Graphics in R

Karline Soetaert
Centre for Estuarine and Marine Ecology
Netherlands Institute of Ecology
The Netherlands
October 2009

Abstract

R (R Development Core Team 2010) is the open-source (read: free-of-charge) version of the language S.

It is best known as a package that performs statistical analysis and graphics. However, R is so much more: it is a high-level language in which one can perform complex calculations, implement new methods, and make high-quality figures.

This document deals with making good-quality figures in R that can be used for scientific publications. It has been used as a half-a-day introduction into this topic.

A longer (and somewhat different) introduction to R can be found on: http://cran.r-project.org/doc/contrib/Soetaert_Scientificcomputing.zip

Keywords: Plotting, R.

1. Iintroduction

1.1. The R-software

Installing R

R is downloadable from the following web site: http://www.r-project.org/ Choose the precompiled binary distribution. On this website, you will also find useful documentation.

To use R for the examples in this document, you need to download package **marelacTeaching** (Soetaert and Meysman 2009) that contains data sets, used in this tutorial.

If you run R within windows, downloading specific packages can best be done within the R program itself. Select menu item `packages / install packages`, choose a nearby site (e.g. France (Paris)) and select the package you need.

Other useful software

We run R from within the Tinn-R editor, which can be downloaded from URL http://sourceforge.net/projects/tinn-r and http://www.sciviews.org/Tinn-R. This editor provides R-sensitive syntax and help.

However, the latest Tinn-R versions do not quite work as we would like it, so we prefer using an older version, 1.19.2.2. This can be downloaded from my personal page http://www.nioo.knaw.nl/users/ksoetaert [1]

From within the Tinn-R program, you launch R via the menu (`R/start preferred Rgui`).

Getting started

To be able to execute some examples, you need to

- Open tinn-R (or some other editor)
- Launch R from within the editor
- load the R-package **marelacTeaching**.

```
> require(marelacTeaching)
```

1.2. Quick overview of R

R-code is highly readable, once you realise that:

- `<-` is the assignment operator.
- everything starting with `#` is considered a comment.
- R is case-sensitive: `a` and `A` are two different objects.

There are two ways in which to work with R.

1. We can type commands into the R console window at the command prompt (>).

   For instance, enter in the console window:

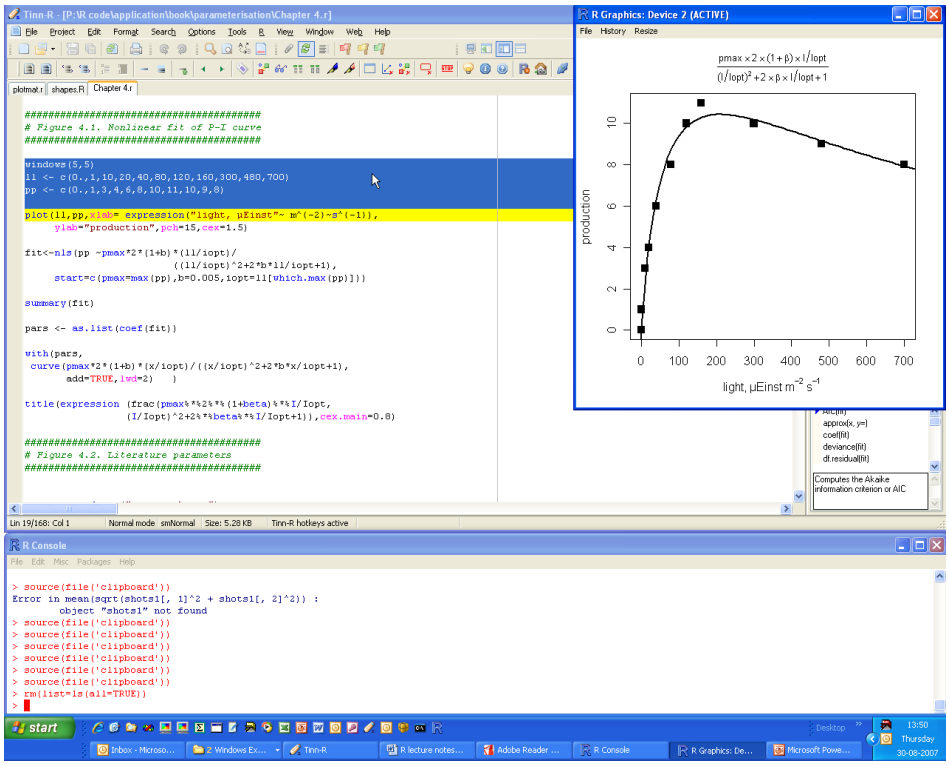
   ```
   > plot(1)
   ```

2. Alternatively, we can create R-scripts in an editor (e.g. Tinn-R) and save them in a file ("filename.R") for later re-use. R-scripts are sequences of R-commands and expressions. These scripts should be submitted to R before they are executed.

    If you use the Tinn-R editor, you can either submit the entire file (buttons 1,2), selected parts of the text (buttons 3,4), submit marked blocks (buttons 5,6) or line-by-line (last button).

A screen capture of a typical Tinn-R session, with the Tinn-R editor (upper window) and the R-console (lower window) is given below (figure 1.2. A script file is opened in the Tinn-R editor. Note the context-sensitive syntax (green=comments, blue= reserved words, rose = R-parameters). Several lines of R-code have been selected (blue area) and sent to the R-console, which has produced the graphics window that floats independently from the other windows.

[1] right-upper part, last item in the blue box



1.3. getting help

R has extensive graphical capabilities, and allows making simple (1-D, x-y), image-like (2-D) and perspective (3-D) figures.

Try:

```
> demo(graphics)
> demo(image)
> demo(persp)
```

to obtain a display of R s simple (1-D, x-y), image-like (2-D) and perspective (3-D) capabilities.

R has an extensive help facility. For instance, typing

```
> ?plot
> ?plot.default
> ?plot.window
> ?par
> ?points
```

will explain about the main plotting function, the default settings the graphical parameters, and the symbols that can be used.

Most of the help files also include examples. You can run all of them by using R-statement `example`.

For instance, typing into the console window:

```
> example(plot.default)
> example(points)
```

will run the examples, displaying each new graph, after you have pressed "<ENTER>" (try it!)

```
> example(pairs)
```

will run all the examples from the `pairs` help file. Function `pairs` is a very powerful way of visualizing pair-wise relationships.

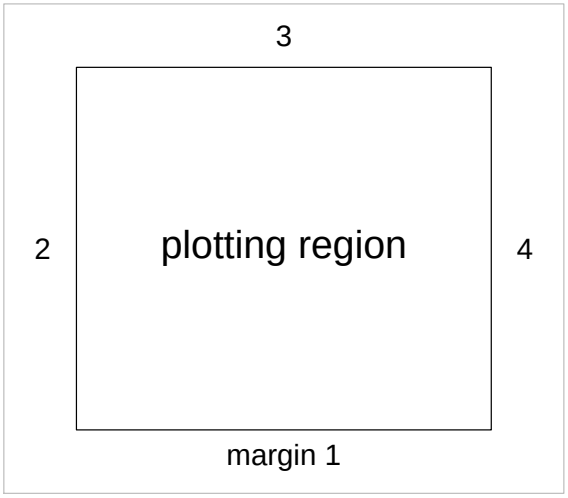


Figure 1: The four margins and the plotting region.

2. Plotting basics

Graphics are plotted in the figure window which floats independently from the other windows. If not already present, it is launched by writing (in windows):

```
> windows()
```

or

```
> x11()
```

A figure consists of a plotting region surrounded by 4 margins, which are numbered clockwise, from 1 to 4, starting from the bottom (figure 1).

R distinguishes between:

1. *high-level commands*. By default, these create a new figure, e.g. `plot`, `curve`, `matplot`, `pairs`,...
2. *low-level commands* that add new objects to an existing figure, e.g.
   - `lines`, `points`, `legend`, `text`, ... These add objects within the plotting region
   - `box`, `axis`, `mtext` (text in margin), `title`, ... which add objects in the plot margin
3. *graphical parameters* that control the appearance of:

- plotting objects, e.g.
  `cex` (size of text and symbols), `col` (colors), `lty` (line type), `lwd` (line width), `pch` (the type of points)
- graphic window, e.g.
  `mar` (the size of the margins), `mfrow` (the number of figures on a row), `mfcol` (number of figures on a column)

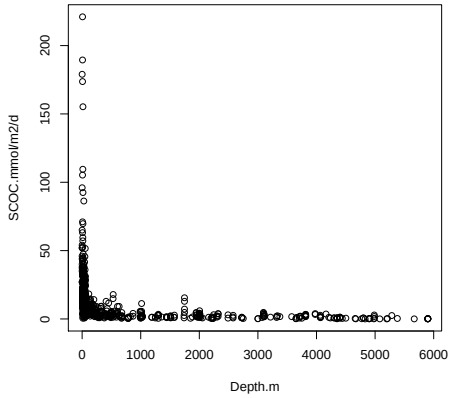


Figure 2: SCOC dataset, visualised with `plot` using the default settings - see text for R-code

3. The plot command

3.1. An x-y (scatter) plot

The data set `SCOC` from package `marelacTeaching` contains 584 measurements of sediment community oxygen consumption rates, as a function of water depth, and performed in deep-water sediments, either by in situ incubations or via modelling of oxygen microprofiles. We first display part of the data:

```
>require(marelacTeaching)
>head (SCOC)
```

```
  Depth.m SCOC.mmol/m2/d
1     7.0       221.0526
2     7.0       189.4737
3     0.5       179.0000
4     7.0       173.6842
5    14.0       155.2565
6    12.0       109.5238
```

A quick plot of this dataset shows the log-log relationship between water depth and SCOC (figure 2):

```
>plot(SCOC)
```

We now make a nicer figure (3), changing the pitch type (`pch = 16`), labelling the axes (`xlab, ylab`), providing a main title (`main`), and log-transforming both the x- and y- axis (`log = "xy"`):

```
>plot(x = SCOC[ ,1], y = SCOC[ ,2], log = "xy", main = "SCOC",
+      xlab = "water depth, m", ylab = "mmol O2/m2/d", pch = 16)
```

