

Chapter 2

Plotting with R

An important part of scientific computing and data analysis is graphical visualization, an area in which R is very strong. R has many specialized graph types, some of which we will explore later. However, for many scientific purposes just a few types will suffice. We note especially graphs for data, functions, and histograms. We'll show simple examples of these first, and then show how they can be customized. For details on R graphics, see the book by Murrell [47].

2.1 Some common plots

2.1.1 Data plot

Graphing is particularly useful in descriptive statistics, to get an initial idea of how the data are distributed, whether their distribution is normal, etc. R makes it easy to do this.

We begin by generating a vector x of numbers from 0 to 100, and a vector $y = 3x - 2$, and then plotting y vs. x .

```
> x = seq(0,100, by=10)
> y = 3*x-2
> plot(x,y)
```

The simple `plot` command gives us a default graph with open circles as points, the names of the x and y variables as axis labels, and no title. To plot data with a line, instead of points, use `plot(x,y, type="l")`. See Section 2.2.2 for more variations on this theme.

A straight-line plot more relevant to biochemistry is the Lineweaver-Burk plot of Michaelis-Menten enzyme kinetics:

```
> Vmax = 10; Km=0.1 # Concentration in millimolar
> # Substrate concentrations
```

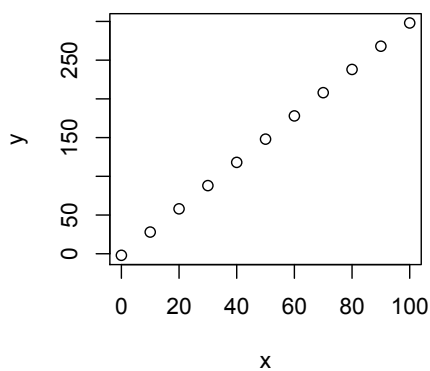


Fig. 2.1 Default data plot

```
> S = c(.01, .02, .05, .1, .2, .5, 1, 2, 5)
> v = Vmax*S/(Km+S) # Reaction velocity
> plot(1/S, 1/v) # Lineweaver-Burk transformation
```

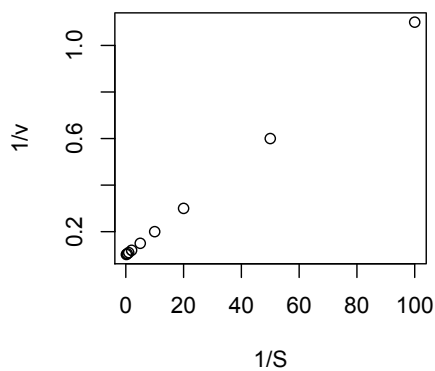


Fig. 2.2 Lineweaver-Burk data plot

2.1.2 Bar plot

Another common way of plotting data is a bar chart, which is known in R as a barplot. In the previous chapter we defined a matrix `drug.test` which gave the percentage survivors in two treatment groups after three time periods.

```
> drug.test = matrix(c(80, 70, 55, 70, 50, 35),
  nrow=2, ncol=3, byrow=TRUE,
  dimnames = list(c("Drug", "Placebo"), c("1", "2", "3")))
> drug.test
      1  2  3
Drug   80 70 55
Placebo 70 50 35
```

We construct a barplot from this matrix with the following commands:

```
> barplot(drug.test, beside=TRUE, legend.text=TRUE,
  ylim=c(0, 100),
  main="Drug Test", xlab="Months", ylab="% Survival")
```

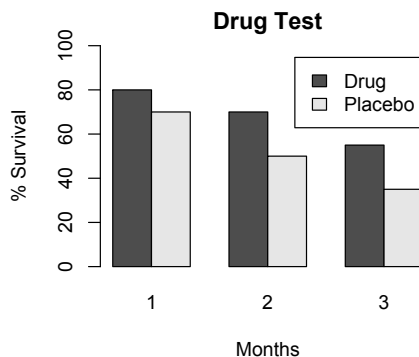


Fig. 2.3 Barplot of `drug.test` data

We have customized this plot by giving it a title with the `main` command, labeling the axes with `xlab` and `ylab`, and setting explicit limits on the `y` axis. We have also said that we wanted a plot in which the two treatment groups are plotted beside each other (the default, `beside = FALSE`), gives stacked bars; and that we wanted a legend (the default is `legend.text = NULL`). See `?barplot` and Chapter 13 to see other options for `barplot`, as well as some more elaborate examples.

2.1.3 Function plot

To graph a function we use the `curve` function.

```
> curve(x*sin(x), -10, 10, main="Function Plot")
```

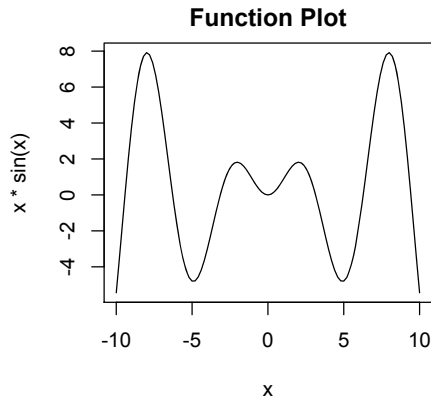


Fig. 2.4 Function plot of $x\sin(x)$ vs x

The expression to be graphed must be a function of x , or the name of the function to be plotted. The `-10, 10` arguments are the `from, to` limits of the range. If they are in positions 2 and 3, just after `expr` in position 1, they don't need to be named explicitly. By default, `curve` evaluates the function at $n = 101$ points; that number can be changed as one of the options of `curve()`.

2.1.4 Histogram

Histograms are important to visually inspect the distribution of repeated measurements. For example, we generate 100 normally-distributed random numbers with `mean = 2` and `standard deviation = 0.2`, and plot them using the `hist` function

```
> y = rnorm(100, mean=2, sd=.2)
> hist(y)
```

The title “Histogram of y ” is automatically added by the `hist` function. You can specify a different title with `main = "Desired Title"`.

