UNIVERSITÀ DI TRENTO

TRENTINO SCHOOL OF MANAGEMENT

— Parte 1. —

Statistica base con GNU-R

Paolo Bosetti (paolo.bosetti@unitn.it)

Data creazione: 2022-01-09 12:19:24

Indice

1 Numeri casuali e Distribuzioni

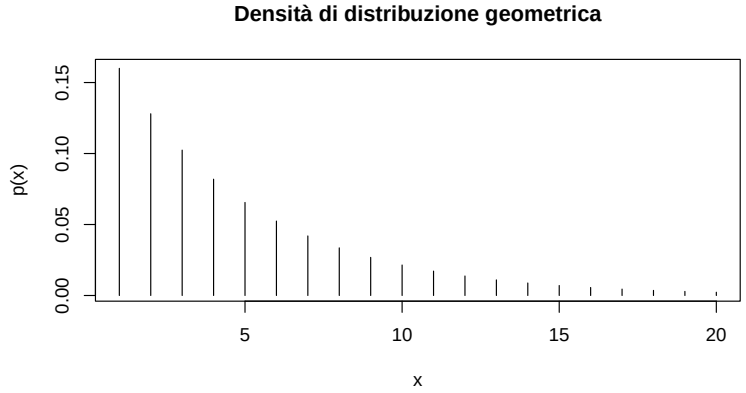
R dispone di una completa serie di funzioni per la gestione di numeri casuali, dalla generazione al calcolo della distribuzione. Le funzioni hanno nomi costituiti secondo questo schema: `<r|d|p|q><dist_name>()`, dove `dist_name` è un nome breve per la corrispondente distribuzione (ad es. `norm` per la *normale*) e il prefisso sta per:

- `r` (random): genera numeri casuali
- `d` (density): funzione densità di distribuzione (*Probability Density Function*, PDF)
- `p` (probability): funzione di probabilità cumulata (*Cumulative Distribution Function*, CDF)
- `q` (quantile): funzione quantile, inversa della CDF

1.1 Distribuzioni di probabilità discrete

Le distribuzioni discrete hanno valore solo sui numeri interi. Le più comuni sono `geom` (geometrica), `binom` (binomiale), `pois` (Poisson). I grafici vengoni generalmente riportati con linee verticali, usando l'opzione `typ="h"` nei comandi di plot:

```
x <- 1:20
plot(dgeom(x, prob=0.2),
     typ="h",
     xlab="x",
     ylab="p(x)",
     main="Densità di distribuzione geometrica")
```



La CDF è invece preferibile plottarla a step, opzione `typ="s"`. Per chiarezza, confrontare il grafico otenuto con `typ="S"` (maiuscolo).

Inoltre, ricordarsi che la CDF della variabile casuale X può riportare la coda alta (upper tail):

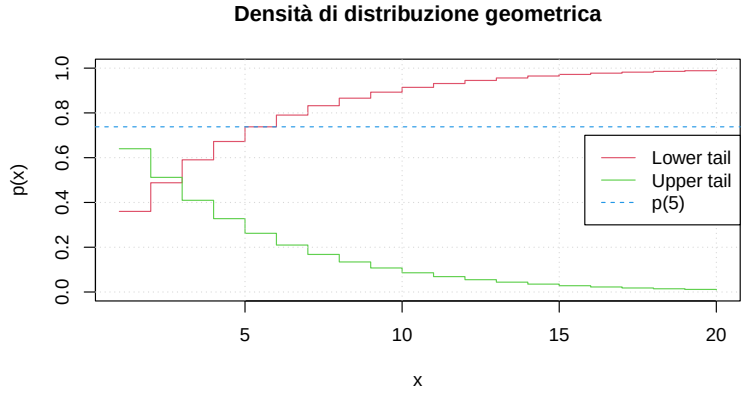
$$F_{X,U}(x) = \mathrm{P}(X \leq x) = \sum_{x_i \leq x} p(x_i)$$

e la coda bassa (lower tail):

$$F_{X,L}(x) = \mathrm{P}(X > x) = \sum_{x_i > x} p(x_i)$$

Per default, R considera la *lower tail* (`lower.tail=TRUE`):

```
plot(pgeom(x, prob=0.2),
     typ="s",
     xlab="x",
     ylab="p(x)",
     ylim=c(0,1),
     main="Densità di distribuzione geometrica",
     col="2")
lines(pgeom(x, prob=0.2, lower.tail=F),
     typ="s",
     xlab="x",
     ylab="p(x)",
     ylim=c(0,1),
     main="Densità di distribuzione geometrica",
     col=3)
grid()
abline(h=pgeom(5, prob=0.2), lty=2, col=4)
legend("right",
       legend=c("Lower tail", "Upper tail", "p(5)"),
       lty=c(1, 1, 2),
       col=2:4,
       bg="white")
```



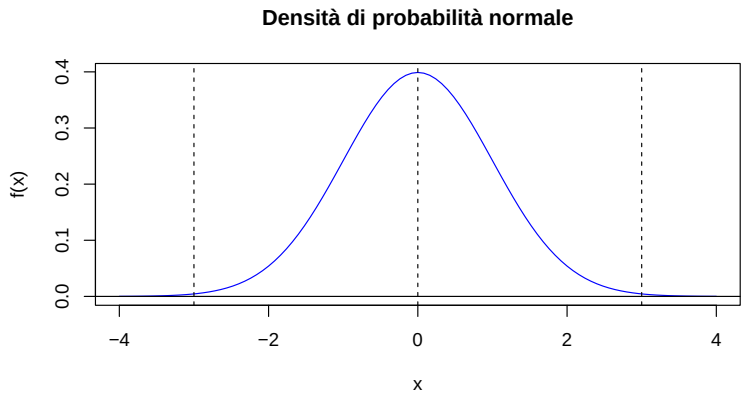
1.2 Distribuzioni di probabilità continue

Poco cambia rispetto alle distribuzioni discrete, salvo l'ovvia differenza che le funzioni hanno valore sui reali e che la CDF è definita come:

$$F_{X,U}(x) = \mathrm{P}(X \leq x) = \int_{-\infty}^{x} f(\xi)\mathrm{d}\xi$$

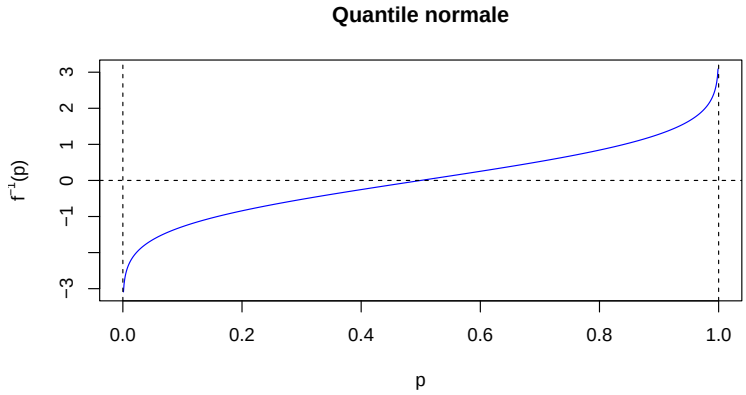
Inoltre, essendo la funzione continua posso creare il grafico con la funzione `curve()`:

```
curve(dnorm(x), from=-4, to=4, ylab="f(x)",
      main="Densità di probabilità normale",
      col="blue")
abline(v=c(-3, 0, 3), lty=2)
abline(h=0)
```



Per realizzare il grafico della funzione quantile è necessario ricordarsi che essa è l'inversa della CDF e, quindi, è definita solo nell'intervallo $(0, 1)$ e va all'infinito agli estremi:

```
curve(qnorm(x), from=0.001, to=0.999, n=1000,
      ylab=TeX("f^{-1}(p)"),
      xlab="p",
      col="blue",
      main="Quantile normale")
abline(h=0, lty=2)
abline(v=c(-1, 0, 1), lty=2)
```



Si noti l'uso della funzione `TeX` della librera `latex2exp` per inserire formule nelle etichette dei grafici ($f^{-1}(p)$).

Le funzioni che cominciano con `r` sono utili per *generare* vettori di numeri casuali. Per ottenere sempre la setessa sequenza pseudo-casuale si può impostare un seme:

```
set.seed(123)
x <- rnorm(100)
x[1:5]
```

```
## [1] -0.56047565 -0.23017749  1.55870831  0.07050839  0.12928774
```

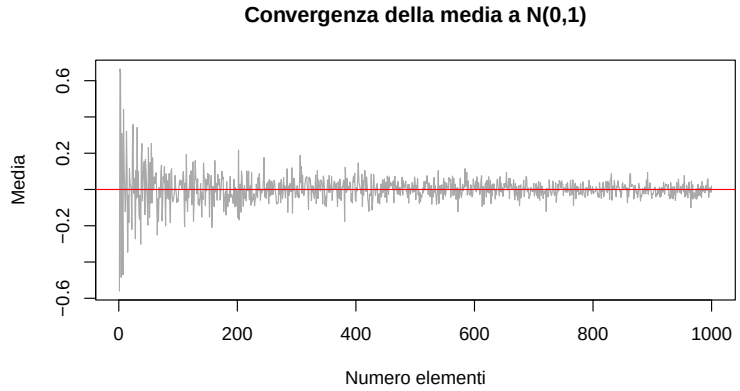
```
cat(paste("Media:", mean(x)),
    paste("Mediana:", median(x)),
    paste("Deviazione standard:", sd(x)),
    sep="\n")
```

```
## Media: 0.0904059086362066
## Mediana: 0.0617563090775401
## Deviazione standard: 0.912815879680979
```

Si noti come le funzioni `cat` e `paste` possono essere utilizzate per comporre *testo interpolato* (cioè testo che contiene i valori di espressioni valutate).

Possiamo studiare la *convergenza in distribuzione*:

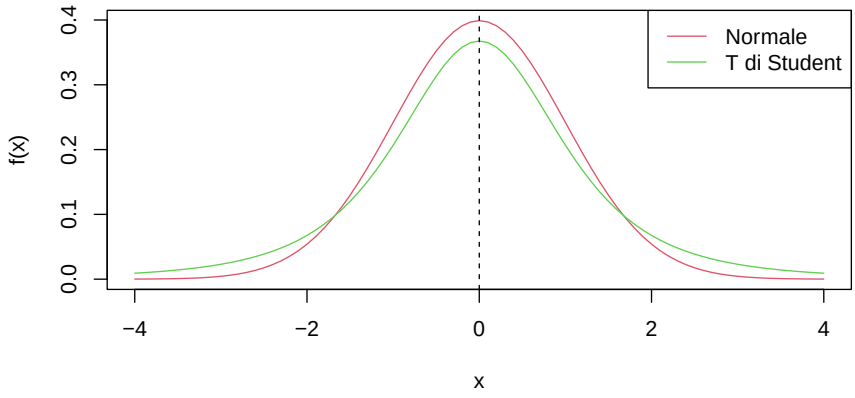
```
set.seed(123)
n <- 1:1000
plot(sapply(n, function(x) mean(rnorm(x))),
     typ="l",
     col="darkgrey",
     xlab="Numero elementi",
     ylab="Media",
     main="Convergenza della media a N(0,1)")
abline(h=0, col="red", lty=1)
```



Si noti che i dati sono stati generati non con un ciclo `for` ma con la funzione `sapply`: laddove possibile, le funzioni di mappatura sono sempre più veloci di un ciclo.