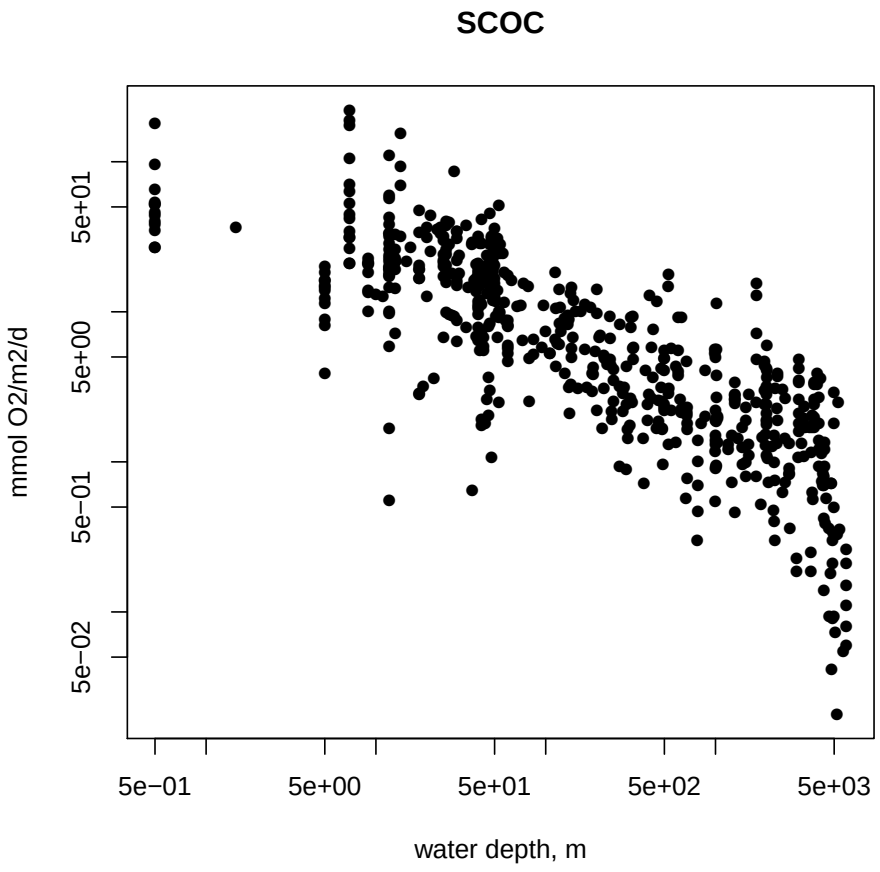


Figure 3: SCOC dataset,visualised with `plot`, improved - see text for R-code

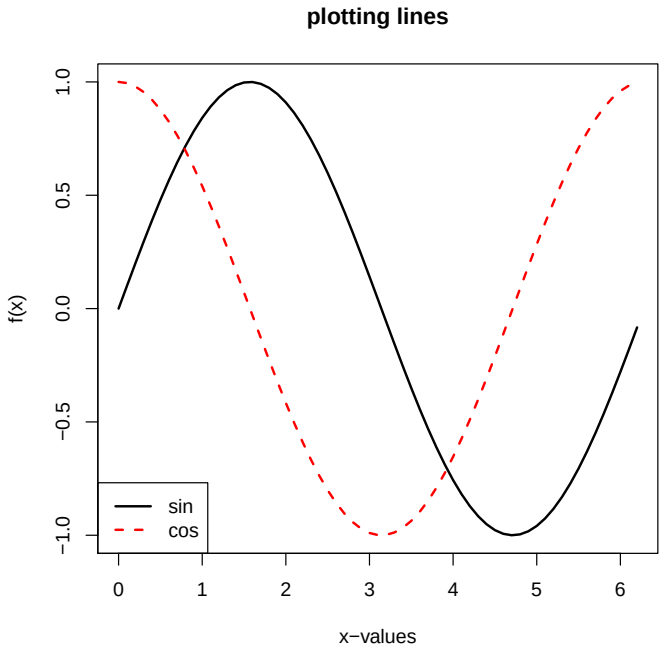


Figure 4: Simple figure with `plot` - see text for R-code

3.2. A lines plot

Often we prefer to display lines rather than dots; in the following we plot a sine function (figure 4). `type = "l"`[2] changes the type from `points` to a `line`, while `lwd=2` makes the line twice as thick. Another commonly used type is "b" (`both` lines and points).

We change the x- and y-axis labels (`xlab, ylab`), and add a `main` title

We add an extra line, change the `line type` (`lty = 2`, and the default color (`col = "red"`).

We add a `legend`, explaining what is on the graph:

```
>a <- seq (0, 2*pi, by = 0.1)
>plot (x = a, y = sin(a), type = "l", lwd = 2,
+   xlab = "x-values", ylab = "f(x)", main = "plotting lines")
>lines (x = a, y = cos(a), lty = 2, lwd = 2, col = "red")
>legend("bottomleft", legend = c("sin", "cos"), lty = c(1, 2),
+   lwd = 2, col = c("black", "red"))
```

[2] "l" is the lowercase of L , not the number one

4. X-Y plots; conditional plotting

As a more sophisticated demonstration of the use of symbols in R-figures, we work on a biological example, from the R data set called "Orange". This data set contains the circumference (in mm, at breast height) measured at different ages for five orange trees. We start by looking at the data; only the first (`head`) and last (`tail`) part is displayed.

```
>head(Orange)
```

```
  Tree  age circumference
1    1  118            30
2    1  484            58
3    1  664            87
4    1 1004           115
5    1 1231           120
6    1 1372           142
```

```
>tail(Orange)
```

```
   Tree  age circumference
30    5  484            49
31    5  664            81
32    5 1004           125
33    5 1231           142
34    5 1372           174
35    5 1582           177
```

and make a rough plot of circumference versus age:

```
>plot(x = Orange$age, y = Orange$circumference, xlab = "age, days",
+      ylab = "circumference, mm", main = "Orange tree growth")
```

As Orange is a `data.frame`, columns can be addressed by their names (`Orange$age` and `Orange$circumference`).

The output (figure) shows that there is a lot of scatter, which is due to the fact that the five trees did not grow at the same rate (figure 5).

It is instructive to plot the relationship between circumference and age differently for each tree. In R, this is simple: we can make the graphical parameters (symbol types, colors, size,...) conditional to certain "factors".

Factors play a very important part in the statistical applications of R. For the graphical applications, it suffices to know that the factors are integers, starting from 1.

In the R-statement below, we simply use different symbols (`pch`) and colors (`col`) for each tree

- `pch=(15:20)[Orange$Tree]` means that, depending on the value of `Orange$Tree` (i.e. the tree number), the symbol (`pch`) will take on the value 15 (tree=1), 16 (tree=2),... 20 (tree=5).

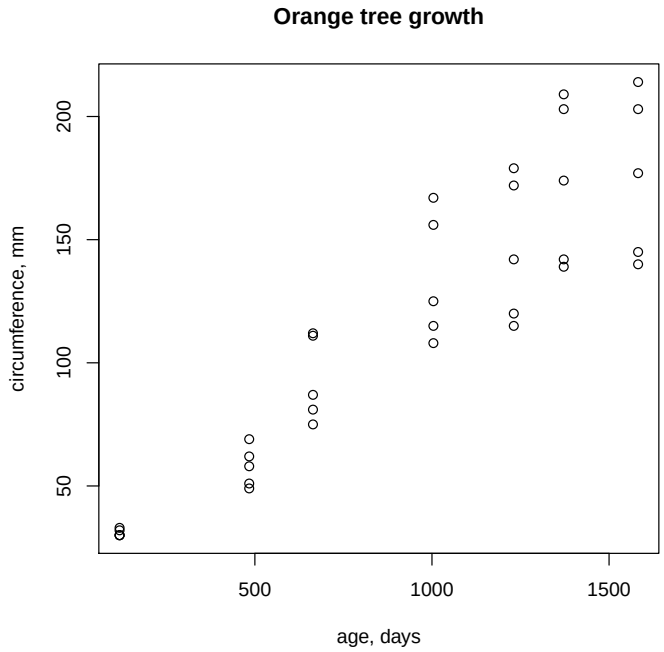


Figure 5: Simple plot of the `orange` dataset - see text for R-code

- `col=(1:5) [Orange$Tree]` does the same for the symbol's color.

The final statement adds a `legend`, positioned at the bottom, on the right side.

```
>plot(x = Orange$age, y = Orange$circumference, xlab = "age, days",
+    ylab = "circumference, mm", main= "Orange tree growth", cex = 1.3,
+    pch = (15:20)[Orange$Tree],
+    col = (1 :5) [Orange$Tree])
>legend("bottomright", pch = 15:20, col = 1:5, legend = 1:5, title = "tree")
```

The output shows that tree number 5 grows fastest, tree number 1 is slowest growing (figure 6). (note: it is also instructive to run the examples in the Orange help file.)

