R Programming/Graphics

R includes at least three graphical systems, the standard **graphics** package, the **lattice** package for Trellis graphs[1] and the *grammar-of-graphics* **ggplot2** package[2]. R has good graphical capabilities but there are some alternatives like gnuplot.

Contents

Interactive Graphics

This section discuss some ways to draw graphics without using R scripts.

The **playwith** package provides a graphical user interface to customize the graphs, add a title, a grid, some text, etc and it exports the R code you need if you want to replicate the analysis[3]. If you want to know more, you can have a look at the screenshots on the website (link (http://code.google.com/p/playwith/wiki/Screenshots)). See also the example on "R you Ready" [1] (http://ryouready.wordpress.com/2010/03/23/playing-with-the-playwith-package/). This package require GTK+ libraries.

```
library("playwith")
playwith(plot(x1))
```

There is also a graphical user interface **GrapheR** which makes it very easy to draw graphs for beginners[4]. This solution is cross-platform.

```
> library(GrapheR)
```

latticist (link (http://code.google.com/p/latticist/)) is another similar project.

Note also that some graphical user interface such as RKward and R Commander makes it easy to draw graphs.

Standard R graphs

In this section we present what you need to know if you want to customize your graphs in the default graph system.

- `plot()` is the main function for graphics. The arguments can be a single point such as 0 or c(.3,.7), a single vector, a pair of vectors or many other R objects.
- `par()` is another important function which defines the default settings for plots.
- There are many other plot functions which are specific to some tasks such as `hist()`, `boxplot()`, etc. Most of them take the same arguments as the `plot()` function.

```
> N <- 10^2
> x1 <- rnorm(N)
> x2 <- 1 + x1 + rnorm(N)
> plot(0)
> plot(0,1)
> plot(x1)
> plot(x1,x2) # scatter plot x1 on the horizontal axis and x2 on the vertical axis
> plot(x2 ~ x1) # the same but using a formula (x2 as a function of x1)
> methods(plot) # show all the available methods for plot (depending on the number of loaded packages).
```

Titles, legends and annotations

Titles

`main` gives the main title, `sub` the subtitle. They can be passed as argument of the `plot()` function or using the `title()` function. `xlab` the name of the x axis and `ylab` the name of the y axis.

```
plot(x1,x2, main = "Main title", sub = "sub title" , ylab = "Y axis", xlab = "X axis")
plot(x1,x2 ,  ylab = "Y axis", xlab = "X axis")
title(main = "Main title", sub = "sub title" )
```

The size of the text can be modified using the parameters `cex.main`, `cex.lab`, `cex.sub`, `cex.axis`. Those parameters define a *scaling factor*, ie the value of the parameter multiply the size of the text. If you choose `cex.main=2` the main title will be twice as big as usual.

Legend

`legend()`. The position can be "bottomleft", "bottomright", "topleft", "topright" or exact coordinates.

```
plot(x1, type = "l", col = 1, lty = 1)
lines(x2, col = 2, lty = 2)
legend("bottomleft", legend = c("x1","x2"), col = 1:2, lty = 1:2)
```

Text in the margin

`mtext()` puts some texts in the margin. The margin can be at the bottom (1), the left (2), the top (3) or the right (4).

```
plot(x1, type = "l", col = 1, lty = 1) ; mtext("some text", side = 1) # the bottom
plot(x1, type = "l", col = 1, lty = 1) ; mtext("some text", side = 2) # the left
plot(x1, type = "l", col = 1, lty = 1) ; mtext("some text", side = 3) # the top
plot(x1, type = "l", col = 1, lty = 1) ; mtext("some text", side = 4) # the right margin
```

Text in the graph

`text()`

Mathematical annotations

We can add mathematical symbols using `expression()` and makes some substitution in a formula using `substitute()`.