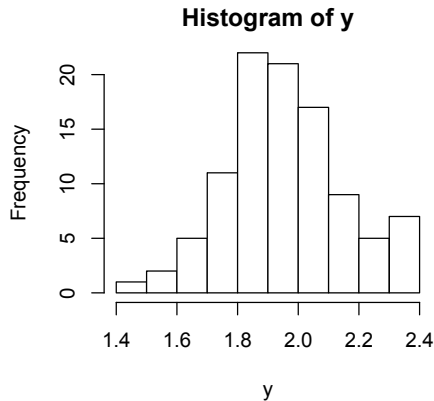


Fig. 2.5 Histogram of 100 normally-distributed random numbers

2.1.5 Three-dimensional plot

Sometimes we want to plot a dependent variable as a function of two independent variables. The R function `persp` gives such a plot with a customizable perspective that enables us to get the most informative view. A good example (simplified a bit to print in black and white) is given in R Help for `persp`. We first view z from the front as a function of x and y , and then rotate it by 30 degrees in the horizontal and vertical directions. (User-defined functions will be discussed in the next chapter.)

```
x = seq(-10, 10, length= 30)
y = x
f = function(x,y) { r = sqrt(x^2+y^2); 10 * sin(r)/r }
z = outer(x, y, f) # Forms matrix z using function f
par(mfrow = c(1,2))
persp(x,y,z)
persp(x,y,z,theta = 30, phi = 30)
```

Consult R Help to learn the many options available to customize the appearance of the plot produced by `persp`.

The value of z as a function of x and y can also be visualized by a contour map or a two-dimensional image colored according to the value of z , as in a topographical map. See `contour` and `image` in R Help for information on these functions.

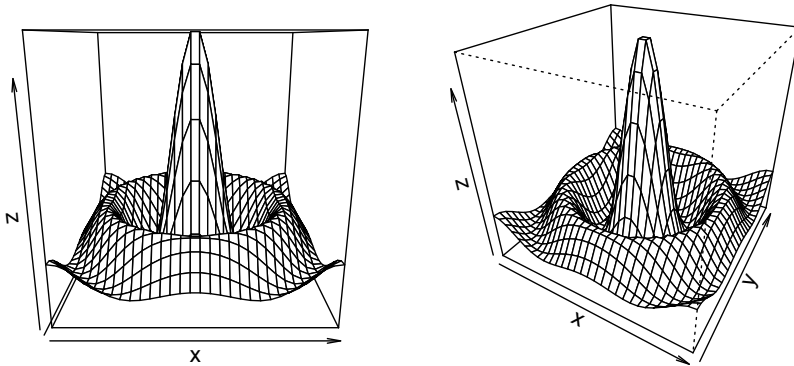


Fig. 2.6 Perspective plots. (left) Front view; (right) rotated by 30 degrees about θ and ϕ angles

2.2 Customizing plots

2.2.1 Different plot characters

By default, R plots points with open circles, but many other symbols are available. For example, to plot with closed circles, use `pch=16`, where `pch` stands for “plot character”. Here’s a Lineweaver-Burk plot with closed circles.

```
> plot(1/S, 1/v, pch=16)
```

Other point types are available using `pch` values between 0 and 25. The default (open circles) is `pch = 1`. Here is some code (don’t worry if you don’t understand it now) that shows the various point characters (`pch`) and line types (`lty`) available.

```
> plot.new()
> plot.window(xlim=c(-.5,26.5),ylim=c(0,8), asp=1)
> k = 0:25
> zero = 0*k
> text(k,8+zero, labels=k)
> points(k,7+zero,pch=k,cex=2)
> i = 6:1
> abline(h=7-i,lty=i)
> axis(2,at=1:8,labels=c(paste("lty =",i),"pch","k"), las=2)
```

In most circumstances you would use the simplest plot characters, 1–3 and 15–17.

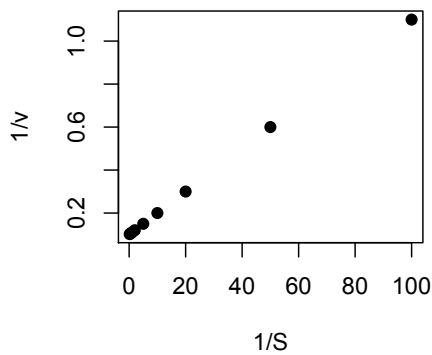


Fig. 2.7 Lineweaver-Burk plot with filled circles as data points

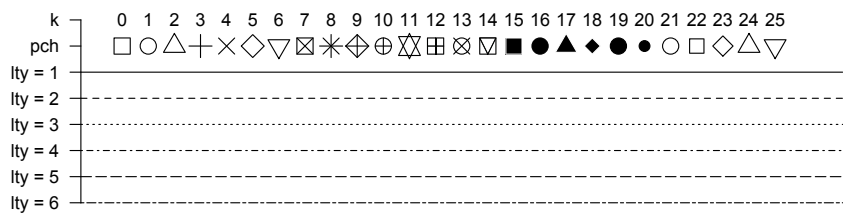


Fig. 2.8 Plot symbols and line types

The size of the plot characters can be controlled with `cex`. The default size is `cex=1`. Point sizes can be halved with `cex=0.5`, or made 50% larger with `cex=1.5`, etc.

```
> x = seq(0,100, by=10)
> y = 3*x-2
> plot(x,y, pch=17, cex=2)
```

2.2.2 Plotting data with a line

To plot with a line, use `type = "l"`. (To plot with points, you can also explicitly specify `type = "p"`.) To get both points and a line, `type = "o"` (for over-strike), or `type = "b"` for both.