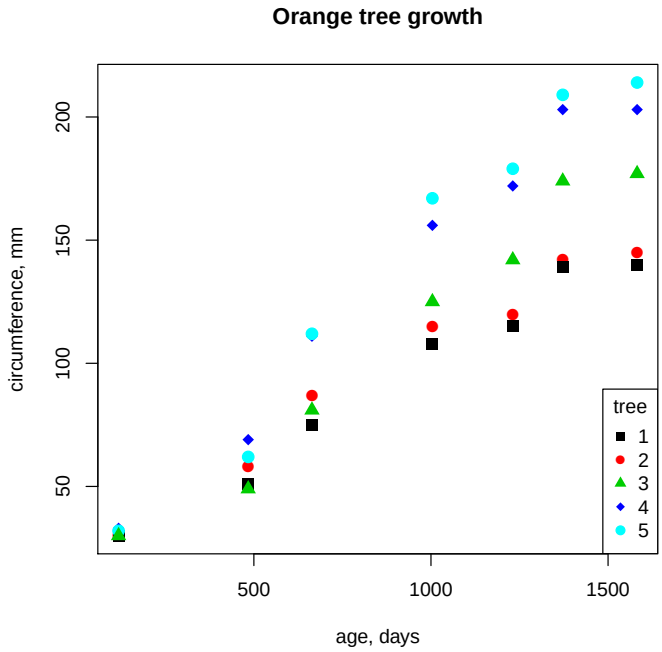


Figure 6: Simple figure with `plot`; using conditional plotting - see text for R-code

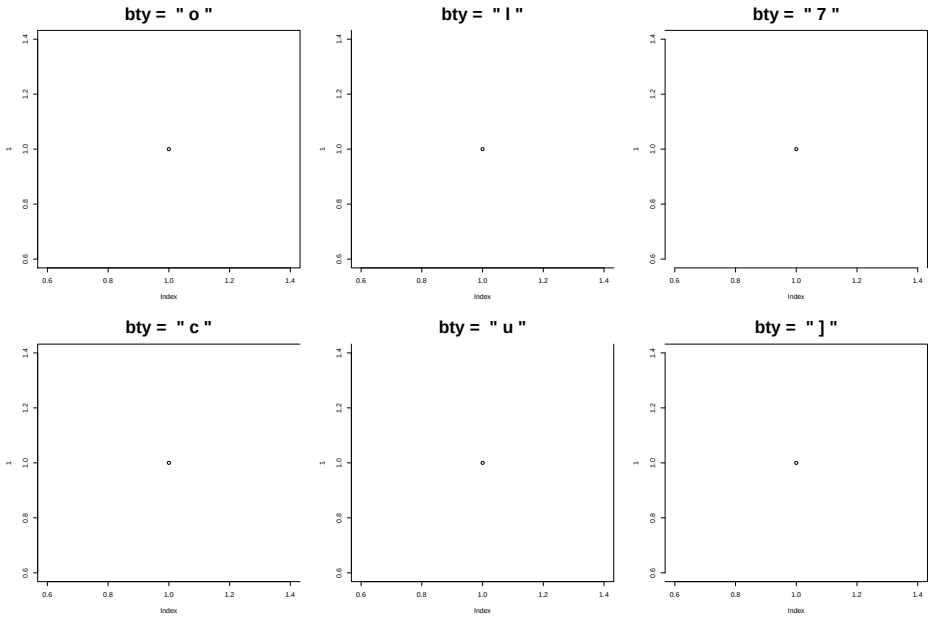


Figure 7: The types of boxes, surrounding a plot.

5. Figure axes

5.1. framing the plot

By default, R draws a box around a figure. In case you do not like that, it can be changed with the following settings (figure 7):

- `bty`: determines the type of box to be drawn; the default is "o", which will draw a full box; "n" which will draw NO box, "l" in which case the box will resembe "L", "7", "c", "u", or "]". (see figure).
- by setting `frame.plot=F`, the box is not drawn.

In the following code, 10 uniform random numbers are plotted. First, using the default plot, then by removing the "frame", next using boxtype "u" and finally with the axes positioned on top (margin 3) and on right (margin 4) (figure 8). For the latter, first the plot is created without axes, and without framing the plot; then an x-axis is positioned in margin 3, and an y-axis in margin 4.

```
> par (mfrow = c(2, 2))
> plot(runif(10), main = "default")
> plot(runif(10), frame.plot = FALSE, main = "frame.plot=FALSE")
> plot(runif(10), bty = "u", main = "bty='u'")
>#
> plot(runif(10), axes = FALSE, frame.plot = FALSE, xlab="", ylab="",
+    main = "axis command")
```

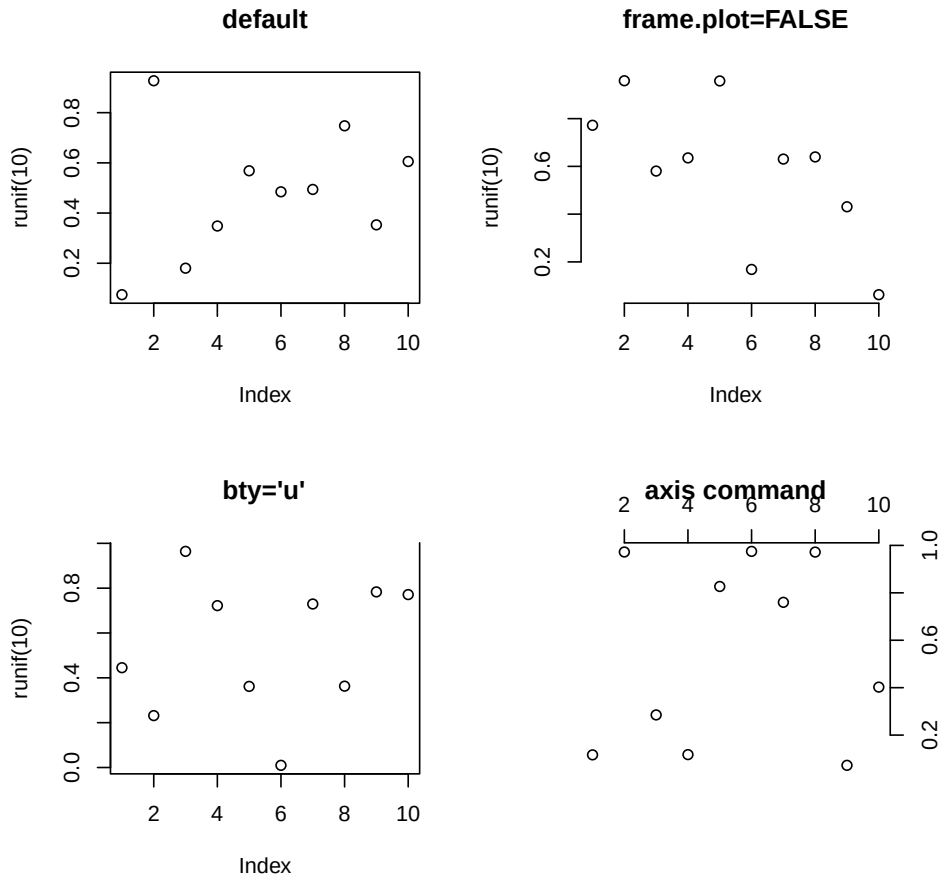


Figure 8: Figure without frame - see text for R-code

```
> axis(3)
> axis(4)
```

5.2. multiple axes

Often we will want to use multiple axes in a plot, for instance if we have two sets of data that we want to plot against two y-axes.

In the next example, we plot two data sets versus distance (figure 9). First the data are inputted (here simple sine and cosine functions).

```
>Data <- data.frame(Dist = 1 : 10, Set1 = sin(1:10), Set2 = cos(1:10))
>head(Data)
```

```
  Dist       Set1       Set2
1    1  0.8414710  0.5403023
2    2  0.9092974 -0.4161468
```

```
3    3  0.1411200 -0.9899925
4    4 -0.7568025 -0.6536436
5    5 -0.9589243  0.2836622
6    6 -0.2794155  0.9601703
```

`Set1` will be plotted against the y-axis on the left; `Set2` against the y-axis positioned on the right.

By default, the amount of space at the right side of the graph (margin 4) is not sufficient to allow an y-axis there. We increase the size of the fourth margin by 2 lines; margin sizes are set with parameter `mar`.

We check the margin size before we adapt it:

```
>(pm <- par ("mar"))

[1] 5.1 4.1 4.1 2.1

>par ("mar" = pm + c(0,0,0,2))
>par ("mar")

[1] 5.1 4.1 4.1 4.1
```

First `Set1` is plotted.

```
attach(Data)
plot(Dist, Set1, pch = 16, cex=1.2, col = "darkblue", xlab = "Dist",
  ylab = "Set1", main = "Two axes")
```

Now we start a new figure, but without overwriting the previous one; `par(new = TRUE)` does that.

We then add the second data set, now both lines and symbols (`type = "b"`), but make sure that axes are not plotted (`axes = FALSE`).

We end by detaching the data set.

```
par(new = TRUE)
plot(Dist, Set2, pch = 18, cex = 1.2, type = "b", col = "darkred",
  xlab = "", ylab = "", axes = FALSE)
```

The axis on margin 4 is added later, using the `axis` command.

```
axis(side = 4)
mtext(side = 4, text = "Set2", line = 2)
```

We end by detaching the data:

```
detach(Data)
```

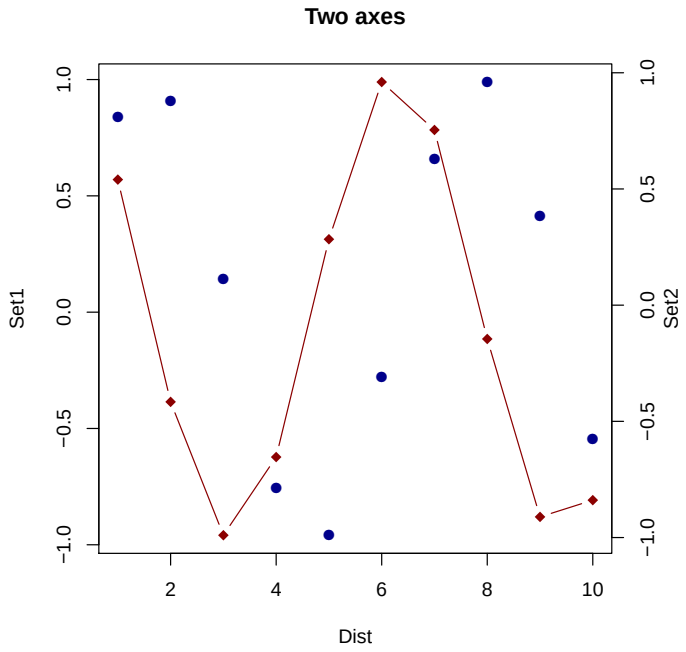


Figure 9: A figure with two y-axes

5.3. Improving the axes labels

Now assume that the units of a data set are in m^2 and $\mu g\ m^{-2}$ for the x- and y-axis respectively. To represent that, we need to write a mathematical `expression` (figure 10).

See `demo(plotmath)` for how to do this.

```
>plot(x = 1:5, y = runif(5), xlab = expression(m^2),
+   ylab = expression(mu*g~m^-2),
+   main = "using expressions")
>text(3, 0.5, labels = expression(alpha[1]==sum(x[i], i=1, n)), cex=2)
```

5.4. Axes styles, specifying axes ranges

By default R first extends the axis by 4% at each end, before drawing the axes. If not specified, R will find an axis with pretty labels.

