8.1.3 Come analizzare i dati: rappresentazione grafica dei dati

Insegnamento di Informatica

Elisabetta Ronchieri

Corso di Laurea di Economia, Universitá di Ferrara

I semestre, anno 2014-2015

Argomenti

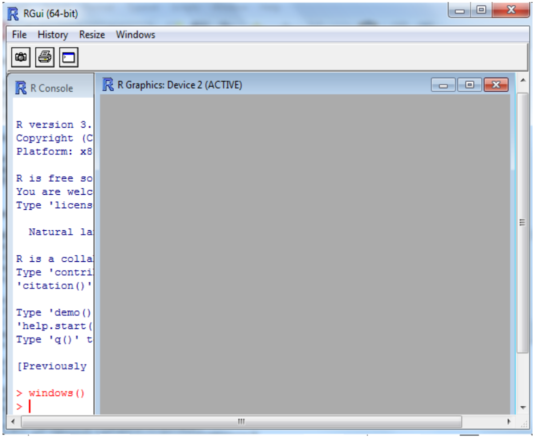
Argomenti

Plotting

- The plotting and graphics engine in R is encapsulated in a few base and recommend packages:
  - **graphics**:
    - contains plotting functions for the base graphing systems;
    - includes functions such as plot, hist, boxplot and many others.
  - **lattice**:
    - contains code for producing Trellis graphics, which are independent of the base graphics system;
    - includes functions such as xyplot, bwplot and levelplot.
  - **grid**:
    - implements a different graphing system independent of the base system;
    - the lattice package builds on top of grid.
  - **grDevices**:
    - contains all the code implementing the various graphics devices;
    - includes X11, PDF, PostScript, PNG and others.

Making a plot 1

- When making a plot one must first make a few choices not necessarily in a given order.
  - To what device will the plot be sent?
  - The default in Unix is *x*11(); on Windows it is *windows*(); on Mac OS X it is *quartz*().



```
R version 3.
Copyright (C
Platform: x8

R is free so
You are welc
Type 'licens

  Natural la

R is a colla
Type 'contri
'citation()'

Type 'demo()
'help.start(
Type 'q()' t

[Previously

> windows()
> 
```

Making a plot 2

- Plots included in a paper or presentation need to use a file device rather than a screen device.
- Base graphics are usually constructed piecemeal, with each aspect of the plot handled separately through a series of function calls.
- Lattice or grid graphics are usually created in a single function call, so all of the graphics parameters have to be specified at once.
  - Specifying everything at once allows R to automatically calculate the necessary spacings and font sizes.

Base graphics

- Base graphics are used most commonly and are a very powerful system for creating 2-D graphics.
  - Calling $plot(x, y)$ or $hist(x)$ will launch a graphics device (if one is not already open) and draw the plot on the device.
  - If the arguments to plot are not of some special class, then the default method for plot is called.
  - This function has many arguments, letting you set for example the title, x axis lable and y axis label.
  - The base graphics system has many parameters that can set and tweaked.
  - These parameters are documented in $?par$.

Base graphics parameters 1

- The *par* function is used to specify global graphics parameters that affect all plots in an R session.
- These parameters can often be overridden as arguments to specific plotting functions.
- The *par* options are:

  pch: the plotting symbol (default is open circle)
  lty: the line type (default is solid line), can be dashed, dotted, etc.
  lwd: the line width, specified as an integer multiple
  col: the plotting color, specified as a number, string, or hex code:

  - the colors function gives you a vector of colors by name

Base graphics parameters 2

- The *par* options are:

  las: the orientation of the axis labels on the plot
  bg: the background color
  mar: the margin size
  oma: the outer margin size (default is 0 for all sides)
  mfrow: number of plots per row, column (plots are filled row-wise)
  mfcol: number of plots per row, column (plots are filled column-wise)

Base graphics parameters 3

- Some default values.

```
> par("lty") #The line type
[1] "solid"
> par("lwd") #The line width
[1] 1
> par("col") #The plotting color
[1] "black"
> par("pch") #The plotting symbol
[1] 1
```

Base graphics parameters 4

- Some default values.

```
> par("bg") #The background color
[1] "transparent"
> par("mar") #THe margine size
[1] 5.1 4.1 4.1 2.1
> par("oma") #The outer margine size
[1] 0 0 0 0
> par("mfrow")
[1] 1 1
> par("mfcol")
[1] 1 1
```

Base plotting functions

- **plot**: make a scatterplot, or other type of plot depending on the class of the object being plotted.
- **lines**: add lines to a plot, given a vector x values and a corresponding vector of y values (or a 2-column matrix);
  - This function just connects the dots.
- **points**: add points to a plot.
- **text**: add text labels to a plot using specified x, y coordinates.
- **title**: add annotations to x, y axis labels, title, subtitle, outer margin.
- **mtext**: add arbitrary text to the margins (inner or outer) of the plot.
- **axis**: adding axis ticks/labels.