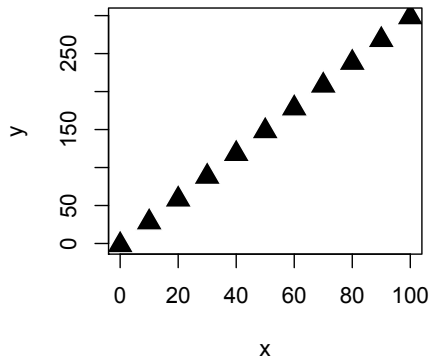


Fig. 2.9 Enlarged plot characters

```
> par(mfrow=c(1,2))
> plot(x,y,pch=16, type="o")
> plot(x,y,pch=16, type="b")
```

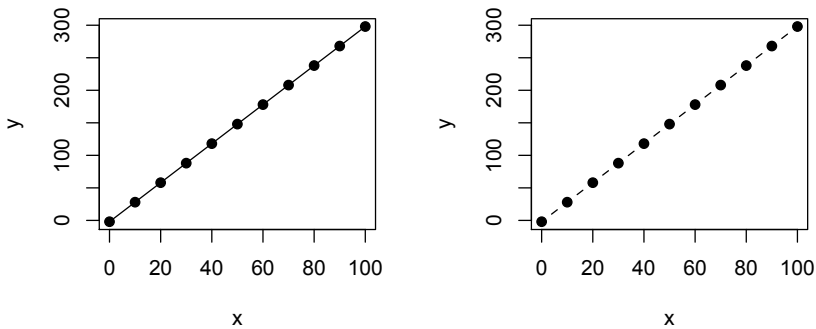


Fig. 2.10 Combining points and lines. (left) “o”; (right) “b”

In either data plots or function curves, you can specify the line type with `lty = n`, where n ranges from 1 to 6 (see above). You specify the line width with `lwd`. For example, here’s a plot of the initial velocity of an enzyme according to Michaelis-Menten kinetics. . You can thicken the line , or plot with dots rather than a solid line.

```
> Vmax = 10; Km = 0.1 # Concentration in millimolar
> S = c(.01, .02, .05, .1, .2, .5, 1, 2, 5)
> v = Vmax*S/(Km+S)
```



```
> plot(S,v,type="l")
> plot(S,v,type="l",lwd=2) # Thicker line
> plot(S,v,type="l",lty=3)  # Dotted line
```

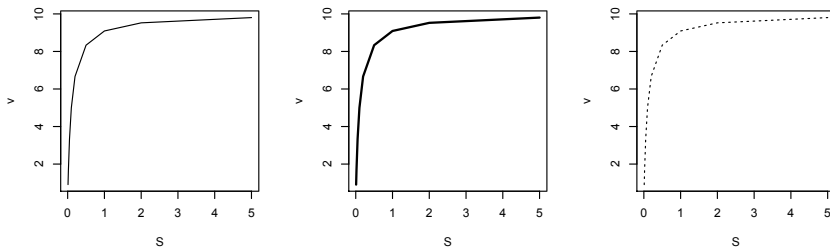


Fig. 2.11 Plotting data points with a (left) line, (center) thicker line, (right) dotted line

2.2.3 Adding title and axis labels

To put a title on the graph, use "main". (If a subtitle below the graph is desired, add it with "sub".) To change the x- and y-axis labels from the defaults, use "xlab" and "ylab". (We already did this for the `barplot` example above.) Be sure to surround the names with quotation marks.

```
> plot(S,v,main="Michaelis-Menten Kinetics",
      xlab="S/mM", ylab="v/(mM/s)")
```

If a graph can be produced without title, subtitle, legend, or axis labels, these items can be added later (while the graph is still active) with the `title` command. See `?title` in R Help for details.

2.2.4 Adding colors

To change the colors of points or lines, use the `col` command:

```
> x = 1:10
> y = x^3-2*x^2+4*x-1
> plot(x,y,pch=16,cex=1.3,col="red")
```

Since this book is printed in black and white, we do not show the resulting colored graph.

You can specify colors either by name (in quotes) or by number. The first four are (1) "black", (2) "red", (3) "green", and (4) "blue". You can learn more

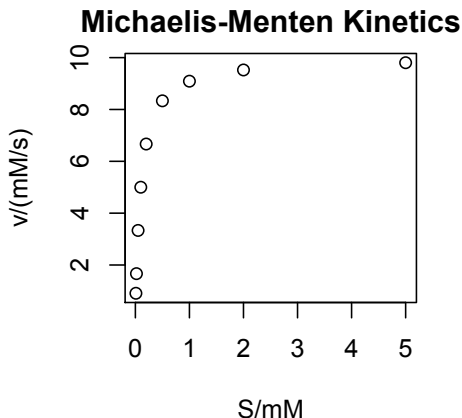


Fig. 2.12 Placing a title above the graph

about the colors available in R, and how to use them, by typing `?colors`. Note that in a typical graph for publication, you would generally stick to black and white, distinguishing data series by point or line type. However, for a poster or computer-based presentation, colors are helpful.

To customize colors or shading of barplots, as well as many other barplot features, see `?barplot` in the R help system.

2.2.5 Adding straight lines to a plot

You can add straight lines to a plot with the `abline` function. This can take several forms. If you want to draw a line with specified intercept a and slope b (hence the name), use `abline(a, b)`. For example, to draw a horizontal dashed line (slope $b = 0$) on the function plot at the beginning of this chapter, to estimate where the function crosses the y axis (intercept $a = 0$):

```
> curve(x*sin(x), -10, 10, main="Function Plot")
> abline(0, 0, lty=2)
```

Note that you add the line to the already drawn basic curve. This method of sequentially adding details to a graph, rather than specifying everything at once initially, is standard in R. But the graph must be “open”, the latest one drawn.

