Notes on statistics and computing with data using R

Data science using R; Aalborg, Summer, 2017

Compilation: Wednesday 18th October, 2017

Søren Højsgaard
Department of Mathematical Sciences
Aalborg University, Denmark
http://people.math.aau.dk/~sorenh/

Contents

Chapter 1

Introduction to R

After starting R we see a prompt (>) where commands are typed followed by hitting the enter button. R will evaluate the commands and print the result.

Anything that exists in R is an OBJECT (sometimes also called a VARIABLE); anything we do in R involves calling functions. A function take zero, one or more objects as input arguments and returns an object as output.

We create two objects (or variables), m and n, holding the values 10 and 1.56. This can be done in different ways:

```
m <- 10
n = 1.56
```

We say that "<–" and "=" are ASSIGNMENT operators. They can be used interchangably and the effect is to create variables m and n and store some values in them. The "<–"–assignment operator (mimicing an arrow) is perhaps the most intuitive: It reads: "Take whatever is on the right hand side and store in the object on the left hand side". Typing the names at the command prompt causes R to print the values:

```
m
## [1] 10
n
## [1] 1.56
```

Objects can be modified. For example

```
m <- m * 10 + 7
m
## [1] 107
n = n + m
n
## [1] 108.6
```

We can have multiple computations/assignments on one line but then they must be separated by semi colon (";"). Anything after a hashmark (#) is regarded as comments in R. For example:

Symbol	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
<	Less than
>	Greater than
<=	Less or equal
>=	Greater or equal
==	Logical equal
** or ^	Exponentiation
%%	Remainder after division (modulo)
%/%	Integer part
exp(x)	Exponential function
log(x)	Natural logarithm
sqrt(x)	Square root
abs(x)	Absolute value
sin(x) and cos(x)	Sine and cosine functions

Table 1.1: Mathematical operators and functions.

```
## multiple instructions on the same line are separated by ";"
sqrt.n <- sqrt( n ); sqrt.n   # Another comment
## [1] 10.42
```

In the code above the `sqrt()` function takes a single argument as input and computes the square root.

Another example of a function in R is `q()` which is used for exititing (or quitting) R. If we simply type q then we see how the function is defined (which is not of interest to us at this stage). To actually invoke the function we must do

```
q()
```

1.1 R as a calculator

R functions nicely as a simple calculator. Table 1.1 shows examples of simple mathematical operations and functions. Below is an example on how to do some of the calculations.

```
-3^2              ## power
## [1] -9
sqrt(15)          ## square root
## [1] 3.873
sin(2)            ## sine function
## [1] 0.9093
log(5)            ## natural log with base e
## [1] 1.609
log(5, base=10)   ## log with base 10 (alternative: log10(5)) )
```

```
## [1] 0.699
exp(5)              ## exponential function
## [1] 148.4
a <- 1/0; a         ## Infinity
## [1] Inf
b <- 0/0; b         ## Not a number
## [1] NaN
7 %/% 2             ## integer division
## [1] 3
7 %%  2             ## modulo; remainder
## [1] 1
pi                  ## pi
## [1] 3.142
1e-6                ## 10^(-6)
## [1] 1e-06
```

1.2 Vectors and indexing

R has several data structures and vectors is the most fundamental one. The variables `m` and `n` created previously are actually vectors of length 1. Additional data structures will be discussed in Chapter **??**. We shall create more interesting vectors below: The function `c()` will concatenate its input into a vector. For example we can register the production of oil in Norway (in mio. tons) for different years as two different vectors.

```
year <- c(1971, 1976, 1981, 1986, 1991, 1996, 2001, 2005)
year
## [1] 1971 1976 1981 1986 1991 1996 2001 2005
production <- c(.3, 9.3, 24.0, 42.5, 93.3, 156.8, 162.1, 138.1)
production
## [1]   0.3   9.3  24.0  42.5  93.3 156.8 162.1 138.1
```

Computations in R are VECTORIZED which has as a consequence that we can easily convert the production to tons or calculate the square root of all elements in `prod` as

```
production * 100
## [1]    30   930  2400  4250  9330 15680 16210 13810
sqrt(production)
## [1]  0.5477  3.0496  4.8990  6.5192  9.6592 12.5220 12.7318 11.7516
```

One main virtue of R is that it is so easy to work with data using a simple indexing mechanism. For example, on the square root scale, oil production grows approximately linearly until the end of the 20th century. Suppose that we want to create new vectors only with the data from the 20th century by extracting sub-vectors of `year` and `production`.

A very simple approach is to notice that the relevant data are in the first 6 entries in the two data vectors. We can extract these data by creating a vector with the entries we want and put this into square brackets:

```
entries <- 1:6 # same as c(1, 2, 3, 4, 5, 6) but much shorter.
entries
## [1] 1 2 3 4 5 6
production2 <- production[ entries ]
production2
## [1]   0.3   9.3  24.0  42.5  93.3 156.8
```

This works fine in this small example but pinpointing specific entries in a longer vector becomes tedious and error prone. We can therefore do the following:

```
b <- year < 2000
b
## [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE
```

The statement `year < 2000` means that for each element in `year` it is checked whether the year is larger than 2000 or not. Hence this is another example of a vectorized computation. The result is a logical vector which is stored in the variable `b`.

To extract the year and production corresponding to the condition that `year` is smaller than 2000 we can do

```
year[b]
## [1] 1971 1976 1981 1986 1991 1996
production[b]
## [1]   0.3   9.3  24.0  42.5  93.3 156.8
```

So we have seen that we can index a vector by specifying entries or via a logical vector (of the same length as the vector we want to index). We notice that we can easily get from the latter to the former using `which()`.[1]

```
which(b)
## [1] 1 2 3 4 5 6
```

The steps above do not change `year` and `production`, they only extract sub vectors of these vectors. Let us create new variables containing these vectors:

```
year2 <- year[b]
production2 <- production[b]
```

To continue with the indexing, consider this:

```
production
## [1]   0.3   9.3  24.0  42.5  93.3 156.8 162.1 138.1
b <- production > 150; b
## [1] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE
```

The statement `production > 150` implies that for each element in `production` it is checked whether it is larger than 150 or not.

[1] FiXme Note: Need any, all, NA, is.na(); need also saplly and lapply; also class and is()/as()...

```
which( b )  ## where in 'production' are these values
## [1] 6 7
production[ b ]   ## what are these values
## [1] 156.8 162.1
year[ b ]   ## in which year was this production
## [1] 1996 2001
```

We can find the productions and years for the cases where the production is smaller than 150 using logical negation `!` (which turns `TRUE` into `FALSE` and vice versa):

```
production[ !b ]
## [1]   0.3   9.3  24.0  42.5  93.3 138.1
year[ !b ]
## [1] 1971 1976 1981 1986 1991 2005
```

We can find data for production over 50 before the turn of the 20th century by combining logical expressions:

```
b <- (production > 50) & (year < 2000); b ## '&' is logical AND
## [1] FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE
production[ b ]; year[ b ]
## [1]  93.3 156.8
## [1] 1991 1996
```

Similarly, we can find production data before 1980 and after 2000:

```
b <- (year < 1980) | (year > 2000); b ## '|' is logical OR
## [1]  TRUE  TRUE FALSE FALSE FALSE FALSE  TRUE  TRUE
production[ b ]; year[ b ]
## [1]   0.3   9.3 162.1 138.1
## [1] 1971 1976 2001 2005
```

Suppose we discover that all years before the end of the 20th century is off by one year. This is easy to repair by replacing some elements of a vector

```
b <- year < 2000
b
## [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE
year[b]     ## The wrong years
## [1] 1971 1976 1981 1986 1991 1996
year[b] + 1 ## The correct years
## [1] 1972 1977 1982 1987 1992 1997
## Replace the relevant entries of year by the correct values:
year[b] <- year[b] + 1; year
## [1] 1972 1977 1982 1987 1992 1997 2001 2005
```

1.3 Data frames

A data frame is a list of vectors all of the same length and can be viewed as the "spreadsheet" of R. Data frames are usually formed by importing data from a file into R (more about this in Chapter **??**), but a data frame can also be created from vectors using the `data.frame()` function; for example:

```
oil <- data.frame(yr=year, prod=production)
```

Hence, a dataframe provides a handle on many vectors at one time. The first rows of `oil` are displayed with

```
head( oil, 4 )
##     yr prod
## 1 1972  0.3
## 2 1977  9.3
## 3 1982 24.0
## 4 1987 42.5
```

The indexing mechanism applies to dataframes as well; but here we use two indices:

```
oil[2:4, c(2,1)]
##   prod   yr
## 2  9.3 1977
## 3 24.0 1982
## 4 42.5 1987
```

The first index refers to rows and the second to columns. If we do not write anything for the first index then all rows are selected and likewise for columns.

We can EXTRACT A COLUMN in a data frame in different ways: either by through the $ operator or by indexing the column number.

```
oil$prod # Get the 'prod' variable from the 'oil' data frame
## [1]   0.3   9.3  24.0  42.5  93.3 156.8 162.1 138.1
## oil[["prod"]] ## same
oil[,2]  # Get the 2nd column from the 'oil' data frame
## [1]   0.3   9.3  24.0  42.5  93.3 156.8 162.1 138.1
## oil[[2]] ## same
```

We can ADD A COLUMN and DELETE A COLUMN with

```
oil$extra <- 1:8
oil$extra <- NULL
```

1.4 Using add–on packages

We can use a scatter plot to visuaize the relationship between year and production. Much of R's versatility comes from the many add-on packages available from CRAN (the Comprehensive R Archive Network), see `www.r-project.org` (at the time of writing this, there are some 10.000 packages on CRAN). One package which we shall use extensively is the **ggplot2** package. This package does not come with the default installation of R, so it must be installed separately on the computer which can be done as

```
install.packages("ggplot2")
```

This installation must be done only once. To make the package available in an R session the package must be loaded with the `library()` function

```
library(ggplot2)
```

1.5 Plotting

Once the **ggplot2** package is loaded functions in the package can be used, e.g.

```
qplot(yr, prod, data=oil)
```

Sometimes one does not wish to load an entire package just to call one function once. In this case one can use `::` as

```
ggplot2::qplot(yr, prod, data=oil)
```

1.6 Simple (linear) regression

Next we look at a simple dataset: Age and fat percentage in 9 adults:

```
age <- c(23, 28, 38, 44, 50, 53, 57, 59, 60)
fatpct <- c(19.2, 16.6, 32.5, 29.1, 32.8, 42, 32, 34.6, 40.5)
```

Data is shown as dots in Figure 1.1. By LINEAR REGRESSION we find the best approximating straight line, using the method of ORDINARY LEAST SQUARES (OLS). The function in R is `lm()` (short for linear model).

```
reg <- lm(fatpct ~ age)
summary(reg)
##
## Call:
## lm(formula = fatpct ~ age)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -5.115 -3.599 -0.521  1.759  7.053
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)    6.225      5.711    1.09   0.3118
## age            0.542      0.120    4.51   0.0028 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

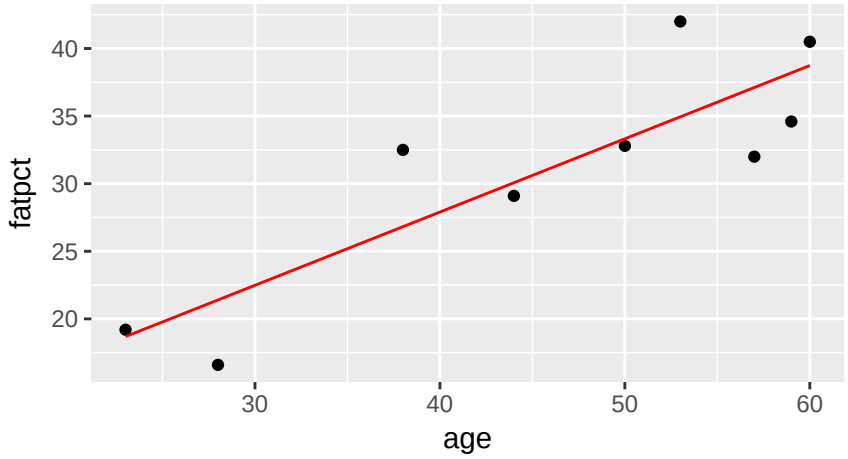


Figure 1.1:

```
## Residual standard error: 4.61 on 7 degrees of freedom
## Multiple R-squared:  0.744,Adjusted R-squared:  0.707
## F-statistic: 20.3 on 1 and 7 DF,  p-value: 0.00277
```

The column `Estimate` shows the estimated regression coefficients. The interpretation is that when year increases by one year then the square root of the production increases by 0.54.

The plot in Figure 1.1 shows estimated regression line added on top of the data points. The plot is created as:

```
qplot(age, fatpct) +
    geom_line(aes(age, predict(reg)), color="red")
```

1.7 Getting help

Commands in `R` are really function calls. Example: `R` can be terminated by the function `q()`.

Help about a command is obtained with `help()` or `?`. Example: `rnorm()` simulates data from a normal distribution and help is obtained with

```
help( rnorm )
```

The possible arguments to a function are shown with

```
args( rnorm )
## function (n, mean = 0, sd = 1)
## NULL
```

The command `help( help )` will give you an overview over the help facilitites and `help.start()` will open a web–browser with the help facilities.

Most help pages have an examples-section at the end and these examples are often worthwhile to study for inspiration.

1.7.1 Getting help on a function that you know the name of

Use `?` or, equivalently, `help()`:

```
?mean
help(mean) # same
```

`args` shows you the arguments for a function.

```
args(mean)
## function (x, ...)
## NULL
args(mean.default)
## function (x, trim = 0, na.rm = FALSE, ...)
## NULL
args(read.csv)
## function (file, header = TRUE, sep = ",", quote = "\"", dec = ".",
##     fill = TRUE, comment.char = "", ...)
## NULL
```

For non-standard names use quotes or backquotes:

```
?`if`
?"if"       # same
help("if")  # same
```

There are also help pages for datasets, general topics and some packages:

```
?iris
?Syntax
?lubridate
```

Use the `example()` function to see examples of how to use it.

```
example(paste)
example(`for`)
```

The `demo()` function gives longer demonstrations of how to use a function.

```
demo()                          # all demos in loaded pkgs
demo(package = .packages(all.available = TRUE)) # all demos
demo(plotmath)
demo(graphics)
```

1.7.2 Finding a function that you do not know the name of

Use `??` or, equivalently, `help.search`:

```
??regression
help.search("regression")
```

Again, non-standard names and phrases need to be quoted.

```
??"logistic regression"
```

apropos finds functions and variables that match a regular expression.

```
apropos("z$") # all fns ending with "z"
```

RSiteSearch() searches several sites directly from R. findFn() in **sos** wraps RSiteSearch() returning the results as a HTML table.

```
RSiteSearch("logistic regression")
library(sos)
findFn("logistic regression")
```

rseek.org is an R search engine with a Firefox plugin.

1.7.3 Finding packages

available.packages() tells you all the packages that are available in the repositories that you set via setRepositories().

installed.packages() tells you all the packages that you have installed in all the libraries specified in .libPaths. library() (without any arguments) is similar, returning the names and tag-line of installed packages.

```
View(available.packages())
View(installed.packages())
library()
.libPaths()
```

Similarly, data() with no arguments tells you which datasets are available on your machine.

```
data()
```

search() tells you which packages have been loaded.

```
search()
```

packageDescription() shows you the contents of a package's DESCRIPTION file. Likewise news read the NEWS file.

```
packageDescription("utils")
news(package = "ggplot2")
```

For finding which packages are loaded sessionInfo is quite nice.

```
sessionInfo()
```

Help on a specific package is achieved with `help()`:

```
help(package="sos")
```

1.7.4 Getting help on variables

`ls` lists the variables in an environment.

```
ls()                 # global environment
ls(all.names = TRUE) # including names beginning with '.'
ls("package:sp")     # everything for the sp package
```

Most variables can be inspected using `str` or `summary`:

```
str(sleep)
summary(sleep)
```

`ls.str` is like a combination of `ls` and `str`.

```
ls.str()
ls.str("package:grDevices")
lsf.str("package:grDevices")  # only functions
```

For large variables (particularly data frames), the `head` function is useful for displaying the first few rows.

```
head(sleep)
##   extra group ID
## 1   0.7     1  1
## 2  -1.6     1  2
## 3  -0.2     1  3
## 4  -1.2     1  4
## 5  -0.1     1  5
## 6   3.4     1  6
```

1.7.5 General learning about R

The Info page `https://stackoverflow.com/tags/r/info` is a very comprehensive set of links to free R resources.

Many topics in R are documented via vignettes, listed with `browseVignettes()`:

```
browseVignettes()
vignette("intro_sp", package = "sp")
```

By combining `vignette` with `edit`, you can get its code chunks in an editor.

```
edit(vignette("intro_sp",package="sp"))
```

1.8 Technicalities

1.8.1 On calling functions

Everything we do in R amounts to calling functions. For example addition of two numbers can be done as:

```
"+"(7, 9)
```

and typing 7 + 9 is just syntactic sugar on top of the function call above.

Consider this example: Add 10 to the value of elements number 1, 2 and 4 in the vector v below:

```
v <- c(7, 9, 0, 4, 8)
v[c(1, 2, 4)] <- v[c(1, 2, 4)] + 10
v
## [1] 17 19  0 14  8
```

The "square brackets" above have two meanings, so to speak and that can cause confusion. Extracting elements is done with the "square bracket extractor function" while replacement is done with the "square bracket assignment function":

```
v <- c(7, 9, 0, 4, 8)
z <- v[c(1, 2, 4)]
v[c(1, 2, 4)] <- z + 10
v
## [1] 17 19  0 14  8
```

Under the hood, there is a call to the "square bracket extractor function" `"["` and the "square bracket assignment function" `"[<"`:

```
v <- c(7, 9, 0, 4, 8)
z <- "["(v, c(1, 2, 4))
v <- "[<-"(v, c(1, 2, 4), z + 10)
v
## [1] 17 19  0 14  8
```

Therefore, the calls below are equivalent

```
v <- c(7, 9, 0, 4, 8)
v[c(1, 2, 4)] <- v[c(1, 2, 4)] + 10
v
## [1] 17 19  0 14  8
v <- c(7, 9, 0, 4, 8)
v <- "[<-"(v, c(1, 2, 4), "["(v, c(1, 2, 4)) + 10)
v
## [1] 17 19  0 14  8
```

1.8.2 Vectors, lists, and data frames

Here we give some more information about data structures in R.

1.8.3 Vectors

The basic data structure in R is a vector. Think of a vector as a train wagon in which all passengers are of the same type.

Vectors can be created using the c() function (short for concatenate)

```
v1 <- c("here", "comes", "the", "sun") # Character vector
v2 <- c(7, 9, 13)                      # Numeric vector
v3 <- c(T, F, T)                       # Logical vector
v1; v2; v3
## [1] "here"  "comes" "the"   "sun"
## [1]  7  9 13
## [1]  TRUE FALSE  TRUE
```

Elements of vectors can be named (and used for indexing)

```
x <- c(a=123, b=234, c=345); x
##   a   b   c
## 123 234 345
x[c("a", "c")]
##   a   c
## 123 345
```

Elements are coerced to the least restrictive type:

```
x <- c(1, 2, 3, "hello world"); x
## [1] "1"           "2"           "3"           "hello world"
```

In this case the vector contains a character string and a string can represent both a number and a string so the full vector is coerced to a character vector.

1.8.4 Lists

If a vector is a train wagon in which all passengers have the same type, then a list is a train consisting of a sequence of train wagons. Wagons can be named.

```
x <- list(a=c(2, 3), b="hello world", c(T, F))
x
## $a
## [1] 2 3
##
## $b
## [1] "hello world"
##
## [[3]]
## [1]  TRUE FALSE
```

Indexing:

```
x[ c(1, 2) ] # The train consisting of wagons 1 and 2
## $a
## [1] 2 3
##
## $b
## [1] "hello world"
x[ c("a", "b") ]
## $a
## [1] 2 3
##
## $b
## [1] "hello world"
x[ 1 ]       # The train consisting of wagon 1 only
## $a
## [1] 2 3
x[ "a" ]
## $a
## [1] 2 3
x[[ 1 ]]     # Wagon 1
## [1] 2 3
x$a
## [1] 2 3
```

Remove and add wagons:

```
x$a <- NULL; x
## $b
## [1] "hello world"
##
## [[2]]
## [1]  TRUE FALSE
x$h <- 2:4; x    # Wagons are added at the end
## $b
## [1] "hello world"
##
## [[2]]
## [1]  TRUE FALSE
##
## $h
## [1] 2 3 4
x[5] <- 1000; x  # Empty wagons are added too
## $b
## [1] "hello world"
##
## [[2]]
## [1]  TRUE FALSE
##
## $h
## [1] 2 3 4
##
## [[4]]
## NULL
```

```
##
## [[5]]
## [1] 1000
```

The train analogy does not carry through all the way: An element of a list can be a list itself (i.e. a wagon can be a train itself):

```
x <- list(a=c(2, 3), b="hello world",
          c=c(T, F), d=list(g=23, 45)); x
## $a
## [1] 2 3
##
## $b
## [1] "hello world"
##
## $c
## [1]  TRUE FALSE
##
## $d
## $d$g
## [1] 23
##
## $d[[2]]
## [1] 45
str( x )
## List of 4
##  $ a: num [1:2] 2 3
##  $ b: chr "hello world"
##  $ c: logi [1:2] TRUE FALSE
##  $ d:List of 2
##   ..$ g: num 23
##   ..$  : num 45
```

1.8.5 Dataframes

A dataframe is basically a list in which each element has the same length.

```
d <- data.frame(x=5:8, y=c("here", "comes", "the", "sun"),
                z=c(T, F, T, T))
d
##   x     y     z
## 1 5  here  TRUE
## 2 6 comes FALSE
## 3 7   the  TRUE
## 4 8   sun  TRUE
```

Operate on dataframes as on lists. In addition, subscript as for matrices:

```
d[c(1, 4), c(1, 3)]
##   x    z
## 1 5 TRUE
## 4 8 TRUE
```

1.8.6 Matrices

Matrices can be constructed with the `matrix()` function. For example, a 3×3 matrix with the integers from one to nine is defined as follows.

```
A <- matrix(1:9, nrow=3, ncol=3); A;
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
dim( A )
## [1] 3 3
```

The numbers are read column-wise; if this is not what you want, use the optional argument byrow=TRUE to `matrix()`. Indexing matrices is simlar to indexing vectors except that an index vector defining rows and an index vector defining columns are needed.

```
A[1:2, c(1,3)]  ## extract submatrix by some rows and cols
##      [,1] [,2]
## [1,]    1    7
## [2,]    2    8
A[1:2, ]        ## extract submatrix by some rows and and all cols
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
A[2, ]          ## no longer a matrix, but a vector
## [1] 2 5 8
A[2, , drop=FALSE]   ## still a matrix
##      [,1] [,2] [,3]
## [1,]    2    5    8
```

1.8.7 Iterating over rows and columns of a matrix

To calculate e.g. the sum of each row we can use APPLY() or ROWSUMS()

```
rs <- apply(A, 1, sum) ## or rs <- rowSums(A)
rs
## [1] 12 15 18
```

Say we want to subtract from each column the mean of that column. We can use e.g. SWEEP

```
cm <- colMeans(A)
sweep(A, 2, cm, FUN="-")
##      [,1] [,2] [,3]
## [1,]   -1   -1   -1
## [2,]    0    0    0
## [3,]    1    1    1
```

1.8.8 Simulating random and systematic data

In addition to `c()`, vectors can be created in other ways, for example using `:` and `seq()` which both cretes sequences of numbers, and `rep()` which replicates a vector.

```
1:4    # Sequence of integer numbers from 1 to 4
## [1] 1 2 3 4
3:-3   # Sequence of integer numbers from 3 to -3
## [1]  3  2  1  0 -1 -2 -3
seq(1,3, by=0.5)   # Sequence from 1 to 3 with step size 0.5
## [1] 1.0 1.5 2.0 2.5 3.0
seq(1,3, length=4) # Sequence of length 4 starting at 1 and ending at 3
## [1] 1.000 1.667 2.333 3.000
rep(1:3, times=2)  # Replicate 1, 2, 3 twice
## [1] 1 2 3 1 2 3
rep(1:3, each=2)   # Replicate each of 1, 2 3 twice
## [1] 1 1 2 2 3 3
```

Random data can be generated as follows: Samples from 5 independent uniform random random variables on $[0, 10]$ are generated by `runif()`:

```
runif( n = 5, min = 0, max = 10 )
## [1] 6.386 5.042 9.595 1.221 3.705
```

Samples from 5 independent normal random random variables with mean 1 and standard deviation 2 are generated by `rnorm()`:

```
rnorm( n = 5, mean = 1, sd = 2 )
## [1]  3.7288 -0.8903  0.1128 -0.1204  2.4977
```

1.8.9 Getting data into functions

If we want to plot `production` against `year` we can do

```
plot(prod ~ yr, data=oil)
```

This works because there is a version of `plot()` that takes data as an argument; above `data=oil`. A function like `smooth.spline()`, however, does not take data as input:

```
args(smooth.spline)
## function (x, y = NULL, w = NULL, df, spar = NULL, lambda = NULL,
##     cv = FALSE, all.knots = FALSE, nknots = .nknots.smspl, keep.data = TRUE,
##     df.offset = 0, penalty = 1, control.spar = list(), tol = 1e-06 *
##         IQR(x), keep.stuff = FALSE)
## NULL
```

Hence we must provide data in another way. Can be done different ways:

- Using dollar sign

```
sm <- smooth.spline(oil$yr, oil$prod, df=4)
```

- or - more elegantly - using `with()`

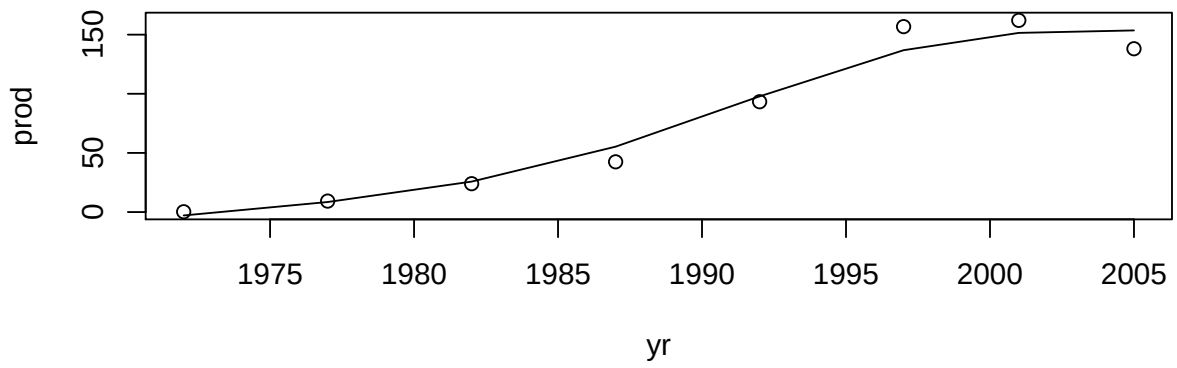
```
sm <- with(oil, smooth.spline(yr, prod, df=4))
```

- or - but careful here - using ATTACH and DETACH

```
attach(oil)
sm <- smooth.spline(yr, prod, df=4)
detach(oil)
```

The spline fit looks like:

```
plot(prod ~ yr, data=oil)
lines(sm)
```



1.8.10 R as a programming language

In addition to the statstical and mathematical facilities, one virtue of R is that it is a programming language. While large scale programming with R is not the main focus of this book, we shall from time to time write small functions "on the fly" as we need them.

Example: Calculate the sum of the numbers $1, 2, 3, \ldots, N$. This sum can be computed as $N(N+1)/2$, but if we did not know that we could simply do:

```
N <- 100
result <- 0
for (i in 1:N)
    result <- result + i
cat("The sum of the numbers from 1 to", N, "is:", result, "\n")
## The sum of the numbers from 1 to 100 is: 5050
```

We can create a general purpose function that does the trick:

```
add1toN <- function( N ){
    result <- 0
    for (i in 1:N)
        result <- result + i
    cat("The sum of the numbers from 1 to", N, "is:", result, "\n")
    result
}
add1toN( 1000 )
## The sum of the numbers from 1 to 1000 is: 500500
## [1] 500500
add1toN( 10 )
## The sum of the numbers from 1 to 10 is: 55
## [1] 55
```

Chapter 2

Programming - R functions

2.1 Hello World

Functions in R are like mathematical functions: Rules that converts some input arguments (possibly none, possibly many) to some output (possibly none represented by NULL in R).

```
## define function
hello <- function(){
  cat("Hello world\n")
}
## call function
hello()
## Hello world
```

The general syntax for R functions is

```
f <- function(arg1, arg2, arg3, ...){
  ## expr1
  ## expr2
  ## ...
  ## exprn (The last expression is returned)
}
```

2.2 Input and output

The last expression is returned:

```
add <- function(x, y){
  x + y
}
add(2, 5)
## [1] 7
```

Arguments can be given a default value:

```r
add <- function(x, y=0){
  x + y
}
add(2)
## [1] 2
## or
add <- function(x, y=x){
  x + y
}
add(7)
## [1] 14
```

To return more values we can use a list. For example:

```r
add <- function(x,y){
  list(sum=x + y, x=x, y=y)
}
add(2, 5)
## $sum
## [1] 7
##
## $x
## [1] 2
##
## $y
## [1] 5
```

Returning multiple values from function can be obtained with attributes. Attributes can be set with `attr()`:

```r
add <- function(x, y){
  out <- x + y
  attr(out, "xy") <- c(x, y)
  out
}
z <- add(2, 5)
z
## [1] 7
## attr(,"xy")
## [1] 2 5
attr(z, "xy")
## [1] 2 5
```

2.3 Further details*

Using triple–dots `...` in argument list allows for unspecified additional arguments.