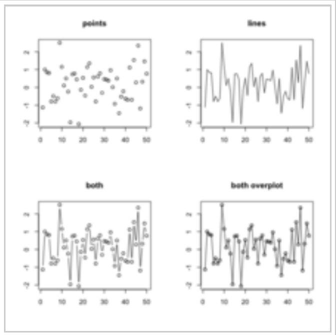
```
?plotmath # gives help for mathematical annotations
```

Types

The type of a plot can be :

- `n` for none (nothing is printed),
- `p` for points,
- `l` for lines,
- `b` for both,
- `o` for both overlayed,
- `h` for histogram-like
- and `s`/`S` for steps.

R code	Output

`x1 <- rnorm(50)`
`png("plottype.png")`
`par(mfrow = c(2,2))`
`plot(x1, type = "p", main = "points", ylab = "", xlab = "")`
`plot(x1, type = "l", main = "lines", ylab = "", xlab = "")`
`plot(x1, type = "b", main = "both", ylab = "", xlab = "")`
`plot(x1, type = "o", main = "both overplot", ylab = "", xlab = "")`
`dev.off()`
	
	click on the graph to zoom

Axes

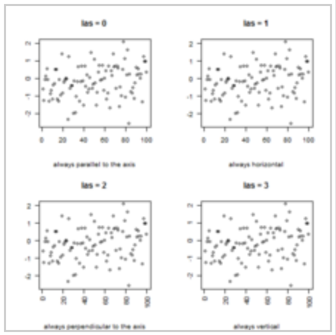
The default output print the axes. We can remove them with `axes=FALSE`. We can also change them using the `axis()` function.

```
> plot(x1,x2,axes=FALSE)
>
> plot(x1,x2,axes=FALSE)
> axis(1,col="red",col.axis="blue",font.axis=3)
> axis(2,col="red",col.axis="blue",font.axis=2,las=2)
```

`las` specifies the style of axis labels. It can be 0, 1, 2 or 3.

- 0 : always parallel to the axis [default],
- 1 : always horizontal,
- 2 : always perpendicular to the axis,
- 3 : always vertical.

R code	Output

`x1 <- rnorm(100)`
`par(mfrow = c(2,2))`
`plot(x1, las = 0, main = "las = 0", sub = "always parallel to the axis", xlab = "", ylab = "")`
`plot(x1, las = 1, main = "las = 1", sub = "always horizontal", xlab = "", ylab = "")`
`plot(x1, las = 2, main = "las = 2", sub = "always perpendicular to the axis", xlab = "", ylab = "")`
`plot(x1, las = 3, main = "las = 3", sub = "always vertical", xlab = "", ylab = "")`
	
	click on the graph

It is also possible to add another y axis on the right by adding `axis(4,)`.

Margins

Margins can be computed in inches or in lines. The default is `par(mar = c(5,4,4,2))` which means that there are 5 lines at the bottom, 4 lines on the left, 4 lines in the top and 2 lines on the right. This can be modified using the `par()` function. If you want to specify margins in inches, use `par(mai = c(bottom, left, top, right)`. If you want to modify margins in lines, use `par(mar = c(bottom, left, top, right)`. See `?par` to learn more about the topic.

Colors

The color of the points or lines can be changed using the `col` argument, `fg` for foreground colors (boxes and axes) and `bg` for background colors.

- `show.col(object=NULL)` (**Hmisc**) package plots the main R colors with their numeric code.
- The list of all colors in R (pdf (http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf))

```
colors() # list the r colors
show.col(object=NULL) # graphs the main R colors
plot(x1, col = "blue")
plot(x1, col = "red")
plot(x1, col = "red", col.axis = "dodgerblue", col.lab = "firebrick", col.main = "darkgreen", col.sub = "cyan4", main = "Testing colors", sub
```

