Computational Methods and Principles of Bayesian Inference

Tapio Nummi
tapio.numme@tuni.fi

Arto Luoma
arto.luoma@tuni.fi

Fall 2023

Introduction

Some R features

Matrix computations

Computer intensive methods

Principles of Bayesian inference

Introduction

Introduction

Topics to be considered:

- Introduction to R language
- Some matrix operations
- Simulation
  - Monte Carlo methods
  - Simulation of random numbers
  - Simulation of random variables from known distribution
- Computer intensive methods
  - Permutation test
  - Cross-validation
  - Jackknife
  - Bootstrap
- Principles of Bayesian inference
  - The Maximum Likelihood method
  - The Law of Total Probability
  - Bayes Theorem
  - Bayesian analysis

What is R?

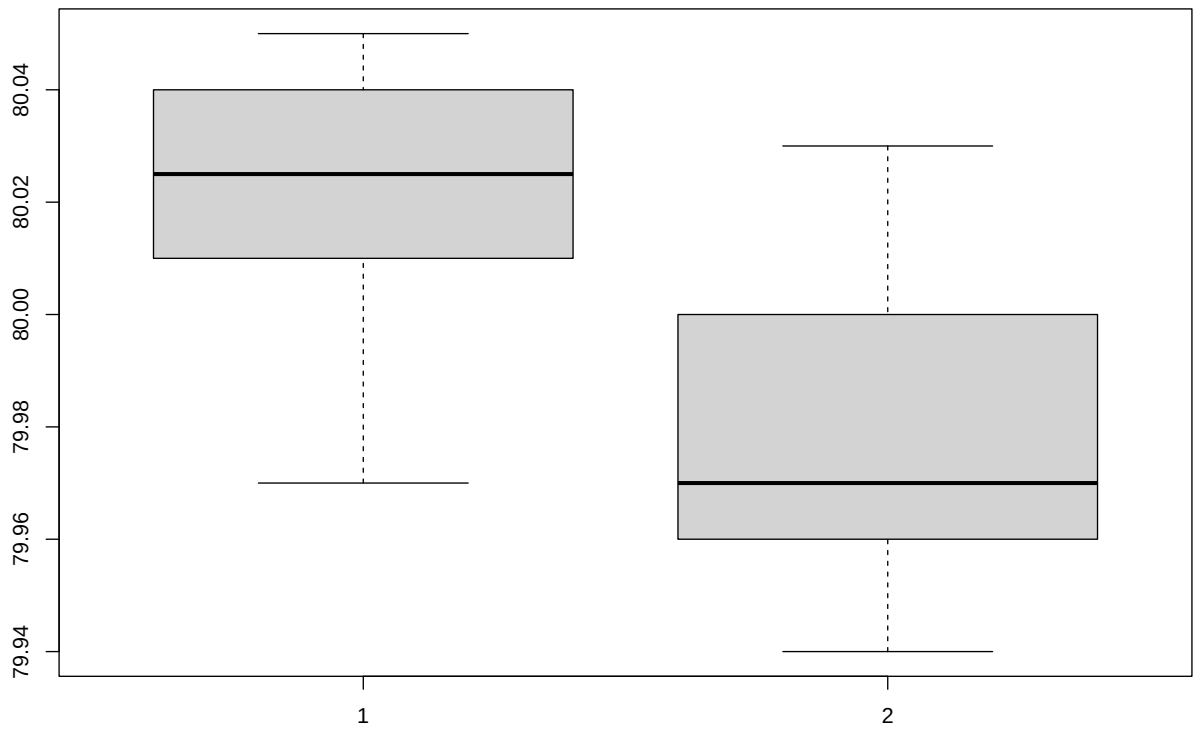
- R is a statistical analysis system created by Ros Ihaka & Robert Gentleman.
- R is both a language and software; its remarkable features are:
  - efficient data handling and storage,
  - a suite of operators for calculations on arrays, matrices, and other complex operations,
  - a large, coherent, integrated collection of tools for statistical analysis,
  - numerous graphical facilities, and
  - programming language facilities.
- R is freely distributed.
- R is available for many operation systems including Windows, Unix, Linux, macOS etc.

For more material see the website: https://www.r-project.org/.

See also the contributed material by Maindonald *Using R for Data Analysis and Graphics – An Introduction*.

Simple R examples

```
a <- c(79.98,80.04,80.02,80.04,80.03,80.04,79.97,80.05,80.03,80.02,80.00,80.02)
b <- c(80.02,79.94,79.98,79.97,80.03,79.95,79.97)
boxplot(a,b)
```



```r
t.test(a,b)
```

```
##
##  Welch Two Sample t-test
##
## data:  a and b
## t = 2.7381, df = 9.88, p-value = 0.02112
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  0.00739569 0.07260431
## sample estimates:
## mean of x mean of y
##     80.02     79.98
```

```r
var.test(a,b)
```

```
##
##  F test to compare two variances
##
## data:  a and b
## F = 0.54545, num df = 11, denom df = 6, p-value = 0.3632
## alternative hypothesis: true ratio of variances is not equal to 1
## 95 percent confidence interval:
##  0.1008278 2.1167188
## sample estimates:
## ratio of variances
##          0.5454545
```

```r
t.test(a,b, var.equal=T)
```

```
##
##  Two Sample t-test
##
## data:  a and b
## t = 2.9736, df = 17, p-value = 0.008521
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  0.01161907 0.06838093
## sample estimates:
## mean of x mean of y
##     80.02     79.98
```

```r
x <- runif(90)
x
```

```
##  [1] 0.451446014 0.180226504 0.303841808 0.166568921 0.283306815 0.132241166
##  [7] 0.113003061 0.873402377 0.402877737 0.450523604 0.431639661 0.230640675
## [13] 0.747558007 0.921275473 0.933595363 0.610128442 0.114519875 0.397290532
## [19] 0.221478159 0.898448580 0.756822964 0.012483260 0.526826740 0.638653502
## [25] 0.739761710 0.744834932 0.342402134 0.068242663 0.979720374 0.988538467
## [31] 0.346430909 0.922390601 0.430858555 0.063515225 0.331261619 0.245578734
## [37] 0.687306069 0.968339661 0.677709863 0.579252150 0.001999601 0.945979547
## [43] 0.697423714 0.475738388 0.404211729 0.154854480 0.221918321 0.400205006
## [49] 0.153370066 0.068244345 0.083512332 0.709137392 0.061277013 0.779316165
## [55] 0.369249075 0.434249500 0.117996885 0.919284253 0.753426433 0.252534187
## [61] 0.774323259 0.680458257 0.073400542 0.752330818 0.415728462 0.008207312
## [67] 0.736475524 0.090472981 0.868324575 0.740264174 0.584649870 0.779369466
## [73] 0.041230023 0.102986051 0.667170376 0.836556946 0.592103329 0.425008520
## [79] 0.048575619 0.641023746 0.144116315 0.840206054 0.617277231 0.159696013
## [85] 0.379917283 0.138751179 0.570963477 0.967978221 0.936129274 0.587443511
```

```r
x <- trunc(5*x)
x
```

```
##  [1] 2 0 1 0 1 0 0 4 2 2 2 1 3 4 4 3 0 1 1 4 3 0 2 3 3 3 1 0 4 4 1 4 2 0 1 1
## [39] 3 2 0 4 3 2 2 0 1 2 0 0 0 3 0 3 1 2 0 4 3 1 3 3 0 3 2 0 3 0 4 3 2 3 0 0
## [77] 2 2 0 3 0 4 3 0 1 0 2 4 4 2
```

```r
table(x)
```

```
## x
##  0  1  2  3  4
## 24 13 17 21 15
```

