Introduction
High-level plotting commands
Low-level plotting commands
Pimp your Graph
Other graph styles
lattice()

R

Alejandra E. Medina Rivera
Licenciatura en Ciencias Genómicas.
Centro de Ciencias Genómicas, UNAM

Cuernavaca, Mexico
Feb, 2010

Introduction to R and Statistics

1 Introduction

2 High-level plotting commands

3 Low-level plotting commands

4 Pimp your Graph

5 Other graph styles

6 `lattice()`

R plots

Despite the fact that R has almost no graphical interface, its capabilities at producing high quality graphical outputs are probably even more than we will ever need. Just to give ourselves an idea of the variety of graphics we can draw:

```
> demo(graphics)
```

Graphical procedures

- The graphics facilities can be used in both interactive and batch modes, but in most cases, interactive use is more productive.
- Interactive use is also easy because at startup time R initiates a graphics device driver which opens a special graphics window for the display of interactive graphics.
- Although this is done automatically, it is useful to know that the command used is `X11()`under `UNIX`, `windows()` under Windows and `quartz()` under Mac OS X.
- Once the device driver is running, `R` plotting commands can be used to produce a variety of graphical displays and to create entirely new kinds of display.
- Plotting commands are divided into three basic groups:

R

Introduction

High-level plotting commands

Low-level plotting commands

Pimp your Graph

Other graph styles

lattice()

Graphical procedures

- **High-level plotting functions** create a new plot on the graphics device, possibly with axes, labels, titles and so on.
- **Low-level plotting functions** add more information to an existing plot, such as extra points, lines and labels.
- **Interactive graphics functions** allow you interactively add information to, or extract information from, an existing plot, using a pointing device such as a mouse.

- In addition, R maintains a list of graphical parameters which can be manipulated to customize your plots.
- Basic graphical parameters can be changed using the function `par()`, be careful since this parameters are applied for **ALL** graphs.

Graphic Device

- We know already that `R` uses default graphic devices: `X11()` under `UNIX`, `windows()` and `quartz()` under Mac OS X.
- To open any of this graphic devices you just have to call `X11()`, and `R` will open the appropriate device.
- We can open graphic devices pointing to files with the appropriate functions: `postscript()`, `pdf()`, `png()`
- Lets open three independent graphic devices

```
> x11()
> x11()
> pdf()
```

- Now list them

```
> dev.list()
```

Graphic Device

- When we open several devices, by default R will modify the last one. So we need to be able to know where we are drawing and change it.

```
> dev.cur()
> dev.set(3)
```

- Once the graph is done, we need to close the device specially if we are using a file, the pdf will be blank till we close the device

```
> dev.off(2)
> dev.off()
```

Set the canvas

- The graphic device is like your canvas, and you can change the distribution, and draw more than one graph on it.

```
> x11()
> split.screen(c(1, 2))
```

- We can select where to draw

```
> screen(1)
> screen(2)
```

- We can split the window into a matrix.

```
> layout(matrix(1:4, 2, 2))
> mat <- matrix(1:4, 2, 2)
> mat
> layout(mat)
```

- Cool, so far we just believe this is happening

```
> layout.show(4)
```

Small practice

- Divide the device in 6 parts, distributed in 2 columns
- Now the 6 parts in 3 columns
- Divide the device in 3, where 2 parts are in one column

Set the canvas

- We haven't used the `byrow` parameter of the `matrix()` function, for this reason the sub-devices are numbered by columns, to change this just specify `matrix(..., byrow=TRUE)`.
- However, you can number the sub-devices as you want `matrix(c(2, 1, 4, 3), 2, 2)`
- By default `layout()` divides the device in regular dimensions, this can changed modifying parameters `widths` and `heights`

Set the canvas

```
> m <- matrix(1:4, 2, 2)
> layout(m, widths = c(1, 3), heights = c(3,
+     1))
> layout.show(4)
> m <- matrix(1:4, 2, 2)
> layout(m, widths = c(1, 3), heights = c(3,
+     1))
> layout.show(4)
```

High-level plotting

- High-level plotting functions are designed to generate a complete plot of the data passed as arguments to the function.
- Where appropriate, axes, labels and titles are automatically generated (unless you request otherwise.)
- High-level plotting commands always start a new plot, erasing the current plot if necessary
- There are several functions available in R, and some packages include their own

plot(x)	graphs values of x (in y axis) ordered on the x axis
plot(x, y)	graphics bivariate data of x (x axis) and y (y axis)
piechart(x)	pie chart
boxplot(x)	box-and-whiskers chart

