Tutorial R

Iniziamo il tutorial. Primi comandi.

In [ ]:
```
x = 2
x
```

2

L' "=" è equivalente a "<-"

In [ ]:
```
x = 2
x <- 2
```

In [ ]:
```
y = 3
z = x * y
```

Per vedere gli oggetti creati, potete usare

In [ ]:
```
ls()
```

'area_rettangolo' · 'sd' · 'x' · 'xseq' · 'y' · 'z' · 'z_dscreta'

Invece, per rimuovere un oggetto, potete usare "rm"

In [ ]:
```
rm(x)
```

In [ ]:
```
ls()
```

'area_rettangolo' · 'sd' · 'xseq' · 'y' · 'z' · 'z_dscreta'

Oppure, se volete pulire tutto l'enviroment, potete usare

In [ ]:
```
rm(list=ls())
ls()
```

Per vedere l'help di una funzione, si utilizza il punto interrogative

In [ ]:
```
?rm
```

remove package:base R Documentation

_R_e_m_o_v_e _O_b_j_e_c_t_s _f_r_o_m _a _S_p_e_c_i_f_i_e_d _E_n_v_i_r_o_n_m _e_n_t

_D_e_s_c_r_i_p_t_i_o_n:

'remove' and 'rm' can be used to remove objects. These can be specified successively as character strings, or in the character vector 'list', or through a combination of both. All objects thus specified will be removed.

If 'envir' is NULL then the currently active environment is searched first.

If 'inherits' is 'TRUE' then parents of the supplied directory are searched until a variable with the given name is encountered. A warning is printed for each variable that is not found.

_U_s_a_g_e:

```
remove(..., list = character(), pos = -1,
       envir = as.environment(pos), inherits = FALSE)

rm    (..., list = character(), pos = -1,
       envir = as.environment(pos), inherits = FALSE)
```

_A_r_g_u_m_e_n_t_s:

...:	the objects to be removed, as names (unquoted) or character strings (quoted).
list:	a character vector (or 'NULL') naming objects to be removed.
pos:	where to do the removal. By default, uses the current environment. See 'details' for other possibilities.
envir:	the 'environment' to use. See 'details'.
inherits:	should the enclosing frames of the environment be inspected?

_D_e_t_a_i_l_s:

The 'pos' argument can specify the environment from which to remove the objects in any of several ways: as an integer (the position in the 'search' list); as the character string name of an element in the search list; or as an 'environment' (including using 'sys.frame' to access the currently active function calls). The 'envir' argument is an alternative way to specify an environment, but is primarily there for back compatibility.

It is not allowed to remove variables from the base environment and base namespace, nor from any environment which is locked (see 'lockEnvironment').

Earlier versions of R incorrectly claimed that supplying a character vector in '...' removed the objects named in the character vector, but it removed the character vector. Use the 'list' argument to specify objects _via_ a character vector.

_R_e_f_e_r_e_n_c_e_s:

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) _The New S Language_. Wadsworth & Brooks/Cole.

```
_S_e_e _A_l_s_o:

     'ls', 'objects'

_E_x_a_m_p_l_e_s:

     tmp <- 1:4
     ## work with tmp  and cleanup
     rm(tmp)

     ## Not run:

     ## remove (almost) everything in the working environment.
     ## You will get no warning, so don't do this unless you are really sur
e.
     rm(list = ls())
     ## End(Not run)
```

diamo un'occhiata agli operatori logici.

In [ ]:
```
x = 2
y = 3
x == y
x != y
x > y
x >= y
z_dscreta = x == y
```

FALSE

TRUE

FALSE

FALSE

Adesso voglio trasformare il TRUE/FALSE in numerico

In [ ]:
```
z_dscreta
```

FALSE

se voglio vedere dei dettagli di una variabile, posso sare "str"

In [ ]:
```
str(z_dscreta)

z_dscreta + 3
T +3
```

```
 logi FALSE
```

3

4

Quindi, R tratta T e F come fossero 1 e 0.

Vediamo come si creano sequenze di numeri equispaziati

In [ ]:
```
1:10
xseq = seq( from = 0.2, to = 10.1,length.out = 3)
xseq
```

$1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 6 \cdot 7 \cdot 8 \cdot 9 \cdot 10$