Vediamo ora come utilizzare i data frame per realizzare strutture dati più complesse:

```
df <- data.frame(x=seq(-4,4,0.1))
df$norm <- dnorm(df$x)
df$t <- dt(df$x, 3)
plot(norm~x, data=df, col=2, typ="l", ylab="f(x)")
lines(t~x, data=df, col=3)
legend("topright",
       legend=c("Normale", "T di Student"),
       lty=1,
       col=2:3)
abline(v=0, lty=2)
```



Si noti come, una volta creato, è possibile aggiungere nuove colonne ad un data frame con una semplice assegnazione mediante l'operatore `$`. Inoltre, la funzione `plot` è una *funzione generica*, che supporta cioè anche il metodo per la classe `formula`. In questo caso, la formula `norm~x` significa *colonna* ***norm*** *in funzione della colonna* ***x***.

2 Lettura e scrittura file

2.1 Scrivere file

In R scrivere dati su file è relativamente semplice. Ci sono sostanzialmente tre soluzioni:

1. *salvare* oggetti in formato proprietario R: `save()` (e l'opposto `load()`)
2. *scrivere* testo libero su file ASCII: `cat()`
3. *scrivere* dati tabulati ASCII: `write.table()` e `write.csv()`

La prima soluzione non permette lo scambio dati con altri software. La seconda soluzione è più flessibile, mentre la terza è più semplice.

In particolare, `cat()` e `write.table()` possono essere usate in sequenza per salvare una tabella anticipata da qualche riga di commento.

Creiamo una tabella delle probabilità cumulate della distribuzione T. Per inciso, simili tabelle erano utilizzate per effetuare i T-test prima dell'avvento dei calcolatori.

```
file <- "t_values.txt"
n <- 1:120
p <- c(0.4, 0.25, 0.1, 0.05, 0.025, 0.01, 0.005, 0.0025, 0.001, 0.0005)
m <- t(sapply(n, function(x) round(qt(p, x, lower.tail=F), 3)))
rownames(m) <- as.character(n)
colnames(m) <- as.character(p)
cat(file=file, "# Probabilità della distribuzione T\nDoF ")
write.table(m, file, quote = F, sep="\t", append = TRUE)
knitr::kable(head(m))
```

0.4	0.25	0.1	0.05	0.025	0.01	0.005	0.0025	0.001	5e-04
0.325	1.000	3.078	6.314	12.706	31.821	63.657	127.321	318.309	636.619
0.289	0.816	1.886	2.920	4.303	6.965	9.925	14.089	22.327	31.599
0.277	0.765	1.638	2.353	3.182	4.541	5.841	7.453	10.215	12.924
0.271	0.741	1.533	2.132	2.776	3.747	4.604	5.598	7.173	8.610
0.267	0.727	1.476	2.015	2.571	3.365	4.032	4.773	5.893	6.869
0.265	0.718	1.440	1.943	2.447	3.143	3.707	4.317	5.208	5.959

Come si vede, `cat()` oltre che per stampare stringhe in standard output può essere utilizzato per scrivere su file: è sufficiente passare il parametro `file`.

La matrice `m` può anche essere convertita in data frame per maggiore comodità, e esportata i ìn formato di interscambio CSV. Si noti comunque che `write.csv()` supporta in input sia matrici che data frame.

```
df <- as.data.frame(m)
write.csv(df, file="t_values_en.csv")
write.csv2(df, file="t_values_it.csv")
```

Si noti che `write.csv()` usa la virgola come separatore di campo e il punto come separatore dei decimali, mentre `write.csv2()` usa il punto e virgola come separatore di campo e la vorgola come separatore dei decimali. Quindi, `write.csv2()` è da usarsi se si intende importare il file creato, ad esempio, in versioni di Excel localizzate in Italiano o in lingue che usano la vorgola come separatore decimale.

2.2 Leggere da file

La lettura da file di testo libero può essere effettuata medante la funzione `scan()`. Tuttavia nella maggior parte dei casi è sufficiente leggere tabelle ASCII o csv. In questo caso si usano le funzioni `read.table()` o `read.csv()`/`read.csv2()`. Si noti che in questo caso la stringa che specifica il percorso di origine è un URI generico, quindi può essere sia un file locale che un percorso HTTP o HTTPS:

```
df <- read.table("http://repos.dii.unitn.it:8080/data/diet.dat", header=T)
str(df)
```

```
## 'data.frame':    24 obs. of  4 variables:
##  $ stdOrder: int  1 2 3 4 5 6 7 8 9 10 ...
##  $ runOrder: int  2 10 11 20 4 8 9 12 14 15 ...
##  $ diet    : chr  "A" "A" "A" "A" ...
##  $ cTime   : int  60 59 63 62 65 66 67 63 64 71 ...
```

Per comodità, visto che dalla stessa URI caricheremo altre risorse, ci definiamo una funzione di utilità:

```
mydata <- function(file) paste0("http://repos.dii.unitn.it:8080/data/", file)
df <- read.table(mydata("diet.dat"), header=T)
```

In particolare, l'opzione `header=T` specifica che i dati contengono i nomi delle colonne nella prima riga di intestazione.

2.3 Serializzazione

A volte è utile *serializzare* una struttura dati, ovvero convertirla in un formato che può essere scritto su file e passato ad una differente sessione di R. Ciò può essere fatto nei seguenti modi:

- `dump()`: converte un elenco di oggetti (passato come vettore di *nomi*) in una rappresentazione R e la salva su file (di default, `dumpdata.R`); questo file può essere ricaricato mediante `source()`
- `save()`: salva uno o più oggetti in formato binario, ricaricabili con `load()`
- `save.image()`: salva *tutto* l'ambiente in un file (default `.RData`): consente di chiudere la sessione e riavviare in seguito recuperando esattamente tutto quanto fatto precedentemente.

Si noti che `save()` e `save.image()` non sono garantiti funzionare tra una versione e l'altra di R, e sono quindi sconsigliati per il salvataggio di dati a lungo termine.

3 Statistica descrittiva

3.1 Stimatori

È spesso utile descrivere un campione di numeri casuali mediante *indicatori* (come media, moda, mediana, deviazione standard) e mediante grafici. Tra i metodi grafici più utili ci sono gli istogrammi, di box-plot e i diagrammi quantile-quantile.

Vediamo gli stimatori più comuni:

```
v <- rnorm(10)
mean(v)
```

```
## [1] -0.2065041
```

```
median(v)
```

```
## [1] -0.1000487
```

```
var(v)
```

```
## [1] 0.6719288
```

```
sd(v)
```

```
## [1] 0.8197126
```

```
sd(v) == sqrt(var(v))
```

```
## [1] TRUE
```

```
quantile(v)
```

```
##         0%        25%        50%        75%       100%
## -1.7821402 -0.4729879 -0.1000487  0.3573861  0.9733320
```

Si noti la funzione `quantile()`: l'argomento opzionale `probs` è il vettore di probabilità per cui si vogliono i quantili (default a `seq(0, 1, 0.25)`).

Purtroppo R non fornisce una funzione per calcolare la moda (cioè il valore più frequente). È però facile costruirla:

```
set.seed(123)
(l <- sample(letters, replace = T)) # campionamento con reinserimento
```

```
##  [1] "o" "s" "n" "c" "j" "r" "v" "k" "e" "t" "n" "v" "y" "z" "e" "s" "y" "y" "i"
## [20] "c" "h" "z" "g" "j" "i" "s"
```

```
unique(l) # valori unici
```

```
##  [1] "o" "s" "n" "c" "j" "r" "v" "k" "e" "t" "y" "z" "i" "h" "g"
```

```
match(l, unique(l)) # indici dei valori unici che costruiscono l
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10  3  7 11 12  9  2 11 11 13  4 14 12 15  5 13
## [26]  2
```

```
tabulate(match(l, unique(l))) # conta le ripetizioni degli indici
```

```
##  [1] 1 3 2 2 2 1 2 1 2 1 3 2 2 1 1
```

```
which.max(tabulate(match(l, unique(l)))) # posizione del massimo
```

```
## [1] 2
```

```
unique(l)[which.max(tabulate(match(l, unique(l))))] # moda
```

```
## [1] "s"
```

```
mymode <- function(x) {
  xu <- unique(x)
  xu[which.max(tabulate(match(x, xu)))]
}
mymode(l)
```

```
## [1] "s"
```

Si noti che la funzione `mode()` già esiste e ritorna lo *storage mode* di un oggetto. Inoltre, si noti che `mymode()` restituisce *il primo* elemento più frequente, tralasciando eventuali parimerito. In genere, è opportuno ordinare il vettore in modo da restituire il più comune e più grande (o più piccolo) elemento:

```
mymode(sort(l, decreasing = T))
```

```
## [1] "y"
```

È frequente il caso in cui i dati in ingresso hanno valori mancanti, rappresentabili in R con la costante speciale `NA`. Gli stimatori statistici hanno l'opzione `na.rm` (default `FALSE`) che specifica se rimuovere o meno i valori mancanti (e quindi modificare la dimensione del vettore) prima di calcolare la stima:

```
set.seed(123)
v <- sample(10)
```

```
v[sample(10, size=2)] <- NA
v
```

```
##  [1]  3 10  2  8 NA  9  1  7  5 NA
```

```
mean(v) # nota: x + NA = NA, per ogni x
```

```
## [1] NA
```

```
mean(v, na.rm=T)
```

```
## [1] 5.625
```

Le funzioni `na.fail()`, `na.omit()` sono d'aiuto a manipolare i casi di NA, e sono automaticamente invocate dalle funzioni che supportano la gestione dei NA. Spesso si decide di sostituire i NA con valori medi dei restanti elementi:

```
v[is.na(v)] <- mean(v, na.rm=T)
v
```

```
##  [1]  3.000 10.000  2.000  8.000  5.625  9.000  1.000  7.000  5.000  5.625
```

```
mean(v)
```

```
## [1] 5.625
```

Sono utili anche gli stimatori di covarianza:

$$\mathrm{COV}(X,Y) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_x)(y_i - \mu_y)$$

e correlazione:

$$\mathrm{CORR}(X,Y) = \frac{\mathrm{COV}(X,Y)}{\sigma_X \sigma_Y} \quad \in [-1,1]$$

In R:

```
set.seed(123)
n <- 10
x1 <- rnorm(n, 3, 0.5)
x2 <- rnorm(n, 6, 1)
x3 <- x1 * 2 + rnorm(n, sd=0.1)
c(cov(x1, x2), cov(x1, x3))
```

```
## [1] 0.2859477 0.4368317
```

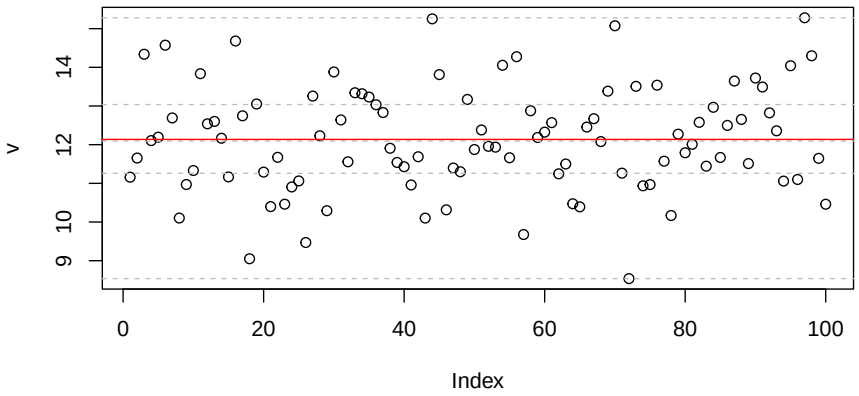
```
c(cor(x1, x2), cor(x1, x3))
```

```
## [1] 0.5776151 0.9957156
```

3.2 Metodi grafici

È spesso utile rappresentare un vettore di dati casuali mediante metodi grafici. Possiamo utilizzare un diagramma a dispersione per visalizzare l'andamento ed evidenziare eventuali tendenze, e un istogramma per studiarne la distribuzione. La funzione kernel densty è inoltre una versione continua dell'istogramma, molto utile quando la dimensione del campione è molto grande.

```
set.seed(123)
n <- 100
v <- rnorm(n, 12, 1.5)
plot(v)
abline(h=quantile(v), col="gray", lty=2)
abline(h=mean(v), col="red")
```

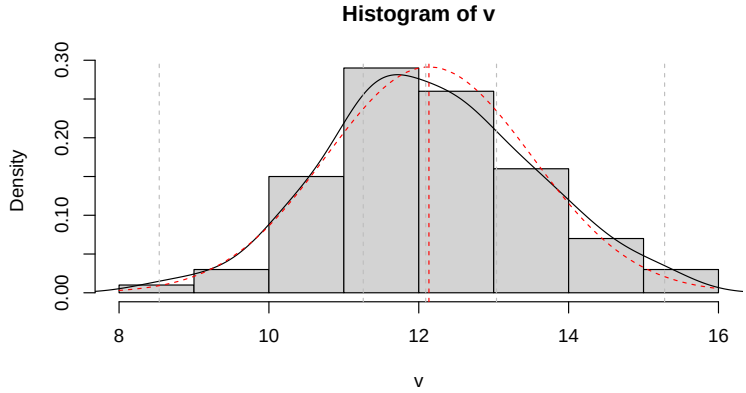


La serie non mostra tendenze o pattern (ovviamente!) e studiando i quantili osserviamo che la distribuzione appare leggermente gobab a sinistra, dato che la mediana è leggermente più bassa della media.