```r
test <- function(x, y, ...){
    print(x)
    args = list(...)
```

```
    if ('z' %in% names(args))
        print(args$z)
}
```

```
test(123)
## [1] 123
test(234, z="a")
## [1] 234
## [1] "a"
```

Example: Consider

```
test <- function(x, y, ...){
    cat("x=", x, "\n")
    if (missing(y)) cat("y missing\n") else print(y)
    args = list(...)
    if('z' %in% names(args))
        print(args$z)
}
```

```
test(234, z="a")
## x= 234
## y missing
## [1] "a"
test(234, y=3.14, z="a")
## x= 234
## [1] 3.14
## [1] "a"
```

2.4 Repetetive and conditional execution

```
f1 <- function(n){
  for (i in 1:n){
      print(i)
  }
}
f1(3)
## [1] 1
## [1] 2
## [1] 3
f2 <- function(n){
    i <- 1
    while(i <= n){
        print(i)
        i <- i+1
    }
}
f2(3)
```

```
## [1] 1
## [1] 2
## [1] 3
f3 <- function(n){
    i <- 1
    repeat{
        print(i)
        if (i == n)
            break
        i <- i + 1
    }
}
f3(3)
## [1] 1
## [1] 2
## [1] 3
```

The general forms are:

```
for (name in expr1) {
    expr2
}
repeat {
    expr
}
while (condition) {
    expr
}
```

Here `condition` must be a single logical value.

The `break` statement can be used terminate any loop (and it is the only way to terminate a `repeat` loop).

A collection of expressions can be grouped by curly braces.

Conditional execution:

```
if (condition) {
    expr1
} else {
    expr2
}
```

2.5 Example: Calculating the square root

Finding the square root of $x > 0$ is the same as solving

$$f(z) = z^2 - x = 0$$

for z, which in turn is the same as solving $z - x/z = 0$.

The "Babylonian method":

1. Start with initial guess for z (any starting value will do; for example x itself).
2. Iteratively replace z by the average of z and x/z, i.e. set $z \leftarrow \frac{1}{2}(z + x/z)$.
3. Stop when z and x/z are as close as desired.

This method is really the same as a Newton–Rapson method:

$$z_{n+1} \leftarrow z_n - \frac{f(z_n)}{f'(z_n)} = \frac{1}{2}\left(z_n + \frac{x}{z_n}\right)$$

```
## Using while(){}
mysqrt1 <- function(x, z=x){
  while( abs(z - x/z) > 1e-7 ){
    z <- (z + x/z)/2
  }
  z
}
```

```
x <- c(.01, 1, 10, 100, 1000, 10000)
sapply(x, mysqrt1)
## [1]   0.100   1.000   3.162  10.000  31.623 100.000
sapply(x, sqrt)
## [1]   0.100   1.000   3.162  10.000  31.623 100.000
```

```
## Using repeat{}
mysqrt2 <- function(x, z=x){
    repeat{
        z.old <- z
        z <- (z + x/z)/2
        if (abs(z - z.old) < 1e-7)
            break
    }
    z
}
```

```
sapply(x, mysqrt2)
## [1]   0.100   1.000   3.162  10.000  31.623 100.000
sapply(x, sqrt)
## [1]   0.100   1.000   3.162  10.000  31.623 100.000
```

```
## Using for(){}
mysqrt3 <- function(x, z=x){
    for( i in 1:10 ){
        z <- (z + x / z) / 2
    }
    z
}
```

```
sapply(x, mysqrt3)
## [1]   0.100   1.000   3.162  10.000  31.623 100.000
sapply(x, sqrt)
## [1]   0.100   1.000   3.162  10.000  31.623 100.000
```

2.6 Example: Recursion and iteration

The factorial function $n! = n \cdot (n-1) \cdot (n-2) \cdot \cdots \cdot 2 \cdot 1$ can be defined recursively as $n! = f(n) = nf(n-1)$ with $f(1) = 1$.

```
## Recursive implementation:
fact.rec <- function(n){
  if (n == 1) {
      1
  } else {
    n * fact.rec(n - 1)
  }
}
## Iterative implementation:
fact.it <- function(n){
  out <- 1
  for (i in 2:n){
    out <- out * i
  }
  out
}
```

2.7 Exercises – programming

2.7.1 Calculating the (natural) logarithm

Calculating the (natural) logarithm of $x > 0$ can be done by solving $f(z) = \exp(z) - x = 0$. The Newton step becomes

$$z_{n+1} \leftarrow z_n - \frac{f(z_n)}{f'(z_n)} = z_n - 1 + \frac{x}{\exp(z_n)}$$

1. Implement this in the function `mylog()`. Think about what a good starting value for the iteration could be.

2. Benchmark your function along these lines:

```
d <- (mylog(1000) - log(1000)) / log(1000); d
## [1] 0
d <- (mylog(10) - log(10)) / log(10); d
## [1] 0
```

2.7.2 Calculating the pth root

Finding the pth root of $x > 0$ where p a positive integer is the same as solving

$$f(z) = z^p - x = 0$$

for z.

1. Derive the Newton–Rapson step for solving this equation iteratively and implement a function called `myrootp(x, p)`.
2. Benchmark your function along these lines:

```
x <- 8; p <- 2
d <- (myrootp(x, p) - x^(1 / p)) / x^(1 / p); d
## [1] -1.57e-16
x <- 8; p <- 3
d <- (myrootp(x, p) - x^(1 / p)) / x^(1 / p); d
## [1] 0
```

Chapter 3

Computer arithmetic

3.1 Computer arithmetic is not exact

The following R statement appears to give the correct answer.

```
0.3 - 0.2
## [1] 0.1
```

However,

```
0.3 - 0.2 == 0.1
## [1] FALSE
```

The difference between the values being compared is small, but important.

```
(0.3 - 0.2) - 0.1
## [1] -2.776e-17
```

3.2 Floating point arithmetic

A computers representation of real numbers is by floating point numbers. Floating point numbers are represented as

$$x = s \times b^E$$

where s is the SIGNIFICAND (or MANTISSA), b is the BASE and E is the EXPONENT.

If $1 \leq |s| < b$ then x is said to be NORMALIZED.

R has "numeric" (floating points) and "integer" numbers

```
class(1)
## [1] "numeric"
class(1L)
## [1] "integer"
```

Mathematical operations on integers are "exact"; mathematical operations on floating point numers are only approximate.

3.3 Addition and subtraction

In $x = s \times b^E$, let s be a 7–digit number

A simple method to add floating-point numbers is to first represent them with the same exponent.

```
123456.7 = 1.234567 * 10^5
101.7654 = 1.017654 * 10^2 = 0.001017654 * 10^5
```

So the true result is

```
(1.234567 + 0.001017654) * 10^5 = 1.235584654 * 10^5
```

But if the computer can only work with 7 significant digits, the approximate result the computer would give is (the last digits (654) are lost)

```
1.235585 * 10^5    (final sum: 123558.5)
```

In extreme cases, the sum of two non-zero numbers may be equal to one of them

Quiz: In which order should on sum a sequence of positive numbers $x_1, \ldots, x_n$ to obtain large accuracy?

3.4 The Relative Machine Precision

The accuracy of a floating-point system is measured by the relative machine precision or machine epsilon which is the smallest positive value which can be added to 1 to produce a value different from 1.

A machine epsilon of 10^{-7} indicates that there are roughly 7 decimal digits of precision in the numeric values stored and manipulated by the computer.

It is easy to write a program to determine the relative machine precision

```
.Machine$double.eps
## [1] 2.22e-16
```

```
machine.eps <- function(){
  eps <- 1
  while(1+eps/2 != 1)
    eps <- eps / 2
  eps
}
machine.eps()
## [1] 2.22e-16
```

The preceding program shows that there are roughly 16 decimal digits of precision to R arithmetic.

3.5 Error in Floating-Point Computations

Subtraction of positive values is one place where the finite precision of floating-point arithmetic is a potential problem.

```
x = 1 + 1.234567890e-10
print(x, digits = 20)
## [1] 1.0000000001234568003
```

```
y = x - 1
print(y, digits = 20)
## [1] 1.2345680033831740 73e-10
```

There are 16 correct digits in x, but only 6 correct digits in y.

Subtraction of nearly equal quantities (known as NEAR CANCELLATION or CATASTROPHIC CANCELLATION) is a major source of inaccuracy in numerical calculations.

3.6 Example: Numerical derivatives

The derivative $f'(x)$ may be approximated by

$$f'(x) \approx \frac{f(x + h/2) - f(x - h/2)}{h}, \quad h \text{ small}$$

A generic R function is

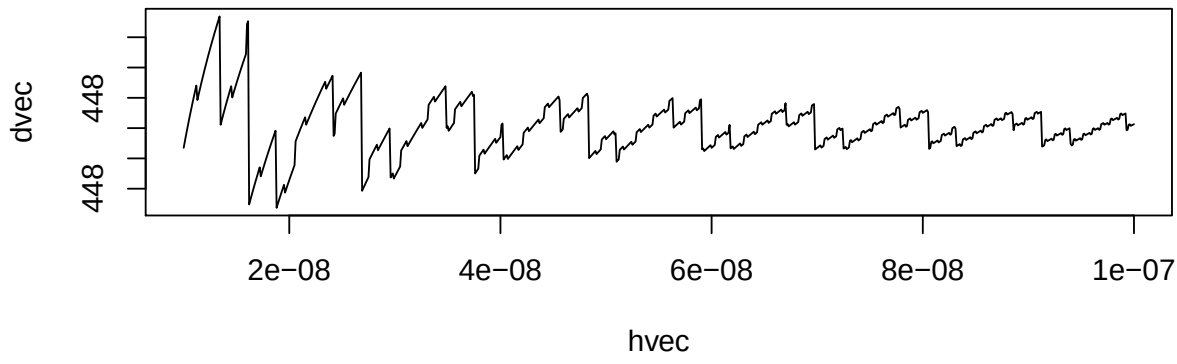
```
numDeriv <- function(f, x, h=1e-8){
  (f(x + h / 2) - f(x - h / 2)) / h
}
```

For small h we get near cancellation errors. For large h we get imprecise derivative:

```
f <- function(x){x^7}
x <- 2
d <- 7 * x^6 # True derivative
d
## [1] 448
print(numDeriv(f, x, h=1), digits=20)
## [1] 593.265625
print(numDeriv(f, x, h=.01), digits=20)
## [1] 448.01400005248979141
print(numDeriv(f, x, h=1e-8), digits=20)
## [1] 447.99999869837847655
print(numDeriv(f, x, h=1e-15), digits=20)
## [1] 397.90393202565604724
```

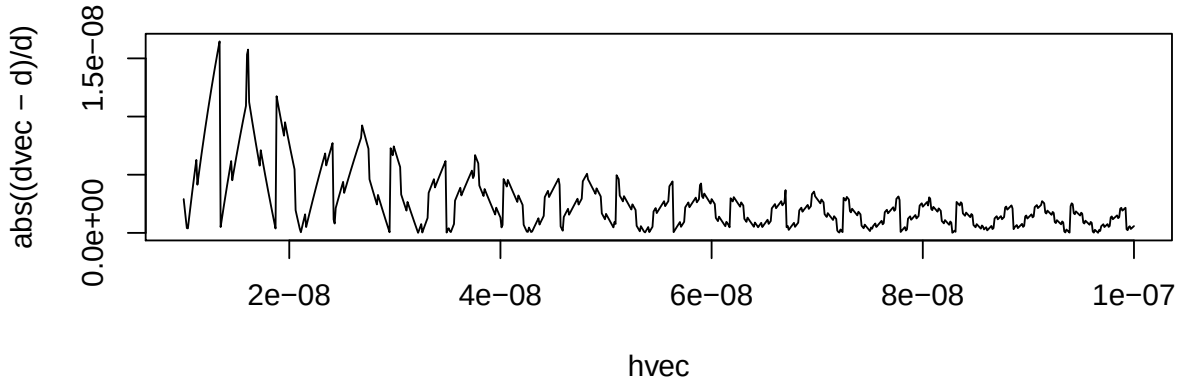
Interesting plot:

```
hvec <- seq(1e-8, 1e-7, 1e-10)
dvec <- numDeriv(f, x, hvec)
plot(hvec, dvec, type="l");
```



Plot the absolute value of the relative error

```
plot(hvec, abs((dvec-d)/d), type="l")
```



3.7 Example: Computing the exponential

Recall the exponential function

$$\exp(x) = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2} + \frac{x^3}{3 \cdot 2} + \frac{x^4}{4 \cdot 3 \cdot 2} + \cdots$$

Let $t_n = \frac{x^n}{n!}$. Then $t_0 = 1$ and

$$t_{n+1} = \frac{x^{n+1}}{(n+1)!} = t_n \frac{x}{n+1}$$

so these terms must eventually become small. Hence we may stop the computations when nothing significant is added to the result.

```
myexp <- function(x){
  n <- 0; t <- 1; ans <- 1
  while(abs( t ) > .Machine$double.eps) {
    n = n + 1
    t = t * x / n
    ans <- ans + t
  }
  ans
}
```

Compare `myexp()` with R's built in function. For positive values, the results are good:

```
(myexp(1) - exp(1))/exp(1)
## [1] 1.634e-16
(myexp(20) - exp(20))/exp(20)
## [1] -1.229e-16
```

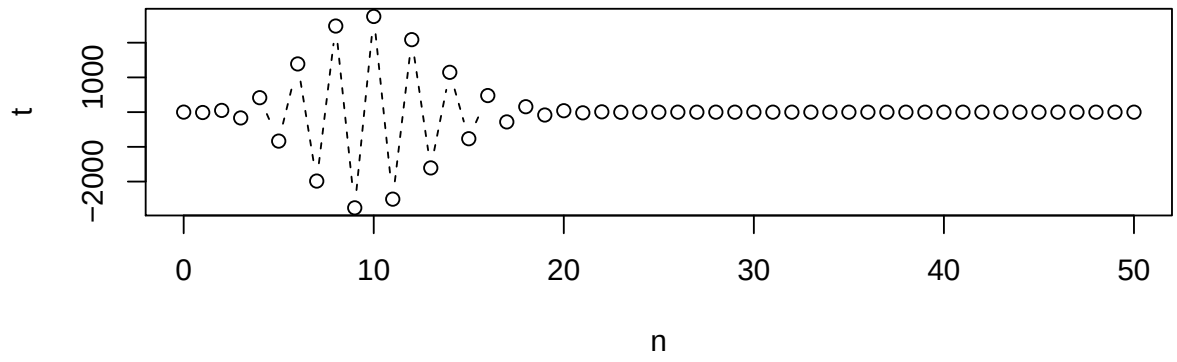
For negative values less so:

```
(myexp(-1) - exp(-1))/exp(-1)
## [1] 3.018e-16
(myexp(-20) - exp(-20))/exp(-20)
## [1] 1.728
```

Why? Look at the terms $t_n = \frac{x^n}{n!}$ in

$$\exp(x) = \sum_{n=0}^{p} \frac{x^n}{n!}$$

```
x <- -10
p <- 50
n <- 0:p
t <- x^n / factorial(n)
t
##  [1]  1.000e+00 -1.000e+01  5.000e+01 -1.667e+02  4.167e+02 -8.333e+02
##  [7]  1.389e+03 -1.984e+03  2.480e+03 -2.756e+03  2.756e+03 -2.505e+03
## [13]  2.088e+03 -1.606e+03  1.147e+03 -7.647e+02  4.779e+02 -2.811e+02
## [19]  1.562e+02 -8.221e+01  4.110e+01 -1.957e+01  8.897e+00 -3.868e+00
## [25]  1.612e+00 -6.447e-01  2.480e-01 -9.184e-02  3.280e-02 -1.131e-02
## [31]  3.770e-03 -1.216e-03  3.800e-04 -1.152e-04  3.387e-05 -9.678e-06
## [37]  2.688e-06 -7.265e-07  1.912e-07 -4.902e-08  1.226e-08 -2.989e-09
## [43]  7.117e-10 -1.655e-10  3.762e-11 -8.360e-12  1.817e-12 -3.867e-13
## [49]  8.055e-14 -1.644e-14  3.288e-15
plot(n, t, type='b', lty=2)
```



For numerical large but negative x values, $\exp(x)$ is small while at least some of the terms are large.

Hence, at some point, there has to be near-cancellation of the accumulated sum and the next term of the series. This is the source of the large relative error.

Notice: When the argument to `myexp()` is positive, all the terms of the series are positive and there is no cancellation.

There is an easy remedy: Since $\exp(-x) = 1/\exp(x)$ it is for negative x better to compute the result as $\exp(x) = 1/\exp(-x)$

```
myexp2 <- function(x){
  if (x < 0) {
    1 / myexp(-x)
  } else {
    myexp(x)
  }
}
```

```
(myexp2(-1) - exp(-1)) / exp(-1)
## [1] -1.509e-16
(myexp2(-20) - exp(-20)) / exp(-20)
## [1] 2.007e-16
```

3.8 Example: Solving a quadratic equation

Consider solving

$$ax^2 + bx + c = 0$$

Letting $D = b^2 - 4ac$, the roots are

$$r_1 = \frac{-b + \sqrt{D}}{2a}, \quad r_2 = \frac{-b - \sqrt{D}}{2a}$$

If $b^2 \gg ac$ then $\sqrt{D} = \sqrt{b^2 - 4ac} \approx |b|$.

In this case, if $b > 0$ then $-b + \sqrt{D}$ involves a near cancellation (same for $-b - \sqrt{D}$ if $b < 0$).

Solution: Rewrite the problem. Multiply numerator and dominator of r_1 by $-b - \sqrt{D}$ (and and numerator and denominator of r_2 by $-b + \sqrt{D}$) by to obtain

$$r_1 = \frac{2c}{-b - \sqrt{D}}, \quad r_2 = \frac{2c}{-b + \sqrt{D}}$$

Example:

```
## Pick the roots
r1 <- 1717; r2 <- 1.234e-12
## Then derive a, b, c
a  <-  10 ## Any value will do
b  <- -a * (r1 + r2)
c  <-  a * r1 * r2
c(a, b, c)
## [1]  1.000000e+01 -1.717000e+04  2.118778e-08
## Santity check
a * r1^2 + b * r1 + c
## [1] 2.561329e-09
a * r2^2 + b * r2 + c
## [1] 3.308722e-24
```

```
D <- b^2 - 4 * a * c
## Near cancellation errors are likely
sqrt(D)
## [1] 17170
b
## [1] -17170
- b + sqrt(D) ## OK
## [1] 34340
- b - sqrt(D) ## Potentially problematic
## [1] 2.546585e-11
```

```
## First solution
##
(x1 <- (- b + sqrt(D)) / (2 * a)) ## OK
## [1] 1717
(x2 <- (- b - sqrt(D)) / (2 * a)) ## Problematic
## [1] 1.273293e-12
## Relative errors:
c( (x1 - r1) / r1, c(x2 - r2) / r2 )
## [1] 0.00000000 0.03184164
## Second solution
##
(x1 <- 2 * c / (-b - sqrt(D))) ## Problematic
## [1] 1664.015
(x2 <- 2 * c / (-b + sqrt(D))) ## OK
## [1] 1.234e-12
## Relative errors:
c( (x1 - r1) / r1, c(x2 - r2) / r2 )
## [1] -0.03085904  0.00000000
```

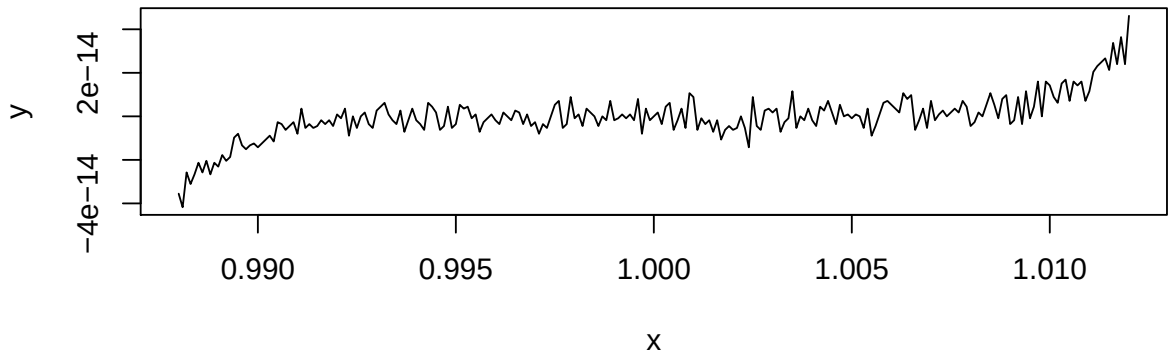
3.9 Example: A polynomial

A 7th degree polynomial; its graph should appear very smooth:

$$f(x) = x^7 - 7x^6 + 21x^5 - 35x^4 + 35x^3 - 21x^2 + 7x - 1$$

but:

```
x = seq(.988, 1.012, by = 0.0001)
y = x^7 - 7 * x^6 + 21 * x^5 - 35 * x^4 + 35 * x^3 - 21 * x^2 + 7 * x - 1
plot(x, y, type = "l")
```



To see where the cancellation error comes split the polynomial into individual terms and see what happens when we sum them.

```
options("digits"=7)
x = .999
y = c(x^7, -7 * x^6, 21 * x^5, -35 * x^4, 35 * x^3, -21 * x^2, 7 * x, -1)
sum(y)
## [1] -4.440892e-16
y
## [1]   0.993021  -6.958105  20.895210 -34.860210  34.895105 -20.958021   6.993000
## [8]  -1.000000
cumsum(y)
## [1]  9.930210e-01 -5.965084e+00  1.493013e+01 -1.993008e+01  1.496502e+01
## [6] -5.993000e+00  1.000000e+00 -4.440892e-16
```

It is the last subtraction (of 1) which causes the catastrophic cancellation and loss of accuracy.

```
sum(y[1:7]) - 1
## [1] -4.440892e-16
```

We can reformulate the problem by noticing that

$$f(x) = x^7 - 7x^6 + 21x^5 - 35x^4 + 35x^3 - 21x^2 + 7x - 1 = (x-1)^7$$

Notice that although we are still getting cancellation, when 1 is subtracted from values close to 1, we are only losing a few digits of accuracy:

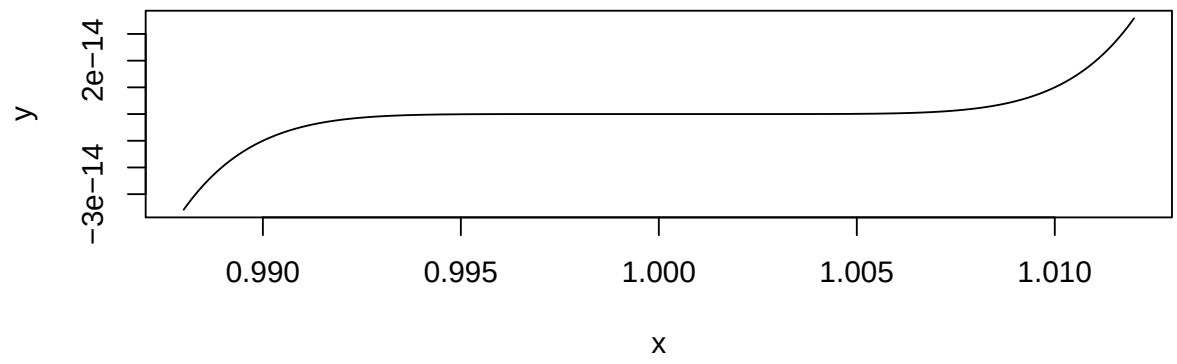
```
(x - 1)^7
## [1] -1e-21
```

The relative error is remarkable:

```
( sum(y) - (x - 1)^7 ) / (x - 1)^7
## [1] 444088.2
```

The difference is apparent in the plot.

```
x = seq(.988, 1.012, by = 0.0001)
y = (x - 1)^7
plot(x, y, type = "l")
```



3.10 Remarks and Resources

- Peter S. Turner: Guide to scientific computing.
- What Every Computer Scientist Should Know About Floating-Point Arithmetic: http://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html
- http://www.johndcook.com/blog/2009/04/06/numbers-are-a-leaky-abstraction/
- http://www.johndcook.com/blog/2009/04/06/anatomy-of-a-floating-point-number/

```
options("digits"=4)
```

Chapter 4

Summarizing data

The following vectors are an excerpt from the `mtcars` dataset: Weight `wt` (in 1000 lbs) and fuel efficiency `mpg` (in miles per gallon).

```
x <- wt <- c(2.6, 2.9, 2.3, 3.2, 3.4, 3.5, 3.6, 3.2, 3.1, 3.4, 3.4, 4.1,
3.7, 3.8, 5.2, 5.4, 5.3, 2.2, 1.6, 1.8, 2.5, 3.5, 3.4, 3.8, 3.8,
1.9, 2.1, 1.5, 3.2, 2.8, 3.6, 2.8)

y <- mpg <- c(21, 21, 22.8, 21.4, 18.7, 18.1, 14.3, 24.4, 22.8, 19.2, 17.8,
16.4, 17.3, 15.2, 10.4, 10.4, 14.7, 32.4, 30.4, 33.9, 21.5, 15.5,
15.2, 13.3, 19.2, 27.3, 26, 30.4, 15.8, 19.7, 15, 21.4)
```

We shall look at measures of where is the "location" or "center" of the data and what is the "spread" of data.

4.1 Notation

We have n observations in the vector x. We denote these symbolically by

$$x_1, x_2, x_3, \ldots, x_{n-1}, x_n$$

and they read "x one", "x two" etc. For the sum $x_1 + x_2 + x_3 + \cdots + x_n$ we write

$$x_\cdot = \Sigma_{i=1}^{n} x_i = x_1 + x_2 + x_3 + \cdots + x_n$$

and the left hand side reads "x dot" and $\Sigma_{i=1}^{n} x_i$ reads "the sum of x_i as i goes from 1 to n". The symbol $x_\cdot$ is of course an arbitrary name for the sum.

4.2 Mesures of location

If we divide $x_\cdot$ by the number of observations n we get the mean (or AVERAGE). Again, we can invent any name we like for this average, but it is quite common to write $\bar{x}$:

$$\bar{x} = \frac{1}{n} x_\cdot = \frac{1}{n} \sum_{i=1}^{n} x_i = \frac{1}{n}(x_1 + x_2 + x_3 + \cdots + x_n)$$

The most commonly used MEASURE OF LOCATION is the SAMPLE MEAN (as opposed to a theoretical quantity defined later), (or EMPIRICAL MEAN or AVERAGE):

```
sum(x) / length(x)
## [1] 3.206
mean(x)
## [1] 3.206
```

The MEDIAN is a related measure: We sort the data:

```
x2 <- sort(x)
x2
##  [1] 1.5 1.6 1.8 1.9 2.1 2.2 2.3 2.5 2.6 2.8 2.8 2.9 3.1 3.2 3.2 3.2 3.4 3.4 3.4
## [20] 3.4 3.5 3.5 3.6 3.6 3.7 3.8 3.8 3.8 4.1 5.2 5.3 5.4
```

If the number of data points is odd, the median is the middle data point. If the number is even, the median is the average of the two points in the middle:

```
median(x)
## [1] 3.3
```

4.3 Measures of spread

The most common MEASURE OF SPREAD is the SAMPLE STANDARD DEVIATION (as opposted to a theoretical quantity discussed later) (or EMPIRICAL STANDARD DEVIATION).

The squared distance of x_i to $\bar{x}$ is $(x_i - \bar{x})^2$. The average squared distance $\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$ is a measure of spread of data. For technical reasons we divide by $n-1$ rather than n and obtain the SAMPLE VARIANCE

$$s_x^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

If the units of the x_is are, say meters, then the unit of $\bar{x}$ is also meters but the unit of s_x^2 is "square meter". This leads to the SAMPLE STANDARD DEVIATION which also has unit meter:

$$s_x = \sqrt{s_x}$$

```
var( x )
## [1] 0.9496
sd( x )
## [1] 0.9745
```

There are alternative measures of spread. One is the interquartile tange (IQR). Consider this

```
quant <- quantile(x); quant
##    0%   25%   50%   75%  100%
## 1.500 2.575 3.300 3.625 5.400
```

The 50% quantile is the median. The 75% quantile is the median of the highs and the 25% quantile is the median of the lows. The difference between these two quantiles is the IQR:

```
quant[4] - quant[2]
##  75%
## 1.05
```

4.4 A practical interpretation and an empirical rule

A practical interpretation of the mean $\bar{x}$ and and standard deviation s_x is that for nearly symmetrical mound shaped datasets, about 68% of data is within one standard deviation of the mean and 95% is within two standard deviations of the mean.

Often we can get reasonable estimates of the sample mean and standard deviation by simply looking at data:

If data is not highly skewed then the mean and median are roughly the same. An estimate of the median is obtained by looking at the center of the (sorted) data vector:

```
x2 <- sort(x)
x2
##  [1] 1.5 1.6 1.8 1.9 2.1 2.2 2.3 2.5 2.6 2.8 2.8 2.9 3.1 3.2 3.2 3.2 3.4 3.4 3.4
## [20] 3.4 3.5 3.5 3.6 3.6 3.7 3.8 3.8 3.8 4.1 5.2 5.3 5.4
```

```
median( x )
## [1] 3.3
mean( x )
## [1] 3.206
```

About 95% of the observations will fall in the interval $\bar{x} \pm 2s_x$.

If we say that 95% of the observations are "practically all observations" then practically all observations fall in this interval

Hence the largest minus the smallest value is about 4 standard deviations and hence that a crude estimate of the standard deviation is

```
(max(x) - min(x)) / 4
## [1] 0.975
sd(x)
## [1] 0.9745
```

4.5 Covariance and correlation

The measurements in `x` and `y` "co–vary"

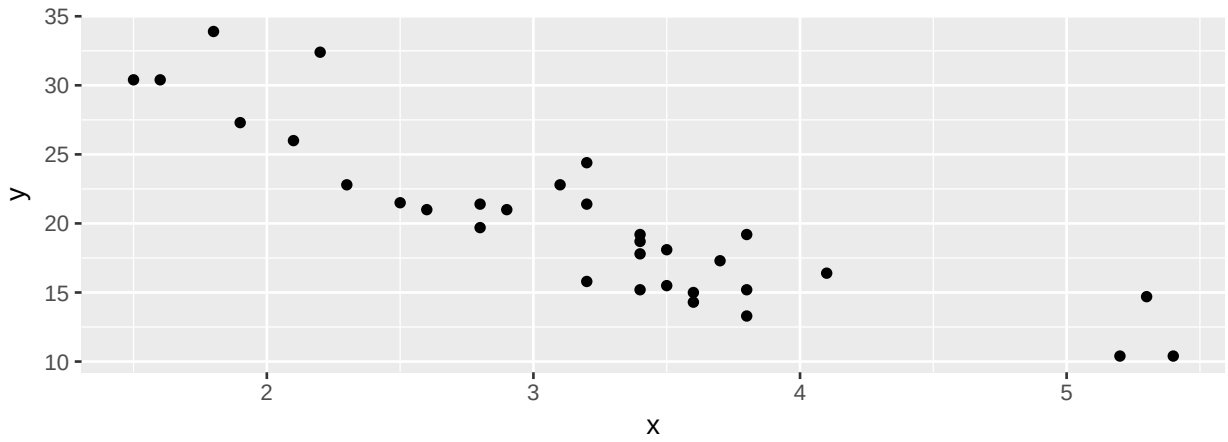


Figure 4.1: Correlated measurements.

```
library(ggplot2)
qplot(x, y)
```

A measure of how two variables co–vary is the SAMPLE COVARIANCE or EMPIRICAL COVARIANCE between x and y

$$s_{xy} = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$$

Notice: If we replace y_i by x_i and $\bar{y}$ by $\bar{x}$ above then we get the sample covariance between x and x which is the (sample) variance of x.

Closely related to the (sample) covariance is the (sample) CORRELATION COEFFICIENT:

$$r_{xy} = \frac{s_{xy}}{s_x s_y} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

The correlation is always in the interval ± 1. If the correlation is 1 or -1 there is a perfect linear association between x and y. If the correlation is 0 there is no linear association at all.

