margins Trimmed",
        outer = TRUE)
```

Here we have trimmed space from the top of each plot, and placed a 4 line *outer margin* at the top of of the layout.


Plots with Margins Trimmed
rnorm(10)
Index
rnorm(10)
Index
rnorm(10)
Index
rnorm(10)
Index

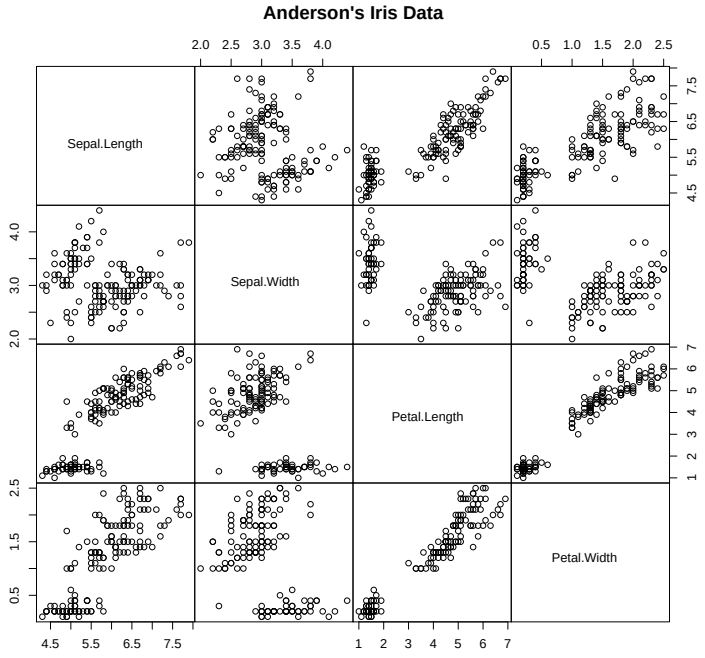
An Example: Scatterplot Matrices

A scatterplot matrix is a multidimensional generalisation of a simple scatterplot.

Given n variables, a scatterplot matrix presents scatterplots of every possible variable in the set against every other possible variable in the set.

The results are displayed in an $n \times n$ matrix, with the element in the ij-th position being the plot of variable i against variable j.

Since a plot of variable i against variable i is relatively uninteresting, something more useful can be displayed in that position.


Anderson's Iris Data
Sepal.Length
Sepal.Width
Petal.Length
Petal.Width
2.0 2.5 3.0 3.5 4.0
0.5 1.0 1.5 2.0 2.5
4.5 5.5 6.5 7.5
1 2 3 4 5 6 7

Setting Up a Scatterplot Matrix

To create a scatterplot matrix we need to arrange a series of plots on the page.

The basic arragement id to have the n^2 plots arranged in an $n \times n$ array.

To ensure that the plots are properly juxtaposed, we need to eliminate the margins around the individual plots but to create an outer margin around the whole array to hold the axes and an overall title.

```
par(mfrow = c(ncol(x), ncol(x)),
    mar = rep(0, 4),
    oma = c(4.1, 4.1, 5.1, 4.1))
```

Saving and Restoring Parameter Values

Since the layout parameters for the scatterplot matrix only apply to that matrix, it is useful to be able to revert back to the previous set of values when the plot is complete.

The `par` function returns the previous values of any parameters which are changed by a a call to it and these can be save for later use.

```
opar = par(mfrow = c(ncol(x), ncol(x)),
           mar = rep(0, 4),
           oma = rep(4.1, 4))
```

The values can be restored later with another call to `par`.

```
par(opar)
```

Drawing The Scatterplots

Once the layout has been created it is easy to produce the individual scatterplots.

```
for(i in 1:n)
    for(j in 1:n) {
        xrange = range(x[,j])
        yrange = range(x[,i])
        plot.new()
        plot.window(xlim = xrange, ylim = yrange)
        if (i == j)
            text(mean(xrange), mean(yrange),
                 colnames(x)[j])
         else points(x[,j], x[,i])
         box()
    }
```

Drawing Axes

Clearly, not every scatterplot in the matrix has axes and we need a computation to determine which (if any) axes should be drawn.

Only axes at the edge of the plot are drawn, and then only alternate axes are drawn to avoid label collisions.

```
if (i == 1 && j %% 2 == 0)
    axis(3)
if (i == n && j %% 2 == 1)
    axis(1)
if (j == 1 && i %% 2 == 0)
    axis(2)
if (j == n && i %% 2 == 1)
    axis(4)
```

Printing a Title

If we want to have an overall title on the plot we need to ensure that there is enough space in the top outer margin.

```
opar = par(mfrow = c(ncol(x), ncol(x)),
           mar = rep(0, 4),
           oma = c(4.1, 4.1, 5.1, 4.1))
```

We can then produce the title using the `mtext` function.