L'istogramma è creato dalla funzione `hist()`. Il numero di canne, o *bin*, in un istogramma è controllato dall'argomento `breaks`, che accetta o un vettore di punti di interruzione, o il nome dell'algoritmo (`"Sturges"`, `"Scott"`, `"FD"/"Freedman-Diaconis"`).

La versione continua dell'istogramma è ottenuta con la funzione `density()`, che è utile confrontare con la distribuzione di riferimento (in questo caso la normale):

```
hist(v, freq=F) # freq=T riporta i conteggi invece delle frequenze
lines(density(v))
curve(dnorm(x, mean(v), sd(v)), col="red", lty=2, add=T)
abline(v=quantile(v), col="gray", lty=2)
abline(v=mean(v), col="red", lty=2)
```

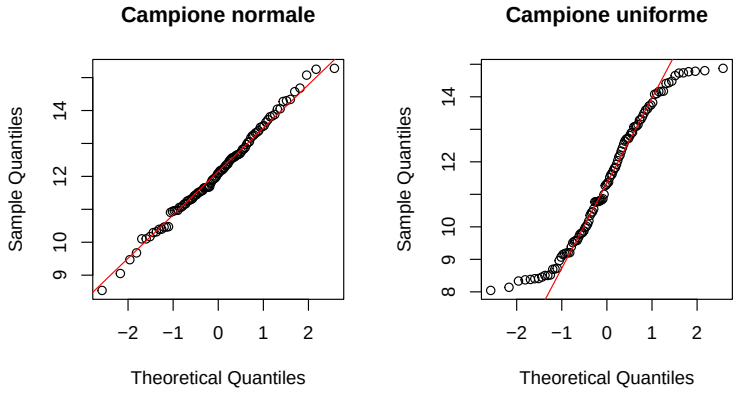


La densità e l'istogramma confermano una leggera gobba a sinistra, anche se—come c'era da aspettarsi—il campione appare distribuito normalmente.

La *verifica di normalità* è un tema molto importante in statistica: generalmente si preferisce associare a tale verifica un *test statistico* che consenta di associare una probabilità di errore al risultato (come vedremo nel capitolo successivo). Tuttavia i metodi grafici risultano comunque utili a integrare i test. Ancora più utile dell'istogramma è il **diagramma quantile-quantile** (o *QQ-plot*), che confronta i quantili teorici con quelli campionari. Tanto più il grafico è allineato alla diagonale, tanto più la distribuzione del campione è simile a quella di riferimento (tipicamente la normale).

```
vu <- runif(length(v), 8, 15)
par(mfrow=c(1,2)) # grafici multipli su una riga, due colonne
qqnorm(v, main="Campione normale")
qqline(v, col="red")
```

```
qqnorm(vu, main="Campione uniforme")
qqline(vu, col="red")
```

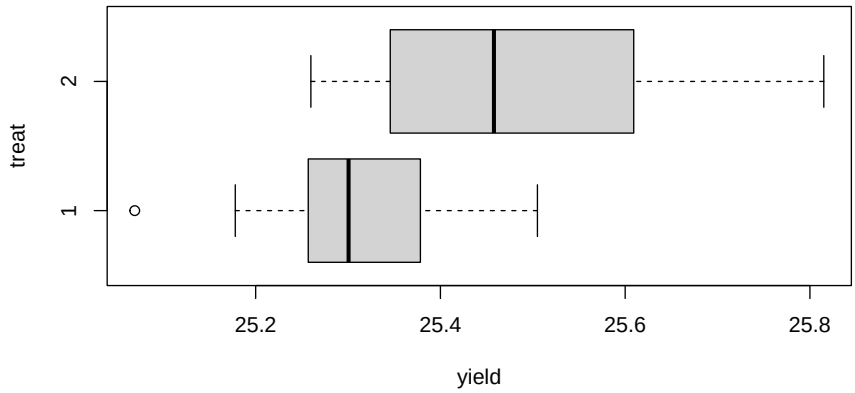


Nel caso di campioni bivariati o multivariati uno strumento molto utile per l'osservazione preliminare dei dati è il boxplot. Carichiamo i dati che riportano il tempo di reazione di un processo chimico in funzione di due diversi *trattamenti*, cioè condizioni di processo:

```
df <- read.table(mydata("twosample.dat"), header=T)
str(df)
```

```
## 'data.frame':    38 obs. of  2 variables:
##  $ treat: int  1 2 1 2 1 2 1 2 1 2 ...
##  $ yield: num  25.4 25.3 25.5 25.4 25.3 ...
```

```
boxplot(yield~treat, data=df, horizontal=T)
```



L'interfaccia più comoda utilizza una *formula*: `yield~treat` significa "plotta i valori della colonna `yield` raggruppati per valori della colonna `treat`". Ogni *box* è costituito da:

- una linea nera spessa, che rappresenta la mediana della classe
- un rettangolo, o *box*, che va dal primo al terzo quartile
- due baffi, o *whisker*, che si estendono al punto più estremo ma non oltre 1.5 volte l'intervallo interquartile (questo rapporto è configurabile)
- i punti più distanti dell'estensione del *whisker* sono marcati come possibili *outlier*

Tanto più due box sono sfalsati, tanto più è probabile che le due classi siano significativamente distinti, e viceversa. Un'analisi più dettagliata di cosa vuol dire "significativamente distinti" richiede ovviamente l'introduzione di un **test di inferenza**.

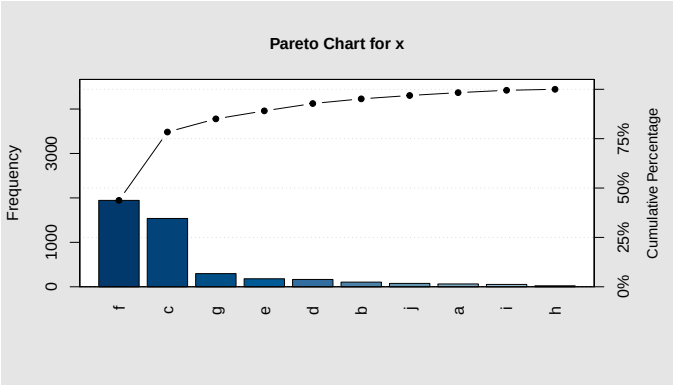
A volte è utile reaizzare un *diagramma di Pereto* per visualizzare la distribuzione cumulativa. La libreria `qcc` mette a disposizione il la funzione `pareto.chart()`:

```
library(qcc)
```

```
## Package 'qcc' version 2.7

## Type 'citation("qcc")' for citing this R package in publications.
```

```
set.seed(123)
n <- 10
x <- exp(rnorm(n, mean=5, sd=1.5))
names(x) <- letters[1:n]
pareto.chart(x)
```




```
##
## Pareto chart analysis for x
##        Frequency    Cum.Freq.   Percentage Cum.Percent.
##   f 1944.1837076 1944.1837076   43.7510035   43.7510035
##   c 1537.7298297 3481.9135372   34.6043549   78.3553584
##   g  296.3005495 3778.2140867    6.6678094   85.0231678
##   e  180.1755409 3958.3896276    4.0545863   89.0777541
##   d  164.9698264 4123.3594540    3.7124040   92.7901582
##   b  105.0812401 4228.4406941    2.3646992   95.1548574
##   j   76.0588243 4304.4995184    1.7115923   96.8664497
##   a   64.0258258 4368.5253442    1.4408073   98.3072570
##   i   52.9697355 4421.4950797    1.1920062   99.4992633
##   h   22.2514717 4443.7465513    0.5007367  100.0000000
```

Da cui si vede, ad esempio, che i primi due elementi, `f` e `c`, sono responsabili del 75% dell'effetto.

4 Statistica inferenziale

Fare inferenza significa estendere l'osserazione di un campione al comportamento dell'intera popolazione. Il più semplice test di inferenza è il test di Student, che studia la *media* di un campione. Ogni test di inferenza, per complicato che sia, si conclude sempre nel calcolare la probabilità di errore, detta *p-value*, di commettere un errore di tipo I, ossia rifiutare l'ipotesi nulla (non-significatività) quando essa è invece vera.

4.1 Test di Student

4.1.1 A un campione

Il test di Student può essere a uno o a due campioni, a uno o a due lati. Inoltre, se è un test a due campioni può assumere che i campioni abbiano varianza uguale o no. La funzione da utilizzare in questo caso è la `t.test()`.

Cominciamo con qualche esempio ad un campione.

```
set.seed(123)
n <- 10
m <- 12.1
s <- 0.1
v <- rnorm(n, m, s)
quantile(v)
```

```
##       0%      25%      50%      75%     100%
## 11.97349 12.04682 12.09202 12.13780 12.27151
```

Vogliamo valutare la coppia di ipotesi:

$$\begin{aligned} H_0: &\quad \mu = 12 \\ H_1: &\quad \mu \neq 12 \end{aligned}$$

e rifiutare l'ipotesi nulla quando la probabilità di un errore di tipo I è inferiore al valore di soglia $\alpha = 1\%$:

```
alpha <- 0.01
(tt <- t.test(v, alternative="two.sided", mu=12, conf.level=1-alpha))
```

```
##
##  One Sample t-test
##
## data:  v
## t = 3.5629, df = 9, p-value = 0.006091
## alternative hypothesis: true mean is not equal to 12
## 99 percent confidence interval:
##  12.00944 12.20548
## sample estimates:
## mean of x
##  12.10746
```

Il *p-value* risulta 0.61% che è minore della soglia α, quindi possiamo rifiutare H_0 con una probabilità d'errore pari a 0.61%.

Il risultato del test di Student riporta anche i limiti dell'intervallo di confidenza al 99%: tale intervallo è centrato sulla media del campione ed ha un'ampiezza che dipende dal parametro `conf.level`: se il valore target (12) è *esterno* a tale intervallo, allora possiamo rifiutare H_0 con una probabilità d'errore inferiore a 1%.