Another variation on `abline` is to use it to draw a vertical or horizontal line, using “v” and “h”, respectively. For example, to draw a vertical line through the value of S corresponding to the Michaelis constant K_m , and a horizontal line through the value of v corresponding to the half-maximum velocity $V_{max}/2$:

```
> # Kinetic parameters
> Vmax = 10; Km = .001
```

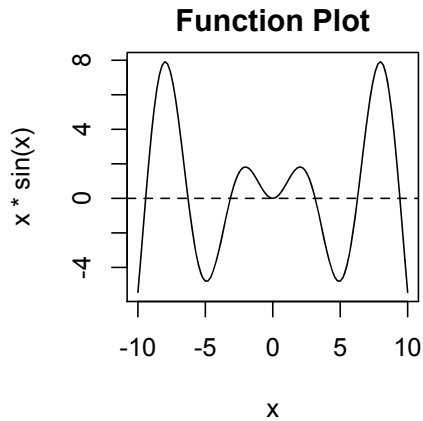


Fig. 2.13 Adding a line to the graph with `abline`

```
> # Substrate concentrations
> S = c(.0001, .0003, .001, .002, .005, .01, .02)
> v = Vmax*S/(Km+S) # Calculated velocities
> plot(S, v, type="b")
> abline(v=Km, lty=3) # Vertical line at Km
# Don't confuse v here, which means "vertical",
# with the enzyme velocity
> abline(h=Vmax/2, lty=3) # Horizontal line at Vmax/2
```

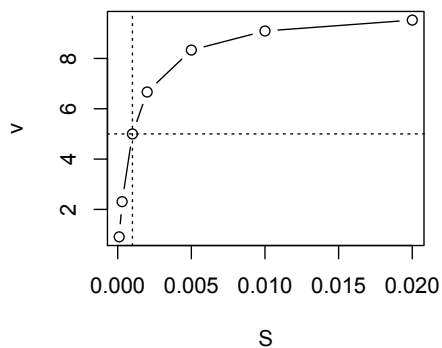


Fig. 2.14 Adding horizontal and vertical lines with `abline` to locate the half-maximum velocity and Michaelis constant

2.2.6 Adjusting the axes

R automatically sets the upper and lower limits of the horizontal and vertical axes. Sometimes you may want to use different limits, which you can do with `xlim=c(xlo,xhi)` and `ylim=c(ylo,yhi)`. For example, in the Lineweaver-Burk plot of the simulated data above,

```
> plot(1/S, 1/v, pch=16, main="Lineweaver-Burk Plot",
      ylim=c(0,1.2))
# Draw line corresponding to intercept and slope of
# Lineweaver-Burk plot
> abline(1/Vmax,Km/Vmax, lty=3)
```

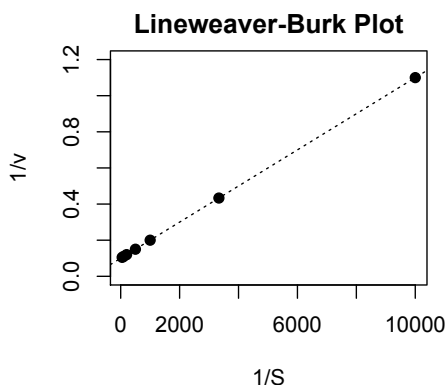


Fig. 2.15 Lineweaver-Burk plot with axes adjusted and line added to show slope and intercept

2.2.7 Customizing ticks and axes

By default, R draws the axis ticks outside the graph. Most commonly in scientific journals, the ticks are inside. This can be set by a command like `tck=0.03`, where the positive value of `tck` means ticks inside the graph, and 0.03 is the length of the ticks relative to the graph.

```
> plot(1/S, 1/v, pch=16, tck=0.03,
      main="Lineweaver-Burk Plot",ylim=c(0,1.2))
> abline(1/Vmax,Km/Vmax, lty=3)
```

R's choice of ticks on the axes is usually adequate. To learn how to customize, call `?axis` in the R help system. The `axis` documentation will also tell you, among other things, how to draw additional axes on the top- or right-hand side of

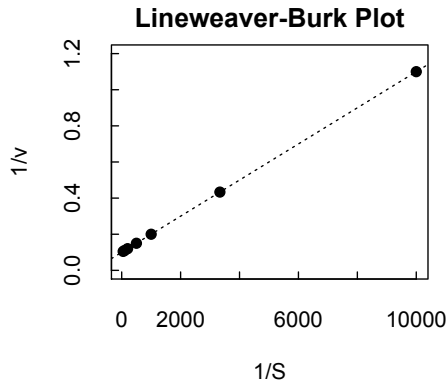


Fig. 2.16 Graph with customized ticks

the plot, using the `side` option. This would be useful, for example, if you wanted to plot two variables which had significantly different magnitudes. The right-hand axis (4) could show the scale for the second variable.

2.2.8 Setting default graph parameters

Rather than setting graph parameters individually for each graph, if you have a preferred style you can set it with the `par` command. For example, to generate default plots with ticks inside and filled circles, use

```
> par(tck=0.03, pch=16)
```

Every graph that you draw from then on until you quit the session will use these defaults, unless you change or override them. Of course, settings for a particular graph can be changed as above. `par` has many options, as you can see from reading R Help, that enable you to customize many aspects of a plot layout.

2.2.9 Adding text to a plot

You may wish to annotate a plot by adding text within it. For example, you can place labels indicating the intercept $1/V_{max}$ and slope K_m/V_{max} on the Lineweaver-Burk plot from the data at the beginning of this chapter as follows:

```
> text(15, 0.5, "1/Vmax")
> text(60, 0.5, "Km/Vmax")
```

To add text in the margins of a plot, rather than within the plotting area, use `mtext`; see R Help for details. If there are several data series or symbols in a graph,

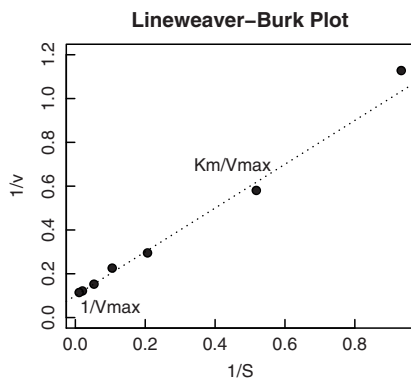


Fig. 2.17 Graph with added text and customized graph parameters

one can annotate with a legend. We'll see examples of this below. Plots can also be annotated with mathematical symbols and expressions. To learn about this, call `?plotmath` in R.