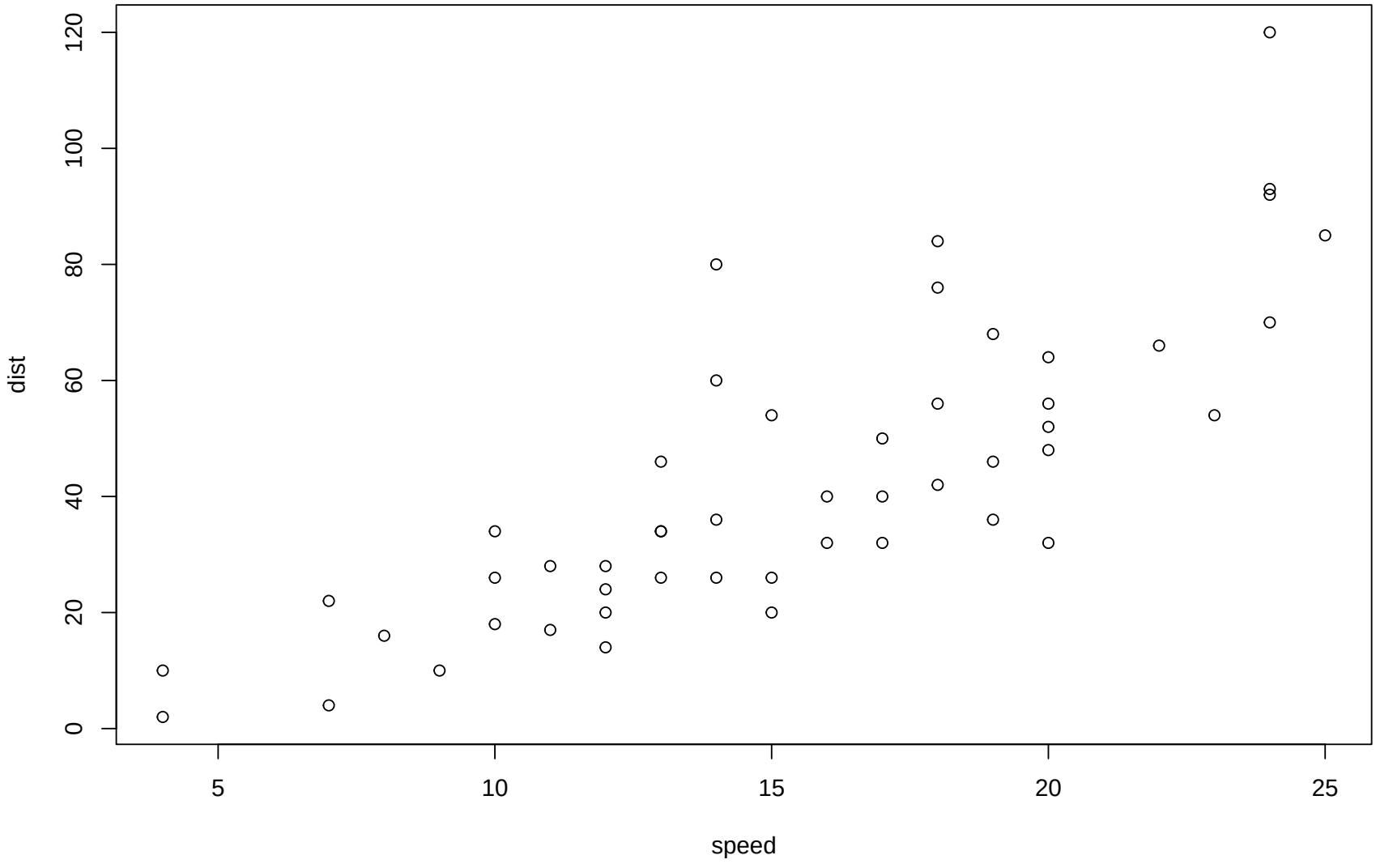
```r
chisq.test(table(x))
```

```
##
##  Chi-squared test for given probabilities
##
## data:  table(x)
## X-squared = 4.4444, df = 4, p-value = 0.3492
```

```
data()
```

```
Data sets in package 'datasets':

AirPassengers                           Monthly Airline Passenger Numbers 1949
BJsales                                 Sales Data with Leading Indicator
BJsales.lead (BJsales)                  Sales Data with Leading Indicator
BOD                                     Biochemical Oxygen Demand
CO2                                     Carbon Dioxide Uptake in Grass Plants
ChickWeight                             Weight versus age of chicks on differe
DNase                                   Elisa assay of DNase
EuStockMarkets                          Daily Closing Prices of Major European
Formaldehyde                            Determination of Formaldehyde
HairEyeColor                            Hair and Eye Color of Statistics Stude
Harman23.cor                            Harman Example 2.3
Harman74.cor                            Harman Example 7.4
Indometh                                Pharmacokinetics of Indomethacin
InsectSprays                            Effectiveness of Insect Sprays
JohnsonJohnson                          Quarterly Earnings per Johnson & Johns
LakeHuron                               Level of Lake Huron 1875-1972
LifeCycleSavings                        Intercountry Life-Cycle Savings Data
Loblolly                                Growth of Loblolly pine trees
Nile                                    Flow of the River Nile
Orange                                  Growth of Orange Trees
OrchardSprays                           Potency of Orchard Sprays
...
```

```r
data(cars)
?cars
plot(cars)
```



```r
cor(cars)
```

```
##           speed      dist
## speed 1.0000000 0.8068949
## dist  0.8068949 1.0000000
```

```r
attach(cars)
cor.test(speed, dist)
```

```
##
##  Pearson's product-moment correlation
##
## data:  speed and dist
## t = 9.464, df = 48, p-value = 1.49e-12
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.6816422 0.8862036
## sample estimates:
##       cor
## 0.8068949
```

```r
l <- lm(dist~speed, data=cars)
summary(l)
```

```
##
## Call:
## lm(formula = dist ~ speed, data = cars)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -29.069  -9.525  -2.272   9.215  43.201
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -17.5791     6.7584  -2.601   0.0123 *
## speed         3.9324     0.4155   9.464 1.49e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 15.38 on 48 degrees of freedom
## Multiple R-squared:  0.6511, Adjusted R-squared:  0.6438
## F-statistic: 89.57 on 1 and 48 DF,  p-value: 1.49e-12
```

Some R features

Assignment

Variables are assigned to a value in an assignment statement.

```
    a <- 1
    a # This will print the content of the variable
## [1] 1

    a = 1 # This also works but is not good R style
```

Data types

Variables may contain numeric values, characters, or "logical" values (also imaginary numbers are possible).

```
 b <- "2"
 b
## [1] "2"

 c <- TRUE
 c
## [1] TRUE

 d <- FALSE
```

Data Structures

Vectors are one-dimensional ordered sets composed of a single data type.

```
x <- c(1,2,3,4,5)
x
## [1] 1 2 3 4 5

x <- array(data = 1:5, dim=5)
x
## [1] 1 2 3 4 5

x <- array(data = 1:3, dim=5)
x
## [1] 1 2 3 1 2

v <- c(TRUE, FALSE, TRUE, TRUE)
v
## [1]  TRUE FALSE  TRUE  TRUE

v[c(1,2)]
## [1]  TRUE FALSE
```

Some functions that generate vectors:

```
x <- 1:5
x
## [1] 1 2 3 4 5

i <- rep(1,10)
i
##  [1] 1 1 1 1 1 1 1 1 1 1

k <- rep(1:3, 2)
k
## [1] 1 2 3 1 2 3

h <- rep(1:3, each = 2)
h
## [1] 1 1 2 2 3 3

ss <- seq(3,9,2)
ss
## [1] 3 5 7 9
```

Factor: A special type of vector, where the values represent categories. **In analyses a clear distinction must be made between a numeric vector and a factor with numeric category labels**.

```
pain <- c(0,3,2,2,1)
fpain <- factor(pain, levels=0:3)
fpain
## [1] 0 3 2 2 1
## Levels: 0 1 2 3
levels(fpain) <- c("none", "mild", "medium", "severe")
fpain
## [1] none   severe medium medium mild
## Levels: none mild medium severe
sum(fpain) # Note that arithmetic operations are no applicable
## Error in Summary.factor(structure(c(1L, 4L, 3L, 3L, 2L), levels = c("none", :
```

Matrices are two-dimensional ordered sets composed of a single data type.

```
x <- array(1:20, dim=c(4,5))
x
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    5    9   13   17
## [2,]    2    6   10   14   18
## [3,]    3    7   11   15   19
## [4,]    4    8   12   16   20

# the same content:
x <- matrix(nr = 4, nc = 5, data = 1:20)

x <- matrix(nr = 4, nc = 5, data = 1:20, byrow=TRUE)
x
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    2    3    4    5
## [2,]    6    7    8    9   10
## [3,]   11   12   13   14   15
## [4,]   16   17   18   19   20
```

```r
# 3-d array
z <- array(1:20, dim = c(4,5,2))

#indexing an array:
i <- array(c(1:3,3:1), dim=c(3,2))
i
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    2
## [3,]    3    1

x[i]
## [1]  3  7 11

x[i] <- 0
x
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    2    0    4    5
## [2,]    6    0    8    9   10
## [3,]    0   12   13   14   15
## [4,]   16   17   18   19   20
```

```r
m1 <- matrix(1, nr = 2, nc = 2)
m1
##      [,1] [,2]
## [1,]    1    1
## [2,]    1    1

m2 <- matrix(2, nr = 2, nc = 2)
m2
##      [,1] [,2]
## [1,]    2    2
## [2,]    2    2

rbind(m1, m2)
##      [,1] [,2]
## [1,]    1    1
## [2,]    1    1
## [3,]    2    2
## [4,]    2    2
```

```r
cbind(m1,m2)
##      [,1] [,2] [,3] [,4]
## [1,]    1    1    2    2
## [2,]    1    1    2    2

cbind(1,m2)
##      [,1] [,2] [,3]
## [1,]    1    2    2
## [2,]    1    2    2
```

Data frames are multivariate data sets, typically composed of data vectors of different data types. This is how the data matrix is usually presented in R.

```
weight <- c(80, 52, 79, 95, 130, 101)
height <- c(1.75, 1.6, 1.72, 1.80, 1.79, 1.85)
nn <- c("John", "George", "Sam", "Tim", "Ronald", "Nick")
wd <- data.frame(pe = nn, we = weight, he = height,
                 bmi = weight / height^2)
wd
##       pe  we   he      bmi
## 1   John  80 1.75 26.12245
## 2 George  52 1.60 20.31250
## 3    Sam  79 1.72 26.70362
## 4    Tim  95 1.80 29.32099
## 5 Ronald 130 1.79 40.57302
## 6   Nick 101 1.85 29.51059

names(wd)
## [1] "pe"  "we"  "he"  "bmi"

wd$we
## [1]  80  52  79  95 130 101
```

A list is a collection of possibly different kind of objects (types and lengths).

```
# pain, fpain and wd created earlier
l <- list(pain = pain, fpain = fpain, mta = wd)
l # prints the whole list
## $pain
## [1] 0 3 2 2 1
##
## $fpain
## [1] none   severe medium medium mild
## Levels: none mild medium severe
##
## $mta
##       pe  we   he      bmi
## 1   John  80 1.75 26.12245
## 2 George  52 1.60 20.31250
## 3    Sam  79 1.72 26.70362
## 4    Tim  95 1.80 29.32099
## 5 Ronald 130 1.79 40.57302
## 6   Nick 101 1.85 29.51059

l$pain # prints the component 'pain'
## [1] 0 3 2 2 1
```

Note: Often R output is given in the form of a list.

Functions

A function call takes the form `y <- fnk(x1, x2, ..., xk)`, where `fnk` is the name of the function, `x1, x2, ..., xk` are possible arguments, and `y` is the output. Some examples:

```
bmicalc <- function(weight, height){
  bmi <- weight / height^2
  bmi # note that the last line determines the value that is returned
}
bmicalc(80, 1.90)
## [1] 22.16066

# This function can also take vector arguments
y <- bmicalc(c(80, 75, 95), c(1.90, 1.8, 1.75))
y
## [1] 22.16066 23.14815 31.02041
```

```r
# means and standard deviations from data matrix
mean.sd <- function(X){
  xa <- sapply(X, mean)
  sa <- sapply(X, sd)
  list(means = xa, stddev = sa)
}
data(cars)
mean.sd(cars)
## $means
## speed  dist
## 15.40 42.98
##
## $stddev
##     speed      dist
##  5.287644 25.769377
```

```r
# Example of recursion
fact  <-  function(x) {
  if (x <= 1) 1 else x * fact(x - 1)
}

fact(10)
## [1] 3628800

gamma(11)
## [1] 3628800
```

For build-in functions se
e.g. https://www.javatpoint.com/r-built-in-functions

```r
# Example
log(1:5)
```

```
## [1] 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379
```

Workspace

All R object that are created during the session are stored in a workspace. The workspace can be saved (with some name) and loaded for further use. The function `ls()` lists and `rm()` removes the objects.

```
ls()
```

```
##  [1] "a"        "b"        "bmicalc" "c"        "cars"     "
##  [8] "fpain"    "h"        "height"  "i"        "k"        "
## [15] "m2"       "mean.sd"  "nn"      "pain"     "ss"       "
## [22] "weight"   "x"        "y"       "z"
```

```
rm(a, b)
rm(list = ls())
```