Ne consegue che si può anche fare a meno di specificare il valore target μ_0 e guardare solo l'intervallo di confidenza (che non dipende da μ_0):

```
t.test(v, conf.level=0.99)
```

```
##
##  One Sample t-test
##
## data:  v
## t = 401.42, df = 9, p-value < 2.2e-16
## alternative hypothesis: true mean is not equal to 0
## 99 percent confidence interval:
##  12.00944 12.20548
## sample estimates:
## mean of x
##  12.10746
```

Come si vede, un ipotetico $\mu_0 = 12$ è esterno all'intervallo di confidenza, mentre non lo sarebbe, ad esempio, $\mu_0 = 12.1$. Inoltre, è facile verificare che **l'ampiezza dell'intervallo di confidenza cresce se α cresce**.

Possiamo anche verificare un'ipotesi ad un lato:

$$H_0: \quad \mu = 12$$
$$H_1: \quad \mu > 12$$

```
t.test(v, alternative="greater", mu=12, conf.level=1-alpha)
```

```
##
##  One Sample t-test
##
## data:  v
## t = 3.5629, df = 9, p-value = 0.003046
## alternative hypothesis: true mean is greater than 12
## 99 percent confidence interval:
##  12.02236      Inf
## sample estimates:
## mean of x
##  12.10746
```

Come si vede, dato che escludiamo già che il valore atteso sia $\mu < 12$, la probabilità di errore, o *p-value*, risulta diminuita (è la metà).

4.1.2 A due campioni

Recuperiamo lo stesso data frame utilizzato per realizzare il box plot e verifichiamo quanto i due trattamenti possano essere considerati significativamente differenti con la coppia di ipotesi:

$$H_0: \quad \mu_1 = \mu_2$$
$$H_1: \quad \mu_1 \neq \mu_2$$

Si noti che la funzione `t.test()` per due campioni può essere invocata passando due vettori oppure una formula e un data frame (come per `boxplot()`). Quest'ultima soluzione è generalmente più comoda:

```
t.test(yield~treat, data=df)
```

```
##
##  Welch Two Sample t-test
##
## data:  yield by treat
## t = -4.2239, df = 30.469, p-value = 0.0002007
## alternative hypothesis: true difference in means between group 1 and group 2 is not equal to 0
## 95 percent confidence interval:
##  -0.26283515 -0.09158325
## sample estimates:
## mean in group 1 mean in group 2
##        25.30640        25.48361
```

```
# oppure:
# t.test(df$yield[df$treat==1,], df$yield[df$treat==2,])
```

Si noti che l'intestazione parla di "Welch Two Sample t-test": si tratta del test applicato ai casi in cui i due campioni provengano da popolazioni con uguale varianza. Questa condizione viene specificata con l'opzione `var.equal`, default a `FALSE`.

Prima di effettuare il test è quindi opportuno verificare questa condizione con un altro test: il test di varianza:

```
alpha <- 0.95
(vt <- var.test(yield~treat, data=df, conf.level=1-alpha))
```

```
##
##  F test to compare two variances
##
## data:  yield by treat
## F = 0.40247, num df = 18, denom df = 18, p-value = 0.06106
## alternative hypothesis: true ratio of variances is not equal to 1
## 5 percent confidence interval:
##  0.3905900 0.4147212
## sample estimates:
## ratio of variances
##          0.4024748
```

```
(tt <- t.test(yield~treat, data=df,
              var.equal=vt$p.value > alpha,
              conf.level=1-alpha))
```

```
##
##  Welch Two Sample t-test
##
## data:  yield by treat
## t = -4.2239, df = 30.469, p-value = 0.0002007
## alternative hypothesis: true difference in means between group 1 and group 2 is not equal to 0
## 5 percent confidence interval:
##  -0.1798618 -0.1745567
## sample estimates:
## mean in group 1 mean in group 2
##        25.30640        25.48361
```

Questa coppia di test mi conferma quindi che:

- i due campioni hanno varianze *significativamente* differenti, con *p-value* uguale a 0.061057
- le medie dei due campioni sono *significativamente* differenti, con *p-value* uguale a 2.006936×10^{-4}

4.1.3 T-test accoppiato

Spesso è utile raggruppare le osservazioni dei due campioni a due a due, in modo da escludere effetti ambientali ignote e incontrollate. È il caso ad esempio di quando si voglia confrontare la differente efficacia di due strumenti *indipendentemente dall'abiente in cui operano*, che si sa poter essere non costante.

Supponiamo ad esempio di voler confrontare la velocità di penetrazione di due diverse trivelle per prospezioni geologica, pur sapendo che essa dipende dalle caratteristiche del terreno, che sono a priori ignote e che possono cambiare localmente quando la distanza tra due perforazioni è superiore alla distanza minima tra due perforazioni.

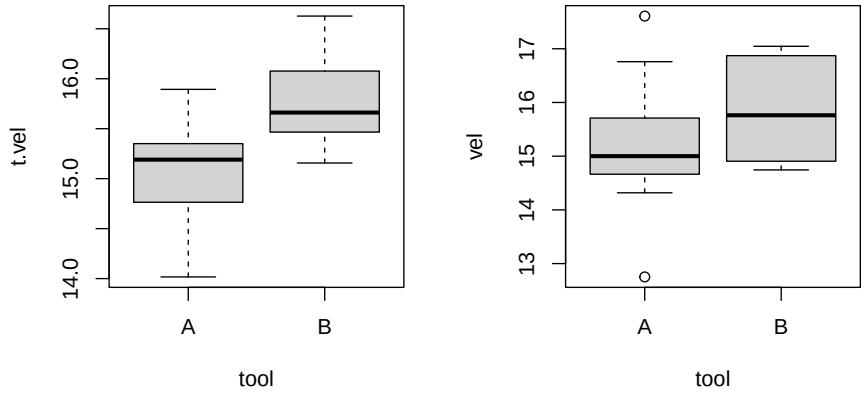
Generiamo artificialmente i dati per mostrare la superiorità del test accoppiato in queste condizioni.

```
set.seed(123)
n <- 10
v1 <- 15
v2 <- 16
s <- 0.5
df <- data.frame(tool=rep(c("A", "B"), n), soil=rep(rnorm(n, 0, 1), each=2), t.vel=NA)
df$t.vel[df$tool=="A"] <- rnorm(n, v1, s)
df$t.vel[df$tool=="B"] <- rnorm(n, v2, s)
df$vel <- df$t.vel + df$soil
str(df)
```

```
## 'data.frame':    20 obs. of  4 variables:
```

```
##  $ tool : chr  "A" "B" "A" "B" ...
##  $ soil : num  -0.56 -0.56 -0.23 -0.23 1.56 ...
##  $ t.vel: num  15.6 15.5 15.2 15.9 15.2 ...
##  $ vel  : num  15.1 14.9 14.9 15.7 16.8 ...
```

```
par(mfrow=c(1,2))
boxplot(t.vel~tool, data=df)
boxplot(vel~tool, data=df)
```



Come si vede, l'effetto del suolo (colonna `soil`) maschera l'effetto dell'utensile, che pure sembra significativo. Nella realtà, tuttavia, noi conosceremmo solo il comportamento complessivo, colonna `vel`, e quindi non saremmo in grado di stabilire una differenza tra le due trivelle. Ciò è ben evidente confrontando un test di Student normale e accoppiato:

```
t.test(vel~tool, data=df, var.equal=T)
```

```
##
##  Two Sample t-test
##
## data:  vel by tool
## t = -1.3622, df = 18, p-value = 0.1899
## alternative hypothesis: true difference in means between group A and group B is not equal to 0
## 95 percent confidence interval:
##  -1.7374146  0.3705954
## sample estimates:
## mean in group A mean in group B
##        15.17894        15.86235
```

```
t.test(vel~tool, data=df, paired=T)
```

```
##
##  Paired t-test
##
## data:  vel by tool
## t = -2.4788, df = 9, p-value = 0.03506
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -1.3070845 -0.0597347
## sample estimates:
## mean of the differences
##              -0.6834096
```

Come si vede, mentre il test normale non dà significatività, il test accoppiato conferma la differenza tra le due trivelle con un *p-value* inferiore al 5%: la trivella B è la più veloce.

4.1.4 Curve caratteristiche operative

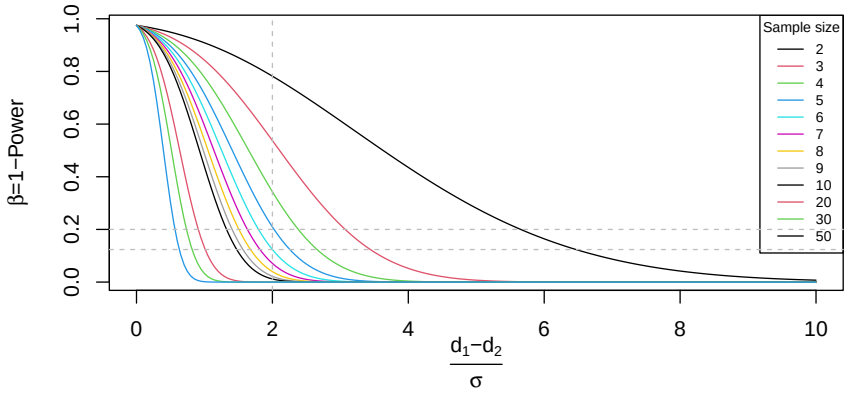
Il test di Student mi fornisce solo la probabilità di un errore di tipo I, detta α, corrispondente a un *falso positivo*. È utile però avere anche la probabilità di commettere un errore di tipo II, β, corrispondente ad un *falso negativo* o mancato allarme. Il complemento a 1 è noto come *potenza* di un test, ed è la sua affidabilità.

È evidente che la potenza di un test dipende dalla dimensione del campione: più piccolo è il campione, più assa sarà la potenza a pari condizioni.

Si costruiscono quindi le cosiddette *curve caratteristiche operative* (OCC), che riportano la potenza o il suo complemento a 1 per un test effettuato su due campioni con una determinata dimensione e una data differenza tra le medie in rapporto alla varianza.

Queste curve possono essere calcolate e costruite in R mediante le funzioni della famiglia `power.*.test()`. Per un T-test, ad esempio:

```
d <- seq(0, 10, 0.05)
nv <- c(3:10, 20, 30, 50)
plot(d, 1-power.t.test(2, d)$power, typ="l",
       ylab=TeX("$\\beta$=1-Power"),
       xlab=TeX("$\\frac{d_1-d_2}{\\sigma}$"))
for (i in seq_along(nv)) {
  lines(d, 1-power.t.test(nv[i], d)$power, typ="l", col=i+1)
}
legend("topright", lty=1, col=seq_along(nv+1),
       legend=c(2,nv),
       title="Sample size",
       cex=2/3)
abline(v=2, col="gray", lty=2)
abline(h=1-power.t.test(6, 2)$power, col="gray", lty=2)
abline(h=0.2, col="gray", lty=2)
```



Ad esempio, si capisce che con 6 osservazioni per campione è possibile discriminare una differenza relativa di 2 con una probabilità di falso negativo pari a 12.4%.

Ma le curve possono essere usate anche per *determinare la dimensione del campione*: per discriminare una differenza relativa pari a 2 con una probabilità di falso negativo inferiore al 20% ho bisogno di almeno 6 osservazioni per campione.

4.2 ANOVA a una via

Il test di Student consente di verificare la significatività di uno o due trattamenti. Se i trattamenti sono più di due, è consigliabile evitare di effettuare più test di Student su tutte le possibili combinazioni, perché in questo modo si finisce per ridurre la potenza complessiva del test, dato che le probabilità di errore si moltiplicano.

È preferibile effettuare quindi un test di analisi della varianza, o ANOVA, che studia la coppia di ipotesi:

$$\begin{aligned} H_0: &\quad \mu_i = \mu_j \quad \forall i \neq j \\ H_1: &\quad \exists (i,j) \mid \mu_i \neq \mu_j \end{aligned}$$

Si noti che il test *non dice* quali delle coppie di trattamenti (i, j) siano statistiamente differenti, ma solo che c'è *almeno* una coppia che lo è.

