Advanced Graphics with R

Paul Murrell
Universitat de Barcelona

April 30 2009

Session overview:

(i) Introduction

- Graphic formats: Overview and creating graphics in R
- Graphical parameters in R: `par()`
- Selected graphical functions

(ii) Graphical packages

- `'rgl'`, `'grid'` and `'lattice'`

(iii) Miscellanea

- Useful functions in R (`legend()`, `polygon()`, annotations, etc.)
- Selected examples
- Creating animations in R

1 Introduction

Graphics formats: Overview and creating graphics in R

Starting point: The built-in R graphics device(i.e. `win.graph()` on Windows platforms). Most comments here apply to Windows, but everything also works on Mac OS and Linux (eventually some modifications are required). The participants are supposed to be familiar with the basics of R (and fundamentals of statistics).

An overview of graphics formats:

(i) Bitmaps ("pixel-based"):

- uncompressed (e.g. *.bmp*, *.tif*
- compressed:
  - lossless (e.g. *.png*, partly *.tif*
  - lossy (e.g. *.jpg*

(ii) Vector based (e.g. *.(e)ps*, *.emf*, *.pdf*)

The choice of graphics formats depends on the purpose. (E.g. Word/Powerpoint and like: *.emf*; LaTeX: *.ps* for LaTeX and *.pdf* or *.png* for PDFLaTeX.)

Alternative ways to create (i.e. save) graphs:

(i) Context menu (i.e. right-click and copy or save)

(ii) File menu

(iii) By R command ("batch-mode")

Some examples:

```
> # Creating a JPG-file
> jpeg("C:/test.jpg")
> hist(rnorm(1000))
> dev.off()
```

Functions for the other formats:

- `bmp()` / `png()`
- `postscript()`
- `pdf()`
- `win.metafile()`

The pixel-based format have *width* and *height* (in pixels) as additional arguments. For the other formats, *width* and *height* specify the page width and height in inches (1 inch = 2.54cm). Alternatively (e.g. in `pdf()` one can specify the *paper* option (e.g. `paper="a4"` or `paper="letter"`). Note that `pdf()` and `postscript()` can save multiple pages, i.e.:

```
> # saving multiple graphics in a PDF file:
> pdf("C:/test.pdf", paper = "a4")
> hist(rnorm(1000), main = "page 1")
> hist(rchisq(1000, 5), main = "page 2")
> dev.off()
```

Graphical parameters: `par()`

Most "graphical parameters" are the same across various graphical functions, i.e. options like *main* (titles), *xlim* and *ylim* (axes range), *col* (colours).

These parameters are covered by the "auxiliary" function `par()`. This means that the corresponding helpfiles etc. are not covered in the graphics function (such as `hist()`, see `?hist`), but in the `par()` helpfile (see `?par`). (As usual, some exceptions exist.) Some examples for the graphical options are given below:

adj Adjustment of text (0: left-justified, 0.5: centered, 1: right-justified). In e.g. `text()` different adjustment in x- and y-direction is possible. (Numeric (0,1), default: 0.5.)

ask Controls whether to ask the user for input before drawing a new figure. Logical, default: `FALSE`.

bg Background colour of plots . "Color", default: `"transparent"`.

cex Scaling factor for text and symbols. Related parameters are *cex.axis* (axis annotation), *cex.lab* (x and y labels), *cex.main* (main title) and *cex.sub* (sub-titles). (Numeric, default: 1.)

col The default plotting colour. As with *cex*, related parameters are *col.axis* (axis annotation), *col.lab* (x and y labels), *col.main* (main title) and *col.sub* (sub-titles). ("Color", default: `"black"`.)

bg Foreground colour of plots (e.g. axes and boxes). ("Color", default: `"black"`.)

font Font used for text. Related parameters are *font.axis* (axis annotation), *font.lab* (x and y labels), *font.main* (main title) and *font.sub* (sub-titles). (Integer code (1,2,3,...; system specific!), default: 1 (plain text).)

las Axis label style, possible values are 0 (parallel to axis), 1 (horizontal), 2 (perpendicular) and 3 (vertical). (Default: 0).

lend Line end style (draw a line with e.g. *lwd* = *5* to see). Specified as integer or string: 0 ("round"), 1 ("butt") and 2 ("square"). (Default: 0 ("round").)

lty Line type. Specified either as integer or string: 0 (“blank”), 1 (“solid”), 2 (“dashed”), 3 (“dotted”), 4 (“dot-dash”), 5 (“longdash”) and 6 (“twodash”). (Default: 1 (“solid”).)

lwd Line width as non-negative numeric (not accurately displayed on all devices). (Default: 1.)