Packages

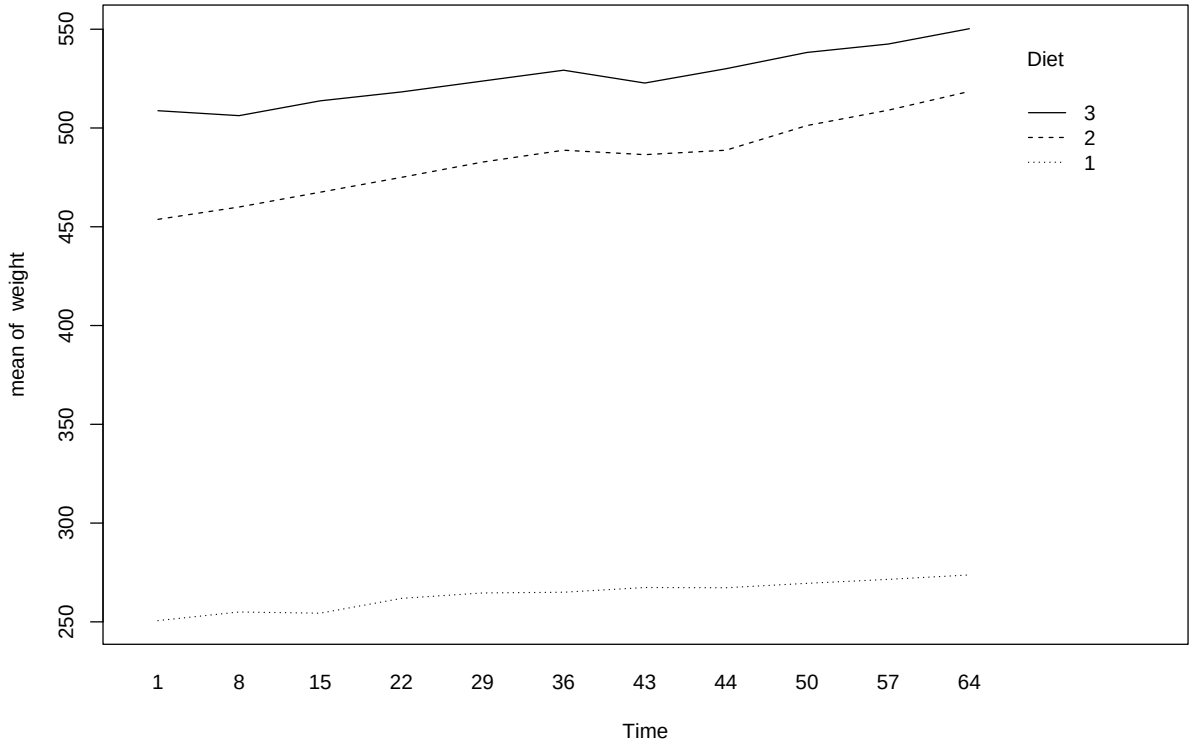
R contains a huge set of program packages for different purposes. Some packages are available in a basic installation, but some need to be downloaded from CRAN.

```
library() # lists the installed R packages
library(MASS)
```

The use of R programs

R programs (functions) have a fairly standard format. Formally they are functions that contain arguments, and the result is a list (or a graphical plot). Consider the following example:

```
library(nlme) # If not available should be downloaded
attach(BodyWeight)
interaction.plot(Time, Diet, weight)
```



```
?BodyWeight
```

This data frame contains the following columns:

weight
a numeric vector giving the body weight of the rat (grams).

Time
a numeric vector giving the time at which the measurement is made (days).

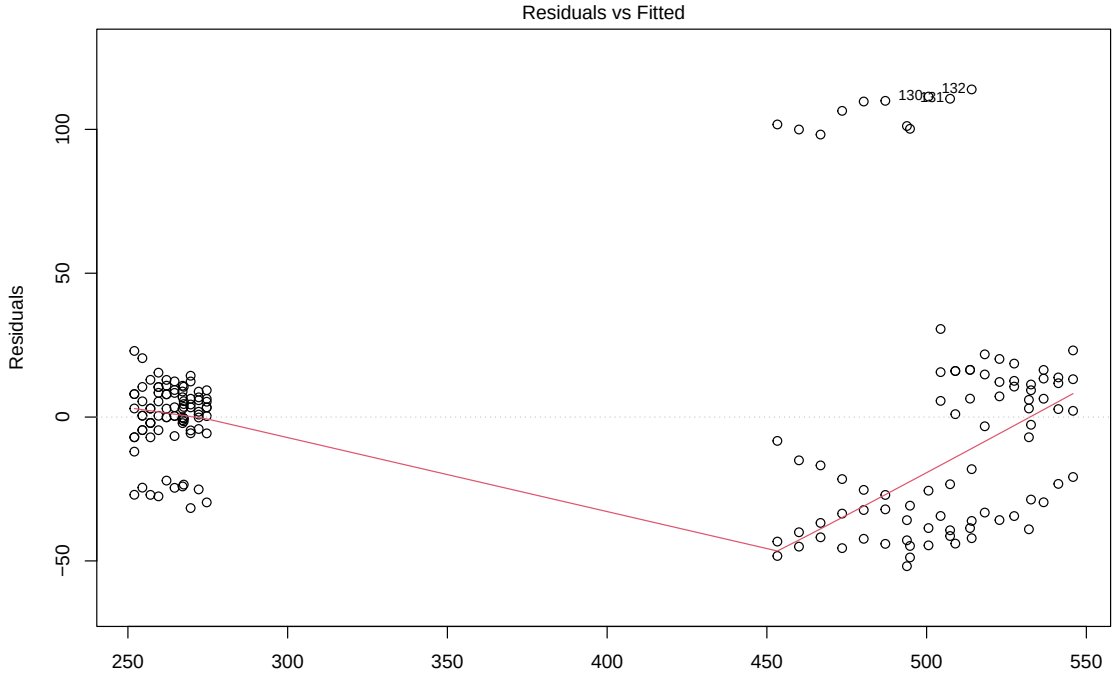
Rat
an ordered factor with levels 2 < 3 < 4 < 1 < 8 < 5 < 6 < 7 < 11 < 9 < 10 < 12 < 13 < 15 < 14 < 16 identifying the rat whose weight is measured.

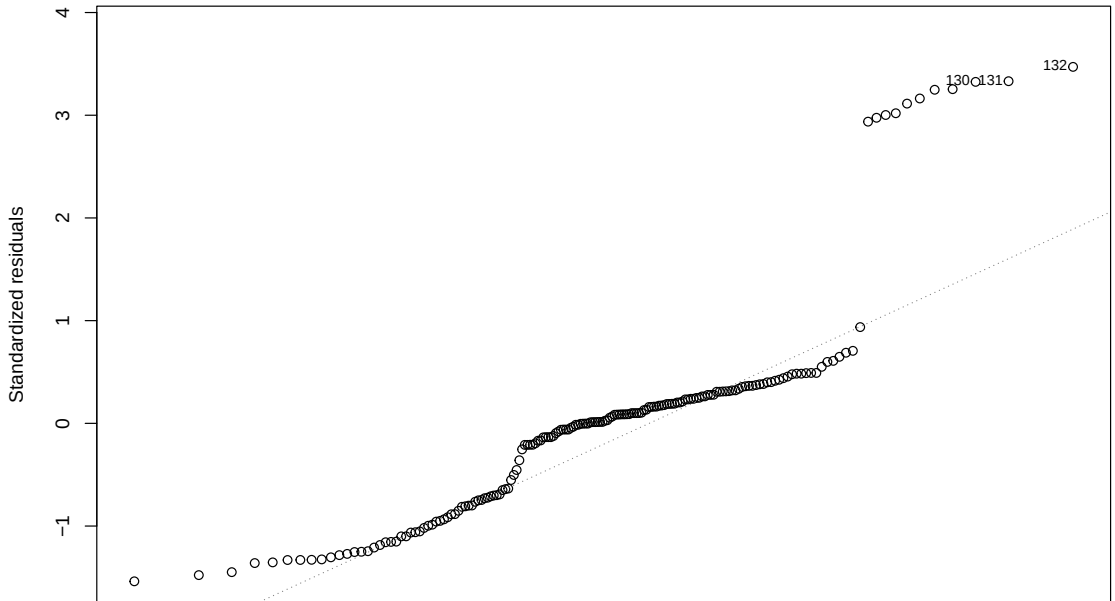
Diet
a factor with levels 1 to 3 indicating the diet that the rat receives.

```r
# Our aim is to model the weight development
# with time, diets and their interaction
l <- lm(weight ~ Time + Diet + Diet:Time, data = BodyWeight)
# or l <- lm(weight ~ Time*Diet, data = BodyWeight)

summary(l)
##
## Call:
## lm(formula = weight ~ Time + Diet + Diet:Time, data = BodyWeight)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -51.83 -24.22   0.66  10.44 113.89
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 251.6517     7.2635  34.646   <2e-16 ***
## Time          0.3596     0.1873   1.920   0.0565 .
## Diet2       200.6655    12.5807  15.950   <2e-16 ***
## Diet3       252.0717    12.5807  20.036   <2e-16 ***
## Time:Diet2    0.6058     0.3244   1.867   0.0636 .
## Time:Diet3    0.2983     0.3244   0.920   0.3591
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 34.18 on 170 degrees of freedom
## Multiple R-squared:  0.9298, Adjusted R-squared:  0.9277
## F-statistic: 450.4 on 5 and 170 DF,  p-value: < 2.2e-16
```

```r
anova(l)
## Analysis of Variance Table
##
## Response: weight
##             Df  Sum Sq Mean Sq   F value    Pr(>F)
## Time         1   22847   22847   19.5541  1.74e-05 ***
## Diet         2 2604050 1302025 1114.3820 < 2.2e-16 ***
## Time:Diet    2    4190    2095    1.7933    0.1696
## Residuals  170  198625    1168
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

names(l) # The components of the list
##  [1] "coefficients"  "residuals"     "effects"       "rank"
##  [5] "fitted.values" "assign"        "qr"            "df.residual"
##  [9] "contrasts"     "xlevels"       "call"          "terms"
## [13] "model"
```

```r
plot(l) # The model needs to be improved!
```





Matrix computations

Preliminaries

An $n \times p$ matrix consists of np real numbers arranged in n rows and p columns. The entry in row i and column j of the matrix $\boldsymbol{U}$ is denoted by u_{ij}. Here **matrices are denoted by capital bold letters**, while **vectors** ($n \times 1$ matrices) **are denoted by small bold letters**.