Vediamo come efettuare ANOVA in R su un dataset che contiene i valori di resistenza a trazione di filati misti in funzione di differenti percentuali di fibre di cotone. Il trattamento, in questo caso, è la percentuale di fibre di cotone nel filato.

```
df <- read.table(mydata("cotton.dat"), header=T)
str(df)
```

```
## 'data.frame':    25 obs. of  3 variables:
##  $ Run     : int  14 23 20 16 21 24 7 11 8 9 ...
##  $ Cotton  : int  15 15 15 15 15 20 20 20 20 20 ...
##  $ Strength: int  7 7 15 11 9 12 17 12 18 18 ...
```

La colonna `Run` riporta l'ordine, casuale, in cui sono state effettuate le prove. È sempre importante casualizzare la sequenza operativa in modo da distribuire uniformemente l'effetto di fattori ignoti e incontrollabili (es. temperatura). Per inciso, in fase di preparazione di un esperimento si può generare una tabella come sopra con i comandi:

```
levels <- seq(15, 35, by=5)
rep <- 5
df_prep <- data.frame(
  Run=sample(length(levels)*rep),
  Cotton=rep(levels, each=rep),
  Strength=NA)
df_prep <- df_prep[order(df_prep$Run),] # riordinata secondo la sequenza casuale
str(df_prep)
```

```
## 'data.frame':    25 obs. of  3 variables:
##  $ Run     : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Cotton  : num  20 35 35 25 15 20 30 15 20 25 ...
##  $ Strength: logi  NA NA NA NA NA NA ...
```

Questa tabella può poi essere ad esempio esportata in CSV per essere completata da chi esegue le prove, e poi nuovamente importata in R per l'analisi.

Tornando all'analisi della varianza, essa prevede che si costruisca prima un *modello statistico* che correli la variabile dipendente `Strength` con la variabile indipendente `Cotton`. Matematicamente, il modello può essere scritto come:

$$y_{ij} = \mu_i + \epsilon_{ij}, \quad i = 1, \ldots, n, \; j = 1, \ldots, r$$

o, più dettagliatamente, come:

$$y_{ij} = \mu + \tau_i + \epsilon_{ij}, \quad i = 1, \ldots, n, \; j = 1, \ldots, r$$

dove μ è la media complessiva delle osservazioni, τ_i è l'effetto del trattamento i, n è il numero di trattamenti, r è il numero di repliche per ciascun trattamento, e ϵ_{ij} sono i *residui*, cioè la differenza tra il modello di regressione $\hat{y}_i = \mu + \tau_i$ e la generica osservazione y_{ij}. Si ipotizza che i residui siano distribuiti in maniera normale a media nulla.

Sotto queste definizioni, la coppia di ipotesi di test può anche essere riscritta come:

$$\begin{aligned} H_0: &\quad \tau_i = 0 \quad \forall i = 1, \ldots, n \\ H_1: &\quad \exists i \mid \tau_i \neq 0 \end{aligned}$$

In R i modelli statistici vengono definiti mediante le *formule*. Nel nostro caso, il modello $y_{ij} = \mu_i + \epsilon ij$ può essere espresso come `Strength~Cotton` (lasciando i residui inespressi e sostituendo l'uguale con la `~`). La formula viene poi passata alla funzione `lm()` per creare l'oggetto modello. Il nome della funzione sta per *linear model*, perché consente di creare modelli statistici lineari *nei coefficienti*.

```
df.lm <- lm(Strength~Cotton, data=df)
anova(df.lm)
```

```
## Analysis of Variance Table
##
## Response: Strength
##           Df Sum Sq Mean Sq F value Pr(>F)
## Cotton     1  33.62  33.620  1.2816 0.2693
## Residuals 23 603.34  26.232
```

ATENZIONE: la prima cosa da verificare, in questi casi, è sempre il numero di gradi di libertà. Dalla teoria sappiamo che il numero di gradi di libertà del trattamento è $1 - n$, nel nostro caso 4, mentre la tabella riporta 1. Ciò è dovuto al fatto che la colonna `Cotton` è di tipo `int`: in questi casi, R assume che non sia una variabile di raggruppamento, ma una variabile numerica veera e propria che nel nostro caso solo incideltalmente assume valori uguali 5 a 5.

Per un'analisi della varianza, invece, `Cotton` dovrebbe rappresentare puramente una variabile categorica, detta *fattore*. Possiamo convertirla mediante la funzione `factor()`:

```
df$Cotton <- factor(df$Cotton, ordered=T) # ordered qui è opzionale
df.lm <- lm(Strength~Cotton, data=df)
anova(df.lm)
```

```
## Analysis of Variance Table
##
## Response: Strength
##           Df Sum Sq Mean Sq F value    Pr(>F)
## Cotton     4 475.76  118.94  14.757 9.128e-06 ***
## Residuals 20 161.20    8.06
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Ora il *p-value* del fattore `Cotton` risulta molto basso, il che significa che almeno uno dei trattamenti ha un effetto significativo sulla resistenza dei filati (ma non sappiamo quale).

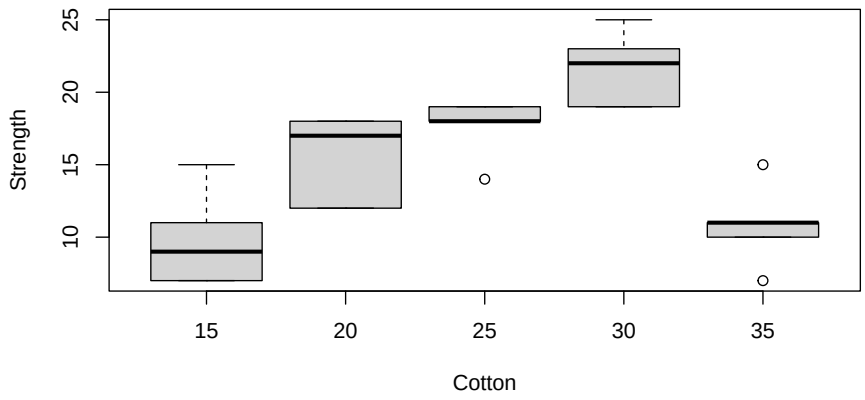
Si noti che la tabella ANOVA può essere ottenuta alternativamente con la funzione `aov()`, un'interfaccia più antica agli stessi algoritmi:

```
df.aov <- aov(Strength~Cotton, data=df)
summary(df.aov)
```

```
##             Df Sum Sq Mean Sq F value   Pr(>F)
## Cotton       4  475.8  118.94   14.76 9.13e-06 ***
## Residuals   20  161.2    8.06
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Possiamo osservare la situazione mediante un box plot:

```
boxplot(Strength~Cotton, data=df)
```

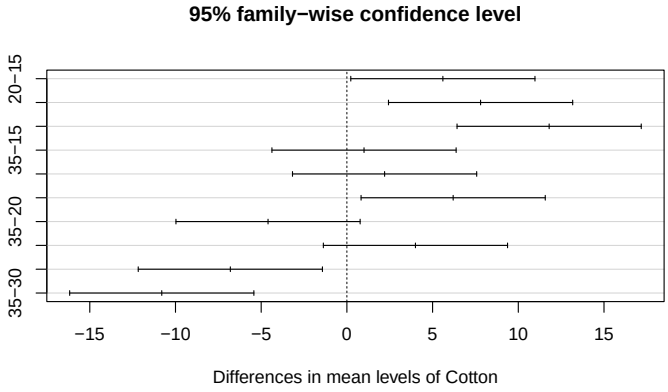


È evidente che almeno uno dei trattamenti è significatvo, ma quali di essi lo sono, reciprocamente? Come sopra detto è sconsigliabile effettuare coppie di test di Student, ma si può ottenere un'analisi più dettagliata mediante il **test di Tukey**.

4.3 Test di Tukey

Il test di Tukey calcola i *p-value* per tutte le possibili differenze tra coppie di trattamenti, compensando automaticamente gli effetti di combinazione delle probabilità di errore:

```
df.tuk <- TukeyHSD(df.aov)
plot(df.tuk)
```



In particolare, le differenze a cui corrisponde un *p-value* minore del 5% sono:

```
knitr::kable(df.tuk$Cotton[df.tuk$Cotton[,"p adj"] < 0.05,])
```

	diff	lwr	upr	p adj
20-15	5.6	0.2270417	10.972958	0.0385024
25-15	7.8	2.4270417	13.172958	0.0025948
30-15	11.8	6.4270417	17.172958	0.0000190
30-20	6.2	0.8270417	11.572958	0.0188936
35-25	-6.8	-12.1729583	-1.427042	0.0090646
35-30	-10.8	-16.1729583	-5.427042	0.0000624

Le altre differenze sono invece **non significative**.

4.4 ANOVA a due vie

È naturale estendere l'analisi della varianza a casi multivariati, in cui abbiamo due o più fattori, ciascuno con due o più trattamenti. Nel caso a due fattori con a e b trattamenti (o livelli) il modello statistico è:

$$y_{ijk} = \alpha_i + \beta_j + (\alpha\beta)_{ij} + \epsilon_{ijk}, \quad i = 1 \dots a, \ j = 1 \dots b, \ j = 1 \dots r$$

e la formula R corrispondente è `y~a+b+a:b`, dove `a:b` significa *interazione* tra fattori a e b; la sintassi algebrica delle formule in R definisce che `a*b==a+b+a:b`, quindi la formula può essere scritta più sinteticamente come `y~a*b`. Vediamo un esempio, caricando un data frame che contiene

```
df <- read.table(mydata("battery.dat"), header=T)
str(df)
```

```
## 'data.frame':    36 obs. of  6 variables:
##  $ RunOrder     : int  34 25 16 7 8 1 26 36 6 13 ...
##  $ StandardOrder: int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Temperature  : int  15 70 125 15 70 125 15 70 125 15 ...
##  $ Material     : int  1 1 1 2 2 2 3 3 3 1 ...
##  $ Repeat       : int  1 1 1 1 1 1 1 1 1 2 ...
##  $ Response     : int  130 34 20 150 136 25 138 174 96 155 ...
```

Il data frame contiene i risultati di un esperimento che sudia l'effetto di differenti elettroliti (colonna `Material`) e differenti temperature di esercizio (colonna `Temperature`) sul tempo di scarica di una batteria (colonna `Response`). Materiale e temperatura sono entrambi variabili categoriche, quindi vanno convertite in fattori prima di effettuare l'analisi della varianza:

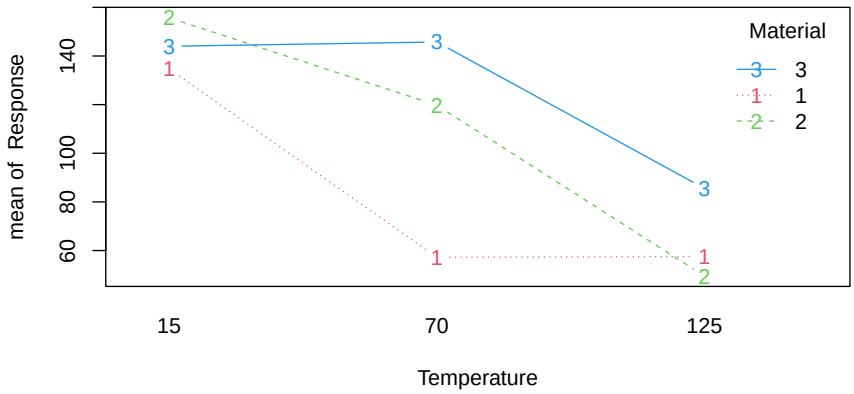
```
df$Material <- factor(df$Material)
df$Temperature <- factor(df$Temperature, ordered=T)
df.lm <- lm(Response~Material*Temperature, data=df)
anova(df.lm)
```

```
## Analysis of Variance Table
##
## Response: Response
##                      Df Sum Sq Mean Sq F value    Pr(>F)
## Material              2  10684  5341.9  7.9114  0.001976 **
## Temperature           2  39119 19559.4 28.9677 1.909e-07 ***
## Material:Temperature  4   9614  2403.4  3.5595  0.018611 *
## Residuals            27  18231   675.2
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Come si vede, risultano significativi entrambi i fattori, anche se l'interazione lo è meno dei fattori stessi. Si noti che *interazione* significa che *l'effetto di un fattore dipende dal livello dell'altro* (e viceversa).

Possiamo visualizzare gli effetti con un *interaction plot*:

```
with(df, interaction.plot(Temperature, Material, Response, typ="b", col=2:4))
```



Rimane il dubbio di quali differenze di materiale siano significative alle varie temperature. Possiamo realizzare tre test di Tukey:

```
for (t in levels(df$Temperature)) {
  cat(paste("Temperature: ", t, "°C", "\n"))
  print(TukeyHSD(aov(Response~Material, data=df[df$Temperature==t,]))$Material)
  cat("\n")
}
```

```
## Temperature:  15 °C
##       diff     lwr    upr     p adj
## 2-1  21.00 -45.344 87.344 0.6633090
## 3-1   9.25 -57.094 75.594 0.9205830
## 3-2 -11.75 -78.094 54.594 0.8756855
##
## Temperature:  70 °C
##     diff       lwr       upr        p adj
## 2-1 62.5  22.59911 102.40089 0.0045670072
## 3-1 88.5  48.59911 128.40089 0.0004209222
## 3-2 26.0 -13.90089  65.90089 0.2177840478
##
## Temperature:  125 °C
##     diff        lwr      upr     p adj
## 2-1   -8 -51.607557 35.60756 0.8673817
## 3-1   28 -15.607557 71.60756 0.2261123
## 3-2   36  -7.607557 79.60756 0.1062483
```

Dalle tabelle si deduce che a 15°C e a 125°C il materiale è ininfluente, mentre a 70°C i materiali 2 e 3 sono indistinguibili.

4.5 Verifica di normalità

L'analisi di varianza assume come ipotesi la normalità dei residui. È un'ipotesi abbastanza debole, nel senso che il test statistico su cui si basa ANOVA (un F-test) è robusto a modeste deviazioni dalla normalità. Tuttavia è sempre opportuno *verificare* che i residui siano *normali* e *privi di pattern*.

La *normalità* dei residui può essere verificata sia con metodi grafici che con test di inferenza. I test più comuni sono il test del Chi-quadro e il test di Shapiro-Wilk. Quest'ultimo è il più semplice da effetuare in R:

```
shapiro.test(residuals(df.lm))
```

```
##
##  Shapiro-Wilk normality test
##
```

```
## data:  residuals(df.lm)
## W = 0.97606, p-value = 0.6117
```

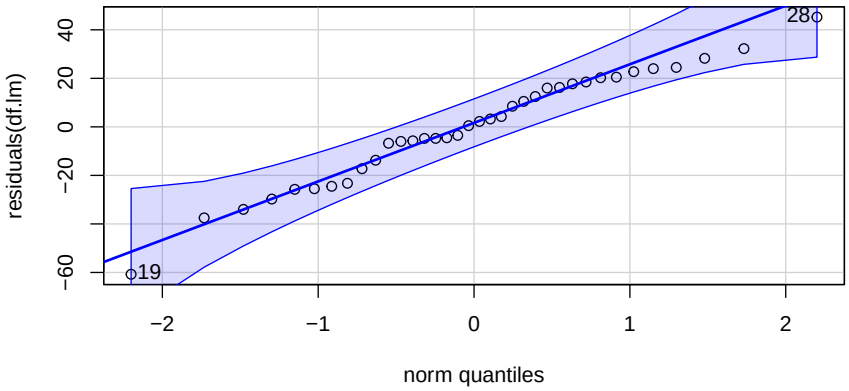
Come si vede si può utilizzare la funzione `residuals()` per estrarre i residui da un modello lineare. In alternativa si può anche scrivere `df.lm$residuals`. Nel nostro caso il *p-value* risulta grande: dato che l'ipotesi nulla del test di Shapiro-Wilk è quella di normalità, concludiamo che i residui nel nostro caso sono normali.

Il metodo grafico più comunemente usato per il controllo di normalità è il diagramma quantile-quantile, visto più sopra. La libreria `car` ne mette a disposizione una versione migliorata che riporta anche l'intervallo di confidenza:

```
library(car)
```

```
## Loading required package: carData
```

```
qqPlot(residuals(df.lm))
```

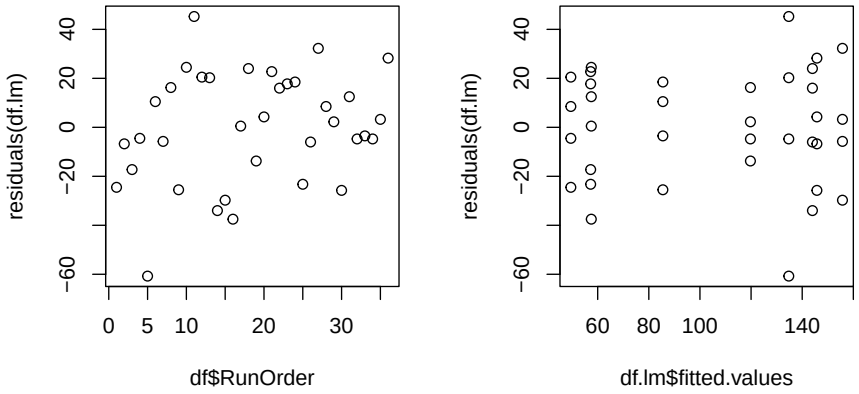



```
## [1] 19 28
```

Come si vede, tutti i punti stanno nella fascia di confidenza, il che conferma l'ipotesi di normalità dei residui.

Oltre alla normalità è importante verificare anche l'*assenza di pattern*: l'andamento dei residui non deve cioè mostrare dipendenze né dalla sequenza operativa (altrimenti significa che le condizioni cambiano durante le prove), né dal valore predetto (altrimenti il modello adottato non è adeguato):

```
par(mfrow=c(1,2))
plot(df$RunOrder, residuals(df.lm))
plot(df.lm$fitted.values, residuals(df.lm))
```



Escludendo due soli punti estremi (che sono pochi), non si evidenziano particolari pattern quindi si accetta il modello.

4.6 Altri test

4.6.1 Chi-quadro

Il test del Chi-quadro (distribuzione χ^2) consente di verificare l'ipotesi nulla che non ci sia correlazione tra due variabili categoriche, ad esempio la taglia di vestito e il sesso:

```
set.seed(123)
df <- data.frame(
  sex=sample(factor(c("M", "F"), ordered=F), size=100, replace=T),
  size=sample(factor(c("S", "M", "L"), ordered=T), size=100, prob=c(20,50,30), replace=T)
)
table(df)
```

```
##    size
## sex  L  M  S
##   F 15 18 10
##   M 16 32  9
```

```
chisq.test(df$sex, df$size)
```

```
##
##  Pearson's Chi-squared test
##
## data:  df$sex and df$size
## X-squared = 2.0858, df = 2, p-value = 0.3524
```

4.6.2 Kolmogorov-Smirnov

Il test KS consente di verificare l'ipotesi nulla che un dato campione provenga da una determinata popolazione di distribuzione nota (inclusi i parametri):

```
ks.test(rnorm(1000), "pnorm", 0, 1)
```

```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  rnorm(1000)
## D = 0.022946, p-value = 0.6683
## alternative hypothesis: two-sided
```

```
ks.test(rnorm(1000), "pt", 3)
```

```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  rnorm(1000)
## D = 0.060559, p-value = 0.001305
## alternative hypothesis: two-sided
```

```
ks.test(rnorm(1000), runif(100, -3, 3))
```

```
##
##  Two-sample Kolmogorov-Smirnov test
##
## data:  rnorm(1000) and runif(100, -3, 3)
## D = 0.195, p-value = 0.001988
## alternative hypothesis: two-sided
```

```
ks.test(rnorm(100, mean=1), rnorm(100, mean=1.5))
```

```
##
##  Two-sample Kolmogorov-Smirnov test
##
## data:  rnorm(100, mean = 1) and rnorm(100, mean = 1.5)
## D = 0.28, p-value = 0.0007873
## alternative hypothesis: two-sided
```

5 Piani fattoriali (DoE)

Per brevità considereremo qui solo i casi di un piano fattoriale completo e di uno frazionato, entrambi non replicati. Casi più semplici (piani non frazionati) possono essere dedotti facilmente da questo caso più completo.

Realizzare e analizzare un esperimento fatoriale richiede i seguenti passi:

1. definizione dei *fattori* e dei loro livelli
2. scelta del frazionamento e del numero di repliche
3. generazione della *design matrix*, cioè della lista di combinazioni di livelli
4. randomizzazione dell'ordine operativo
5. esecuzione prove e raccolta dati
6. ANOVA
7. verifica di adeguatezza del modello (normalità e assenza di pattern nei residui)
8. eventuale revisione del modello (Box-Cox)

Questi punti saranno esplorati in dettaglio nei punti seguenti per due esempi: piano fattoriale completo e frazionato.

5.1 Caso 1: piano fattoriale completo

5.1.1 Definizione fattori

Vogliamo studiare la velocità di trivellazione (Y) in funzione di alcuni parametri di processo (A–D):

- A = carico assiale
- B = portata fango di trivellazione
- C = velocità di rotazione
- D = tipo di fango di trivellazione
- Y = velocità di trivellazione

Di solito la scelta dei fattori è fatta in forma cautelativa: nel dubbio, ogni possibile parametro deve essere considerato come fattore.

5.1.2 Progettazione piano fattoriale

Decidiamo di effettuare un piano fattoriale 2^4 non replicato. Per ogni fattore individuiamo quindi due livelli, basso e alto, convenzionalmente indicati con `-` e `+`. Ovviamente, il *valore* di tali livelli dipende dal fattore (ad es. 400 N e 600 N per il carico assiale A, 100 rpm e 200 rpm per la velocità di rotazione C, ecc.).