$0.2 \cdot 5.15 \cdot 10.1$

In [ ]:
```
?seq
```

```
seq                     package:base                     R Documentation

_S_e_q_u_e_n_c_e _G_e_n_e_r_a_t_i_o_n

_D_e_s_c_r_i_p_t_i_o_n:

     Generate regular sequences.  'seq' is a standard generic with a
     default method.  'seq.int' is a primitive which can be much faster
     but has a few restrictions.  'seq_along' and 'seq_len' are very
     fast primitives for two common cases.

_U_s_a_g_e:

     seq(...)

     ## Default S3 method:
     seq(from = 1, to = 1, by = ((to - from)/(length.out - 1)),
         length.out = NULL, along.with = NULL, ...)

     seq.int(from, to, by, length.out, along.with, ...)

     seq_along(along.with)
     seq_len(length.out)

_A_r_g_u_m_e_n_t_s:

     ...: arguments passed to or from methods.

from, to: the starting and (maximal) end values of the sequence.  Of
          length '1' unless just 'from' is supplied as an unnamed
          argument.

      by: number: increment of the sequence.

length.out: desired length of the sequence.  A non-negative number,
          which for 'seq' and 'seq.int' will be rounded up if
          fractional.

along.with: take the length from the length of this argument.

_D_e_t_a_i_l_s:

     Numerical inputs should all be finite (that is, not infinite,
     'NaN' or 'NA').

     The interpretation of the unnamed arguments of 'seq' and 'seq.int'
     is _not_ standard, and it is recommended always to name the
     arguments when programming.

     'seq' is generic, and only the default method is described here.
     Note that it dispatches on the class of the *first* argument
     irrespective of argument names.  This can have unintended
     consequences if it is called with just one argument intending this
     to be taken as 'along.with': it is much better to use 'seq_along'
     in that case.

     'seq.int' is an internal generic which dispatches on methods for
     '"seq"' based on the class of the first supplied argument (before
     argument matching).

     Typical usages are

     seq(from, to)
     seq(from, to, by= )
```

```
     seq(from, to, length.out= )
     seq(along.with= )
     seq(from)
     seq(length.out= )

     The first form generates the sequence 'from, from+/-1, ..., to'
     (identical to 'from:to').

     The second form generates 'from, from+by', ..., up to the sequence
     value less than or equal to 'to'.  Specifying 'to - from' and 'by'
     of opposite signs is an error.  Note that the computed final value
     can go just beyond 'to' to allow for rounding error, but is
     truncated to 'to'.  ('Just beyond' is by up to 1e-10 times
     'abs(from - to)'.)

     The third generates a sequence of 'length.out' equally spaced
     values from 'from' to 'to'.  ('length.out' is usually abbreviated
     to 'length' or 'len', and 'seq_len' is much faster.)

     The fourth form generates the integer sequence '1, 2, ...,
     length(along.with)'.  ('along.with' is usually abbreviated to
     'along', and 'seq_along' is much faster.)

     The fifth form generates the sequence '1, 2, ..., length(from)'
     (as if argument 'along.with' had been specified), _unless_ the
     argument is numeric of length 1 when it is interpreted as '1:from'
     (even for 'seq(0)' for compatibility with S).  Using either
     'seq_along' or 'seq_len' is much preferred (unless strict S
     compatibility is essential).

     The final form generates the integer sequence '1, 2, ...,
     length.out' unless 'length.out = 0', when it generates
     'integer(0)'.

     Very small sequences (with 'from - to' of the order of 10^{-14}
     times the larger of the ends) will return 'from'.

     For 'seq' (only), up to two of 'from', 'to' and 'by' can be
     supplied as complex values provided 'length.out' or 'along.with'
     is specified.  More generally, the default method of 'seq' will
     handle classed objects with methods for the 'Math', 'Ops' and
     'Summary' group generics.

     'seq.int', 'seq_along' and 'seq_len' are primitive.

_V_a_l_u_e:

     'seq.int' and the default method of 'seq' for numeric arguments
     return a vector of type '"integer"' or '"double"': programmers
     should not rely on which.

     'seq_along' and 'seq_len' return an integer vector, unless it is a
     _long vector_ when it will be double.

_R_e_f_e_r_e_n_c_e_s:

     Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) _The New S
     Language_.  Wadsworth & Brooks/Cole.

_S_e_e _A_l_s_o:

     The methods 'seq.Date' and 'seq.POSIXt'.

     ':', 'rep', 'sequence', 'row', 'col'.
```

```
_E_x_a_m_p_l_e_s:

     seq(0, 1, length.out = 11)
     seq(stats::rnorm(20)) # effectively 'along'
     seq(1, 9, by = 2)     # matches 'end'
     seq(1, 9, by = pi)    # stays below 'end'
     seq(1, 6, by = 3)
     seq(1.575, 5.125, by = 0.05)
     seq(17) # same as 1:17, or even better seq_len(17)
```