High-level plotting

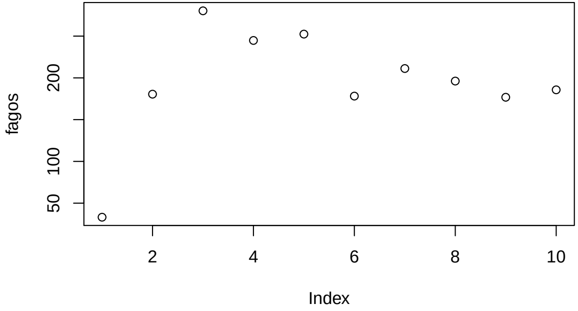
Now lets use the `plot()` function, imagine we have the data of the genome size of some phages [1]

```
> fagos <- c(33.2, 180.5, 280.3, 244.8,
+       252.4, 178.2, 211.2, 196.2, 176.8,
+       185.7)

> plot(fagos)
```

Introduction
High-level plotting commands
Low-level plotting commands
Pimp your Graph
Other graph styles
lattice()

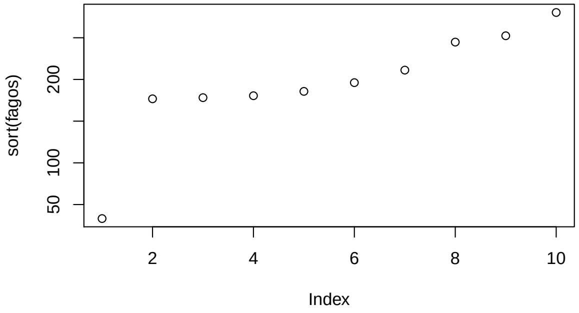
High-level plotting



What happen if we sort the vector?

```
> plot(sort(fagos))
```

High-level plotting



[1] Example from Leonardo Collado.

High-level plotting

Introduction
High-level plotting commands
Low-level plotting commands
Pimp your Graph
Other graph styles
lattice()

Check out the help for `plot()`, what parameters can help you to pimp your graph?

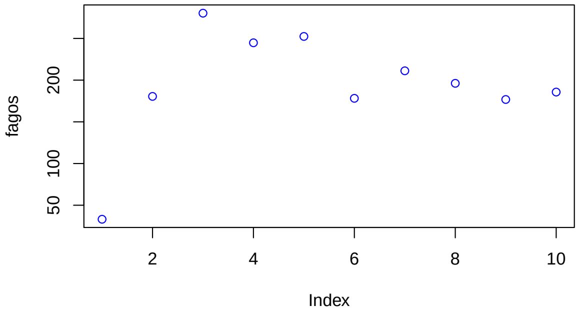
```
> help(plot)
```

Now lets change the color of the points

```
> plot(fagos, col = "blue")
```

Introduction
High-level plotting commands
Low-level plotting commands
Pimp your Graph
Other graph styles
lattice()

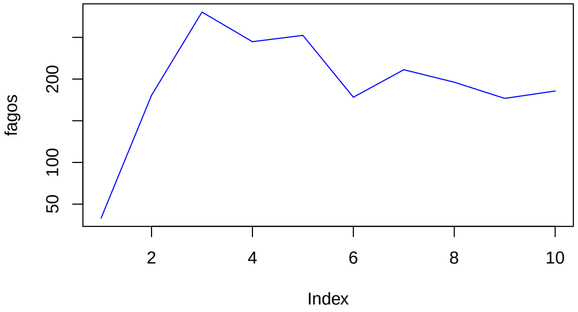
High-level plotting



High-level plotting

And if we don't want points?

```
> plot(fagos, col = "blue", type = "l")
```



High-level plotting

R

Introduction

High-level plotting commands

Low-level plotting commands

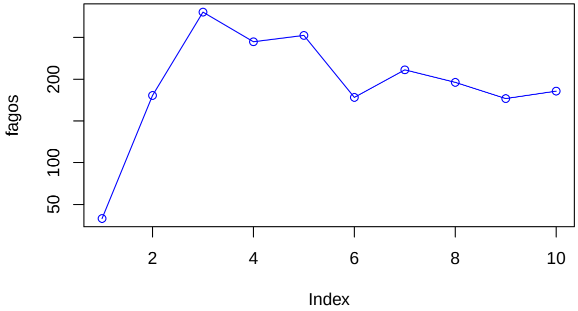
Pimp your Graph

Other graph styles

lattice()