2.2.10 Adding math expressions and arrows

R has useful capabilities for adding mathematical symbols and expressions to a plot, and for pointing out significant features with arrows. As a simple example, we plot the function $\sin(x)/x$ which appears in x-ray diffraction, give the equation within the plot, and draw an arrow to indicate a significant maximum.

```
> x = seq(-20,20,by=0.1)
> y = sin(x)/x
> plot(x,y, type = "l")
> text(-15,0.7,expression(frac(sin(x),x)))
> arrows(11,0.3,8,0.15, length=0.1)
> text(11,0.45,"2nd\nmaximum") # \n means "newline"
```

For details, see `plotmath`, `demo(plotmath)`, and `arrows`. As we'll see in the section on error bars at the end of this chapter, arrows are also used to draw error bars.

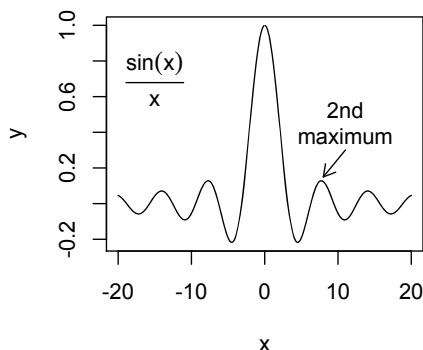


Fig. 2.18 Plot with added math expressions and arrows

2.2.11 Constructing a diagram

Sometimes you will want to construct a block diagram, organization chart, or similar figure composed mainly of straight lines, arrows, and text. R, with its large number of graphic constructs, enables you to do this fairly readily. As an example, we consider the code used to construct the diagram for countercurrent distribution in Chapter 8. Comparing the commented code with the diagram should illuminate most of the steps, but several points are worth noting.

- It is useful to define variables that specify the dimensions, so that sizes and proportions can be changed in one place rather than in each line of code.
- The `plot.new()` function defines a new plot without axes, labels, or outlining box. The `plot.window()` function sets the horizontal and vertical scales.
- The `segments(x0, y0, x1, y1)` function draws straight lines from x_0, y_0 to x_1, y_1 . It uses the same syntax as the `arrows` function, but does not include arrowhead length and angle. Note the use of `for...` loops to place repetitive vertical lines.
- Subscripts in plotted text are indicated by square brackets. See `plotmath`, `expression`, and `substitute` in R Help for details and other possibilities for math notation. The `tH` and `tL` functions are defined for brevity and convenience in writing the commands to place text on the high and low levels.

```
# Assume cells with square sides of length 1
# Assign variables to the dimensions in the diagram.
lmarg = 1; rmarg = 1 # Margins for text and arrows
n1 = 4; n2 = 1.5; n3 = 2 # Lengths of horizontal walls
nv = 0.3 # Lengths of vertical walls
pw = lmarg + n1 + n2 + n3 + rmarg # Plotwidth
ph = 2 # Plotheight
```

```

# Prepare a new, empty plot and dimension its window
plot.new()
plot.window(c(0,pw),c(0,ph))

# Draw straight lines using segments
# Lower horizontals
# Lower left
segments(lmarg,0,lmarg + n1,0)
# Lower middle
segments(lmarg + n1,0,lmarg + n1 + n2,0,lty=2)
# Lower right
segments(lmarg + n1 + n2,0,lmarg + n1 + n2 + n3,0)
# Upper horizontals
# Upper left
segments(lmarg,2,lmarg + n1,2)
# Upper middle
segments(lmarg + n1,2,lmarg + n1 + n2,2,lty=2)
# Upper right
segments(lmarg + n1 + n2,2,lmarg + n1 + n2 + n3,2)
# Middle horizontals
# Middle left
segments(lmarg,1,lmarg + n1,1, lty=3)
# Middle middle
segments(lmarg + n1,1,lmarg + n1 + n2,1,lty=2)
# Middle right
segments(lmarg + n1 + n2,1,lmarg + n1 + n2 + n3,1,
  lty=3)
# Left verticals
for (i in 1:(n1+1)){
  segments(lmarg + i -1,2,lmarg + i -1,2-nv)
  segments(lmarg + i -1,1+nv,lmarg + i -1,1-nv)
  segments(lmarg + i -1,nv,lmarg + i -1,0)
}
# Right verticals
for (i in 1:(n3+1)){
  segments(lmarg + n1 + n2 + i -1,2,
    lmarg + n1 + n2 + i -1,2-nv)
  segments(lmarg + n1 + n2 + i -1,1+nv,
    lmarg + n1 + n2 + i -1,1-nv)
  segments(lmarg + n1 + n2 + i -1,nv,
    lmarg + n1 + n2 + i -1,0)
}

# Annotate diagram with text, including subscripts
# For brevity, define functions for high and low text
tH = function(x,i) text(x, 1.5, substitute(H[i]))

```



```

tL = function(x,i) text(x, 0.5, substitute(L[i]))
# Upper
tH(lmarg-0.8,"in"); tH(pw-rmarg+0.8,out)
tH(1.5,1); tH(2.5,2); tH(3.5,3); tH(4.5,4)
tH(n1+n2+1.5,n-1); tH(n1+n2+2.5,n)
# Lower
tL(lmarg-0.8,out); tL(pw-rmarg+0.8,"in")
tL(1.5,1); tL(2.5,2); tL(3.5,3); tL(4.5,4)
tL(n1+n2+1.5,n-1); tL(n1+n2+2.5,n)

# Draw arrows for in- and out-flows
arrows(lmarg-0.4,1.5,lmarg,1.5,.1,30)
arrows(pw-rmarg,1.5,pw-rmarg+0.4,1.5,.1,30)
arrows(lmarg,0.5,lmarg-0.4,0.5,.1,30)
arrows(pw-rmarg+0.4,0.5,pw-rmarg,0.5,.1,30)