In [ ]:
```
data
```

```
function (..., list = character(), package = NULL, lib.loc = NULL,
    verbose = getOption("verbose"), envir = .GlobalEnv, overwrite =
TRUE)
{
    fileExt <- function(x) {
        db <- grepl("\\.[^.]+\\.(gz|bz2|xz)$", x)
        ans <- sub(".*\\.", "", x)
        ans[db] <- sub(".*\\.([^.]+\\.)(gz|bz2|xz)$", "\\1\\2",
            x[db])
        ans
    }
    my_read_table <- function(...) {
        lcc <- Sys.getlocale("LC_COLLATE")
        on.exit(Sys.setlocale("LC_COLLATE", lcc))
        Sys.setlocale("LC_COLLATE", "C")
        read.table(...)
    }
    stopifnot(is.character(list))
    names <- c(as.character(substitute(list(...))[-1L]), list)
    if (!is.null(package)) {
        if (!is.character(package))
            stop("'package' must be a character vector or NULL")
    }
    paths <- find.package(package, lib.loc, verbose = verbose)
    if (is.null(lib.loc))
        paths <- c(path.package(package, TRUE), if (!length(packag
e)) getwd(),
            paths)
    paths <- unique(normalizePath(paths[file.exists(paths)]))
    paths <- paths[dir.exists(file.path(paths, "data"))]
    dataExts <- tools:::.make_file_exts("data")
    if (length(names) == 0L) {
        db <- matrix(character(), nrow = 0L, ncol = 4L)
        for (path in paths) {
            entries <- NULL
            packageName <- if (file_test("-f", file.path(path,
                "DESCRIPTION")))
                basename(path)
            else "."
            if (file_test("-f", INDEX <- file.path(path, "Meta",
                "data.rds"))) {
                entries <- readRDS(INDEX)
            }
            else {
                dataDir <- file.path(path, "data")
                entries <- tools::list_files_with_type(dataDir,
                  "data")
                if (length(entries)) {
                  entries <- unique(tools::file_path_sans_ext(basena
me(entries)))
                  entries <- cbind(entries, "")
                }
            }
            if (NROW(entries)) {
                if (is.matrix(entries) && ncol(entries) == 2L)
                  db <- rbind(db, cbind(packageName, dirname(path),
                    entries))
```

```
                else warning(gettextf("data index for package %s is
invalid and will be ignored",
                  sQuote(packageName)), domain = NA, call. = FALSE)
            }
        }
        colnames(db) <- c("Package", "LibPath", "Item", "Title")
        footer <- if (missing(package))
            paste0("Use ", sQuote(paste("data(package =", ".packages
(all.available = TRUE))")),
                "\n", "to list the data sets in all *available* pack
ages.")
        else NULL
        y <- list(title = "Data sets", header = NULL, results = db,
            footer = footer)
        class(y) <- "packageIQR"
        return(y)
    }
    paths <- file.path(paths, "data")
    for (name in names) {
        found <- FALSE
        for (p in paths) {
            tmp_env <- if (overwrite)
                envir
            else new.env()
            if (file_test("-f", file.path(p, "Rdata.rds"))) {
                rds <- readRDS(file.path(p, "Rdata.rds"))
                if (name %in% names(rds)) {
                  found <- TRUE
                  if (verbose)
                    message(sprintf("name=%s:\t found in Rdata.rds",
                      name), domain = NA)
                  thispkg <- sub(".*/([^/]*)/data$", "\\1", p)
                  thispkg <- sub("_.*$", "", thispkg)
                  thispkg <- paste0("package:", thispkg)
                  objs <- rds[[name]]
                  lazyLoad(file.path(p, "Rdata"), envir = tmp_env,
                    filter = function(x) x %in% objs)
                  break
                }
                else if (verbose)
                  message(sprintf("name=%s:\t NOT found in names() o
f Rdata.rds, i.e.,\n\t%s\n",
                    name, paste(names(rds), collapse = ",")),
                    domain = NA)
            }
            if (file_test("-f", file.path(p, "Rdata.zip"))) {
                warning("zipped data found for package ", sQuote(bas
ename(dirname(p))),
                  ".\nThat is defunct, so please re-install the pack
age.",
                  domain = NA)
                if (file_test("-f", fp <- file.path(p, "filelist")))
                  files <- file.path(p, scan(fp, what = "", quiet =
TRUE))
                else {
                  warning(gettextf("file 'filelist' is missing for d
irectory %s",
```

```
                        sQuote(p)), domain = NA)
                      next
                    }
                }
                else {
                    files <- list.files(p, full.names = TRUE)
                }
                files <- files[grep(name, files, fixed = TRUE)]
                if (length(files) > 1L) {
                    o <- match(fileExt(files), dataExts, nomatch = 100L)
                    paths0 <- dirname(files)
                    paths0 <- factor(paths0, levels = unique(paths0))
                    files <- files[order(paths0, o)]
                }
                if (length(files)) {
                    for (file in files) {
                      if (verbose)
                        message("name=", name, ":\t file= ...", .Platfor
m$file.sep,
                          basename(file), "::\t", appendLF = FALSE,
                          domain = NA)
                      ext <- fileExt(file)
                      if (basename(file) != paste0(name, ".", ext))
                        found <- FALSE
                      else {
                        found <- TRUE
                        zfile <- file
                        zipname <- file.path(dirname(file), "Rdata.zip")
                        if (file.exists(zipname)) {
                          Rdatadir <- tempfile("Rdata")
                          dir.create(Rdatadir, showWarnings = FALSE)
                          topic <- basename(file)
                          rc <- .External(C_unzip, zipname, topic,
                            Rdatadir, FALSE, TRUE, FALSE, FALSE)
                          if (rc == 0L)
                            zfile <- file.path(Rdatadir, topic)
                        }
                        if (zfile != file)
                          on.exit(unlink(zfile))
                        switch(ext, R = , r = {
                          library("utils")
                          sys.source(zfile, chdir = TRUE, envir = tmp_en
v)
                        }, RData = , rdata = , rda = load(zfile,
                          envir = tmp_env), TXT = , txt = , tab = ,
                          tab.gz = , tab.bz2 = , tab.xz = , txt.gz = ,
                          txt.bz2 = , txt.xz = assign(name, my_read_tabl
e(zfile,
                            header = TRUE, as.is = FALSE), envir = tmp_e
nv),
                          CSV = , csv = , csv.gz = , csv.bz2 = ,
                          csv.xz = assign(name, my_read_table(zfile,
                            header = TRUE, sep = ";", as.is = FALSE),
                            envir = tmp_env), found <- FALSE)
                      }
                      if (found)
                        break
```

```
                }
                if (verbose)
                  message(if (!found)
                    "*NOT* ", "found", domain = NA)
            }
            if (found)
                break
        }
        if (!found) {
            warning(gettextf("data set %s not found", sQuote(name)),
                domain = NA)
        }
        else if (!overwrite) {
            for (o in ls(envir = tmp_env, all.names = TRUE)) {
                if (exists(o, envir = envir, inherits = FALSE))
                  warning(gettextf("an object named %s already exist
s and will not be overwritten",
                    sQuote(o)))
                else assign(o, get(o, envir = tmp_env, inherits = FA
LSE),
                  envir = envir)
            }
            rm(tmp_env)
        }
    }
    invisible(names)
}
```

Scriviamo una funzione

la struttura di una funzione è

nome_funzione = function(arg1, arg2, arg3, ...)\ {\ calcoli vari\ return(valore da ritornare)\ }

Calcolimao l'aread di un rettangolo

In [ ]:
```
area_rettangolo = function(lato1=20, lato2 = 10)
{
    area = lato1*lato2
    return(area)
}

area_rettangolo()
```

200

Fate attenzione che potreste sovrascrivere funzioni già esistenti

In [ ]:
```
sd = function(x)
{
    return(x)
}
```

Per installare un pacchetto dovete usare il comando "install.packages("ggplot2")"
qeusto va fatto una sola volta