As an example consider a data matrix $\boldsymbol{A}$

$$\boldsymbol{A}_{n\times p} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1p} \\ a_{21} & a_{22} & \dots & a_{2p} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \dots & a_{np} \end{pmatrix},$$

where the entry a_{ij} is the jth measurement taken from the ith individual.

An $n \times 1$ matrix $\boldsymbol{a}$ is called a column vector of order n. Thus $\boldsymbol{A}$ may be written as

$$\boldsymbol{A} = (\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_p)$$

or if $\boldsymbol{A}$ is written row-wise

$$\boldsymbol{A} = \begin{pmatrix} \boldsymbol{a}'_{(1)} \\ \boldsymbol{a}'_{(2)} \\ \vdots \\ \boldsymbol{a}'_{(n)} \end{pmatrix},$$

where $\boldsymbol{a}'_{(i)}$ is the ith row of $\boldsymbol{A}$.

Some special matrices

Diagonal matrix

$$\operatorname{diag}(\boldsymbol{A}) = \begin{pmatrix} a_{11} & 0 & \dots & 0 \\ 0 & a_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & a_{nn} \end{pmatrix}.$$

The identity matrix is defined as $\boldsymbol{I} = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{pmatrix}$

The column vector of ones, $\mathbf{1}_n = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}$, has an important role in matrix computations.

The matrix $\boldsymbol{J}$ is defined as $\boldsymbol{J} = (1/n)\mathbf{1}\mathbf{1}' = (1/n)\begin{pmatrix} 1 & 1 & \dots & 1 \\ 1 & 1 & \dots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \dots & 1 \end{pmatrix}$.

If all entries of a matrix are zero it is called a zero matrix and is denoted by $\boldsymbol{O}$.

R examples:

```
A <- matrix(nr = 2, nc = 2, data = 1:4)
diag(A)
## [1] 1 4
diag(1:2)
##      [,1] [,2]
## [1,]    1    0
## [2,]    0    2
diag(2)
##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1
i <- matrix(nr = 3, data = 1)
i
##      [,1]
## [1,]    1
## [2,]    1
## [3,]    1
J <- matrix(nr = 3, nc = 3, data = 1/3)
J
##           [,1]      [,2]      [,3]
## [1,] 0.3333333 0.3333333 0.3333333
## [2,] 0.3333333 0.3333333 0.3333333
## [3,] 0.3333333 0.3333333 0.3333333
```

Some basic operations

The transpose of $n \times p$ matrix $\boldsymbol{A}$, denoted by $\boldsymbol{A}'$, is the $p \times n$ matrix whose (i, j)-entry is a_{ji} (columns are changed to rows or vice-verse). Therefore, if $\boldsymbol{A} = (\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_p)$, the transpose is

$$\boldsymbol{A}' = \begin{pmatrix} \boldsymbol{a}_1' \\ \boldsymbol{a}_2' \\ \vdots \\ \boldsymbol{a}_p' \end{pmatrix}.$$

It is easily verified that

$$(\boldsymbol{A}')' = \boldsymbol{A}, \quad (\boldsymbol{A} + \boldsymbol{B})' = \boldsymbol{A}' + \boldsymbol{B}' \quad \text{and} \quad (\boldsymbol{A}\boldsymbol{B})' = \boldsymbol{B}'\boldsymbol{A}'.$$

If $\boldsymbol{A}$ is a matrix and c is a real number (scalar multiplication) $c\boldsymbol{A}$ is obtained by multiplying each element of $\boldsymbol{A}$ by c. If $\boldsymbol{A}' = \boldsymbol{A}$, the matrix $\boldsymbol{A}$ is called symmetric.

```r
# define A
A <- matrix(nr=2, nc=2, data=1:4)
A
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4

t(A) # A'
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4

A + A # sum
##      [,1] [,2]
## [1,]    2    6
## [2,]    4    8

3 * A # scalar multiplication
##      [,1] [,2]
## [1,]    3    9
## [2,]    6   12
```

The product of $m \times p$ matrix $\boldsymbol{A}$ and $p \times n$ matrix $\boldsymbol{B}$ is $\boldsymbol{AB} = \boldsymbol{C}$, where the elements of $\boldsymbol{C}$ are defined as

$$c_{ij} = \sum_{k=1}^{p} a_{ik} b_{kj}.$$

Schematically it is

$$\begin{bmatrix} i\text{th row} \\ \longrightarrow \end{bmatrix}_{m \times p} \begin{bmatrix} j\text{th column} \\ \downarrow \end{bmatrix}_{p \times n} = \begin{bmatrix} ij\text{th} \\ \text{elem.} \end{bmatrix}_{m \times n}$$

Example. If

$$\boldsymbol{A} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 2 & 2 \\ 1 & 3 & 1 \end{pmatrix}, \quad \text{what is} \quad (1/3)\mathbf{1}'\boldsymbol{A}\,?$$

Simple calculation shows that the column means are observed

$$(1/3)(1,1,1) \begin{pmatrix} 1 & 1 & 0 \\ 1 & 2 & 2 \\ 1 & 3 & 1 \end{pmatrix} = (1/3)(3,6,3) = (1,2,1).$$

It can be shown that if

$$\boldsymbol{A} = (\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_n) \quad \text{and} \quad \boldsymbol{B} = \begin{pmatrix} \boldsymbol{b}'_{(1)} \\ \boldsymbol{b}'_{(2)} \\ \vdots \\ \boldsymbol{b}'_{(n)} \end{pmatrix}$$

then

$$\boldsymbol{AB} = \boldsymbol{a}_1 \boldsymbol{b}'_{(1)} + \cdots + \boldsymbol{a}_n \boldsymbol{b}'_{(n)}.$$

Some properties of transpose and sum:

1. $(\boldsymbol{AB})' = \boldsymbol{B}'\boldsymbol{A}'$
2. $(\boldsymbol{A} + \boldsymbol{B})' = \boldsymbol{A}' + \boldsymbol{B}'$
3. $\boldsymbol{A}(\boldsymbol{B} + \boldsymbol{C}) = \boldsymbol{AB} + \boldsymbol{AC}$
4. $(\boldsymbol{A} + \boldsymbol{B})\boldsymbol{C} = \boldsymbol{AC} + \boldsymbol{BC}$.

It is possible e.g. to define a set of linear equations by using matrices. For example

$$\begin{aligned} 2x_1 - 7x_2 + 12x_3 &= 4 \\ 8x_1 - 9x_2 &= 7 \\ 2x_1 - x_2 - 16x_3 &= -1 \end{aligned}$$

can be written as

$$\begin{pmatrix} 2 & -7 & 12 \\ 8 & -9 & 0 \\ 2 & -1 & -16 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 4 \\ 7 \\ -1 \end{pmatrix},$$

$$\boldsymbol{Ax} = \boldsymbol{b}.$$

In R, we can solve the equation as follows:

```
A <- matrix(c(2, 8, 2, -7, -9, -1, 12, 0, -16), nr = 3)
b <- c(4, 7, -1)
x <- solve(A,b) ###### Solution of linear equation ######
x
```

```
## [1]  0.81967213 -0.04918033  0.16803279
```

```
######### This is matrix product in R #########
A %*% x
##      [,1]
## [1,]    4
## [2,]    7
## [3,]   -1

# Some further examples about matrix product
A <- matrix(nr=3,nc=2, 1:6)
A
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
## [3,]    3    6

B <- matrix(nr=3, nc=3, 9:1)
B
##      [,1] [,2] [,3]
## [1,]    9    6    3
## [2,]    8    5    2
## [3,]    7    4    1
```

```
################### A'B ######################

C <- crossprod(A,B)
C
##      [,1] [,2] [,3]
## [1,]   46   28   10
## [2,]  118   73   28

################### A'A ######################

D <- crossprod(A)
D
##      [,1] [,2]
## [1,]   14   32
## [2,]   32   77
```

An important operation for square matrices is the trace-operator. The trace is defined as the sum of diagonal elements

$$\mathrm{tr}(\boldsymbol{A}) = \sum_{i=1}^{n} a_{ii}.$$

Some properties:

1. $\mathrm{tr}(\alpha\boldsymbol{A} + \beta\boldsymbol{B}) = \alpha\,\mathrm{tr}(\boldsymbol{A}) + \beta\,\mathrm{tr}(\boldsymbol{B})$
2. $\mathrm{tr}(\boldsymbol{AB}) = \mathrm{tr}(\boldsymbol{BA})$
3. $\mathrm{tr}(\boldsymbol{A}') = \mathrm{tr}(\boldsymbol{A})$
4. $\mathrm{tr}(\boldsymbol{A}'\boldsymbol{A}) = \mathrm{tr}(\boldsymbol{A}\boldsymbol{A}') = \sum_{i,j} a_{ij}^2$.

Note that the trace in 4) equals zero if and only if $\boldsymbol{A} = \boldsymbol{0}$ (zero matrix).

Example

$$\begin{aligned}\operatorname{tr}[\boldsymbol{I}_n - (1/n)\mathbf{1}_n\mathbf{1}_n'] &= \operatorname{tr}(\boldsymbol{I}_n) - (1/n)\operatorname{tr}(\mathbf{1}_n\mathbf{1}_n') \\ &= n - (1/n)\operatorname{tr}(\mathbf{1}_n'\mathbf{1}_n) = n - (1/n)n = n - 1.\end{aligned}$$

```
################## Trace ####################
sum(diag(B))
## [1] 15

# simple function to calculate trace
tr <- function(X){
  tr <- sum(diag(X))
  tr
}

tr(B) # function call
## [1] 15
```

Determinant

Here determinant is defined by example:

1. For a scalar $\det(a) = a$.
2. $\det \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} = a_{11}a_{22} - a_{21}a_{12}$.
3. For $n > 2$ we may utilize cofactors

$$c_{ij} = (-1)^{i+j} \det(\boldsymbol{M}_{ij}),$$

where $\boldsymbol{M}_{ij}$ obtained from $\boldsymbol{A}$ with ith row and jth column deleted. Then

$$\det(\boldsymbol{A}) = \sum_j a_{ij}c_{ij} \quad \text{according to } i\text{th row or}$$

$$\det(\boldsymbol{A}) = \sum_i a_{ij}c_{ij} \quad \text{according to } j\text{th column.}$$

For example

$$\det\begin{pmatrix}1 & 2\\ 3 & 4\end{pmatrix} = 1\times 4 - 2\times 3 = -2,$$

$$\det\begin{pmatrix}2 & 0 & 0\\ 1 & 2 & 1\\ 0 & 1 & 2\end{pmatrix} = 2\times \det\begin{pmatrix}2 & 1\\ 1 & 2\end{pmatrix} = 2\times(2\times 2 - 1\times 1) = 6.$$

Some properties.