```
cov( x, y )
## [1] -5.126
cor( x, y )
## [1] -0.8728
```

4.6 The measurement unit matters – sometimes

The weight x is in 1000 pounds (lbs). One pound is about 0.45 kg, so to get the weight in kg we must multiply by $1000 \cdot .45$, so the weight in metric tonnes is $.45$ times the weight.

Now consider weight data on the new and old scale:

```
x.metric <- .45 * x
mean(x)
## [1] 3.206
mean(x.metric)
## [1] 1.443
sd(x)
## [1] 0.9745
sd(x.metric)
## [1] 0.4385
var(x)
## [1] 0.9496
var(x.metric)
## [1] 0.1923
cov(x, y)
## [1] -5.126
cov(x.metric, y)
## [1] -2.307
cor(x, y)
## [1] -0.8728
cor(x.metric, y)
## [1] -0.8728
```

More generally: Take four numbers, a, b, c and d and create new variables u_i and v_i as

$$u_i = a + bx_i, \quad v_i = c + dy_i$$

This corresponds to changing the scale and location of the data; for example changing temperature measurements from Celcius to Fahrenheit.

Then for the new variables we have

$$\bar{u} = a + b\bar{x}, \quad s_u^2 = b^2 s_x^2, \quad s_u = bs_x, \quad s_{uv} = bds_{xy}$$

but

$$\rho_{uv} = \rho_{xy}$$

Hence: mean, variance, standard deviation and covariance depends on the scale on which the variables are measured but correlation does not.

4.7 Correlation and regression

Fig 4.1 suggests that y (`mpg`) is linearly related to x (`wt`). In LINEAR REGRESSION we model variation ib y as a function of variation in x by asumming a linear relation:

$$y \approx \beta_0 + \beta_1 x \tag{4.1}$$

The parameters β_0 (the intercept) and β_1 (the slope) are to be estimated from data. The most common method is the method of LEAST SQUARES: The estimates for β_0 and β_1 are those values that minimizes the RESIDUAL SUM OF SQUARES

$$\sum_{i=1}^{n} [y_i - (\beta_0 + \beta_1 x_i)]^2$$

The estimated slope is closely related to the correlation coefficient:

$$\hat{\beta}_1 = r_{xy}\frac{s_y}{s_x} = \frac{s_{xy}}{s_x^2} = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2}$$

The estimated intercept is

$$\hat{\beta} = \bar{y} - \hat{\beta}_1 \bar{x}$$

```
beta_1 <- cov(x, y) / var(x)
beta_1
## [1] -5.398
beta_0 <- mean(y) - beta_1 * mean(x)
beta_0
## [1] 37.4
```

The LINEAR REGRESSION model in (4.1) and many other models with a similar structure can be handled with the R function `lm()`:

```
lm(y ~ x)
##
## Call:
## lm(formula = y ~ x)
##
## Coefficients:
## (Intercept)            x
##        37.4         -5.4
```

The FITTED values from the regression model are the points on the estimated line:

```
y.hat <- beta_0 + beta_1 * x
y.hat
##  [1] 23.363 21.744 24.983 20.124 19.045 18.505 17.965 20.124 20.664 19.045
## [11] 19.045 15.266 17.425 16.885  9.328  8.248  8.788 25.523 28.762 27.682
## [21] 23.903 18.505 19.045 16.885 16.885 27.142 26.062 29.301 20.124 22.284
## [31] 17.965 22.284
```

We can plot data and fitted values as:

```
qplot(x, y) + geom_line(aes(x, y.hat))
```

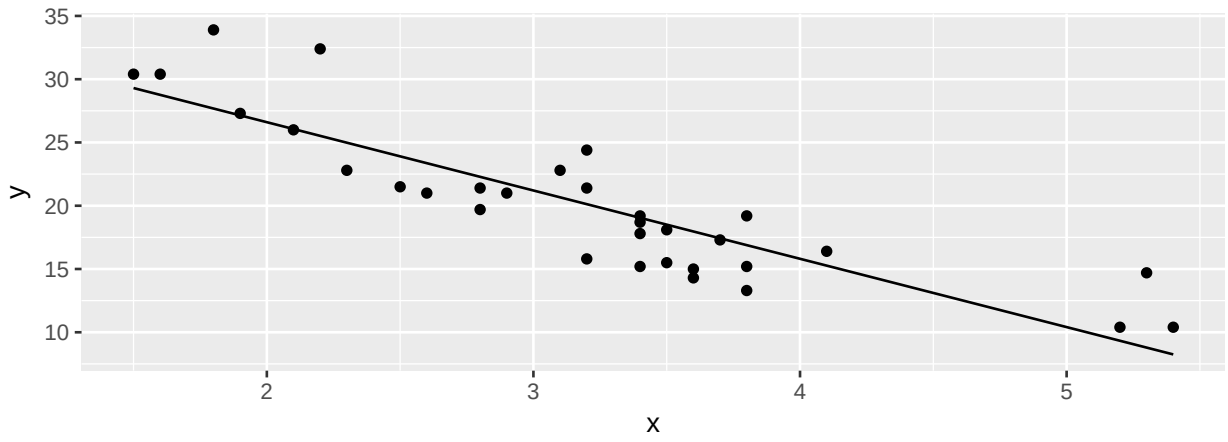


Figure 4.2: Correlation and regression are related topics.

Chapter 5

Basic concepts

5.1 Events and probabilities

A RANDOM EXPERIEMENT is the process of observing the outcome of a random phenomenon. The ELEMENTARY OUTCOMES are all possible outcomes of the random experiment. The SAMPLE SPACE is the set of all elementary outcomes.

If the random experiment is to observe the outcome of tossing a coin, the elementary outcomes are head (H) and tail (T), so the sample space in the set $\{H, T\}$.

If the random experiment is to observe the outcome of tossing a die, the sample space has six elements: $\{1, 2, 3, 4, 5, 6\}$. If the experiment is to observed the outcome of tossing two dice, the sample space has 36 elements: $\{(1,1), (2,1), \ldots, (6,1), (1,2), (2,2), \ldots, (6,6)\}$.

Imagine a random experiment with n elementary outcomes $O_1, \ldots, O_n$. To each outcome O_i we assign a PROBABILITY which is a measure of how likely it is that the outcome will happen. We write the probability as $P(O_i)$, where $0 \leq P(O_i) \leq 1$. For example, if we toss a FAIR COIN then head and tail are equally likely, and we assign the probabilities

$$P(H) = P(T) = 0.5$$

If O_i can not happen, then $P(O_i) = 0$; if O_i always happens then $P(O_i) = 1$. Notice that the probability of the sample space must be 1: If we do the experiment, then one of the elementary outcomes must happen.

If a die is fair, then the probability of each elementary outcome must be $1/6$. If we throw two fair dice, then the probability of each elementary outcome must be $1/36$.

An EVENT is a set of elementary outcomes. The probability of an event is the sum of the probabilities of the elementary outcomes that consitutes the experiment. Suppose we throw two dice, a black and a white:

Events can be combined to form new events using logical operions `AND` , `OR` and `NOT` . If E and F are events then we can combine these to form new events

Consider the events in Table 5.1. The event C `OR` D is the event that at least one die shows 1. The event C `AND` D is the event that both dice shows 1.

The addition rule for events is:

$$P(E \texttt{ OR } F) = P(E) + P(F) - P(E \texttt{ AND } F) \tag{5.1}$$

Description	Elementary outcomes	Probability
A: dice add to 3	{(1,2),(2,1)}	$P(A) = 2/36$
B: dice add to 6	{(1,5),(2,4),(3,3),(4,2),(5,1)}	$P(B) = 5/36$
C: white die shows 1	{(1,1), (1,2), ..., (1,6)}	$P(C) = 6/36$
D: black die shows 1	{(1,1), (2,1), ..., (6,1)}	$P(D) = 6/36$

Table 5.1: Events, elementary outcomes and probabilities.

New event	Description
E AND F:	The events E and F both occur.
E OR F:	The event E or the event F (or both) occur.
NOT E:	The event E does not occur.

Table 5.2: New events created from existing events.

Just adding $P(E) + P(F)$ means that the probability of the elementary outcomes shared by E and F are counted twice and this probability $P(E \text{ AND } F)$ must therefore be subtracted. Consequently we also have $P(E \text{ AND } F) = P(E) + P(F) - P(E \text{ OR } F)$.

Consider again the events in Table 5.1. The event C OR D corresponds to 11 elementary outcomes so $P(C \text{ OR } D) = 11/36$. Likewise, $P(C \text{ AND } D) = 1/36$. This confirms the formula (5.1):

$$P(C \text{ OR } D) = P(C) + P(D) - P(C \text{ AND } D) = 6/36 + 6/36 - 1/36 = 11/36$$

If two events E and F have nothing in common we say that they are MUTUALLY EXCLUSIVE. In this case $P(E \text{ AND } F) = 0$ and hence $P(E \text{ OR } F) = P(E) + P(F)$. Finally we have

$$P(\text{ NOT } E) = 1 - P(E)$$

so $P(E) = 1 - P(\text{ NOT } E)$. This is useful if $P(\text{ NOT } E)$ is easier to compute than $P(E)$.

Next we introduce CONDITIONAL PROBABILITY: Consider the event A in Table 5.1: The sum of the two dice is 3, where $P(A) = 2/36$. Suppose we first throw the white die and then the black die. Suppose the white die shows 1 (that is event C has occured). The probability that A occurs given that C has occured is written $P(A|C)$ and is called the conditional probability of A given C. The answer lies in the reduced sample space defined by that C has occured: $\{(1,1),(1,2),\ldots,(1,6)\}$. Only one elementary outcome, $(1,2)$ sums to 3 so the conditional probability is $1/6$. In general, for two events E and F we calculate the conditional probability as

$$P(E|F) = \frac{P(E \text{ AND } F)}{P(F)} \tag{5.2}$$

Formula 5.2 is also known as BAYES' FORMULA. From this we notice that $P(E|E) = 1$ and that if E and F are mutually exclusive then $P(E|F) = 0$. Notice that (5.2) can be rewritten as

$$P(E \text{ AND } F) = P(E|F)P(F)$$

and notice also that $P(E|F)P(F) = P(F|E)P(E)$.

Two events E and F are INDEPENDENT if occurence of one has no influence on the probility of the occurence of the other. This is the same as saying that $P(E|F) = P(E)$ (and that $P(F|E) = P(F)$). In this case we get

$$P(E \text{ AND } F) = P(E)P(F)$$

Returning to Table 5.1, it is straight forward to verify that $P(C|D) = P(C)$ and that $P(A|C) \neq P(A)$, so C and D are independent, but C and A are not.

5.2 Where do probabilities come from

Consider tossing a fair die and recording the number of dots facing up. We would on SYMMETRY grounds expect that the probability of observing i dots is $1/6$. That is, we say that probability of each elementary event is 1 divided by the number of elementary events, because we assume all the elementary events have the same probability.

But what about tossing a pin? In this case there is no obvious physical consideration that dictates which number p to assign to $P(U)$ where U is the event that the arrow points up. Here we can take a RELATIVE FREQUENCY approach. Toss the pin N times and let $N(U)$ be the number of times we see U. The relative frequency $N(U)/N$ is then an ESTIMATE of the p. Then we can imagine letting N go to infinity so that in the long run the relative frequency will converge to $P(U)$.

But what if the question was: "What is the probability that Real Madrid will win the Champions League"? Here, repeated sampling is not an option. Instead we can work with SUBJECTIVE probability which is a persons degree of belief in that the event takes place. Most humans have difficulty in assigning numbers to such an event. Is the probability 0.4 or 0.48 or something else? Often we have an intuitive feeling for when two events are equally likely. Imagine a wheel of fortune where $1/3$ of the area is gray and the rest is white. If I feel that the probability of Real Madrid winning is the same as the probability of landing in the gray area is the same, then $1/3$ is my subject probability that Real Madrid will win. If I feel the probability that Real Madrid is larger than $1/3$, then I can make the gray area larger until I feel that the probabilities are the same.

5.3 Example: Diagnostic testing, Bayes theorem and false positives

In a population, there is a rare disease which have infected 1 in every 100000 persons. The only way of telling if a person has the disease is by making a certain test based on a blood sample. The test is good but not perfect: It gets the right answer 99% of the time.

We introduce this terminology:

- D^1: A person has the disease (and D^0: a person does not have the disease).
- T^1: The test result positive (and T^0: a person tests negative).

Somewhat counter intuitive, a POSITIVE TEST RESULT indicates the presence of the disease, but that terminology is common. Imagine picking a random person from the population and conducting the test. The elementary outcomes are $\Omega = \{D^1T^1, D^0T^1, D^1T^0, D^0T^0\}$. An additional event is $\{D^1T^1, D^1T^0\}$ which we shall simply call D^1 and describe as "the event that a random person has the disease". Likewise $\{D^1T^1, D^0T^1\}$ which we shall call T^1 and describe as "the event that a random person tests positive"; see Tab. 5.3.

	T^1	T^0
D^1	D^1T^1	D^1T^0
D^0	D^0T^1	D^0T^0

Table 5.3: Basic events and combinations of these

The disease PREVALENCE is the probability $P(D^1)$ of a disease being present in a random individual, i.e. $P(D) = 1/100000 = 0.00001$. The SENSITIVITY of a test is the $P(T^1|D^1)$, i.e. the probability of

a positive test result given that the disease is present. The SPECICIFICITY of a test is the $P(T^0|D^0)$, i.e. the probability of a negative test result given that the disease is absent. Both sensitivity and specicificity is 0.99 in this example.

Given that a person tests positive what is then the probability that the person has the disease; that is, what is $P(D^1|T^1)$? Bayes formula can be written as

$$P(D^1|T^1) = \frac{P(D^1 \text{ AND } T^1)}{P(T^1)} = \frac{P(T^1|D^1)P(D^1)}{P(T^1)} \tag{5.3}$$

We know $P(T^1|D^1) = 0.99$ and $P(D^1) = 0.00001$ so all we need is $P(T^1)$ i.e. the probability of a positive test. To derive $P(T^1)$ we argue as follows: Since $T^1 = \{D^1T^1, D^0T^0\}$ and the events D^1T^1 and D^1T^0 are MUTUALLY EXCLUSIVE we have $P(T^1) = P(D^1T^1) + P(D^0T^1)$. The terms on the right hand side can again be found from rewriting Bayes' formula (see (5.2)):
$P(D^1T^1) = P(T^1|D^1)P(D^1)$ and $P(D^0T^1) = P(T^1|D^0)(1 - P(D^1))$.

In a similar way we find $P(D^1|T^0)$, the probability of having the disease if the test result is negative, as

$$P(D^1|T^0) = \frac{P(T^0|D^1)P(D^1)}{P(T^0)} = \frac{(1 - P(T^1|D^1))P(D^1)}{1 - P(T^1)}$$

```
p.D1 <- 0.00001  # P(D1)
p.T1_D1 <- 0.99  # P(T1 | D1)
p.T0_D0 <- 0.99  # P(T0 | D0)

p.T1 <- p.D1 * p.T1_D1 + (1 - p.D1) * (1 - p.T0_D0)
p.T1             # P(T1) - probability that a random person gets a positive test result
## [1] 0.01001
p.D1_T1 <- p.T1_D1 * p.D1 / p.T1 # P(D1 | T1)
p.D1_T1
## [1] 0.000989
p.D0_T1 <- 1 - p.D1_T1           # P(not D | T) - false positive rate (FPR)
p.D0_T1
## [1] 0.999
```

5.4 Random variables

5.5 Random variables – discrete

A RANDOM VARIABLE is the NUMERICAL OUTCOME of a RANDOM EXPERIMENT. It is customary to write a random variable with a capital letter, e.g. as X.

Example: draw a student at random from class. The students height, weight, family income and math grade are all numerical quantities describing properties of the randomly selected student; they are all random variables.

Example: toss two coins. The elementary outcomes are $\{(T,T),(H,T),(T,H),(H,H)\}$. Record the number of heads; there can be 0, 1 and 2. Think of a random variable as a rule which translates an elementary outcome into a number.

Word about notation: X denotes a random varible. In contrast x represents an outcome or REALIZATION of X. If head comes up twice, then $x = 2$.

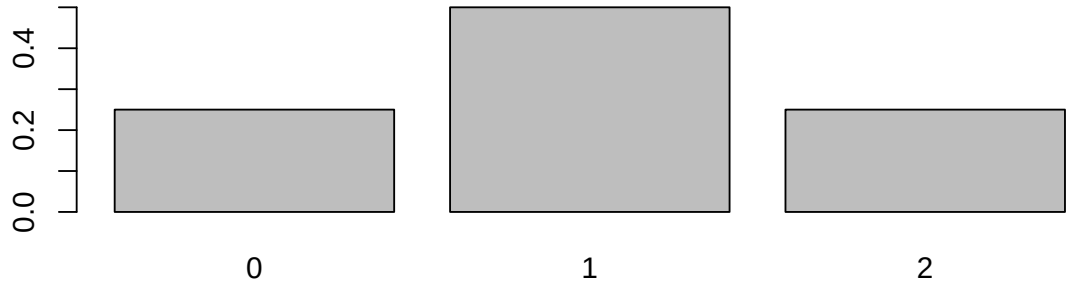
We often write $p_X(x)$ for the probability $P(X = x)$; we say that $p_X(x)$ is the DENSITY for the random variable X. The probabilities for the outcomes are

x	$p_X(x)$
0	1/4
1	1/2
2	1/4

Table 5.4: Tossing two coins and recording number of heads.

The density can be displayed as:

```
dens <- c(.25, .50, .25)
barplot(dens, names.arg=0:2)
```




```
x.rep <- sample(0:2, size=1000, replace=T, prob=c(.25, .5, .25))
table(x.rep)
## x.rep
##   0   1   2
## 248 512 240
```

5.6 Mean and variance

The MEAN or EXPECTATION of the random variable X is

$$\mathbb{E}(X) = \mu = \sum_{\text{all } x} x p_X(x)$$

The VARIANCE of the random variable X is

$$\mathbb{V}\text{ar}(X) = \sigma^2 = \sum_{\text{all } x} (x - \mu)^2 p_X(x)$$

5.7 Random variables – continuous

Figure 5.1: Darts in a dart board. What is the probability of hitting a disc with radius d on the board?

Darts is a throwing game where darts are thrown at a circular target (dartboard) fixed to a wall. Let the dartbord have radius r (the official size is $r = 17cm$). Then the area of the dartboard is $A = \pi r^2$. The outcome of an experiment is the point where a dart hits. There are infinitely many points on the dartboard so the set of elementary outcomes is has infinitely many elements.

A person who is not very good at the game plays: He is "equally likely" to hit all points on the dartboard. What does this mean? Take two non–overlapping discs D_1 and D_2 on the dartboard centered at points c_1 and c_2 and each with radius d and hence with area $S = \pi d^2$.

That he is equally likely to hit all points must mean that the probability of landing in D_1 is the same as the probability of landing in D_2. Hitting disc D_i is the event R_i and we have that

$$Pr(R_1) = Pr(R_2) = S/A = d^2/r^2.$$

Now consider what happens as the radius d of the discs D_1 and D_2 becomes smaller and smaller and goes to zero: At the end, the discs D_1 and D_2 will contain only their center points c_1 and c_2. But by the argument above we must then have

$$Pr(\{c_1\}) = Pr(\{c_2\}) = 0$$

So the probability of landing in a single point (which has no area) is zero but the probability of landing in an a region with area S is S/A.

Let X be a random variable which we describe as "the distance from the center of the dartboard to where the dart lands". The possible values of X are $[0, r]$ and there are infinitely many such values. We want to answer the question: What is $F_X(x) = P(X \leq x)$ for some value x (where $0 \leq x \leq r$) where F_X is the DISTRIBUTION FUNCTION. By the arguments above,

$$F_X(x) = Pr(X \leq x) = x^2/r^2$$

The DENSITY for the dart example is

$$f_X(x) = \begin{cases} 0 & \text{if } x < 0 \\ 2x/r^2 & \text{if } 0 \leq x < r \\ 0 & \text{if } x \geq r \end{cases}$$

The distribution and density functions are shown in Fig. 5.2. The distribution function and density function are closely connected: The probability $P(4 \leq X \leq 8)$ can be found from the distribution

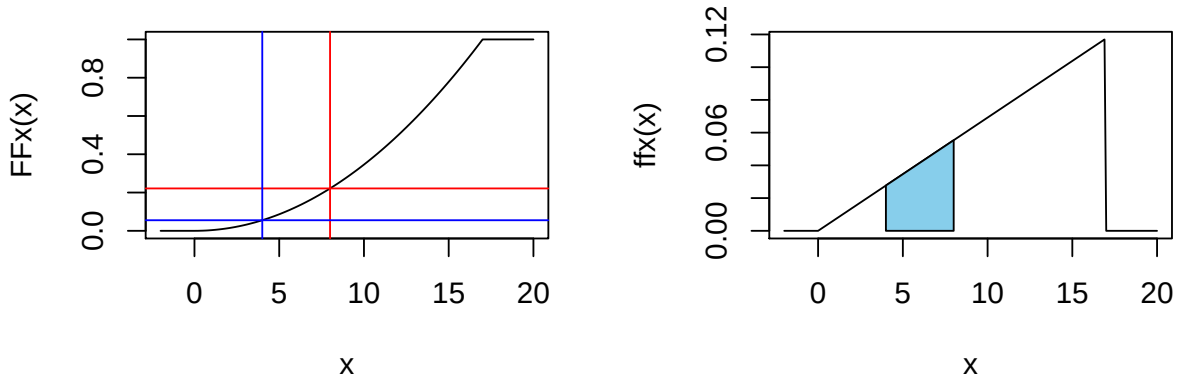


Figure 5.2: Left: Distribution function F_X. Right: Density function f_X. The dart board has radius 17.

function in Fig. 5.2 (left) as $P(4 \leq X \leq 8) = F_X(8) - F_X(4) = 0.2215 - 0.0554 = 0.1661$. The probability can also be found as the shaded area under the density function in Fig. 5.2 (right) as $P(4 \leq X \leq 8) = \int_4^8 f_X(x)dx$. But this also means that for a small number $h > 0$,

$$Pr(4 \leq X \leq 4 + h) = F_X(4 + h) - F_X(4) \approx f_X(4)h$$

5.8 Mean and variance

The MEAN or EXPECTATION of the random variable X is

$$\mathbb{E}(X) = \mu = \int_{-\infty}^{\infty} xp(x)dx$$

The VARIANCE of the random variable X is

$$\mathbb{V}\text{ar}(X) = \sigma^2 = \int_{-\infty}^{\infty} (x - \mu)^2 p(x)dx$$

5.9 Properties of mean and variance

Some properties of expectations:

- If X is equal to some constant c with probability 1 (that is, there is no uncertainty about the outcome of the experiment) then
$$\mathbb{E}(X) = c$$
- If a, b are constants then
$$\mathbb{E}(a + bX) = a + b\,\mathbb{E}(X)$$

Some properties of variance:

- If X is equal to some constant c with probability 1 (that is, there is no uncertainty about the outcome of the experiment) then
$$\mathbb{V}\mathrm{ar}(X) = 0$$
- If a, b are constants then
$$\mathbb{V}\mathrm{ar}(a + bX) = b^2\,\mathbb{V}\mathrm{ar}(X)$$

5.10 Random vector

Consider a fair 4–sided die, see Fig. 5.3.[1] The number rolled is indicated by the number shown upright at all three visible faces and can be with numbers $1, 2, 3, 4$.

Figure 5.3: Four sided dice. The number rolled is indicated by the number shown upright at all three visible faces.

A RANDOM VECTOR is a vector of random variables. We consider the experiment consisting of throwing two fair 4–sided dice. Define the random variables X and Y as follows:

- X: The absolute difference of the outcomes. The possible values of X are $0, 1, 2, 3$.
- Y: The sum of numbers facing down. The possible values of Y are $2, 3, \ldots, 8$

We say that (X, Y) is a 2–dimensional RANDOM VECTOR. We may now ask the question: what is $P(X = x, Y = y)$; for example what is $P(X = 2, Y = 4)$? The elementary outcomes are

$$\{(1,1), (2,1), (3,1), (4,1), (1,2), \ldots, (3,4), (4,4)\}$$

and each has probability $1/16$. From this we find that $P(X = x, Y = y)$ is given as in Table 5.5. The function $p_{XY}(x, y) = P(X = x, Y = y)$ is the JOINT DENSITY for (X, Y).

Now focus on X alone: What is $P(X = x)$? The answer is given by the MARGINAL DENSITY for X obtained by summing over the rows in Tab. 5.5:

$$\begin{aligned} p_X(x) = P(X = x) &= P(X = x, Y = 2) + P(X = x, Y = 3) + \cdots + P(X = x, Y = 8) \\ &= \sum_y P(X = x, Y = y) = \sum_y p_{XY}(x, y) \end{aligned}$$

	2	3	4	5	6	7	8
0	$\frac{1}{16}$		$\frac{1}{16}$		$\frac{1}{16}$		$\frac{1}{16}$
1		$\frac{2}{16}$		$\frac{2}{16}$		$\frac{2}{16}$	
2			$\frac{2}{16}$		$\frac{2}{16}$		
3				$\frac{2}{16}$			

Table 5.5: Joint pmf p_{XY}. Rows: X; Columns: Y.

	2	3	4	5	6	7	8	$p_X(x)$
0	$\frac{1}{16}$		$\frac{1}{16}$		$\frac{1}{16}$		$\frac{1}{16}$	$\frac{4}{16}$
1		$\frac{2}{16}$		$\frac{2}{16}$		$\frac{2}{16}$		$\frac{6}{16}$
2			$\frac{2}{16}$		$\frac{2}{16}$			$\frac{4}{16}$
3				$\frac{2}{16}$				$\frac{2}{16}$
$p_Y(y)$	$\frac{1}{16}$	$\frac{2}{16}$	$\frac{3}{16}$	$\frac{4}{16}$	$\frac{3}{16}$	$\frac{2}{16}$	$\frac{1}{16}$	

Table 5.6: Join pmf p_{XY} and marginals p_X and p_Y. Rows: X; Columns: Y.

These probilities are given in the margins of Table 5.6.

Suppose we are given the information that $Y = 5$. What is the probability that $X = 0$? that $X = 1$ and so on? If Y is 5 then the outcomes can be $(1, 4)(2, 3)(3, 2)(4, 1)$ so the absolute value of the difference between the outcomes must be either 1 or 3. These outcomes are equally likely so we concluded that each occur with probability $1/2$. More generally, the answer lies in the CONDITIONAL DENSITY of Y given X which is given as the following variant of BAYES FORMULA

$$p_{Y|X}(y|x) = \frac{p_{XY}(x, y)}{p_X(x)}$$

The conditional probabilities are

	$p_{X\|Y}(x\|y = 5)$	$p_X(x)$
0	$0/\frac{4}{16} = 0$	$\frac{4}{16}$
1	$\frac{2}{16}/\frac{4}{16} = \frac{1}{2}$	$\frac{6}{16}$
2	$0/\frac{4}{16} = 0$	$\frac{4}{16}$
3	$\frac{2}{16}/\frac{4}{16} = \frac{1}{2}$	$\frac{2}{16}$

Table 5.7: Left: Conditional density of X given that $Y = 5$. Right: Marginal density of X. Since these distributions are not the same we conclude that X and Y are not independent.

We see that $p_{X|Y}(x|y = 5)$ and $p_X(x)$ are different. This means that the information that $Y = 5$ provides some information about X. We say that X and Y are DEPENDENT RANDOM VARIABLES.

5.11 Mean vector and covariance matrix

We may calculate the expectation and variance of X and Y as before. An additional quantity is the COVARIANCE between X and Y defined as

$$\sigma_{XY} = \mathbb{C}\text{ov}(X, Y) = \mathbb{E}([X - \mu_X][Y - \mu_Y]) = \sum_{xy}(x - \mu_X)(y - \mu_Y)p_{XY}(x, y)$$

[1] FiXme Note: Just found 4-sided dies more amusing, but perhaps a 6-sided i less confusing

A related quantity is the CORRELATION coefficient between X and Y which is

$$\rho_{XY} = \frac{\sigma_{XY}}{\sqrt{\sigma_X^2 \sigma_Y^2}} = 0$$

For the specific example, the covariance and hence the correlation is 0. Notice that X and Y are dependent (information about X provides information about Y) but X and Y are uncorrelated. This phenomenon arises because the correlation coefficient is a measure of the degree of linear dependence between X and Y.

It is customary to collect these terms into a MEAN VECTOR and a COVARIANCE MATRIX:

$$\begin{bmatrix} \mathbb{E}(X) \\ \mathbb{E}(Y) \end{bmatrix}, \begin{bmatrix} \sigma_X^2 & \sigma_{XY} \\ \sigma_{YX} & \sigma_Y^2 \end{bmatrix}$$

5.12 Independent random variables

Let Z be the result from throwing the first dice and U the result of throwing the second dice. The elementary outcomes are $\Omega = \{(1,1),(2,1),(3,1),(4,1),(1,2),\ldots,(3,4),(4,4)\}$ and each has probability $1/16$ so $p_{ZU}(x,y) = P(Z = x, U = y) = 1/16$.

It is straight forward to verify that $p_Z(x) = 1/4$ for $x = 1,2,3,4$ and similarly for $p_U(y)$. Consequently, $p_{ZU}(x,y) = p_Z(x)p_U(y)$. We say that two random variables Z and U are INDEPENDENT if $p_{ZU}(x,y) = p_Z(x)p_U(y)$. The intuitive interpretation is that information about the value of Z provides no information about the value of U. This intuition is supported by Bayes formula:

$$p_{U|Z}(y|x) = \frac{p_{ZU}(x,y)}{p_Z(x)} = \frac{p_Z(x)p_U(y)}{p_Z(x)} = p_U(y)$$

5.13 Two distributions

5.14 Binary trials and the binomial distribution

Consider the following experiment:

1. Toss as coin $N = 5$ times and
2. Count how often the coin lands head up. Denote this number by x.

Suppose the probability of head (H) is θ. Then the probability of tail (T) is $1 - \theta$. We assume the the $N = 5$ tosses are INDEPENDENT. Suppose that in a single experiment the tosses came out as

$$HTHTT$$

such that $x = 2$.

The probability of observing these data is

$$P(HTHTT) = \theta \cdot (1-\theta) \cdot \theta \cdot (1-\theta) \cdot (1-\theta)\theta^2(1-\theta)^3.$$

However, the probability of observing $HHTTT$ and $HTTTH$ is also $\theta^2(1-\theta)^3$.

Hence if all we are interested is the probability of observing 2 heads in 5 tosses we must count the number of ways in which we can obtain 2 heads.

The answer is 10 as the table shows:

Table 5.8: Possible ways in which we can obtain 2 heads in 5 tosses of a coin.

1	1	1	1	0	0	0	0	0	0
1	0	0	0	1	1	1	0	0	0
0	1	0	0	1	0	0	1	1	0
0	0	1	0	0	1	0	1	0	1
0	0	0	1	0	0	1	0	1	1

This number of 10 possibilities can be computed via

$$\binom{n}{y} = \frac{n!}{y!(n-y)!} = \binom{5}{2} = \frac{5 \cdot 4}{2 \cdot 1} = 10.$$

where e.g. $5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1$.

Note we say "5 factorial" for $5!$ and "n choose y" for $\binom{n}{y}$. By definition $0! = 1$.