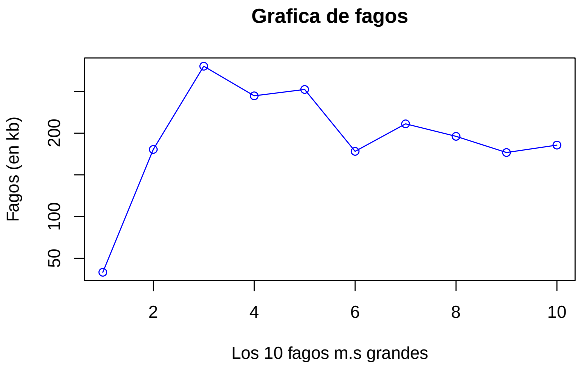
May be we want points connected with the line

```
> plot(fagos, col = "blue", type = "o")
```



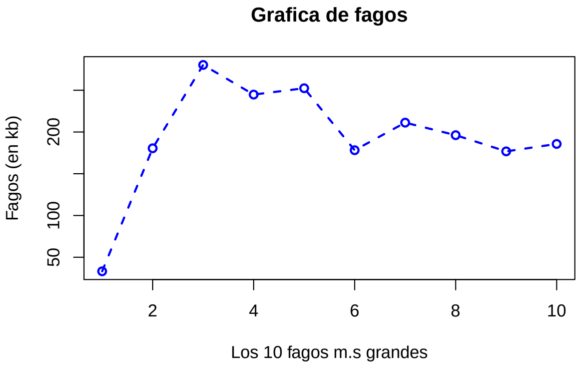
High-level plotting

How can we add the labels and title?



High-level plotting

You can also change the type of line and the width of it, how?



High-level plotting

R

Introduction

High-level plotting commands

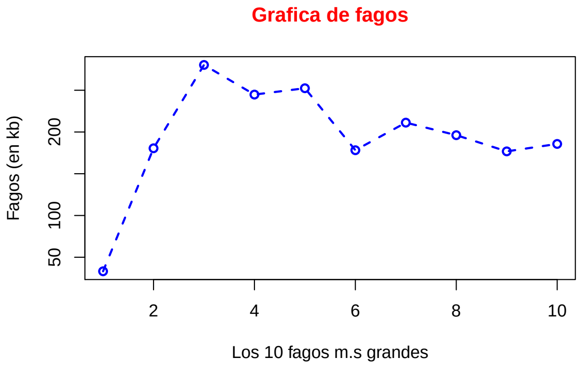
Low-level plotting commands

Pimp your Graph

Other graph styles

lattice()

Use `col.main` and change the color of the title



Low-level plotting

- Sometimes the high-level plotting functions don't produce exactly the kind of plot you desire.
- In this case, low-level plotting commands can be used to add extra information (such as points, lines or text) to the current plot.
- Low-leve command won't open the graphical device, so you might have to use a simple high-level command first.
- Some of the more useful low-level plotting functions are:

Low-level plotting

points(x, y) lines(x, y)	Adds points or connected lines to the current plot. [2]
text(x, y, labels, ...)	Add text to a plot at points given by x, y.
abline(a, b) abline(h=y) abline(v=x)	Adds a line of slope b and intercept a to the current plot
legend(x, y, legend, ...)	Adds a legend to the current plot at the specified position
title(main, sub)	Adds a title main to the top of the current plot in a large font and (optionally) a sub-title sub at the bottom in a smaller font.
axis(side, ...)	Adds an axis to the current plot on the side given by the first argument (1 to 4, counting clockwise from the bottom.)

Remember we already know how to use this

```
> x <- seq(0, 1, 0.05)
> plot(x, x, ylab = "y", type = "l")
> for (j in 2:8) {
+      lines(x, x^j)
+ }
```

Low-level plotting

Introduction

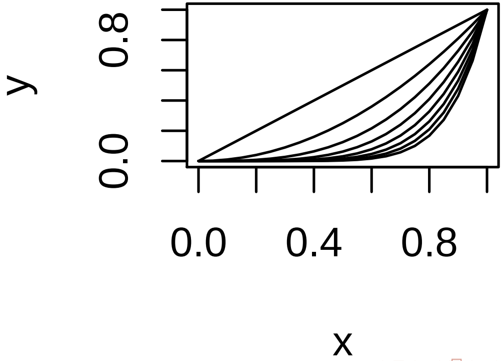
High-level plotting commands

Low-level plotting commands

Pimp your Graph

Other graph styles

lattice()



Low-level plotting

Introduction
High-level plotting commands
Low-level plotting commands
Pimp your Graph
Other graph styles
lattice()