1. $\det(\boldsymbol{A}') = \det(\boldsymbol{A})$.
2. $\det(\boldsymbol{A}) = 0$ if and only if $\mathrm{rank}(\boldsymbol{A}) < n$, where $\boldsymbol{A}$ is $n \times n$.
3. $\det(s\boldsymbol{A}) = s^n \det(\boldsymbol{A})$, where $\boldsymbol{A}$ is $n \times n$ and s is a scalar.
4. $\det(\boldsymbol{A}'\boldsymbol{A}) \neq 0$ if and only if $\mathrm{rank}(\boldsymbol{A}) = n$, where $\boldsymbol{A}$ is $m \times n$.
5. $\det(\boldsymbol{AB}) = \det(\boldsymbol{A})\det(\boldsymbol{B})$ if $\boldsymbol{A}$ and $\boldsymbol{B}$ are square and of the same order.

Inverse

The matrix $\boldsymbol{A}^{-1}$ is the inverse of $\boldsymbol{A}$ if

$$\boldsymbol{A}^{-1}\boldsymbol{A} = \boldsymbol{A}\boldsymbol{A}^{-1} = \boldsymbol{I}.$$

The inverse of $n \times n$ matrix $\boldsymbol{A}$ exists if and only if $\det(\boldsymbol{A}) \neq 0$ or $\mathrm{rank}(\boldsymbol{A}) = n$ (nonsingular matrix). Determining the inverse is computationally a bit tricky. One possibility is to use the result

$$\boldsymbol{A}^{-1} = \frac{1}{\det(\boldsymbol{A})}\{c_{ij}\}',$$

where elements c_{ij} (cofactors) are defined earlier (see determinants).

Examples:

$$\begin{pmatrix} a & b \\ b & a \end{pmatrix}^{-1} = \frac{1}{a^2 - b^2}\begin{pmatrix} a & -b \\ -b & a \end{pmatrix}' = \frac{1}{a^2 - b^2}\begin{pmatrix} a & -b \\ -b & a \end{pmatrix},$$

$$\begin{pmatrix} a_1 & 0 & \dots & 0 \\ 0 & a_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & a_n \end{pmatrix}^{-1} = \frac{1}{\prod_{i=1}^{n} a_i} \begin{pmatrix} \prod_{i=2}^{n} a_i & 0 & \dots & 0 \\ 0 & \prod_{i=1_{i\neq 2}}^{n} a_i & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \prod_{i=1}^{n-1} a_i \end{pmatrix}$$

$$= \begin{pmatrix} 1/a_1 & 0 & \dots & 0 \\ 0 & 1/a_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1/a_n \end{pmatrix}.$$

The inverse is used in many situations. For example, if we have the linear equation

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x},$$

where $\det(\boldsymbol{A}) \neq 0$, the vector $\boldsymbol{x}$ can be solved as

$$\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{y}.$$

Some properties:

1. The inverse is unique.
2. $\det(\boldsymbol{A}^{-1}) = 1/\det(\boldsymbol{A})$
3. $\boldsymbol{A}'^{-1} = [\boldsymbol{A}^{-1}]'$
4. $(\boldsymbol{A}^{-1})^{-1} = \boldsymbol{A}$
5. $(\boldsymbol{AB})^{-1} = \boldsymbol{B}^{-1}\boldsymbol{A}^{-1}$

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

############# Determinant ##############
det(B)
## [1] 0
#There is no inverse matrix!

M <- matrix(nr=3, data=sample(1:10, 9))
M
##      [,1] [,2] [,3]
## [1,]   10    6    3
## [2,]    8    2    9
## [3,]    5    4    7

############# Matrix inverse #############
MI <- solve(M)
MI
##       [,1]        [,2]       [,3]
## [1,]  0.10  0.13636364 -0.2181818
## [2,]  0.05 -0.25000000  0.3000000
## [3,] -0.10  0.04545455  0.1272727
```

```
MI %*% M
##                  [,1]          [,2]          [,3]
## [1,]  1.000000e+00 1.110223e-16 -3.053113e-16
## [2,]  1.110223e-16 1.000000e+00 -1.110223e-16
## [3,] -2.775558e-17 0.000000e+00  1.000000e+00

M %*% MI
##                  [,1]          [,2]          [,3]
## [1,]  1.000000e+00 -6.938894e-18 -1.942890e-16
## [2,] -2.775558e-17  1.000000e+00  8.326673e-17
## [3,] -8.326673e-17  2.081668e-17  1.000000e+00
```

Random vectors

Random variables $y_1, \ldots, y_n$ can be assigned to a random vector $\boldsymbol{y} = (y_1, \ldots, y_n)'$. We denote the expected value of a random vector as

$$\boldsymbol{\mu} = \mathrm{E}(\boldsymbol{y}) = \begin{pmatrix} \mathrm{E}(y_1) \\ \mathrm{E}(y_2) \\ \vdots \\ \mathrm{E}(y_n) \end{pmatrix} = \begin{pmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_n \end{pmatrix}$$

The covariance σ_{ij} between random variables is defined as

$$\sigma_{ij} = \mathrm{Cov}(y_i, y_j) = \mathrm{E}[(y_i - \mu_i)(y_j - \mu_j)].$$

The covariance matrix of a random vector is defined as

$$\mathrm{Var}(\boldsymbol{y}) = \mathrm{E}[(\boldsymbol{y} - \boldsymbol{\mu})(\boldsymbol{y} - \boldsymbol{\mu})'] = \begin{pmatrix} \sigma_{11} & \sigma_{12} & \cdots & \sigma_{1n} \\ \sigma_{21} & \sigma_{22} & \cdots & \sigma_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{n1} & \sigma_{n2} & \cdots & \sigma_{nn} \end{pmatrix} = \boldsymbol{\Sigma},$$

where $\sigma_{ii} = \sigma_i^2 = \mathrm{Var}(y_i)$, $i = 1, \ldots, n$.

Sample Covariance and Correlation. Define a data matrix as

$$\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k),$$

where $\boldsymbol{x}_j$ contains values of the jth variable. The mean of the jth variable is obtained as

$$\bar{x}_j = (x_{1j} + \cdots + x_{nj})/n = \frac{1}{n}(1, \ldots, 1)\begin{pmatrix} x_{ij} \\ \vdots \\ x_{nj} \end{pmatrix} = \frac{1}{n}\mathbf{1}'\boldsymbol{x}_j$$

and the mean over all variables is

$$(\bar{x}_1, \ldots, \bar{x}_k) = \frac{1}{n}\mathbf{1}'(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k) = \frac{1}{n}\mathbf{1}'\boldsymbol{X}.$$

Denote further $\boldsymbol{J} = \frac{1}{n}\mathbf{1}\mathbf{1}'$. It is easily seen that $\boldsymbol{J}$ is symmetric ($\boldsymbol{J} = \boldsymbol{J}'$) and idempotent, because

$$\boldsymbol{J}^2 = \boldsymbol{J} \times \boldsymbol{J} = \left[\frac{1}{n}\right]^2 \mathbf{1}\mathbf{1}'\mathbf{1}\mathbf{1}' = \frac{1}{n}\mathbf{1}\mathbf{1}' = \boldsymbol{J}$$

also the matrix $\boldsymbol{I} - \boldsymbol{J}$ is idempotent

$$(\boldsymbol{I} - \boldsymbol{J})(\boldsymbol{I} - \boldsymbol{J}) = \boldsymbol{I} - \boldsymbol{J} - \boldsymbol{J} + \boldsymbol{J} = \boldsymbol{I} - \boldsymbol{J}.$$

Define the vector of multiple means

$$\bar{\boldsymbol{x}}_j = \boldsymbol{J}\boldsymbol{x}_j = \mathbf{1}\bar{x}_j.$$

The sample variance is obtained as

$$s_j^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_{ij}-\bar{x}_j)^2 = \frac{1}{n-1}\boldsymbol{x}_j'(\boldsymbol{I}-\boldsymbol{J})(\boldsymbol{I}-\boldsymbol{J})\boldsymbol{x}_j = \frac{1}{n-1}\boldsymbol{x}_j'(\boldsymbol{I}-\boldsymbol{J})\boldsymbol{x}_j$$

and the sample covariance

$$s_{ij}^2 = \frac{1}{n-1}\boldsymbol{x}_i'(\boldsymbol{I}-\boldsymbol{J})\boldsymbol{x}_j.$$

The whole sample covariance matrix is then

$$\boldsymbol{S} = \frac{1}{n-1}\boldsymbol{X}'(\boldsymbol{I}-\boldsymbol{J})\boldsymbol{X}.$$

If we define the diagonal matrix of standard deviations as $\boldsymbol{D} = \mathrm{diag}(s_1, s_2, \ldots, s_k)$, the correlation matrix can be calculated as

$$\boldsymbol{R} = \boldsymbol{D}^{-1}\boldsymbol{S}\boldsymbol{D}^{-1}.$$

As a further example we consider the ordinary regression model

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i, \quad i = 1, \ldots, n,$$

where the errors are assumed to be independently normally distributed as $\varepsilon_i \sim N(0, \sigma^2)$. We begin by writing the equations as

$$\begin{aligned} y_1 &= \beta_0 + \beta_1 x_1 + \varepsilon_1, \\ y_2 &= \beta_0 + \beta_1 x_2 + \varepsilon_2, \\ &\vdots \\ y_n &= \beta_0 + \beta_1 x_n + \varepsilon_n. \end{aligned}$$

In the matrix form

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix} + \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{pmatrix},$$

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}.$$

Now the assumptions for errors can also be written in the matrix form as $\boldsymbol{\varepsilon} \sim N(\boldsymbol{0}, \sigma^2 \boldsymbol{I})$.

A popular method to estimate the parameters of the model is the method of least squares. If $\boldsymbol{X}'\boldsymbol{X}$ is non-singular (inverse exists), we have the simple well-known OLSE (Ordinary Least Squares Estimator) formula

$$\hat{\boldsymbol{\beta}} = (\boldsymbol{X}'\boldsymbol{X})^{-1}\boldsymbol{X}'\boldsymbol{y}.$$

```
y <- c(1, 1, 2, 2, 3, 3, 3, 2)

