R programming in Life Sciences

Introduction to R graphics.

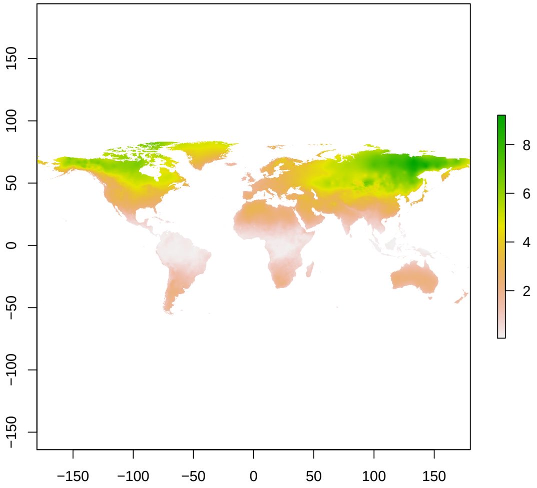
Marcin Kierczak*

*Department of Medical Biochemistry and Microbiology
Faculty of Medicine and Pharmacy
Uppsala University
Uppsala, SWEDEN

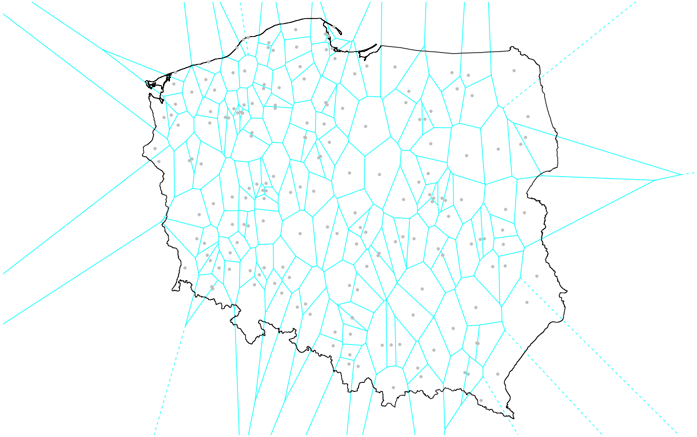
04 Nov 2016, Uppsala

Graphics

Example – stability of the climate, courtesy of dr. Mats Pettersson

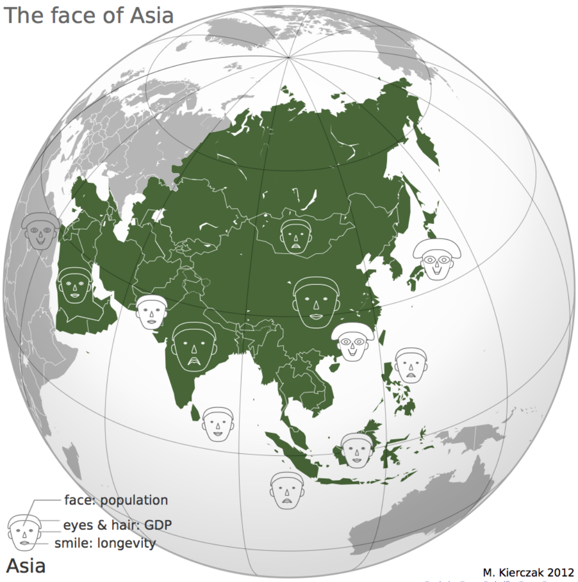


Graphics
Example – airport-based Voronoi tesselation of Poland

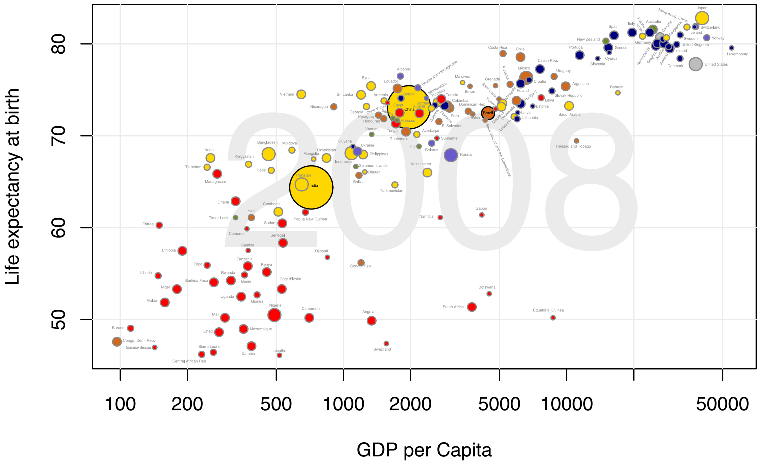


Graphics

Example – The face of Asia



Graphics
Example – Gapminder-type plot



The concept of a **graphical device** is crucial for understanding R graphics. A device can be a screen (default) or a file. Some R packages introduce their own devices, e.g. Cairo device.
Creating a plot entails:

- opening a graphical device (not necessary for plotting on screen),
- plotting to the graphical device,
- closing the graphical device (very important!).