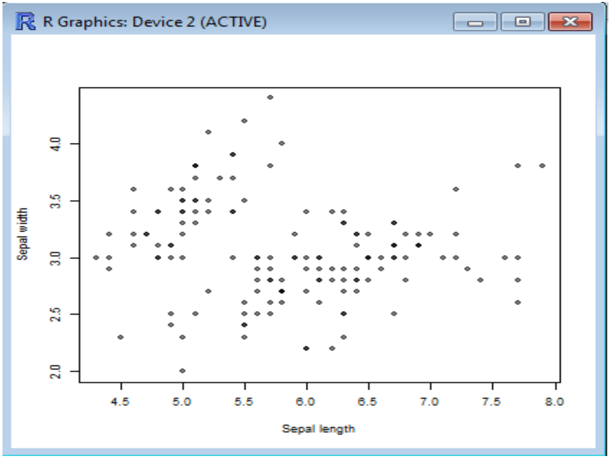
See
`http://www.statmethods.net/advgraphs/axes.html`

Plotting Example

- Let's use the *plot* function and explain a few of the options.
- The function plot uses a series of numbers for the x-values and a series of numbers for the y-values.
- The pch is set to 19.
- The colour is black with 80 percentage of opacity #00000080 to show overlapping points.
- Set axis labels.

```
> plot(iris$Sepal.Length, iris$Sepal.Width, pch = 19, col = "#00000080",
+ xlab = "Sepal length", ylab = "Sepal width")
```

Plotting Example



Graphics devices 1

- The list of devices is found in ?*Devices*.
- There are also devices created by users on CRAN.
  - **pdf**: useful for line-type graphics, vector format, resizes well, usually portable.
  - **postscript**: older format, also vector format and resizes well, usually portable, can be used to create encapsulated postscript files, Windows systems often don't have a postscript viewer.
  - **xfig**: good of you use Unix and want to edit a plot by hand.

Graphics devices 2

- The list of devices is found in ?*Devices*.
  - **png**: bitmapped format, good for line drawings or images with solid colors, uses lossless compression (like the old GIF format), most web browsers can read this format natively, good for plotting many many many points, does not resize well.
  - **jpeg**: good for photographs or natural scenes, uses lossy compression, good for plotting many many many points, does not resize well, can be read by almost any computer and any web browser, not great for line drawings
  - **bitmap**: needed to create bitmap files (png, jpeg) in certain situations (uses Ghostscript), also can be used to create a variety of other bitmapped formats not mentioned.
  - **bmp**: a native Windows bitmapped format.

Copying plots 1

- There are two basic approaches to plotting:
    1. Launch a graphics device, make a plot annotating if needed, close graphics device;
    2. Make a plot on a screen device (default) annotating if needed, copy the plot to another device if necessary (not an exact process).

Copying plots 2

- Copying a plot to another device can be useful because some plots require a lot of code and it can be a pain to type all that in again for a different device.
- Copying functions are:
  - **dev.copy**: copies a plot from one device to another.
  - **dev.copy2pdf**: copies a plot to a Portable Document Format (PDF) file.
  - **dev.list**: shows the list of open graphics devices.
  - **dev.next**: switches control to the next graphics device on the device list.
  - **dev.set**: sets control to a specific graphics device.
  - **dev.off**: closes the current graphics device.

See
http://www.stat.berkeley.edu/~s133/saving.html

Argomenti

Lattice functions

- **xyplot**: this is the main function for creating scatterplots.
- **bwplot**: box-and-whiskers plots (boxplots).
- **histogram**: histograms.
- **stripplot**: like a boxplot but with actual points.
- **dotplot**: plot dots on violin strings.
- **splom**: scatterplot matrix;
  - like **pairs** in base graphics system.
- **levelplot**, **contourplot**: for plotting image data.