- We can also generate new colors using the `rgb()` function. The first argument is the intensity of red, the second, the intensity of green and the third, the intensity of blue. They vary between 0 and 1 by default but this can be modified with the option `max = 255`. `col2rgb()` returns the RGB code of R colors. `col2hex()` (**gplots**) gives the hexadecimal code. `col2grey()` and `col2gray()` (**TeachingDemos**) converts colors to grey scale.

```
> mycolor <- rgb(.2,.4,.6)
> plot(x1, col = mycolor)
> col2rgb("pink")
      [,1]
red    255
green  192
blue   203
> library("gplots")
> col2hex("pink")
[1] "#FFC0CB"
```

Points

For points the symbols can be changed using the `pch` option which takes integer values between 0 and 25 or a single character. `pch` can also takes a vector as argument. In that case the first points will use the first element of the vector as symbol, and so on.

```
plot(x1, type = "p", pch = 0)
plot(x1, type = "p", pch = 10)
plot(x1, type = "p", pch = 25)
plot(x1, type = "p", pch = "a")
plot(x1, type = "p", pch = "*")
plot(x1[1:26], type = "p", pch = 0:25)
plot(x1[1:26], type = "p", pch = letters)
```

The following code displays all the symbols on the same plot :

```
x <- rep(1,25)
plot(x, pch = 1:25, axes = F, xlab = "", ylab = "")
text(1:25,.95,labels = 1:25)
```

`points()` adds points to an existing plot.

```
> plot(x1, pch = 0) # plot x1
> points(x2, pch = 1, col = "red") # add x2 to the existing plot
```

Lines

We can change the line type with `lty`. The argument is a string ("blank", "solid", "dashed", "dotted", "dotdash", "longdash", or "twodash") or an integer (0=blank, 1=solid (default), 2=dashed, 3=dotted, 4=dotdash, 5=longdash, 6=twodash). The line width can be changed with `lwd`. The default is `lwd=1`. `lwd=2` means that the width is twice the normal width.

```
plot(x1, type = "l", lty = "blank")
plot(x1, type = "l", lty = "solid")
plot(x1, type = "l", lty = "dashed")
plot(x1, type = "l", lty = "dotted")
plot(x1, type = "l", lty = "dotdash")
plot(x1, type = "l", lty = "longdash")
plot(x1, type = "l", lty = "twodash")
```

`lines()` adds an additional lines on a graph.

```
plot(x1, type = "l", lty = "solid")
lines(x2, type = "l", lty = "dashed", col = "red")
```

`abline()` adds an horizontal line (`h=`), a vertical line (`v=`) or a linear function to the current plot (`a=` for the constant and `b=` for the slope). `abline()` can also plot the regression line.

```
> plot(x1, type = "l", lty = "solid")
> abline(h= -3, lty = "dashed", col = "gray")
> abline(v = 0, lty = "dashed", col = "gray")
> abline(a = -3 , b = .06, lty = "dotted", col = "red")
```

Boxes

Each graph is framed by a box. `bty` specifies the box type.

```
plot(x1, bty = "o") # the default
plot(x1, bty = "n") # no box
plot(x1, bty = "l")
plot(x1, bty = "7")
plot(x1, bty = "u")
plot(x1, bty = "c")
plot(x1, bty = "]")
```

See also `box()` to add a box to an existing plot.

Grid

`grid()` adds a grid to the current graph.

```
> plot(x1)
> grid()
```

Although grid has an optional argument nx for setting the number of grid lines, it is not possible to tell it explicitly where to place those lines (it will usually not place them at integer values). A more precise and manageable alternative is to use abline().

```
> abline(v=(seq(0,100,5)), col="lightgray", lty="dotted")
> abline(h=(seq(0,100,5)), col="lightgray", lty="dotted")
```

Arrows and segments

Polygons

Other figures

We can also add a circle to a plot with the `circle()` function in the **calibrate** package.