```

2.3 Superimposing data series in a plot

Often you'll want to plot two sets of data on the same graph, or two functions, or data points and a function. To do this, graph one of the series first, then the other. You can then add a legend to identify the series. It's important to note that, for this to work, the axis limits of the first-drawn graph must be large enough to accommodate the subsequent series. For example,

```

> x = 1:10
> y1 = x^2-3*x+2
> y2 = x^2-2*x+3
> plot(x,y1, pch=16)
# Use points(), not plot(), to add the second series
> points(x,y2,pch=1)

```

The problem with this graph (Figure 2.19) is that the highest value of `y2` is greater than the axis limit set by `y1`, so the last point is cut off. To avoid this, first test the minima and maxima of the series using `range()`, then set the axis limits accordingly. Note also that the y axis is labeled "y1", which probably should be changed to the generic "y".

```

> range(y1)
[1] 0 72
> range(y2)
[1] 2 83
> plot(x,y1,ylim=c(0,90), ylab="y",pch=16)
> points(x,y2,pch=1)

```

A more general way to set `ylim` would be to use `min(c(y1,y2))` for the minimum value (or perhaps `min(c(0,y1,y2))` if you wanted the plot to start at 0

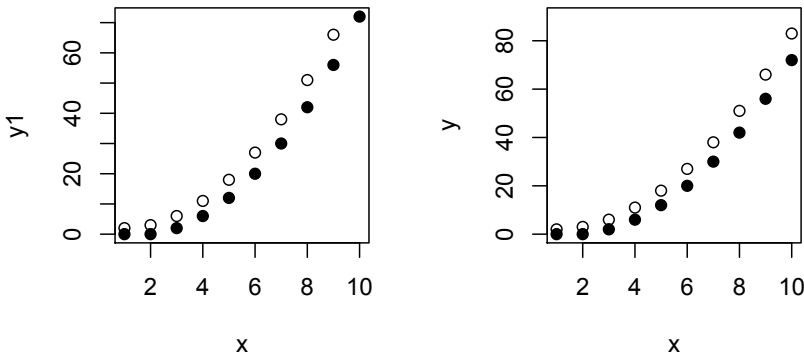


Fig. 2.19 Plotting two data sets on the same graph. (left) original, (right) after adjustment of y axis limits

and knew that no y values were below 0), and `max(c(y1, y2))` for the maximum value. Combining,

```
ylim = c(min(c(0, y1, y2)), max(c(y1, y2))).
```

The function `matplot` provides a more concise way to superimpose data series. `matplot` plots the columns of a matrix individually as a function of x. The matrix can be assembled using `cbind`. Using the example above, `cbind(y1, y2)` is a 10×2 matrix, and x is a length-10 column vector. If you simply enter the command `matplot(x, cbind(y1, y2))` you will get a graph in which the points in the two data series are denoted by the numbers 1 or 2, and in which 1 is printed in black (`col=1`) and 2 in red (`col=2`). (Try it.) To get a plot like the one above, all in black and with point characters, you must use something like

```
> matplot(x, cbind(y1, y2), type="p", pch=c(16, 1),
  col=c(1, 1), ylab="y")
```

or

```
> y = cbind(y1, y2)
> matplot(x, y, type="p", pch=c(16, 1), col=c(1, 1))
```

Note that `matplot` automatically sets `ylim` to include both data series.

To identify the series, add a legend with the `legend` function, which specifies the x and y coordinates of the legend, its text, accompanying symbols or line types, and whether the legend is surrounded by a box. To suppress the box, use `bty="n"` where `bty` stands for “box type”.

```
> legend(1, 85, legend=c("y1", "y2"), pch=c(16, 1), bty="n")
```

To plot a line (for example, a fitting function) on a data plot, first plot the points, then the line with the function `lines`, then the legends. You may have to experiment a bit to get the legends located properly.

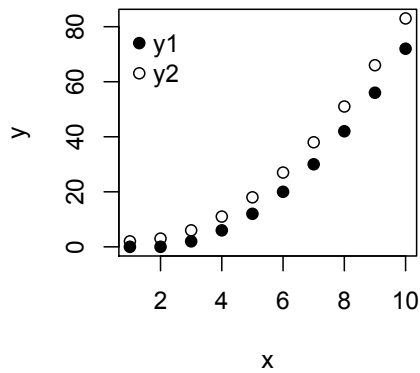


Fig. 2.20 Matplot with legend

```
> x = 1:10
> y1 = x^2-3*x+1 + rnorm(10,0,5)
> plot(x,y1)
> lines(x^2-3*x+1)
> legend(2, 60, legend="y1", pch=16, bty="n")
> legend(1.5, 50, legend="fit", lty=1, bty="n")
```

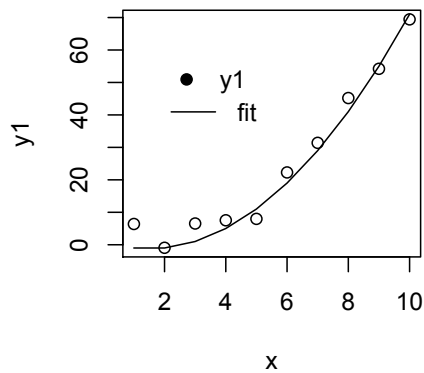


Fig. 2.21 Data plot with superimposed fitting curve

To plot two or more curves, use the `add = TRUE` option to the second and later curve command. For example,

```
> curve(x^3-3*x^2+2*x-4, -10, 10,
```

```

ylim=c(-1500,1500),ylab="f(x) ")
> curve(x^3+3*x^2+2*x-100,-10,10,
  lty=2, add=TRUE)
> legend(-9,1000,legend=c("f1(x)", "f2(x)"),bty="n",
+ lty=c(1,2))
# "+" is line continuation in R