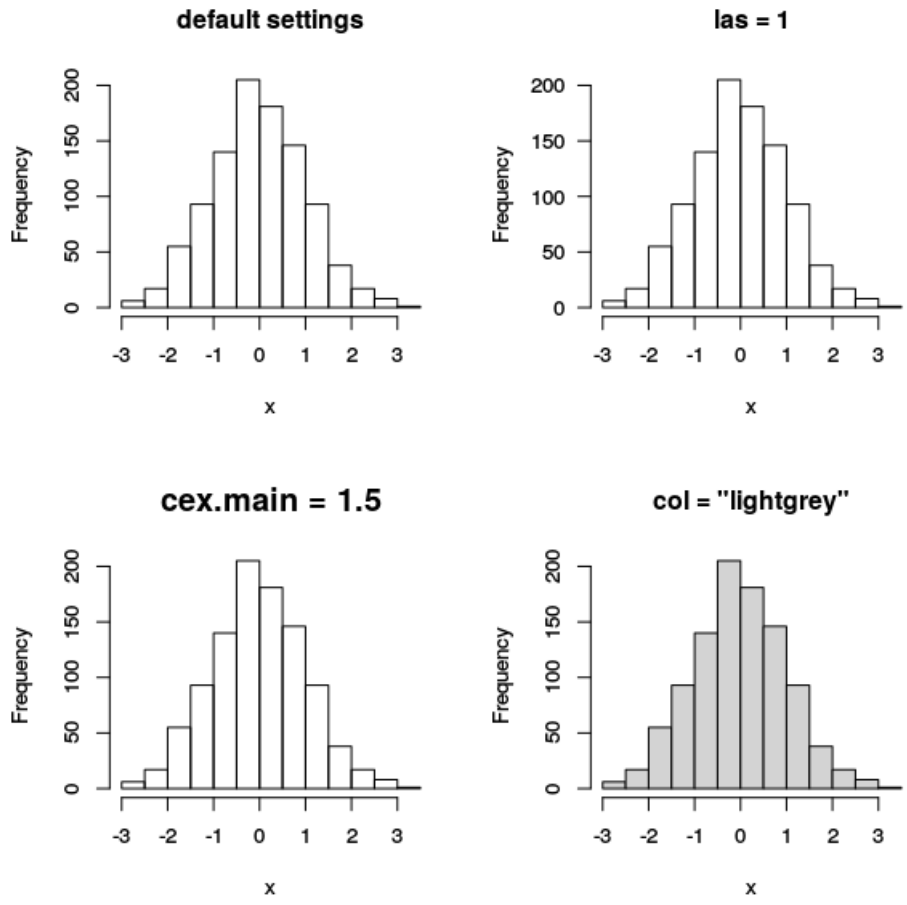
mai Margin size in inches. A numerical vector of length 4 for the following margins: bottom, left, top, right. (Default: c(0.95625, 0.76875, 0.76875, 0.39375).)

new
Controls if the device should (not) be cleared before drawing the next figure. (Logical, default: FALSE.)

pch Specifies the symbol for plotting points. Specified either as integer or a single character. (Default: 1.)

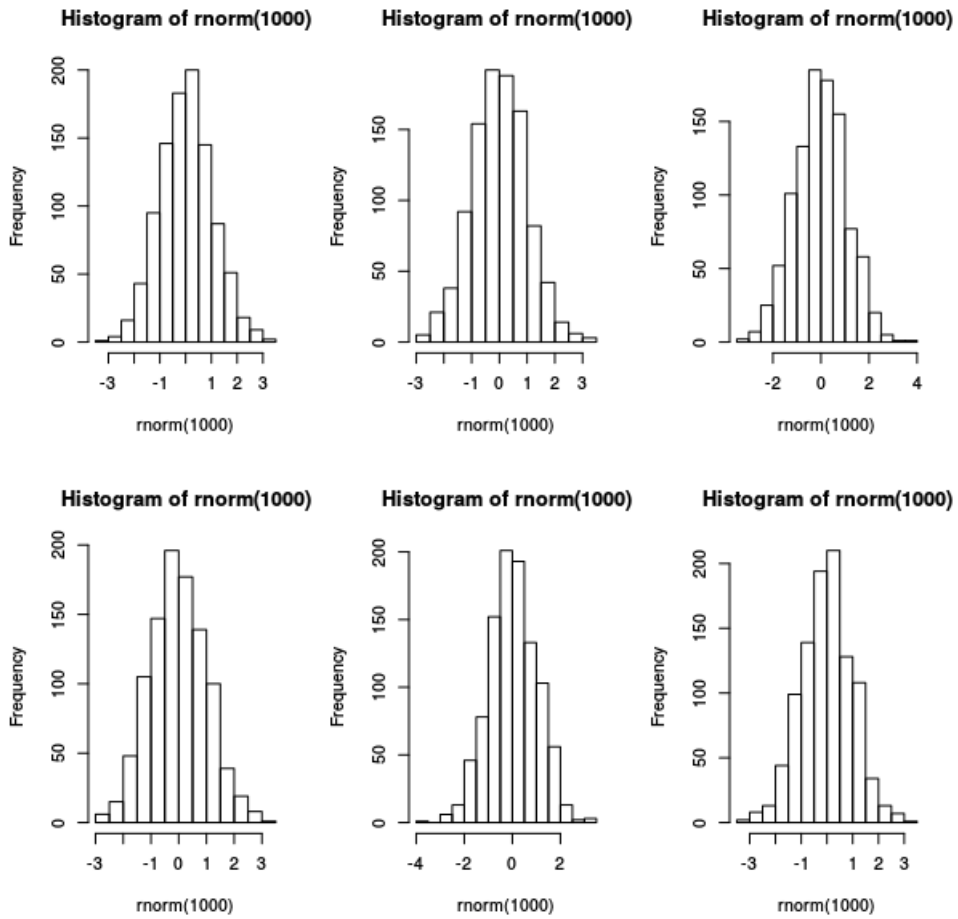
A full list is available with `?par`. In general, these parameters can be used either directly within the graphics-function (e.g. `plot(x, y, lty = 3)`) or separately via the `par()` command (e.g. `par(new = TRUE)`). The current value of a parameter is displayed with `par()`, e.g. `par()$col` returns the default colour.

```
> layout(matrix(c(1, 2, 3, 4), ncol = 2, byrow = TRUE))
> x <- rnorm(1000)
> hist(x, main = "default settings")
> hist(x, main = "las = 1", las = 1)
> hist(x, main = "cex.main = 1.5", cex.main = 1.5)
> hist(x, main = "col = \"lightgrey\"", col = "lightgrey")
```