Parameters `xaxs` and `yaxs` allow to specify *where* exactly R puts the axes.

The axes *ranges* can be specified with `xlim`, `ylim`. For instance:

```
>plot(x= 1:5, y = runif(5), xlim = c(1, 5), ylim = c(1, 0),
+   xaxs = "i", yaxs = "r", main = "axes styles")
```

will use the default, 4% extended version for the y-axis (`yaxs = "r"`) (the default), but the x-axis will exactly fit the range (`xaxs = "i"`).

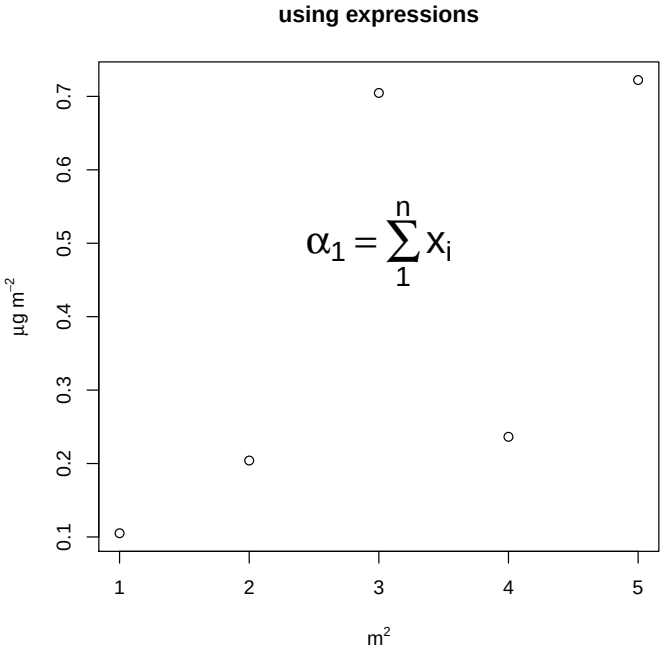


Figure 10: Using expressions to represent mathematics notations - see text for R-code

The x- and y-ranges are set to (1, 5) and (1, 0) respectively. Note that the y-axis extends from 1 to 0 (`ylim = c(1,0)`) and thus the axis has been reversed (figure 11).

5.5. Specifying tick labels

Often, the default tick marks positioned by R are not desirable.

The `axis` command allows to specify where the tick marks are to be positioned AND how to label them; command `format` allows formatting of numbers (figure 12):

```
>plot(1:5, 100*(0:4), axes = FALSE, xlab = "", ylab = "",
+   main = "the axis commmand")
>axis(1, at = 1:5, labels = c("j", "f", "m", "a", "m"))
>ticks <- c(1, 100, 200, 400)
>axis(2, at = ticks, labels = format(ticks, scientific = TRUE))
```

5.6. The style of tick labels

By default, R writes the labels always parallel to the axis.

The parameter `las` can override this default, i.e. make the labels always parallel (`las = 0`), always horizontal (`las = 1`), perpendicular (`las = 2`) or always vertical (`las = 3`).

The parameter can also be used to change the rotation of text written in the margin (figure 13).

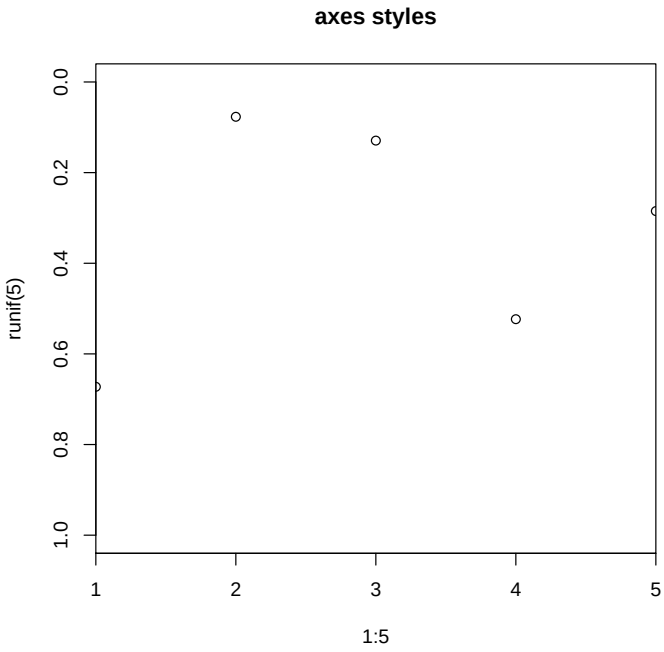


Figure 11: Using xaxs and yaxs to specify the axes styles; y-axis is the default - see text for R-code

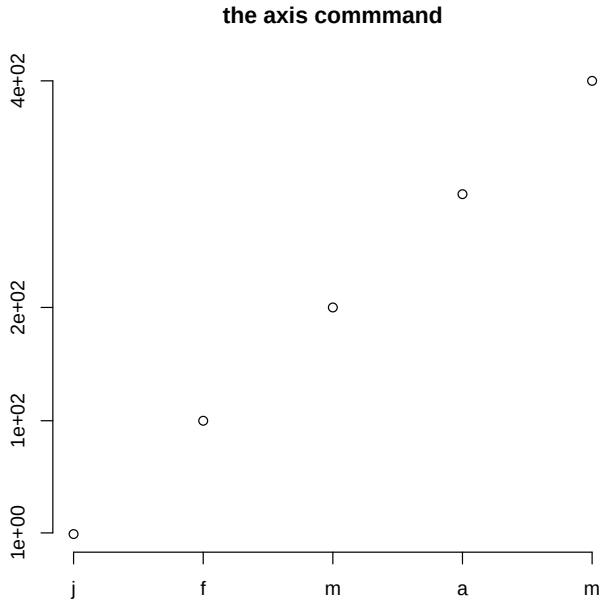


Figure 12: Using the axis command to position the tick marks - see text for R-code

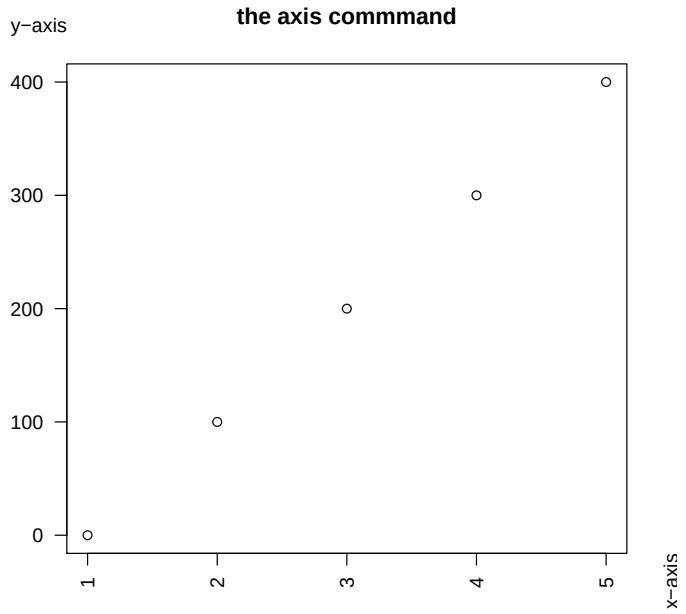


Figure 13: Using the parameter `las` command to change the style of axis labels - see text for R-code

```
>plot(1:5, 100*(0:4), axes = FALSE, xlab = "", ylab = "",  frame.plot = TRUE,
+   main = "the axis commmand")
>axis(side = 1, las = 3) # vertical
>axis(side = 2, las = 1) # horizontical
>mtext(side = 2, at = 450, las = 1, "y-axis")
>mtext(side = 1, at = 5.5, las = 3, "x-axis")
```

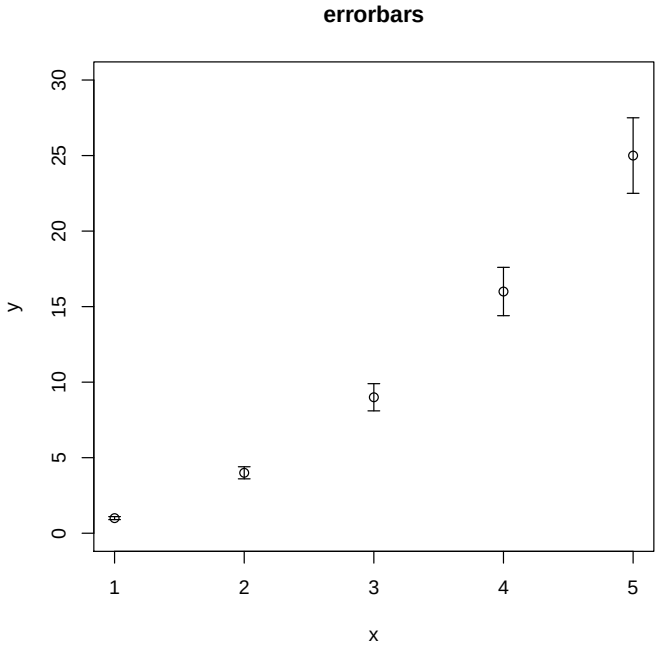


Figure 14: Adding errorbars to a plot - see text for R-code

6. Miscellaneous

6.1. error bars

Strangely enough, there is no default function to plot error bars. So, one has to be creative to do so.

The easiest way to produce error bars is by using the `arrows` command.

Thus, we draw an arrow from *mean - SE* to *mean + SE*, and setting the angle of the shaft equal to 90 degrees.