```

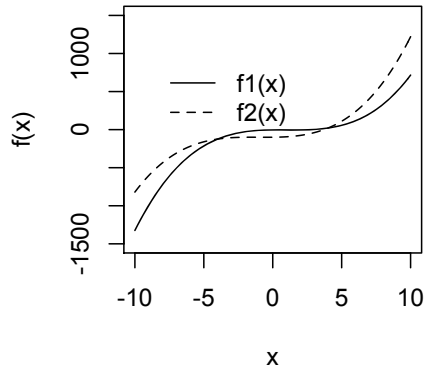


Fig. 2.22 Two superimposed curves with legend

2.4 Placing two or more plots in a figure

R provides several methods for placing several plots in a single figure. The most common method is `par(mfrow=c(nr,nc))` where `nr` is the number of rows of plots and `nc` is the number of columns. `mf` stands for “multiple figures”. With `mfrow`, the plots are successively filled in row order. The alternative `mfcol` fills the plots in column order.

For example, consider Michaelis-Menten enzyme kinetics with values of the reaction velocity v calculated from the substrate concentration S and the kinetic parameters V_{max} and K_m . We plot v vs. S and the three common linear transformations of the data in a single figure as follows.

```

> S = c(0.1,0.2,0.5,1,2,5,10,20)*1e-6
> Km = 2e-6
> Vmax = 10
> v = Vmax*S/(Km+S)
> par(mfrow=c(2,2)) # Set up 2 x 2 plot figure
> plot(S,v,type="o", main="Michaelis-Menten")

```

```
> plot(1/S, 1/v, type="o", main="Lineweaver-Burk")
> plot(v/S, v, type="o", main="Eadie-Hofstee")
> plot(S, S/v, type="o", main="Hanes-Woolf")
> par(mfrow=c(1,1)) # Return to single plot
```

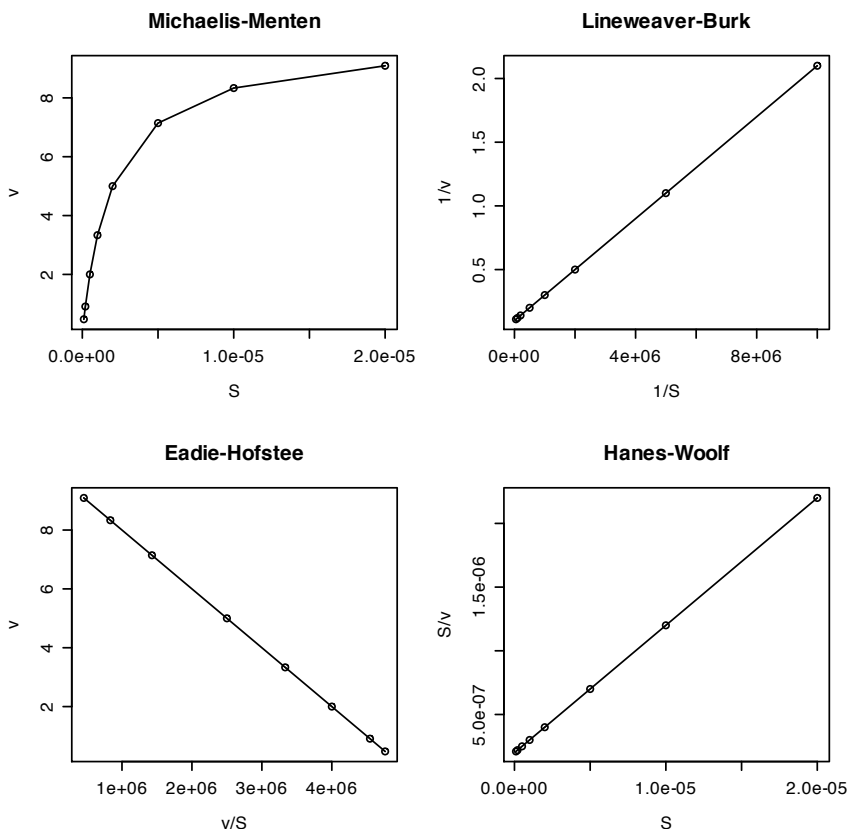


Fig. 2.23 Four plots in the same figure

Note that at the end we used `par(mfrow=c(1,1))` to return the graphic parameters to a single plot format. This is necessary since `par` commands stay in effect until changed.

There is a good deal of space between the plots in this figure. It might be desired to compress this space, especially if the individual plots did not have titles. This can be done with the `mar` option to `par`. As explained in R Help for `par`, `mar` is “A numerical vector of the form `c(bottom, left, top, right)` which gives the number of lines of margin to be specified on the four sides of the plot. The default is `c(5, 4, 4, 2) + 0.1`.” To compress the space between the plots, these values could be decreased.

`mfrow` and `mfcoll` divide the figure into equal-sized regions; all plots are the same size. If it is desired to have different-sized plots, R provides the functions `layout` and `split.screen`. Consult R Help to see how these work.