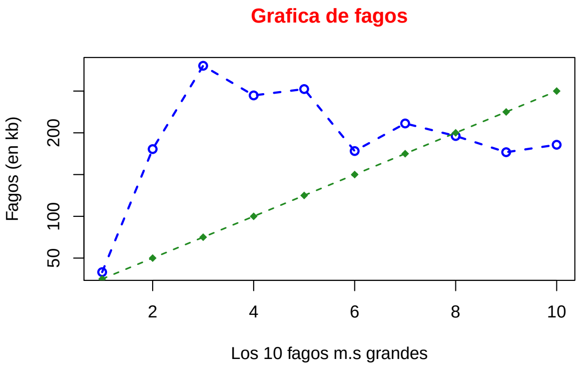
Lets take our phages graph and improve it with low-level commands.
Imagine we want to know if there is a lineal tendency of genome size between the phages.

```
> fagos2 <- seq(25, 250, by = 25)
```

```
> plot(fagos, ylab = "Fagos (en kb)",
+      main = "Grafica de fagos", xlab = "Los 10 fagos
+      col = "blue", type = "o", lty = 2,
+      lwd = 2, col.main = "red")
> lines(fagos2, col = "forest green",
+      type = "o", lty = 2, lwd = 1.5,
+      pch = 18)
```

Low-level plotting

Introduction
High-level plotting commands
Low-level plotting commands
Pimp your Graph
Other graph styles
lattice()



Low-level plotting

R

Introduction

High-level plotting commands

Low-level plotting commands

Pimp your Graph

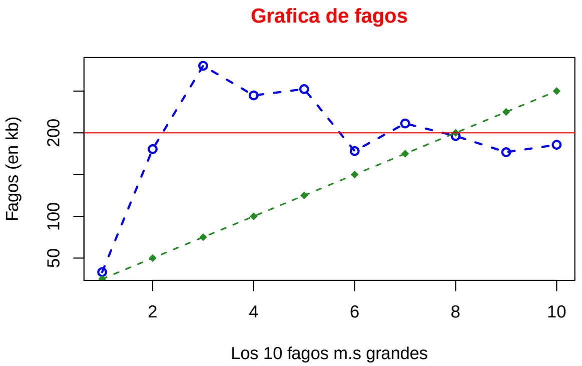
Other graph styles

lattice()

Now, we want to highlight the values higher than 200kbs

```
> plot(fagos, ylab = "Fagos (en kb)",
+       main = "Grafica de fagos", xlab = "Los 10 fagos
+       col = "blue", type = "o", lty = 2,
+       lwd = 2, col.main = "red")
> lines(fagos2, col = "forest green",
+       type = "o", lty = 2, lwd = 1.5,
+       pch = 18)
> abline(a = 200, b = 0, col = "red")
```

Low-level plotting



Graphic Parameters

There are more parameters that affect directly the graphic environment and to tune this parameters you have to use the function `par()`

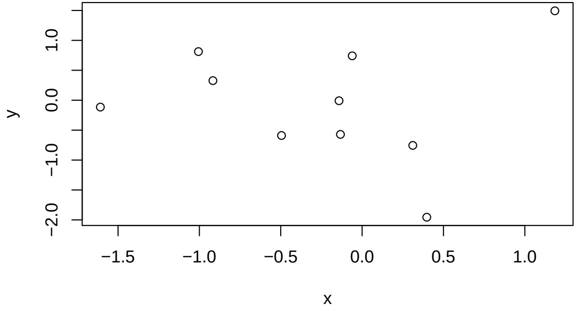
```
> par(bg = "yellow")
> `?`(par)
```

Remmember all editing of the function par() will afect all future plots, so if you acutually setted the backgoudn to yellow, all your graphs will be yellow from now on. Now lest Practice