The "arrow" function has as input `(x0, y0, x1, y1)` for an arrow extending from `(x0, y0)` to `(x1, y1)`. We also specify the length of the shaft; `code = 3` draws the arrow heads at both ends (figure 14).

```
>x <- 1:5
>y <- x^2
>SE <- y*0.1
>plot(x, y, main = "errorbars", ylim = c(0,30))
>arrows(x, y+SE, x, y-SE, angle = 90, length = 0.05, code = 3)
```

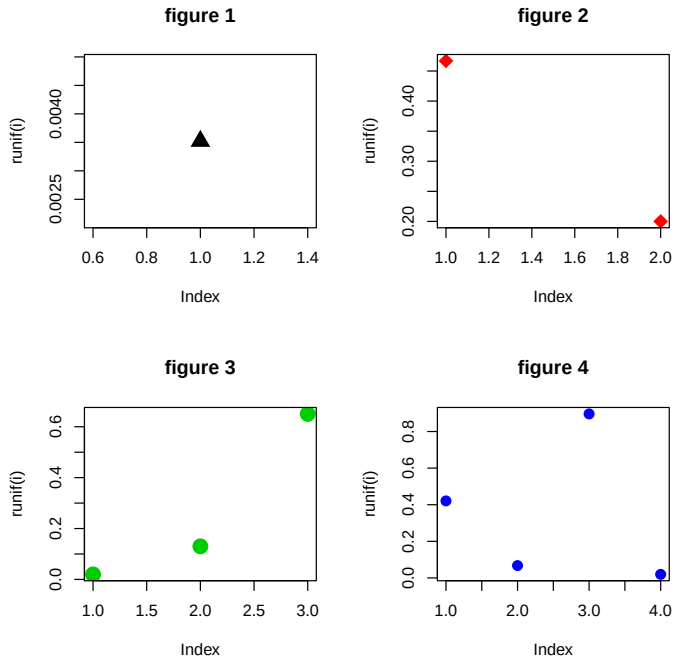


Figure 15: Multiple figures in a plot, using function `mfrow` - see text for R-code

7. Multiple figures

There are several ways in which to arrange multiple figures on a plot.

7.1. multiple figures on a row or column

The simplest way to create multiple figures is by specifying the number of figures on a row (`mfrow`) and on a column (`mfcol`):

```
> par(mfrow = c(3, 2))
```

will arrange the next plots in 3 rows, 2 columns. Graphs will be plotted row-wise.

```
> par(mfcol = c(3, 2))
```

will arrange the plots in 3 columns, 2 rows, in a columnwise sequence. Note that both mfrow and mfcol must be inputted as a vector.

Try (figure 15):

```
>par(mfrow = c(2, 2))
>for ( i in 1:4)
+   plot(runif(i), col = i, pch = 16+i, main = paste("figure",i), cex = 2)
>par(mfrow = c(1, 1))
```

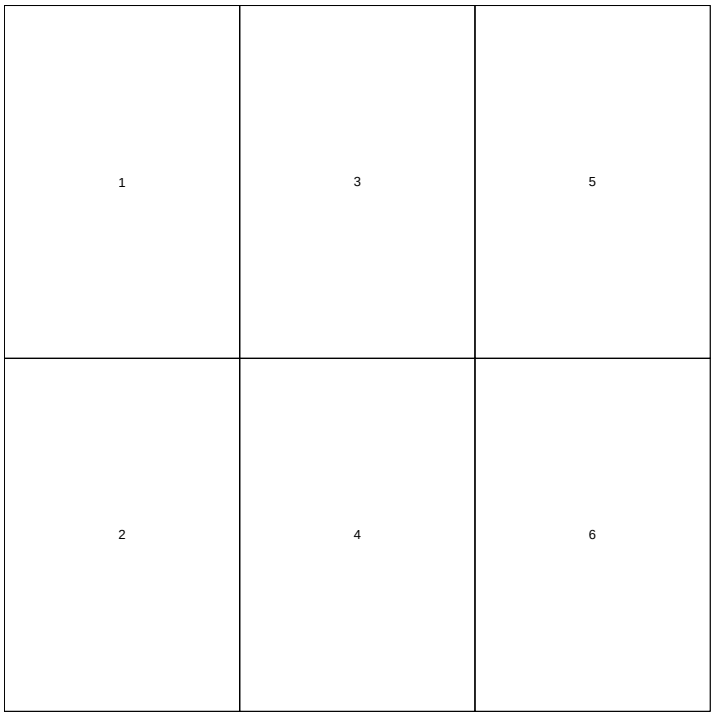


Figure 16: 6 figures on two rows, using layout - see text for R-code

7.2. more complex layouts

R-function `layout` allows much more complex plot arrangements. It is much more difficult to use, but more flexible. Try:

```
> ? layout
```

In the following, we first create a layout of 6 equally-sized figures, in 2 rows and three columns (the same can be done with `mfrow`). `layout.show` plots the layout (figure 16)

```
>mat2 <- matrix(nrow = 2, ncol = 3, data = 1:6)

>nf <- layout(mat2, respect = FALSE)
>layout.show(nf)
```

In the next script, the graphics window is divided in three columns and two rows (of equal size). Then we merge several boxes: the first two on row 1 are merged; the third box on row one is not used (0); the last two boxes on row 2 are also merged (figure 17).

```
>layoutmat <- matrix(data = c(1, 1, 0, 2, 3, 3), nrow = 2, ncol = 3,
+                     byrow = TRUE)
>layoutmat

     [,1] [,2] [,3]
[1,]    1    1    0
[2,]    2    3    3
```

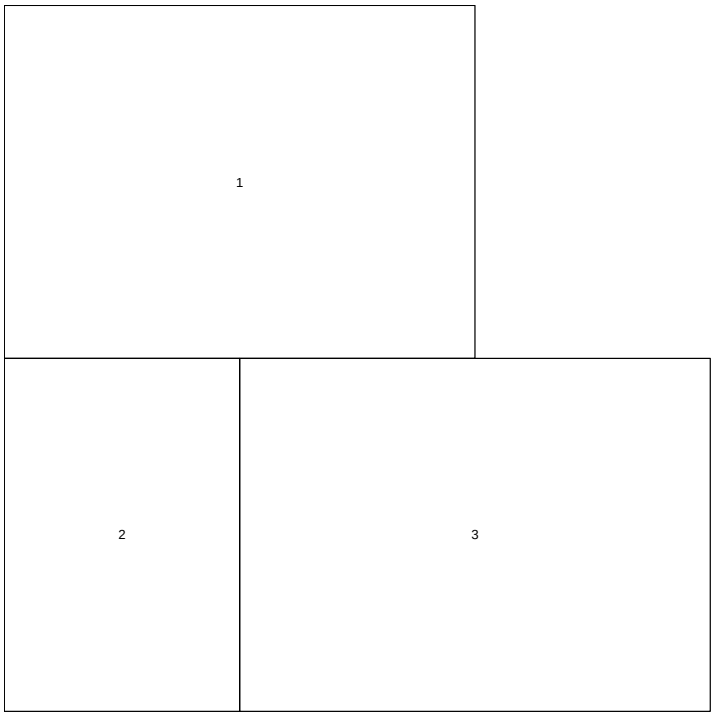


Figure 17: The layout of a more complex composition of figures - see text for R-code

We show the layout:

```
>nf <- layout(layoutmat, respect=FALSE)
>layout.show(nf)
```

7.3. labelling multiple figures

There is no standard way (that I know of) to write a label next to each figure. The easiest way to do this, is to use function `writelabel` from R -package `shape` (Soetaert 2009).

For instance, we now make plots, using the layout of previous section, each time, labelling it (figure 18). The first figure (`1`) will occupy the first two boxes in the first row; the last figure in the upper row will be unused. The second figure (`2`) occupies the first box in the second row, the third figure (`3`) the last two boxes in the second row.

```
>layout(matrix(data = c(1, 1, 0, 2, 3, 3), nrow = 2, ncol = 3,
+                    byrow = TRUE), respect=FALSE)
>require(shape)
>for (i in 1:3) {
+   plot(1, type = "n", main = paste ("figure", i))
+   writelabel (nr = i)
+ }
```

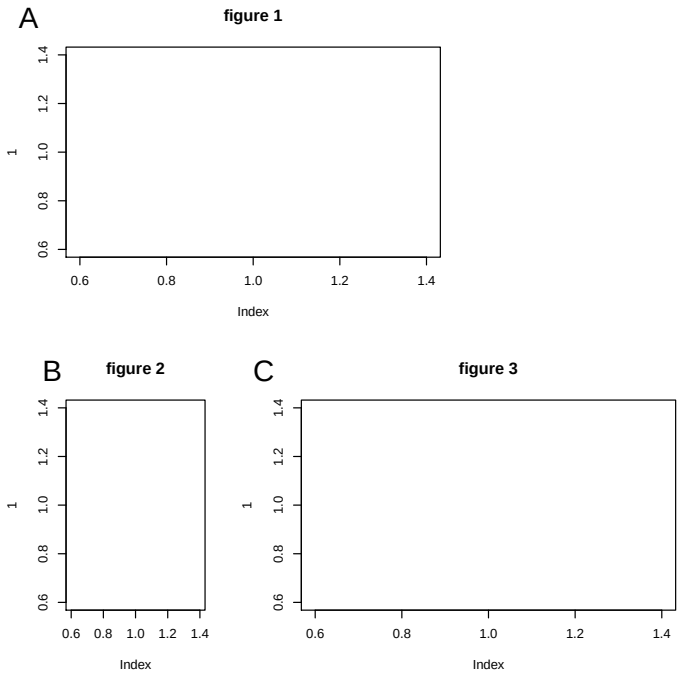


Figure 18: Complex figure layout - see text for R-code

7.4. a plot within a plot

If we want to draw a plot within an existing plot, we first open a new window *within* the existing window, giving the (relative) position of c(x1, x2, y1, y2) :

```
par (new = TRUE)
par (fig = c(0.1, 0.5, 0.1, 0.5))
```

For instance (figure 19):

```
>plot(1:5, 1:5, pch = 16, col = "blue", main = "Figure with inset")
># inset 1
>par(new = TRUE)
>par(fig = c(0.1, 0.6, 0.5, 0.95))
>plot(1:5, 5:1, pch = 18, col = "red", xlab = "", ylab = "",
+   main = "inset1", cex.axis = 0.7)
># restore original size
>par(fig = c(0, 1, 0, 1))
># inset 2
>par(new = TRUE)
>par(fig = c(0.45, 0.95, 0.1, 0.55))
>plot(1:5, runif(5), pch = 20, col = "green", xlab = "",ylab = "",
+   main = "inset2", cex.axis = 0.7)
>par(fig = c(0, 1, 0, 1))
```