```
factorial(5)
## [1] 120
choose(5, 2)
## [1] 10
```

Hence, the probability of observing x heads in N trials is given by the DENSITY FUNCTION:

$$P(X = x) = p_X(x) = \binom{N}{x}\theta^x(1-\theta)^{N-x}; \quad x = 0, 1, \ldots, N \tag{5.4}$$

A random variable X has a BINOMIAL DISTRIBUTION

$$X \sim Bin(N, \theta)$$

if the density for X has the form (5.4).

In Fig. 5.4 these probabilities are shown for different θs and different N.

Notice: We can not calculate these probabilities because θ is unknown. However we may be prepared to make the ASSUMPTION that the coin is FAIR such that $\theta = 1 - \theta = 1/2$.

Under this assumption we then find

$$p_X(2) = 10(1/2)^2(1/2)^{5-2} = 0.3125$$

i.e.

```
choose(5, 2) * .5^2 * .5^(5-2)
## [1] 0.3125
```

We may calculate $p_X(x)$ for $x = 0, 1, \ldots, 5$ as

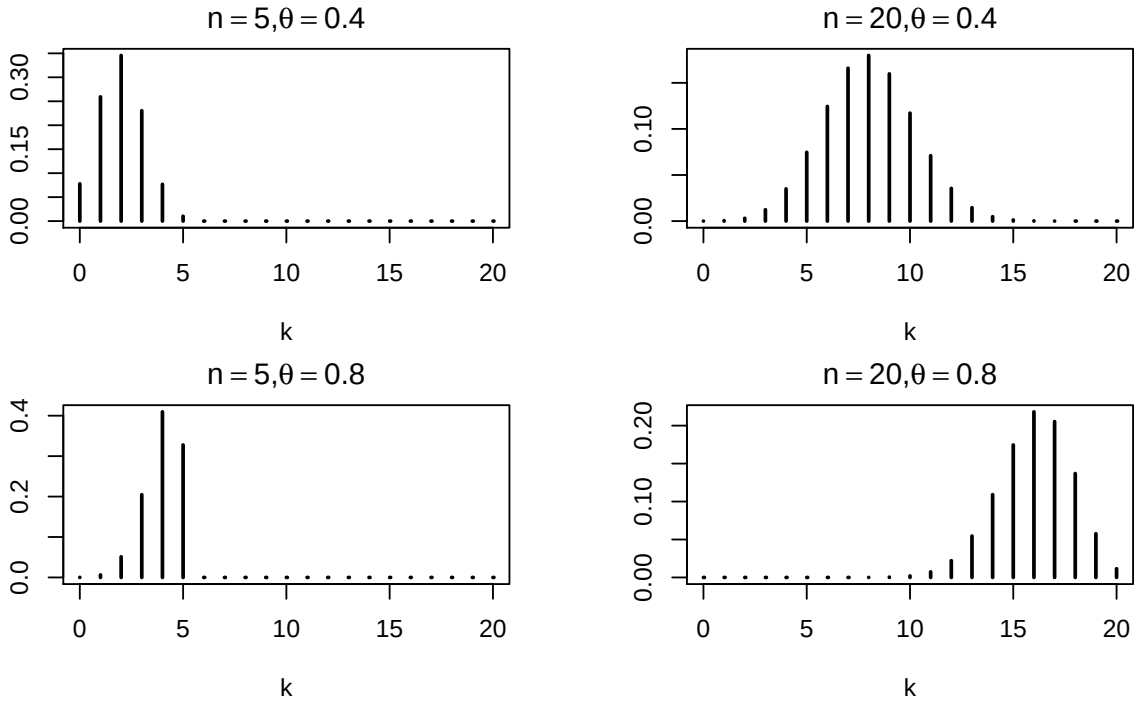


Figure 5.4: The probabilities $p_X(x)$.

```
dens <- dbinom(0:5, size=5, prob=0.5)
round(dens, 4)
## [1] 0.0312 0.1562 0.3125 0.3125 0.1562 0.0312
```

Notice the symmetry in the probabilities.

The probability of observing ≤ 3 heads in an experiment is

$$P(X \leq 3) = P(X = 1) + P(X = 1) + P(X = 2) + P(X = 3)$$

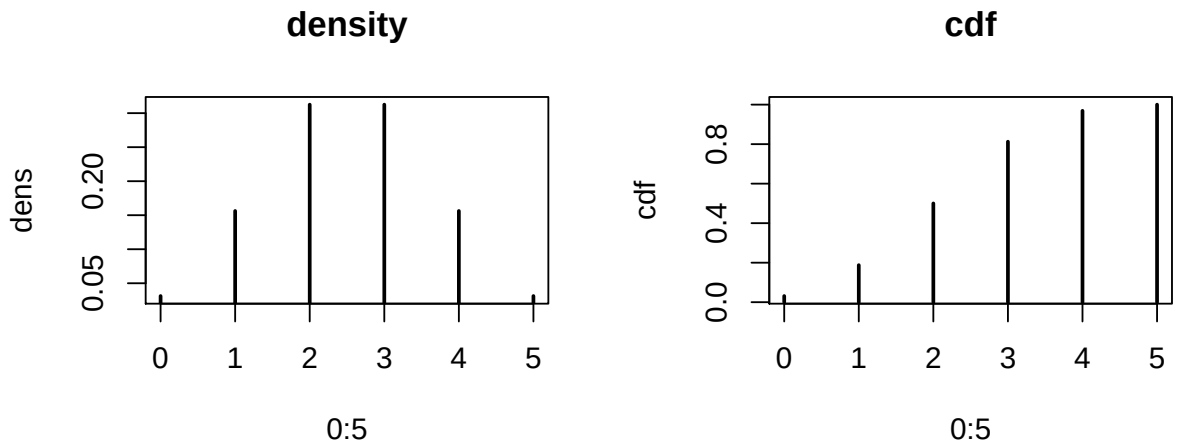
More generally,

$$F_X(x) = Pr(X \leq x) = \sum_{x'=0}^{x} Pr(X = x') = \sum_{x'=0}^{x} p(x')$$

where F_X is called the CUMULATIVE DISTRIBUTION FUNCTION (or CDF in short) We can find the cdf as

```
sum(dens[1:4])
## [1] 0.8125
pbinom(3, size=5, prob=.5)
## [1] 0.8125
cdf <- pbinom(0:5, size=5, prob=.5)
cdf
## [1] 0.03125 0.18750 0.50000 0.81250 0.96875 1.00000
```

```r
par(mfrow=c(1,2))
plot(0:5, dens, type="h", lwd=2); title("density")
plot(0:5, cdf, type="h", lwd=2); title("cdf")
```



It is illustrative to make simulation studies by repeating an experiment many times

```r
rbinom(1, size=5, prob=0.5) # One experiment
## [1] 1
rbinom(1, size=5, prob=0.5) # One experiment
## [1] 3
rbinom(1, size=5, prob=0.5) # One experiment
## [1] 2
```

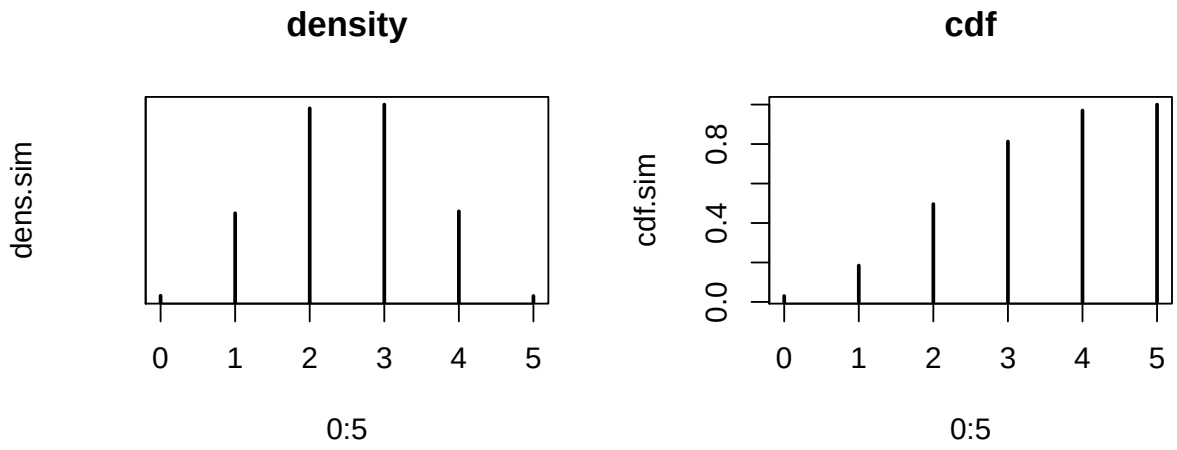
We apply the data generating mechanism (toss a coin 5 times and count the number of heads) many times to generate data $x^1, \ldots, x^M$ and then count the relative frequency of 2's Random samples from a binomial distribution are produced with `rbinom()`:

```r
set.seed(12345)
M <- 10000 # Replicates of experiment
x.rep <- replicate(M, rbinom(1, size=5, prob=0.5))
head( x.rep )
## [1] 3 4 3 4 2 1
table(x.rep)
## x.rep
##    0    1    2    3    4    5
##  304 1541 3114 3170 1570  301
## empirical density function
dens.sim <- table(x.rep)/M
round(dens.sim, 3)
## x.rep
##     0     1     2     3     4     5
## 0.030 0.154 0.311 0.317 0.157 0.030
## empirical cdf
cdf.sim <- cumsum(dens.sim)
round(cdf.sim, 3)
```

```
##     0     1     2     3     4     5
## 0.030 0.184 0.496 0.813 0.970 1.000
```

So $Pr(X = 2) = 0.311$ and $F(3) = Pr(X \leq 3) = 0.813$

```
par(mfrow=c(1,2))
plot(0:5, dens.sim, type="h", lwd=2); title("density")
plot(0:5, cdf.sim, type="h", lwd=2); title("cdf")
```



Lastly, suppose we want smallest value x' such that e.g. $P(X \leq x') = 0.6$

```
qbinom(p=0.6, size=5, prob=0.5)
## [1] 3
```

This value is called the 60% quantile.

5.15 Mean and variance

From data we can calculate quantities like SAMPLE MEAN, SAMPLE VARIANCE, SAMPLE STANDARD DEVIATION

$$\bar{x} = \frac{1}{M}\sum_i x_i, s_x^2 = \frac{1}{M-1}\sum_i (x_i - \bar{x})^2 s_x = \sqrt{s_x^2}$$

```
mean(x.rep)
## [1] 2.506
var(x.rep)
## [1] 1.235
sd(x.rep)
## [1] 1.111
```

Related to the random variable data generate data, there are theoretical counterparts:

The MEAN (or EXPECTATION) and VARIANCE of a random variable with possible outcomes $0, 1, \ldots, 5$ is

of a random variable $X \sim bin(N, \theta)$ is

$$\begin{aligned}
\mathbb{E}(X) &= 0P(X=0) + 1P(X=1) + \cdots + 5P(X=5) \\
&= \sum_{x=0}^{5} xP(X=x) \\
\mathbb{V}\text{ar}(X) &= (0-\mu)^2P(X=0) + \cdots + (5-\mu)^2P(X=5) \\
&= \sum_{x=0}^{5} (x-\mu)^2P(X=x)
\end{aligned}$$

For the binomial distribution $X \sim bin(N, \theta)$ it can be shown that

$$\mathbb{E}(X) = N\theta, \quad \mathbb{V}\text{ar}(X) = N\theta(1-\theta)$$

We have for the coin example (if $\theta = 1/2$)

```
N <- 5; theta <- .5
E <- N * theta; E                # Mean
## [1] 2.5
V <- N * theta * (1 - theta); V  # Variance
## [1] 1.25
S <- sqrt(V); S                  # Standard deviation
## [1] 1.118
```

5.16 The normal distribution – and why it is so normal

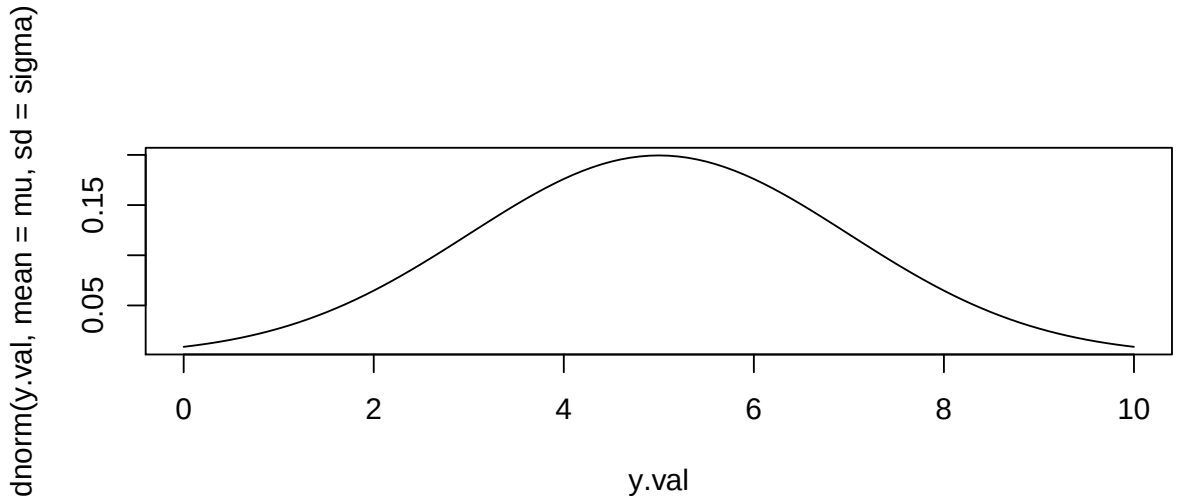
The density for a random variable with a normal distribution $Y \sim N(\mu, \sigma^2)$ has the form

$$f_Y(y) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{ -\frac{1}{2\sigma^2}(y-\mu)^2 \right\}$$

The mean of Y is $\mathbb{E}(Y) = \mu$ and the variance is $\mathbb{V}\text{ar}(Y) = \sigma^2$.

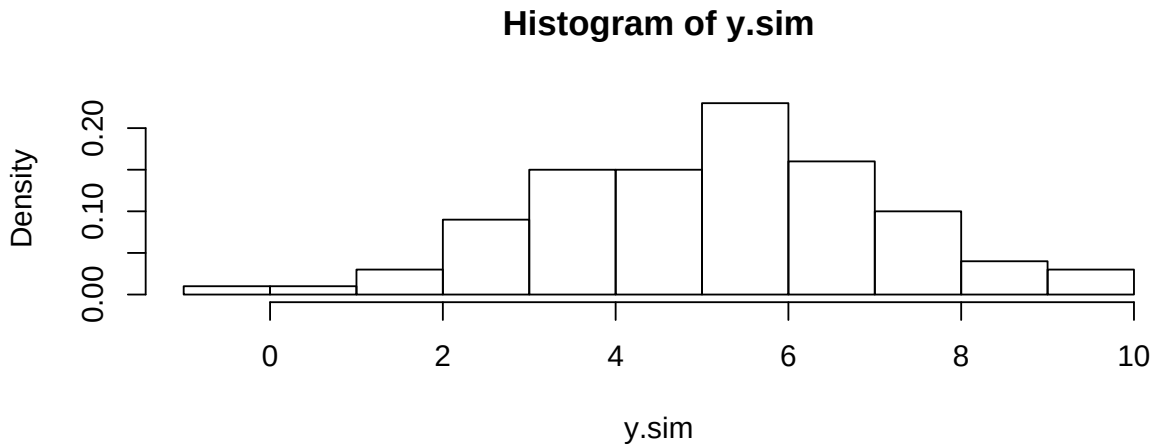
When $\mu = 0$ and $\sigma = 1$ we have the STANDARD NORMAL DISTRIBUTION.

```
mu <- 5
sigma <- 2
y.val <- seq(0, 10, by=0.1)
plot(y.val, dnorm(y.val, mean=mu, sd=sigma), type='l')
```



If we have many samples from such a distribution, we get a histogram:

```
N <- 100
y.sim <- rnorm(N, mean=mu, sd=sigma)
hist(y.sim, prob=T)
```



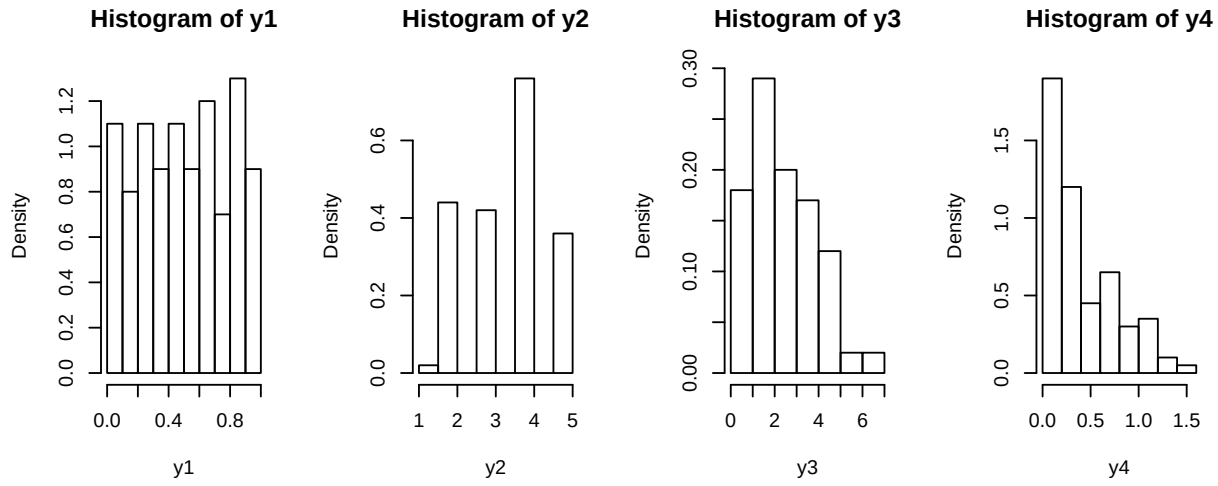
Why is the normal distribution so normal, and why does the normal distribution pop up in our world?

Partial answer: A variable which is the sum of many independent contributions follow approximately a normal distribution:

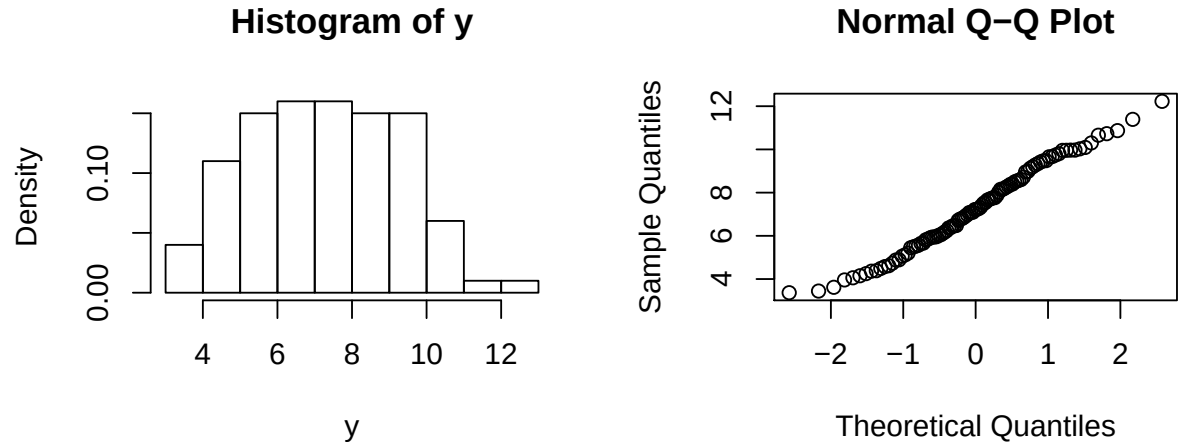
5.17 Example: Sum of four independent distribtions

```
par(mfrow=c(1, 4))
y1 <- runif(N);
y2 <- rbinom(N, size=5, prob=.7);
y3 <- rpois(N, lambda=3)
```

```
y4 <- rexp(N, rate=2)
hist(y1, prob=T)
hist(y2, prob=T)
hist(y3, prob=T)
hist(y4, prob=T)
```



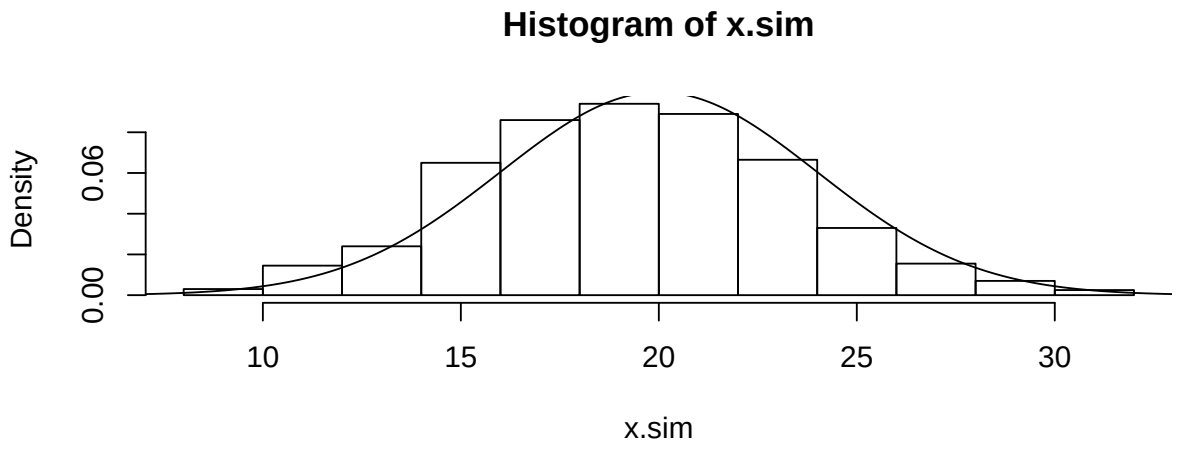

```
par(mfrow=c(1,2))
y <- y1 + y2 + y3 + y4
hist(y, prob=T)
qqnorm(y)
```



5.18 The binomial is approximately normal

If $X \sim bin(N, \theta)$ then X is approximately normal for large N

```r
N <- 100
theta <- .2
M <- 1000
x.sim <- rbinom(M, size=N, prob=theta)
hist(x.sim, prob=T)
y.val <- seq(0, 100, by=0.1)
lines(y.val, dnorm(y.val, mean=N*theta, sd=sqrt(N*theta*(1-theta))), type='l')
```



5.19 Example: Average waiting time in a que

On February 23 2017, a group of students were asked how long time (in minutes) they waited in line in last time they went to the canteen at AAUs Copenhagen campus: One student said he had waited for 20 minutes, but he said so with a smile so we doubt his report. The distribution of their WAITING TIMES is shown in Fig. 5.5.

```r
y <- c(2, 5, 1, 6, 1, 1, 1, 1, 3, 4, 1, 2, 1, 2, 2, 2, 4, 2, 2, 5, 20, 2, 1, 1, 1, 1)
y <- y[y < 10]
table(y)
## y
##  1  2  3  4  5  6
## 11  8  1  2  2  1
summary(y)
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1.00    1.00    2.00    2.16    2.00    6.00
hist(y)
```

Consider this question: Suppose the students go the the canteen every morning for a week. What is the distribution of their average waiting time? We only have data for one day, so we can not really tell, but we can mimic what data could have been for the other four days.[2] Fig. 5.6 shows the result.

[2] FiXme Note: prob: Much better explanation needed

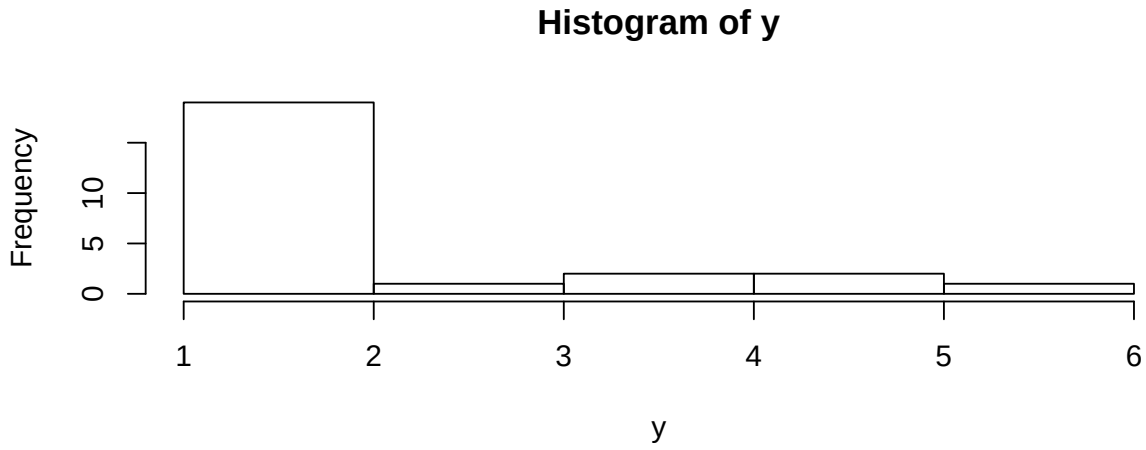


Figure 5.5: Waiting time at the canteen at AAUs Copenhagen campus.

```
y.mon <- y
y.tue <- sample(y)
y.wed <- sample(y)
y.thu <- sample(y)
y.fri <- sample(y)
y.avg <- (y.mon + y.tue + y.wed + y.thu + y.fri) / 5
y.avg
##  [1] 2.8 2.0 1.6 2.6 1.0 1.6 2.2 2.8 1.6 2.0 2.2 1.4 2.0 2.4 2.0 1.8 2.8 1.6 3.0
## [20] 2.2 3.8 2.4 1.6 2.8 1.8
```

```
par(mfrow=c(1,2))
hist(y.avg, prob=T)
curve(dnorm(x, mean=mean(y.avg), sd=sd(y.avg)), 0, 5, add=T)
qqnorm(y.avg)
```

5.20 Further topics

5.21 Parameter estimation

Consider the binomial distribtution. As θ is unknown we must ESTIMATE its value from data. With $N = 5$ and $x = 2$ intuition tells us to estimate θ as

$$\hat{\theta} = x/N = 2/5 = 0.4$$

Here we are really estimate the METHOD OF MOMENT: We equate the observed data with the expectation and solve for θ:

$$x = 2 = N\theta = 5\theta$$

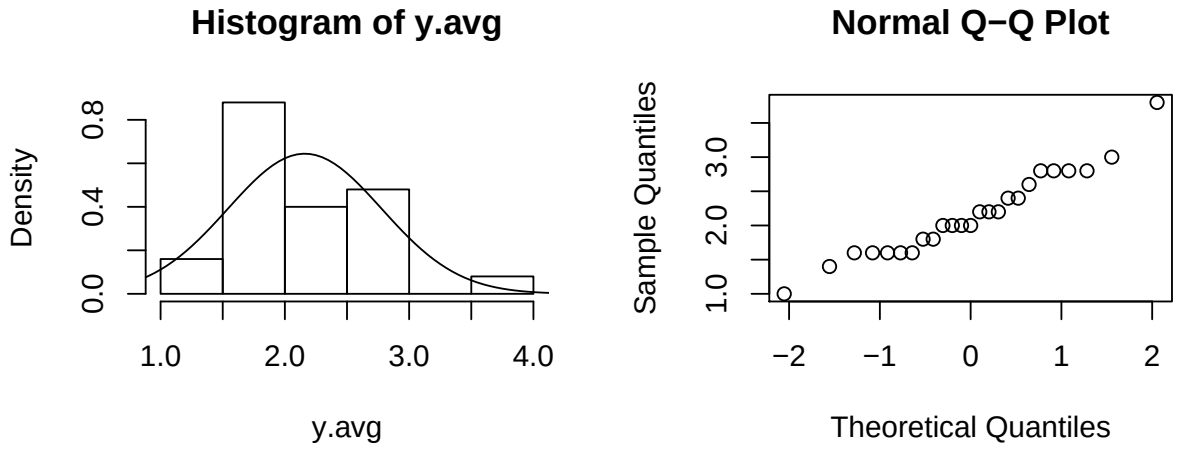


Figure 5.6: Average waiting times. Does the average approximately have the bell shaped normal curve.

Another method is the MAXIMUM LIKELIHOOD method: The density for the binomial variable is

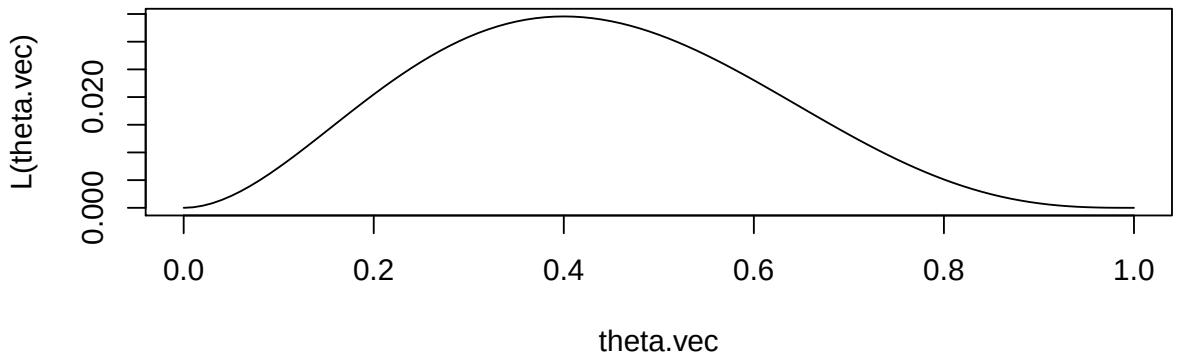
$$p(x) = \binom{N}{x} \theta^x (1-\theta)^{N-x}$$

This function is really also a function of θ. If x is fixed, we can think of the density as a function of θ; this function is called a LIKELIHOOD FUNCTION:

$$L(\theta) = \theta^x (1-\theta)^{N-x}$$

We ignore $\binom{N}{x}$ because this term is constant. We plot L against θ

```
N <- 5
x <- 2
L <- function(theta) {
    theta^x * (1 - theta)^{N-x}
    }
theta.vec <- seq(0, 1, by=0.01)
plot(theta.vec, L(theta.vec), type="l")
```



So for any value of θ, $L(\theta)$ gives the probability of seeing the data we have. This suggests to pick as estimate for θ the value that maximizes $L(\theta)$. It is easy to spot that we should take $\hat{\theta} = 0.4$.

The method of maximum likelihood is the preferred method of estimation for many reasons. One very practical reason is the following: Once we have formulated the model, i.e. the density function, then we have also defined the likelihood function. Parameter estimation then amounts to maximizing this function, and this is a purely mathematical task.

Sometimes this can be done with pen-and-paper (here); generally numerical optimization methods must be used, e.g. by `optimize()`:

```
optimize(L, lower=0, upper=1, maximum=T)
## $maximum
## [1] 0.4
##
## $objective
## [1] 0.03456
```

5.21.1 Estimate and estimator

Notice that $\hat{\theta}$ is a function of x so it would be more clear if we write $\hat{\theta}(x) = x/N$.

If $X \sim bin(N, \theta)$ we can think of $\hat{\theta}(X) = X/N$ as a function of a random variable. We say that $\hat{\theta}(x)$ is an ESTIMATE and that $\hat{\theta}(X)$ is an ESTIMATOR.

The expectation of $\hat{\theta}(X) = X/N$ is

$$\mathbb{E}(\hat{\theta}(X)) = \frac{1}{N} N\theta = \theta$$

The variance of $\hat{\theta}(X)$ is

$$\mathbb{V}\mathrm{ar}(\hat{\theta}(X)) = \frac{1}{N^2} N\theta(1-\theta) = \frac{1}{N}\theta(1-\theta)$$

Theoretical values

```
N
## [1] 5
theta
## [1] 0.2
theta * (1 - theta) / N
## [1] 0.032
```

Empirical values

```
head(x.rep)              # data from repeated experiments
## [1] 3 4 3 4 2 1
theta.hat <- x.rep / N  # corresponding estimates
head(theta.hat)
## [1] 0.6 0.8 0.6 0.8 0.4 0.2
## More observations are better
##
mean(theta.hat[1:5])
## [1] 0.64
var(theta.hat[1:5])
## [1] 0.028
##
mean(theta.hat[1:100])
## [1] 0.498
var(theta.hat[1:100])
## [1] 0.05212
##
mean(theta.hat)
## [1] 0.5013
var(theta.hat)
## [1] 0.04941
```

5.22 Transformations

If X is a random variable then any function of X is also a random variable.

Let us think in terms of data generating mechanisms (i.e. random variables):

1. Toss a fair coin $N = 10$ times,
2. record the number x of heads and
3. calculate $t = (x - 3)^2$.

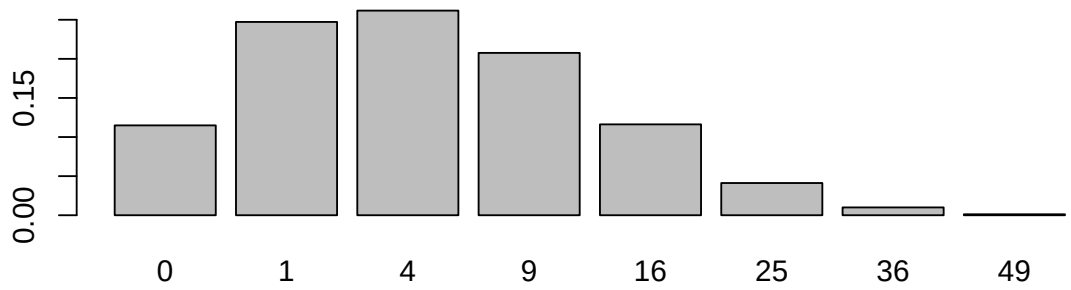
This data generating mechanism will be denoted by the random variable T.

Notice: Steps 1) and 2) above were the steps which defined the random variable X. So the random variable T is obtained by applying the function $t(x) = (x - 3)^2$ to the random variable X, that is

$$T = t(X)$$

It is relatively easy to calculate the probability mass function for T i.e. $Pr(T = t')$, but for now we shall determine the function by simulation.