Background

You can choose the background of your plot. For instance, you can change the background color with `par(bg=)`.

```
par(bg="whitesmoke")
par(bg="transparent")
```

Overlaying plots

`matplot()` can plot several plots at the same time.

```
N <- 100
x1 <- rnorm(N)
x2 <- rnorm(N) + x1 + 1
y <- 1 + x1 + x2 + rnorm(N)
mydat <- data.frame(y,x1,x2)
matplot(mydat[,1],mydat[,2:3], pch = 1:2)
```

Multiple plots

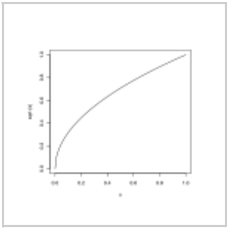
With `par()` we can display multiple figures on the same plot. `mfrow = c(3,2)` prints 6 figures on the same plot with 3 rows and 2 columns. `mfcol = c(3,2)` does the same but the order is not the same.

```
par(mfrow = c(3,2))
plot(x1, type = "n")
plot(x1, type = "p")
plot(x1, type = "l")
plot(x1, type = "h")
plot(x1, type = "s")
plot(x1, type = "S")

par(mfcol = c(3,2))
plot(x1, type = "n")
plot(x1, type = "p")
plot(x1, type = "l")
plot(x1, type = "h")
plot(x1, type = "s")
plot(x1, type = "S")
```

Plotting a function

- `curve()` plots a function. This can be added to an existing plot with the option `add = TRUE`.
- `plot()` can also plots functions.

```
curve(x^2, from = -1 , to = 1, main = "Quadratic function", ylab = "f(x)=x^2")

plot(rnorm(100))
curve((x/100)^2, add = TRUE, col = "red")
```



square root function, made using `plot()`

Exporting graphs

How can you export a graph ?

- First you can plot the graph and use the context menu (right click on Windows and Linux or control + click

on Mac) to copy or save the graphs. The available options depend on your operating system. On Windows, you can also use copy the current graph to the clipboard as a Bitmap file (raster graphics) using CTRL + C or as a Windows Metafile (vector graphics) using CTRL + W. You can then paste it into another application.
- You can export a plot to **pdf**, **png**, **jpeg**, **bmp** or **tiff** by adding `pdf("filename.pdf")`, `png("filename.png")`, `jpeg("filename.jpg")`, `bmp("filename.bmp")` or `tiff("filename.tiff")` prior to the plotting, and `dev.off()` after the plotting.
- You can also use the `savePlot()` function to save existing graphs.
- Sweave also produce ps and pdf graphics (See the Sweave section).

It is better to use vectorial devices such as **pdf**, **ps** or **svg**.

How can you know the list of all available devices ?

- `?Devices`
- Use the `capabilities()` function to see the list of available devices on your computer.

```
?Devices
> capabilities()
    jpeg      png     tiff    tcltk      X11     aqua http/ftp  sockets
    TRUE     TRUE     TRUE     TRUE    FALSE    FALSE     TRUE     TRUE
  libxml     fifo   cledit    iconv      NLS  profmem    cairo
    TRUE    FALSE     TRUE     TRUE     TRUE     TRUE    FALSE
```

```
png("r_plot.png", width = 420, height = 340)
plot(x1, main = " Example")
dev.off()

pdf("r_plot.pdf", width = 420, height = 340)
plot(x1, main = " Example")
dev.off()

postscript(file="graph1.ps",horizontal=F,pagecentre=F,paper="special",width=8.33,height=5.56)
plot(x1, main = "Example")
dev.off()

plot(x1, main = "Example")
savePlot("W:/Bureau/plot.pdf", type = "pdf")
savePlot("W:/Bureau/plot.png", type = "png")
```

We can also export to SVG using the `svg()` function.

```
svg("scatterplot.svg", width = 7, height = 7)
plot(x, y)
dev.off()
```

The **RSvgDevice** library which was used in earlier versions of R seems now outdated.