```
lvl <- c('-', '+')
```

5.1.3 Design matrix

La matrice di progetto deve contenere tutte le possibili combinazioni dei due livelli per ogni fattore. Secondo l'ordine standard, i livelli vengono alternati da - a + con frequenza massima per il primo fattore e via via dimezzando la frequenza. In R, questo tipo di data frame può essere agevolmente costruito con il comando `expand.grid()`:

```
df <- expand.grid(A=lvl, B=lvl, C=lvl, D=lvl, Y=NA)
knitr::kable(head(df))
```

A	B	C	D	Y
-	-	-	-	NA
+	-	-	-	NA
-	+	-	-	NA
+	+	-	-	NA
-	-	+	-	NA
+	-	+	-	NA

5.1.4 Randomizzazione

È sempre necessario randomizzare la sequenza operativa in modo da distribuire omogeneamente effetti ignoti e incontrollati su tutti i fattori. Possiamo quindi aggiungere due colonne all'inizio del data frame con la sequenza standard e con la sequenza casualizzata:

```
df <- data.frame(
  StdOrder=seq_along(df$A),
  RunOrder=sample(length(df$A)),
  df
)
knitr::kable(head(df))
```

StdOrder	RunOrder	A	B	C	D	Y
1	1	-	-	-	-	NA
2	14	+	-	-	-	NA
3	9	-	+	-	-	NA
4	15	+	+	-	-	NA
5	8	-	-	+	-	NA
6	12	+	-	+	-	NA

Ora il data frame può essere salvato su un file esterno da utilizzare come log per la raccolta dati, dopo averlo ordinato secondo il `RunOrder`:

```
file <- "drill.txt"
cat(c("# Trivelazione",
      paste("# File creato il ", date()),
      "# Fattori:",
      "#   A = carico assiale, 40-60 N",
      "#   B = portata fango di trivellazione, 30-50 l/min",
      "#   C = velocità di rotazione, 100-200 rpm",
      "#   D = tipo di fango di trivellazione, A-B",
      "#   Y = velocità di trivellazione, m/min"),
    file="drill.txt",
    sep="\n")
write.table(df[order(df$RunOrder),], "drill.txt", quote=F, append=T)
```

5.1.5 Raccolta dati

Il file `drill.txt` completo dei dati `Y` viene poi caricato nuovamente in R, riordinando secondo l'ordine standard:

```
df <- read.table(mydata(file), header=T, stringsAsFactors=T)
df <- df[order(df$StdOrder),]
```

5.1.6 ANOVA

L'analisi comincia creando un modello lineare che comprenda tutti i fattori e tutte le possibili interazioni. Tale modello non può essere analizzato direttamente con ANOVA, dato che non avendo ripetizioni non consente di valutare la varianza:

```
df.lm <- lm(Y~A*B*C*D, data=df)
anova(df.lm)
```

```
## Warning in anova.lm(df.lm): ANOVA F-tests on an essentially perfect fit are
## unreliable
```

```
## Analysis of Variance Table
##
## Response: Y
##           Df  Sum Sq Mean Sq F value Pr(>F)
## A          1   3.367   3.367     NaN    NaN
## B          1 165.766 165.766     NaN    NaN
## C          1  43.362  43.362     NaN    NaN
## D          1  20.976  20.976     NaN    NaN
## A:B        1   1.392   1.392     NaN    NaN
## A:C        1   0.096   0.096     NaN    NaN
## B:C        1   9.120   9.120     NaN    NaN
## A:D        1   2.806   2.806     NaN    NaN
## B:D        1  10.144  10.144     NaN    NaN
## C:D        1   0.801   0.801     NaN    NaN
## A:B:C      1   0.106   0.106     NaN    NaN
## A:B:D      1   2.310   2.310     NaN    NaN
## A:C:D      1   1.369   1.369     NaN    NaN
## B:C:D      1   0.122   0.122     NaN    NaN
## A:B:C:D    1   1.177   1.177     NaN    NaN
## Residuals  0   0.000     NaN
```

È quindi necessario applicare il metodo di Daniel. La libreria `FrF2` mette a disposizione la funzione `DanielPlot()`:

```
library(FrF2)
```

```
## Loading required package: DoE.base
```

```
## Loading required package: grid
```

```
## Loading required package: conf.design
```

```
## Registered S3 method overwritten by 'DoE.base':
##   method           from
##   factorize.factor conf.design
```

```
##
## Attaching package: 'DoE.base'
```

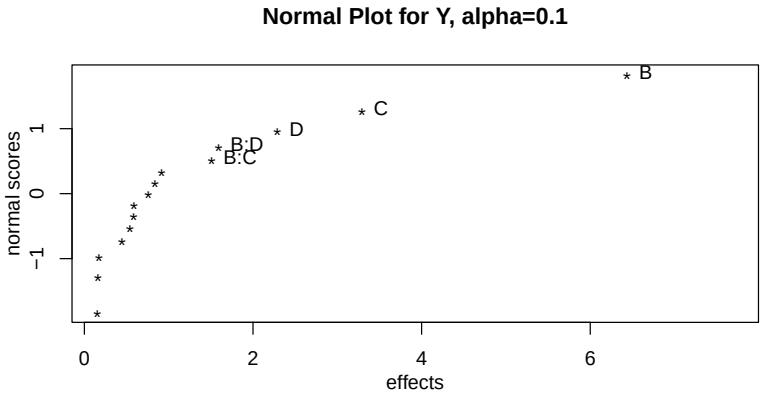
```
## The following objects are masked from 'package:stats':
##
##     aov, lm
```

```
## The following object is masked from 'package:graphics':
##
```

```
##     plot.design

## The following object is masked from 'package:base':
##
##     lengths
```

```
DanielPlot(df.lm, alpha=0.1)
```



Dal grafico risulta che solo i fattori B, C e D sono significativi, con una confidenza del 10%. Possiamo quindi riformulare il modello considerando solo questi fattori e le loro interazioni e rimuovendo completamente il fattore A. In questo modo il piano fattoriale 2^4 non ripetuto diventa un piano fattoriale 2×2^3:

```
df.lm2 <- lm(sqrt(Y)~B*C*D, data=df)
anova(df.lm2)
```

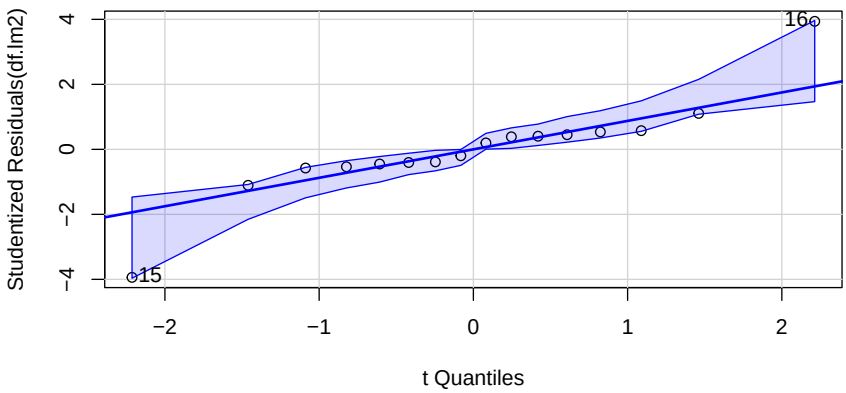
```
## Analysis of Variance Table
##
## Response: sqrt(Y)
##           Df Sum Sq Mean Sq  F value    Pr(>F)
## B          1 7.0253  7.0253 208.5012 5.175e-07 ***
## C          1 1.7282  1.7282  51.2890 9.599e-05 ***
## D          1 0.6925  0.6925  20.5538  0.001915 **
## B:C        1 0.0696  0.0696   2.0668  0.188482
## B:D        1 0.1896  0.1896   5.6267  0.045101 *
## C:D        1 0.0041  0.0041   0.1208  0.737093
## B:C:D      1 0.0013  0.0013   0.0373  0.851706
## Residuals  8 0.2696  0.0337
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Si confermano come significativo solo i fattori B, C e D e le interazioni tra B e C e tra B e D.

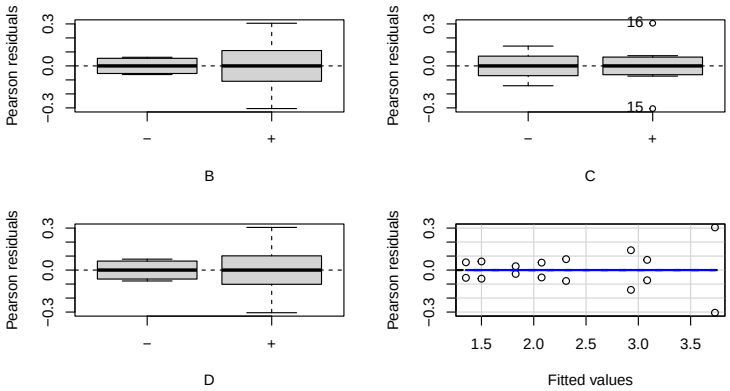
5.1.7 Verifica di adeguatezza

Verifichiamo l'adeguatezza del modello con QQ-plot dei residui e un'analisi dei pattern:

```
invisible(qqPlot(df.lm2))
```




```
residualPlots(df.lm2)
```

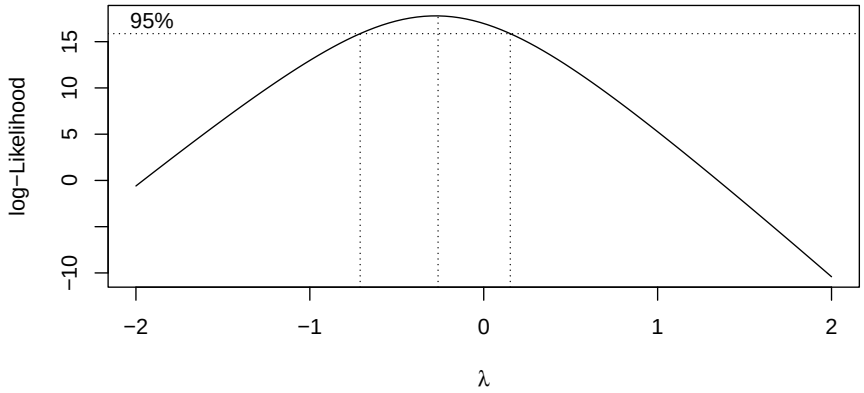


Come si vede i punti 15 e 16 sono sospetti ma, soprattutto, c'è un'evidente differenza delle distribuzioni dei residui per vari livelli e un pattern in aumento verso i *fitted values*.

5.1.8 Revisione del modello

È quindi opportuno cercare una trasformazione della resa Y che elimini questi problemi. Le trasformazioni possono essere individuate a mano, tentando varie combinazioni finché si trova quella che fornisce i residui migliori. Un metodo formale più comodo, invece, è il metodo Box-Cox, che individua il parametro di trasformazione λ che minimizza gli scarti (o massimizza la cosiddetta *Log-likelihood*):

```
library(MASS)
boxcox(Y~B*C*D, data=df)
```

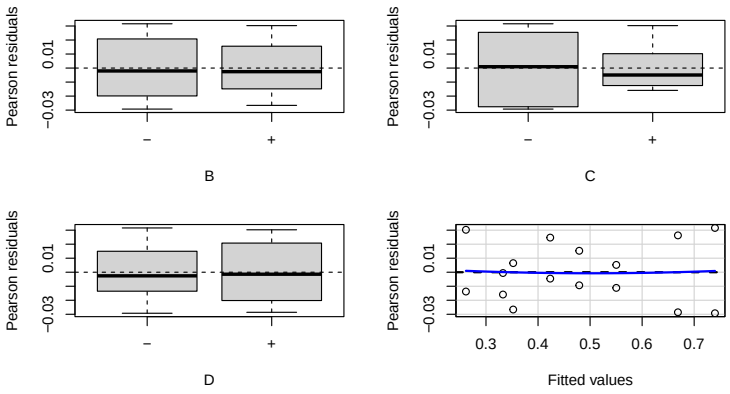


Nell'intervallo individuato tra le linee tratteggiate verticali e vicino all'ottimo ricadono sia il valore 0 (che corrisponde al logaritmo), sia il valore -1/2, cioè l'inverso della radice quadrata. Proviamo con quest'ultimo:

```
df.lm3 <- lm(1/sqrt(Y)~B*C+D, data=df)
anova(df.lm3)
```

```
## Analysis of Variance Table
##
## Response: 1/sqrt(Y)
##           Df   Sum Sq  Mean Sq F value    Pr(>F)
## B          1 0.285402 0.285402 477.466 2.064e-10 ***
## C          1 0.078621 0.078621 131.530 1.850e-07 ***
## D          1 0.020254 0.020254  33.884 0.0001157 ***
## B:C        1 0.009761 0.009761  16.330 0.0019449 **
## Residuals 11 0.006575 0.000598
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
residualPlots(df.lm3, test=F)
```

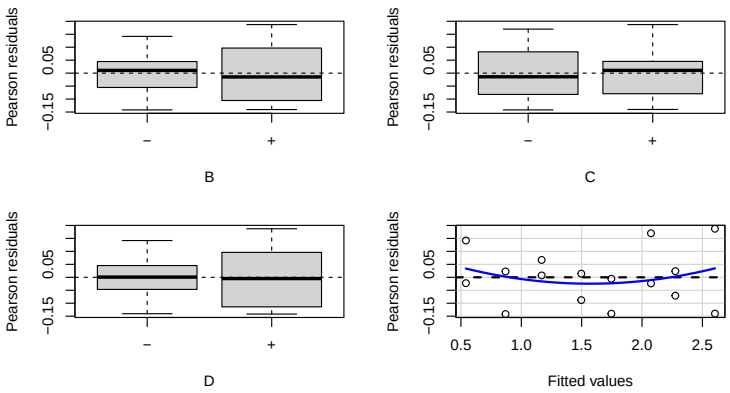


Proviamo ora con il logaritmo:

```
df.lm4 <- lm(log(Y)~B*C+D, data=df)
anova(df.lm4)
```

```
## Analysis of Variance Table
##
## Response: log(Y)
##           Df Sum Sq Mean Sq  F value    Pr(>F)
## B          1 5.3452  5.3452 360.3081 9.355e-10 ***
## C          1 1.3389  1.3389  90.2501 1.231e-06 ***
## D          1 0.4305  0.4305  29.0215 0.0002209 ***
## B:C        1 0.0095  0.0095   0.6387 0.4410937
## Residuals 11 0.1632  0.0148
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
residualPlots(df.lm4, test=F)
```



Dei due, è forse preferibile l'inverso della radice quadrata, dato che ha distribuzioni dei residui più omogenee.