```r
set.seed(12345)
M <- 10000 # Replicates of experiment
x.rep <- replicate(M, rbinom(1, size=10, prob=0.5))
head(x.rep, 10)
##  [1] 6 7 6 7 5 3 4 5 6 9
tt <- function(x){(x-3)^2}
t.rep <- tt(x.rep)
head(t.rep, 10)
##  [1]  9 16  9 16  4  0  1  4  9 36
dens <- table(t.rep) / M
dens
## t.rep
##      0      1      4      9     16     25     36     49
## 0.1149 0.2472 0.2617 0.2076 0.1162 0.0412 0.0100 0.0012
barplot(dens)
```



The possible values for T are

```r
t.val <- sort(unique(t.rep))
t.val
## [1]  0  1  4  9 16 25 36 49
```

Now we can calculate, e.g. $P(T \leq 15)$:

```r
t.val <= 15
## [1]  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE
sum(dens[t.val <= 15])
## [1] 0.8314
```

5.22.1 Mean and variance of linear transformations

The following results can be found in any elementary text book in statistics but we repeat them here.

Let X_1 and X_2 be random variables with expectation $\mathbb{E}(X_1) = \mu_1$ and $\mathbb{E}(X_2) = \mu_2$. Let a, b, c and d be numbers.

- Then
$$\mathbb{E}(a + bX_1 + c + dX_2) = a + b\,\mathbb{E}(X_1) + c + d\,\mathbb{E}(X_2)$$
and in particular
$$\mathbb{E}(a + bX) = a + b\,\mathbb{E}(X)$$

- Notice that $[X_1 - \mu_1]^2 = X_1^2 + \mu_1^2 - 2\mu_1 X_1$. Hence (why?)
$$\mathbb{V}\mathrm{ar}(X_1) = \mathbb{E}(X_1^2) - \mu_1^2$$

- If X is equal to some constant c with probability 1 (that is, there is no uncertainty about the outcome of the experiment) then
$$\mathbb{V}\mathrm{ar}(X) = 0$$

- If a, b are constants then
$$\mathbb{V}\mathrm{ar}(a + bX) = b^2\,\mathbb{V}\mathrm{ar}(X)$$

- If X_1 and X_2 are random variables then
$$\mathbb{V}\mathrm{ar}(X_1 + X_2) = \mathbb{V}\mathrm{ar}(X_1) + \mathbb{V}\mathrm{ar}(X_2) + 2\,\mathbb{C}\mathrm{ov}(X_1, X_2)$$

5.23 Hypothesis test

The fundamental idea in testing a hypothesis can be summarized as follows

- Unlikely events do not occur. Whether an event is unlikely or not depends on the model.
- If an unlikely event has occured then this is evidence against the model. The model might well be wrong.

There is a saying that if the map does not follow the landscape, then you should follow the landscape. In statistics, data is the landscape and the model is the map. If the model does "match" to data then the model is wrong.

5.24 Example: The boy/girl ratio

In a study it was found that the number of boys and girls born is a specific population was:

Boys	Girls	Total
6389	6135	12524
0.51	0.49	1

Is there a 50–50 chance for a boy and a girl?

Let x denote the number of boys. We shall assume that x is an outcome of a random variable X where

$$X \sim bin(N, \theta)$$

with $N = 12524$.

Our HYPOTHESIS is that $\theta = 0.5$. Based on data the estimate of θ is $\hat{\theta} = x/N = 0.51$.

5.25 General remarks

We have a hypothesis about the value of a population parameter θ; here formulated as a NULL HYPOTHESIS $H_0 : \theta = \theta_0 = 0.5$. The alternative to the null hypothesis is the ALTERNATIVE HYPOTHESIS $H_A : \theta \neq \theta_0$.

We do not know whether H_0 is true or H_A is true.

Based on data and a statistical model we need to decide what to believe in. We say we ACCEPT H_0 or REJECT H_0. However, to accept H_0 should really be formulated as NOT REJECT H_0.

	Accept H_0	Reject H_0
H_0 is true		
H_A is true		

Table 5.9: Hypothesis test

5.26 Back to the boy/girl ratio

The question is: Is 0.51 so "far from" 0.50 that we must conclude that having a boy is more likely than having a girl - or could it merely be due to randomness that we see 0.51 in this study even though the two genders are equally likely?

To test the hypothesis $\theta = 0.5$ we define a TEST STATISTIC which is a function of data.

We choose the test statistic

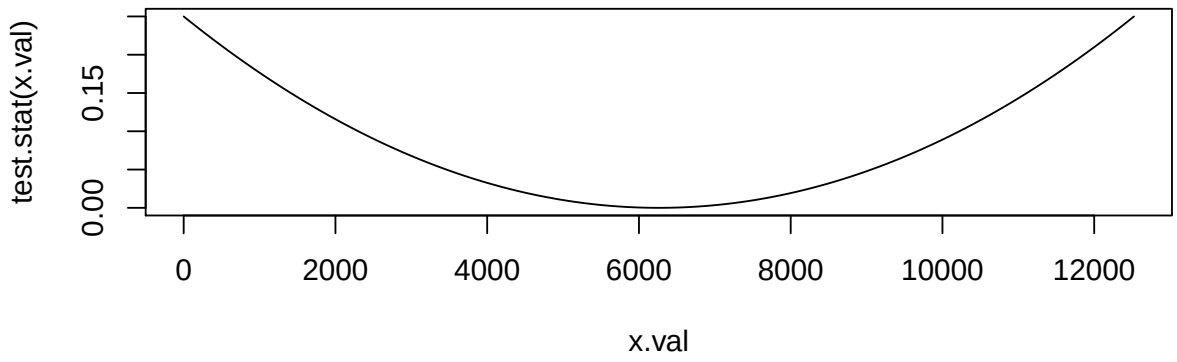
$$t(x) = \left(\frac{x}{N} - 0.5\right)^2$$

where 0.5 appears because this is the hypothesized value of θ. We could have chosen many other test statistics.

Notice:

- If x is close to $N/2$ then $t(x)$ is small (in particular $t(N/2) = 0$).
- If x is very large or very small then $t(x)$ is large. (In particular, $t(0) = t(N) = 0.25$ which is the largest possible value).

```
test.stat <- function(x) {(x/N - 0.5)^2}
N <- 12524
x.val <- 0:N
plot(x.val, test.stat(x.val), type="l")
```



Two things makes $t(x)$ a sensible test statistic:

- Values of $t(x)$ close to 0 is evidence for the null hypothesis and values of $t(x)$ far from 0 is evidence against the null hypothesis.
- Under the hypothesis we can calculate the distribution of $t(X)$.

With $x = 6389$ boys and $N = 12524$ we get

```
test.stat(6389)
## [1] 0.0001028
```

$$t_{obs} = t(6389) = (0.51 - 0.5)^2 = 0.0001$$

The question is then: Is 0.0001 "small" or "large"? That is, is 0.0001 an EXTREME or UNLIKELY value?

5.27 Evidence against the hypothesis

An answer to this question is provided by the following argument.

Let us assume that the hypothesis is true, i.e. that $\theta = 1/2$. What is then the probability of observing a value of $t(X)$ which is larger (or equal) to t_{obs}.

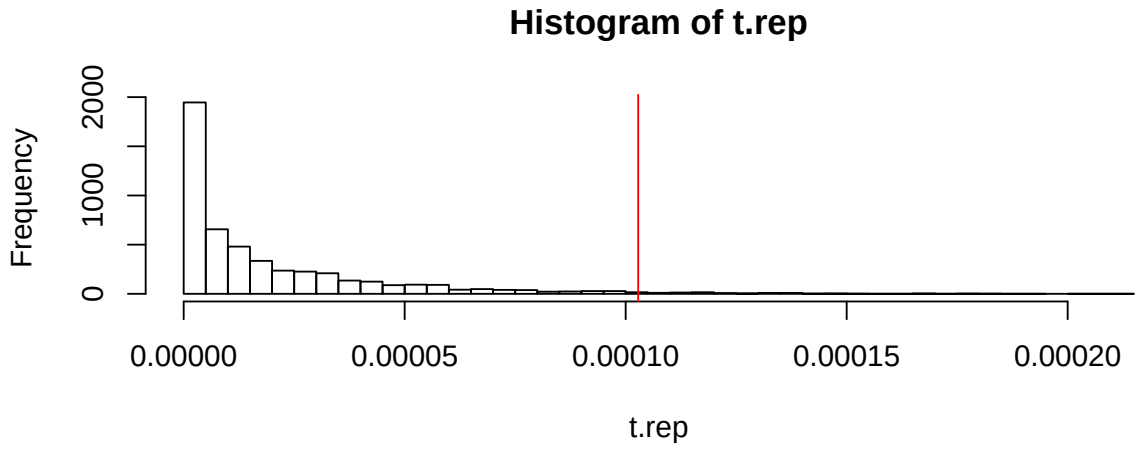
If this probability is e.g. 70% then t_{obs} is not an extreme value, but if this probability is e.g. 0.0001% then t_{obs} is an extreme value.

We can calculate this probability using computer simulations: We then ask the question: Suppose we redo the study say 5000 times. How often would we observe a value of $t(x)$ which is larger or equal to 0.0001?

```
set.seed(12345)
M <- 5000
n.boys  <- 6389
n.total <- 12524
p.hat   <- n.boys / n.total
t.obs   <- (p.hat - 0.5)^2; t.obs
## [1] 0.0001028
x.rep   <- replicate(M, rbinom(1, size=n.total, prob=0.5))
head(x.rep)
## [1] 6272 6315 6253 6260 6249 6380
t.rep  <- (x.rep / n.total - 0.5)^2
sum(t.rep >= t.obs) / M
## [1] 0.0214
```

So if the $\theta = 1/2$ we would see a test statistic larger than t_{obs} in about 2% of the cases.

```
hist(t.rep, nclass=50)
abline(v=t.obs, col='red')
```



5.28 p–value

What we have calculated above (using simulations) is the probability

$$p = Pr(T \geq t_{obs}) = 0.0214$$

This value is called a p–VALUE and we have found this values using MONTE CARLO SIMULATION.

Hence $p = 0.0214$ is the probability of observing a value of $t(X)$ which is at least as extreme as t_{obs}.

In R there are several ways to test hypotheses about binomial proportions. Two of these are `binom.test()` (the recommended) and `prop.test`:

```
binom.test(6389, 12524)
##
##  Exact binomial test
##
## data:  6389 and 12524
## number of successes = 6400, number of trials = 13000, p-value = 0.02
## alternative hypothesis: true probability of success is not equal to 0.5
## 95 percent confidence interval:
##  0.5013 0.5189
## sample estimates:
## probability of success
##                 0.5101
prop.test(6389, 12524)
##
##  1-sample proportions test with continuity correction
##
## data:  6389 out of 12524, null probability 0.5
## X-squared = 5.1, df = 1, p-value = 0.02
## alternative hypothesis: true p is not equal to 0.5
## 95 percent confidence interval:
##  0.5013 0.5189
## sample estimates:
```

```
##      p
## 0.5101
```

5.29 Interpretation of p–value

Let us return to the original question: Is the proportion of boys and girls the same?

A p–value can be regarded as a measure of evidence against the hypothesis. Here the p–value is small and there is evidence against our hypothesis.

But can we conclude from this that our hypothesis is false? That is, have we proven that $\theta \neq 1/2$?

The short answer is NO:

If the hypothesis is true, then the probability of observing 6389 boys in 12524 trials is 0.00054. This is a small probability, sure, but 6389 boys is a possible outcome under the hypothesis.

However, many studies confirm that more boys than girls are born!

Notice: Sometimes a p–value is erroneously interpreted as something like

> "the p–value the probability that the hypothesis is true."

This is a wrong interpretation. The hypothesis is either true or false (and we do not know which is the case because we do not have divine insight).

Hence there is no randomness involved and hence it does not make sense to talk about such a probability.

5.30 The effect of sample size

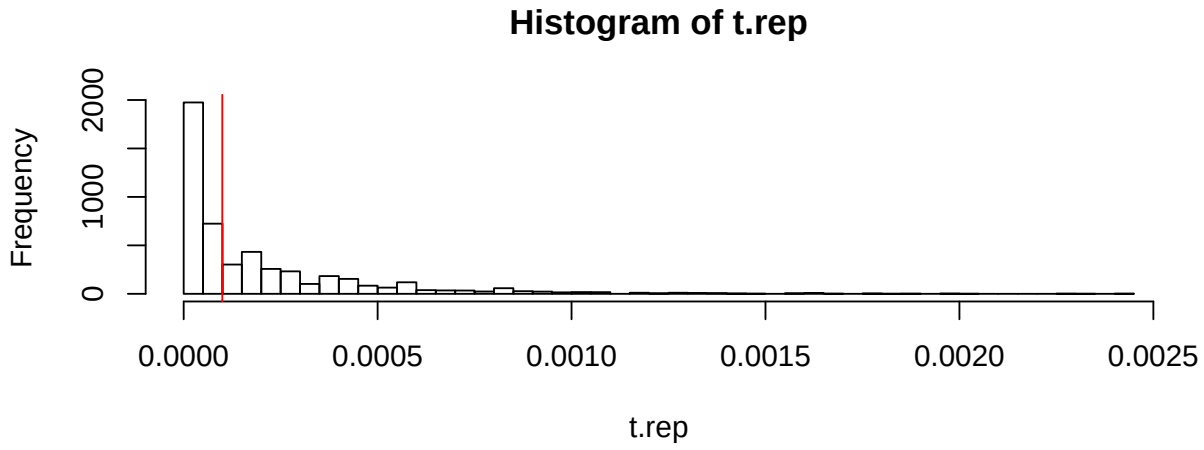
Suppose the result of study was:

Boys	Girls	Total
639	614	1253

(i.e. we have only 10% of the data). Then the fraction of boys is still 0.51.

Then we can redo the calculation of the p–value using computer simulations.

```
set.seed(12345)
par(mfrow=c(1,1))
n.boys <- 639
n.total <- 1253
p.hat <- n.boys / n.total
t.obs <- (p.hat - 0.5)^2
x.rep <-rbinom(n=5000, size=n.total, prob=0.5)
t.rep <- (x.rep / n.total - 0.5)^2
hist(t.rep, nclass=50)
abline(v=t.obs, col='red')
```




```
sum(t.rep >= t.obs) / 5000
## [1] 0.4948
```

The p–value is 0.49 and there is no evidence against the hypothesis $\theta = \frac{1}{2}$. Similarly:

```
binom.test(639, 1253)$p.value
## [1] 0.4978
```

5.31 What to make of this

With 12524 children there is strong evidence against the hypothesis $\theta = \frac{1}{2}$.

With 1253 children there is no evidence against the hypothesis.

In both cases the fraction of boys is 0.51 in both cases. What to make of this?

What we do in significance testing is that we make a hypothesis about "the true state of the world" and then we ask data to tell us if there is evidence against the hypothesis.

- If there is no evidence against the hypothesis then this might be because the hypothesis is true, or
- because the hypothesis is false but that there is not enough data (information) to provide the evidence.

Put more elegantly:

> Absence of evidence (of an effect) is NOT the same as evidence of absence (of an effect).

5.32 The shoes revisited

```
x <- c(13.2, 8.2, 10.9, 14.3, 10.7, 6.6, 9.5, 10.8, 8.8, 13.3)
y <- c(14, 8.8, 11.2, 14.2, 11.8, 6.4, 9.8, 11.3, 9.3, 13.6)
x
##  [1] 13.2  8.2 10.9 14.3 10.7  6.6  9.5 10.8  8.8 13.3
y
##  [1] 14.0  8.8 11.2 14.2 11.8  6.4  9.8 11.3  9.3 13.6
```

Suppose we want to test the hypothesis that shoes of type x have mean $\mu_x = 9$.

If $\mathbb{E}(X_i) = \mu_x$ then $\mathbb{E}(\bar{X}) = \mu_x$. This suggests to look at the difference between $\bar{x}$ and μ_x; if this difference is numerically large we doubt the hypothesis.

We have $\mathbb{E}(\bar{X} - \mu_x) = 0$ if the hypothesis is true.

```
mu.x <- 9
mean(x) - mu.x
## [1] 1.63
```

Is this a large or small difference?

The units of this difference is that same as the units of the measurements. We can make the difference as large or as small as we want it by changing the units (changing from minutes to nano seconds or to centuries).

We must somehow make the difference independent of the units.

If the variance of X_i is σ^2 then the variance of $\bar{X}$ is $\mathbb{V}\mathrm{ar}(\bar{X}) = \sigma^2/n$.

We must have:

$$\mathbb{V}\mathrm{ar}(\bar{X} - \mu_x) = \mathbb{V}\mathrm{ar}(\bar{X}) = \sigma^2/n$$

But this also means that

$$\mathbb{V}\mathrm{ar}\left(\frac{\bar{X} - \mu_x}{\sqrt{\sigma^2/n}}\right) = 1$$

If we estimate the variance σ^2 as s_x then s_x^2/n must be a good estimate for σ^2/N. The units of s_x^2 are "whatever squared" if the units of x_i are "whatever". So to get to the same units we take the square root: $\sqrt{s^2/n}$ has the same units as x_i. Hence

$$\frac{\bar{x} - \mu_x}{\sqrt{s^2/n}} = \sqrt{n}\frac{\bar{x} - \mu_x}{s}$$

has no units; it is independent of the scale on which the measurements are made.

```
z <- sqrt(length(x)) * (mean(x) - mu.x) / sd(x)
z
## [1] 2.103
```

Numerically large values of z are critical to the hypothesis; i.e. makes us doubt in the hypothesis.

Under the hypothesis,

$$z = \frac{\bar{x} - \mu_x}{\sqrt{s^2/n}}$$

should be evaluated in a t_{n-1}–distribution. Numerically large values of z causes us to doubt the hypothesis.

In practice, do a t-test:

```
p.value <- 2 * (1 - pt(abs(z), df=length(x) - 1))
p.value
## [1] 0.06483
```

Of course, there is a built in function that will do this:

```
t.test(x, mu=mu.x)
##
##  One Sample t-test
##
## data:  x
## t = 2.1, df = 9, p-value = 0.06
## alternative hypothesis: true mean is not equal to 9
## 95 percent confidence interval:
##   8.876 12.384
## sample estimates:
## mean of x
##     10.63
```

If n is not too small then z can also be evaluated in a standard normal distribution: In practice, if $|z| > 2$ we doubt the hypothesis.

Chapter 6

Linear models

Summary: Linear models are a super versatile tool and includes several classical analysis techiques such as linear regression, multiple regression, analysis-of-variance, and t test as special cases.

Linear models is the most commonly used class of statistical models. Linear models focus on relating a QUANTITATIVE RESPONSE variable to one or more EXPLANATORY VARIABLES. Many specific types of models that are known under other names in the litterature are really linear models. Examples of this includes LINEAR REGRESSION, MULTIPLE REGRESSION and POLYNOMIAL REGRESSION (Sec. 6.1.1), ANALYSIS OF VARIANCE or ANOVA, (Sec. 6.1.2) and ANALYSIS OF COVARIANCE or ANCOVA (Sec. 6.1.3).

The main difference between these models pertain to the nature of the explanatory variables, namely whether they are purely QUANTIATIVE (i.e. numerical), purely QUALITATIVE (i.e. categorical) or a combination of these. In the litterature, a LINEAR MODEL is also known as a GENERAL LINEAR MODEL [1] or a LINEAR NORMAL MODEL, see Sec. **??** for details.[2] [3]

6.1 What is a linear model (I)

In a linear model, the aim is to model how variation in a RESPONSE y varies with variation in, say, three PREDICTORS or EXPLANATORY VARIABLES or COVARIATES x_1, x_2, x_3. In a linear model, it is

[1] FiXme Note: MMA: Not be be confused with generalized linear model

[2] FiXme Note: linear: SH: linear: Why are LMs the most commonly used?

[3] FiXme Note: linear: SH: MMA: Interpretable, much is linear; also often a first order approx is reasonable; model validation...

Function	Purpose
`lm(y ~ x, data = ...)`	Fit a linear model (y is a quantitative response).
`predict(fit)`	Get the estimated response.
`residuals(fit)`	Raw residuals.
`residuals(fit, type = "pearson")`	Standardised residuals.

Table 6.1: Some important Rfunctions for linear models introduced in this chapter.

assumed that y is related linearly to the predictors as

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \epsilon$$

where ϵ is an error term. The terms $\beta_0, \ldots, \beta_3$ are unknown parameters which are to be estimated from data.

In a linear model the parameters enter linearly, but the predictors need not do so. For example is

$$y = \beta_0 + \beta_1 x_1^2 + \beta_2 \log x_2 + \beta_3 x_1 x_3 + \epsilon$$

a valid linear model. We can just think of x_1^2, $\log x_2$ and $x_1 x_3$ as predictors that can be calculated from data before parameter estimation is started and we are back in the original form. On the other hand $y = \beta_0 + \beta_1 x_1^{\beta_2} + \epsilon$ is not linear (as β_1 and β_2 does not have a linear relationship).

6.1.1 Linear, multiple and polynomial regression

Example 6.1.1 *[Potatoes]*

This dataset contains `weight` and two sizes of 20 potatoes. Weight in grams; size in milimeter. There are two sizes: `length` is the longest length and `width` is the shortest length across a potato. The potatoes originate from the third author's garden in 2015.

```
potatoes <- read.table("./data/potatoes.txt", header = TRUE)
head(potatoes, 4)
##   weight length width
## 1     22     38    29
## 2     41     46    34
## 3     24     40    31
## 4     16     33    28
```

```
multiplot(
  ggplot(potatoes, aes(length, weight)) + geom_point(),
  ggplot(potatoes, aes(width, weight)) + geom_point(),
  cols = 2)
```

□ □

We are interested in modelling how `weight` changes when `length` and `width` changes. We say that `weight` is the RESPONSE VARIABLE while `length` and `width` are EXPLANATORY VARIABLES. A commonly used name for this setting is MULTIPLE REGRESSION. If only one variable is included as explanatory, then the the setting is called LINEAR REGRESSION (or SIMPLE LINEAR REGRESSION).

First consider the simple case with one explanatory variable $x =$ `length` and the response variable is y = `weight`. Mathematically, we write the relationship between y and x as

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i. \tag{6.1}$$

The unknown parameters β_0 and β_1 (which we collect into a vector $\beta = (\beta_0, \beta_1)$) are estimated using the method of LEAST SQUARES, see Section **??**, using the `lm()` function:

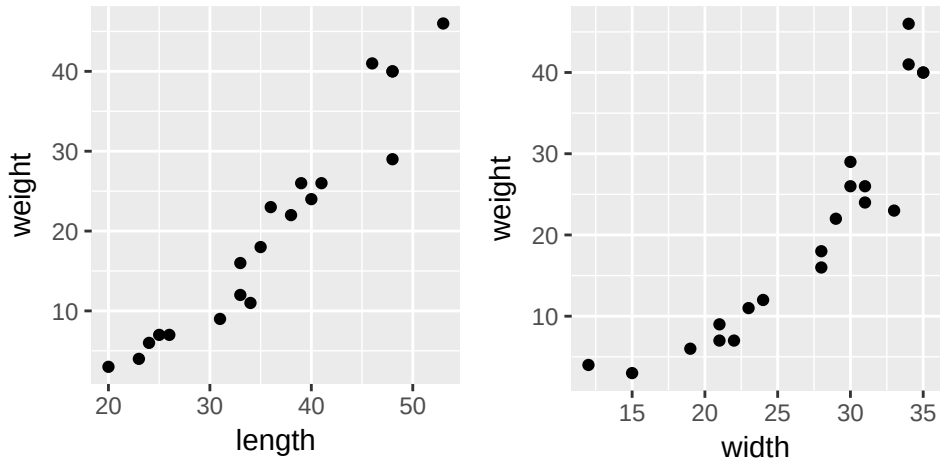


Figure 6.1: Plot of weight against length and against width.

```
pot1_l <- lm(weight ~ length, data = potatoes)
tidy(pot1_l)
##          term estimate std.error statistic   p.value
## 1 (Intercept)  -28.736   3.44249    -8.347 1.331e-07
## 2      length    1.366   0.09249    14.766 1.676e-11
```

The model is illustrated in Fig. 6.2 (left). Above, $\beta_0 + \beta_1 x_i$ models the *expected* value of the outcome for an individual with an observed x-value of x_i, and ϵ_i represents random noise and/or measurement error. We can define $m(x) = \beta_0 + \beta_1 x$. As such we can we can think of $m(x_i)$ as the SYSTEMATIC PART of the model, while ϵ_i denotes the RANDOM PART of the model. Note that $m(x)$ is a straight line with INTERCEPT β_0 and SLOPE β_1. The interpretation of the parameters is as follows: If $x = 0$ then $m(x) = \beta_0$ so β_0 must be the average level of y for subjects with $x = 0$ (which also means that the unit of β_0 is grams). The interpretation of β_1 is that when x increases by one unit (milimeter in the potatoes example) from some value x_0 to $x_0 + 1$ then the average value of y changes by $m(x_0 + 1) - m(x_0) = \beta_1$ units (so the unit of β_1 in the potatoes example is grams per milimeter), and this change is the same for all initial values x_0. Another view is that the natural interpretation of the parameters in $m(x)$ is via the DERIVATIVE which is β_1. Since the derivative is constant the effect of changing x by one unit is the same no matter what the initial value of x is.[4].

Estimated parameters are often denoted with a hat as $\hat{\beta} = (\hat{\beta}_0, \hat{\beta}_1)$ to distinguish them from the true but unknown values. Plugging the estimated values into $\beta_1 + \beta_2 x_i$ gives the points on the estimated regression lines; these are also denoted FITTED VALUES or PREDICTED VALUES.

The expression `weight ~ length` is an example of a MODEL FORMULA in R. See Section **??** for more about model formulae.

Next we take $x_1 =$ `length` and $x_2 =$ `width` as explanatory variables. Mathematically, we write the relationship between y, x_1 and x_2 as

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \epsilon_i, \tag{6.2}$$

Again, $m(x_1, x_2) = \beta_0 + \beta_1 x_1 + \beta_2 x_2$ models the *expected* value of the outcome (`weight`) for an

[4]FiXme Note: linear: derivatives must be handled somehow

individual (a potato) with `length` $= x_1$ and `width` $= x_2$. As before, ϵ_i represents random noise and/or measurement error such that y_i is centered around this expected value.

For any fixed value of x_1, $\beta_0 + \beta_1 x_1 + \beta_2 x_2$ describes a straight line with intercept $\beta_0 + \beta_1 x_1$ and slope β_2. Hence, if we vary both x_1 and x_2, $\beta_0 + \beta_1 x_1 + \beta_2 x_2$ will form a plane in a 3–dimensional space.

The model Eq. (6.2) is fitted in R as follows:

```
pot1_lw <- lm(weight ~ length + width, data = potatoes)
tidy(pot1_lw)
##          term estimate std.error statistic   p.value
## 1 (Intercept) -30.2047    3.5022    -8.624 1.293e-07
## 2      length   1.0923    0.2117     5.159 7.866e-05
## 3       width   0.4234    0.2968     1.427 1.718e-01
```

The interpretation is that increasing the length by 1 mm will on average increase the weight by $\hat{\beta}_1 = 1.09$ grams but only if the width (x_2) is unchanged. However, in practice, potatoes grow both in length and width, and therefore the interpretation of these parameters can be more subtle. We treat this topic in detail in Sec. 6.3, p. 97.

Judging from the graphs in Fig. 6.1, p. 87 one could guess that one should include also `width`^{2} as explanatory variable. This can be done with

```
pot1_lw2 <- lm(weight ~ length + width + I(width^2), data = potatoes)
tidy(pot1_lw2)
##          term estimate std.error statistic   p.value
## 1 (Intercept)  8.84001   7.26011     1.218 2.410e-01
## 2      length  0.83097   0.13488     6.161 1.369e-05
## 3       width -2.61804   0.56984    -4.594 2.992e-04
## 4  I(width^2)  0.06815   0.01213     5.617 3.860e-05
```

where the notation `I(width^2)` will be explained in Sec. 6.14, p. 118, and corresponds to the model

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i2}^2 + \epsilon_i. \tag{6.3}$$

Again, $m(x_1, x_2) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_2^2$ models the *expected* value of the outcome (`weight`) for an individual (a potato) with `length` $= x_1$ and `width` $= x_2$. The interpretation of β_1 is straight forward: The effect on `weight` of changing `length` by one unit when `width` is fixed. The interpretation of β_2 and β_3 is more subtle: Increasing x_2 by one unit means a change in the expected value of `weight` by $m(x_1, x_2 + 1) - m(x_1, x_2) = \beta_2 + 2\beta_3 x_2 + \beta_3 = \beta_2 + \beta_3(2x_2 + 1)$. This means that the change in the expected value of $x_2 =$ `width` to $x_2 + 1$ depends on the value of $x_2 =$ `width`. For example, if $x_2 =$ `width` $= 15$, then $m(x_1, x_2 + 1) - m(x_1, x_2) = \beta_2 + 31\beta_3$, and if $x_2 =$ `width` $= 30$, then $m(x_1, x_2 + 1) - m(x_1, x_2) = \beta_2 + 61\beta_3$.

The model just `width` as predictor and the model with both `width` and `width`^{2} as predictors are illustrated in Fig. 6.2 (right)
[5]

Yet another model is based on the following consideration: Weight is volume times density. If we approximate the shape of a potato by a cylinder of height h and diameter w then the volume is $V = h\pi(w/2)^2$. Taking logs (see Chap. **??**, p. **??**) and collecting terms gives $\log(V) = \beta_0 + \beta_1 \log(h) + 2\log(w)$. This suggests to fit the following model: [6]

[5] FiXme Note: Caption:
[6] FiXme Note: linear: use log_2 instead.

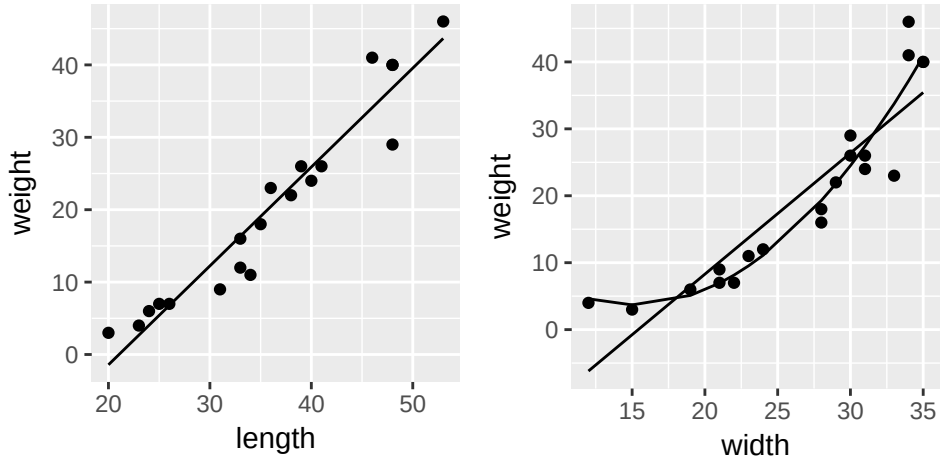


Figure 6.2: On the left hand side `lm(weight ~ width, data = potatoes)` is illustrated, and on the right hand side `lm(weight ~ width + I(width^2), data = potatoes)` is illustrated.