In [ ]:
```
install.packages("ggplot2")
```

```
I pacchetti binari scaricati sono in
        /var/folders/z2/jssz1nk14s7f14zkf0fnb4c40000gn/T//RtmpCc37Q0/downlo
aded_packages
```

se poi lo voglio utilizzare, devo lanciare il comando "library(ggplot2)"

In [ ]:
```
library(ggplot2)
ggplot
```

```
function (data = NULL, mapping = aes(), ..., environment = parent.fr
ame())
{
    UseMethod("ggplot")
}
```

Cicli

il ciclo for ha struttura

for(variabile in valore_su_cui ciclare)\ {\

    #calcolo vari\

}

In [ ]:
```
for(y in seq(0,10, by = 0.5))
{
    print(y)
}
```

```
[1] 0
[1] 0.5
[1] 1
[1] 1.5
[1] 2
[1] 2.5
[1] 3
[1] 3.5
[1] 4
[1] 4.5
[1] 5
[1] 5.5
[1] 6
[1] 6.5
[1] 7
[1] 7.5
[1] 8
[1] 8.5
[1] 9
[1] 9.5
[1] 10
```

Il ciclo while

while(condizione) {

```
    # calcoli vari
```

}

In [ ]:
```
i = 0
while(i<9)
{
    i = i + 1
    print(i)
}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
```

L'if/else si fa come

if(condizione) {

```
    # calcoli da fare se è vera la condizione
```

}

oppure

if(condizione) {

```
    # calcoli da fare se è vera la condizione
```

}else{

```
    # calcoli da fare se non è vera la condizione
```

}

In [ ]:
```
x = 3
if(x == 2)
{
    print("x e' uguale a 2")
}else{
    print("x non e' uguale a 2")
}
```

```
[1] "x non e' uguale a 2"
```

Lavorare con distribuzioni

ci sono dei comandi standard su R per ottenere il calcolo della densità/probabilità (d), comulata (p), quantile (q) e ottenere dei valori da una distribuzione prefissata (r) per esempio, per una normale possiamo usare

dnorm(), pnorm(), qnorm(), rnorm()

?dnorm

per l'esponenziale dexp(), pexp(), qexp(), rexp()

?dexp

per la poisson dpois(), ppois(), qpois(), rpois()

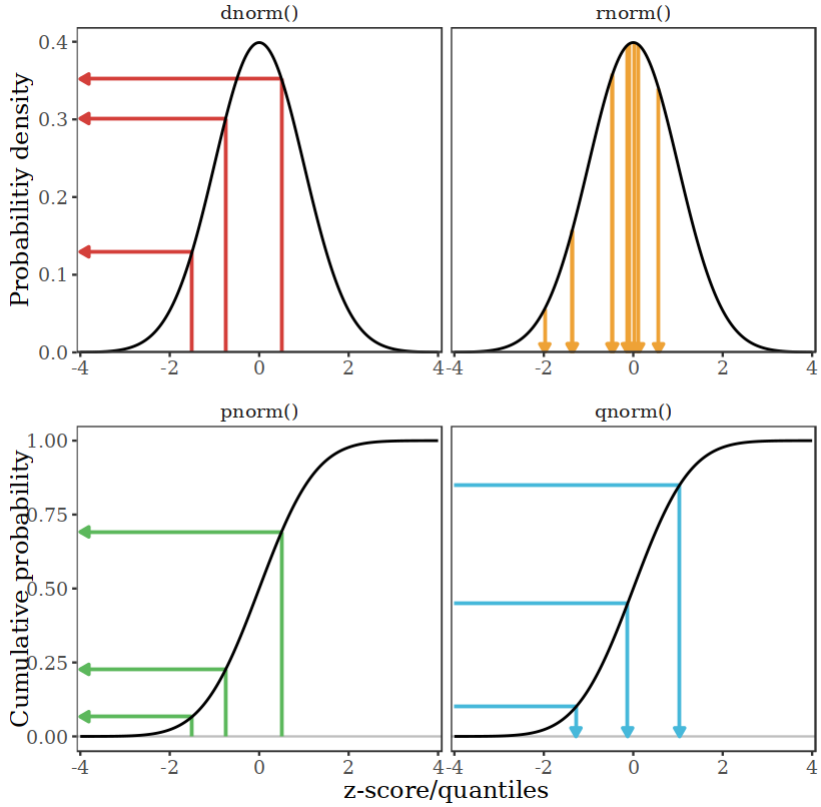
?dpois

I comandi qui sotto e fino alla fine di questo chunk del codice servono solo a far vedere cosa calcolano queste funzioni nel caso di una normale standard. Lanciateli e vedete la figura che producono