```
mtext(main, side = 3, line = 3,
      font = 2, outer = TRUE)
```

This draws the text in the (middle of the) third line of the third outer margin, in bold font.

Final Cleanup

For general hygiene it is important to reset any parameters before leaving the function.

```
par(opar)
```

Customisation

The scatterplot matrix function we've described is very inflexible.

We could customise it by adding of optional arguments.

```
pairs = function(x, col = 1, pch = 1, ...)
```

and then passing these arguments on to the `points` function.

```
points(x[,j], x[,i], col = col, pch = pch)
```

Panel Functions

The heart of the scatterplot matrix function is the call

```
points(x[,j], x[,i])
```

We can make the function highly customisable by making the function which is invoked at this point be an argument to the function.

```
pairs = function(x, y, ...,
                 panel = function(x, y, ...)
                         pairs(x, y, ...))
```

and invoking this function in the body of the function.

Panel Function Example

Here is an example which shows a panel function which plots the points and then superimposes the regression line between the pair of variables.

```
pairs(x,
      panel = function(x, y, ...) {
                  points(x, y)
                  abline(lm(y ~ x))
              })
```

Using `par`

The `par` function can be used to control margins and the layout of plots on the page, using the parameters `mar`, `mai`, `pin`, `oma`, `omi`, `mfrow` and `mfcol`.

This is just the tip of the iceberg; there are 71 graphics parameters in all.

You can view the settings of the parameters by typing

```
> par()
```

and view their names by typing

```
> names(par())
```

The Full Set of Graphical Parameters

xlog	ylog	adj	ann	ask
bg	bty	cex	cex.axis	cex.lab
cex.main	cex.sub	cin	col	col.axis
col.lab	col.main	col.sub	cra	crt
csi	cxy	din	err	family
fg	fig	fin	font	font.axis
font.lab	font.main	font.sub	gamma	lab
las	lend	lheight	ljoin	lmitre
lty	lwd	mai	mar	mex
mfcol	mfg	mfrow	mgp	mkh
new	oma	omd	omi	pch
pin	plt	ps	pty	smo
srt	tck	tcl	usr	xaxp
xaxs	xaxt	xpd	yaxp	yaxs
yaxt				

Important Groups of Graphics Parameters

Graphics parameters come in groups (usually with related names). Important groups correspond to names as follows:

`cex` An abbreviation for *character expansion factor*. This provides a magnification of text relative to the default.

`col` The colour to be used for text or graphics elements.

`font` The font to be used for drawing character strings.

Controlling Font Size

Setting the value of `cex` controls the default size of text elements and symbols in graphs. A value of `cex=2` makes text twice its standard size.

More specialized `cex` values are as follows:

`cex.main`	The expansion factor for the main title.
`cex.lab`	The expansion factor for `xlab` and `ylab`.
`cex.axis`	The expansion used for the text printed at axis tickmarks.
`cex.sub`	The expansion used for plot subtitles (`sub`).

Controlling Colour

Setting the value of `col` controls the default colour of text elements and symbols in graphs.

More specialized `col` values are as follows:

`col.main`	The colour for the main title.
`col.lab`	The colour for `xlab` and `ylab`.
`col.axis`	The colour used for the text printed at axis tickmarks.
`col.sub`	The colour used for plot subtitles (`sub`).

Controlling Fonts

Setting the value of `font` controls the default font of the text elements in graphs.

More specialized `font` values are as follows:

`font.main`	The font for the main title.
`font.lab`	The font for `xlab` and `ylab`.
`font.axis`	The font used for the text printed at axis tickmarks.
`font.sub`	The font used for plot subtitles (`sub`).

The value of `font` is `1` for normal font, `2` for bold, `3` for italic and `4` for bold-italic.

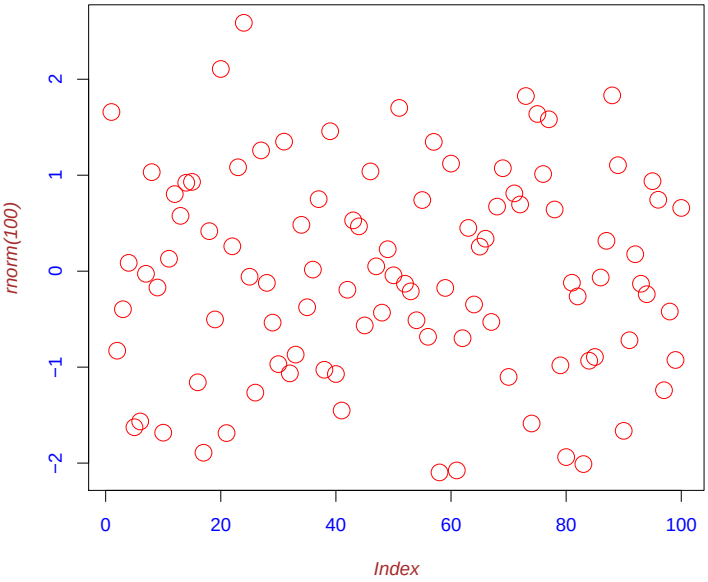
An Example

Many `par` values can be passed as arguments to plotting functions or passed to `par` to set the values for all future plots.

