Applied Statistics: R

Semester 2018-I

Francisco Rosales Marticorena

Faculty of Economics & Finance

Table of Contents:

1 Introduction to R

2 Advanced Graphics

Introduction to R

Plan

Week 1:

1. R basics, GitHub and $\LaTeX$
2. graphical methods in R

Week 2:

1. data management (import/export)
2. functions, debugging, condition handling

Week 3:

1. profiling, performance and parallelization
2. numeric methods and simulations

If possible there is time: Rcpp, building packages.

Classes and Coordination

Classes:

- One exam on mid-term week. 20% of your grade.
- More an R course than an "applied stats" course.
- 3 weeks, 5 hours $\times$ week.
- TA comes the 3rd hour of the three-hour block.

Coordination:

- No Blackboard.
- Course repository here.
- Use the `Homework` file to upload solutions of exercises.

Reading Material

- Hadley Wickham. Advanced R. The R series. CRC Press, 2015. Available online: `http://adv-r.had.co.nz/`.
- Owen Jones, Robert Maillardet, and Andrew Robinson. Introduction to scientific programming and simulation using R. Chapman & Hall/CRC, 2009.
- Brian Everitt and Torsten Hothorn. A handbook of statistical analyses using R. Chapman & Hall/CRC, 2006.
- R Data Import/Export manual. Available online: `http://cran.r-project.org/doc/manuals/r-release/R-data.pdf`
- Deepayan Sarkar. Lattice: multivariate data visualization with R. Use R! Springer, 2008.
- Roger S Bivand, Edzer J Pebesma, and Virgilio Gómez-Rubio. Applied spatial data analysis with R, volume 10 of Use R! Springer, 2008.

Who are you?

- Ever used R?
- Text editor for R: Rstudio, emacs, Vim?
- Github, LaTeX?
- Operating system: Linux, Apple, Windows?
- Other programming languages: C, C++, Java, Python, Fortran, Julia, Matlab, Mathematica?

R-Ladies Announcement

Heard of Laboratoria? kind of like that...

- Worldwide organization.
- Empowers women trough coding.
- Third meet up on April 27th at UP Aula Magna H 15:00.
- Topic: Text-mining in Finance with R.
- Speaker: Leda Basombrío, Strategic Analysis Manager at BCP.

Not mandatory, but recommended. Not only for women.

R Basics

help:	`?topic, help(topic), args(some function)`
assignments:	`x <- 5, x = 5, 5 -> x`
operators:	`+, -, *, /, ^, &, &&, |, ||; see help("+")`
comparisons:	`==, !=, >, >=, <, <=; see help("==")`
loops:	`for, while, repeat`
comments:	`everything that follows #`

case sensitive: usage of CAPITAL and small letters matters!

Basic Data Structures

numeric ($\mathbb{R}$):	`1, 301L, .141, 1.23e-3, NaN, Inf, -Inf`
complex ($\mathbb{C} \setminus \mathbb{R}$)	`1+0i, 1i, 3+5i`
logical:	`TRUE, FALSE, NA`
character:	`"hello", "I'm a string"`
numeric:	no distinction between integers and doubles
missing:	stored as `NA`, and are logical.

Can check the basic structure with `str()`, `mode()` or `storage.mode()`.

```
str(1)                ## num 1
mode(1)               ## [1] "numeric"
storage.mode(1)       ## [1] "double"
storage.mode(1L)      ## [1] "integer"
mode(pi)              ## [1] "numeric"
storage.mode(pi)      ## [1] "double"
str(LETTERS)          ## chr [1:26] "A" "B" "C" "D" "E" ...
```

Construction & Coercion

- vector can be constructed with `c()`
- coerced to a type `xxx` by `as.xxx()`
- when combining different data types, they will be coerced to the most flexible type
- coercion often happens automatically
- check if `xxx` is a specific type by `is.xxx()`

```
storage.mode(c(1,2L))      ## [1] "double"
storage.mode(c("a",1))     ## [1] "character"
x <- c(FALSE, TRUE, NA)
as.numeric(x)              ## [1] 0 1 NA
# Total number of TRUEs
sum(x, na.rm = TRUE)       ## [1] NA
```