```
pot2_lw <- lm(log(weight) ~ log(length) + log(width), data = potatoes)
tidy(pot2_lw)
##          term estimate std.error statistic   p.value
## 1 (Intercept)   -7.255    0.3080   -23.554 2.034e-14
## 2 log(length)    1.858    0.2007     9.255 4.753e-08
## 3  log(width)    1.052    0.1895     5.549 3.535e-05
```

At this stage we shall make very simple comparison of the models: We calculate the squared correlation between the observed and the fitted / predicted values. Calculating the correlation may seem very natural, and the reason for squaring these correlations will be discussed in Sec. 6.4.

```
##   pot1_l   pot1_w  pot1_w2  pot1_lw pot1_lw2  pot2_lw
##   0.9237   0.8253   0.9227   0.9319   0.9771   0.9796
```

The model that fits best to data is the model where all variables are transformed to the log scale. [7]

6.1.2 Analysis of variance (ANOVA) models

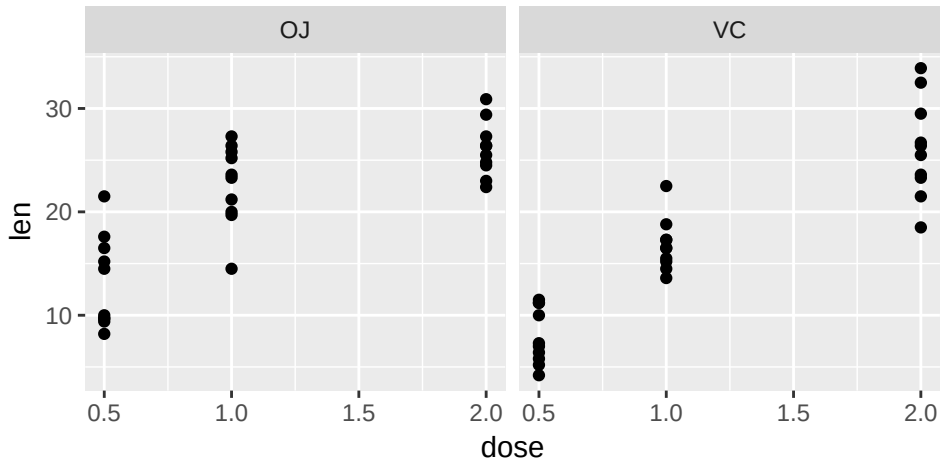
Example 6.1.2 *[ToothGrowth] The response is the length of odontoblasts (cells responsible for tooth growth) in 60 guinea pigs. Each animal received one of three dose levels of vitamin C (0.5, 1, and 2 mg/day) by one of two delivery methods: orange juice (coded as OJ) or ascorbic acid a form of vitamin C and (coded as VC).*

```
head(ToothGrowth, 4)
##    len supp dose
## 1  4.2   VC  0.5
## 2 11.5   VC  0.5
## 3  7.3   VC  0.5
## 4  5.8   VC  0.5
```

[7] FiXme Note: linear: more on interpretation. need to introduce $\mu(x)$ earlier so that we can talk about $\exp(\mu(x))$.

Such grouped data can be visualized in different ways:

```
ggplot(ToothGrowth, aes(dose, len)) + geom_point() + facet_grid(~ supp)
```



The average length in each of the $2 \times 3 = 6$ groups is:

```
toothInt <- ToothGrowth %>% group_by(dose, supp) %>% summarise(val = mean(len))
toothInt
## # A tibble: 6 x 3
## # Groups:   dose [?]
##    dose   supp   val
##   <dbl> <fctr> <dbl>
## 1   0.5     OJ 13.23
## 2   0.5     VC  7.98
## 3   1.0     OJ 22.70
## 4   1.0     VC 16.77
## 5   2.0     OJ 26.06
## 6   2.0     VC 26.14
```

An INTERACTION PLOT *is a graphical visualization of the group averages, see Fig. 6.3, where also variability in each group is displayed:*

```
ggplot(ToothGrowth, aes(x = factor(dose), y = len, colour = supp)) +
    geom_boxplot(outlier.shape = 4) +
    geom_point(data = toothInt, aes(y = val)) +
    geom_line(data = toothInt, aes(y = val, group = supp))
```

□ □

For simplicity, we first focus on data where dose is 0.5:

```
tg <- ToothGrowth %>% filter(dose == 0.5)
```

We can model these data using a REGRESSION MODEL

$$y_i = \beta_0 + \beta_1 x_{i1} + \epsilon_i,$$

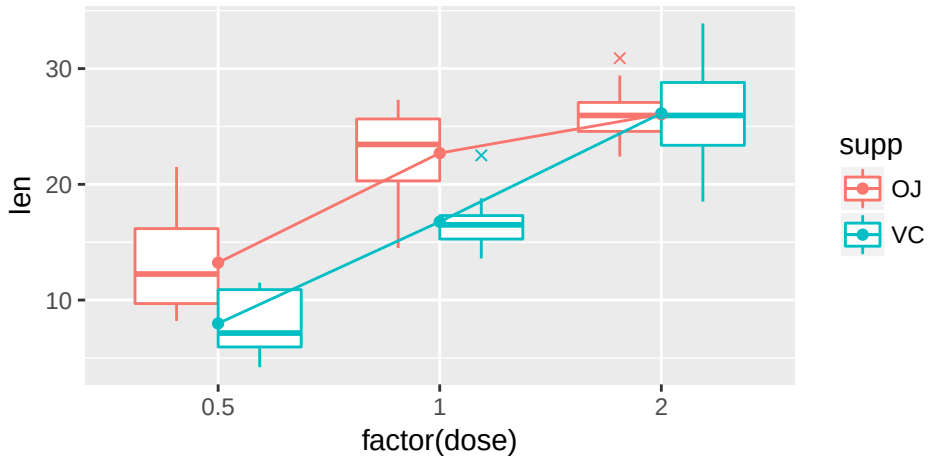


Figure 6.3: Interaction plot for the ToothGrowth data. The average length for each group is a dot. Boxplot outliers are crosses.

where the predictor variable, x_{i1}, is a DUMMY VARIABLE: Create a vector x_1 which is 0 in entries with `supp = OJ` and 1 in entries with `supp = VC`. So the effect of the dummy variable is to "switch" the effect of β_1 on and off.

```
lm(len ~ supp, data = tg)
##
## Call:
## lm(formula = len ~ supp, data = tg)
##
## Coefficients:
## (Intercept)       suppVC
##       13.23        -5.25
```

The `lm()` function will do this for us: Since `supp` is a FACTOR with two levels, `lm()` will know that `supp` must be turned into a dummy variable (had `supp` had tree levels, then `lm()` would have created two dummy variables).

The parameters in the model have a very simple interpretation: The intercept is simply the average value of `len` in the `OJ` group and the slope is the change in average when moving from `OJ` to `VC`:

```
tg %>% group_by(supp) %>% summarise(mean(len))
## # A tibble: 2 x 2
##     supp `mean(len)`
##   <fctr>       <dbl>
## 1     OJ       13.23
## 2     VC        7.98
```

Next consider the whole dataset. We can treat `dose` as a numerical variable or we can think of `dose` as a factor with three levels. We shall do the latter here:

```
ToothGrowth <- ToothGrowth %>%
    mutate(dosef = factor(dose,
                          levels = c(0.5, 1, 2),
                          labels = c("LO", "ME", "HI")))
```

The call

```
tooth2 <- lm(len ~ supp + dosef, data = ToothGrowth)
coef(tooth2)
## (Intercept)      suppVC     dosefME     dosefHI
##       12.45       -3.70        9.13       15.50
```

then corresponds to the model

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i3} + \epsilon_i,$$

where x_2 is 1 when `dosef = ME` and 0 otherwise and x_3 is 1 when `dose = HI` and 0 otherwise. The interpretation is straight forward: Think of a dummy variable as a way of switching a parameter on and off:

- `dose = LO` and `supp = OJ`: only β_0 is switched on so β_0 is the estimated mean.
- `dose = LO` and `supp = VC`: β_0 and β_1 are switched on so the estimated mean is $\beta_0 + \beta_1$.
- `dose = ME` and `supp = OJ`: β_0 and β_2 are switched on so the estimated mean is $\beta_0 + \beta_2$.
- `dose = ME` and `supp = VC`: β_0, β_1 and β_2 are switched on so the estimated mean is $\beta_0 + \beta_1 + \beta_2$.
- and so on...

The effect of the parameters on the mean value in each group is shown below:

	LO	ME	HI
OJ	β_0	$\beta_0 + \beta_2$	$\beta_0 + \beta_3$
VC	$\beta_0 + \beta_1$	$\beta_0 + \beta_1 + \beta_2$	$\beta_0 + \beta_1 + \beta_3$

Table 6.2: fixme...**FiXme Note: fixme**

Hence β_1 is the effect of going from `OJ` to `VC` and that effect is the same for any value of `dosef`. Likewise, β_2 is the effect of going from `LO` to `ME` and that effect is the same for any value of `supp` etc.

Because of this structure we say we have an ADDITIVE MODEL. Parallel profiles in Fig. 6.3 correspond to additivity of the effect of the factors. As seen, the lines are not parallel for the high dose, hence the additive model may not be suitable.

We can add the mean values obtained from `tooth2` as follows:

```
ToothGrowth %>%
  mutate(pred2 = fitted(tooth2)) %>%
  group_by(dosef, supp) %>%
  summarise(mean(len), mean(pred2), mean(len - pred2))
## # A tibble: 6 x 5
## # Groups:   dosef [?]
##    dosef   supp `mean(len)` `mean(pred2)` `mean(len - pred2)`
##   <fctr> <fctr>       <dbl>         <dbl>               <dbl>
## 1     LO     OJ       13.23        12.455               0.775
## 2     LO     VC        7.98         8.755              -0.775
```

```
## 3     ME     OJ       22.70        21.585              1.115
## 4     ME     VC       16.77        17.885             -1.115
## 5     HI     OJ       26.06        27.950             -1.890
## 6     HI     VC       26.14        24.250              1.890
```

An alternative to the additive model is then an INTERACTION MODEL

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i3} + \beta_4 x_{i1} x_{i2} + \beta_5 x_{i1} x_{i3} + \epsilon_i$$

where $x_{i1}x_{i2}$ is 1 when `supp = VC` and `dosef = ME`, and $x_{i1}x_{i3}$ is 1 when `supp = VC` and `dosef = HI`.

The model parameters are estimated in R as follows:

```
tooth3 <- lm(len ~ supp + dosef + supp * dosef, data = ToothGrowth)
coef(tooth3)
##    (Intercept)         suppVC        dosefME        dosefHI suppVC:dosefME
##          13.23          -5.25           9.47          12.83          -0.68
## suppVC:dosefHI
##           5.33
ToothGrowth <- ToothGrowth %>% mutate(pred3 = fitted(tooth3))
ToothGrowth %>%
  group_by(dosef, supp) %>%
  summarise(mean(len), mean(pred3))
## # A tibble: 6 x 4
## # Groups:   dosef [?]
##    dosef   supp `mean(len)` `mean(pred3)`
##   <fctr> <fctr>       <dbl>         <dbl>
## 1     LO     OJ       13.23         13.23
## 2     LO     VC        7.98          7.98
## 3     ME     OJ       22.70         22.70
## 4     ME     VC       16.77         16.77
## 5     HI     OJ       26.06         26.06
## 6     HI     VC       26.14         26.14
```

We see that the fitted values under the interaction model is the same as the group averages, and as such the interaction model represents no simplification.

8

6.1.3 Analysis of covariance (ANCOVA) models

The purpose of ANCOVA is (usually) to compare two or more linear regression lines. It is a way of comparing the y variable among groups while statistically controlling for variation in y caused by variation in the x variable.

Notice: We could have chosen to think of `dose` as a numeric variable in the `ToothGrowth` data so that we would have had one factor and one numerical variable, but since there are only three different doses it would be a little speculative.

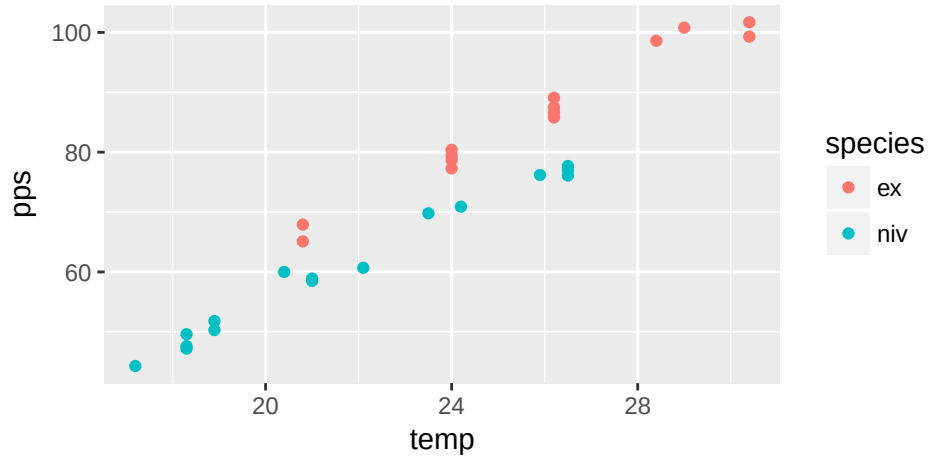
Example 6.1.3 *[cricks] Walker (1962)[9] studied the mating songs of male tree crickets. Each wingstroke by a cricket produces a pulse of song, and females may use the number of pulses per*

[8] FiXme Note: sh:linear: Need to compare the models; need to interpret the models.
[9] FiXme Note: reference til water

second to identify males of the correct species. Walker (1962) wanted to know whether the chirps of the crickets Oecanthus exclamationis *and* Oecanthus niveus *had different pulse rates. See* `http://www.biostathandbook.com/ancova.html` *for details. He measured the pulse rate of the crickets at a variety of temperatures:*

```
crick <- read.table("data/cricket.txt", header = TRUE)
summary(crick)
##  species       temp           pps
##  ex :14   Min.   :17.2   Min.   : 44.3
##  niv:16   1st Qu.:20.8   1st Qu.: 59.2
##           Median :24.0   Median : 76.2
##           Mean   :23.6   Mean   : 72.5
##           3rd Qu.:26.2   3rd Qu.: 84.5
##           Max.   :30.4   Max.   :101.7
head(crick, 4)
##   species temp  pps
## 1      ex 20.8 67.9
## 2      ex 20.8 65.1
## 3      ex 24.0 77.3
## 4      ex 24.0 78.7
```

```
qplot(temp, pps, data = crick, color = species)
```



The main purpose here is to compare `pps` *(more precisely the mean of* `pps`*) for the two species. Consider this summary of data:*

```
crick %>% group_by(species) %>% summarise(m_pps = mean(pps), m_temp = mean(temp))
## # A tibble: 2 x 3
##   species m_pps m_temp
##    <fctr> <dbl>  <dbl>
## 1      ex 85.59  25.76
## 2     niv 61.04  21.72
```

□ □

A formal comparison (where we ignore temperature) can be made with a ONE–WAY ANOVA. Introduce a dummy variable x_1 with $x_{i1} = 1$ if the ith row has `species = niv` and 0 for rows with `species = ex`.

$$y_i = \beta_0 + \beta_1 x_{i1} + \epsilon_i$$

```
crm1 = lm(pps ~ species, data = crick)
tidy(crm1)
##          term estimate std.error statistic   p.value
## 1 (Intercept)    85.59      3.17    27.001 1.363e-21
## 2  speciesniv   -24.55      4.34    -5.656 4.638e-06
```

The large difference in group means of about 25 pps is, however, an artifact.

The graph shows that pulse rate is highly associated with temperature.

1. This CONFOUNDING variable means that we should worry that any difference in mean pulse rate was caused by a difference in the temperatures at which you measured pulse rate, as the average temperature for the O. exclamationis measurements was 3.6 C higher than for O. niveus.
2. We should also worry that O. exclamationis might have a higher rate than O. niveus at some temperatures but not others.

We can CONTROL FOR or ADJUST FOR temperature with ANCOVA as follows: Fit two parallel regression lines: Introduce x_2 as the temperature:

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + e$$

We can think of this as an ADDITIVE MODEL: The effect of changing log–conc is the same for each treatment.

```
crm2 = lm(pps ~ species + temp, data = crick)
tidy(crm2)
##          term estimate std.error statistic   p.value
## 1 (Intercept)   -7.855    2.7605    -2.845 8.365e-03
## 2  speciesniv   -9.898    0.7861   -12.591 8.193e-13
## 3        temp    3.628    0.1055    34.378 7.912e-24
```

Hence after controlling for the effect of temperature the difference in mean pps is about 10 units which is also what the graphics suggests.

```
qplot(temp, pps, color = species, data  =  crick) +
    geom_line(aes(temp, predict(crm2)))
```

The interpretation of the model is the the mean difference in pps is about 10 for any temperature. In this sense we can say that species and temperature have an ADDITIVE EFFECT.

It is evident that the two regression lines are parallel, but suppose that was not the case: We introduce a third dummy variable x_3 which is 0 when `species = ex` and which is equal to temperature when `species = niv`. Hence x_3 is simply the elementwise product of x_1 and x_2.

Two non–parallel regression lines:

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i3} + \epsilon_i$$

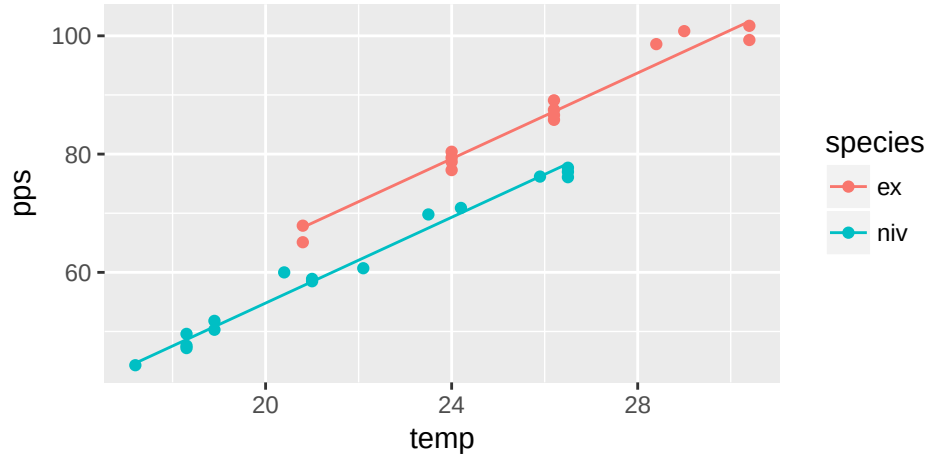


Figure 6.4: fixme

```
crm3 <- lm(pps ~ species + temp + species:temp, data = crick)
tidy(crm3)
##              term estimate std.error statistic   p.value
## 1     (Intercept) -11.0408    4.2219   -2.6152 1.465e-02
## 2      speciesniv  -4.7658    5.2045   -0.9157 3.682e-01
## 3            temp   3.7514    0.1628   23.0380 7.961e-19
## 4 speciesniv:temp  -0.2133    0.2138   -0.9975 3.277e-01
```

In this case a comparison of the species must in general be made at specific temperatures, see Sec. 6.10.

6.2 What is a linear model (II) - the assumptions

The `lm()` function will fit a linear model to data as has been illustrated above. Now we shall be more precise about what a linear model is.

From the perspective of data, the situation is: We have observed data or RESPONSES $y_1, \ldots, y_N$. To each y_i there are p EXPLANATORY VARIABLES $x_{i1}, x_{i2}, \ldots, x_{ip}$.

We shall make assumptions about the RANDOM PROCESS that has generated these data:

1. We shall assume that each y_i is a REALIZATION of a NORMAL DISTRIBUTED random variable Y_i where
$$Y_i \sim N(\mu_i, \sigma^2),$$
such that the mean of Y_i is $\mathbb{E}(Y_i) = \mu_i$ and the variance of Y_i is $\mathbb{V}\mathrm{ar}(Y_i) = \sigma^2$ for $i = 1, \ldots, N$. Notice that we assume CONSTANT VARIANCE, i.e. the variance of each Y_i is the same, namely σ^2.

2. We shall assume that that the mean μ_i can be written as a WEIGHTED SUM of the covariates
$$\mu_i = x_{i1}\beta_1 + x_{i2}\beta_2 + \cdots + x_{ip}\beta_p \tag{6.4}$$

3. We shall assume that the Y_i's are INDEPENDENT such that $\mathbb{C}\text{ov}(Y_i, Y_j) = 0$.

From a practical modelling perspective focus is almost always on the form of the mean μ_i in (6.6) but there many other underlying assumptions: normality, constant variance, independence.

Notice that an equivalent way of writing the first two assumptions is that

$$Y_i = x_{i1}\beta_1 + x_{i2}\beta_2 + \cdots + x_{ip}\beta_p + \epsilon_i$$

where $\epsilon_i \sim N(0, \sigma^2)$.

We have seen that linear, multiple and polynomial regression falls under the category of linear models. So do ANCOVA models and so do ANOVA models.

When these assumptions are satisfied, there is a whole theory about "what to do". in terms of statistical inference.

This raises several questions:

1. What is this theory behind linear models?
2. How to verify that the assumptions are satisfied? This can usually not be done, but sometimes we can do the opposite: Convince ourselves that the assumptions are not satisfied. This requires subject matter knowledge and statistical techniques.
3. Which of the assumptions are most important? Normality, constant variance or independence? And what goes wrong when these assumtions are not satisfied?

6.3 Colinearity [tbw]

6.4 Coefficient of determinations - R^2 [tbc]

One way of summarizing a model into a single numerical quantity is by the COEFFICIENT OF DETERMINATION also denoted R^2.[10] [11]

R^2 tells how large a proportion of the variability in data the model explains in the following sense: If y_i denotes observations, $\bar{y}$ the average of these and $\hat{y}_i$ denotes the fitted or predicted values, then R^2 is defined as

$$R^2 = \frac{\sum_i (\hat{y}_i - \bar{y})^2}{\sum_i (y_i - \bar{y})^2} = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2} \tag{6.5}$$

The better the model fits to data, the smaller is the RESIDUAL SUM OF SQUARES $\sum_i (y_i - \hat{y}_i)^2$ and hence the closer is R^2 to one.

There is no built in function that returns R^2 so we create one on the fly. There is a variant of R^2 which is called the ADJUSTED COEFFICIENT OF DETERMINATION and we shall prepare our function for returning this quantity as well:

[10] FiXme Note: Treat F-test, AIC, R2 BEFORE talking about modelling
[11] FiXme Note: R2: Put details elsewhere; move alternative forms forward in section

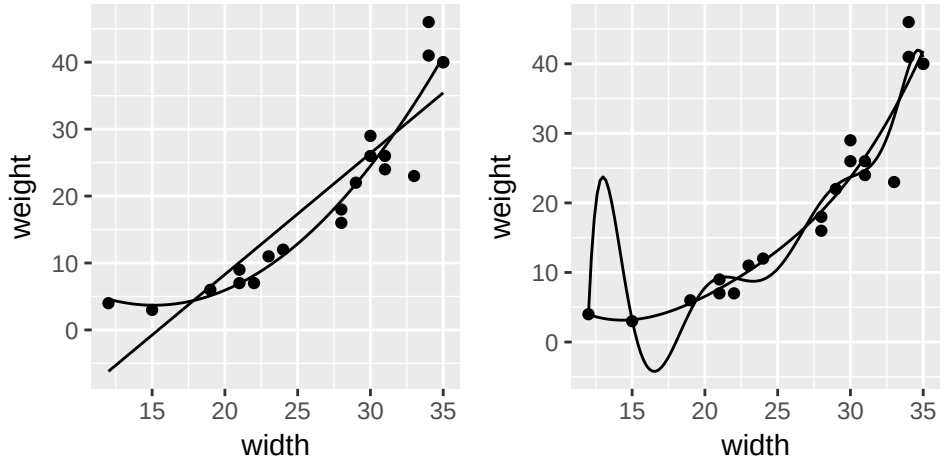
```
getR2 <- function(x, adjust=FALSE){
    if (adjust) summary(x)$adj.r.squared
    else summary(x)$r.squared
}
```

We see that in the R^2–sense, the polynomial with the highest degreee fits best to data:

```
pot1_l <- lm(weight ~ width, data = potatoes)
pot2_l <- lm(weight ~ poly(width, 2), data = potatoes)
pot5_l <- lm(weight ~ poly(width, 6), data = potatoes)
potp_l <- lm(weight ~ poly(width, 8), data = potatoes)
mlist <- list(pot1_l, pot2_l, pot5_l, potp_l)
sapply(mlist, getR2)
## [1] 0.8253 0.9227 0.9253 0.9367
```

However, plotting the fits show that we are really comiover fitting data:

```
nd <- data.frame(width=seq(12, 35, by=.2))
pp <- ggplot(potatoes, aes(width, weight)) + geom_point()
multiplot(
pp + geom_line(aes(width, predict(pot1_l, newdata=nd)), nd) +
    geom_line(aes(width, predict(pot2_l, newdata=nd)), nd),
pp + geom_line(aes(width, predict(pot5_l, newdata=nd)), nd) +
    geom_line(aes(width, predict(potp_l, newdata=nd)), nd),
cols=2)
```



6.4.1 Other views on R^2

When a model contains the constant term, i.e. we have a model of the form

$$y = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p + e$$

with $p + 1$ regression parameters, then R^2 has two additional appealing interpretations: 1) R^2 is also the squared correlation between the observed and fitted values and 2) R^2 is the slope when regressing the fitted values on the observed values:

```
y <- potatoes$weight
y.hat <- fitted(pot1_l)
getR2(pot1_l)
## [1] 0.8253
cor(y, y.hat)^2
## [1] 0.8253
coef(lm(y.hat ~ y))
## (Intercept)           y
##      3.5820      0.8253
```

6.4.2 A problem with R^2 - the adjusted R^2

A problem with R^2 is that it will increase as more and more covariates are thrown into the model – even if the model makes no sense. Consider (6.5) The better the model fits to data, the smaller is $\sum_i (y_i - \hat{y}_i)^2$ and hence the closer is R^2 to one. In a regression model with $p+1$ parameters, the ADJUSTED R^2 is defined as

$$\bar{R}^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2/(n-(p+1))}{\sum_i (y_i - \bar{y})^2/(n-1)}$$

Notice: $\bar{R}^2$ does not necessarily increase as more parameters are added and nothing prevents $\bar{R}^2$ from becomming negative.[12]

We see that in the adjusted–R^2–sense, the second degreee polynomial fits best to data:

```
mlist <- list(pot1_l, pot2_l, pot5_l, potp_l)
sapply(mlist, getR2, adjust=T)
## [1] 0.8156 0.9136 0.8909 0.8907
```

6.4.3 Akaikes Information Criterion (AIC)

The Akaike Information Criterion (AIC) is a measure of the relative quality of a statistical model for a given set of data. AIC deals with the trade-off between the goodness of fit of the model and the complexity of the model. The RESIDUAL SUM OF SQUARES is

$$RSS = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

For a linear normal model with p predictors, the AIC is given as

$$AIC = N \log(RSS/N) + 2p = N \log(RSS) - N \log(N) + 2p$$

Models with small AIC values are preferred. A complex model with many parameters has a small RSS so if it was not for the $2p$ term, complex models would be preferred. But the addition of $2p$ will penalize complex models and as such AIC represents a TRADE–OFF between MODEL FIT and MODEL COMPLEXITY.

We see that in the AIC–sense, the second degreee polynomial fits best to data:

[12] FiXme Note: sh:linear: How much more intution?