In [ ]:
```
library(datasets)
library(catdata)
library(dslabs)
library(mvtnorm)
library(patchwork)
library(paletteer)
library(tidyverse)
z_scores <- seq(-4, 4, by = 0.01)
mu <- 0
sd <- 1


#Functions
##Using `dnorm` and `pnorm` to setup the "skeleton" of related plots.
normal_dists <- list(`dnorm()` = ~ dnorm(., mu, sd),
  `rnorm()` = ~ dnorm(., mu, sd),
  `pnorm()` = ~ pnorm(., mu, sd),
  `qnorm()` = ~ pnorm(., mu, sd))

##Apply functions to data and parameter combinations
df <- tibble(z_scores, mu, sd) %>%
  mutate_at(.vars = vars(z_scores), .funs = normal_dists) %>%
  #"Lengthen" the data
  pivot_longer(cols = -c(z_scores, mu, sd), names_to = "func",
    values_to = "prob") %>%
  #Categorize based on shape of distribution -- need to split up the datafra
  # for plotting later.
  mutate(distribution = ifelse(func == "pnorm()" | func == "qnorm()",
    "Cumulative probability", "Probability density"))

##Split up the data into different pieces that can then be added to a plot.
###Probabilitiy density distrubitions
df_pdf <- df %>%
  filter(distribution == "Probability density") %>%
  rename(`Probabilitiy density` = prob)

###Cumulative density distributions
df_cdf <- df %>%
  filter(distribution == "Cumulative probability") %>%
  rename(`Cumulative probability` = prob)
```

```
###dnorm segments
  #Need to make lines that represent examples of how values are mapped -- tl
  # is probably a better way to do this, but quick and dirty is fine for now
df_dnorm <- tibble(z_start.line_1 = c(-1.5, -0.75, 0.5),
  pd_start.line_1 = 0) %>%
  mutate(z_end.line_1 = z_start.line_1,
    pd_end.line_1 = dnorm(z_end.line_1, mu, sd),
    z_start.line_2 = z_end.line_1,
    pd_start.line_2 = pd_end.line_1,
    z_end.line_2 = min(z_scores),
    pd_end.line_2 = pd_start.line_2,
    id = 1:n()) %>%
  pivot_longer(-id) %>%
  separate(name, into = c("source", "line"), sep = "\\.") %>%
  pivot_wider(id_cols = c(id, line), names_from = source) %>%
  mutate(func = "dnorm()",
    size = ifelse(line == "line_1", 0, 0.03))

###rnorm segments
  #Make it reproducible
set.seed(20200209)
df_rnorm <- tibble(z_start = rnorm(10, mu, sd)) %>%
  mutate(pd_start = dnorm(z_start, mu, sd),
    z_end = z_start,
    pd_end = 0,
    func = "rnorm()")

###pnorm segments
df_pnorm <- tibble(z_start.line_1 = c(-1.5, -0.75, 0.5),
  pd_start.line_1 = 0) %>%
  mutate(z_end.line_1 = z_start.line_1,
    pd_end.line_1 = pnorm(z_end.line_1, mu, sd),
    z_start.line_2 = z_end.line_1,
    pd_start.line_2 = pd_end.line_1,
    z_end.line_2 = min(z_scores),
    pd_end.line_2 = pd_start.line_2,
    id = 1:n()) %>%
  pivot_longer(-id) %>%
  separate(name, into = c("source", "line"), sep = "\\.") %>%
  pivot_wider(id_cols = c(id, line), names_from = source) %>%
  mutate(func = "pnorm()",
    size = ifelse(line == "line_1", 0, 0.03))

###qnorm segments
df_qnorm <- tibble(z_start.line_1 = min(z_scores),
  pd_start.line_1 = c(0.1, 0.45, 0.85)) %>%
  mutate(z_end.line_1 = qnorm(pd_start.line_1),
    pd_end.line_1 = pd_start.line_1,
    z_start.line_2 = z_end.line_1,
    pd_start.line_2 = pd_end.line_1,
    z_end.line_2 = z_end.line_1,
    pd_end.line_2 = 0,
    id = 1:n()) %>%
  pivot_longer(-id) %>%
  separate(name, into = c("source", "line"), sep = "\\.") %>%
  pivot_wider(id_cols = c(id, line), names_from = source) %>%
  mutate(func = "qnorm()",
    size = ifelse(line == "line_1", 0, 0.03))

    cp <- paletteer_d("ggsci::default_locuszoom", 4, )
    names(cp) <- c("dnorm()", "rnorm()", "pnorm()", "qnorm()")

    ##Probabilitiy density
```

```
    p_pdf <- df_pdf %>%
      ggplot(aes(z_scores, `Probabilitiy density`)) +
      geom_segment(data = df_dnorm,
        aes(z_start, pd_start, xend = z_end, yend = pd_end),
        arrow = arrow(length = unit(df_dnorm$size, "npc"), type = "closed"),
        size = 0.8, color = cp["dnorm()"]) +
      geom_segment(data = df_rnorm,
        aes(z_start, pd_start, xend = z_end, yend = pd_end),
        arrow = arrow(length = unit(0.03, "npc"), type = "closed"),
        size = 0.8, color = cp["rnorm()"]) +
      geom_line(size = 0.6) +
      facet_wrap(~ func, nrow = 1) +
      theme_bw() +
      theme(panel.grid = element_blank(),
        axis.title.x = element_blank(),
        strip.background = element_blank(),
        text = element_text(family = "serif", size = 14)) +
      scale_y_continuous(expand = expand_scale(c(0, 0.05))) +
      scale_x_continuous(expand = c(0.01, 0))

    ##Cumulative probability
    p_cdf <- df_cdf %>%
      ggplot(aes(z_scores, `Cumulative probability`)) +
      geom_hline(yintercept = 0, color = "grey") +
      geom_segment(data = df_pnorm,
        aes(z_start, pd_start, xend = z_end, yend = pd_end),
        arrow = arrow(length = unit(df_dnorm$size, "npc"), type = "closed"),
        size = 0.8, color = cp["pnorm()"]) +
      geom_segment(data = df_qnorm,
        aes(z_start, pd_start, xend = z_end, yend = pd_end),
        arrow = arrow(length = unit(df_qnorm$size, "npc"), type = "closed"),
        size = 0.8, color = cp["qnorm()"]) +
      geom_line(size = 0.6) +
      facet_wrap(~ func, nrow = 1) +
      labs(x = "z-score/quantiles") +
      theme_bw() +
      theme(panel.grid = element_blank(),
        strip.background = element_blank(),
        text = element_text(family = "serif", size = 14)) +
      scale_x_continuous(expand = c(0.01, 0))

    ##Combine the plots
    p_pdf + p_cdf + plot_layout(ncol = 1)
#### #### #### ####
```


dnorm()
rnorm()
pnorm()
qnorm()
Probabilitiy density
Cumulative probability
z-score/quantiles