```
> plot(rnorm(100), col = "red", cex = 2,
       font.axis = 1, col.axis = "blue",
       font.main = 4, col.main = "green4",
       font.lab = 3, col.lab = "brown",
       cex.main = 2, main = "Par Overload")
```

```
> par(col = "red", cex = 2,
      font.axis = 1, col.axis = "blue",
      font.main = 4, col.main = "green4",
      font.lab = 3, col.lab = "brown",
      cex.main = 2)
```

Par Overload



All Those Other pars . . .

There are many other `par` values which we have not discussed.

You get a description of all the `par` values by looking at the documentation for `par`.

```
> ?par
```

```
> help("par")
```

More Flexible Layouts

The multifigure layouts which can be specified with `par` are very rigid.

All the panels in the plot have exacly the same size.

The `layout` function provides an alternative way of producing multiple figures.

The function still imposes constraints, but they only require that the widths of figures in the same column be equal and that the heights of figures in the same row be equal.

1	2	3
4	5	6
7	8	9

Combining Figures

The arrangements produced by layout are a little more flexible because adjacent figures can be combined.

In the next arrangement, the 4th and 5th figures of the previous layout are combined into a single figure.

Beware, however, that attempts to combine non-adjacent figures produces very strange results.

1	2	3
4		5
6	7	8

Arguments to Layout

The three main arguments to `layout`; `mat`, `widths` and `heights`.

The `mat` argument associates a figure number with each element of a rectangular layout.

The first example layout had the following `mat` argument.

```
> layout(matrix(1:9, nc = 3, byrow = TRUE),
                  ... )
```

The second layout combined the 4-th and 5-th figures from this layout as follows

```
> layout(matrix(c(1:4,4:8), nc = 3, byrow = TRUE),
                  ... )
```

Arguments to Layout

The widths of the columns in the arrangement are specied by the `widths` argument.

These can either be relative widths - e.g. `c(1, 2)` specifies that the second column is twice as big as the first, or they can be absolute widths specified in centimetres.

The specification