Graphics
The most common graphical devices

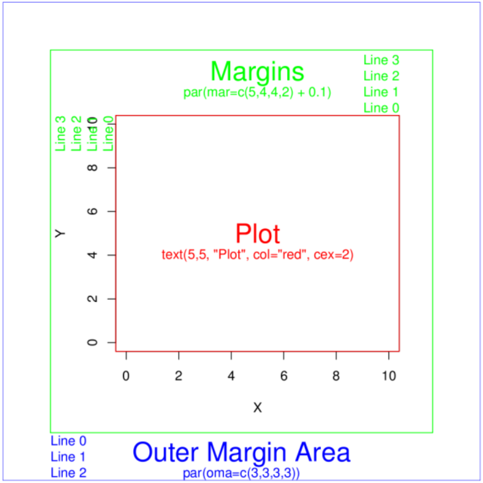
The most commonly used graphical devices are:

- screen,
- bitmap/raster devices: png(), bmp(), tiff(), jpeg()
- vector devices: svg(), pdf(),
- Cairo versions of the above devices – for Windows users they offer higher quality graphics,
- for more information visit http://stat.ethz.ch/R-manual/R-devel/library/grDevices/html/Devices.html

Graphics
Working with graphical devices

```r
png(filename = 'myplot.png', width = 320, height = 320,
    antialias = T)
plot(x=c(1,2,7), y=c(2,3,5))
dev.off()
```

Graphics
Standard graphical device viewport

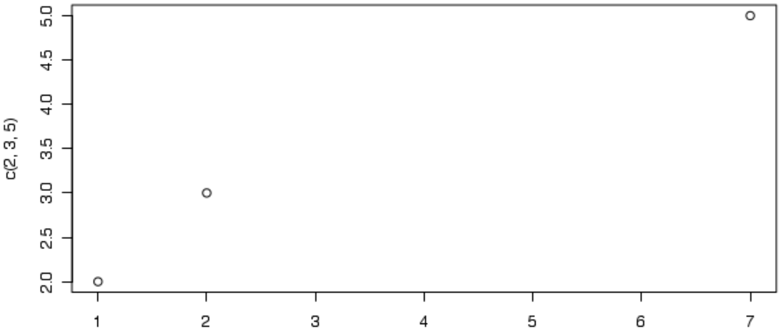


source: rgraphics.limnology.wisc.edu

Plot – basics

`plot()` is the basic command that lets you visualize your data and results. It is very powerful yet takes some effort to fully master it. Let's begin with plotting three points: A(1,2); B(2,3); C(7,5).

```
plot(x = c(1, 2, 7), y = c(2, 3, 5))
```

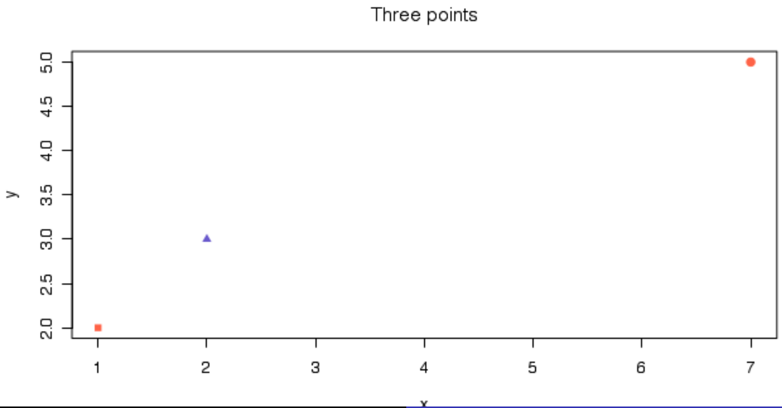


For convenience, we will create a data frame with our points:

```
df <- data.frame(x=c(1,2,7), y=c(2,3,5),
                 row.names=c("A","B","C"))
df

##   x y
## A 1 2
## B 2 3
## C 7 5
```

Let's make our plot a bit fancier...

```
plot(df, pch=c(15,17,19),
     col=c("tomato", "slateblue"), # recycling rule in action!
     main="Three points", sub="Stage 1")
```