In [ ]:
```
?dnorm
```

```
Normal                  package:stats                  R Documentation

_T_h_e _N_o_r_m_a_l _D_i_s_t_r_i_b_u_t_i_o_n

_D_e_s_c_r_i_p_t_i_o_n:

     Density, distribution function, quantile function and random
     generation for the normal distribution with mean equal to 'mean'
     and standard deviation equal to 'sd'.

_U_s_a_g_e:

     dnorm(x, mean = 0, sd = 1, log = FALSE)
     pnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
     qnorm(p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
     rnorm(n, mean = 0, sd = 1)

_A_r_g_u_m_e_n_t_s:

    x, q: vector of quantiles.

       p: vector of probabilities.

       n: number of observations. If 'length(n) > 1', the length is
          taken to be the number required.

    mean: vector of means.

      sd: vector of standard deviations.

log, log.p: logical; if TRUE, probabilities p are given as log(p).

lower.tail: logical; if TRUE (default), probabilities are P[X <= x]
          otherwise, P[X > x].

_D_e_t_a_i_l_s:

     If 'mean' or 'sd' are not specified they assume the default values
     of '0' and '1', respectively.

     The normal distribution has density

            f(x) = 1/(sqrt(2 pi) sigma) e^-((x - mu)^2/(2 sigma^2))

     where mu is the mean of the distribution and sigma the standard
     deviation.

_V_a_l_u_e:

     'dnorm' gives the density, 'pnorm' gives the distribution
     function, 'qnorm' gives the quantile function, and 'rnorm'
     generates random deviates.

     The length of the result is determined by 'n' for 'rnorm', and is
     the maximum of the lengths of the numerical arguments for the
     other functions.

     The numerical arguments other than 'n' are recycled to the length
     of the result.  Only the first elements of the logical arguments
     are used.

     For 'sd = 0' this gives the limit as 'sd' decreases to 0, a point
     mass at 'mu'.  'sd < 0' is an error and returns 'NaN'.
```

_S_o_u_r_c_e:

For 'pnorm', based on

Cody, W. D. (1993) Algorithm 715: SPECFUN - A portable FORTRAN package of special function routines and test drivers. _ACM Transactions on Mathematical Software_ *19*, 22-32.

For 'qnorm', the code is based on a C translation of

Wichura, M. J. (1988) Algorithm AS 241: The percentage points of the normal distribution. _Applied Statistics_, *37*, 477-484; doi:10.2307/2347330 <https://doi.org/10.2307/2347330>.

which provides precise results up to about 16 digits for 'log.p=FALSE'. For log scale probabilities in the extreme tails, since R version 4.1.0, extensively since 4.3.0, asymptotic expansions are used which have been derived and explored in

Maechler, M. (2022) Asymptotic tail formulas for gaussian quantiles; 'DPQ' vignette <https://CRAN.R-project.org/package=DPQ/vignettes/qnorm-asymp.pdf>.

For 'rnorm', see RNG for how to select the algorithm and for references to the supplied methods.

_R_e_f_e_r_e_n_c_e_s:

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) _The New S Language_. Wadsworth & Brooks/Cole.

Johnson, N. L., Kotz, S. and Balakrishnan, N. (1995) _Continuous Univariate Distributions_, volume 1, chapter 13. Wiley, New York.

_S_e_e _A_l_s_o:

Distributions for other standard distributions, including 'dlnorm' for the _Log_normal distribution.

_E_x_a_m_p_l_e_s:

```
require(graphics)

dnorm(0) == 1/sqrt(2*pi)
dnorm(1) == exp(-1/2)/sqrt(2*pi)
dnorm(1) == 1/sqrt(2*pi*exp(1))

## Using "log = TRUE" for an extended range :
par(mfrow = c(2,1))
plot(function(x) dnorm(x, log = TRUE), -60, 50,
     main = "log { Normal density }")
curve(log(dnorm(x)), add = TRUE, col = "red", lwd = 2)
mtext("dnorm(x, log=TRUE)", adj = 0)
mtext("log(dnorm(x))", col = "red", adj = 1)

plot(function(x) pnorm(x, log.p = TRUE), -50, 10,
     main = "log { Normal Cumulative }")
curve(log(pnorm(x)), add = TRUE, col = "red", lwd = 2)
mtext("pnorm(x, log=TRUE)", adj = 0)
mtext("log(pnorm(x))", col = "red", adj = 1)

## if you want the so-called 'error function'
erf <- function(x) 2 * pnorm(x * sqrt(2)) - 1
## (see Abramowitz and Stegun 29.2.29)
```

```
## and the so-called 'complementary error function'
erfc <- function(x) 2 * pnorm(x * sqrt(2), lower = FALSE)
## and the inverses
erfinv <- function (x) qnorm((1 + x)/2)/sqrt(2)
erfcinv <- function (x) qnorm(x/2, lower = FALSE)/sqrt(2)
```

Vettori e matrici

in R i vettori si definiscono come

nome_var = c(elem1, elem2, elem3)

In [ ]:
```
x = c(1,2,3,1,2,4.3)
str(x)
x[2]
x[2] = 10
```

```
 num [1:6] 1 2 3 1 2 4.3
```

2

In [ ]:
```
x[7]
x[7] = 10
str(x)
```

<NA>

```
 num [1:7] 1 10 3 1 2 4.3 10
```

In [ ]:
```
x[20] = 1
x

y = rep(NA,11)

for(i in 1:10)
{
    y[i] = rnorm(1,0,1)
}
y
```

1 · 10 · 3 · 1 · 2 · 4.3 · 10 · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · <NA> · 1