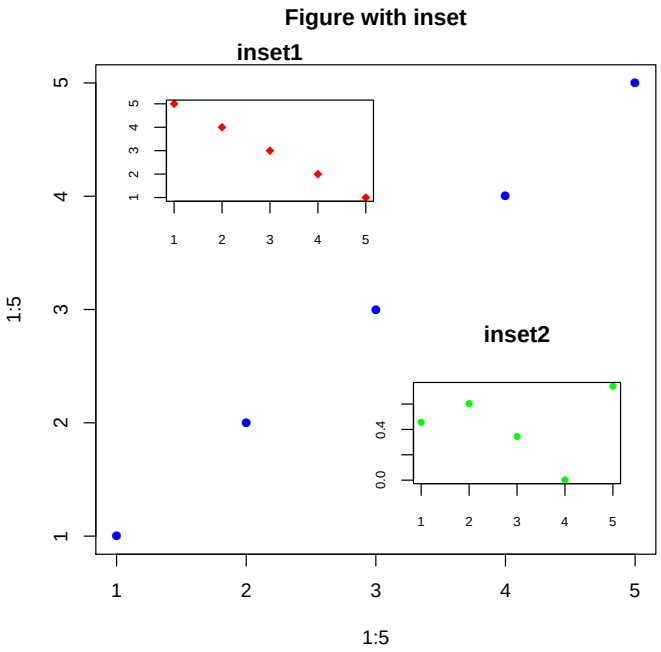


Figure 19: Figure with two subfigures - see text for R-code

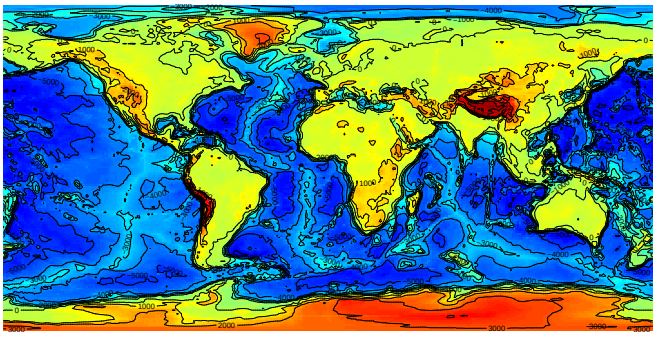


Figure 20: Image plot of ocean bathymetry - see text for R-code

8. Other types of plots

8.1. Images, contour plots, perspective plots

R has some very powerful functions to create images and add contours. For example, the data set `Bathymetry` from the **marelac** package can be used to generate the bathymetry (and hypsometry) of the world oceans (and land) (figure 20):

```
>image(Bathymetry$x, Bathymetry$y, Bathymetry$z, col = femmecol(100),
+       asp = TRUE, xlab = "", ylab = "", axes = FALSE)
>contour(Bathymetry$x, Bathymetry$y, Bathymetry$z, add = TRUE)
```

Note the use of "asp=TRUE", which maintains the aspect ratio.

In the next example, data set `volcano` is plotted as a perspective (3-D) plot (figure 21).

```
>persp(volcano, theta = 135, phi = 30, col = "grey",
+         ltheta = -120, shade = 0.75, border = NA, box = FALSE)
```

8.2. histograms, boxplots

The data set `morley` contains the results of a classical experiment of Michaelson and Morley on the speed of light. The data consists of five experiments, each consisting of 20 consecutive "runs". The response is the speed of light measurement.

We create 4 figures, arranged in two rows (figure 22):

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

Then, we plot a histogram of the given measurements:

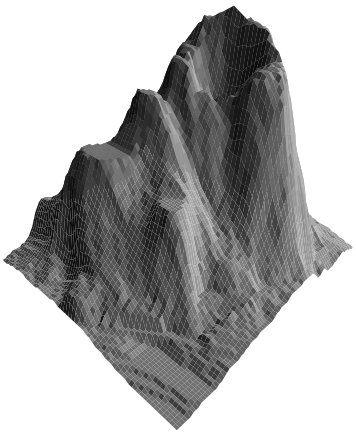


Figure 21: Perspective plot of a volcano - see text for R-code

```
>hist(morley$Speed, xlab = "speed of light",
+   main = "Michaelson-Morley")
```

This can also be visualised as a "density plot", or a Box-And-Whisker plot:

```
>plot(density(morley$Speed), xlab = "speed of light",
+   main = "Michaelson-Morley")
>boxplot(morley$Speed, ylab = "speed of light",
+   main = "Michaelson-Morley")
```

A box-and whisker plots are also convenient ways to see whether a certain treatment has an impact. To make such a Box-And-Whisker plot, we need to use the "formula" representation of R, e.g. `y~grp` means that y should be expressd as a function of the groups in x.

For instance, to visualise whether the experiment had an effect on the speed-of-light measuremente, we make a boxplot depicting the speed as a function of the experiment:

```
>boxplot(Speed ~ Expt, data = morley, ylab = "speed of light",
+      main = "Speed of Light", xlab = "Experiment No.")
```

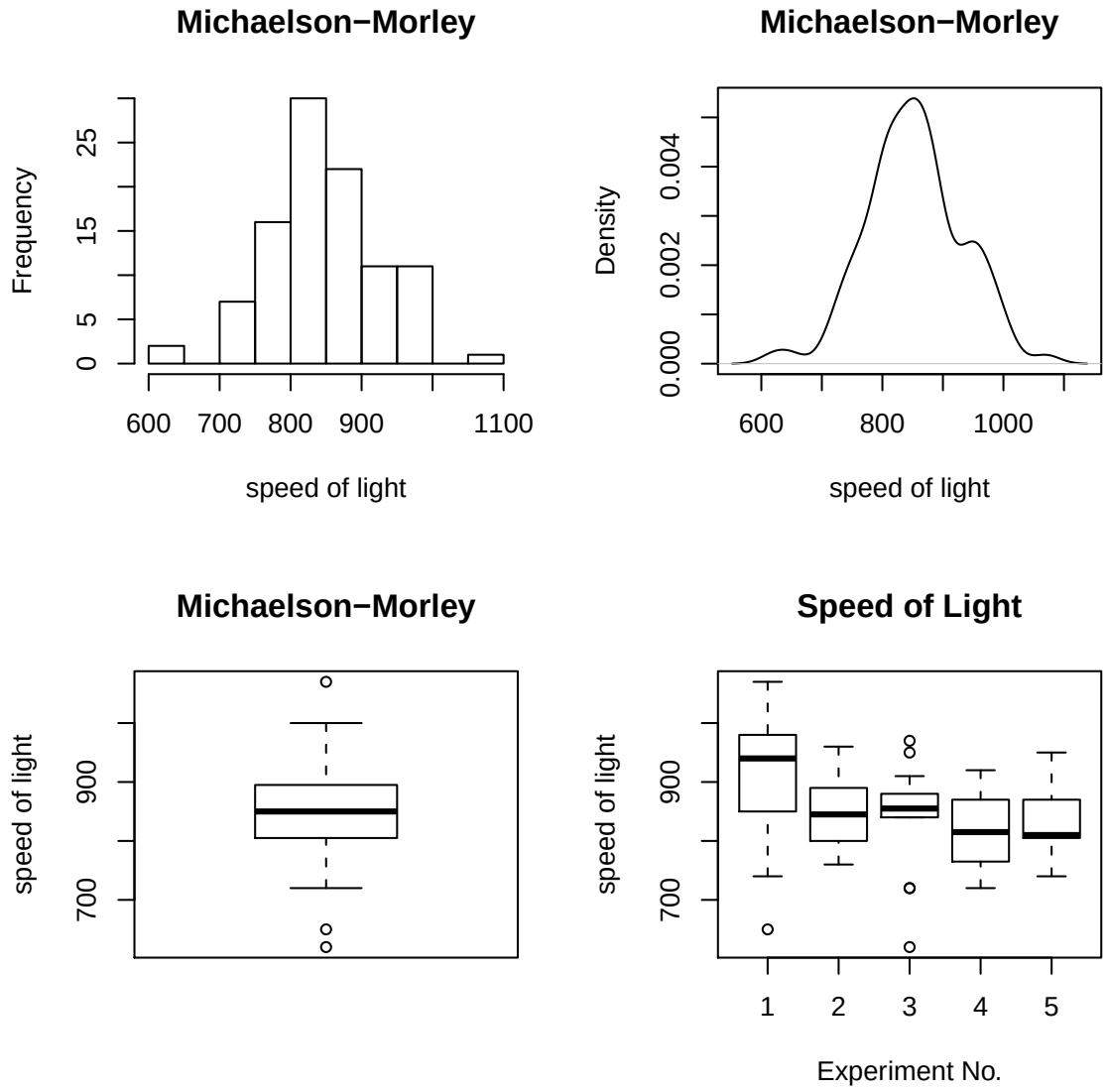


Figure 22: Four ways to look at the same data.

9. a publishable line plot

Scientific graphs are often composites, where plots that have equal axes are plotted next to each other. Rather than repeating all the axes for every plot, often axes inside the graph are left out, or only the axis and its ticks are given, but not the numbers. Also, sometimes two axes share a single label that is plotted in between them.

It is really simple to make composite graphs in R, but the default settings produce graphs that are too much spaced-out, and the graphs sometimes differ in dimensions.

In this section, we will discuss how to make composite graphs that are publishable. We will show how to align the plots closer to each other by adjusting the margins, how to make a plot function in which the properties that the plots have in common (e.g., line width, text size) are set, and how to use this function to plot the specific panels.

First we invent a dataset to be plotted in which the soil temperature and the growth of plants is described as a function of the month within two years. We generate this "data" using a simple sine wave:

```
>Month    <- 1:12
>TempY1   <- cos(Month/pi*1.8+3)*10 + 10
>TempY2   <- cos(Month/pi*1.8+2.4)*8 + 7
>GrowthY1 <- 0.1*TempY1^2
>GrowthY2 <- 0.1*TempY2^2
```

A straightforward plot of this data as a couple of simple graphs is:

```
>par(mfrow = c(2, 2))
>plot(x = Month, y = TempY1, main = "First year", type = "b")
>plot(x = Month, y = TempY2, main = "Second year", type = "b")
>plot(x = Month, y = GrowthY1, main = "First year", type = "b")
>plot(x = Month, y = GrowthY2, main = "Second year", type = "b")
```

The resulting graphs looks horrible (figure 23), and are not suitable for publication. By adding `ylab = "Temperature"` , this looks a little better, but still not very professional.