Graphics
Plot – parameters

There is many parameters one can set in `plot()`. Let's have a closer look at some of them:

- pch – type of the plotting symbol
- col – color of the points
- cex – scale for points
- main – main title of the plot
- sub – subtitle of the plot
- xlab – X-axis label
- ylab – Y-axis label
- las – axis labels orientation
- cex.axis – axis lables scale

Graphics
Graphical parameters

Graphical parameters can be set in two different ways:

- as plotting function arguments, e.g. plot(dat, cex=0.5)
- using par()

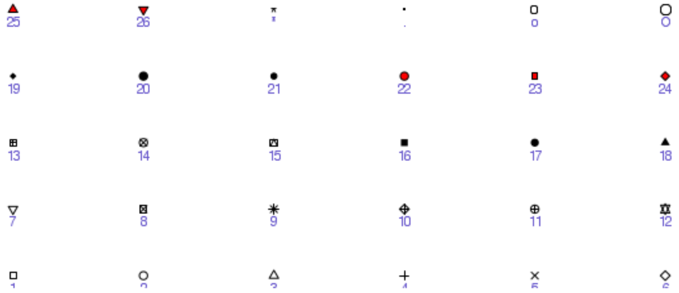
```
# read current graphical parameters
par()
# first, save the current parameters so that you
# can set them back if needed
opar <- par() # should work in theory, practise varies :-(
# now, modify what you want
par(cex.axis = 0.8, bg='grey')
# do your plotting
plot(................)
# restore the old parameters if you want
par(opar)
```

Graphics
Graphical parameters – pch

```r
# create a grid of coordinates
coords <- expand.grid(1:6,1:6)
# make a vector of numerical pch symbols
pch.num <- c(0:25)
# and a vector of character pch symbols
pch.symb <- c('*', '.', 'o', 'O', '0', '-', '+', '|', '%', '#')
# plot numerical pch
plot(coords[1:26,1], coords[1:26,2], pch=pch.num,
     bty='n', xaxt='n', yaxt='n', bg='red',
     xlab='', ylab='')
# and character pch's
points(coords[27:36,1], coords[27:36,2], pch=pch.symb)
# label them
text(coords[,1], coords[,2], c(1:26, pch.symb), pos = 1,
     col='slateblue', cex=.8)
```

Graphics

Graphical parameters – pch result

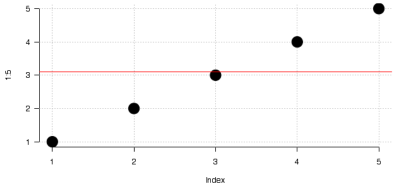


Now, make your only cheatsheet for the **lty** parameter!

Graphics
Layers

Elements are added to a plot in the same order you plot them. It is like layers in a graphical program. Think about it! For instance the auxiliary grid lines should be plotted before the actual data points.

```r
# make an empty plot
plot(1:5, type='n', las=1,
     bty='n')
# plot grid
grid(col='grey', lty=3)
# plot the actual data
points(1:5, pch=19, cex=3)
# plot a line
abline(h = 3.1, col='red')
# you see, it overlaps the
# data point. It is better
# to plot it before points()
```



Graphics
General considerations

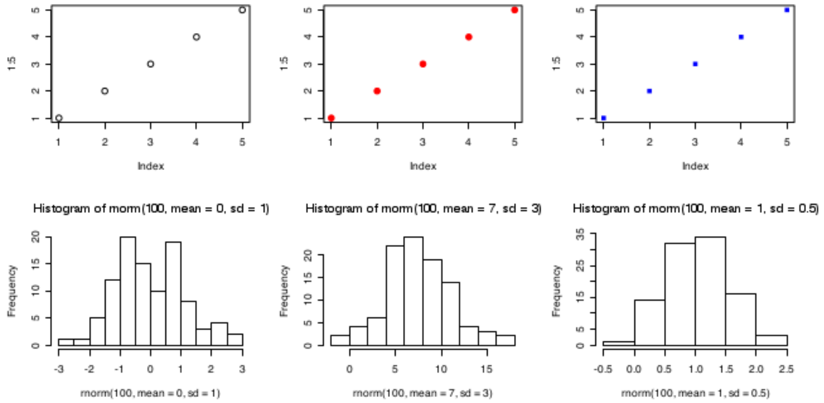
There is a few points you should have in mind when working with plots:

- raster or vector graphics,
- colors, e.g. color-blind people, warm vs. cool colors and optical illusions,
- avoid complications, e.g. 3D plots, pie charts etc.,
- use black ang greys for things you do not need to emphasize, i.e. basically everything except your main result,
- avoid 3 axis.