1.32602489544181 · -2.15439887932742 · -0.0231619166394951 · -1.34835128863856 · 0.135730968237259 · -0.728652240795565 · 2.45348470804333 · 0.93755590716786 · -1.55719210225402 · 2.06311674133698 · <NA>

dentro i vettori ci può essere qualsiasi cosa, ma le variabili devono essere dello stesso tipo

In [ ]:
```
z = c("Gian", "Elena","Maria")
str(z)

z1 = c("Gian", "Elena","Maria", 1.2)
str(z1)

z2 = c(T,T,F, FALSE)
str(z2)
```

```
 chr [1:3] "Gian" "Elena" "Maria"
 chr [1:4] "Gian" "Elena" "Maria" "1.2"
 logi [1:4] TRUE TRUE FALSE FALSE
 chr [1:4] "Gian" "Elena" "Maria" "1.2"
 logi [1:4] TRUE TRUE FALSE FALSE
```

per selezionare elementi di vettori possiamo utilizzare un vettore

In [ ]:
```
z1[1]
z1[2]



z1[c(T,T,F,T)]

x = c(1,2,3,4)
z1[x<2.2]
```

'Gian'

'Elena'

'Gian' · 'Elena' · '1.2'

'Gian' · 'Elena'

Operazione tra vettori sono elemento per elemento

In [ ]:
```
x = rnorm(10, 0,1)
y = rnorm(10, 10,1)

h = x + y
h_molt = x*y
h_div = x/y
```

Se invece volete fare operazioni matriciali, la sintassi è

%operazione%

In [ ]:
```
x = c(1,2,3,4)
y = c(1,2,3,4)

x+y

x+y[1:3]

x+y[1]
```

2 · 4 · 6 · 8

```
Warning message in x + y[1:3]:
"la lunghezza pi`u lunga dell'oggetto non `e un multiplo della lunghezza pi
`u corta dell'oggetto"
```

2 · 4 · 6 · 5

2 · 3 · 4 · 5

Creiamo una matrix

In [ ]:
```
x = matrix(c(1,2),nrow=2, ncol=3)
str(x)
x[ ,2 ]
```

```
x[1,3] = 10
x
x[1,3] = "A"
x
str(x)
x[1,3] = 2
str(x)
as.numeric(x)
```

```
 num [1:2, 1:3] 1 2 1 2 1 2
```

1 · 2

A matrix: 2
x 3 of type
dbl

1	1	10
2	2	2

A matrix:
2 x 3 of
type chr

1	1	A
2	2	2

```
 chr [1:2, 1:3] "1" "2" "1" "2" "A" "2"
 chr [1:2, 1:3] "1" "2" "1" "2" "2" "2"
```

1 · 2 · 1 · 2 · 2 · 2

Vediamo i fattori e come sono differenti dali caratteri

In [ ]:
```
# creo un vettore di caratteri

s = c("A", "B", "D")
str(s)
s[3] = "F"
s
```

```
 chr [1:3] "A" "B" "D"
```

'A' · 'B' · 'F'

Invece di chr, voglio un factor

In [ ]:
```
sfactor  = as.factor(s)
str(sfactor)
sfactor[3] = "A"
str(sfactor)

c = as.factor(c("A", "B","A","C","A"))
c
as.numeric(c)
```

```
 Factor w/ 3 levels "A","B","F": 1 2 3
 Factor w/ 3 levels "A","B","F": 1 2 1
```

A · B · A · C · A

▶ **Levels**:

1 · 2 · 1 · 3 · 1

vediamo le liste e ciome si creano

In [ ]:
```
xlist = list(obj1 = c(1,3,2), obj2  = matrix(c("A","B"), nrow=2, ncol=3), ob
str(xlist)
xlist$obj3
xlist[[3]]

xlist[[2]][2,]
```

```
List of 3
 $ obj1: num [1:3] 1 3 2
 $ obj2: chr [1:2, 1:3] "A" "B" "A" "B" ...
 $ obj3: num 3
```

3

3

'B' · 'B' · 'B'

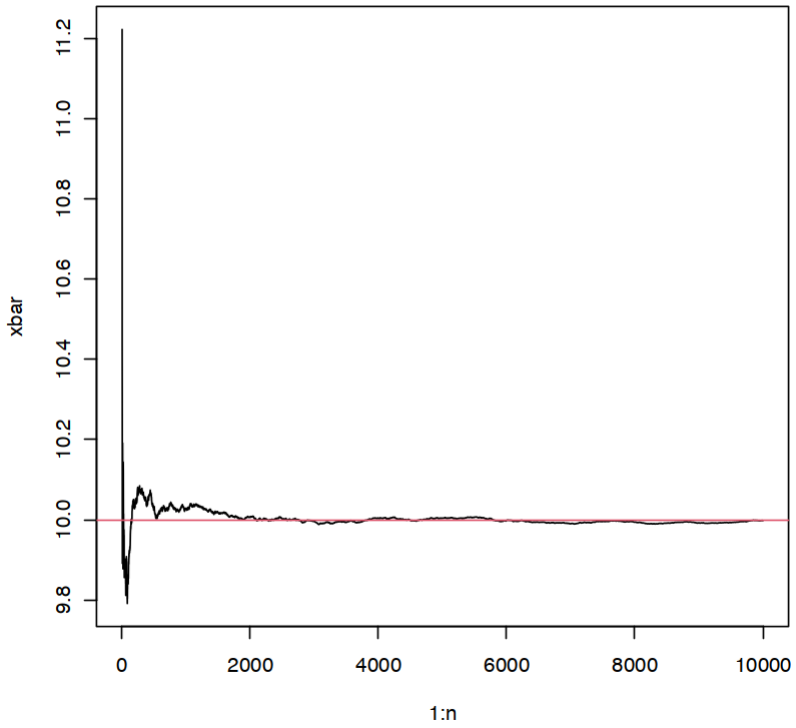
Leggi dei grandi numeri

abbiamo che se $X \sim F(\theta)$, iid, allora se $n \to \infty$

$$\bar{X}_n = \frac{\sum_{i=1}^{n} X_i}{n} \to E(X)$$

Assumiamo che $X_i \sim N(\mu, \sigma^2)$, e vediamo che succede per n che cresce