NA, NaN and NULL

In R, there three ways to represent “nothing”, but the reason for the missingness of the information can be distinguished:

`NA`	missing sample values
`NaN`	wrong math, e.g. `log(-1), 1/0`)
`NULL`	null pointer

```
c(3, NA)     ## [1] 3 NA
c(3, 0/0)    ## [1] 3 NaN
c(3, NULL)   ## [1] 3
max(3, NA)   ## [1] NA
```

Inifinity

Some math operations can be performed with `Inf` and `-Inf`:

```
max(3, Inf)
## [1] Inf
```

```
min(3, Inf)
## [1] 3
```

```
c(Inf + Inf, (-Inf) * Inf, Inf - Inf)
## [1] Inf -Inf NaN
```

Convoluted Data Structures

- data structures in R can be organized by their dimensionality and if all their contents are of the same type (or not):

object	dimension	homogeneous	heterogeneous
1d	`length()`	atomic vector	list
2d	`dim()`	matrix	data frame
nd	`dim()`	array	–

- a `data.frame` is a matrix with different data type columns.
- a `list` can have different data type elements.

```
x <- matrix(1, nrow = 5, ncol = 2)
is.matrix(x)            ## [1] TRUE
as.vector(x)            ## [1] 1 1 1 1 1 1 1 1 1 1
x <- list(a = "Hallo", b = 1:10, pi = pi)
```

Attributes

All objects can have arbitrary additional attributes, used to store metadata about the object.

- can be thought of as a named list (with unique names); other frequently encountered attributes: "dimnames", "names", "class"(!)
- can be accessed all at once (as a list) with `attributes()`, or individually with `attr()`.
- arrays are simply vectors with a "dim" attribute.
- factor is a vector with the "levels" attribute
- `as.xxx()` functions delete all attributes including dimensionality

Attributes

```
x <- matrix(1:10, ncol = 5)
attributes(x)                 ## $dim
                              ## [1] 2 5
rownames(x) <- c("Eins", "Zwei")
attributes(x)                 ## $dim
                              ## [1] 2 5
                              ## $dimnames
                              ## $dimnames[[1]]
                              ## [1] "Eins" "Zwei"
                              ## $dimnames[[2]]
                              ## NULL
as.character(x)               ## [1] "1"  "2"  "3"  "4"  "5"...
attributes(as.character(x))   ## NULL
```

Subsetting

- There are three subsetting operators: `[`, `[[`, and `$`
- the three types of subsetting:
  - Positive integers return elements at the specified positions.
  - Logical vectors select elements where the corresponding logical value is `TRUE`; application of logical expressions.
  - character vectors to return elements with matching names.
- important differences in behaviour of different objects (e.g., vectors, lists, factors, matrices, and data frames).
- More advanced subsetting, in particular in combination with convoluted logical expressions, can be done using the functions `subset()` and `which()`.
- The default `drop=TRUE` simplifies the data type of the result.

Atomic Vectors

Use "["-operator and number, logical vector or name of the element you want to pull out.

```
x <- c(2.1, 4.2, 3.3, 5.4)
x[c(3, 1)]                         ## [1] 3.3 2.1
x[-c(3, 1)]                        ## [1] 4.2 5.4
x[c(TRUE, TRUE, FALSE, FALSE)]     ## [1] 2.1 4.2

(y <- setNames(x, letters[1:4]))   ##   a   b   c   d
                                   ## 2.1 4.2 3.3 5.4
y[c("d", "c", "a")]                ##   d   c   a
                                   ## 5.4 3.3 2.1
```

Matrices & Arrays

Subsetting matrices and arrays with “[”-operator like vectors, while the dimension is separated by comma:

```
x <- matrix(1:10, ncol=2)
colnames(x) <- c("Eins", "Zwei")

x[1:2,]                      # results a matrix
x[,"Zwei"]                   # results a vector
x[,"Zwei", drop = FALSE] # results a matrix
x[-3,]                       # everything, but not third row
```