- Create two vectors with 10 random numbers each from the normal distribution

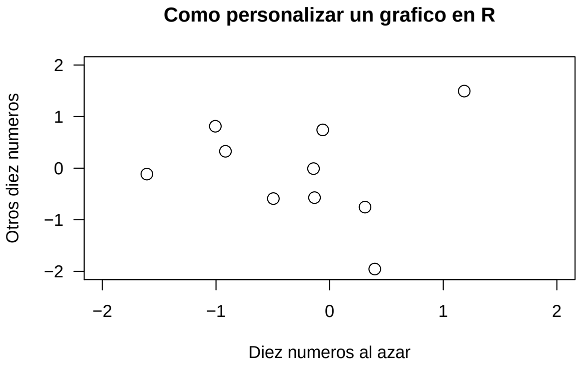
Graphic Parameters

- Plot them using one vector for x axis and the other one for the y axis



Graphic Parameters

- Now move the x and y limits, and set them between -2 and 2



Graphic Parameters

R

Introduction

High-level plotting commands

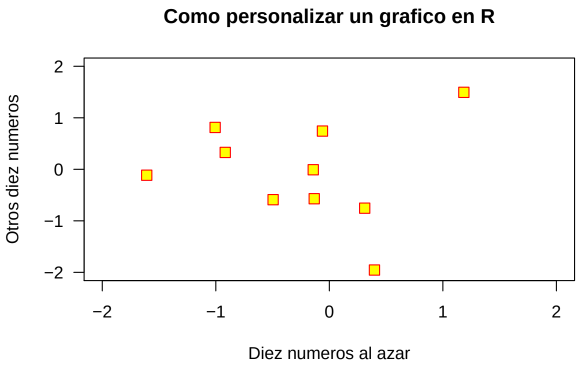
Low-level plotting commands

Pimp your Graph

Other graph styles

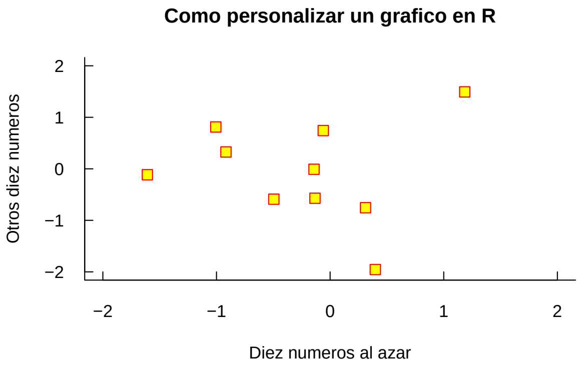
lattice()

- Now instead of boring circles lets use yellow with red squares, use parameters `pch` and `bg`



Graphic Parameters

- To edit the square drawn around the plot use `bty` and `tcl`. What are this two parameters doing?



Graphic Parameters

Introduction

High-level plotting commands

Low-level plotting commands

Pimp your Graph

Other graph styles

lattice()

- We have been using for the lasts graphs the parameters `las` and `cex`, What for?

Exercise

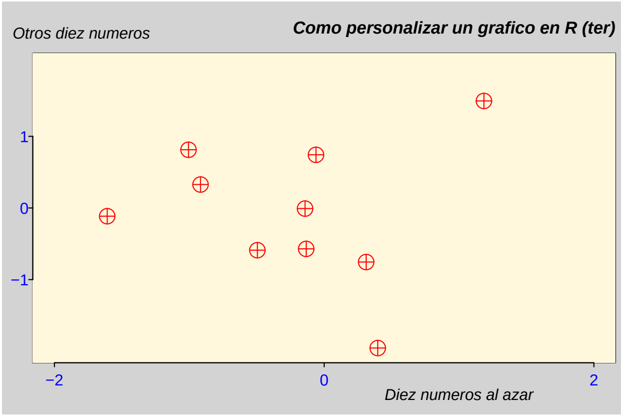
Now using parameters on command `par()` and low-level commands, re-do the graph

- with a light gray background color
- Use function `mar()` to move the margins 2.5, 1.5, 2.5, 0.25, this is South, West, North, East.
- With none black axis
- The title most be added with the lower command `title()`, so you can use a nice font color
- With `rect()` draw a square of 3x3 soraunding the backgoudn of the central area of the graph.
- Use `points()` to draw the x,y points.
- Use `axis()` to draw the axis y from -2 to 2 and the axis x from -1 and 1.

Introduction
High-level plotting commands
Low-level plotting commands
Pimp your Graph
Other graph styles
lattice()

Exercise

- Use `mtext()` to put the labels on the axis.



R

Introduction

High-level plotting commands

Low-level plotting commands

Pimp your Graph

Other graph styles

lattice()

Bar Charts

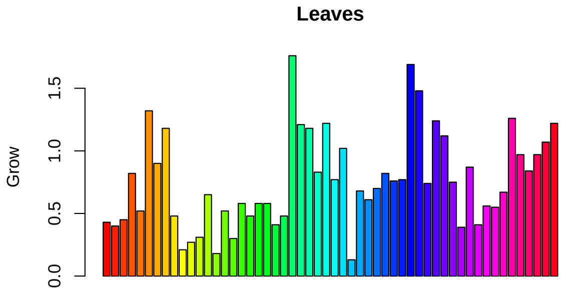
We have data of how much the leaves of a plant have grown with high CO_2

```
> workingDir <- "/Users/amedina/Documents/PHD/Sem7/Cu
> w1 <- read.csv(file = paste(sep = "",
+      workingDir, "/w1.dat"), sep = ",",
+      head = TRUE)
> names(w1)
```

```
[1] "vals"
```

```
> len <- length(as.vector(w1[, 1]))
> barplot(as.vector(w1[, 1]), main = "Leaves",
+      ylab = "Grow", col = rainbow(len))
```