```
sapply(mlist, AIC)
## [1] 130.9 116.6 123.9 124.6
```

6.5 Model comparisons via F-tests etc. [tbw]

6.6 Connecting F–test and R^2 [tbw]

6.7 Interpreting a polynomial

Consider the polynomial model for the the `potatoes` data in Sec. 6.1.1

```
pot1_lw2
##
## Call:
## lm(formula = weight ~ length + width + I(width^2), data = potatoes)
##
## Coefficients:
## (Intercept)       length        width   I(width^2)
##      8.8400       0.8310      -2.6180       0.0681
```

corresponding to the mathematical formulation[13]

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i2}^2 + \epsilon_i.$$

If we have a linear relation between y and x, i.e. $y = f(x) = \alpha_0 + \alpha_1 x$ then the interpretation is that when x changes from x to $x + h$ then y changes from $\alpha_0 + \alpha_1 x$ to $\alpha_0 + \alpha_1(x + h) = \alpha_0 + \alpha_1 x + \alpha_1 h$. Hence the change on the y–scale is $f(x + h) - f(x) = \alpha_1 h$ which is caused by a change on the x–scale by h. This effect is the same no matter which x we start with.

If instead $f(x) = \alpha_0 + \alpha_1 x + \alpha_2 x^2$ the interpretation is more involved. We have

$$\begin{aligned} f(x + h) &= \alpha_0 + \alpha_1(x + h) + \alpha_2(x + h)^2 \\ &= \alpha_0 + \alpha_1 x + \alpha_2 x^2 + \alpha_1 h + \alpha_2(h^2 + 2hx) \\ &= f(x) + \alpha_1 h + \alpha_2(h^2 + 2hx) \end{aligned}$$

Hence a change on the x-scale from x to $x + h$ means a change on the y scale of $f(x + h) - f(x) = \alpha_1 h + \alpha_2(h^2 + 2hx)$ and this amount depends (linearly) on the value of x.

This change (along with standard errors) can be calculated in different ways, for example as follows: We let $\beta_3 = \alpha_2$ and $\beta_2 = \alpha_1$.

```
x <- 5; h <-1
doBy::linest(pot1_lw2, c(0, 0, h, 2*x*h+h^2))
##          estimate    se df t.stat   p.value
## estimate   -1.868 0.445 16 -4.199 0.0006799
```

[13] FiXme Note: linear: SH: linear: this is fragile if notation changes

```
library(car)
deltaMethod(pot1_lw2, "b2*h + b3*(2*x*h + h^2)",
            parameterNames = c("b0", "b1", "b2", "b3"),
            constants = c(x = 5, h = 1))
##                                 Estimate    SE 2.5 %  97.5 %
## b2 * h + b3 * (2 * x * h + h^2)   -1.868 0.445 -2.74 -0.9963
```

Let us se the effect for a range of different x-values (speed)

```
f <- function(x.val){
    deltaMethod(pot1_lw2, "b2*h + b3*(2*x*h + h^2)",
                parameterNames = c("b0", "b1", "b2", "b3"),
                constants = c(x = x.val, h = 1))
}
speed <- seq(0, 50, by = 10)
out <- lapply(speed, f)
out <- do.call(rbind, out)
out <- as.data.frame( cbind( out, speed ) )
rownames(out) <- NULL
out
##   Estimate     SE   2.5 %  97.5 % speed
## 1  -2.5499 0.5583 -3.6442 -1.4556     0
## 2  -1.1869 0.3372 -1.8478 -0.5260    10
## 3   0.1761 0.1828 -0.1823  0.5345    20
## 4   1.5391 0.2664  1.0170  2.0612    30
## 5   2.9021 0.4756  1.9698  3.8343    40
## 6   4.2651 0.7066  2.8802  5.6500    50
names(out)[c(3, 4)] <- c("lwr", "upr")
```

The effect of increasing speed by one unit from 1 to 2 is 1.213 while the effect of increasing speed by one unit from 3 to 4 is 1.613.

Notice that the standard errors (confidence bands) of these predictions increase the further x moves away from the center of data. See Fig.6.5.

```
qplot(speed, Estimate, data = out) + geom_ribbon(aes(ymin = lwr, ymax = upr), alpha = .2)
```

6.8 Model checking – residuals etc

6.8.1 A minimal check

Checking a model is in many ways an art more than a science, and it takes quite some experience to decide when a model does not fit adequately to data. However, the following is the minimum to do: The RAW RESIDUALS are computed as the observed minus the fitted values;[14]

$$r_i = y_i - \hat{\mu}_i$$

If the model fits to data, then one can show that the r_i's are approximately $N(0, \sigma^2)$ distributed and approximately independent.

[14] FiXme Note: sh:linear: Use residual plot from MESS package.

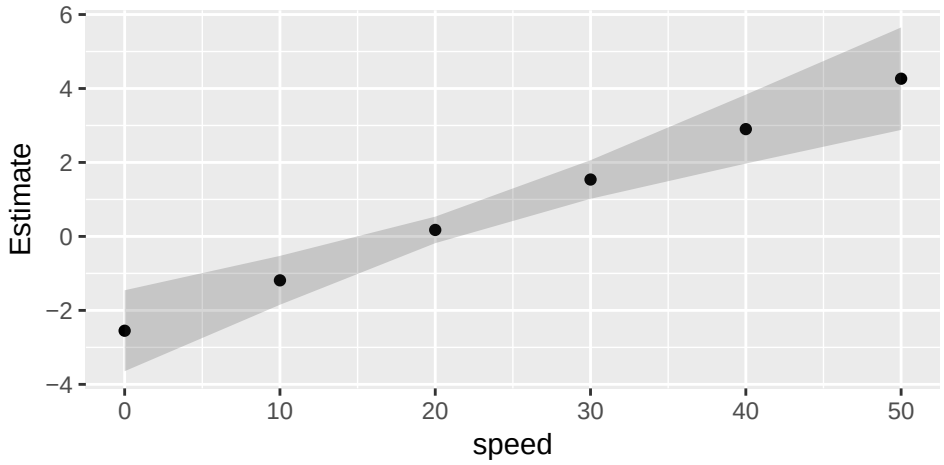


Figure 6.5: Applying the delta method to a polynomial regression. Gray indicate 95 pct confidence bands.

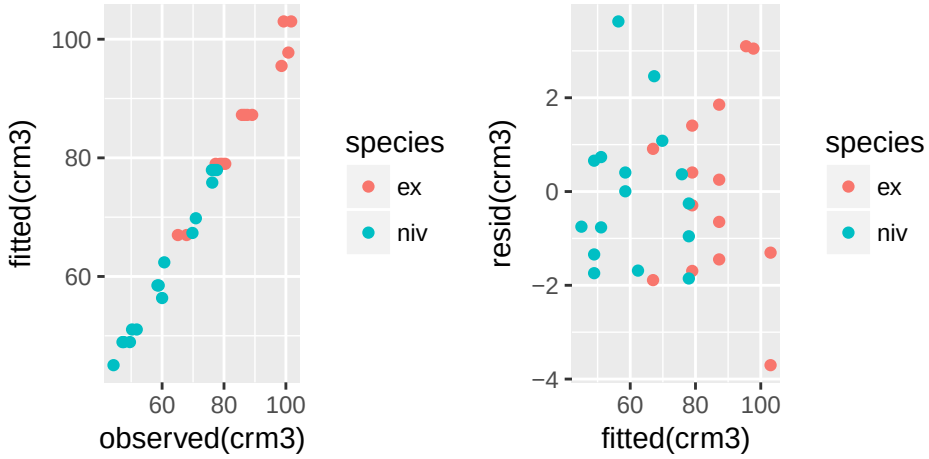


Figure 6.6: Diagnostic plots.

A minimum is to make the plots in Fig. 6.6.

```
observed <- function(object)
    model.response(model.frame(object))

multiplot(
    qplot(observed(crm3), fitted(crm3), color = species, data = crick),
    qplot(fitted(crm3), resid(crm3), color = species, data = crick),
    cols = 2)
```

Points should scatter randomly around the line if the models hold. This is not the case here, and moreover the residuals show a clustering according to treatment.

6.8.2 Example of some model deficiencies

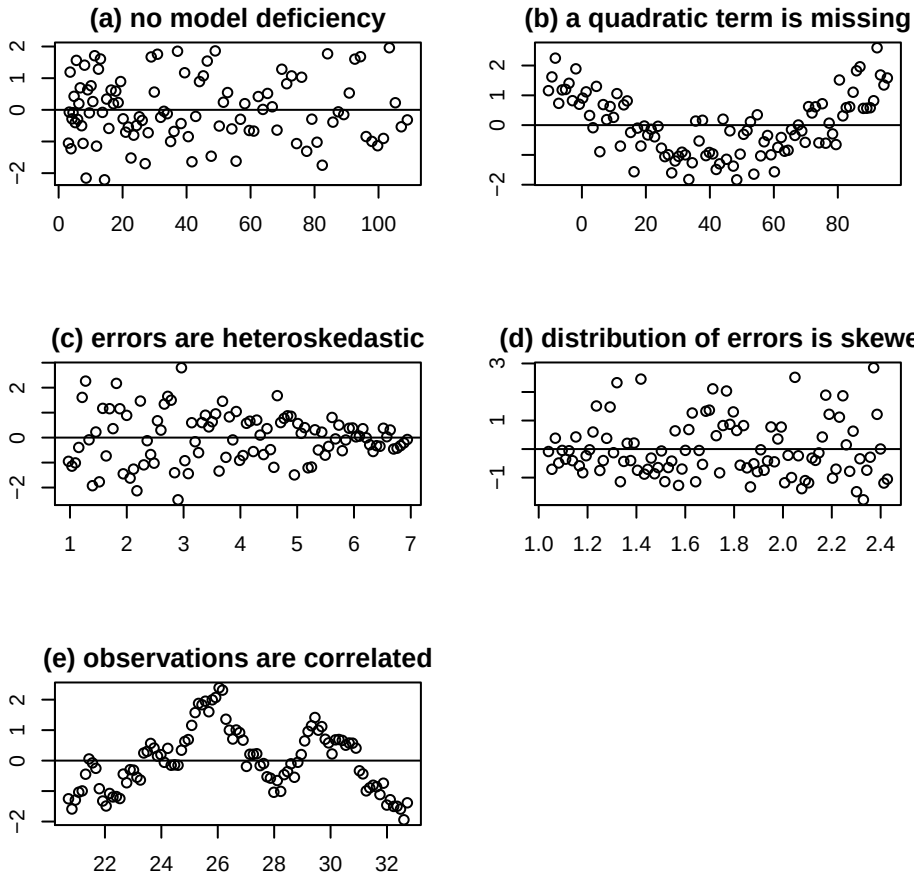


Figure 6.7: Examples of model deficiencies.

Some comments on Figure 6.7:

When things look right: Figure 6.7, (a), is a plot of what residuals could look like when the model assumptions are met.

When the mean is not adequately specified: If the systematic part of the model is not adequate, i.e. if it does not account for all systematic variation in data, this can sometimes be seen when plotting the residuals against the fitted values, see Figure 6.7 (b). Remedy: add a covariate, here the square of the x-variable.

When the variance is not constant: In some cases the variability of data increases with the mean such that the variance is not constant but increases with the mean. Figure 6.7, (c), shows what the residuals could look in this case. Remedy: Transform the response or use a generalized linear model with a variance function depending on the mean.

When data are not normal: Figure 6.7, (d), shows what residuals could look like if data are not normal (but, in this case, right skewed). Remedy: use a different distribution for the response with a generalized linear model or transform the response.

When observations are not independent: Figure 6.7, (e), shows what residuals could look like if data are not indpendent.

Remedy: general advice: account for the dependency in the model, i.e. do not use a linear model.

6.9 Testing effects

Effects of factors are described by the "differences" their levels "cause" in the response. Classically, the term "contrast" is used for such differences. Consider again the additive model for the bacteria data

```
tooth2
##
## Call:
## lm(formula = len ~ supp + dosef, data = ToothGrowth)
##
## Coefficients:
## (Intercept)       suppVC      dosefME      dosefHI
##       12.45        -3.70         9.13        15.50
```

An initial step is usually to test for removal of each effect; i.e.

```
anova(tooth2)
## Analysis of Variance Table
##
## Response: len
##           Df Sum Sq Mean Sq F value  Pr(>F)
## supp       1    205     205    14.0 0.00043 ***
## dosef      2   2426    1213    82.8 < 2e-16 ***
## Residuals 56    820      15
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
drop1(tooth2, test = "F")
```

```
## Single term deletions
##
## Model:
## len ~ supp + dosef
##        Df Sum of Sq  RSS AIC F value  Pr(>F)
## <none>               820 165
## supp    1       205 1026 176    14.0 0.00043 ***
## dosef   2      2426 3247 244    82.8 < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We have concluded that there is a significant effect of having `lucine` in the model. Next we go one step further and try to find out where this difference comes from: The `glht()` function in the **multcomp** package can do this and so can the `esticon` from **doBy**.

We can compute all pairwise differences as:

```
library(multcomp)
ddd <- glht(tooth2, mcp(dosef = "Tukey"))
ddd
##
##   General Linear Hypotheses
##
## Multiple Comparisons of Means: Tukey Contrasts
##
##
## Linear Hypotheses:
##              Estimate
## ME - LO == 0     9.13
## HI - LO == 0    15.50
## HI - ME == 0     6.37
## same as
## glht(bm, c("dosefME = 0", "dosefHI = 0", "dosefHI - dosefME = 0"))
```

In passing we mention, that We can compute the global F–test for removing `dosef` from the model (same as obtained with anova() above) as:

```
summary(ddd, test = Ftest())
```

We can get confidence intervals for the contrasts with

```
## Multiplicity adjusted confidence intervals:
confint(ddd)
##
##   Simultaneous Confidence Intervals
##
## Multiple Comparisons of Means: Tukey Contrasts
##
##
## Fit: lm(formula = len ~ supp + dosef, data = ToothGrowth)
##
## Quantile = 2.407
```

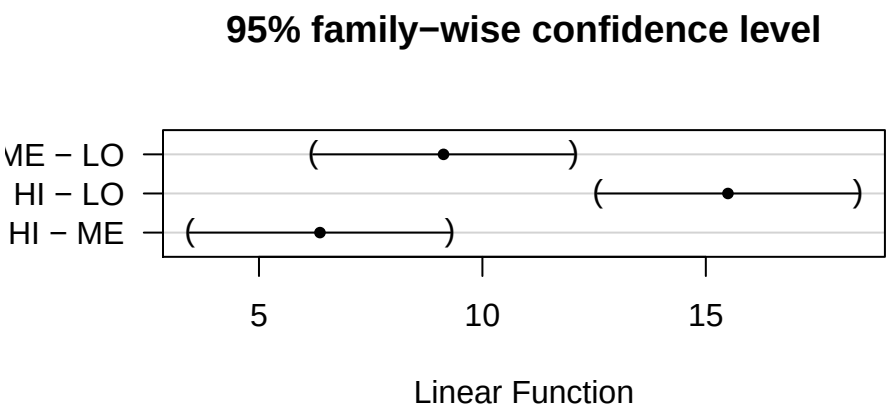


Figure 6.8: Adjusted confidence intervals for all pairwise comparisons for dosef.

```
## 95% family-wise confidence level
##
##
## Linear Hypotheses:
##              Estimate lwr    upr
## ME - LO == 0  9.130    6.217 12.043
## HI - LO == 0 15.495   12.582 18.408
## HI - ME == 0  6.365    3.452  9.278
## Unadjusted confidence intervals:
## confint(ddd, calpha = univariate_calpha())
```

There is a `plot()` function for plotting the confidence intervals, (see Fig. 6.8)

```
plot(confint(ddd))
```

If we want to control the FAMILYWISE ERROR RATE use

```
summary(ddd, test = adjusted())
##
##   Simultaneous Tests for General Linear Hypotheses
##
## Multiple Comparisons of Means: Tukey Contrasts
##
##
## Fit: lm(formula = len ~ supp + dosef, data = ToothGrowth)
##
## Linear Hypotheses:
##              Estimate Std. Error t value Pr(>|t|)
## ME - LO == 0     9.13       1.21    7.54  < 1e-06 ***
## HI - LO == 0    15.50       1.21   12.80  < 1e-06 ***
## HI - ME == 0     6.37       1.21    5.26  5.1e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## (Adjusted p values reported -- single-step method)
```

```
## same as summary(ddd)
```

Alternatively, we can test each contrast individually (the tests are NOT adjusted for multiplicity):

```
summary(ddd, test = univariate())
```

Similarly, we can compute all differences relative to the reference level:

```
ddd2 <- glht(tooth2, mcp(dosef = "Dunnett"))
summary(ddd2)
```

6.10 Prediction, estimates and contrasts, LSmeans

6.10.1 Linear estimates

Suppose a linear model contains the regression parameters $\beta_1, \ldots, \beta_P$. One is often interested in estimating a weighted sum of the model parameters

$$\theta = \lambda_1\beta_1 + \lambda_2\beta_2 + \cdots + \lambda_P\beta_P = \sum_{m=1}^{P} \lambda_m\beta_m$$

where $\lambda = (\lambda_1, \ldots, \lambda_P)$ is a vector of known weights. We shall call such a weighted sum for a[15] LINEAR PREDICTOR. After fitting the model the estimated linear predictor is

$$\hat{\theta} = \sum_{m=1}^{P} \lambda_m\hat{\beta}_m$$

Example: Calculate the estimated mean value on `supp = VC` and `supp = OJ` for `dosef = ME`. This can be obtained directly as:

```
new.data <- data.frame(supp = c("VC", "OJ"), dosef = "ME")
predict(tooth2, newdata = new.data, interval = "confidence")
##     fit   lwr   upr
## 1 17.88 15.91 19.86
## 2 21.58 19.61 23.56
```

An easier approach is as follows:[16][17]

```
library(doBy)
at <- list(supp = c("VC" ,"OJ"), dosef = "ME")
K <- LSmatrix(tooth2, at = at)
K
```

[15] FiXme Note: SH: linear: motivation for name 'linear predictor'
[16] FiXme Note: Somewhat cumbersome
[17] FiXme Note: Something with doBy versions...

```
##      (Intercept) suppVC dosefME dosefHI
## [1,]           1      1       1       0
## [2,]           1      0       1       0
linest(tooth2, K = K)
##   estimate     se df t.stat   p.value supp dosef
## 1    17.88 0.9883 56  18.10 4.598e-25   VC    ME
## 2    21.58 0.9883 56  21.84 4.461e-29   OJ    ME
```

Next, suppose the task is to compute the difference between `dosef = HI` and `dosef = ME`. This can be obtained as a difference between two linear predictors. We notice that this difference does not depend on `supp` and we get

```
lambda <- c(0, 0, -1, 1)
sum(coef(tooth2) * lambda)
## [1] 6.365
```

In practice we always want standard errors of such quantities:

```
library(doBy)
esticon(tooth2, cm = lambda)
##   beta0 Estimate Std.Error t.value DF Pr(>|t|) Lower Upper
## 1     0    6.365      1.21   5.259 56  2.4e-06  3.94  8.79
```

6.10.2 Estimation averaged across linear predictors

Suppose the task is to compute the mean response for `dosef = ME` averaged across the two `supp` methods.

We can obtain this by specifying λ manually (remembering that the coefficient for `supp = OJ` is set to zero):

```
lambda = c(1, 1/2, 1, 0)
esticon(tooth2, lambda)
##   beta0 Estimate Std.Error t.value DF Pr(>|t|) Lower Upper
## 1     0    19.73    0.8559   23.06 56        0 18.02 21.45
```

Such an average is called POPULATION AVERAGED MEAN or POPULATION MEAN or LEAST SQUARES MEAN or LSMEAN[18]

```
library(doBy)
at <- list(dosef = "ME")
LSmeans(tooth2, at = at)
##          estimate     se df t.stat   p.value dosef
## estimate    19.73 0.8559 56  23.06 2.885e-30    ME
```

The rule is that

1. covariates not listed in the `at`–argument are fixed at their average and

[18] FiXme Note: LSmeans: Consider crick data, implement differences in LSmeans

2. an average is calculated over the levels of factors not listed in the `at`–argument.

To obtain the population means for all 3 levels of `dosef` when `supp = OJ` we do:

```
LSmeans(tooth2, effect = "dosef", at = list(supp = "OJ"))
##   estimate     se df t.stat   p.value dosef supp
## 1    12.45 0.9883 56  12.60 5.490e-18    LO   OJ
## 2    21.58 0.9883 56  21.84 4.461e-29    ME   OJ
## 3    27.95 0.9883 56  28.28 7.627e-35    HI   OJ
```

```
LSmeans(crm2, effect = "species")
##   estimate     se df t.stat   p.value species temp
## 1    77.77 0.5333 27  145.8 1.185e-40      ex 23.6
## 2    67.87 0.4931 27  137.6 5.643e-40     niv 23.6
LSmeans(crm2, effect = "species", at = list(temp = 27))
##   estimate     se df t.stat  p.value species temp
## 1    90.09 0.4999 27  180.2 3.93e-43      ex   27
## 2    80.20 0.7171 27  111.8 1.52e-37     niv   27
```

6.11 Practical usage of `lm()`

In this section we shall explain more practical aspects of using the LM() function. As starting point we take the model with two parallel regression lines for the `cricks` data in Sec. 6.1.3.

```
crm2 = lm(pps ~ species + temp, data = crick)
tidy(crm2)
##          term estimate std.error statistic   p.value
## 1 (Intercept)   -7.855    2.7605    -2.845 8.365e-03
## 2  speciesniv   -9.898    0.7861   -12.591 8.193e-13
## 3        temp    3.628    0.1055    34.378 7.912e-24
```

6.11.1 Model objects

The model object `crm2` is technically a list (see Sec. **??**, p. **??** for information about lists)[19] with all sorts of information in it; for example the model specification, the data, the parameter estimates and so on.

```
class(crm2)
## [1] "lm"
is.list(crm2)
## [1] TRUE
names(crm2)
##  [1] "coefficients"  "residuals"     "effects"       "rank"
##  [5] "fitted.values" "assign"        "qr"            "df.residual"
##  [9] "contrasts"     "xlevels"       "call"          "terms"
## [13] "model"
```

[19] FiXme Note: linear: reference to lists

Since the model object is a list, we may extract values from the model object using the $–operator, e.g.

```
crm2$coefficients
## (Intercept)  speciesniv        temp
##      -7.855      -9.898       3.628
head(crm2$residuals)
##       1       2       3       4       5       6
##  0.2975 -2.5025 -1.9113 -0.5113  0.1887  1.1887
head(crm2$fitted)
##     1     2     3     4     5     6
## 67.60 67.60 79.21 79.21 79.21 79.21
```

For some of the attributes there exist EXTRACTOR FUNCTIONS and it is recommended to use these:

```
formula(crm2)
## pps ~ species + temp
coef(crm2)
## (Intercept)  speciesniv        temp
##      -7.855      -9.898       3.628
head(residuals(crm2))
##       1       2       3       4       5       6
##  0.2975 -2.5025 -1.9113 -0.5113  0.1887  1.1887
head(fitted(crm2))
##     1     2     3     4     5     6
## 67.60 67.60 79.21 79.21 79.21 79.21
```

For example, we can extract the model formula using the $–operator, but it requires that one knows where to look for it

```
crm2$formula       # Nope!
## NULL
crm2$call$formula  # Yes!
## pps ~ species + temp
```

Moreover, there are various methods available for model objects and each of these methods perform a specific task. Some of these methods are `print()`, `summary()`, `plot()`, `coef()`, `fitted()`, `predict()`. In addition, in several packages there are additional methods for `lm` objects. For example in the **doBy** package there are methods for computing linear estimates, see Sec. 6.10, p. 107.

6.11.2 Updating a model – `update()`

Say we want to extend the existing model to allow for the regression lines being non-parallel; that is we want to add an interaction between `species` and `temperature`.

We can create a new model from scratch or use the `update()` function:

```
crm22 = lm(pps ~ species + temp + species:temp, data = crick)
crm22 <- update(crm2, . ~ . + species:temp)
crm22
##
## Call:
## lm(formula = pps ~ species + temp + species:temp, data = crick)
```

```
##
## Coefficients:
##     (Intercept)       speciesniv             temp  speciesniv:temp
##         -11.041           -4.766            3.751           -0.213
```

6.11.3 Plotting a model

There is a `plot()` method for `lm`-objects. We shall not go into details about these four plots but merely refer to Sec. 6.8, p. 101.

6.11.4 Summary information – `summary()`

```
summary(crm22)
##
## Call:
## lm(formula = pps ~ species + temp + species:temp, data = crick)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -3.703 -1.332 -0.124  0.867  3.628
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)      -11.041      4.222   -2.62    0.015 *
## speciesniv        -4.766      5.204   -0.92    0.368
## temp               3.751      0.163   23.04   <2e-16 ***
## speciesniv:temp   -0.213      0.214   -1.00    0.328
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.81 on 26 degrees of freedom
## Multiple R-squared:  0.99,Adjusted R-squared:  0.989
## F-statistic:  854 on 3 and 26 DF,  p-value: <2e-16
```

There is a detail about `summary()`: A call to `summary()` returns an object (which is a list):

```
crm22.sum <- summary(crm22)
class(crm22.sum)
## [1] "summary.lm"
names(crm22.sum)
##  [1] "call"          "terms"         "residuals"     "coefficients"
##  [5] "aliased"       "sigma"         "df"            "r.squared"
##  [9] "adj.r.squared" "fstatistic"    "cov.unscaled"
```

These elements of the list can be accessed directly using $ and in some cases by accessor methods:

```
coef(crm22)
##     (Intercept)      speciesniv            temp speciesniv:temp
##        -11.0408         -4.7658          3.7514         -0.2133
```

```
coef(crm22.sum)
##                 Estimate Std. Error t value  Pr(>|t|)
## (Intercept)     -11.0408     4.2219 -2.6152 1.465e-02
## speciesniv       -4.7658     5.2045 -0.9157 3.682e-01
## temp              3.7514     0.1628 23.0380 7.961e-19
## speciesniv:temp  -0.2133     0.2138 -0.9975 3.277e-01
```

Table contains

- Parameter estimates
- Standard errors of estimates
- Test statitics for testing whether these parameters are zero
- Corresponding p–values

Notice that `coef()` acts differently when applied to a `summary.lm` and a `lm` object.

```
sigma(crm22)
## [1] 1.805
```

6.11.5 Summaries using broom

Shorter summaries using the **broom** package:

```
broom::tidy(crm22)
##              term estimate std.error statistic   p.value
## 1     (Intercept) -11.0408    4.2219   -2.6152 1.465e-02
## 2      speciesniv  -4.7658    5.2045   -0.9157 3.682e-01
## 3            temp   3.7514    0.1628   23.0380 7.961e-19
## 4 speciesniv:temp  -0.2133    0.2138   -0.9975 3.277e-01
```

```
broom::glance(crm22)
##   r.squared adj.r.squared sigma statistic   p.value df logLik   AIC   BIC
## 1      0.99        0.9888 1.805     854.5 4.392e-26  4 -58.14 126.3 133.3
##   deviance df.residual
## 1    84.72          26
sigma(crm22)
## [1] 1.805
```

Output contains:

- Estimate $\hat{\sigma}$ of the residual standard error.
- Degrees of freedom are number of observations minus number of regression parameters
- Coefficient of determination (R^2 and adjusted R^2): Percentage of variation in data explained by the model.
- F–statistic: The result from comparing the model with the simplest possible model, namely the model with `1` on the right hand side.

6.11.6 Other information from the model object

20

```
broom::tidy(crm22)
##              term estimate std.error statistic   p.value
## 1     (Intercept) -11.0408    4.2219   -2.6152 1.465e-02
## 2      speciesniv  -4.7658    5.2045   -0.9157 3.682e-01
## 3            temp   3.7514    0.1628   23.0380 7.961e-19
## 4 speciesniv:temp  -0.2133    0.2138   -0.9975 3.277e-01
broom::glance(crm22)
##   r.squared adj.r.squared sigma statistic   p.value df logLik   AIC   BIC
## 1      0.99        0.9888 1.805     854.5 4.392e-26  4 -58.14 126.3 133.3
##   deviance df.residual
## 1    84.72          26
```

```
## Residual standard deviation:
sigma(crm22)
## [1] 1.805
crm22.sum$sigma
## [1] 1.805
##sigma <- sqrt(sum(residuals(crm22)^2) / crm22£df.residual)
##sigma
```

```
## Parameter estimates
coef(crm22)
##     (Intercept)      speciesniv            temp speciesniv:temp
##        -11.0408         -4.7658          3.7514         -0.2133
```

```
## Variance of parameter estimates
V <- vcov(crm22); V
##                 (Intercept) speciesniv     temp speciesniv:temp
## (Intercept)          17.824    -17.824 -0.68297         0.68297
## speciesniv          -17.824     27.087  0.68297        -1.10006
## temp                 -0.683      0.683  0.02652        -0.02652
## speciesniv:temp       0.683     -1.100 -0.02652         0.04572
##X <- model.matrix(crm22)
##sigma(crm22)^2 * solve(t(X) %*% X)
```

t-statistic; $t = \hat{\beta}_i / \sqrt{\mathbb{V}\mathrm{ar}(\hat{\beta}_i)}$:

```
## t-test statistic
t.stat <- coef(crm22) / sqrt(diag(V))
t.stat
##     (Intercept)      speciesniv            temp speciesniv:temp
##         -2.6152         -0.9157         23.0380         -0.9975
## p-values
2 * (1 - pt(abs(t.stat), df = crm22$df.residual))
##     (Intercept)      speciesniv            temp speciesniv:temp
##         0.01465         0.36823         0.00000         0.32769
```

[20] FiXme Note: Other information: Move this to "Matrix representation ..."

6.11.7 Confidence interval – `confint()`

Confidence interval $\hat{\beta}_i \pm 1.96 se(\hat{\beta}_i)$:

```
confint(crm22)
##                    2.5 %  97.5 %
## (Intercept)     -19.7190 -2.3627
## speciesniv      -15.4638  5.9321
## temp              3.4167  4.0862
## speciesniv:temp  -0.6528  0.2262
```

6.11.8 Fitted values and predicting new cases – `predict()`

Fitted values can be obtained with `fitted()` and with `predict()`. In addition, `predict()` method can be used for predicting new cases for a model. Details about the difference between CONFIDENCE INTERVAL and PREDICTION INTERVAL will be provided in Sec. 6.13, p. 118

```
pred.data <- data.frame(species = "ex", temp = 25:27)
predict(crm22, newdata = pred.data)
##     1     2     3
## 82.75 86.50 90.25
predict(crm22, newdata = pred.data, interval =  "confidence")
##     fit   lwr   upr
## 1 82.75 81.72 83.77
## 2 86.50 85.50 87.49
## 3 90.25 89.17 91.32
predict(crm22, newdata = pred.data, interval = "prediction")
##     fit   lwr   upr
## 1 82.75 78.90 86.59
## 2 86.50 82.66 90.34
## 3 90.25 86.38 94.11
```

6.12 Model comparison with `lm()`

```
crm2
##
## Call:
## lm(formula = pps ~ species + temp, data = crick)
##
## Coefficients:
## (Intercept)   speciesniv         temp
##       -7.85        -9.90         3.63
crm22
##
## Call:
## lm(formula = pps ~ species + temp + species:temp, data = crick)
##
## Coefficients:
```

```
##     (Intercept)       speciesniv             temp  speciesniv:temp
##         -11.041           -4.766            3.751           -0.213
```

The additive model is a submodel of the interaction model: The additive model can be formed by setting certain parameters to zero in the interaction model.