```
widths = c(lcm(1), 1, 2)
```

says that the first column is exactly 1cm wide and the third column is twice the width of the second.

The row heights are specified by heights in a similar fashion.

Layouts and Margins

It is important to note that `layout` ignores the outer margin parameters set with `par`.

All `par` values which work within figures (such as `mar` and `mai`) still have an effect.

It can be useful to set figure margins to 0 when working with sets of figures which are to be juxtaposed.

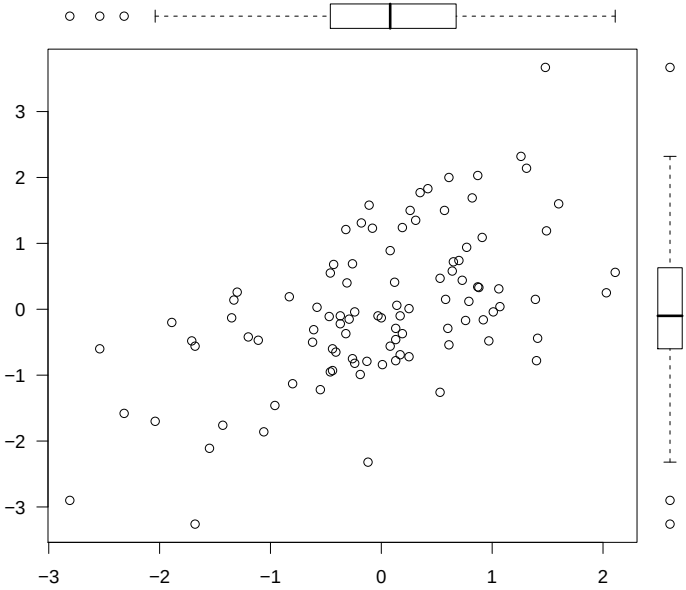
The effect of outer margins can be obtained by specifing extra rows and columns in the `mat` argument.

These extra rows and columns can be associated with "figure 0" so that no plotting is done in them.

Example: An Augmented Scatterplot

We'll illustrate what layouts can be used for with a specific example.

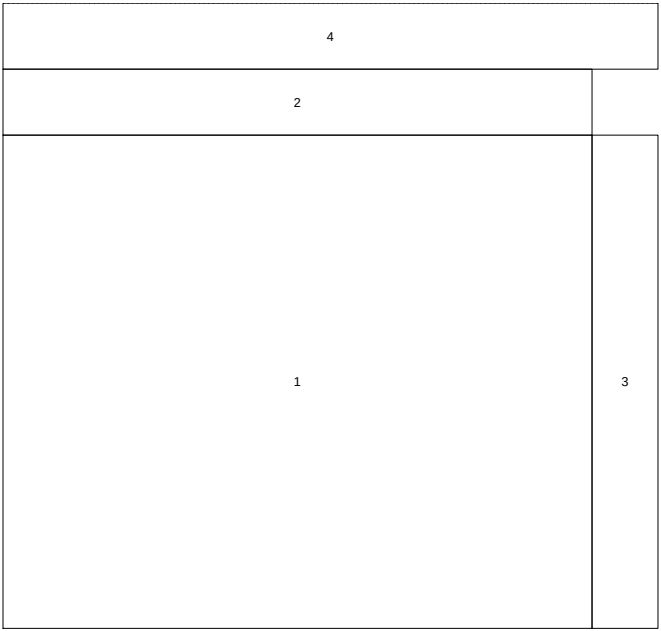
The example will produce a standard scatterplot which is enhanced with boxplots of the marginal distributions of the x and y variables.


An Enhanced Scatterplot
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2

Choosing a Layout

We need a big figure for the scatterplot, regions along its top and right for the scatterplots and a figure at the very top for a title.

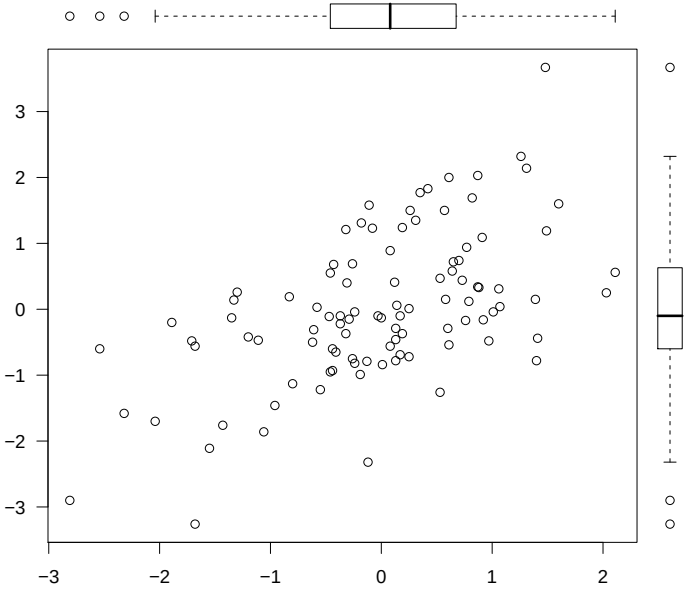
```
> layout(rbind(c(0,4,4,0),
               c(0,2,0,0),
               c(0,1,3,0),
               c(0,0,0,0)),
         height = c(lcm(2), lcm(2), 1, lcm(2)),
         width = c(lcm(2), 1, lcm(2), lcm(1)))
> layout.show(4)
> box("outer", lty = "dotted")
```


4
2
1
3

Producing the Graph

We constrain the margins of the plots to be zero so that the plots are juxtaposed. Then we produce the plots, in the order specified in `mat`. The last plot is a simple one which contains just the title.