Many plots on one figure

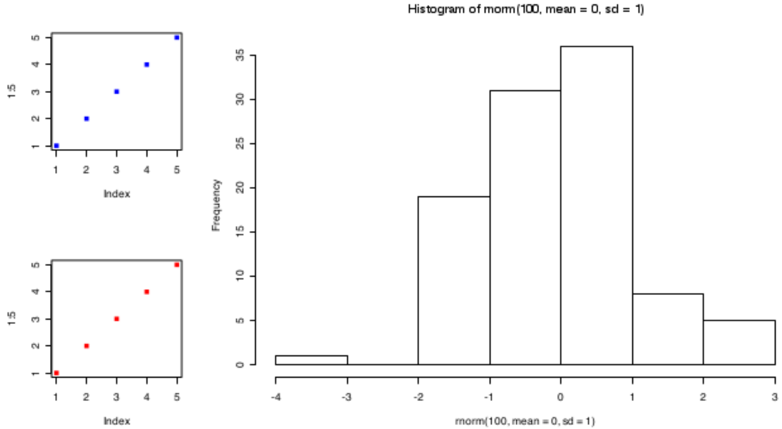
```
par(mfrow=c(2,3))
plot(1:5)
plot(1:5, pch=19, col='red')
plot(1:5, pch=15, col='blue')
hist(rnorm(100, mean = 0, sd=1))
hist(rnorm(100, mean = 7, sd=3))
hist(rnorm(100, mean = 1, sd=0.5))
par(mfrow=c(1,1))
```

Graphics
Many plots on one figure – result

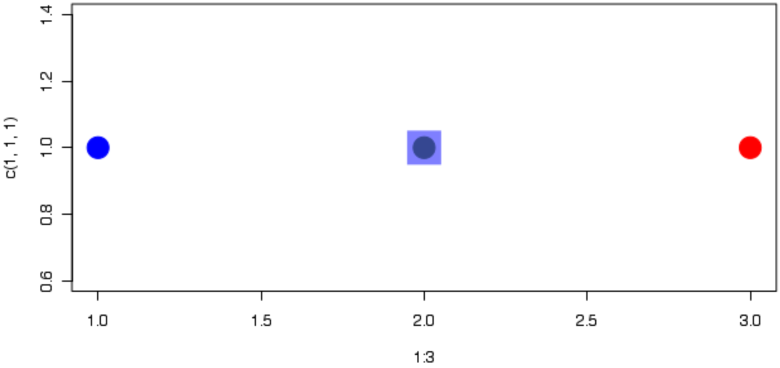


Many plots on one figure – layout

```
layout(matrix(c(1, 2, 2, 2, 3, 2, 2, 2), nrow = 2, ncol = 4, byrow = T)
plot(1:5, pch = 15, col = "blue")
hist(rnorm(100, mean = 0, sd = 1))
plot(1:5, pch = 15, col = "red")
```



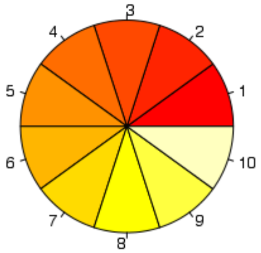
```r
mycol <- c(rgb(0, 0, 1), "olivedrab", "#FF0000")
plot(1:3, c(1, 1, 1), col = mycol, pch = 19, cex = 3)
points(2, 1, col = rgb(0, 0, 1, 0.5), pch = 15, cex = 5)
```



Graphics
Color palettes

There some built-in palettes: default, hsv, gray, rainbow, terrain.colors, topo.colors, cm.colors, heat.colors

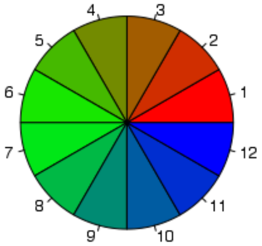
```
mypal <- heat.colors(10)
mypal


##  [1] "#FF0000FF" "#FF2400FF" "#FF4900FF" "#FF6D00FF" "#FF9200FF"
##  [6] "#FFB600FF" "#FFDB00FF" "#FFFF00FF" "#FFFF40FF" "#FFFFBFFF"


pie(x = rep(1, times = 10), col = mypal)
```



Graphics
Custom palettes

You can easily create custom palettes:

```
mypal <- colorRampPalette(c("red", "green", "blue"))
pie(x = rep(1, times = 12), col = mypal(12))
```



Note that `colorRampPalette` returns a function!

Graphics
Palettes cted.