Lists

Subsetting lists with “[”-operator returns always a list, while [[, and $ pull out elements of the list:

```
x <- list(a = "Hallo", b = 1:10, pi = pi)

x$a       # first element of the list
x[['a']]
x[[1]]
x[1]      # list with one element
x[2:3]    # list with two elements
x[[2:3]]  # wrong result
```

Data Frames

Data frames possess the characteristics of both lists and matrices: if you subset with a single vector, they behave like lists; if you subset with two vectors, they behave like matrices.

```
iris[1:10,]               # data frame with 10 rows
iris[,1]                  # numerical
iris$Sepal.Length         # the name
iris$Sepa1.Length         # Oops! what happened?
iris[,"Sepal.Length"]     # again first column
iris[,"Sepa1.Length"]     # Error:  undefined columns selected
iris[,1, drop = FALSE]    # data frame with one column
```

Flow Control

- conditional evaluation: `if`, `else`, `ifelse`
- loops: `for`, `while`, `repeat`, `switch`

basic vocabulary:

```
if, &&, ||
for, while, repeat
next, break
switch
ifelse
```

if/else Conditions

```
if (<test>) {
  <expression1>
} else {
  <expression2>
}
```

- else block is optional
- <test> has to result in a value that is TRUE or FALSE
- only the first element of <test> is used, ow a warning is triggered
- to eval more than one statement use &, | or all() and any()
- can be nested

`for` Loop

```
for (<var> in <vector>) {
  <expression>
}
```

```
sum <- 0
for(i in 1 : length(x)) {
  sum <- sum + x[i]
}
```

```
sum <- 0
for(x_value in x) {   ## more efficient
  sum <- sum + x_value
}
```

Use `seq_along(x)` instead of `1:length(x)`

`while` Loop

```
while(<test>) {
  <expression>
}
```

E.g., the sum until the first `NA`:

```
sum <- 0
i <- 1
while((i <= length(x)) && !is.na(x[i])) {
  sum <- sum + x[i]
  i <- i + 1
}
```

Be aware of infinite loops!

next & break

- `next` jumps to the next iteration in `for` or `while` loops
- `break` terminates `for` or `while` loops.

```
x <- c(1, 1, 1, NA, 2)
sum <- 0
for(val in x) {
  if(is.na(val)) break
  sum <- sum + val
}

x <- c(1, 1, 1, NA, 2)
sum <- 0
for(val in x) {
  if(is.na(val)) next
  sum <- sum + val
}
```

Style Example: Bad

```
fWLM<-function(y,X_mat,w){T0<-t(X_mat)%*%diag(w)%*%X_mat
t<-system.time({t_1<-solve(T0)%*%t(X_mat)%*%(w*y);
t2<-X_mat%*%t_1})
return(list(beta=t_1,hat=t2,stddev=sqrt(sum(w*(t2-y)^2))/
(length(y)-ncol(X_mat)), wts=w,t=t[[3]]))}
```

Style Example: Good

```
fit_weighted_lm <- function(response, design, weights) {
  n_obs <- length(response)
  n_coef <- ncol(design)
  time_start <- Sys.time()
  wcrossprod_design <- crossprod(design * weights, design)
  weighted_response <- weights * response
  coef <- solve(wcrossprod_design, t(design) %*% weighted_response)
  time <- Sys.time() - time_start
  fitted <- design %*% coef
  residuals <- response - fitted
  weighted_rss <- sum(weights * residuals^2)
  sd_resid <- sqrt(weighted_rss / (n_obs - n_coef))
  list(coef      = coef,
      fitted     = fitted,
      sd_resid   = sd_resid,
      weights    = weights,
      time       = time)
}
```

Notation & Names

- find meaningful file names; if files need to be run in sequence, prefix them with numbers.

  ```
  0-download.R
  1-parse.R
  2-explore.R
  ```

- avoid uppercase
- use an underscore to separate words within a name
- use nouns for variable names and verbs for function names

Notation & Names

- strive for names that are concise and meaningful (this is not easy!).

```
# Good       # Bad
day_one      first_day_of_the_month
day_1        dayone
             djm1
```

- avoid using names of existing functions and variables.

```
# Bad
T <- FALSE
c <- 10
t <- temporal_variable
mean <- function(x) sum(x)
```