Si accetta quindi il modello `1/sqrt(Y)~B*C+D`.

5.2 Caso 2: piano fattoriale frazionato

In questo caso vogliamo analizzare la resa di un impianto di fabbricazione di circuiti integrati (fotolitografia) considerando i seguenti fattori:

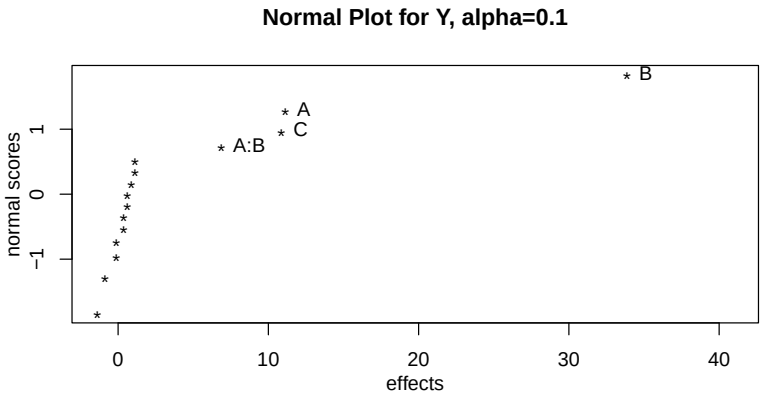
- A = apertura
- B = tempo di esposizione
- C = tempo di sviluppo
- D = parametro di dimensione delle maschere
- E = tempo di attacco
- Y = risposta

Progettiamo un piano fattoriale 2^{5-1}_{IV} non replicato, con la relazione definente $I = ABCDE$. Dovremmo seguire i passi già visti al punto precedente, ma per brevità inseriamo direttamente i risultati nel data frame:

```
lvl <- c(-1,1)
df <- expand.grid(A=lvl, B=lvl, C=lvl, D=lvl) # E=ABCD
attach(df)
df$E <- A*B*C*D # E=ABCD
detach(df)
df$Y <- c(
  8, 9, 34, 52,
  16, 22, 45, 60,
  6, 10, 30, 50,
  15, 21, 44, 63
)
for (k in LETTERS[1:5]) df[k] <- factor(df[[k]])
```

Dato che il piano non è replicato dobbiamo applicare il metodo di Daniel:

```
df.lm <- lm(Y~A*B*C*D*E, data=df)
DanielPlot(df.lm, alpha=0.1)
```



Il modello ridotto può quindi essere `Y~A*B+C`. Per essere conservativi possiamo comunque usare `Y~A*B*C` e avere comunque abbastanza ridondanza da effettuare una ANOVA:

```
df.lm <- lm(Y~A*B*C, data=df)
anova(df.lm)
```

```
## Analysis of Variance Table
##
## Response: Y
##           Df Sum Sq Mean Sq   F value    Pr(>F)
## A          1  495.1   495.1  214.0811 4.672e-07 ***
## B          1 4590.1  4590.1 1984.8919 7.112e-11 ***
## C          1  473.1   473.1  204.5676 5.570e-07 ***
## A:B        1  189.1   189.1   81.7568 1.791e-05 ***
## A:C        1    0.6     0.6    0.2432    0.6351
## B:C        1    1.6     1.6    0.6757    0.4349
## A:B:C      1    7.6     7.6    3.2703    0.1082
## Residuals  8   18.5     2.3
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

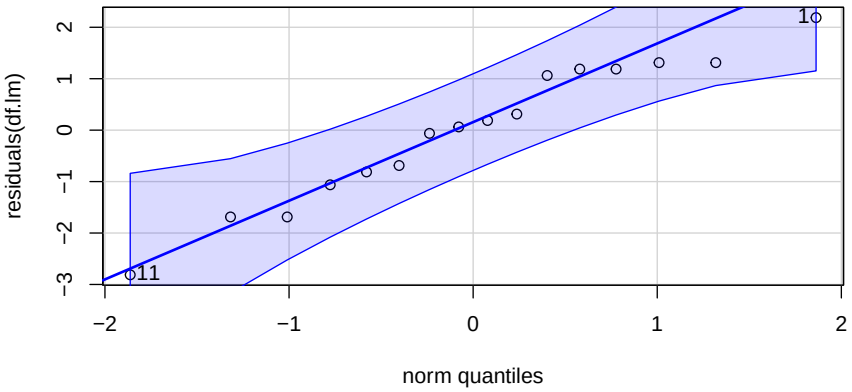
Si conferma quindi il modello `Y~A*B+C`.

```
df.lm <- lm(Y~A*B+C, data=df)
anova(df.lm)
```

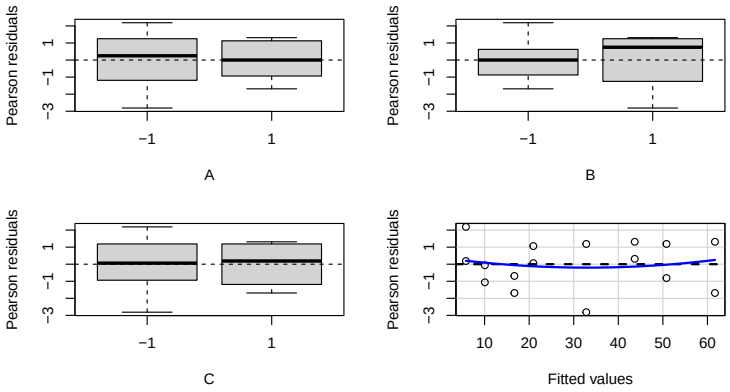
```
## Analysis of Variance Table
##
## Response: Y
##           Df Sum Sq Mean Sq  F value    Pr(>F)
## A          1  495.1   495.1  193.195 2.535e-08 ***
## B          1 4590.1  4590.1 1791.244 1.560e-13 ***
## C          1  473.1   473.1  184.610 3.214e-08 ***
## A:B        1  189.1   189.1   73.781 3.302e-06 ***
## Residuals 11   28.2     2.6
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Ora è necessario verificare l'adeguatezza:

```
invisible(qqPlot(residuals(df.lm)))
```

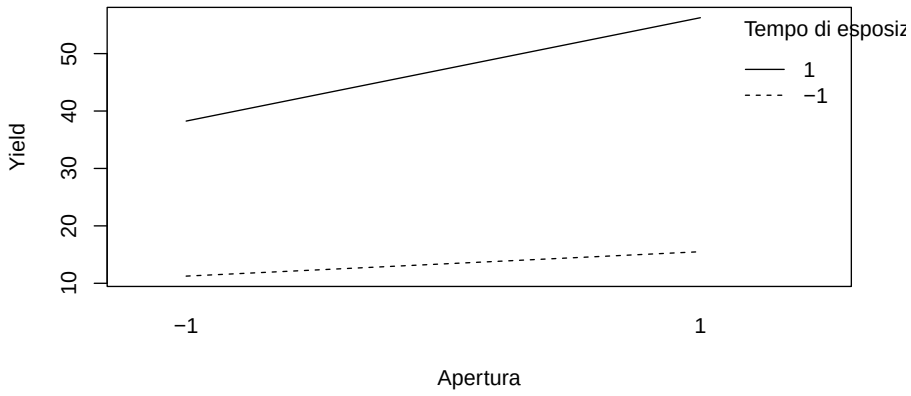



```
residualPlots(df.lm, test=F)
```

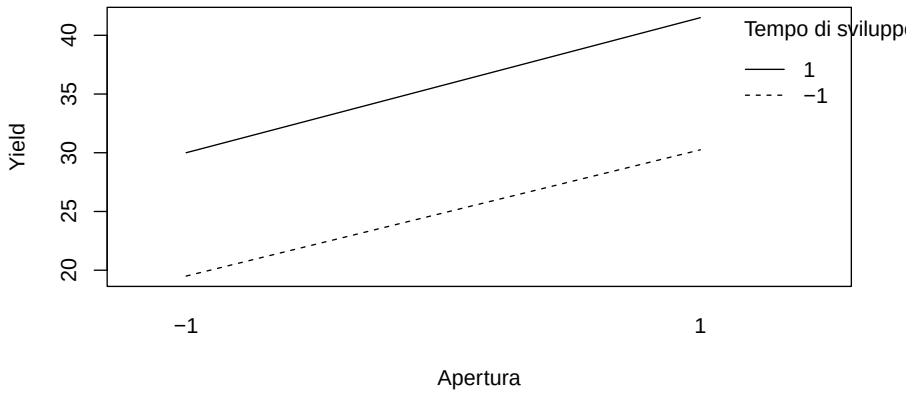


Non risultano particolari problemi quindi possiamo accettare il modello e studiare il processo con i grafici di interazione:

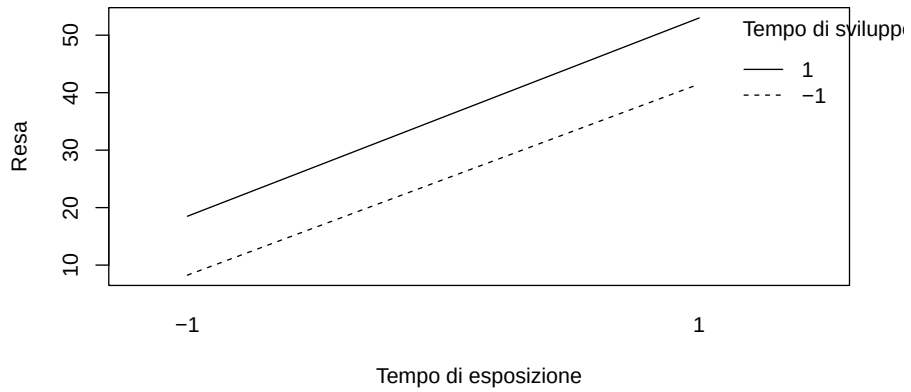
```
attach(df)
interaction.plot(A, B, Y, xlab="Apertura", ylab="Yield", trace.lab="Tempo di esposizione")
```




```
interaction.plot(A, C, Y, xlab="Apertura", ylab="Yield", trace.lab="Tempo di sviluppo")
```




```
interaction.plot(B, C, Y, xlab="Tempo di esposizione", ylab="Resa", trace.lab="Tempo di sviluppo")
```




```
detach(df)
```