There are many things not optimal in this graph:

- The figures are too far apart
- The main title is repeated too many times
- Axes are repeated
- The Y-axis label gives the variable name rather than the plotted parameter and its unit.

To improve the first point, we start a new graph and set the margins somewhat tighter:

```
windows(width = 7, height = 6)
nf <- layout(matrix (c(1, 2, 3, 4), nrow = 2, ncol = 2, byrow = TRUE),
             widths  =lcm(c(7, 7, 7, 7)),
             heights =lcm(c(6, 6, 6, 6)) )
```

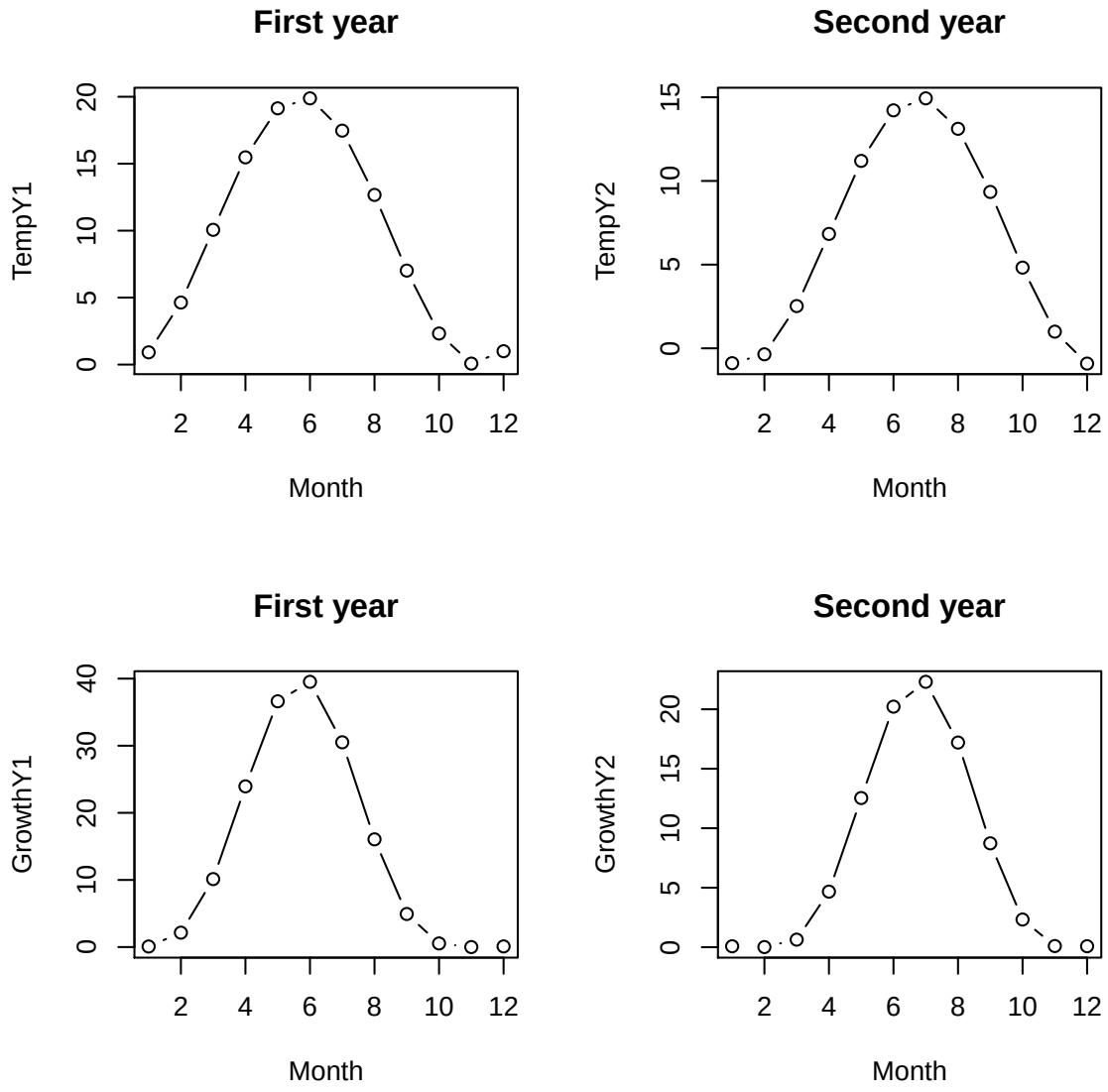


Figure 23: Quick and dirty plotting of 4 data sets, not very pretty - see text for R-code

We then set up a function that generates a plot and sets the properties that all the plots have in common:

```
PlotPanel <- function (Date, Mean, SE, ...) {
          par(mar=c(1, 1, 1, 0.5))
          plot(Date, Mean, type="b", frame.plot=F,
               cex.main=1.2,    # Size of the title
               ...)
 }
```

Now we call this plotting function, providing the data and adding specific properties that set the specific characteristics for each graph (you can leave out anything behind a # symbol).

```
PlotPanel(Month, TempY1,
          pch = 1,          # The symbol type, given by its number
          xaxt = "n",       # The X-axis is left out
          main = "2007",    # The plot title is given
          ylim = c(-2, 25)  # The limits of the Y-axis are set
          )

PlotPanel(Month, TempY2,
          pch = 1,
          axes = F,         # No axes are plotted
          main = "2008",
          ylim = c(-2, 25)
          )

PlotPanel(Month, GrowthY1,
          pch=17, xlab = "Month", ylim = c(0, 40) )
```

We plot the Y-axis label in a separate `mtext` call

```
mtext(text = "Temperature (C)",
          side = 2,   # 2 = left axis
          line = 2.5, # determines position in x direction
          adj = 2    # determines position in x direction
)

PlotPanel(Month, GrowthY2,
          pch = 17, yaxt = "n", ylim = c(0, 40), xlab = "Month" )

mtext(text = "Time of the year (Month)", side = 1, line = 2.5, adj = -1.1)
```

The result is much better (figure 24)

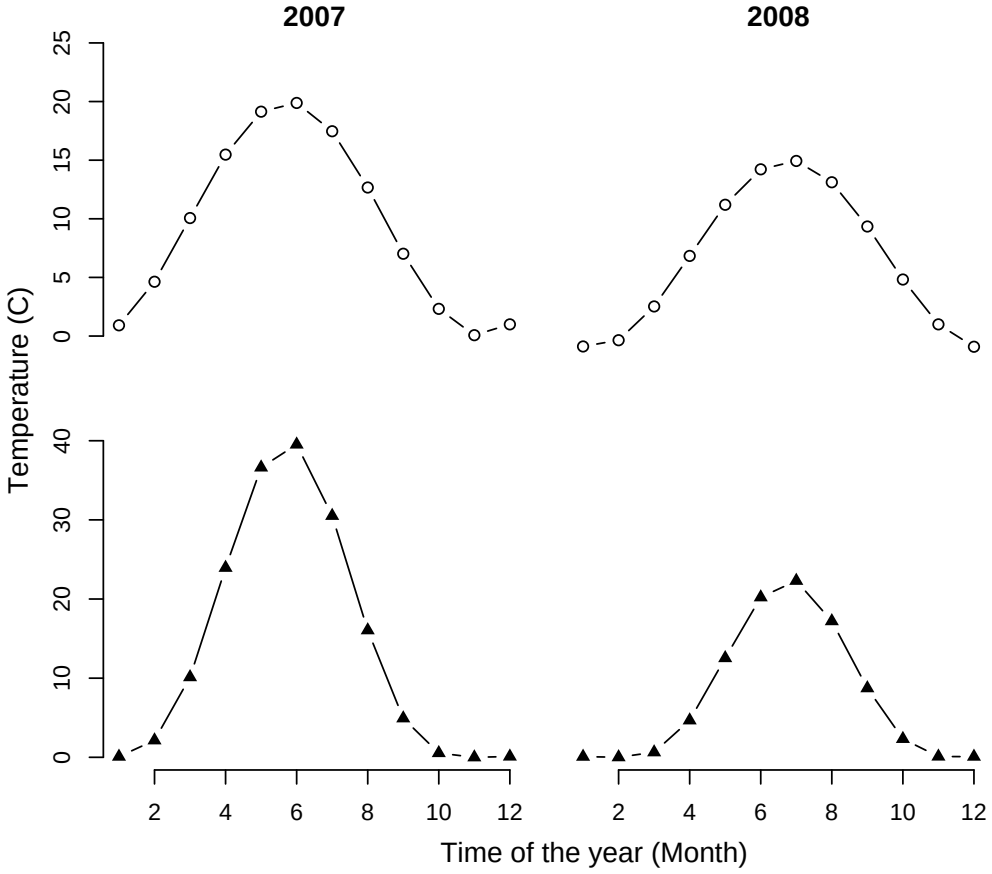


Figure 24: The four plots, improved - see text for R-code

10. A publishable barplot

The next example is considerably more complex; we make a `bar` plot of the chlorophyll a content in two sites, called SiteA and SiteB.

First we input the data, consisting of the mean, and the standard error of Chloropyll a (`chla`) For each set, we also calculate the mean + standard error

```
>chla      <- c(SiteA = 57.1, SiteB = 20.0)
>chlaSE    <- c(8.8, 1.8)
>chla_plus <- chlaSE + chla
```

Then we set the graphical parameters, i.e. the sizes of labels, numbers, titles, and significances, as well as the colours that we will use to fill the bar plots:

```
>LabelSize   <- 1
>NumberSize  <- 1
>NumberTitle <- 1
>NameSize    <- 1
>SigSize     <- 0.8
>SymbolSize  <- 1.2
>TitleSize   <- 1.0
>fillcolours <- c("grey20", "grey60")
```

After opening the window, we create the barplot (figure 25)

```
windows()
barplot_chla <-
  barplot(chla, beside = TRUE, col = fillcolours, ylim = c(0, 80),
  space = c(0.1, 0.2), mgp = c(2.2,1,0), cex.lab = LabelSize,
  cex.names = NameSize, cex.axis = NumberSize, cex.main = NumberTitle,
  ylab = expression(Chlorophyll-a~content~(mu*g~~g^-1)))
```

Use `?barplot` to see what the different parameters mean.