See
`http://www.statmethods.net/advgraphs/trellis.html`

Lattice functions

- Lattice functions generally take a formula for their first argument, usually of the form:

  ```
  y ~ x | f * g
  ```

  - On the left of the $\sim$ is the y variable, on the right is the x variable
  - After the $|$ are conditioning variables - they are optional:
    - the $*$ indicates an interaction
- The second argument is the data frame or list from which the variables in the formula should be obtained.
  - If no data frame or list is passed, then the parent frame is used.
  - If no other arguments are passed, there are defaults that can be used.

Lattice behavior

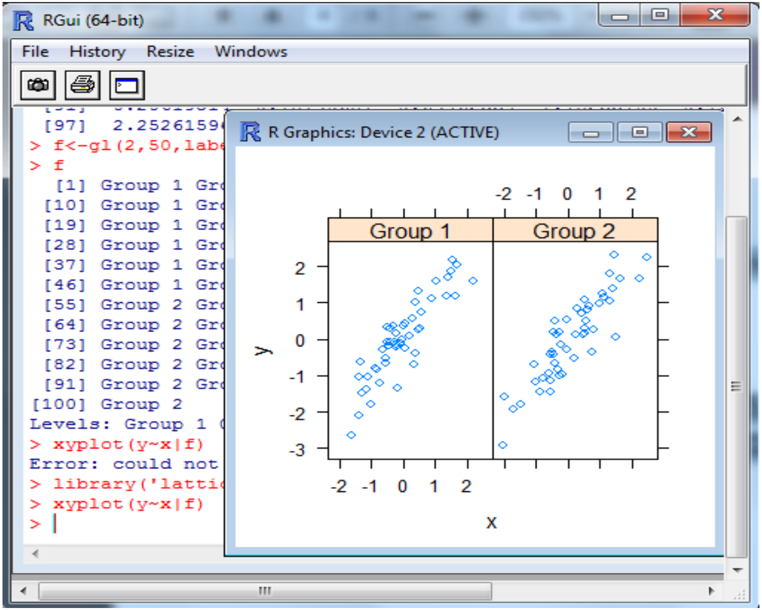
- Lattice functions behave differently from base graphics functions in one critical way.
- Base graphics functions plot data directly the graphics device
- Lattice graphics functions return an object of class *trellis*.
- The print methods for lattice functions actually do the work of plotting the data on the graphics device.
- Lattice functions return **plot objects** that can, in principle, be stored (but it's usually better to just save the code + data).
- On the command line, *trellis* objects are auto-printed so that it appears the function is plotting the data

Lattice Panel Functions 1

- Lattice functions have a panel function which controls what happens inside each panel of the entire plot.
- xyplot plots y vs. x conditioned on f.

```
> x <- rnorm(100)
> y <- x + rnorm(100, sd = 0.5)
> f <- gl(2, 50, labels = c("Group 1", "Group 2"))
> xyplot(y ~ x | f)
```

Lattice Panel Functions 1



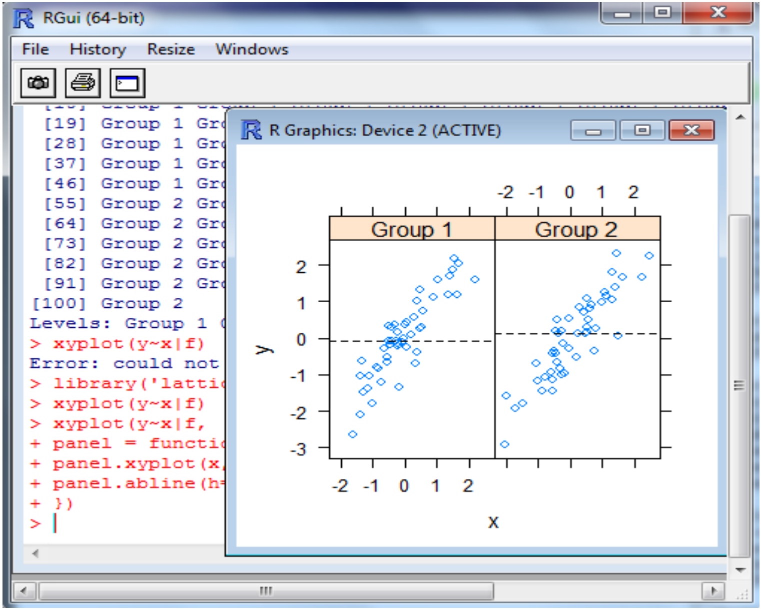
```
 [97]  2.2526159
> f<-gl(2,50,labe
> f
  [1] Group 1 Gr
 [10] Group 1 Gr
 [19] Group 1 Gr
 [28] Group 1 Gr
 [37] Group 1 Gr
 [46] Group 1 Gr
 [55] Group 2 Gr
 [64] Group 2 Gr
 [73] Group 2 Gr
 [82] Group 2 Gr
 [91] Group 2 Gr
[100] Group 2
Levels: Group 1
> xyplot(y~x|f)
Error: could not
> library('latti
> xyplot(y~x|f)
> |
```