Formatting

- strive to limit your code to 80 characters per line.
- when indenting your code, use two spaces. Never use tabs.
- place spaces around all infix operators (=, +, -, <-, etc.), before parentheses, and after comma (just like in regular English)

```
# Good
average <- mean(feet / 12 + inches, na.rm = TRUE)
# Bad
average<-mean(feet/12+inches,na.rm=TRUE)

# Good
if (debug) do(x)
plot(x, y)
# Bad
if(debug)do(x)
plot (x, y)
```

Formatting

- an opening (or closing) curly brace should always be followed by a new line, unless it's followed by `else`.

```
 if (y == 0) {
  log(x)
} else {
  y^x
}
```

- use commented lines of - and = to break up your file into chunks.

```
# Load data ---------------------------
# Plot data ---------------------------
```

- `formatR::tidy_source(source="input.R",file="output.R")` cleans up and does some automatic formatting

Advanced Graphics

Motivation

- creating interest and attention of the reader
- essential meaning can be visualized at a glance
- comprehensive picture of a problem gives more complete and balanced understanding
- the human visual system is very powerful in detecting patterns: outliers, diagnose models, search for perhaps unexpected phenomena

Graphical Devices

- A graphical device can be thought as a paper on which you can draw with different pens and colours, but nothing can be deleted.
- It can be opened more than one device, but there is only one active.
- There is no difference no matter which device is used.
- Typical steps to produce a graphic is:
  1. start device, e.g. `pdf('testgraphics.pdf')`
  2. generate graphic, e.g. `plot(1:10)`
  3. close device: `dev.off()`
- If no device is open, using a high-level graphics function will cause a device to be opened.

Graphical Devices

The following graphics devices are currently available:

- `pdf()`: write PDF graphics commands to a file; can be handy for distribution to cooperation partners, integration in PDFLaTeX, or viewing many graphics
- `postscript()`: writes PostScript graphics commands to a file
- `bitmap()`: bitmap pseudo-device via 'Ghostscript' (if available).

Interactive plotting with GUI:

- `x11()`: The graphics device for the X11 windowing system
- `png()`: compressed Bitmap, without loss
- `jpeg()`: compressed Bitmap with information loss, optimized for pictures with many color shades

For more info see `?Devices`.

High-level Plots

High-level functions generate/initialize a graphic, e.g.:

`plot()`	depend of context
`barplot()`	Barplot
`boxplot()`	Boxplot
`coplot ()`	Conditioning plots
`contour()`	Contour line plot
`curve()`	Plotting functions
`dotchart()`	Dot Plots
`hist()`	Histogram
`image()`	Countour Plot (3. Dim. as color)
`mosaicplot()`	Mosaicplots (categorial data)
`pairs()`	Scatterplot matrix
`persp()`	perspective surface
`qqplot()`	QQ-Plot

High-level Plots

Many functions can be applied to different object types. They react in a "intelligent" way, so that a meaningful graphic can be found.

```
plot(trees)                           ## scatterplot matrix
plot(Volume ~ Girth, data = trees)    ## scatterplot

tree.lm <- lm(Volume ~ Girth, data = trees)
abline(tree.lm)                       ## regression line
plot(tree.lm)                         ## residual/diagnostic plots
boxplot(trees)                        ## boxplots
qqnorm(trees$Volume)                  ## quantile plot
```

Graphical Parameter

- for nicer graphics, the `par` graphical parameters can be adapted
- some graphical parameters can be adjusted in high-level functions
- for help check `?plot`, `?plot.default`, or more comprehensive `?par`

Graphical Parameter

Some graphical parameters can can be set in high-level functions like `plot()`. For example:

`axes`	should the axes be plotted?
`col`	color
`log`	logarithmic scale
`main, sub`	title and subtitle
`pch`	symbol for points
`type`	type (l=line, p=point, b=both, n=none)
`xlab, ylab`	x-/y-axis label xlim, ylim
`xlim, ylim`	x-/y-axis range

Graphical Parameter

The most commonly used arguments in `par()`:

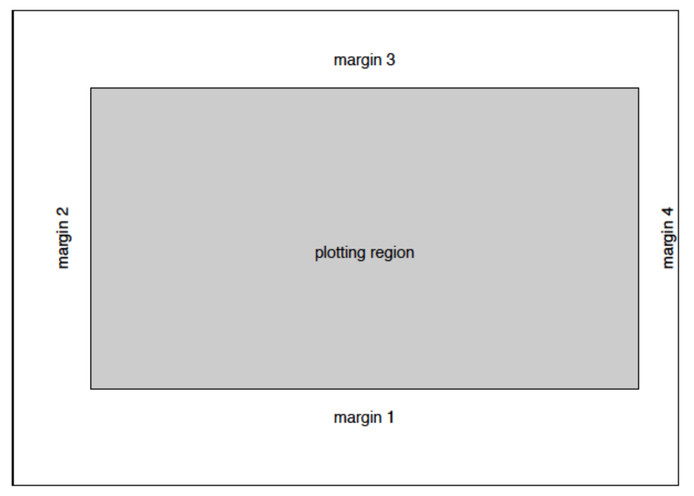
`bg`	background color
`cex`	size of a point or a letter
`las`	should labels be placed parallel wrt the axes
`lty`, `lwd`	line type (dashed, ...) and line width
`mar`	size of the margins
`mfcol`, `mfrow`	multiple plots in one device in rows/columns
`mfg`	which plot in a device should be chosen?
`oma`	size of the outer margins
`usr`	current extrema of the user coordinates
`xaxt`, `yaxt`	x-/y-axis scaling

Graphical Parameter

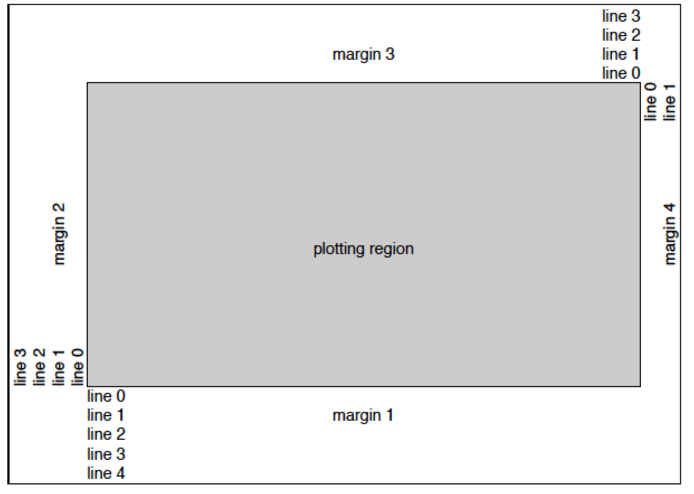
```
opar <- par(mfrow = c(1, 1),bg = "White")
# example 1
par(mfrow = c(2, 2))
boxplot(trees, col = "blue")
hist(trees$Volume, las = 1)
qqnorm(trees$Volume, cex.axis = 2, pch = (trees$Girth > 14) + 8)
plot(trees$Girth,trees$Height,cex = scale(trees$Volume,center=FALSE))
par(opar)

# example 2
set.seed(123)
x <- rnorm(100)
par(bg = "lightgreen")
hist(x, freq = FALSE, col = "red", las = 1,
     xlim = c(-5, 5), ylim = c(0, 0.6),
     main = "100 Draws from N(0,1)-distributed random variables")
curve(dnorm, from = -5, to = 5, add = TRUE, col = "blue", lwd = 3)
par(opar)
```