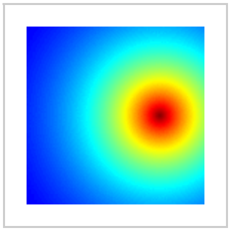
Advanced topics

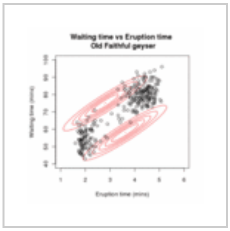
Animated plots

The **animation** package provides dynamic graphics capabilities. It is possible to export the animation in flash, mpeg or gif format. There are more example on the aniwiki website : http://animation.yihui.name/.

You can also create *motion charts* using the **googleVis** package[5].

Examples





Interactive Graphics

The **iplots** package provides a way to have interactive data visualization in R[6] .[7].

- R GUI now offers interactive graphics – Deducer 0.4-2 connects with iplots (http://www.r-statistics.com/2010/10/r-gui-now-offers-interactive-graphics-deducer-0-4-2-connects-with-iplots/)

To create an interactive, animated plot viewable in a web browser, the animint package (https://github.com/tdhock/animint) can be used. The main idea is to define an interactive animation as a list of ggplots with two new aesthetics:

- showSelected=variable means that only the subset of the data that corresponds to the selected value of variable will be shown.
- clickSelects=variable means that clicking a plot element will change the currently selected value of variable.

Graphics gallery

In this section, we review all kind of statistical plots and review all alternatives to draw them using R. This include code for the standard graphics package, the **lattice** package and the **ggplot2** package. Also, we add some examples from the commons repository. We only add examples which are provided with the R code. You can click on any graph and find the R code.

Line plot

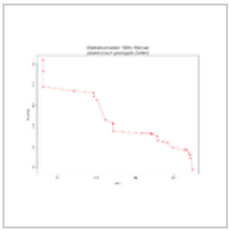
To draw a line plot, use the generic `plot()` function by setting `type="l"`.

```
> x <- seq(0, 2*pi, pi/10)
> plot(x, sin(x), type="l")
```

Then, you can add further lines on the same plot using the `lines()` function.

```
> lines(x, cos(x))
```

Examples



Scatter plot

- `plot(x,y)`
- `plot(y ~ x)`
- `xyplot(y ~ x)` (**lattice**)
- `qplot(x,y)` (**ggplot2**)

Log scale

Sometimes it is useful to plot the log of a variable and to have a log scale on the axis. It is possible to plot the log of a variable using the `log` option in the `plot()` function.

- For a log log plot, use `log = "xy"`
- For a log in the x axis only, use `log = "x"`
- For a log in the x axis only, use `log = "y"`

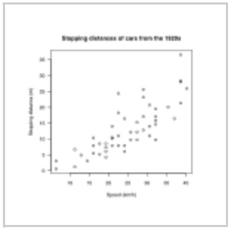
```
plot(x, y , log = "xy")
```

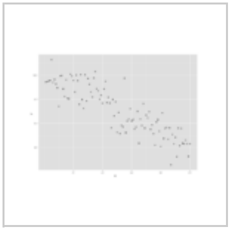
Label points in a plot

- It is possible to add labels with the `text()` function.
- `textxy()` (**calibrate**) makes it easy to add labels.

```
N <- 10
u <-rnorm(N)
x <- 1 + rnorm(N)
y <- 1 + x + u
plot(x, y)
textxy(x, y,labs = signif(x,3), cx=0.7)
```

Examples



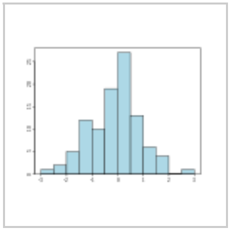
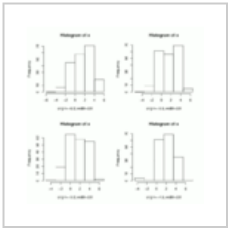
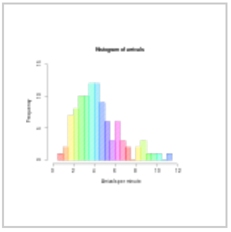
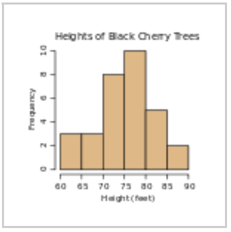


Histogram

- `hist()`
- `histogram()` (**lattice**)

You can learn more about histograms in the Non parametric methods page.