In order to set up multiple graphs on one page, one can use `par(mfrow=c(x,y))` (where *x* gives the number of horizontal entries and *y* the number of vertical entries), e.g.:

```
> par(mfrow = c(2, 3))
> for (i in 1:6) hist(rnorm(1000))
> par(mfrow = c(1, 1))
```

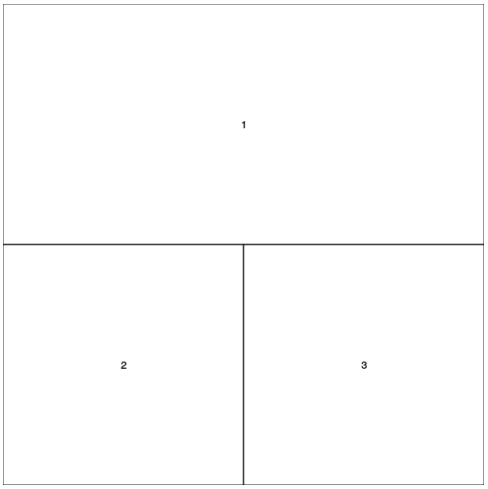


A preferable way is to use the function `layout()`, which allows for greater flexibility, e.g.:

```
> lay <- matrix(c(1, 1, 2, 3), nrow = 2, byrow = TRUE)
> lay

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

```
> layout(lay)
> layout.show(3)
> layout(1)
```

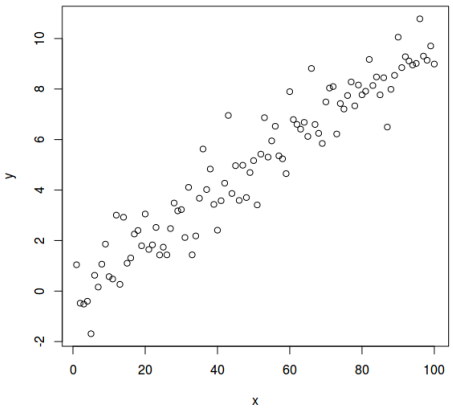


Selected graphical functions

`plot(x,y)` `# X-Y plotting`

Example:

```
x <- 1:100
y <- 0.1 * x + rnorm(100)
plot(x,y)
```



Some options:

`type` Specifies the plot type (`"p"` for points, `"l"` for lines, `"h"` for vertical lines, etc.)

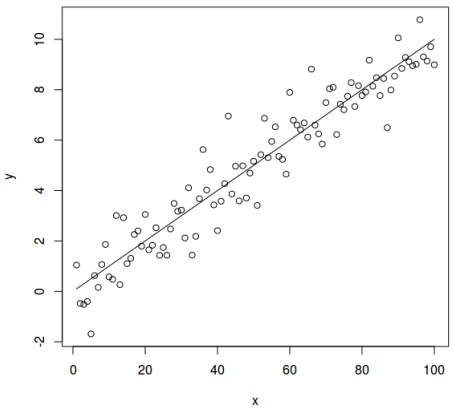
`pch` Point symbol used (integer or character)

`...`

`lines(x,y)` `# Adding lines to a plot`

Example:

```
x <- 1:100
y <- 0.1 * x + rnorm(100)
plot(x,y)
lines(x,0.1*x)
```



Some options:

`lty` Specifies the line type (`1`: solid line, `2`: dashed line, etc.)

`lwd` Line width

`...`

see also `abline(h,v)`

`hist(x)` `# Plotting histograms`

Example:

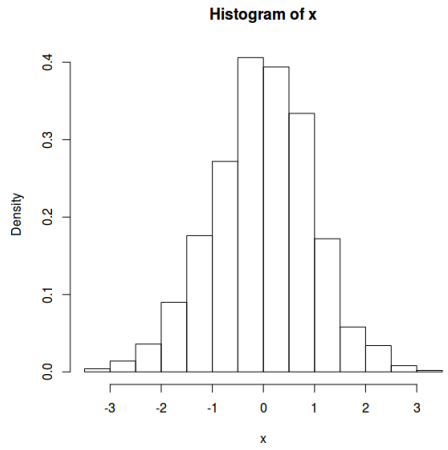
```
x <- rnorm(1000)
hist(x, prob = TRUE)
```

Some options:

`prob` Specifies if frequencies (`FALSE`) or relative frequencies (`TRUE`) are represented

`breaks` Set the histogram breaks

`...`



`boxplot(x)` `# Plotting boxplots`

Example:

```
x0 <- rnorm(100)
x1 <- rnorm(100, 0.5)
x2 <- rnorm(100, 1)
boxplot(x0, x1, x2)
```

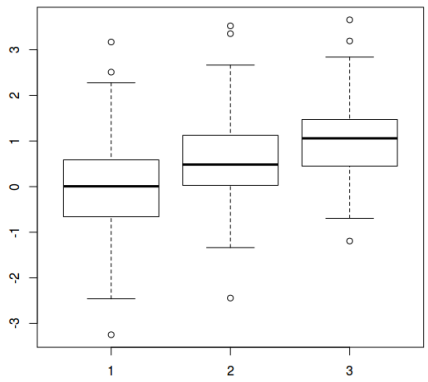
Some options:

`width` Specifies the box width

`horizontal` Sets horizontal or vertical boxes

`notch` Logical indicating if notches should be drawn

`...`



`barplot(x)` `# Bar plots`

Example:

```
x <- sample(15, 100, repl=T)
barplot(table(x), space=1)
```

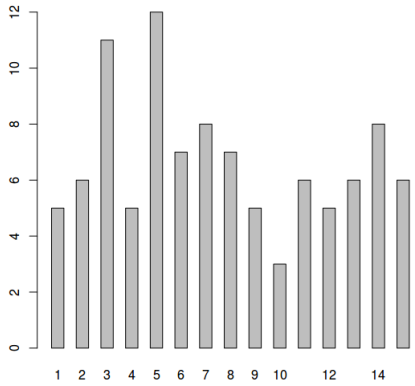
Some options:

`width` Specifies the bar widths

`horiz` Sets horizontal or vertical bars

`space` Space between bars

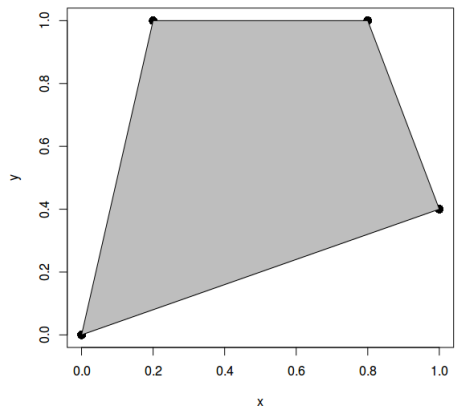
`...`



`polygon(x,y)` `# Drawing polygons`

Example:

```
x <- c(0,0.2,0.8,1)
y <- c(0,1,1,0.4)
plot(x,y,cex=1.5,pch=16)
polygon(x, y, col="grey")
```



Some options:

`col` The polygon fill color

`border` Specifies the border colour (if `NA`, no border line is drawn)

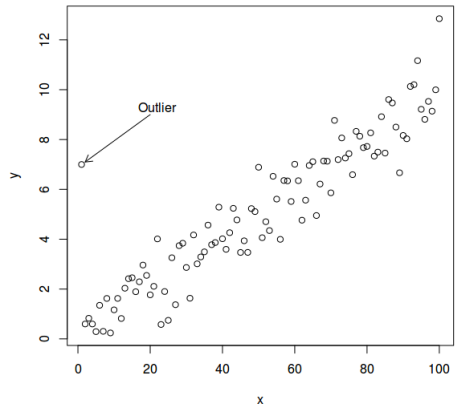
`lty` Line type for the border

`...`

`arrows(x0,y0,x1,y1)` `# Adding arrows to a plot`
`text(x,y,"text")` `# Adding text to a plot`

Example:

```
x <- 1:100
y <- 0.1 * x + rnorm(100)
y[1] <- 7
plot(x,y)
arrows(20,9,2,7.1, length=.1)
text(22,9.1,"Outlier", adj=c(0.5,0))
```

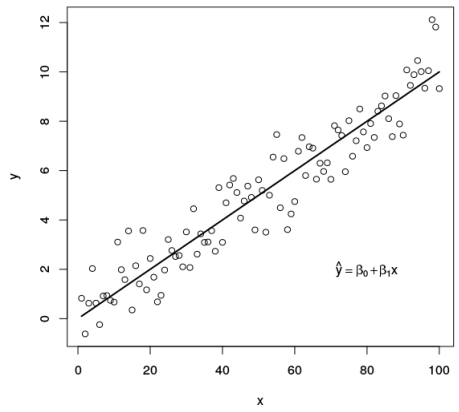


`expression()` `# Expressions`

Example:

```
x <- 1:100
y <- 0.1 * x + rnorm(100)
plot(x,y)
lines(x,0.1*x, lwd=2)
text(80,2, expression(
   hat(y)==beta[0]+beta[1]*x))
```

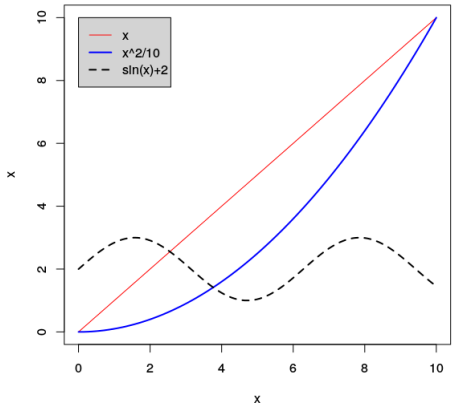
Comment:
A complete list of available mathematical expressions is given in the web page [**?**].



`legend()` `# Adding legends to plots`

Example:

```
x <- seq(0,10,length=250)
y1 <- x^2/10
y2 <- sin(x) + 2
plot(x,x, type="l", col=2)
lines(x,y1, lwd=2, col=4)
lines(x,y2, lwd=2, lty=2)
legend(0,10,
   c("x","x^2/10","sin(x)+2"),
   col=c(2,4,1),
   lwd=c(1,2,2),
   lty=c(1,1,2),
   bg="lightgray")
```

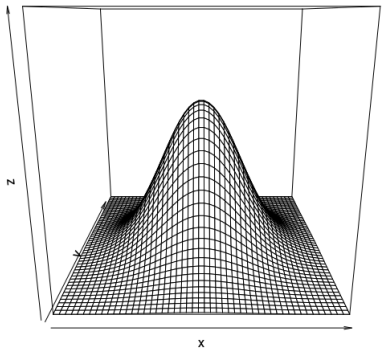


`persp()` `# perspective plots`

Example:

```
y <- x <- seq(-3,3,length=50)
f <- function(x,y){
          dnorm(x)*dnorm(y)}
z <- outer(x,y,f)
persp(x,y,z,zlim=c(0,0.25))
```

Notes:
x and y give the grid marks for the x- and the y-axis.
z is a matrix containing the height values for the corresponding (x,y) pair.

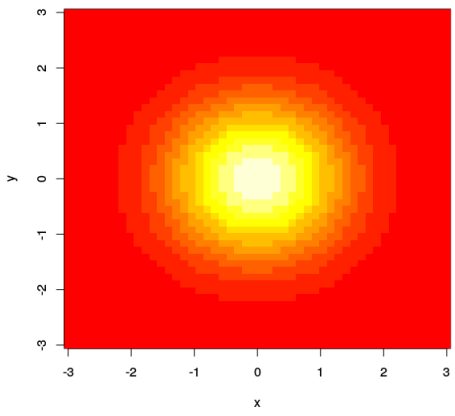


`image()` `# image plots`

Example:

```
y <- x <- seq(-3,3,length=50)
f <- function(x,y){
          dnorm(x)*dnorm(y)}
z <- outer(x,y,f)
image(x,y,z)
```

Notes:
x and y give the grid marks for the x- and the y-axis.
z is a matrix containing the height values for the corresponding (x,y) pair.

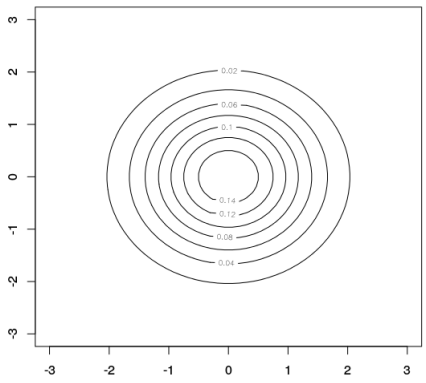


`contour()` `# contour plots`

Example:

```
y <- x <- seq(-3,3,length=50)
f <- function(x,y){
          dnorm(x)*dnorm(y)}
z <- outer(x,y,f)
contour(x,y,z)
```

Notes:
x and y give the grid marks for the x- and the y-axis.
z is a matrix containing the height values for the corresponding (x,y) pair.



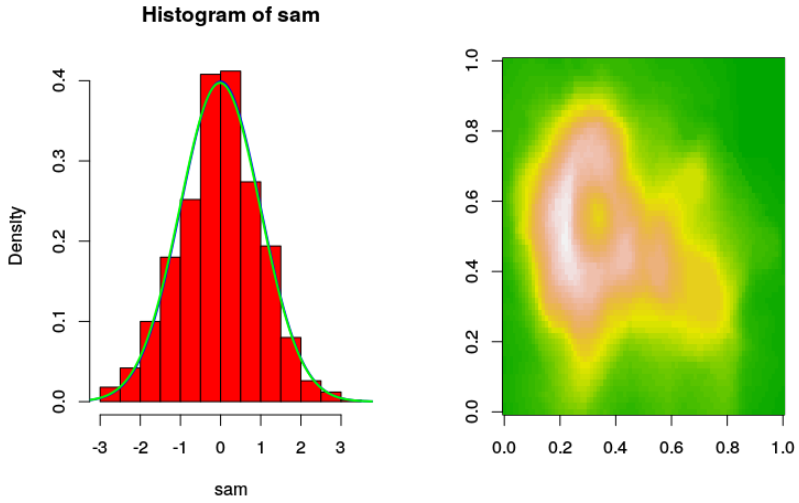
Specification of colours

There are multiple ways of specifying colours in R:

- Integer code (1: black, 2: red, 3: green, . . .)
- "R-name" (`"black"`, `"red"`, etc.; a full list is available with `colors()`)
- Colour-mixers such as `rgb(x,y,z)` and `hsv(h,s,v)`
- Pre-defined palettes (`rainbow()`, `heat.colors()`, `terrain.colors()`, etc.)

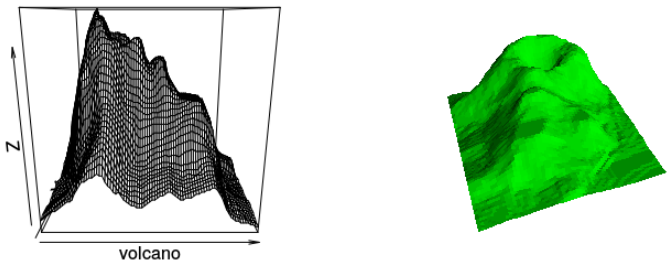
Some simple examples