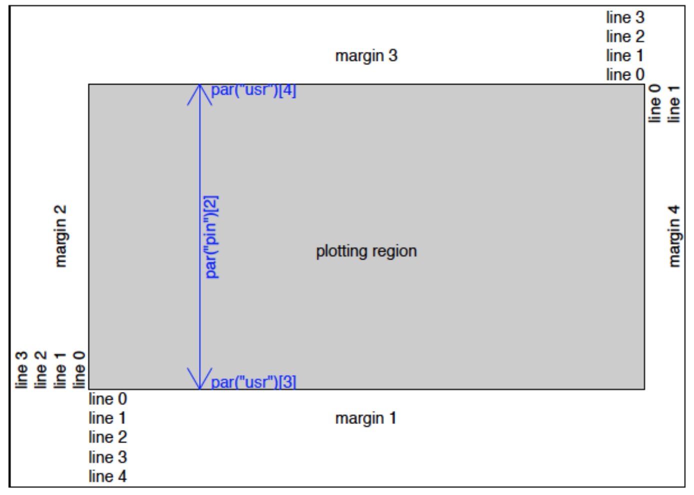
Device Control



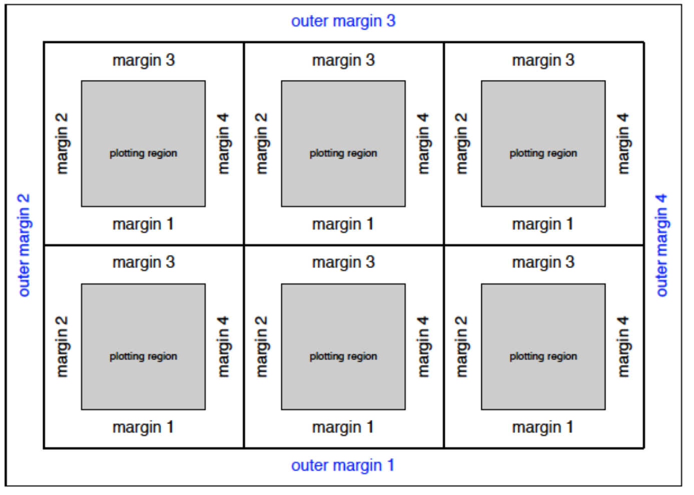
Device Control



Device Control



`par(mfrow=c(2,3))`



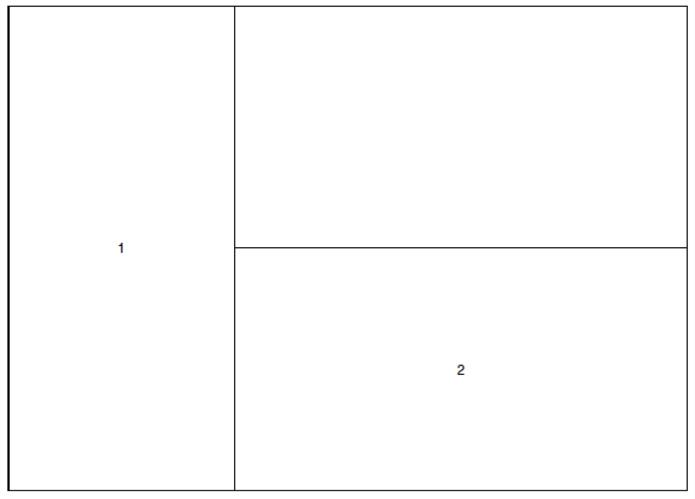
`layout()` Function

- `layout()` organizes independent plots on one plotting device, also in irregular grids
- boxes can have different widths
- neighboring boxed can be combined
- boxes can be left empty

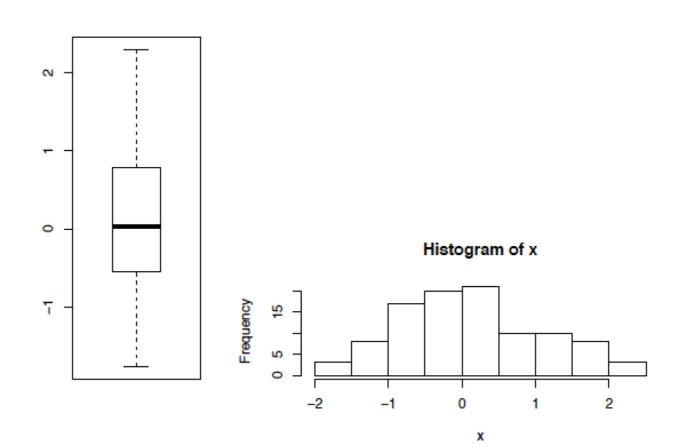
```
m <- matrix(c(1,1,0,2), 2, 2)
m
##      [,1] [,2]
## [1,]    1    0
## [2,]    1    2

layout(m, widths=c(1,2))
x <- rnorm(100)
boxplot(x)
hist(x)
```