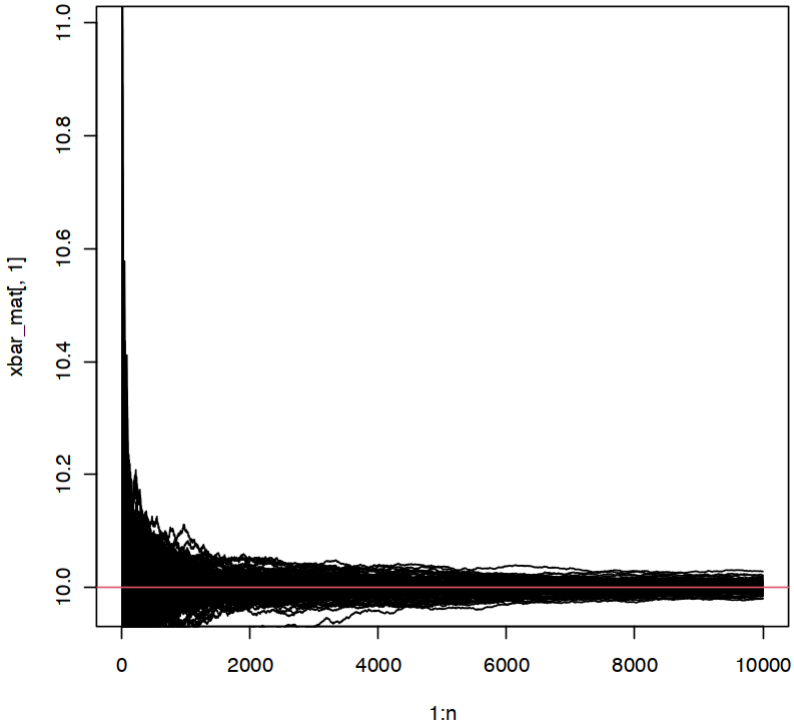
In [ ]:
```
n = 10000
mu = 10
sigma2 = 1

x = rnorm(n, mu, sigma2^0.5)

xbar = rep(NA, n)
xbar[1] = x[1]
for(i in 2:n)
{
    xbar[i] = sum(x[1:i])/i
}
plot( 1:n, xbar  , type="l" )
abline(h = mu, col = 2)
```



Adesso simuliamo diverse traiettorie

In [ ]:
```
nsim = 100
n = 10000
mu = 10
sigma2 = 1

xbar_mat = matrix(NA, nrow=n, ncol=nsim)


for(icol in 1:nsim)
{

    x = rnorm(n, mu, sigma2^0.5)
    xbar_mat[1,icol] = x[1]
    for(i in 2:n)
    {
        xbar_mat[i,icol] = sum(x[1:i])/i
    }

}
```

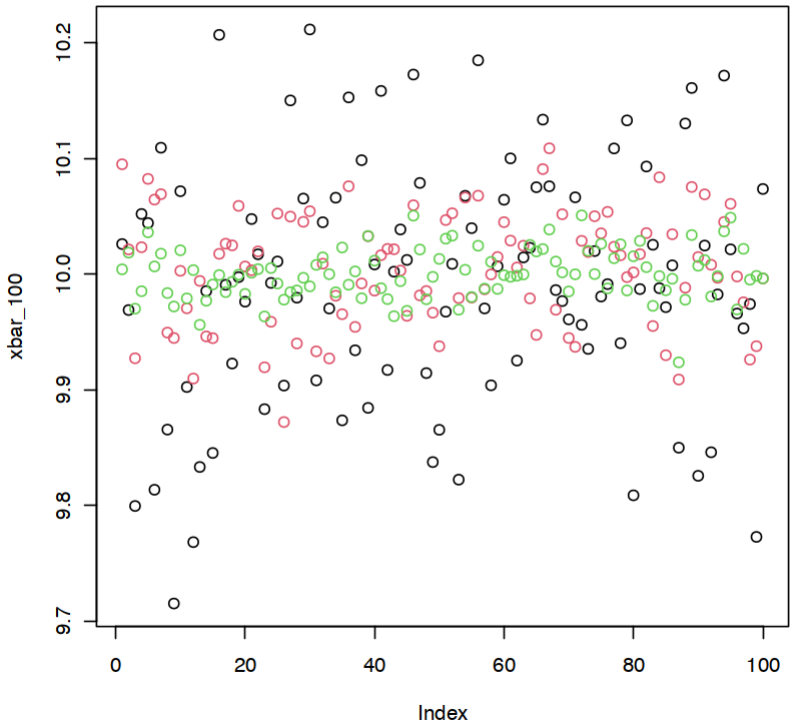
In [ ]:
```
plot( 1:n, xbar_mat[,1]  , type="l" )
for(icol in 2:nsim)
{
    lines( 1:n, xbar_mat[,icol] )
}

abline(h = mu, col = 2)
```



Quello che sappiamo è che

$$\bar{X}_n \sim N(\mu, \sigma^2/n)$$

In [ ]:
```
xbar_100 = xbar_mat[100,]
xbar_500 = xbar_mat[500,]
xbar_2000 = xbar_mat[2000,]
plot(xbar_100)
points(xbar_500, col=2)
points(xbar_2000, col=3)
```


xbar_100
9.7
9.8
9.9
10.0
10.1
10.2
0
20
40
60
80
100
Index