Lattice Panel Functions 2

- Adding an horizontal dashed line.
- xyplot plots y vs. x conditioned on f with horizontal (dashed) line drawn at the median of y for each panel.

```
> xyplot(y~x|f,
+ panel = function(x,y,...) {
+ panel.xyplot(x,y,...)
+ panel.abline(h=median(y), lty=2)
+ })
```

Lattice Panel Functions 2



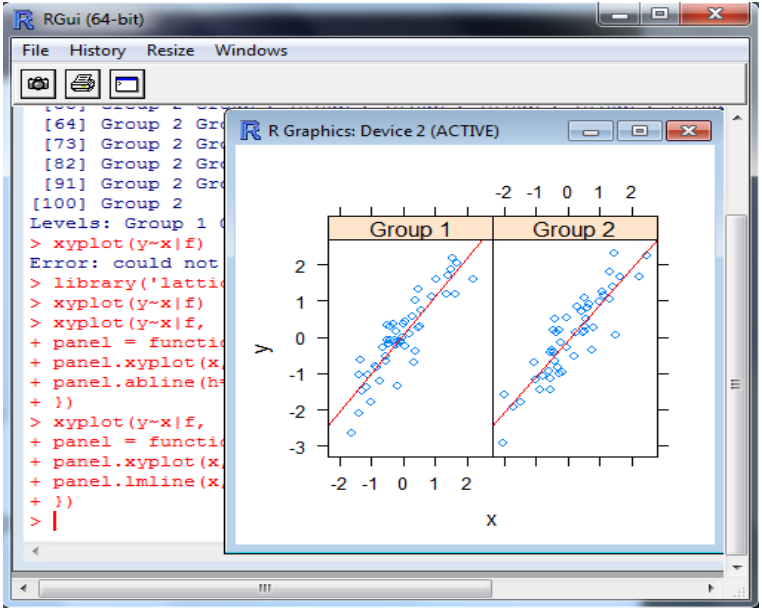
```
 [19] Group 1 Gr
 [28] Group 1 Gr
 [37] Group 1 Gr
 [46] Group 1 Gr
 [55] Group 2 Gr
 [64] Group 2 Gr
 [73] Group 2 Gr
 [82] Group 2 Gr
 [91] Group 2 Gr
[100] Group 2
Levels: Group 1 (
> xyplot(y~x|f)
Error: could not
> library('latti
> xyplot(y~x|f)
> xyplot(y~x|f,
+ panel = functi
+ panel.xyplot(x,
+ panel.abline(h=
+ })
>
```

Lattice Panel Functions 3

- Adding a regression line
- xyplot fits and plots a simple linear regression line to each panel of the plot.

```
> xyplot(y~x|f,
+ panel = function(x,y,...) {
+ panel.xyplot(x,y,...)
+ panel.lmline(x,y,col=2)
+ })
```

Lattice Panel Functions 3



```
 [64] Group 2 Gro
 [73] Group 2 Gro
 [82] Group 2 Gro
 [91] Group 2 Gro
[100] Group 2
Levels: Group 1 G
> xyplot(y~x|f)
Error: could not
> library('lattic
> xyplot(y~x|f)
> xyplot(y~x|f,
+ panel = functio
+ panel.xyplot(x,
+ panel.abline(h=
+ })
> xyplot(y~x|f,
+ panel = functio
+ panel.xyplot(x,
+ panel.lmline(x,
+ })
> |
```