`layout()` Function



`layout()` Function



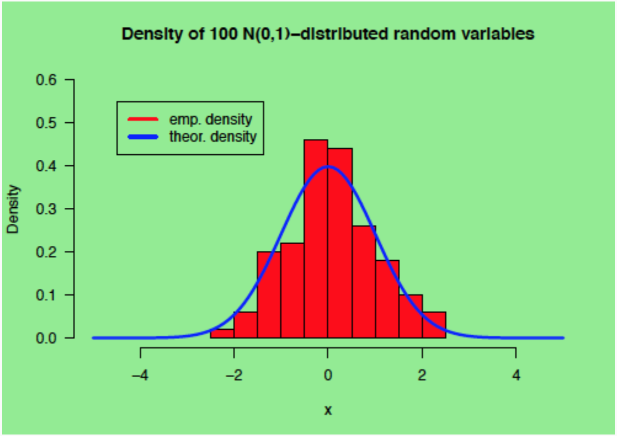
Low-level Graphics

Low-level functions add elements to a (with high-level function) generated graphic, e.g., additional points, legends, etc.

`abline()`	“intelligent” lines
`arrows()`	arrows
`axis()`	axes
`grid()`	gridlines
`legend()`	legend
`lines()`	(stepwise) lines
`mtext()`	text in margins
`points()`	points
`polygon()`	(filled) polygons
`segments()`	vector lines
`text()`	text
`title()`	title label

Low-level Graphics

```
# example 2 (cont.):
legend(-4.5, 0.55, legend = c("emp. density", "theor. density"),
       col = c("red", "blue"), lwd = 3)
```

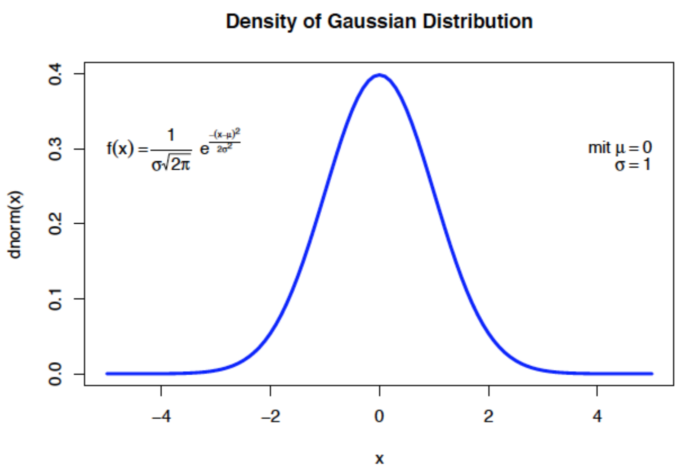


Mathematical Expressions

- mathematical notation and symbols formatted similar to LaTeXcode can be integrated in functions such as `axis()`, `legend()`, `mtext()`, `text()`, and `title()`
- For help check `?plotmath` or run `demo(plotmath)`

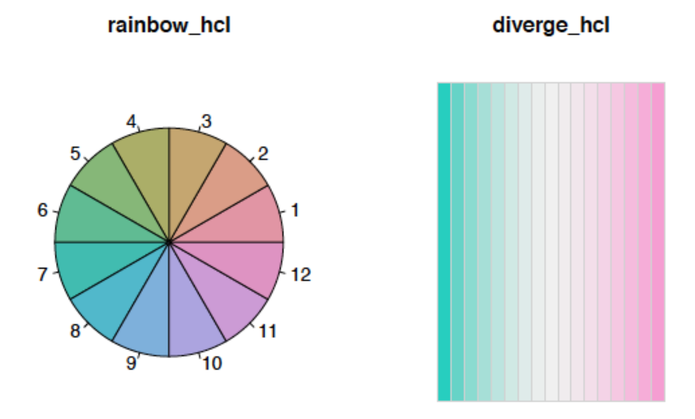
```
curve(dnorm, main = "Density of Gaussian Distribution",
      from = -5, to = 5, col = "blue", lwd = 3)
text(-3, 0.3,
     expression(f(x) == frac(1, sigma * sqrt(2*pi)) ~~
         e^{frac(-(x - mu)^2, 2 * sigma^2)}))
text(4, 0.3, expression(paste("mit ", mu == 0)))
sigma <- 1
text(4.2, 0.28, bquote(sigma == .(sigma)))
```