2.5 Error bars

We use error bars in graphs to denote statistical variability or uncertainty. To add error bars, we use the `arrows` function in R. According to the documentation for that function in the help system,

Sample usage:

```
arrows(x0, y0, x1, y1, length = 0.25, angle = 30)
```

Arguments:

```
x0, y0: coordinates of points from which to draw.
x1, y1: coordinates of points to which to draw.
length: length of the edges of the arrow head (inches).
angle: angle from the shaft of the arrow to the edge
       of the arrow head.
```

There are several other arguments that we won't need. Here's an example of a plot of ten uniformly distributed numbers, with error bars equal to $\pm 10\%$ of the y value:

```
> x = 1:10
> y = runif(10)
> plot(x, y)
> arrows(x, y, x, y+.1*y, .05, 90); arrows(x, y, x, y-.1*y,
      .05, 90)
```

Here's an example of a bar chart with error bars, with data from six uniformly distributed random numbers between 0 and 1. This example also shows how to generate cross-hatching of the bars at a density of 15 lines per inch and a black color (medium gray is the default), to set the counterclockwise angle of the hatching to 45 degrees, and to show error bars only above the top of the bar, as is the common convention.

```
> x = runif(6)
> bar = barplot(x, names.arg = month.abb[1:6],
+ density = 15, angle = 45, col="black", ylim=c(0,1.1))
> stdev = x/10
> arrows(bar, x, bar, x + stdev, length=0.1,
      angle = 90)
```

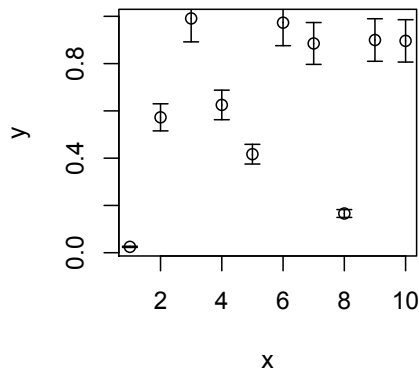


Fig. 2.24 Data plot with error bars

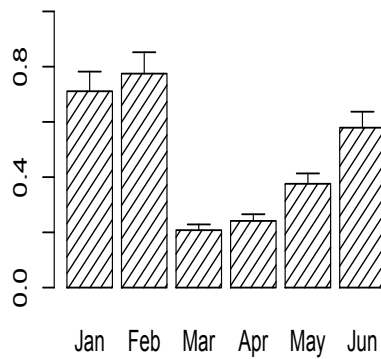


Fig. 2.25 Bar chart with cross-hatching and error bars

The labels are generated from the built-in R values for the three-letter abbreviations for the months (`month.abb`). The only other R built-in constants are the full names of the months (`month.name`), the upper-case (`LETTERS`) and lower-case (`letters`) letters, and `pi`.

2.6 Locating and identifying points on a plot

R has functions that enable identifying and placing points on a plot, and might be useful for digitizing a plot. The function `locator`, which “reads the position of

the graphics cursor when the (first) mouse button is pressed” can usefully serve as a digitizer for a plot. For example, the command `xy = locator(10)` followed by ten mouse clicks on an experimental curve or set of points, will yield a set of 10 (x,y) coordinates, which can be accessed by `xy$x` and `xy$y`. If desired, `locator` will put a point at each point clicked, or draw a line between clicked points.

A related function is `identify`. According to the R documentation, “`identify` reads the position of the graphics pointer when the (first) mouse button is pressed. It then searches the coordinates given in `x` and `y` for the point closest to the pointer. If this point is close enough to the pointer, its index will be returned as part of the value of the call.”

See R Help for details on how to apply these functions.

2.7 Problems

1. The BOD data in the R “datasets” package gives biological oxygen demand (mg/l) vs time (days) in an evaluation of water quality.

```
time = c(1, 2, 3, 4, 5, 7)
demand = c(8.3, 10.3, 19.0, 16.0, 15.6, 19.8)
```

Plot demand vs time, using filled black circles as the points. Give the plot an appropriate title.

2. Repeat Problem 1, with the following modifications: Change the minimum value of the y-axis to 0. Change the x-axis label to “days” and the y-axis label to “BOD”. Move the ticks to the inside of the graph. Connect the black points with a solid red line.
3. Assume that the demand measurements have $\pm 10\%$ error. Repeat Problem 2, with error bars on the points.
4. Assume that the water was treated to reduce BOD, and a second set of measurements gave the following results:

```
time = c(1, 2, 3, 4, 5, 6, 7, 8)
treated = c(5.1, 7.7, 10.2, 12.3, 14.0, 13.3, 15.5, 13.9)
```

Assume $\pm 10\%$ error for these data as well. Plot both untreated and treated data sets on the same graph, using different point and line types and/or colors for each plot. Add a legend (unboxed) that labels the data.

5. Add horizontal lines to the plot in Problem 4 to represent the mean values of the BODs in the two measurements. Make the line types the same as those that connect the points.
6. You guess that an approximate functional approximation to the data in Problem 4 is

```
treated = 5 + 8*time/(2 + time)
```

Add this line to a plot of the treated data.

7. Plot the untreated and treated BOD data as a barplot, with appropriate legend and title. Since data for 6 and 8 days are missing from the untreated set, use `NA` for those values. Make the untreated data solid blue, and the treated data solid red. See `?barplot` in R Help.
8. Modify the plot in Problem 7 to make the untreated bars shaded gray at 45 degrees counterclockwise, and the treated bars 45 degrees clockwise. Add error bars to both sets of bars.
9. Generate a vector `rn` of 1000 normally distributed random numbers with mean 0 and standard deviation 1. Use the command `par(mfcol=c(1,2))` to tell R to plot two graphs side-by-side. (See R Help on `?par`.) In the left-hand graph plot the histogram of `rn`; in the right-hand graph plot `hist(rn, freq=F)`. Note the difference in ordinate.
10. While still placing two graphs side-by-side, compare `hist(rn, breaks=10)` with `hist(rn, breaks=20)`. Put a title on each graph so you can tell them apart. Do you see a difference in the number of breaks? Consult R Help about `?hist` to learn how to control the number of breaks. Repeat the two graphs using that approach. Remember to reset the graphing system to one row, one column when you are done.



<http://www.springer.com/978-1-4419-0084-5>

Computer Simulation and Data Analysis in Molecular
Biology and Biophysics

An Introduction Using R

Bloomfield, V.

2009, XVI, 321 p., Hardcover

ISBN: 978-1-4419-0084-5