Examples

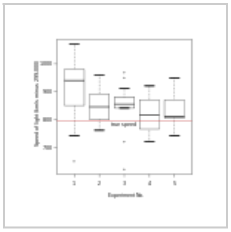





Box plot

Box plot :

- `boxplot()`

Examples



See also

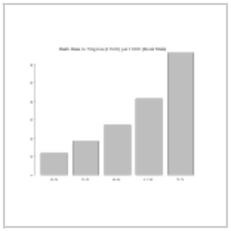
- How to label all the outliers in a boxplot (http://www.r-statistics.com/2011/01/how-to-label-all-the-outliers-in-a-boxplot/)

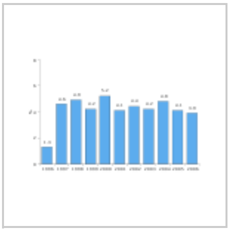
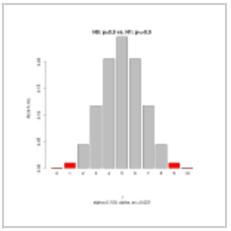
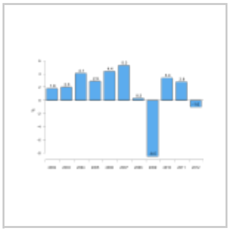
Bar charts

See Bar charts on wikipedia.

- `barplot()` takes a table as argument and returns a bar chart.
- `qlot()` (**ggplot2**) with the option `geom = "bar"` takes a variable as argument and returns a bar chart[8].
- `barchart()` takes a variable as argument and returns a bar chart.

Examples



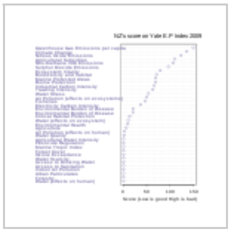




Dot plot

See also Dot plot on Wikipedia.

- dotchart()

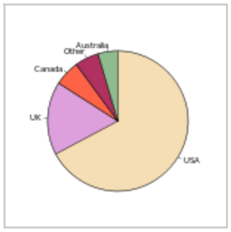
Examples

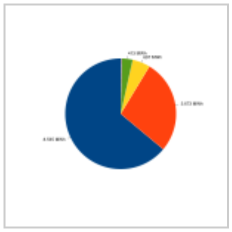


Pie charts

- `pie()`

Examples





Treemap

The `tmPlot()` function in the **treemap** package makes it easy to draw a treemap.

Confidence interval plot

Standard error bar chart are very useful to plot several estimates with confidence intervals.

- The **Hmisc** package has an `errbar()` function. This function takes the upper and lower bounds of the confidence intervals as argument[9].

- `coefplot()` function in Gelman and Hill's **arm** package. This functions is designed to display estimation results. It takes point estimates and standard errors as arguments.

```
coefs <- c(0.2, 1.4, 2.3, 0.5,.3) # vector of point estimates
se <- c(0.12, 0.24, 0.23, 0.15,.2) # standard errors of point estimates
variable <- 1:5 # variable names
library("arm")
# we use CI = qnorm(.975) to have 95% confidence interval
coefplot(coefs, se, variable, vertical = T, CI = qnorm(.975))
coefplot(coefs, se, variable, vertical = F, CI = qnorm(.975))
library("Hmisc")
errbar(variable, coefs, coefs - qnorm(.975) * se, coefs + qnorm(.975) * se)
```

See also

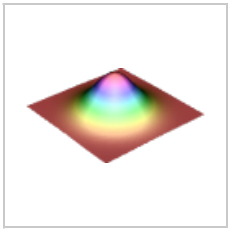
- There is another `errbar()` function in the **sfsmisc** package.
- `plotCI()` (**gplots**) also plot error bars.
- `plotmeans()` (**gplots**)
- `ciplot()` (**hacks**)
- See also Error bar on Wikipedia

3D plots

- `contour()`, `image()`, `persp()`
- `plot3d()` (**rgl**)
- `wireframe()` (**lattice**)

Examples

click on the graph to see the R code



Example with `wireframe()` (**lattice**)

Diagrams

- **grid** package by Paul Murrell[10]
- **diagram** package [11]
- **Rgraphviz** package
- **igraph** package

Arc Diagrams

It is also possible to draw Arc Diagrams[12].