# second degree polynomial fit
X <- outer(1:8, 0:2, "^") # outer 'product' of two vectors
X
##      [,1] [,2] [,3]
## [1,]    1    1    1
## [2,]    1    2    4
## [3,]    1    3    9
## [4,]    1    4   16
## [5,]    1    5   25
## [6,]    1    6   36
## [7,]    1    7   49
## [8,]    1    8   64

b <- lsfit(X, y, intercept = FALSE)
b
## $coefficients
##          X1          X2          X3
## -0.33928571  1.05357143 -0.08928571
##
```

Computer intensive methods

Simulation

In Wikipedia simulation is defined as follows:

- Simulation is the imitation of the operation of a real-world process or system over time.
- The act of simulating something first requires that a model be developed; this model represents the key characteristics or behaviors/functions of the selected physical or abstract system or process.
- The model represents the system itself, whereas the simulation represents the operation of the system over time.

Examples of simulation include:

- Effect of pollution
- Investigation of animal population
- Flight simulator
- Investigation of the atmosphere (e.g. global warming).

In a broad sense simulation is a numerical technique for conducting experiments on the computer, while Monte Carlo simulation involves sampling from a probability distribution.

In statistics the interest is often on the model (or method) itself whose properties are investigated using observations generated from a probability distribution.

For example: Is the new estimator unbiased, what is the variance and the distribution of the estimates? Is the developed new test more powerful than the old one etc, etc. In many cases exact analytical solution may not be available.

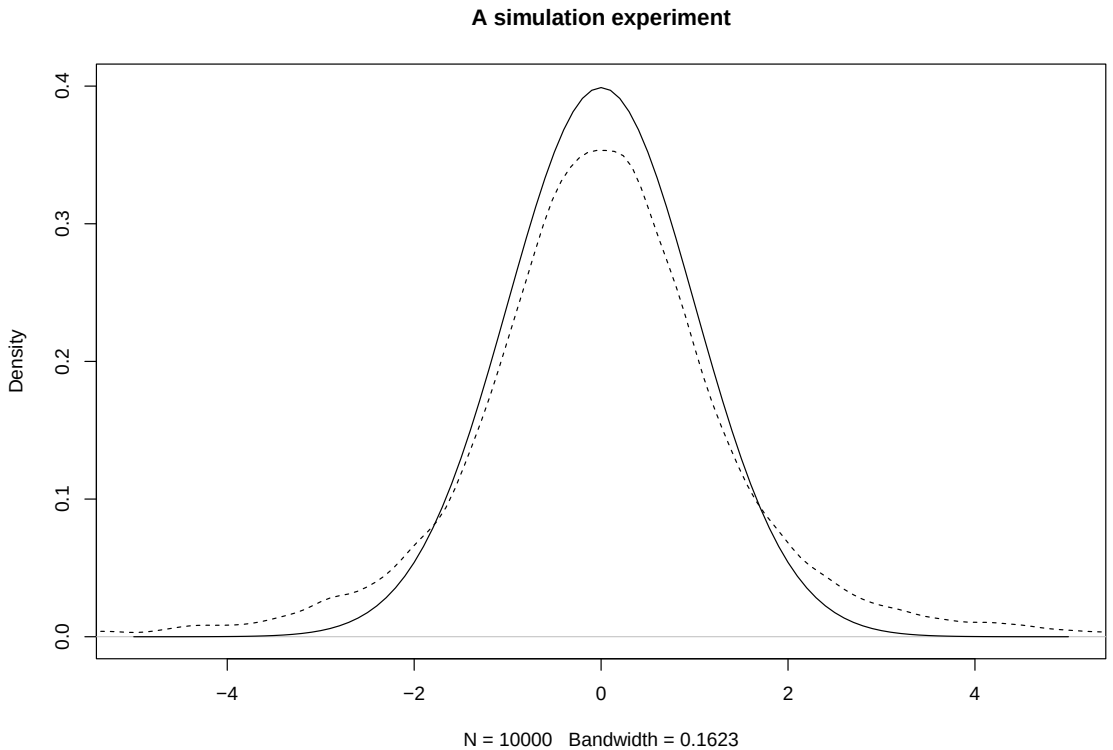
An old example (Gosset, 1908) investigates the distribution of the statistics

$$t = \sqrt{n}(\bar{x} - \mu)/s,$$

where $\bar{x} = \frac{1}{n}\sum x_i$ and $s^2 = \frac{1}{n-1}\sum(x_i - \bar{x})^2$. What is the distribution if $n = 4$ for example?

```
# A simulation experiment with R
mean.sd <- function(X){
  xa <- sapply(X, mean)
  sa <- sapply(X, sd)
  list(means = xa, sddev = sa)
}
data_sim <- matrix(nr = 4, nc = 10000, data = rnorm(40000))
data_sim <- as.data.frame(data_sim)
l <- mean.sd(data_sim) # 10000 means
t_sim <- (2 * l$mean) / l$sddev
```

```r
plot(density(t_sim), ylim = c(0, 0.4), xlim = c(-5, 5),
     lty = 2, main = "A simulation experiment")
curve(dnorm(x), add = TRUE) # Stardard normal density curve
```



Random numbers

Fully random numbers can be produced, for example, by throwing a coin or dice, or using so-called hardware random number generators. In statistical applications, it usually suffices to use so-called pseudo-random numbers, which are generated by an algebraic expression.

Pseudo-random numbers have many advantages. They are very easy to generate and the same sequence can easily be obtained if needed. However, they must satisfy the following requirements:

- apparent independence,
- the sequence (every generator has a sequence) is long enough,
- uniform distribution.

In practice, every generator has to pass a number of tests before it can be used. The most frequently adopted formula is the recursion equation (**Linear congruential generator**)

$$x_{n+1} = (a \times x_n + b) \mod m,$$

where a, b and m are suitable chosen integer constants, and the seed is an integer x_0. Starting from x_0 the formula gives a sequence of integers, each of which lies in the 0 to $m - 1$ range.

Approximations to $U(0, 1)$ variables can be obtained from

$$u_i = x_i/m.$$

Inverse transformation method

When random variates from a continuous distribution are needed, one of the most important methods is the inverse transformation method. It is based on the use of uniformly distributed random variates $U(0,1)$.

The method is fairly simple. Suppose that we wish to generate random numbers from a distribution whose cumulative distribution function is $F(x)$. We first generate U from the uniform distribution $U(0,1)$ and then compute $X = F^{-1}(U)$. This will have the correct distribution.

Proof: It suffices to show that $P(X \leq x) = F(x)$. Because $F(x)$ is a strictly increasing continuous function,

$$P(X \leq x) = P(F^{-1}(U) \leq x) = P(U \leq F(x)).$$

Further, because U is uniformly distributed between 0 and 1,

$$P(U \leq F(x)) = F(x).$$

Combining these results we have that $P(X \leq x) = F(x)$.

Example a. If X has the density

$$f(x) = \begin{cases} \frac{3x^2}{2}, & -1 \leq x \leq 1 \\ 0, & \textit{otherwise} \end{cases}$$

then

$$F(x) = \int_{-1}^{x} \frac{3x^2}{2} dx = \left| \begin{matrix} x \\ -1 \end{matrix} \right. \frac{1}{2}x^3 = \frac{1}{2}(x^3 + 1), \ -1 \leq x \leq 1.$$

If we denote

$$U = \frac{1}{2}(X^3 + 1),$$

then the inverse transformation takes the form

$$X = \sqrt[3]{2U - 1},$$

where $U \sim U(0, 1)$.

Example b. The density of the exponential distribution is

$$f(x) = \begin{cases} (1/\beta)e^{-x/\beta}, & 0 \le x < \infty \\ 0, & x < 0 \end{cases}$$

and its distribution function is

$$F(x) = 1/\beta \int_0^x e^{-\theta/\beta} d\theta = 1/\beta \left|_0^x e^{-\theta/\beta}(-\beta) = 1 - e^{-x/\beta}.\right.$$

If we denote

$$U = 1 - e^{-X/\beta},$$

then

$$\log(1 - U) = \log(e^{-x/\beta}) = -X/\beta.$$

The actual transformation needed takes the form

$$X = -\beta \times \log(U),$$

since $U - 1$ and U have the same distribution.

Basically the same kind of idea applies also for discrete distributions (table lookup method). As an example consider generation of random variables from the Bernoulli distribution $\mathrm{Bern}(p)$. Then for Bernoulli distribution with parameter p

$$\begin{cases} P(X=0) = 1-p, \\ P(X=1) = p. \end{cases}$$

The method is to set

$$X = \begin{cases} 0, & 0 \leq u \leq 1-p, \\ 1, & 1-p < u \leq 1. \end{cases}$$

Thus we have two steps:

1. generate U from $U(0,1)$,
2. if $U \in [0, 1-p]$ set $X = 0$ otherwise $X = 1$.

Starting from the Bernoulli distribution, it would be easy to continue to the related binomial and Poisson distributions.

Some R examples

```
# NOTE. It is always good to save the seed number:
seed <- .Random.seed

########## Uniform distribution ###############
runif(4)
## [1] 0.6493076 0.5484269 0.4068166 0.5969980

runif(4)
## [1] 0.37485805 0.09990088 0.79615059 0.03423378

.Random.seed <- seed

runif(4)
## [1] 0.6493076 0.5484269 0.4068166 0.5969980

x <- runif(1000)
h <- hist(x, plot=F)
names(h)
## [1] "breaks"   "counts"   "density"  "mids"     "xname"    "equidist"

h$counts
##  [1] 105 105 115  98  90 103 108  85  83 108
```

```
########### Normal distribution ##################
rnorm(4)
## [1] -0.94318652 -0.07902574 -1.54844701  1.19482473

pnorm(2.5)
## [1] 0.9937903

qnorm(0.9937903)
## [1] 2.499998
```

R has many functions related to probability distributions. The form of the functions is as follows:

- r is the generic prefix for random variable generator such as `runif()`, `rnorm()`.
- d is the generic prefix for the probability density function such as `dunif()`, `dnorm()`.
- p is the generic prefix for the cumulative density function such as `punif()`, `pnorm()`.
- q is the generic prefix for the quantile function such as `qunif()`, `qnorm()`.

Some example functions:

Function	Distribution
rbeta	The Beta
rbinom	The Binomial
rcauchy	The Cauchy
rchisq	The (non-central) Chi-Squared
rexp	The Exponential
rf	The F
rgamma	The Gamma
rgeom	The Geometric
rhyper	The Hypergeometric
rlnorm	The Log Normal
rlogis	The Logistic
rmultinom	The Multinomial
rnbinom	The Negative Binomial
rnorm	The Normal
rpois	The Poisson
rt	The Student t
runif	The Uniform
rweibull	The Weibull

Quantiles of Student's t-distribution