```
> par(mar = rep(0, 4), cex = 1)
> plot(x, y, las = 1)
> boxplot(x, horizontal = TRUE, axes = FALSE)
> boxplot(y, axes = FALSE)
> plot.new()
> plot.window(xlim = c(0, 1), ylim = c(0, 1))
> text(.5, .25, "An Enhanced Scatterplot",
       cex = 1.5, font = 2)
```


An Enhanced Scatterplot
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2

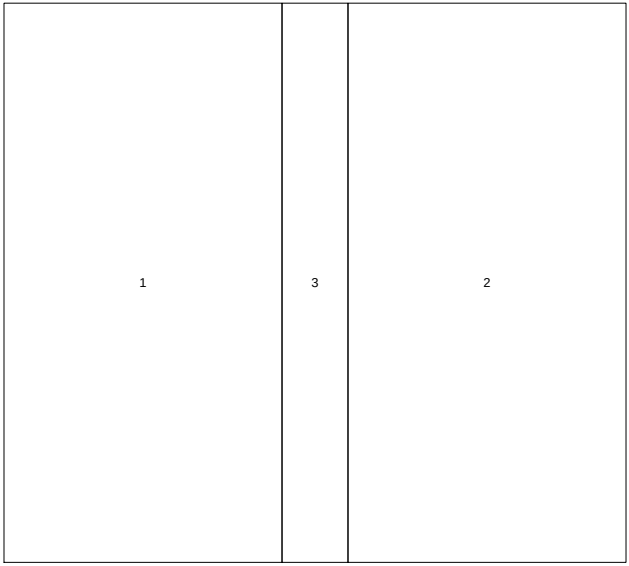
A Two-Panel Plot with Labelling

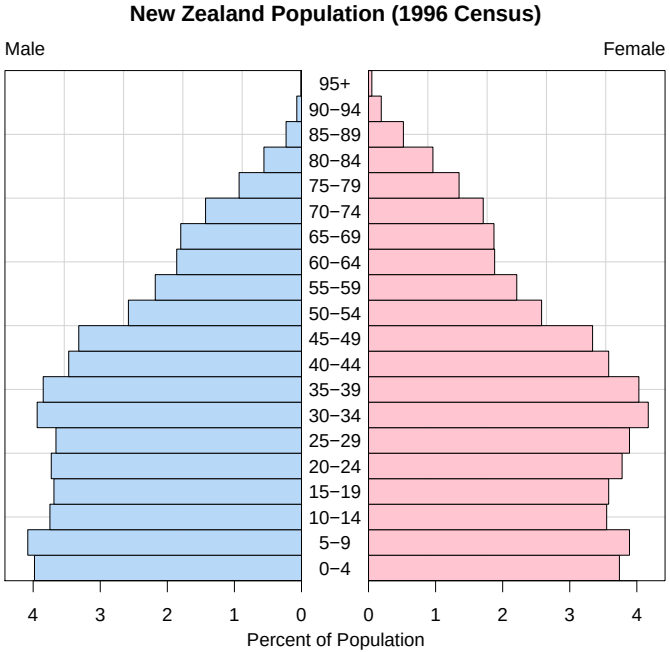
Suppose that (for some reason) we want a plot consisting of two panels with a strip for labelling between them.

How do we arrange this?

```
> layout(matrix(c(0,0,0,0,0,
                  0,1,3,2,0,
                  0,0,0,0,0), nc = 5, byrow = TRUE),
         widths = c(lcm(2), 1, lcm(2), 1, lcm(2)),
         heights = c(lcm(2), 1, lcm(2)))
> layout.show(3)
> box("outer", lty = "dotted")
```

Note that this just a start. You need extra panels for the labelling at the top and bottom, and some space for the axes to appear in.


1
3
2


New Zealand Population (1996 Census)
Male
Female
95+
90−94
85−89
80−84
75−79
70−74
65−69
60−64
55−59
50−54
45−49
40−44
35−39
30−34
25−29
20−24
15−19
10−14
5−9
0−4
4
3
2
1
0
0
1
2
3
4
Percent of Population

Layouts and Plots

We've seen that layouts provide a way to partition the page into a number of regions which can be used to draw multiple figures or a single plot with multiple components.

One simple approach is to use a high-level R plotting function such as `plot`, `hist`, `barplot` or `boxplot`, and to let the high level function take care of all the details.

An alternative, much more flexible, approach is to divide the page into subregions which each forms a single component of a graph.

It is useful to create a software component to match each of the types of region which will appear in the graph.

Handling Margins

Margins are added to plots to provide space for axes and annotations such as axis labels and titles.

Under the component model, such elements are represented by separate plotting areas, so the need for margins is gone.

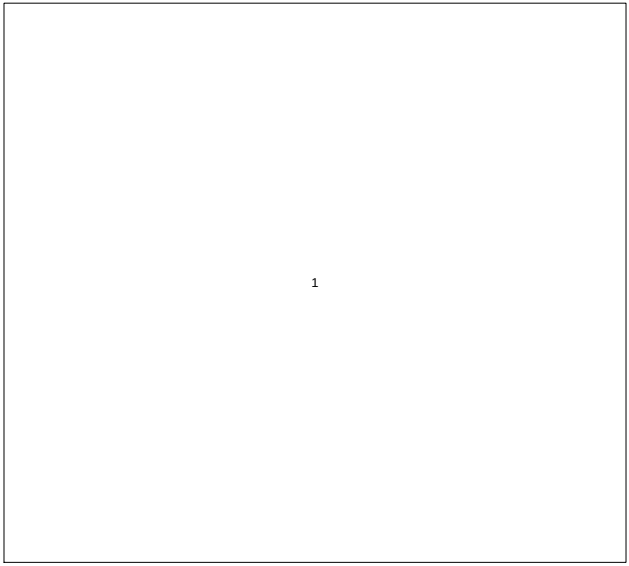
When working with layouts in this way it can be very useful to remove the margins completely with a specification of the form

```
> opar = par(mar = rep(0, 4))
```

Example: A Simple Scatterplot

We can build a layout for a simple scatterplot as follows:

```
> layout(matrix(c(0, 0, 0,
                  0, 1, 0,
                  0, 0, 0), nc = 3, byrow = TRUE),
         widths = c(lcm(2), 1, lcm(2)),
         heights = c(lcm(2), 1, lcm(2)))
> layout.show(1)
```


1

Choice of Margin Size

While it is possible to specify the margin size in centimetres, it is much more natural to specify it in lines of text (just like the `mar` and `oma` parameters).

By defining the function

```
> lines =
      function(x)
      lcm(x * par("csi") * 2.54)
```

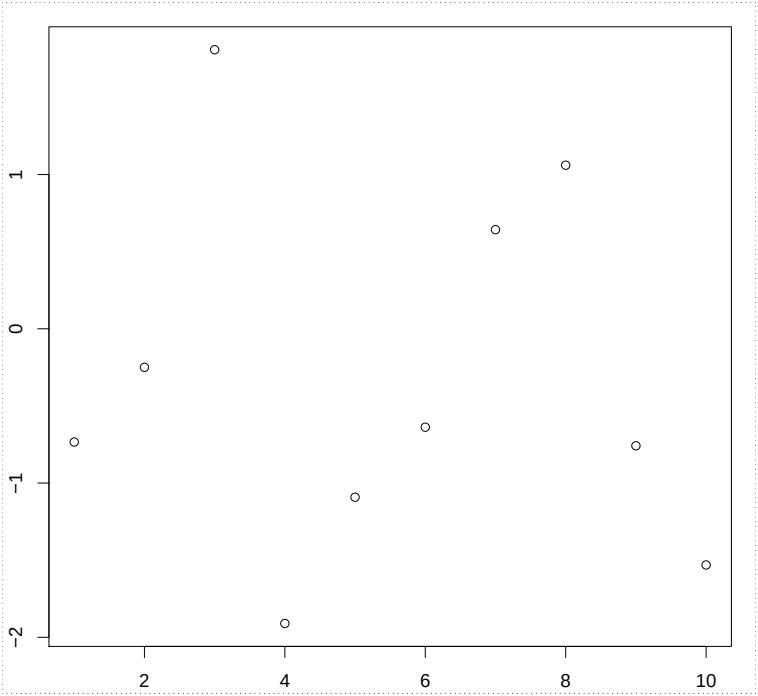
it is possible to specify margin size in lines of text.

A margin of 2.1 lines is just enough to fit the standard axis.

Example: A Simple Scatterplot

We can build a layout for a simple scatterplot, using just the minimum of space needed for the axes as follows:

```
> par(mar = rep(0, 4))
> layout(matrix(c(0, 0, 0,
                  0, 1, 0,
                  0, 0, 0), nc = 3, byrow = TRUE),
         widths = c(lines(2.1), 1, lines(1.1)),
         heights = c(lines(1.1), 1, lines(2.1)))
> par(cex = 1)
> plot(rnorm(10), ann = FALSE)
```


1
0
−1
−2
2
4
6
8
10

Axis Labels

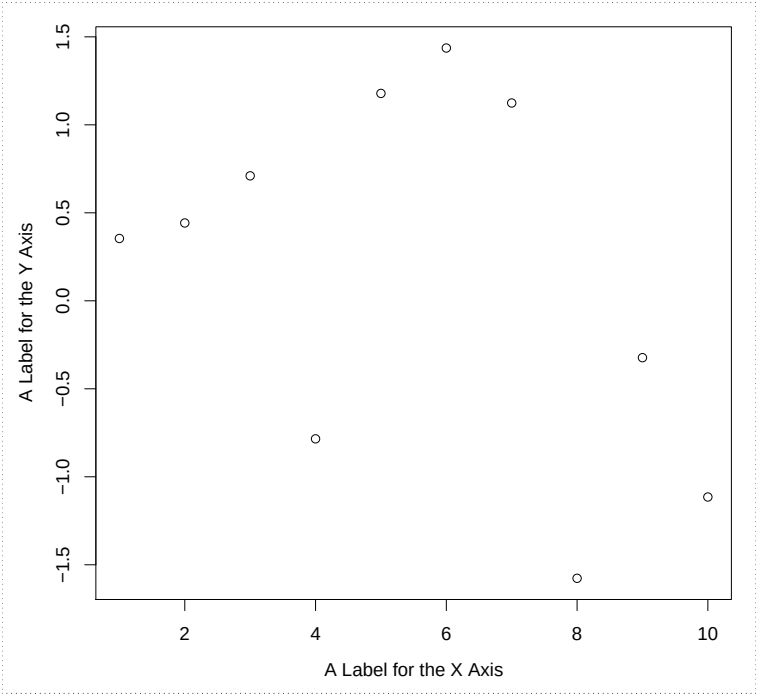
Now suppose that we want to add axis labels to the scatterplot. This means adding another line at the top and left of the layout.