Bar Charts



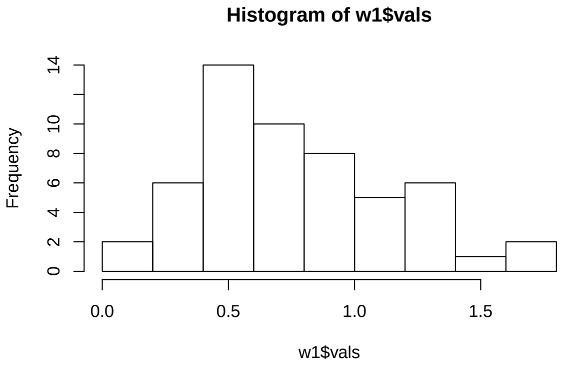
Why did we have to transform the data?

Introduction
High-level plotting commands
Low-level plotting commands
Pimp your Graph
Other graph styles
lattice()

Histogram

- There is an specific function for drawing histograms, remember that on histograms data is grouped.

  > hist(w1$vals)



R

Introduction

High-level plotting commands

Low-level plotting commands

Pimp your Graph

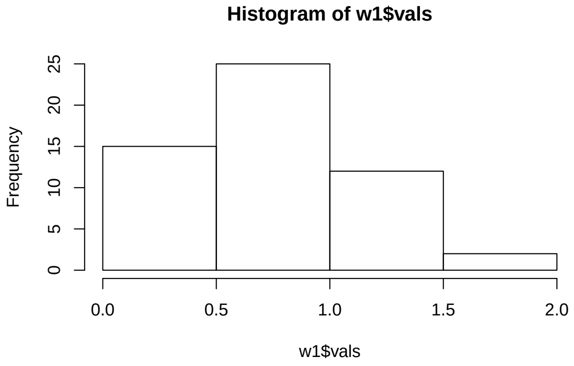
Other graph styles

lattice()

Histogram

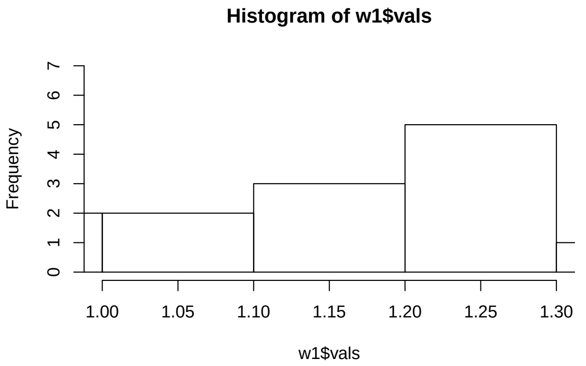
- Automatically R sets the groups, but we can specify them

```
> hist(w1$vals, breaks = 4)
```



Histogram

```
> hist(w1$vals, breaks = 12, xlim = c(1,
+      1.3))
```



- Why is the last graph incomplete?

Box Plot

R

Introduction

High-level plotting commands

Low-level plotting commands

Pimp your Graph

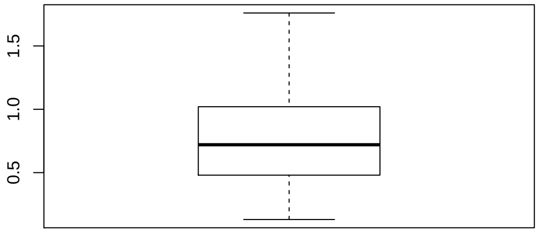
Other graph styles

lattice()

The box plot of an observation variable is a graphical representation based on its quartiles, as well as its smallest and largest values. It attempts to provide a visual shape of the data distribution. We will draw a box plot using the leaves data.

```
> boxplot(w1)
```

Box Plot



- Add a title for the plot

R

Introduction

High-level plotting commands

Low-level plotting commands

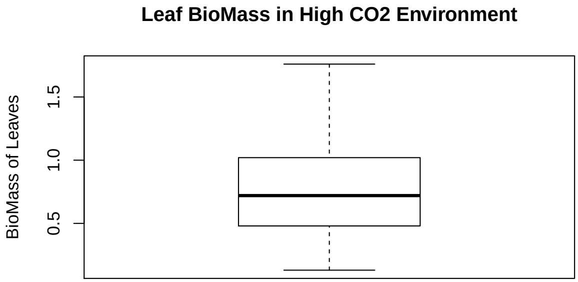
Pimp your Graph

Other graph styles

lattice()