```r
M <- matrix(nc=3, nr=20, data=c(0.9, 0.95, 0.99), byrow=TRUE)
t_table <- qt(M, df = 1:20)
colnames(t_table) <- c("0.90", "0.95", "0.99")
print(t_table)
##            0.90     0.95      0.99
##  [1,] 3.077684 6.313752 31.820516
##  [2,] 1.885618 2.919986  6.964557
##  [3,] 1.637744 2.353363  4.540703
##  [4,] 1.533206 2.131847  3.746947
##  [5,] 1.475884 2.015048  3.364930
##  [6,] 1.439756 1.943180  3.142668
##  [7,] 1.414924 1.894579  2.997952
##  [8,] 1.396815 1.859548  2.896459
##  [9,] 1.383029 1.833113  2.821438
## [10,] 1.372184 1.812461  2.763769
## [11,] 1.363430 1.795885  2.718079
## [12,] 1.356217 1.782288  2.680998
## [13,] 1.350171 1.770933  2.650309
## [14,] 1.345030 1.761310  2.624494
## [15,] 1.340606 1.753050  2.602480
## [16,] 1.336757 1.745884  2.583487
## [17,] 1.333379 1.739607  2.566934
## [18,] 1.330391 1.734064  2.552380
## [19,] 1.327728 1.729133  2.539483
## [20,] 1.325341 1.724718  2.527977
```

```
 ####### Sampling with and without replacement #######
sample(10)
##  [1]  3  8  2  5 10  4  1  9  6  7

sample(10, replace=TRUE)
##  [1]  9  8  9  6 10  9  2  7  9  2

sample(c("boy", "run", "fast"))
## [1] "boy"  "run"  "fast"

sample(c("boy", "run", "fast"))
## [1] "boy"  "run"  "fast"

# Lotto
sample(39,7)
## [1] 11 16 27 20  8 19 21

sample(39,7)
## [1] 29 34  6  3 31 14 30

# Bernoulli with p=0.3
sample(c(0, 1), 5, replace=TRUE, prob=c(0.3, 0.7))
## [1] 0 1 0 1 1

# given discrete distribution
sample(c(0, 1, 2), 5, replace=TRUE, prob=c(0.2, 0.3, 0.5))
## [1] 0 1 1 2 1
```

The normal distribution

The normal distribution plays an important role in statistics. Here we present two methods to generate observations from it. The first is based on the central limit theorem.

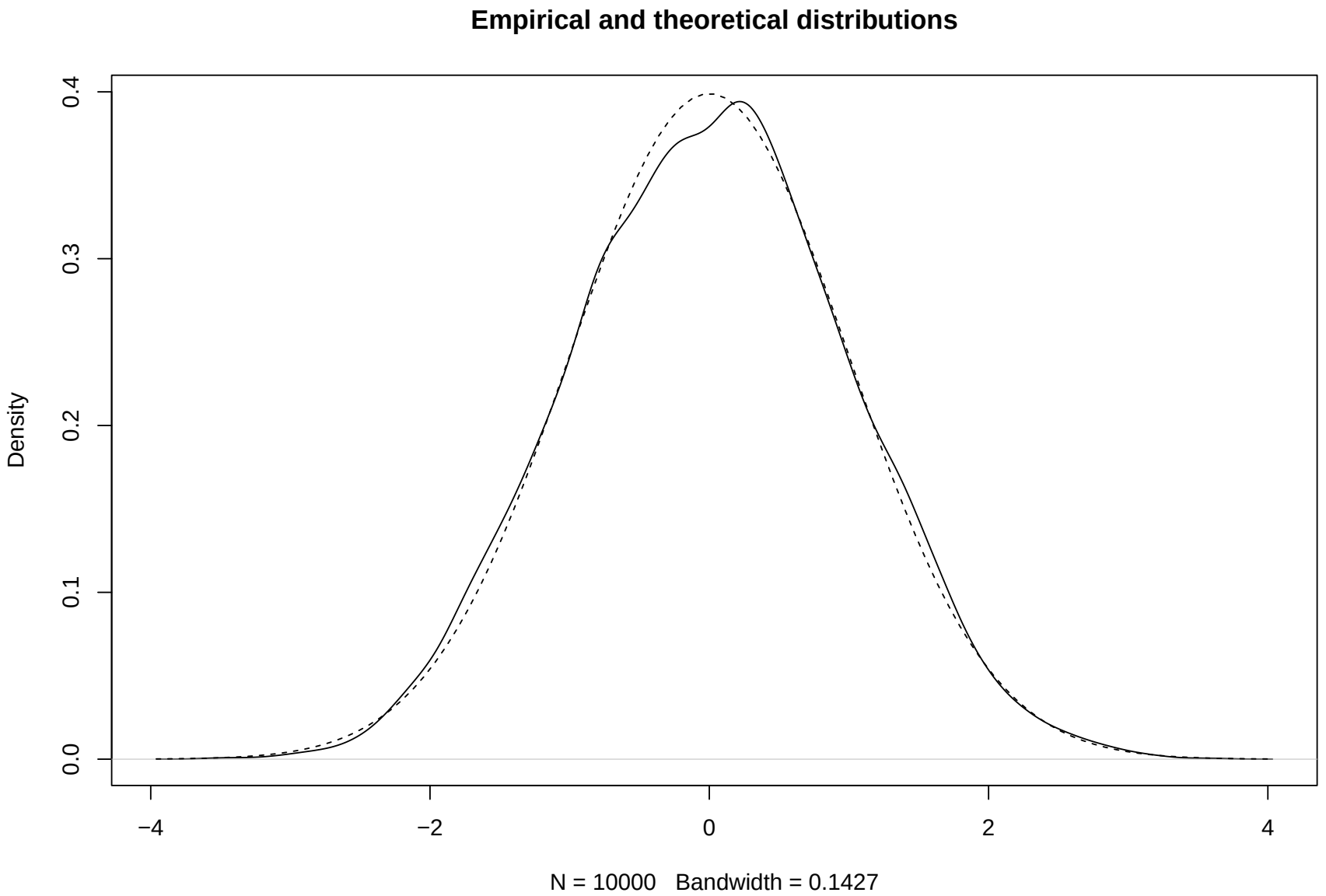
The central limit theorem (CLT) states that, given certain conditions, the arithmetic mean of a sufficiently large number of independent random variables, each with expected value μ and variance σ^2, will be approximately normally distributed, regardless of the underlying distribution.

It is convenient to take $n = 12$ since

$$\Big(\sum_{i=1}^{12} U_i - 6\Big) \approx N(0, 1),$$

because $\mathrm{E}(U_i) = 0.5$ and $\mathrm{Var}(U_i) = 1/12$.

```r
U <- matrix(nc=10000, nr=12, data=runif(12*10000))
z <- apply(U, 2, sum) - 6
plot(density(z), main = "Empirical and theoretical distribu
curve(dnorm(x), add=TRUE, lty=2)
```



Box-Muller From two independent U_1 and U_2 we can generate independently normally distributed ($N(0,1)$) random variables

$$N_1 = (-2\log(U_1))^{0.5}\cos(2\pi U_2),$$
$$N_2 = (-2\log(U_1))^{0.5}\sin(2\pi U_2).$$

***

Example. Confidence intervals for the sample mean.

```
ciformean <- function(n=100,ss=10,m=0,s=1){
# Matrix for upper and lower bounds
limits <- matrix(nr=2,nc=n)
for (i in 1:n)
# a simulation run
{limits[,i] <- t.test(rnorm(ss,mean=m, sd=s),
mean=m)$conf.int[1:2]}
# plotting
ind <- matrix(nr=2, nc=n, data=1:n, byrow=T)
matplot(ind,limits, type="l", ylab="Confidence interval",
main="Confidence intervals for simulated data")
# line for the mean
```

Multivariate normal distribution

Multivariate normal distribution is an extension of the univariate normal distribution to the p-variate situation. This is denoted as

$$\boldsymbol{y} \sim N(\boldsymbol{\mu}, \boldsymbol{\Sigma}),$$

where $\boldsymbol{\mu} = \mathrm{E}(\boldsymbol{y})$ and $\boldsymbol{\Sigma} = \mathrm{Var}(\boldsymbol{y})$ (see random vectors). Random vectors from the multivariate normal distribution can be quite easily generated using the result

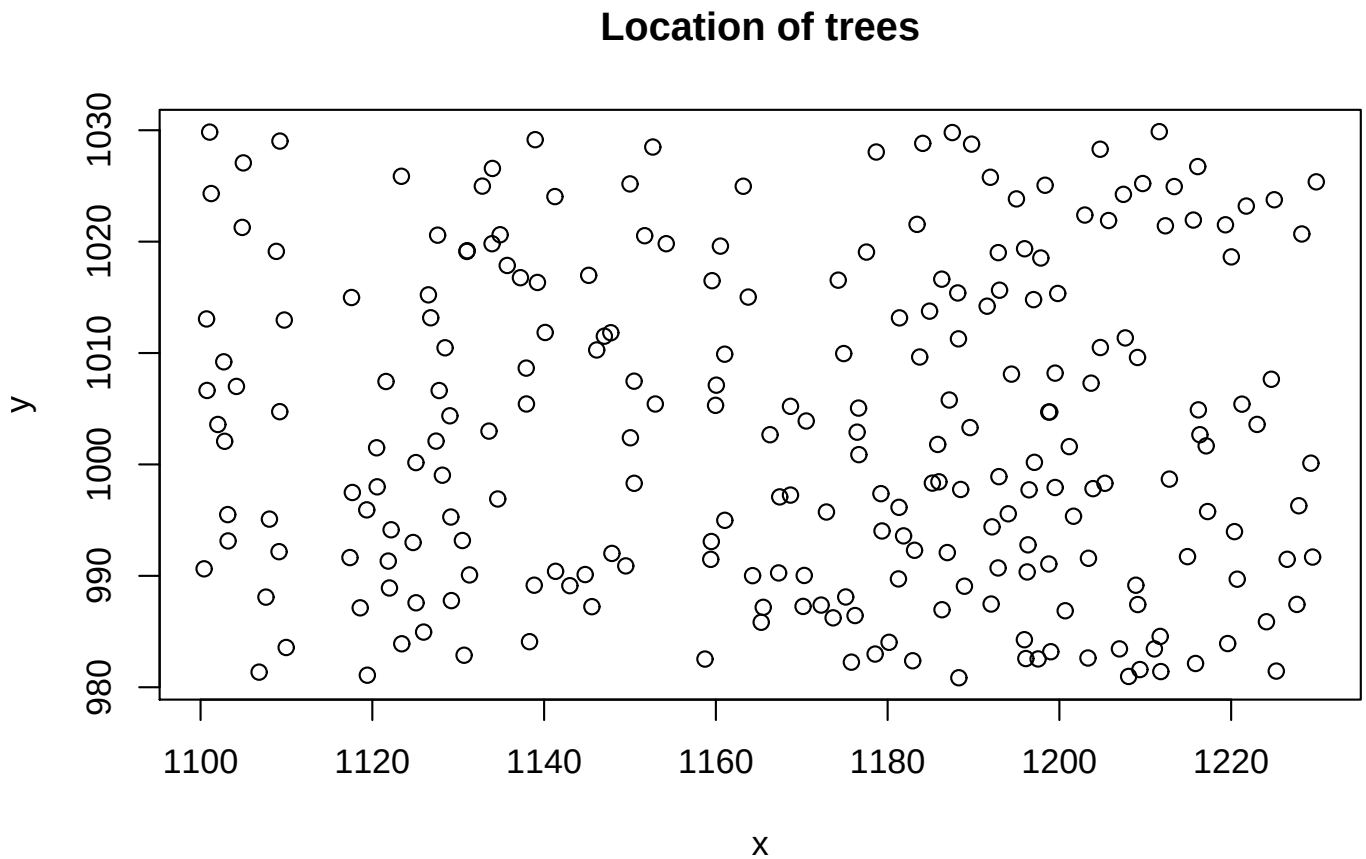
$$\boldsymbol{y} = \boldsymbol{C}\boldsymbol{z} + \boldsymbol{\mu} \sim N(\boldsymbol{\mu}, \boldsymbol{\Sigma}),$$

where independent components z_i of $\boldsymbol{z}$ are normally distributed $N(0, 1)$ and $\boldsymbol{C}$ is a matrix with the property $\boldsymbol{C}\boldsymbol{C}' = \boldsymbol{\Sigma}$. There are many ways to decompose $\boldsymbol{C}\boldsymbol{C}' = \boldsymbol{\Sigma}$. One is to apply the so-called Cholesky factorization, that produces a lower triangular $\boldsymbol{C}$ matrix.