Argomenti

Mathematical Annotation 1

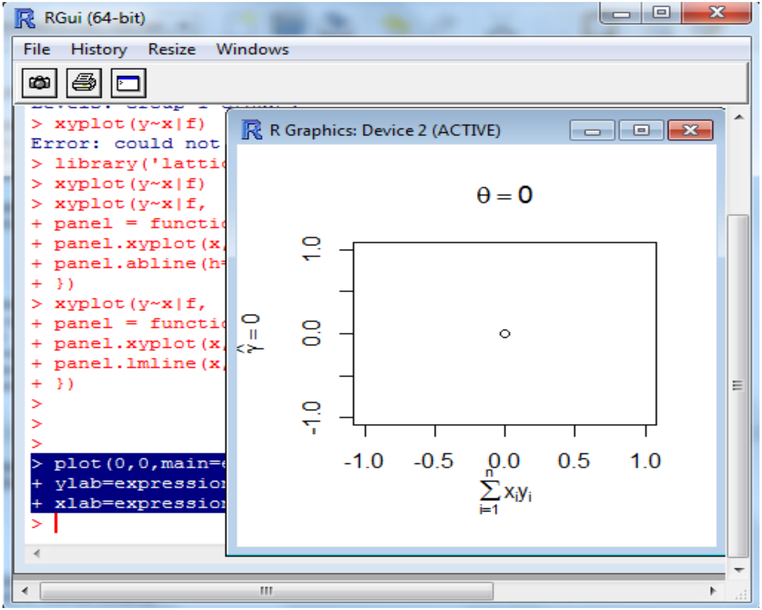
- R can produce LaTeX-like symbols on a plot for mathematical annotation. This is very handy and is useful for making fun of people who use other statistical packages.
- Math symbols are expressions in R and need to be wrapped in the expression function.
- There is a set list of allowed symbols and this is documented in ?*plotmath*
- Plotting functions that take arguments for text generally allow expressions for math symbols.

Mathematical Annotation 2

- Adding the symbol θ as title, γ on the ylabel and a formula on the xlabel.

```
> plot(0,0,main=expression(theta==0),
+ ylab=expression(hat(gamma)==0),
+ xlab=expression(sum(x[i]*y[i],i==1,n)))
```

Mathematical Annotation 2



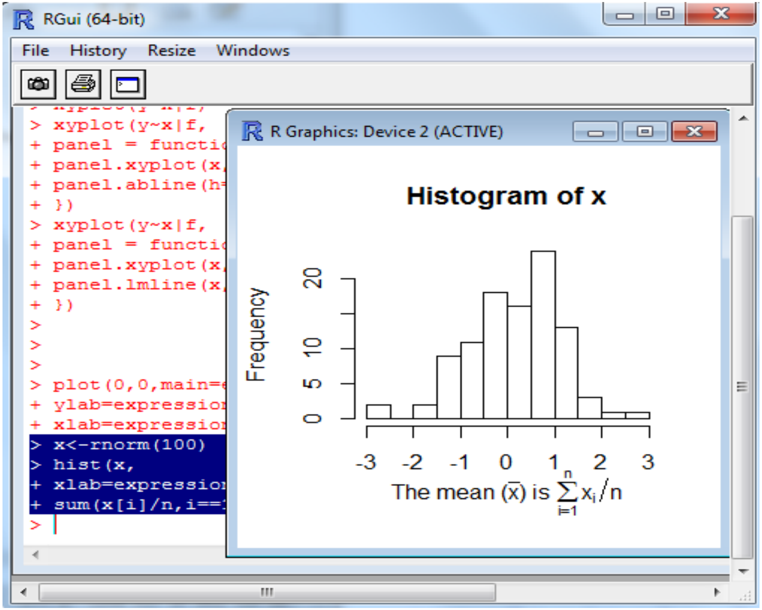
```
> xyplot(y~x|f)
Error: could not
> library('lattic
> xyplot(y~x|f)
> xyplot(y~x|f,
+ panel = functio
+ panel.xyplot(x,
+ panel.abline(h=
+ })
> xyplot(y~x|f,
+ panel = functio
+ panel.xyplot(x,
+ panel.lmline(x,
+ })
>
>
>
> plot(0,0,main=e
+ ylab=expressio
+ xlab=expressio
>
```