Box Plot

- Add a legend to the y axis



Box Plot

R

Introduction

High-level plotting commands

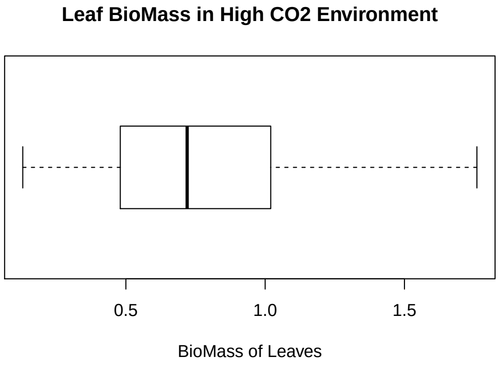
Low-level plotting commands

Pimp your Graph

Other graph styles

lattice()

- Draw the box-plot in an horizontal way



R

Introduction

High-level plotting commands

Low-level plotting commands

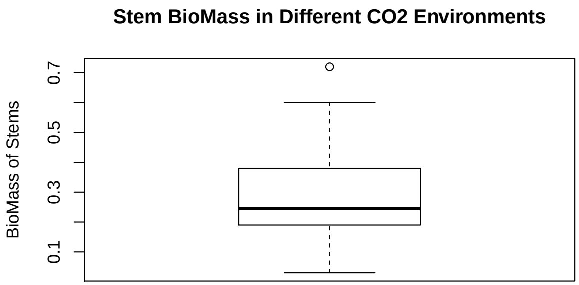
Pimp your Graph

Other graph styles

lattice()

Box Plot

- Now load the tree data and draw a box-plot with the STMB data



Lattice library

- Before describing the library, lets install it, we will also install a package for the example.

```
> install.packages("lattice")
> install.packages("mlmRev")
```

- Lattice is an add-on library for the `R` statistical computing environment.
- Provides a set of **high-level** graphic functions as alternatives to the R base graphics functions such as `plot()`, `text()`, `points()`, etc for the construction of statistical graphics (plots).

Lattice library

- High-level lattice functions like `xyplot()` are different from traditional R graphics functions in that they do not perform any plotting themselves. Instead, they return an object, of class "trellis", which has to be then print-ed or plot-ted to create the actual plot.
- It is usually not necessary to explicitly carry out the printing step, and lattice functions appear to behave like their traditional counterparts.
- The automatic plotting is suppressed when the high-level functions are called inside another function (most often `source()`) or in other contexts where automatic printing is suppressed (e.g., for or while loops). In such situations, an explicit call to `print()` or `plot()` is required.

Introduction
High-level plotting commands
Low-level plotting commands
Pimp your Graph
Other graph styles
`lattice()`

Lattice library

- The `lattice` package is based on the Grid graphics engine and requires the `grid` add-on package. One consquence of this is that it is not (readily) compatible with traditional R graphics tools.
- **In particular, changing par() settings usually has no effect on Lattice plots**; lattice provides its own interface for querying and modifying an extensive set of graphical and non-graphical settings.
- `lattice()` uses the formula syntax

Lattice library

Example

Formula syntax

$$y \sim x^{16} \tag{1}$$

$$y \sim x|z \tag{2}$$

- First formula means we will graph y according to the value in x to the 16th
- Second formula means, we will graph y according to the values in x, in diferent graphs depending on the values of Z.
- Z should be here a factor. Why?

Hands on Lattice

- Now lets load the library

```
> library(lattice)
```

- We will now load the following data and follow the BioC2008 `lattice` lab [3].

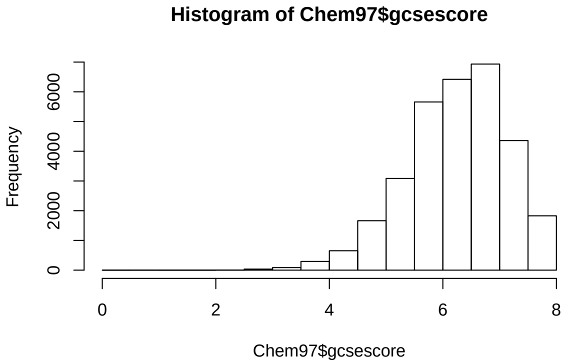
```
> data(Chem97, package = "mlmRev")
```

- Which is the class of Chem97?
- Chem97 data contains variables: `score`, `gcsescore` and `gender`
- Lets compare the usual histogram function with the `lattice` histogram function using the variable `gcsescore`

[3] Data from Leonardo Collado

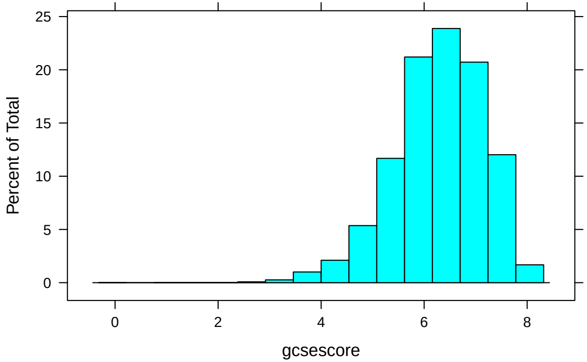
Hands on Lattice

```
> hist(Chem97$gcsescore)
```



```
> print(histogram(~gcsescore, data = Chem97))
```