- There is an excellent package `RColorBrewer` that offers a number of pre-made palettes, e.g. color-blind safe palette.
- Package `wesanderson` offers palettes based on Wes Anderson movies:-)

Graphics
Plot – boxplot

There are also more specialized R functions used for creating specific types of plots. One commonly used is `boxplot()`

Rule of thumb. If median of one boxplot is outside the box of another, the median difference is likely to be significant ($\alpha = 5\%$)

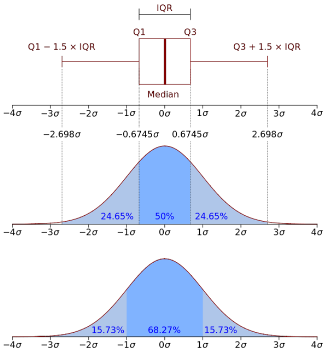
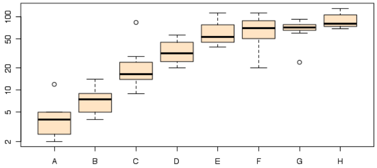


Figure: source: commons.wikimedia.org

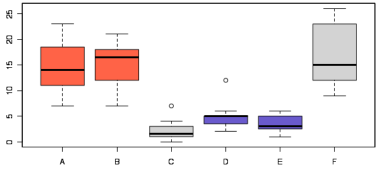
```
boxplot(decrease ~ treatment,
  data = OrchardSprays,
  log = "y", col = "bisque",
  varwidth=T)
```



Graphics
Plot – boxplot cted.

add=TRUE parameter in plots allows you to compose plots using previously plotted stuff!

```
attach(InsectSprays)
boxplot(count~spray,
        data=InsectSprays[spray %in% c("C","F"),],
        col="lightgray")
boxplot(count~spray,
        data=InsectSprays[spray %in% c("A","B"),],
        col="tomato", add=T)
  boxplot(count~spray,
  data=InsectSprays[spray %in% c("D","E"),],
  col="slateblue", add=T)
detach(InsectSprays)
```

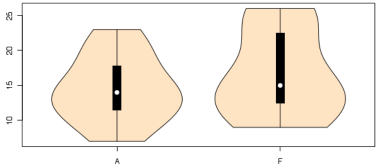


Graphics
Plot – violin plots

Package `vioplot` lets you plot *violin plots*. A violin plot is very similar to a boxplot, but it also visualizes distribution of your datapoints.

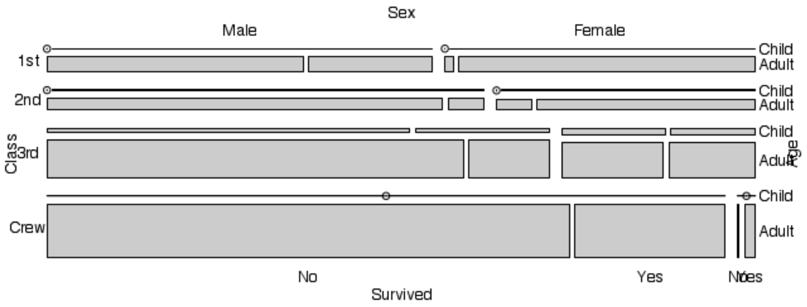
```
library(vioplot)

## Loading required package:  sm
## Package 'sm', version
2.2-5.4:  type help(sm) for
summary information
```

```
attach(InsectSprays)
vioplot(count[spray=="A"],
        count[spray=="F"],
        col="bisque",
        names=c("A","F"))
detach(InsectSprays)
```



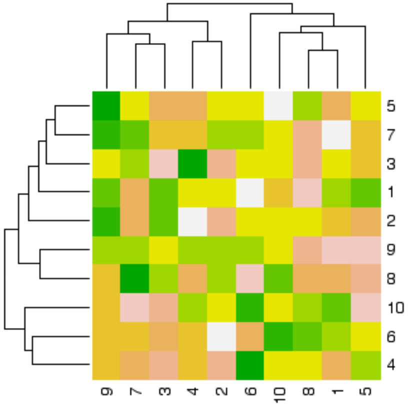
Graph gallery
Categorical data with vcd

```
library(vcd) # load the vcd package


## Loading required package:  grid

data(Titanic) # Load the data
mosaic(Titanic, labeling=labeling_border(rot_labels = c(0,0,0,0))) # Plot the data
```



Graph gallery
Heatmaps

```
heatmap(matrix(rnorm(100, mean = 0, sd = 1), nrow = 10),
        col=terrain.colors(10))
```



For more awesome examples visit: http://www.r-graph-gallery.com