Mathematical Annotation 3

- Pasting string together.

```
> x<-rnorm(100)
> hist(x,
+ xlab=expression("The mean ("* bar(x) *") is " *
+ sum(x[i]/n,i==1,n)))
```

Mathematical Annotation 3



```
> xyplot(y~x|f,
+ panel = functi
+ panel.xyplot(x,
+ panel.abline(h=
+ })
> xyplot(y~x|f,
+ panel = functi
+ panel.xyplot(x,
+ panel.lmline(x,
+ })
>
>
>
> plot(0,0,main=e
+ ylab=expressior
+ xlab=expressior
> x<-rnorm(100)
> hist(x,
+ xlab=expressior
+ sum(x[i]/n,i==1
>
```

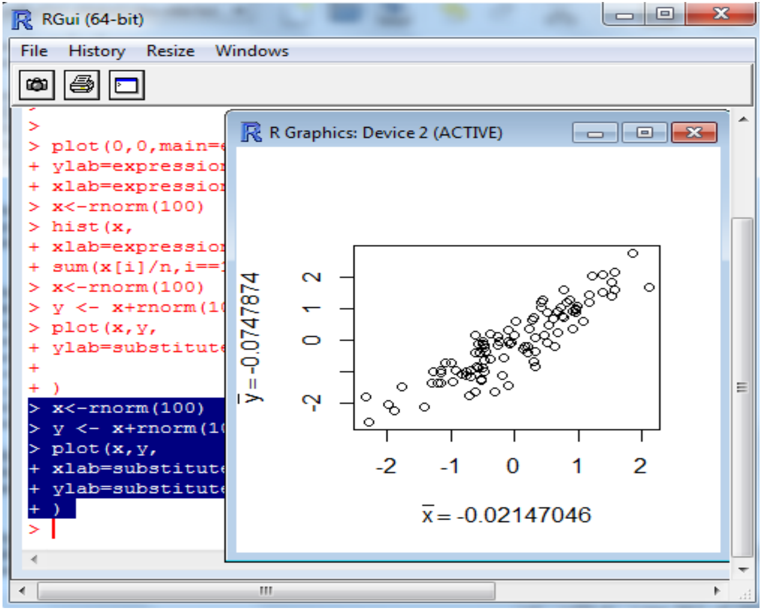
RGui (64-bit)
File
History
Resize
Windows
R Graphics: Device 2 (ACTIVE)
Histogram of x
Frequency
0
5
10
20
-3
-2
-1
0
1
2
3
The mean $(\bar{x})$ is $\sum_{i=1}^{n} x_i / n$

Substituting 1

- What if you want to use a computed value in the annotation?

```
> x<-rnorm(100)
> y <- x+rnorm(100,sd=0.5)
> plot(x,y,
+ xlab=substitute(bar(x)==k,list(k=mean(x))),
+ ylab=substitute(bar(y)==k,list(k=mean(y)))
+ )
```

Substituting 1



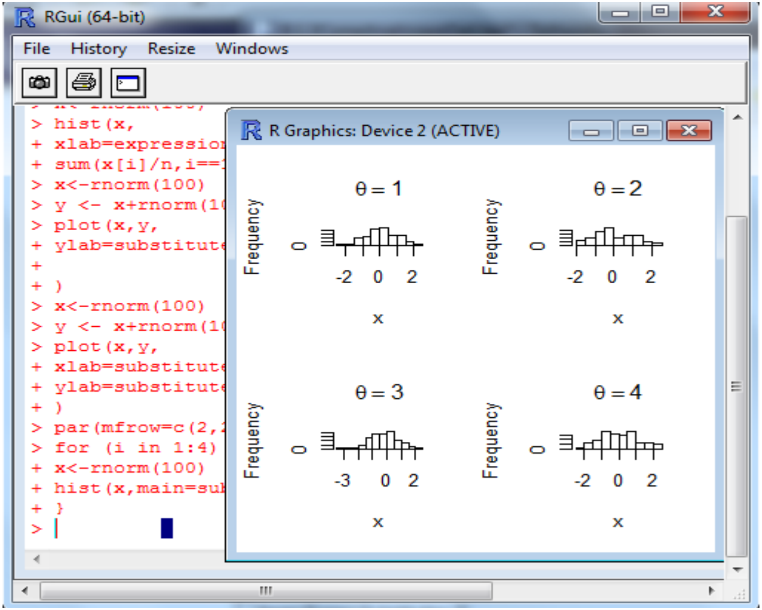
```
>
> plot(0,0,main=
+ ylab=expressio
+ xlab=expressio
> x<-rnorm(100)
> hist(x,
+ xlab=expressio
+ sum(x[i]/n,i==
> x<-rnorm(100)
> y <- x+rnorm(1
> plot(x,y,
+ ylab=substitute
+
+ )
> x<-rnorm(100)
> y <- x+rnorm(1
> plot(x,y,
+ xlab=substitute
+ ylab=substitute
+ )
>
```