Hands on Lattice



- Since the vairable `score` only contains values 0,2,4,6,8 and 10 we can use it as a `factor`

R

Introduction

High-level plotting commands

Low-level plotting commands

Pimp your Graph

Other graph styles

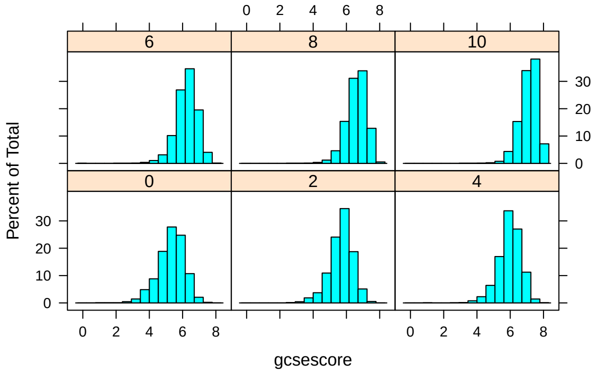
lattice()

Hands on Lattice

- So changing the `score` variable data type to `factor` we can plot the formula

$$\sim gcsescore|score \tag{3}$$

Hands on Lattice



- Now we want to add the `gender` variable, but to plot two histograms on the same graph is not trivial.
- So now we use density lines !

Hands on Lattice

R

Introduction

High-level plotting commands

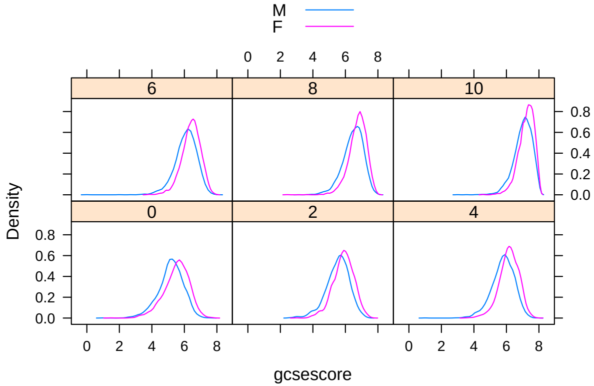
Low-level plotting commands

Pimp your Graph

Other graph styles

lattice()

```
> print(densityplot(~gcsescore | factor(score),
+       Chem97, groups = gender, plot.points = FALSE
+       auto.key = TRUE))
```



R

Introduction

High-level plotting commands

Low-level plotting commands

Pimp your Graph

Other graph styles

lattice()

Hands on Lattice

- An other grpah that is widly used to compare distributions is the Q-Q grpah.
- In statistics, a Q-Q plot (”Q”stands for quantile) is a probability plot, which is a graphical method for comparing two probability distributions by plotting their quantiles against each other.
- Now lets draw a q-q plot for our data.

```
> print(qq(gender ~ gcsescore | factor(score),
+     Chem97, f.value = ppoints(100),
+     type = c("p", "g"), aspect = 1))
```

Hands on Lattice

R

Introduction

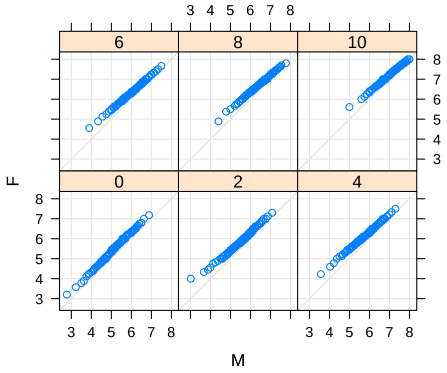
High-level plotting commands

Low-level plotting commands

Pimp your Graph

Other graph styles

`lattice()`



Introduction

High-level plotting commands

Low-level plotting commands

Pimp your Graph

Other graph styles

lattice()

Hands on Lattice

- `lattice` provides a wide gallery of plot functions, we won't see them all, we have so far checked out the very basic ones.
- If you are more interested on `lattice` graphs the book Lattice: Multivariate Data Visualization by Deepayan Sarkar can help you, there are several copies on the CCG library
- In the future you'll discover that this presentation is not enough help to draw the final figure for your paper, but you can find a lot of help in the `R Graph Gallery` , where you will find graphs you never imagin.