Mathematical Expressions



The package `colorspace` provides various functions for perceptually-balanced color palettes

```
rainbow_hcl(12)
diverge_hcl(17, h = c(180, 330), c = 59, l = c(75, 95))
```



Trellis Graphics

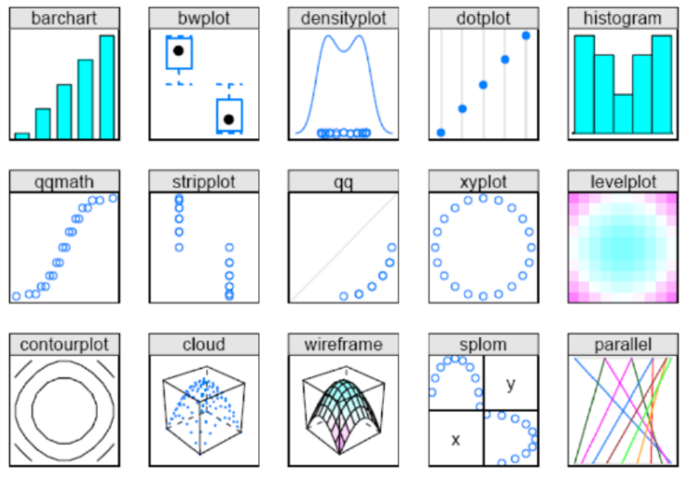
- trellis graphics is a family of techniques for viewing complicated data sets, that are based on basic concepts of human perception
- everything is possible (using a sufficient number of parameters)
- the trellis graphics system is in the `lattice` package
- the typical format is

```
graph_type(formula, data = )
```

Trellis Graphics

`barchart()`	barplot
`bwplot()`	boxplot
`cloud()`	3D point clouds
`contourplot`	3D contour plot
`densityplot()`	kernel density plot
`dotplot()`	point plots
`histogram()`	histogram
`levelplot()`	levelplots
`panel.....()`	functions to add elements
`piechart()`	pie diagram
`print.trellis()`	plotting trellis object
`qq()`	QQ–plots
`stripplot`	strip plots
`wireframe()`	persp. 3D areas
`xyplot()`	scatterplot

Trellis Graphics



Trellis Graphics

```
require(lattice)
attach(mtcars)
# create factors with value labels
gear.f <- factor(gear, levels = c(3,4,5),
   labels = c("3gears", "4gears", "5gears"))
cyl.f <- factor(cyl, levels = c(4,6,8),
   labels = c("4cyl","6cyl","8cyl"))

# kernel density plot
densityplot(~mpg,
   main = "Density Plot",
   xlab = "Miles per Gallon")
# kernel density plots by factor level
densityplot(~mpg|cyl.f,
   main = "Density Plot by Number of Cylinders",
   xlab = "Miles per Gallon")
```

Interactive Graphics

The basic R system does not allow many possibilities for interactive graphics. Some exceptions are:

- `identify()` identifies a selected data point, e.g.:

  ```
  x <- rnorm(10)
  plot(x)
  identify(x)
  ```

- `locator()` returns the coordinates of a selected point, which can be used for instance for the interactive placing of labels:

  ```
  plot(x)
  legend(locator(1), legend = "A legend", pch = 1)
  ```

Further References

- First analysis: Brian Everitt and Torsten Hothorn. A handbook of statistical analyses using R. Chapman & Hall/CRC, 2006
- Trellis: Deepayan Sarkar. Lattice: multivariate data visualization with R. Use R! Springer, 2008
- Colorspace: Achim Zeileis, Kurt Hornik, and Paul Murrell. Escaping RGBland: Selecting colors for statistical graphics. Computational Statistics & Data Analysis, 53:3259?3270, 2009. doi: 10.1016/j.csda.2008.11.033