Substituting 2

- What if you want to use a computed value in a loop of plots?

```
> par(mfrow=c(2,2))
> for (i in 1:4) {
+ x<-rnorm(100)
+ hist(x,main=substitute(theta==num,list(num=i)))
+ }
```

Substituting 2



```
> hist(x,
+ xlab=expressio
+ sum(x[i]/n,i==
> x<-rnorm(100)
> y <- x+rnorm(1
> plot(x,y,
+ ylab=substitute
+
+ )
> x<-rnorm(100)
> y <- x+rnorm(1
> plot(x,y,
+ xlab=substitute
+ ylab=substitute
+ )
> par(mfrow=c(2,
> for (i in 1:4)
+ x<-rnorm(100)
+ hist(x,main=sub
+ }
>
```

Help pages

- ?*par*
- ?*plot*
- ?*xyplot*
- ?*plotmath*
- ?*axis*

Argomenti

Basic plotting in R

- Convenient.
- Can't go back once plot has started.
- Plot is just a series of R commands.

Lattice plotting in R

- Plot are created with a single function call.
- Most useful for conditioning types of plots.
- Good for putting many plots on a screen.
- Annotation in plot is not intuitive.
- Cannot add to the plot once it has created.

ggplot2

- Split the difference between base graphics and lattice.
- Automatically deals with spacings, text, titles, but also allows you to annotate by adding.
- Superficial similarity to lattice but generally more intuitive to use.
- Default mode makes many choices for you.
- Available from CRAN via the function *install.packages*.

See
http://ggplot2.org

qplot

- Works much like the *plot* function in base graphics package.
- Looks for data in a data frame, similar to lattice or in the parent environment.
- Plots are made up of aesthetics (size, shape, color) and geoms (points, lines).
- Factors are important for indicating subsets of the data.
- qplot hides what goes on underneath.
- ggplot is the core function and very flexible for doing things qplot cannot do.

qlot example

Try to run the following piece of code

```
> library(ggplot2)
> str(mpg)
> qplot(displ,hwy,data=mpg)
```

Modifying aesthetics

```
> qplot(displ,hwy,data=mpg, color=drv)
```

Adding a geom

```
> qplot(displ,hwy,data=mpg, color=drv, geom=c("point","smooth"))
```

Changing graphic to have histograms

```
> qplot(displ,hwy,data=mpg, fill=drv)
```

qlot example

Handling facets

```
> library(ggplot2)
> str(mpg)
> qplot(displ,hwy,data=mpg, facets=.~drv)
> qplot(hwy,data=mpg, facets=drv~., binwidth=2)
> qplot(hwy,data=mpg, method='lm', facets=drv~., binwidth=2)
```

Basic components of a ggplot2 plot

- A data frame.
- Aesthetic mappings.
- Geoms for points, lines and shapes.
- Facets for conditional plots.
- Stats for statistical transformations.
- Scales.
- Coordinate system.

See
`http://ggplot2.org`

ggplot example

Try to run the following piece of code

```
> library(ggplot2)
> str(maacs)
> g<-ggplot(maacs,aes(logpm25,NocturnalSympt)) %Initial call
> summary(g)
> print(g)
> p<-g+geom_point() %save and print ggplot object
> print(p)
> g+geom_point() % plot object without saving
> g+geom_point()+geom_smooth()
> g+geom_point()+geom_smooth()+faced_grid(.~bmicat)
```

Per ulteriori letture

Hadley Wickham, *Package ggplot2*, 2014, `http://cran.r-project.org/web/packages/ggplot2/ggplot2.pdf`