```
twodnorm <- function(n=1000, r=0){
mm <- r^(1!=diag(2)); mm <- t(chol(mm))
ma <- mm%*%matrix(rnorm(2*n), nr=2)
ma <- data.frame(x1=ma[1,], x2=ma[2,])
ma}
```

Monte Carlo test

In the example data the location (x, y) of 237 trees of natural forest in Finland were given. **The aim is to investigate if the trees are randomly distributed over the area**. Here we investigate the distance to nearest neighbor and calculate the average d^*. If this value is "too" small or "large" we may argue that the trees are not uniformly distributed.



Permutation and randomization test

Example 1. Suppose n_1 values are observed in the first group and n_2 values are observed in the second group to be tested ($n = n_1 + n_2$). The interest is to **test if the observed difference of sample means $d = \bar{x}_1 - \bar{x}_2$ is statistically significant**.

Our data is as follows:

```
# sample 1
ambient <- c(254,252,239,240,250,256,267,249,259,269)
# sample 2
heated <- c(233,252,237,246,255,244,248,242,217,257,254)
# difference of sample means
d <- mean(ambient)-mean(heated)
d
[1] 9.409091

whole <- c(ambient, heated)
df <- data.frame(whole, group=rep(1:2, 10:11))
boxplot(whole~group, data=df)
```

Cross-validation

Cross-validation is a method of assessing the predictive ability of a statistical model and choosing between alternative models. The main idea is that models should be judged by their ability to predict the future data.

In cross-validation the data are divided into k subsets, whose sizes are as nearly as possible. One subset is selected as the test set and the remaining union of $k-1$ subsets forms a training set. In practice$$

In theoretical studies we may consider the expected value $\mathrm{E}(y-\hat{y})^2$. From the data we can obtain the value of Q that can be considered as an estimate of this expected value.

As example consider the simple regression model

$$y_i = \alpha + \beta x_i + \varepsilon_1, \quad i = 1, \ldots, 10.$$

Denote $\hat{\alpha}_{(i)}$ and $\hat{\beta}_{(i)}$ as parameter estimates of α and β, where the item i is omitted and the corresponding predicted value of the ith observation is $\hat{y}_i = \hat{\alpha}_{(i)} + \hat{\beta}_{(i)} x_i$.

The Jackknife

Let $\hat{\theta}$ be an estimator based on the variables $Y_1, \ldots, Y_n$. Let $\hat{\theta}_{(i)}$ denote estimator that is based on the data without the ith observation. Define the pseudo-values

$$\tilde{\theta}_i = n \times \hat{\theta} - (n-1) \times \hat{\theta}_{(i)}, \quad i = 1, \ldots, n.$$

The Jackknife estimator of θ is

$$\tilde{\theta}_J = \frac{1}{n} \sum \tilde{\theta}_i = n \times \hat{\theta} - (n-1)\bar{\theta},$$

where $\bar{\theta}$ is the mean of $\hat{\theta}_{(1)}, \hat{\theta}_{(2)}, \ldots, \hat{\theta}_{(n)}$. It is easy to see that $\tilde{\theta}_J$ is unbiased if $\hat{\theta}$ and $\hat{\theta}_{(i)}$, $i = 1, \ldots, n$, are unbiased.

It can also be shown that if the expected value of the estimator is of the form

$$\mathrm{E}(\hat{\theta}) = \theta + c_1/n,$$

the Jackknife method eliminates the bias c_1/n completely.

For example, suppose $\mathrm{Var}(Y) = \sigma^2$ and we take

$$\hat{\theta} = \frac{1}{n} \sum (y_i - \bar{y})^2$$

The Bootstrap

This method has an interesting name which loosely means to pull oneself up by ones bootstrap, that is, to do almost the impossible. We start by defining some needed concepts.

Let $Y_1, \ldots, Y_n$ denote iid (independent and identically distributed) random variables with cumulative distribution function (CDF) F. Let $y_1, \ldots, y_n$ be the observed data with the empirical CDF

$$\hat{F}_n(y) = \frac{1}{n}\#\{y_i \leq y\},$$

that puts the same probability $\frac{1}{n}$ for each $y_1, \ldots, y_n$ and $\#$ means number of observations.

To start consider the following example (number of typhoons in North Pacific Ocean over 88 years):

```
typhoon
 [1] 13  7 14 20 13 12 12 15 20 17 11 14 16 12 17 ...
```

Principles of Bayesian inference

Maximum likelihood method

The Maximum Likelihood (ML) method is based on the likelihood function. Given independent and identically distributed sample $y_1, \ldots, y_n$ the likelihood function is

$$L(\boldsymbol{\theta}) = \prod_{i=1}^{n} f(y_i, \boldsymbol{\theta}),$$

where $f(y_i, \boldsymbol{\theta})$ is the density of y_i and $\boldsymbol{\theta}$ is the parameter vector with the domain D.

Often it is more simple to consider the *log* of $L(\boldsymbol{\theta})$. We then have

$$l(\boldsymbol{\theta}) = \sum_{i=1}^{n} log\ f(y_i, \boldsymbol{\theta}), \boldsymbol{\theta} \in D.$$

Note that given i.i.d. sample $l(\boldsymbol{\theta})$ is a function of parameters $\boldsymbol{\theta}$.

If the maximum of $l(\boldsymbol{\theta})$ is obtained as interior point $\hat{\boldsymbol{\theta}}$ of D, it is called the maximum likelihood estimate (MLE) $\hat{\boldsymbol{\theta}}$ of $\boldsymbol{\theta}$.

Since the derivative of $l(\boldsymbol{\theta})$ at $\hat{\boldsymbol{\theta}}$ must be 0, the estimate is obtained

The Law of Total Probability

If events $A_1, \ldots, A_k$ partition the sample space S into mutually exclusive and exhaustive nonempty events, then the Law of Total Probability states that the total probability of an event B is given by

$$P(B) = \sum_{j=1}^{k} P(B|A_j)P(A_j),$$

where $P(B|A_j)$ is the conditional probability of B given A_j, when $j = 1, \ldots, k$.

Example. Suppose that in a country 80 % of right wing party voters support the current president and the corresponding proportion of independent and left wing voters is 5 % for both.

We assume that 38 % support right wing party, 12 % are independent and 50 % support left wing party. What is the probability that a randomly selected voter support the current

Bayes' Theorem

Bayesian analysis is based on the application of Bayes' Theorem. This is given as follows: If A and B are events and $P(B)0$, then

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)},$$

where the Law of Total Probability is often applied for $P(B)$.

Note that the above formula can also be used to reverse conditional probabilities.

Example.

- Assume that the proportion of the virus infected people in a population is $P(A_1 = inf.) = 0.00138$.
- Suppose further that the probabilty of positive test if a person is infected is $P(B = +|A_1 = inf.) = 0.938$.
- Suppose further that there is a small probability that a test result is positive if person is not infected:

Bayesian analysis

- In the classical frequentist approach, the parameters of a distribution are fixed but unknown constants.

- In the Bayesian approach the unknown parameters of a distribution are viewed as random variables whose values are obtained from some known probability distribution.

- The distribution of unknown parameters summarizes our prior beliefs of the parameters. The actual inferential tool in Bayesian analysis is the conditional density of parameters $\boldsymbol{\theta}$ given the sample of observations $\boldsymbol{y}$ and it is called the posterior density.

The posterior density is defined as

$$p(\boldsymbol{\theta}|\boldsymbol{y}) = \frac{f(\boldsymbol{y} \mid \boldsymbol{\theta}) p_{\boldsymbol{\theta}}(\boldsymbol{\theta})}{\int f(\boldsymbol{y} \mid \boldsymbol{\theta}) p_{\boldsymbol{\theta}}(\boldsymbol{\theta}) d\boldsymbol{\theta}},$$

Simulation from the Posterior Distribution

In many cases the posterior distribution can be intractable and lead to unrecognizible distribution. However, we can still do inference from a posterior distribution through simulation.

We can simulate a posterior distribution by drawing N random samples as $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_N$ from the posterior distribution $p(\boldsymbol{\theta} \mid \boldsymbol{y})$.

From these samples, for any function of the parameters $\boldsymbol{\theta}$ of g, the mean of $E[g(\boldsymbol{\theta}) \mid \boldsymbol{y})]$ *(quadratic loss function)* can be estimated by

$$\frac{\sum_i^N g(\boldsymbol{\theta}_i)}{N}$$

and the posterior distribution for $g(\boldsymbol{\theta})$ can be approximated by a histogram of the values $g(\boldsymbol{\theta}_1), \ldots, g(\boldsymbol{\theta}_N)$.

The four most common methods used in the simulation of the posterior distributions are direct simulation, importance sampling