Note the use of an `expression` to create professionally-looking y labels. In this, the $\sim$ inserts a space.

Function `barplot` returns the position along the x-axis of the two bars. We will use that when we add the SE-bars.

Error bars are added using the "arrow" function:

```
arrows(barplot_chla, chla_plus, barplot_chla, chla, angle=90,
  length = 0.05, code = 3)
```

We next add the significance level and a legend. By setting `font=3` we choose the italics font.

```
text(x = 1.8, y = 30.48, labels = "p<0.01", cex = SigSize, font = 3)

legend("topright", c("SiteA", "SiteB"), col = fillcolours, pch = c(15, 15),
  bty = "n", pt.cex = SymbolSize, cex = NumberSize)
```

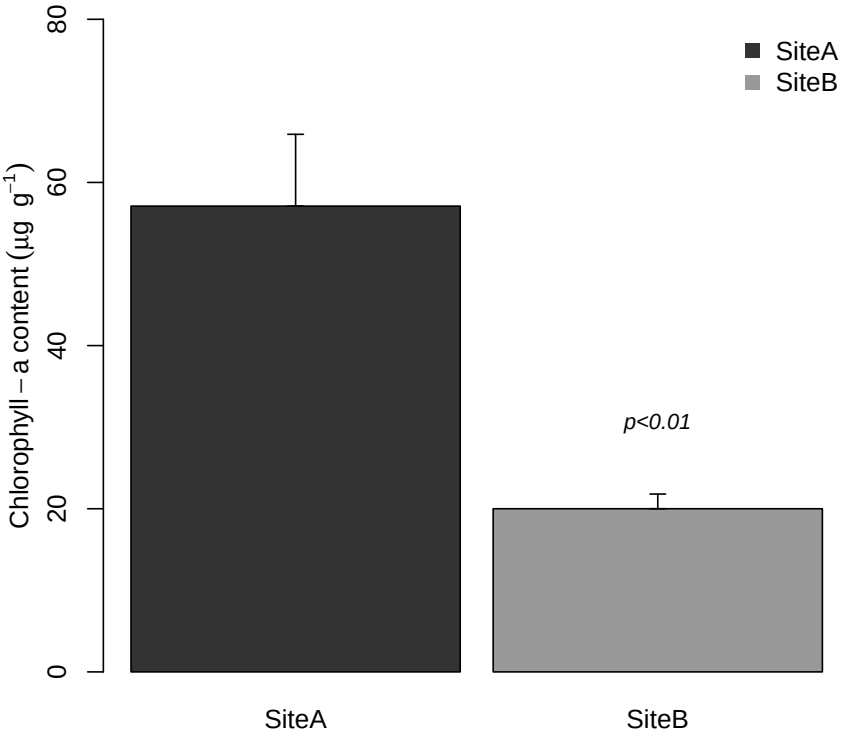


Figure 25: A bar plot

Two barplots

Now we create TWO barplots, aligned in one row (figure 26). The first barplot contains the Chloropyll data as before

The second one contains another variable, (`VAR2`), in kPa, whose data are inputted as:

```
>VAR2   <- c(SiteA = 27.4, SiteB=7.5)
>VAR2SE <- c(3.58, 1.10)
>VAR2_plus <- VAR2SE + VAR2
```

How to plot ONE `barplot` was explained in previous section.

As we need to produce two very similar graphs, it is advisable to create a *function* to do so. This function should receive, as arguments the mean values and standard errors, and the significance.

It should produce ONE barplot (`barplot`), with added standard errors (`arrows`), a legend, and the label of significance at the correct position (`text`). Here we put this label 10% of the y-axis size above the mean value of the second site; `diff(lim)` calculates the size of the y-axis.

Note the use of `...`, both in the function header, and in function `barplot`; Using `...` allows to pass ANY argument that is used in function `barplot`.

```
PlotBar <- function (Mean, SE, sign, ylim, ...) {

  d_plus <- Mean + SE
  barplot_Dat <-
    barplot(Mean, beside = TRUE, col = fillcolours, ylim = ylim,
    space = c(0.1, 0.2), mgp = c(2.2, 1, 0), cex.lab = LabelSize,
    cex.names = NameSize, cex.axis = NumberSize,
    cex.main = NumberTitle, ...)

  arrows(barplot_Dat, d_plus, barplot_Dat, Mean, angle = 90,
    length = 0.05, code = 3)

  text(barplot_Dat[2,1], d_plus[2]+diff(ylim)/10, labels = sign,
    cex = SigSize, font = 3)

  legend("topright", c("SiteA", "SiteB"), col = fillcolours,
    pch = c(15, 15), bty = "n", pt.cex = SymbolSize, cex = NumberSize)

}
```

We then specify that the plots are to be set in two columns, one row:

```
par(mfrow=c(1,2))
```

After entering the first data set, function `PlotBar` is called to plot it; note how we also pass the y-label (`ylab`), although this was not a formal argument in function `PlotBar`; it makes use of the `...` argument.

```
chla      <- c(SiteA = 57.06890, SiteB = 20.70003)
chlaSE    <- c(8.784, 1.789)
PlotBar(chla, chlaSE,  sign = "p<0.01", ylim = c(0, 80),
  ylab = expression(Chlorophyll-a~content~(mu*g~~g^-1)))
```

The second data set is plotted in a similar way:

```
VAR2   <- c(SiteA = 27.407, SiteB = 7.498)
VAR2SE <- c(3.577, 1.101)

PlotBar(VAR2, VAR2SE, sign = "p<0.05", ylim = c(0, 40),
  ylab = expression(Variable~2~(kPA)))
```

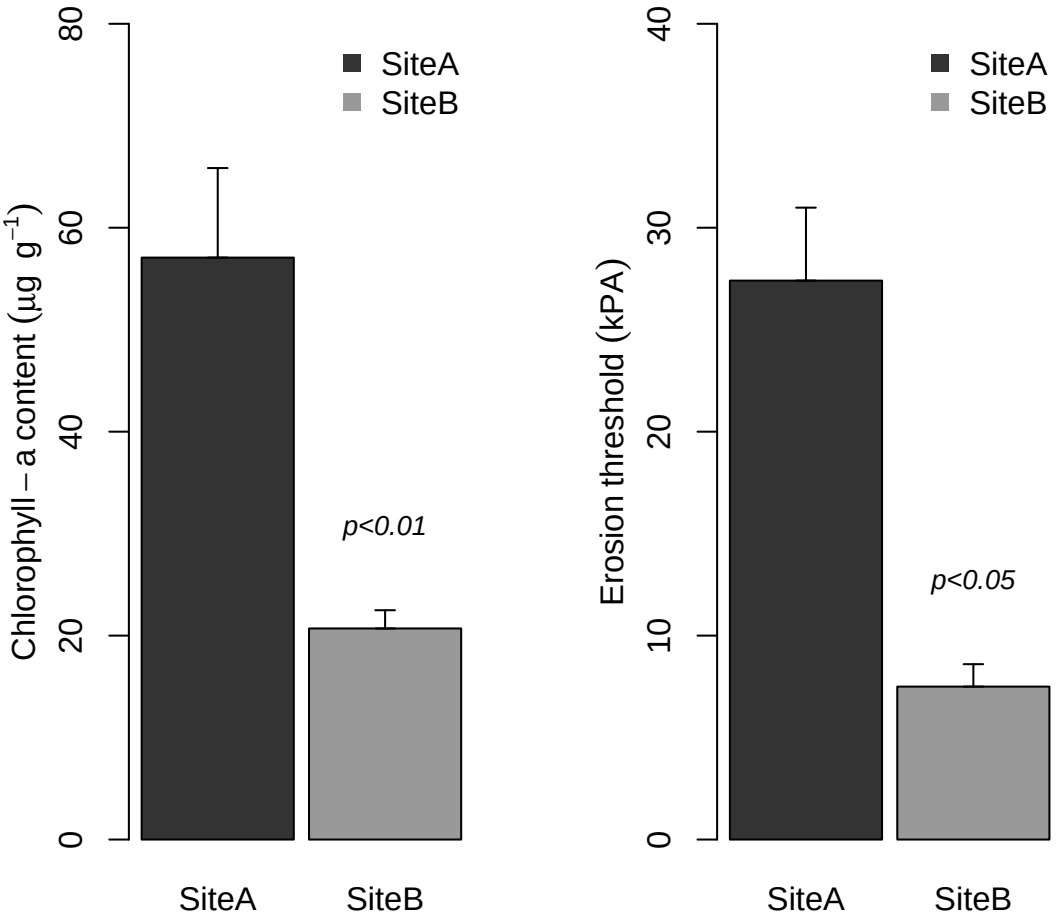


Figure 26: Two bar plots, with error bars.

11. Finally

This document has been generated with LaTeX and making use of R-package **Sweave** (Leisch 2002), which allows to merge LaTeX with R-code.

The notes were made for a workshop I gave together with Johan van de Koppel and Ellen Weerman to teach our NIOO-colleagues how to use R for making graphs.

Johan and Ellen contributed in the sections "A publishable lineplot" and "A publishable barplot".

References

Leisch F (2002). "Sweave: Dynamic Generation of Statistical Reports Using Literate Data Analysis." In W Härdle, B Rönz (eds.), "Compstat 2002 - Proceedings in Computational Statistics," pp. 575–580. Physica Verlag, Heidelberg. ISBN 3-7908-1517-9, URL http://www.stat.uni-muenchen.de/~leisch/Sweave.

R Development Core Team (2010). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org.

Soetaert K (2009). *shape: Functions for plotting graphical shapes, colors*. R package version 1.3.1.

Soetaert K, Meysman F (2009). *marelacTeaching: Datasets and tutorials for use in the MArine, Riverine, Estuarine, LAcustrine and Coastal sciences*. R package version 1.1.

Affiliation:

Karline Soetaert
Centre for Estuarine and Marine Ecology (CEME)
Netherlands Institute of Ecology (NIOO)
4401 NT Yerseke, Netherlands E-mail: k.soetaert@nioo.knaw.nl
URL: http://www.nioo.knaw.nl/users/ksoetaert