Dendrograms

It is possible to plot dendrograms in R[13].

Treemap

It is possible to draw a treemap using the `treemap()` function in the **treemap** package[14].

Wordcloud

There is :

- the `wordcloud()` function in the **wordcloud** package
- the `tagcloud()` function in the **tagcloud** package

Timeline

- `timeline()` in the timeline package

Maps

See also

- Geograpic Maps in R (Revolution Computing Blog) (http://blog.revolution-computing.com/2009/10/geographic-maps-in-r.html)
- Global Administrative Areas (http://gadm.org/)
- Andy Egger's **googlemap.r** function (http://www.iq.harvard.edu/blog/sss/archives/2008/04/google_charts_f_1.shtml)

Resources

- Tables 2 Graphs (http://tables2graphs.com/doku.php)
- **R Graphics** by Paul Murrell[15]
- **ggplot2** [16]

- **Graphical Parameters** [2] (http://www.statmethods.net/advgraphs/parameters.html)

References

1. D. Sarkar. Lattice: Multivariate Data Visualization with R. Springer, 2008. ISBN 9780387759685.
2. *ggplot2: Elegant Graphics for Data Analysis (Use R)* by Hadley Wickham and a list of examples on his own website : http://had.co.nz/ggplot2/
3. playwith : http://code.google.com/p/playwith/
4. Hervé, Maxime (2011). "GrapheR: a Multiplatform GUI for Drawing Customizable Graphs in R". *The R Journal* **3** (2). http://journal.r-project.org/archive/2011-2/RJournal_2011-2_Herve.pdf.
5. Tutorial for the **googleVis** package : http://stackoverflow.com/questions/4646779/embedding-googlevis-charts-into-a-web-site/4649753#4649753
6. http://www.r-bloggers.com/interactive-graphics-with-the-iplots-package-from-%E2%80%9Cr-in-action%E2%80%9D/
7. http://www.r-statistics.com/2012/01/interactive-graphics-with-the-iplots-package-from-r-in-action/ Interactive Graphics with the iplots Package] - a chapter from the *R in action* book
8. Hadley Wickham *ggplot2: Elegant Graphics for Data Analysis*, Springer Verlag, 2009
9. The default output in `errbar()` changed between R version 2.8.1 and R version 2.9.2. Axis are not displayed by default anymore
10. Paul Murrell *Drawing Diagrams with R*, The R Journal, 2009 http://journal.r-project.org/2009-1/RJournal_2009-1_Murrell.pdf
11. (example: Using a binary tree diagram for describing a Bernoulli process (http://www.r-statistics.com/2011/11/diagram-for-a-bernoulli-process-using-r/))
12. Gaston Sanchez (Feburary 3rd, 2013). "Arc Diagrams in R: Les Miserables". http://gastonsanchez.wordpress.com/2013/02/03/arc-diagrams-in-r-les-miserables/. Retrieved February 5th, 2013.
13. Gaston Sanchez (October 3, 2012). "7+ ways to plot dendrograms in R". http://gastonsanchez.wordpress.com/2012/10/03/7-ways-to-plot-dendrograms-in-r/. Retrieved February 5th, 2013.
14. http://cran.r-project.org/web/packages/treemap/treemap.pdf
15. http://www.stat.auckland.ac.nz/~paul/RGraphics/rgraphics.html
16. http://had.co.nz/ggplot2/