```
> par(mar = rep(0, 4))
> layout(matrix(c(0, 0, 0, 0,
                  3, 0, 1, 0,
                  0, 0, 0, 0,
                  0, 0, 2, 0),
                nc = 4, byrow = TRUE),
         widths = c(lines(2.1), lines(2.1),
                    1, lines(1.1)),
         heights = c(lines(1.1), 1,
                     lines(2.1), lines(2.1)))
> par(cex = 1)
```

Producing the Labels

The plot labels are produced by moving to the region for the label (with `plot.new`) and drawing the text at the centre of the region.

```
> plot.new()
> text(0.5, 0.5, "An X Label")
> plot.new()
> text(0.5, 0.5, "A Y Label", srt = 90)
```

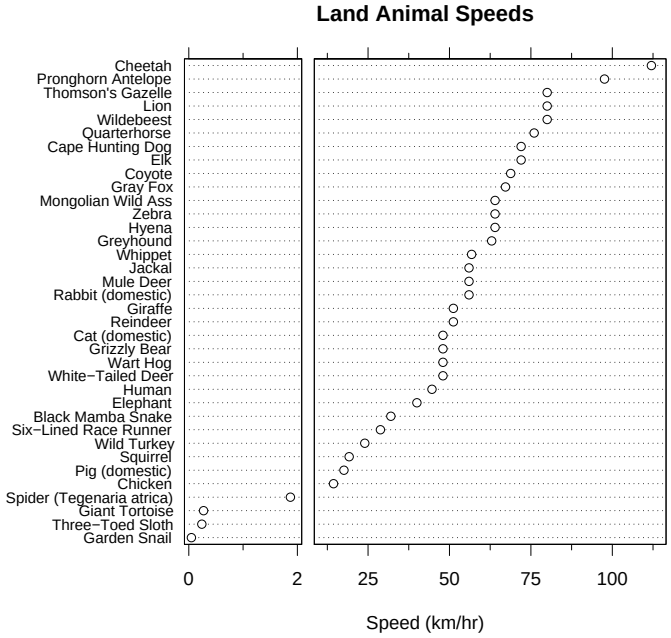

A Label for the Y Axis
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
2
4
6
8
10
A Label for the X Axis

Other Features

It is possible to add other features to the plot such as additional axes at the top and right or an overall title by adding further plotting regions.

With a little willingness to experiment, there is virtually no limit to the kinds of plot which you can produce in this way.

The following plot shows a dotchart comparison of land animal speeds produced in exactly this way.

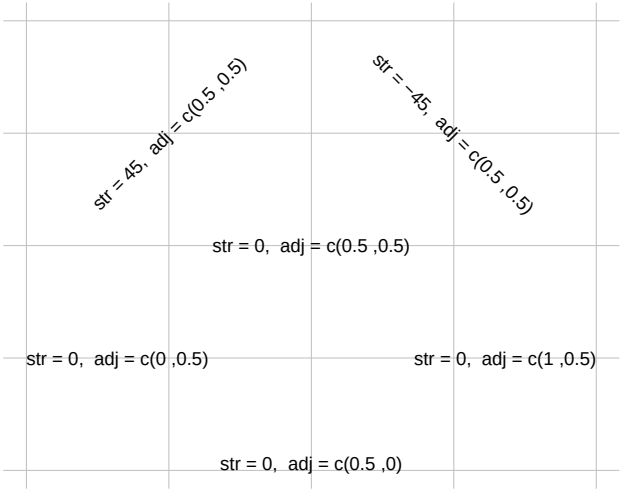

Land Animal Speeds
Cheetah
Pronghorn Antelope
Thomson's Gazelle
Lion
Wildebeest
Quarterhorse
Cape Hunting Dog
Elk
Coyote
Gray Fox
Mongolian Wild Ass
Zebra
Hyena
Greyhound
Whippet
Jackal
Mule Deer
Rabbit (domestic)
Giraffe
Reindeer
Cat (domestic)
Grizzly Bear
Wart Hog
White−Tailed Deer
Human
Elephant
Black Mamba Snake
Six−Lined Race Runner
Wild Turkey
Squirrel
Pig (domestic)
Chicken
Spider (Tegenaria atrica)
Giant Tortoise
Three−Toed Sloth
Garden Snail
0
2
25
50
75
100
Speed (km/hr)

Justification and Rotation of Strings

By default, the R graphics systems draws strings centred at locations specified by the arguments `x` and `y`.

The optional arguments `srt` and `adj` make it possible to rotate the string and change its placement relative to the specified coordinates.

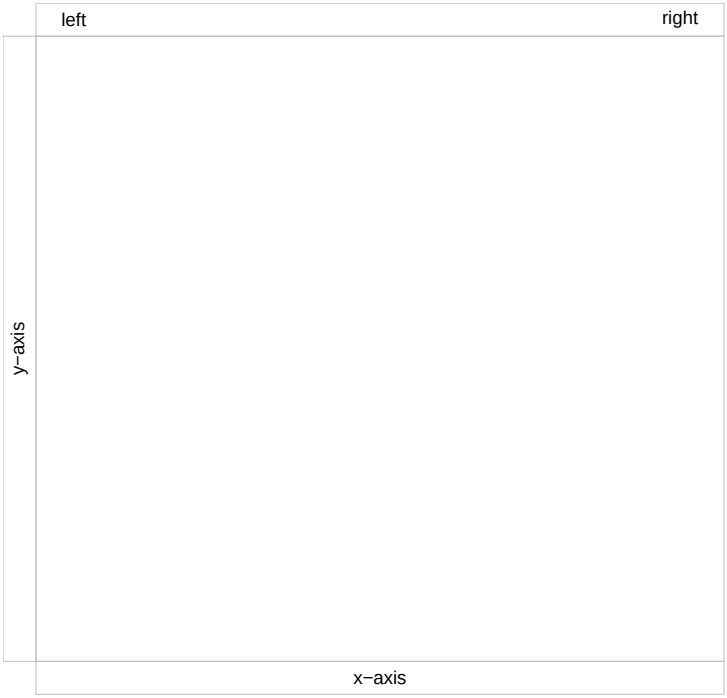
First, rotation takes place about the specified point and then the string is justified with respect to the point.


str = 45, adj = c(0.5 ,0.5)
str = −45, adj = c(0.5 ,0.5)
str = 0, adj = c(0.5 ,0.5)
str = 0, adj = c(0 ,0.5)
str = 0, adj = c(1 ,0.5)
str = 0, adj = c(0.5 ,0)

Label Rotation and Justification

Rotation and justification can be used to place labels within labelling regions.