```
> layout(matrix(c(1, 2), nrow = 1))
> # Example 1 (specification of single colours):
> sam <- rnorm(1000)
> hist(sam, prob = TRUE, col = 2)
> x <- seq(-4, 4, length = 250)
> lines(x, dnorm(x, 0, 1), col = "blue", lwd = 2)
> lines(x, dnorm(x, mean(sam), sd(sam)), col = rgb(0, 1, 0), lwd = 2)
> # Example 2 (using colour palettes):
> data(volcano)
> image(volcano, col = terrain.colors(50))
```



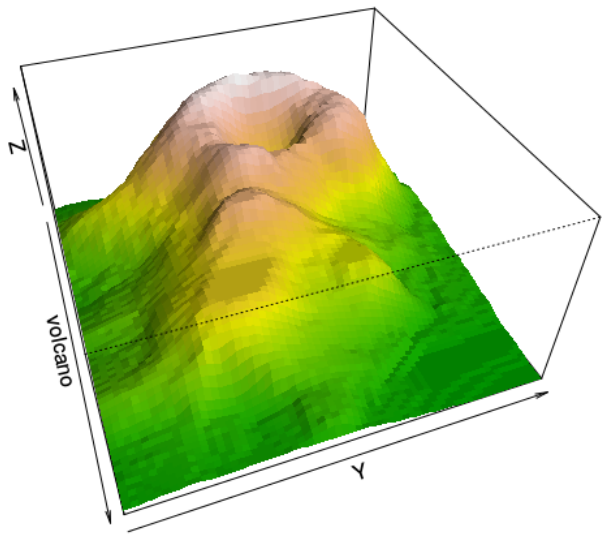
It's a little bit trickier to get example 2 in 3D via `persp()`. Firstly, there are some examples for the function itself:

```
> layout(matrix(c(1, 2), nrow = 1))
> persp(volcano)
> persp(volcano, col = "green", border = NA, shade = 0.9, theta = 70, phi = 40,
+     ltheta = 120, box = FALSE, axes = FALSE, expand = 0.5)
```



In order to obtain height-dependent colours one has to use a "lookup"-table:

```
> layout(1)
> collut <- terrain.colors(101)
> temp <- 1 + 100 * (volcano - min(volcano))/(diff(range(volcano)))
> mapcol <- collut[temp[1:86, 1:61]]
> persp(volcano, col = mapcol, border = NA, shade = 0.9, theta = 70, phi = 40,
+     ltheta = 120, lphi = 30, expand = 0.5)
```



Additional examples (such as adding points/lines to a 3D-plot) are given in the `persp()`-helpfile.

2 Graphical packages

One of the reasons for the success of R is that it offers a convenient way for users to enhance its capabilities via add-ons (packages). This section does not focus on specific packages, but rather how to find out what is offered within a package and how to use it.

Every package has to have built-in helpfiles for its functions (at least the official ones). These can be browsed through by clicking on "Help" → "HTML help" → "Packages". After clicking on the package name, a list of the available functions' helpfiles appear.
The helpfiles contain details on the function and its arguments, and, more importantly, examples of usage.

An optional item is the `demo()`-function:

```
> library(rgl)
> demo(rgl)
>
> library(lattice)
> demo(lattice)
```

Additionally, authors may provide a so-called package vignette with the package, which basically is a paper describing the package.
Since authors usually promote using their packages, one is likely to find further information (papers, presentations, etc.) on the web.

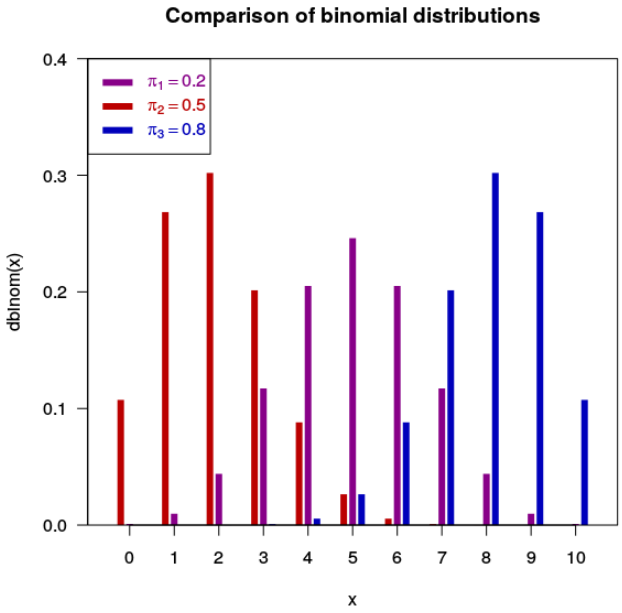
Since 2005, we have a data visualization package called `ggplot2` that has grown in use to become one of the most popular R packages. Created by Hadley Wickham, `ggplot2` is an implementation of Leland Wilkinson's Grammar of Graphics, a general scheme for data visualization which breaks up graphs into semantic components such as scales and layers. `ggplot2` can serve as a replacement for the base graphics in R and contains a number of defaults for web and print display of common scales. More information on web page [**?**].

3 Miscellanea

Example: Comparing discrete distributions

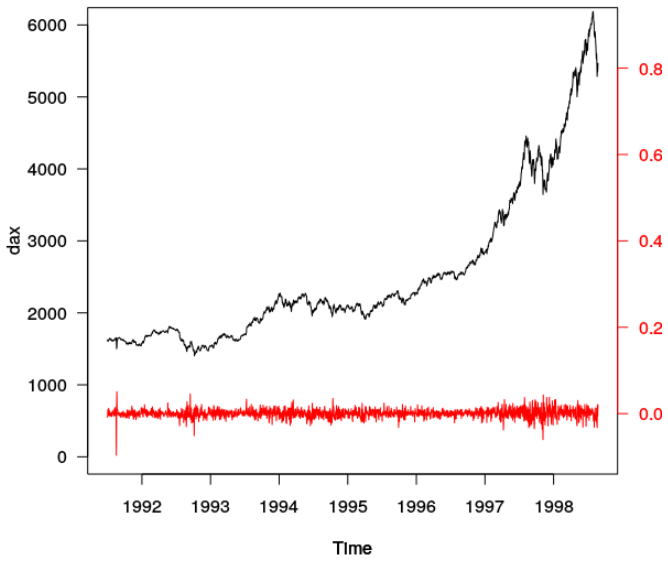
Assume that we want to compare the binomial distribution for different values of π:

```
> x <- 0:10
> y1 <- dbinom(x, 10, 0.2)
> y2 <- dbinom(x, 10, 0.5)
> y3 <- dbinom(x, 10, 0.8)
>
> par.lend <- par()$lend
> par(lend = "butt")
> plot(x, y2, xlim = c(-0.5, 10.5), ylim = c(0, 0.4), lwd = 6, col = "#880088",
+     type = "h", ylab = "dbinom(x)", las = 1, yaxs = "i", axes = FALSE)
> lines(x - 0.2, y1, lwd = 6, col = "#BB0000", type = "h")
> lines(x + 0.2, y3, lwd = 6, col = "#0000BB", type = "h")
> title("Comparison of binomial distributions")
>
> legend("topleft", c(expression(pi[1] == 0.2), expression(pi[2] == 0.5),
+     expression(pi[3] == 0.8)), col = c("#880088", "#BB0000", "#0000BB"),
+     text.col = c("#880088", "#BB0000", "#0000BB"), lwd = c(6, 6, 6))
>
> box()
> axis(2, las = 1)
> axis(1, at = 0:10)
```



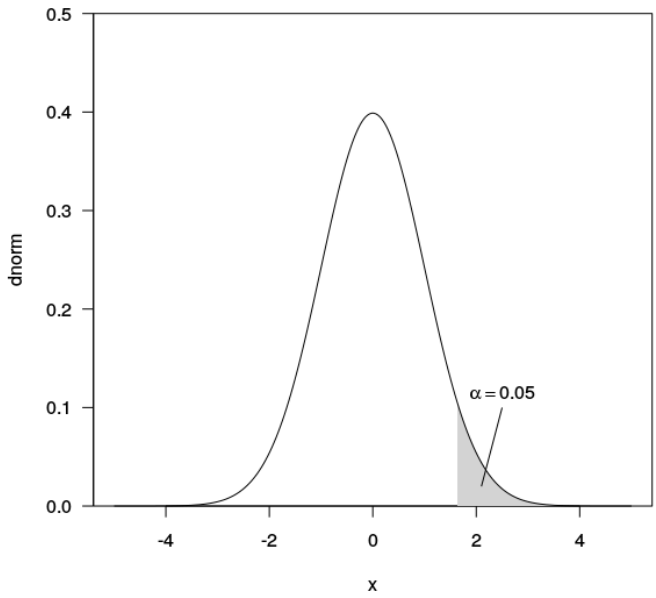
In this case we have created a plot first (for $\pi = 0.5$) and subsequently added other plots with the `lines()`-command.
A similar approach can be taken for showing multiple time series (eventually with different scales) in one plot:

```
> data("EuStockMarkets")
> dax <- EuStockMarkets[, 1]
> plot(dax, ylim = c(0, 6000), axes = FALSE)
> axis(1)
> axis(2, las = 1)
>
> par(new = TRUE)
> plot(diff(log(dax)), ylim = c(-0.1, 0.9), axes = FALSE, col = 2, ylab = "")
> box()
> axis(4, col = 2, col.axis = 2, las = 1)
```



Another useful function is `polygon()`, which can be used for indicating confidence regions (e.g. predictions for time series) or areas under a distribution:

```
> x <- seq(-5, 5, length = 250)
> y <- dnorm(x)
> plot(x, y, las = 1, ylab = "dnorm", type = "n", yaxs = "i", ylim = c(0,
+     0.5))
>
> x2 <- seq(qnorm(0.95), 5, length = 50)
> y2 <- dnorm(x2)
> polygon(c(x2[1], x2, x2[length(x2)]), c(0, y2, 0), border = NA, col = "lightgrey")
> lines(x, y)
>
> lines(c(2.5, 2.1), c(0.1, 0.02))
> text(2.5, 0.115, expression(alpha == 0.05))
```



Creating animations in R

Consider the following case, where we wish to illustrate the amplitude of a cosine function (see figure **??**).
E.g. the amplitude parameter should take values starting from 1, then going up to 1.4, then down to 0.6 and, back to 1.
A movie/animation can be thought of a series of still images which are displayed at a rate of e.g. 25 images per second. So, if we want a 6 second-animation, we would have to create 150 sequential still images for it.

This approach comprises 2 steps:

(i) Generating sequential still images

(ii) Combining (i.e. merging) the images from (i) into a movie file

In our case (i) is taken care of in a loop; the corresponding code look like this:

```
> h <- 1
> rho <- 0
> omega <- 1
> x <- seq(-1.5 * pi + 0.5, 2 * pi + 0.5, length = 250)
```

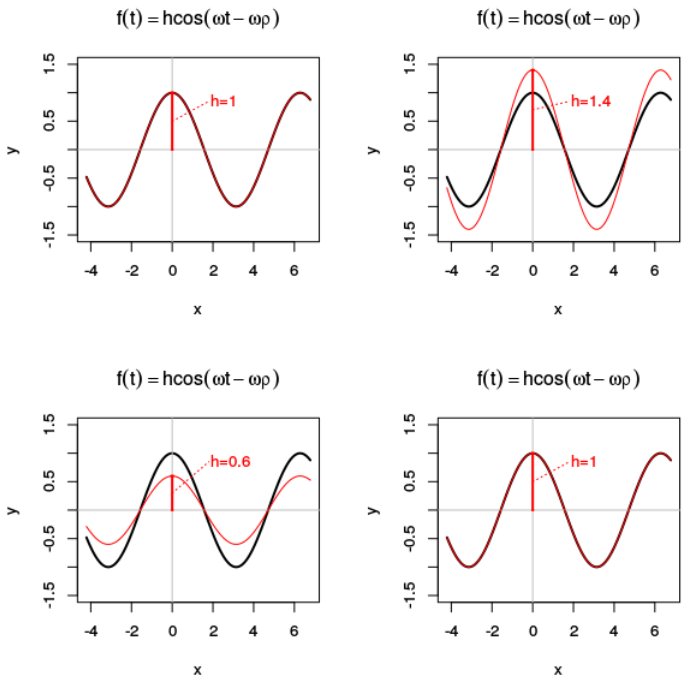


Figure 1: Cosine function $f(t) = h \cdot \cos(\omega t - \omega\rho)$ for $h = 1, 1.4, 0.6, 1$, $\omega = 1$ and $\rho = 0$

```
> y <- h * cos(omega * (x - rho))
>
> # Loop for creating stills:
>
> hnew <- c(seq(1, 1.4, length = 50), seq(1.4, 0.6, length = 50), seq(0.6,
+     1, length = 50))
>
> imgind <- 1000
> outdir <- file.path(getwd(), "tsdemo/")
>
> for (i in 1:length(hnew)) {
+     png(file = paste(outdir, "tsdemo_", imgind, ".png", sep = ""))
+     plot(x, y, type = "l", ylim = c(-1.5, 1.5), main = expression(f(t) ==
+         h * cos(omega * t - omega * rho)), lwd = 2)
+     abline(h = 0, v = 0, col = "grey75")
+     h <- hnew[i]
+     lines(x, h * cos(omega * (x - rho)), col = 2)
+     lines(c(0, 0), c(0, h), col = 2, lwd = 2)
+     lines(c(-0.05, 0.05), c(h, h), col = 2, lwd = 2)
+     lines(c(-0.05, 0.05), c(0, 0), col = 2, lty = 2)
+     lines(c(0, 1.8), c(h/2, 0.85), col = 2, lty = 3)
+     text(1.9, 0.87, paste("h=", round(h, 2), sep = ""), adj = 0, col = 2)
+     dev.off()
+     imgind <- imgind + 1
+ }
```

The second step is done with an external program (i.e. ImageMagick), which can be run in batch mode:

```
> system("convert ./tsdemo/*.png demo.mpeg")
```

References

[1] http://vis.supstat.com/2013/04/mathematical-annotation-in-r/ 1

[2] http://tutorials.iq.harvard.edu/R/Rgraphics/Rgraphics.html 2

[3] P. Murrell, *R Graphics*, Chapman & Hall/CRC, 2011.
https://www.stat.auckland.ac.nz/~paul/RG2e/

[4] J. Verzani, *Using R for Introductory Statistics*. Chapman & Hall/CRC, 2014.