```
> layout(matrix(c(0, 4, 0,
                  3, 1, 0,
                  0, 2, 0), nc = 3, byrow = TRUE),
         width = c(lcm(1), 1, lcm(1)),
         height = c(lcm(1), 1, lcm(1)))
> par(mar = rep(0, 4), cex = 1)
> plot.new()
> plot.new(); text(.5, .5, "x-axis");
> plot.new(); text(.5, .5, "y-axis", srt = 90)
> plot.new()
> text(0, .5, "left", adj = 0)
> text(1, .5, "right", adj = 1)
```


left
right
y−axis
x−axis

Scaling of Axis Limits

Normally, a call to `plot.window` (or any high-level plot which calls `plot.window`) expands the plot limits by 8%.

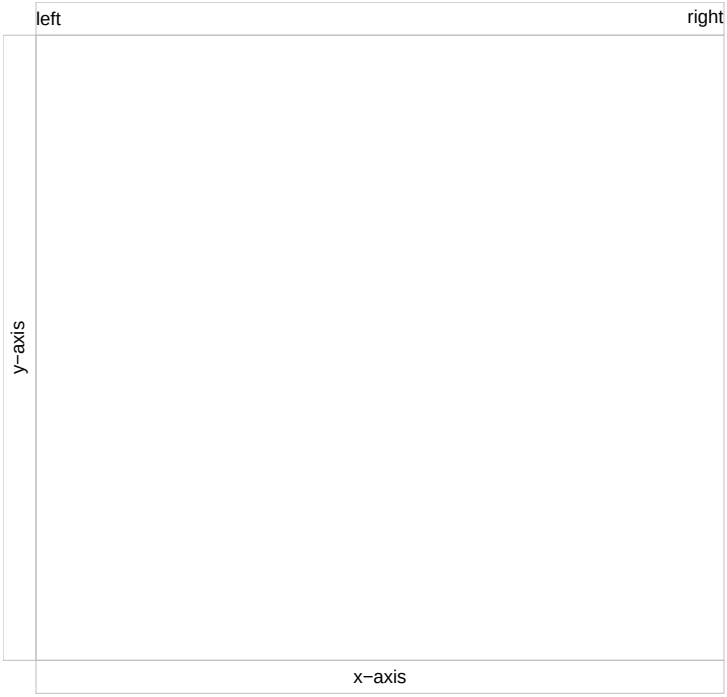
This is to prevent plotting symbols from overlapping the edge of the graphics area.

This expansion can be prevented by adding either or both of the optional arguments `xaxs = "i"` and `yaxs = "i"`.

If the last `plot.new` call was followed by

```
> plot.window(xlim = c(0, 1), xaxs = "i",
              ylim = c(0, 1), yaxs = "i")
```

The labels would be placed at the edges of the label region.


left
right
y−axis
x−axis