Most statistical programs produce standard output tables providing tests of several hypotheses. We will consider three of these tables, which are available in R(Tab. 6.3).

Table 6.3: Three commonly used tables.

R-function	Description
`anova(model)`	Sequential ANOVA table.
`drop1(model)`	Dropping effects.
`coef(summary(model))`	Parameter estimate table.

6.12.1 Sequential ANOVA table: `anova()`

```
anova(crm2, crm22)
## Analysis of Variance Table
##
## Model 1: pps ~ species + temp
## Model 2: pps ~ species + temp + species:temp
##   Res.Df  RSS Df Sum of Sq  F Pr(>F)
## 1     27 88.0
## 2     26 84.7  1      3.24  1   0.33
```

- `DF`: difference in the number of model parameters (=48-47).
- `Sum of Sq`: Difference in the RSS (residual sum of squares).
- `F`: value of the F-statistic.
- `Pr(>F)`: corresponding p–value.

The effects are tested sequentially as they appear in the model equation. It is tested, whether an effect is necessary, assuming that all the preceding effects are in the model. As a consequence, the results of the tests depend on the order in which the terms in the model are specified:

First consider:

```
anova(lm(pps ~ species + temp + species:temp, data = crick))
## Analysis of Variance Table
##
## Response: pps
##              Df Sum Sq Mean Sq F value Pr(>F)
## species       1   4500    4500    1381 <2e-16 ***
## temp          1   3850    3850    1182 <2e-16 ***
## species:temp  1      3       3       1   0.33
## Residuals    26     85       3
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

- The first row (starting with `species`) provides a test for whether `species` can be removed from the model `pps ~ species`, i.e. a test of the model `pps ~ 1` against the larger model `pps ~ species`.
- The second row (starting with `temp`) is a test for whether `temp` can be removed from the model `pps ~ species + temp`, i.e. a test of the model `pps ~ species` against the larger model `pps ~ species + temp`.
- The last row is a test for reducing the large model `pps ~ species + temp + species:temp` to `pps ~ species + temp`.

Next consider:

```
anova(lm(pps ~ temp + species + species:temp, data = crick))
## Analysis of Variance Table
##
## Response: pps
##              Df Sum Sq Mean Sq F value  Pr(>F)
## temp          1   7833    7833    2404 < 2e-16 ***
## species       1    517     517     159 1.4e-12 ***
## temp:species  1      3       3       1    0.33
## Residuals    26     85       3
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Here, the first row is a test of the model `pps ~ 1` against the larger model `pps ~ temp`. The second row is a test of the model `pps ~ temp` against the larger model `pps ~ species + temp` and so on. Hence, in this case the qualitative results remain unchanged but in other cases, it will matter which order the terms are specified in.

6.12.2 Dropping each term in turn using `drop1()`

The `drop1()` function provides a table, where effects are removed in turn – assuming that all the other effects are contained in the model. Consider

```
drop1(crm2, test = "F")
## Single term deletions
##
## Model:
## pps ~ species + temp
##         Df Sum of Sq  RSS   AIC F value  Pr(>F)
## <none>                 88  38.3
## species  1       517  604  94.1     159 8.2e-13 ***
## temp     1      3850 3938 150.3    1182 < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The first row is a test for removing `species` from the model `pps ~ species + temp`; the second row is a test for removing `temp` from the model `pps ~ species + temp`. Next consider

```
drop1(crm22, test = "F")
## Single term deletions
##
## Model:
## pps ~ species + temp + species:temp
##              Df Sum of Sq  RSS  AIC F value Pr(>F)
## <none>                     84.7 39.1
## species:temp  1      3.24 88.0 38.3       1   0.33
```

Notice: When applying the `drop1()` function to the interaction model only the test for the interaction is returned: This is because `R` obeys the principle of marginality for factors: Do not test for a main effect if this is contained in higher order interactions.

6.12.3 Investigating parameter estimates using `coef()`

Some insight can be gained by looking at the parameter estimates, their standard errors and derived quantities:

```
coef(summary(crm22))
##                 Estimate Std. Error t value  Pr(>|t|)
## (Intercept)     -11.0408     4.2219 -2.6152 1.465e-02
## speciesniv       -4.7658     5.2045 -0.9157 3.682e-01
## temp              3.7514     0.1628 23.0380 7.961e-19
## speciesniv:temp  -0.2133     0.2138 -0.9975 3.277e-01
```

However, care must be taken here: Each test is a test for dropping a term from the model assuming that all other parameters are present. For example `spieciesniv` is the the difference in intercept between the `niv` and the `ex` spicies. It appears that this parameter can be removed from the model with two different slopes. Hence the test is really a test for a model with common intercept and different slopes against a model with different intercepts and different slopes. But a model with common intercept (that is when `temp = 0`) depends on the temperature scale: If temperature is changed from Celcisus to Fahrenheit, then the intercept will change. To drive home this point further, consider

```
coef(summary(crm2))
##             Estimate Std. Error t value  Pr(>|t|)
## (Intercept)   -7.855     2.7605  -2.845 8.365e-03
## speciesniv    -9.898     0.7861 -12.591 8.193e-13
## temp           3.628     0.1055  34.378 7.912e-24
```

In a model with common slopes, there is evidently a difference in the intercepts - as Fig. 6.4, p. 96 also suggests.

6.12.4 Which table to use?

All three tables provide a collection of tests. The most useful table is the table (produced by `drop1()` in R) because it tests for each effect in turn whether it is necessary. This feature is especially useful, if one starts with a large model containing several effects.

The sequential anova table is less useful because of the dependence of the tests on the sequence of the model terms.

The least useful table (with respect to testing) is the parameter estimate table. It provides (Wald) tests that each parameter is equal to zero. This can only be used as a test that the complete effect can be dropped, if either the effect is a factor with only two levels or a continuous covariate.

6.13 Confidence and prediction intervals***

6.14 Model specification and model formulae***

We have seen different specifications on the right hand side of a MODEL FORMULA in R, i.e. the part that comes after the tilde (~).

We shall describe such specifications in more details based on this dataset where a and b are factors and x, y and z are variables:

```
dat2 <- cbind(expand.grid(a = c("a1", "a2"), b = c("b1", "b2")),
              x = 1:4, y = 11:14, z = 5:8 )
dat2
##    a  b x  y z
## 1 a1 b1 1 11 5
## 2 a2 b1 2 12 6
## 3 a1 b2 3 13 7
## 4 a2 b2 4 14 8
```

- The right hand side of a formula consists of a series of terms separated by "+" operators.
- A term consists of variable and factor names separated by ":" operators and a term is interpreted as the interaction of all the variables and factors appearing in the term.
- The "*" operator can be seen as a shortcut for creating many terms on the fly: Writing e.g. a*x is by R understood to be a+x+a:x.

From a formula and a dataset, the model matrix can be generated using model.matrix(). Below we use prmatrix() only to remove unimportant output.

```
prmatrix( model.matrix(~a + b:x, data = dat2) )
##   (Intercept) aa2 bb1:x bb2:x
## 1           1   0     1     0
## 2           1   1     2     0
## 3           1   0     0     3
## 4           1   1     0     4
```

Default is to include a column of ones in the model matrix. This can be overwritten with

```
prmatrix( model.matrix(~ -1 + a + b:x, data = dat2) )
##   aa1 aa2 bb1:x bb2:x
## 1   1   0     1     0
## 2   0   1     2     0
## 3   1   0     0     3
## 4   0   1     0     4
```

What does "a:a" now mean in this model language?

It means exactly the interaction between a and a which is a itself:

```
prmatrix( model.matrix(~ a : a, data = dat2) )
##   (Intercept) aa2
## 1           1   0
## 2           1   1
## 3           1   0
## 4           1   1
prmatrix( model.matrix(~ x : x, data = dat2) )
##   (Intercept) x
## 1           1 1
## 2           1 2
## 3           1 3
## 4           1 4
```

In addition to "a * b" being interpreted as "a + b + a : b" we have the following

- The "^" operator indicates crossing to the specified degree.
  For example "(a + b + c)^2" is identical to "(a + b + c)*(a + b + c)" which in turn expands to a formula containing the main effects for "a", "b" and "c" together with their second-order interactions.
- The "-" operator removes the specified terms, so that "(a + b + c)^2 - a:b" is identical to "a + b + c + b:c + a:c". It can also used to remove the intercept term: when fitting a linear model the form "y ~ x - 1" specifies a line through the origin. A model with no intercept can be also specified as "y ~ x + 0" or "y ~ 0 + x".

6.14.1 Formulae with arithmetic expressions

- Formulae can also involve arithmetic expressions. The formula "log(y) ~ a + log(x)" is legal.
- When such arithmetic expressions involve operators which are also used symbolically in model formulae, there can be confusion between arithmetic and symbolic operator use.
- To avoid confusion, the function I() is used to bracket those portions of a model formula where operators are used in their arithmetic sense. For example, in the formula "y ~ a + I(x + z)", the term "x + z" is to be interpreted as the sum of "x" and "z".

Example

```
prmatrix( model.matrix(~ x + z + I(x+z) +I(x^2) + sin(x+z) +
                       log(x), data = dat2) )
##   (Intercept) x z I(x + z) I(x^2) sin(x + z) log(x)
## 1           1 1 5        6      1    -0.2794 0.0000
## 2           1 2 6        8      4     0.9894 0.6931
## 3           1 3 7       10      9    -0.5440 1.0986
## 4           1 4 8       12     16    -0.5366 1.3863
```

6.14.2 Specifying polynomials

The function `poly()` is convenient for specifying a polynomial:

```
p1<- prmatrix( with(dat2, poly(x, 3)) )
##              1    2       3
## [1,] -0.6708  0.5 -0.2236
## [2,] -0.2236 -0.5  0.6708
## [3,]  0.2236 -0.5 -0.6708
## [4,]  0.6708  0.5  0.2236
p2<- prmatrix( with(dat2, poly(x, 3, raw = T)) )
##      1  2  3
## [1,] 1  1  1
## [2,] 2  4  8
## [3,] 3  9 27
## [4,] 4 16 64
```

In the first case, the columns are orthogonal; in the second case they are not (but the two matrices span the same space).

A polynomium can put into a model formula in different ways:

```
prmatrix( model.matrix(~ x + I(x^2), data = dat2) )
##   (Intercept) x I(x^2)
## 1           1 1      1
## 2           1 2      4
## 3           1 3      9
## 4           1 4     16
prmatrix( model.matrix(~ poly(x, 2), data = dat2) )
##   (Intercept) poly(x, 2)1 poly(x, 2)2
## 1           1     -0.6708         0.5
## 2           1     -0.2236        -0.5
## 3           1      0.2236        -0.5
## 4           1      0.6708         0.5
```

6.15 More advanced aspects of linear models***

6.16 What are these standard errors of estimates?***

```
cars.012 <- lm( dist ~ speed + I(speed^2), data = cars )
coef( summary( cars.012 ) )
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.47014   14.81716  0.1667   0.8683
## speed        0.91329    2.03422  0.4490   0.6555
## I(speed^2)   0.09996    0.06597  1.5153   0.1364
```

The `Estimate` column has a straight forward interpretation, but what about the next column in the output?

Suppose we could redo the study again several times. Then for each replicate of the study we would estimate the regression coefficients. These will most likely differ from replicate to replicate.

We can not replicate the study, but we can mimic the situation: Create a new dataset of the same size as `cars` but where rows of `cars` are sampled with replacement (such that some rows from `cars` may appear more than once while other rows may not appear at all in the new dataset).

```
v <- 1:nrow(cars)
w <- sample(v, replace=T)
cars2 <- cars[w,]
head(cars2, 4)
##    speed dist
## 34    18   76
## 19    13   46
## 49    24  120
## 43    20   64
m2 <- lm(dist ~ speed + I(speed^2), data = cars2)
coef(m2)
## (Intercept)       speed  I(speed^2)
##    -5.40281     2.65141     0.04671
```

Now we repeat this scheme many times

```
set.seed(1213)
N <- 200
dat <- matrix(0, nrow = N, ncol = 3)
for (i in 1:N){
    cars2 <- cars[sample(v, replace = TRUE),]
    m2 <- lm(dist ~ speed + I(speed^2), data = cars2)
    dat[i, ] <- coef(m2)
}
head(dat, 4)
##         [,1]  [,2]     [,3]
## [1,]  -4.316 1.721  0.08343
## [2,]  -2.881 1.928  0.06869
## [3,] -13.282 3.745 -0.02079
## [4,] -11.518 3.174  0.02063
apply(dat, 2, sd)
## [1] 12.11746  1.95588  0.06885
```

6.17 What if assumptions are not satisfied?***

Consider this setting: A one way anova model for investigating, say a treatment versus a control. We compare the means μ_1 and μ_2. Since we make the experiment by simulation, we know the answer: We choose μ_1 and μ_2 to be identical:

```
N <- 5
mu1 <- mu2 <- 7
sd1 <- 1; sd2 <- 1
sd <- c(rep(sd1, N), rep(sd2, N))
mu <- c(rep(mu1, N), rep(mu2, N))
fact <- factor(c(rep("ctl", N), rep("trt", N)))
fact
##  [1] ctl ctl ctl ctl ctl trt trt trt trt trt
## Levels: ctl trt
y <- rnorm(2 * N, mean = mu, sd = sd)
y
##  [1] 7.637 6.231 7.456 8.917 7.678 6.707 7.521 6.720 6.628 6.829
```

Compare treatment with control; is there a significant difference between means (short answer: no, because we generated data that way). Null hypothesis: $\mu_1 = \mu_2$; alternative hypothesis: $\mu_1 \neq \mu_2$.

At least two ways of making this test in R:

```
a <- t.test(y ~ fact); a
##
##  Welch Two Sample t-test
##
## data:  y by fact
## t = 1.5, df = 5.1, p-value = 0.2
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.4602  1.8665
## sample estimates:
## mean in group ctl mean in group trt
##             7.584             6.881
p <- a$p.value
lm(y ~ fact) %>% summary()  %>% coef()
##             Estimate Std. Error t value  Pr(>|t|)
## (Intercept)   7.5841     0.3228   23.50 1.144e-08
## facttrt      -0.7032     0.4565   -1.54 1.620e-01
```

Sometimes a hypothesis is rejected; even if the hypothesis is true. This is called an error of type I. If we us a 5% significance level, then there is 5% probability of make an error of type I.

If we redo a study 100 times, then even if the hypothesis is true, we will in 5 cases (on average) reject the hypothesis.

```r
Nsim <- 1000
p.vec <- rep(0, Nsim)
for( i in 1:Nsim ){
    y <- rnorm(2 * N, mean = mu, sd = sd)
    a <- t.test(y ~ fact, var.equal = TRUE)
    p.vec[i] <- a$p.value
}
```

We see that we reject the null hypothesis about the right number of times:

```r
c(sum(p.vec < 0.10), sum(p.vec < 0.05), sum(p.vec < 0.01)) / Nsim
## [1] 0.108 0.056 0.007
```

6.17.1 Non–normality

Let now the setting be that observations do not come from independent normal distributions but from independent Poisson distributions:

```r
Nsim <- 1000
p.vec <- rep(0, Nsim)
for( i in 1:Nsim ){
    y <- rpois(2 * N, lambda = mu)
    a <- t.test(y ~ fact, var.equal = TRUE)
    p.vec[i] <- a$p.value
}
```

We see that we reject the null hypothesis about the right number of times:

```r
c(sum(p.vec < 0.10), sum(p.vec < 0.05), sum(p.vec < 0.01)) / Nsim
## [1] 0.103 0.050 0.010
```

6.17.2 Non–constant variance

Now let us imagine that the standard deviation in the treatment group is much larger than in the control group.

```r
N <- 5
sd1 <- 1; sd2 <- 5
fact <- factor(c(rep("ctl", N), rep("trt", N)))
sd <- c(rep(sd1, N), rep(sd2, N))
```

```
Nsim <- 1000
p.vec <- rep(0, Nsim)
for( i in 1:Nsim ){
    y <- rnorm(2 * N, mean = mu, sd = sd)
    a <- t.test(y ~ fact, var.equal = TRUE)
    p.vec[i] <- a$p.value
}
```

We still reject the hypthesis about the right number of times, but not quite: We reject the hypothesis too often. This means that we are more likely to erroneously claim that we found a significant difference that is not there.

```
c(sum(p.vec < 0.10), sum(p.vec < 0.05), sum(p.vec < 0.01)) / Nsim
## [1] 0.138 0.076 0.024
```

Now make make the standard deviation in one group much much larger than in the other:

```
N <- 5
sd1 <- 1; sd2 <- 25
fact <- factor( c(rep("ctl", N), rep("trt", N)) )
sd <- c(rep(sd1, N), rep(sd2, N))
```

```
Nsim <- 1000
p.vec <- rep(0, Nsim)
for( i in 1:Nsim ){
    y <- rnorm(2 * N, mean = mu, sd = sd)
    a <- t.test(y ~ fact, var.equal = TRUE)
    p.vec[i] <- a$p.value
}
```

We reject the null hypothesis too often

```
c(sum(p.vec < 0.10), sum(p.vec < 0.05), sum(p.vec < 0.01)) / Nsim
## [1] 0.116 0.069 0.026
```

Tentative conclusion: We need a somewhat extreme situation before the tests become really misleading (when it comes to tests).

6.17.3 Non–independence

Suppose the variance of all random variables is σ^2 (constant variance), observations in different groups are independent but observations within each group are all positively correlated with correlation ρ.

```
N <- 5
rho <- 0.8
Vi <- matrix(rho, nr = N, nc = N)
diag(Vi) <- 1
V <- kronecker(diag(1,2), Vi)
V
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
##  [1,]  1.0  0.8  0.8  0.8  0.8  0.0  0.0  0.0  0.0   0.0
##  [2,]  0.8  1.0  0.8  0.8  0.8  0.0  0.0  0.0  0.0   0.0
##  [3,]  0.8  0.8  1.0  0.8  0.8  0.0  0.0  0.0  0.0   0.0
##  [4,]  0.8  0.8  0.8  1.0  0.8  0.0  0.0  0.0  0.0   0.0
##  [5,]  0.8  0.8  0.8  0.8  1.0  0.0  0.0  0.0  0.0   0.0
##  [6,]  0.0  0.0  0.0  0.0  0.0  1.0  0.8  0.8  0.8   0.8
##  [7,]  0.0  0.0  0.0  0.0  0.0  0.8  1.0  0.8  0.8   0.8
##  [8,]  0.0  0.0  0.0  0.0  0.0  0.8  0.8  1.0  0.8   0.8
##  [9,]  0.0  0.0  0.0  0.0  0.0  0.8  0.8  0.8  1.0   0.8
## [10,]  0.0  0.0  0.0  0.0  0.0  0.8  0.8  0.8  0.8   1.0
Nsim <- 1000
Y <- MASS::mvrnorm(Nsim, mu = mu, Sigma = V)
head(Y)
##       [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9] [,10]
## [1,] 7.502 7.523 7.271 6.577 6.863 7.227 7.654 6.141 7.232 7.467
## [2,] 7.270 5.892 5.593 5.738 6.414 7.192 7.384 7.396 7.298 7.346
## [3,] 6.591 7.051 6.588 6.574 7.080 7.073 7.101 6.977 7.230 7.515
## [4,] 5.950 5.423 5.765 5.348 6.153 7.581 8.331 7.593 7.426 7.118
## [5,] 7.582 8.368 8.401 7.804 8.640 8.136 7.780 7.380 8.690 7.541
## [6,] 7.909 7.970 7.097 8.221 8.418 6.167 5.094 6.545 4.896 5.037
```

```
p.vec <- rep(0, Nsim)
for( i in 1:Nsim ){
    y <- Y[i, ]
    a <- t.test(y ~ fact, var.equal = T)
    p.vec[i] <- a$p.value
}
```

Now things go terribly wrong:

```
c(sum(p.vec < 0.10), sum(p.vec < 0.05), sum(p.vec < 0.01)) / Nsim
## [1] 0.682 0.611 0.472
```

We reject a true null hypothesis way too often, so the risk of erroneously concluding that there is a significant difference (when there is not) is very large.

The reason for this is that if we ignor are positive correlation then we pretend to have more information than we really have:

Just look at the sample mean from group i (treatment or control):

$$\bar{y}^i = \frac{1}{N}\sum_{j=1}^{N} y_{ij}$$

The variance of $\bar{y}^i$ is

$$\mathbb{V}\text{ar}(\bar{y}^i) = \frac{1}{N^2}\mathbb{V}\text{ar}(\sum_{j=1}^{N} y_{ij})$$

If observations are independent then the variance of the sum is the sum of the variances and we get

$$\mathbb{V}\text{ar}(\bar{y}^i) = \frac{1}{N^2}N\sigma^2 = \sigma^2/N$$

But this is not the case when observations are not independent.

It is always true that

$$\mathbb{V}\text{ar}(x_1 + x_2 + \cdots + x_n) = \sum_{i=1}^{N}\sum_{j=1}^{N}\mathbb{C}\text{ov}(x_i, x_j)$$

In the specific case we get

$$\mathbb{V}\text{ar}(\sum_{j=1}^{N} y_j^i) = N\sigma^2 + N(N-1)\sigma^2\rho/2 = N\sigma^2(1 + (N-1)\rho/2)$$

so

$$\mathbb{V}\text{ar}(\bar{y}^i) = \frac{1}{N^2}N\sigma^2 = \frac{1}{N}\sigma^2(1 + (N-1)\rho/2)$$

which is larger than σ^2/N.

It is interesting to ask the question: How many independent observations, say M, do the N correlated observations correspond to? That is, which M will give

$$\frac{\sigma^2}{M} = \frac{1}{N}\sigma^2(1 + (N-1)\rho/2)$$

and the answer is

$$M = \frac{N}{1 + (N-1)\rho/2}$$

So for $N = 5$ and $\rho = 0.8$ we get

```
N <- 5
rho <- .8
M <- N / (1 + (N - 1) * rho / 2)
M
## [1] 1.923
```

so we have about "two independent pieces of information" and not 5 as we pretended to have.

6.18 Linear models in some detail***

6.18.1 What is a linear model (II) - the assumptions

The `lm()` function will fit a linear model to data as has been illustrated above. Now we shall be more precise about what a linear model is.

From the perspective of data, the situation is: We have observed data or RESPONSES $y_1, \ldots, y_N$. To each

y_i there are p EXPLANATORY VARIABLES $x_{i1}, x_{i2}, \ldots, x_{ip}$.

We shall make assumptions about the RANDOM PROCESS that has generated these data:

1. We shall assume that each y_i is a REALIZATION of a NORMAL DISTRIBUTED random variable Y_i where
$$Y_i \sim N(\mu_i, \sigma^2),$$
such that the mean of Y_i is $\mathbb{E}(Y_i) = \mu_i$ and the variance of Y_i is $\mathbb{V}\mathrm{ar}(Y_i) = \sigma^2$ for $i = 1, \ldots, N$. Notice that we assume CONSTANT VARIANCE, i.e. the variance of each Y_i is the same, namely σ^2.
2. We shall assume that that the mean μ_i can be written as a WEIGHTED SUM of the covariates
$$\mu_i = x_{i1}\beta_1 + x_{i2}\beta_2 + \cdots + x_{ip}\beta_p \tag{6.6}$$
3. We shall assume that the Y_i's are INDEPENDENT such that $\mathbb{C}\mathrm{ov}(Y_i, Y_j) = 0$.

From a practical modelling perspective focus is almost always on the form of the mean μ_i in (6.6) but there many other underlying assumptions: normality, constant variance, independence.

Notice that an equivalent way of writing the first two assumptions is that

$$Y_i = x_{i1}\beta_1 + x_{i2}\beta_2 + \cdots + x_{ip}\beta_p + e_i$$

where $e_i \sim N(0, \sigma^2)$.

We have seen that linear, multiple and polynomial regression falls under the category of linear models. So do ANCOVA models and so do ANOVA models.

When these assumptions are satisfied, there is a whole theory about "what to do". in terms of statistical inference.

This raises several questions:

1. What is this theory behind linear models?
2. How to verify that the assumptions are satisfied? This can usually not be done, but sometimes we can do the opposite: Convince ourselves that the assumptions are not satisfied. This requires subject matter knowledge and statistical techniques.
3. Which of the assumptions are most important? Normality, constant variance or independence? And what goes wrong when these assumtions are not satisfied?

6.18.2 Matrix representation of a linear model

Return again to the linear model

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + e$$

If we have N cases in a dataset we can organize the y's in the vector $y = (y_1, y_2, \ldots, y_N)$ and the predictors in the matrix X:

$$X = \begin{bmatrix} 1 & x_{11} & x_{12} & x_{13} \\ 1 & x_{21} & x_{22} & x_{23} \\ \vdots & \vdots & \vdots & \\ 1 & x_{N1} & x_{N2} & x_{N3} \end{bmatrix}$$

If we let $\beta = (\beta_0, \beta_1, \beta_2, \beta_3)$ and $e = (e_1, e_2, \ldots, e_N)$ then we can write

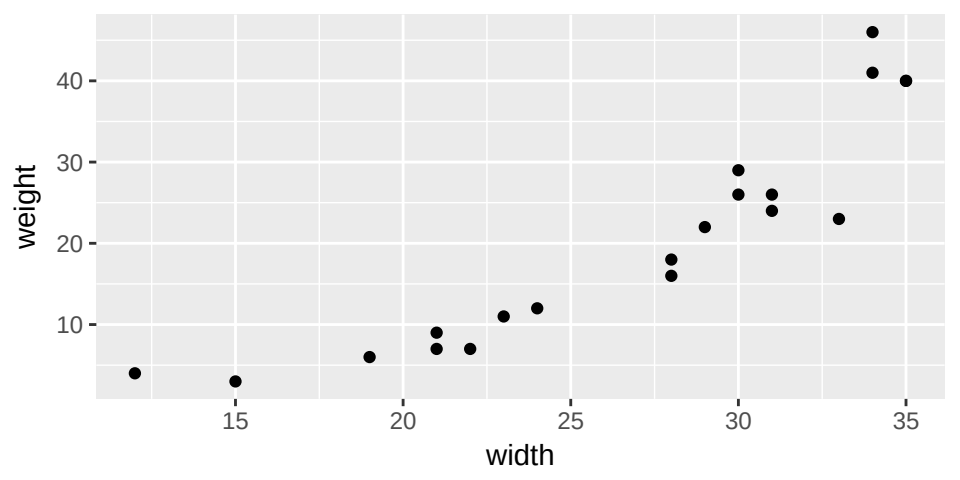
$$y = X\beta + e$$

Notice that what is observed is y and X.

6.18.3 Least squares – a minimization problem

Weight and width of potatoes:

```
library(ggplot2)
qplot(width, weight, data=potatoes)
```



Perhaps an approximately quadratic relation between y=`weight` and x=`width`:

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + e_i$$

```
X <- cbind(1, potatoes$width, potatoes$width^2)
y <- potatoes$weight
head(X)
##      [,1] [,2] [,3]
## [1,]    1   29  841
## [2,]    1   34 1156
## [3,]    1   31  961
## [4,]    1   28  784
## [5,]    1   21  441
## [6,]    1   35 1225
head(y)
## [1] 22 41 24 16  7 40
```

Let $\beta = (\beta_0, \beta_1, \beta_2)$. One way of estimating β is by the method of LEAST SQUARES: The best β is the vector that minimizes the sum–of–squares:

$$\sum_{i=1}^{N} [y_i - (\beta_0 + \beta_1 x_i + \beta_2 x_i^2)]^2$$

In R, this vector can be found using the `lm()` function:

```
mm <- lm(weight ~ width + I(width^2), data=potatoes)
beta <- coef(mm); beta
## (Intercept)       width  I(width^2)
##    25.15450    -2.83987     0.09394
```

We can write

$$y_i \approx \beta_0 + \beta_1 x_i + \beta_2 x_i^2$$

in matrix form for all N observations:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \approx \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & \vdots & \vdots \\ 1 & x_N & x_N^2 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{bmatrix}$$

or more compact

$$y \approx X\beta$$

Let $r = y - X\beta$. Then the LEAST SQUARES criterion can be formulated as: Choose as β the vector that makes the squared length of r (denoted by $||r||^2$) as small as possible.

The squared length is $r^\top r$ is called the RESIDUAL SUM OF SQUARES denoted RSS so the task is to minimize

$$RSS = (y - X\beta)^\top (y - X\beta)$$

One approach to doing so is to think of RSS as a function of β. We differentiate RSS with respect to β and set the derivatives to zero.

This leads to a system of equations called the NORMAL EQUATIONS

$$(X^\top X)\beta = X^\top y$$

and the solution to this system of equations is

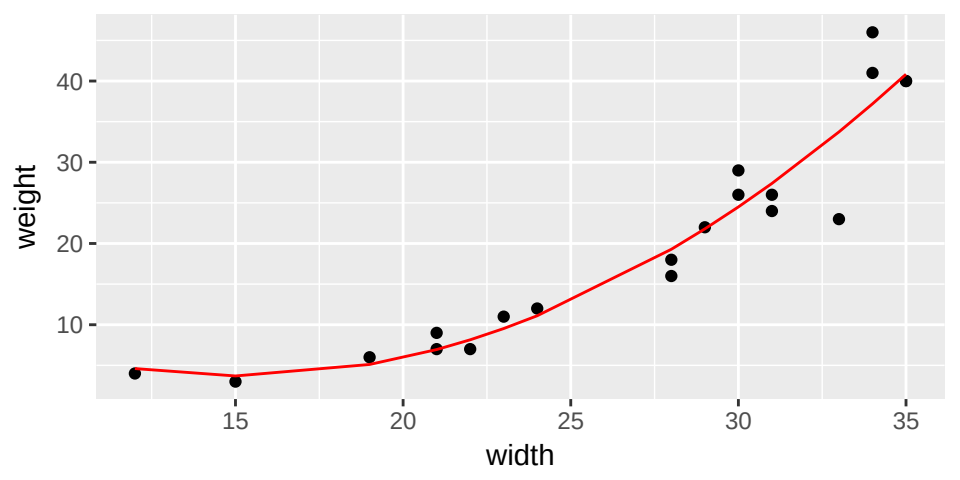
$$\beta = (X^\top X)^{-1} X^\top y$$

Define X and y and find β

Multiplying X and β gives the vector of FITTED VALUES:

```
fv <- X %*% beta; head(fv, 3)
##       [,1]
## [1,] 21.80
## [2,] 37.19
## [3,] 27.39
```

```
library(ggplot2)
qplot(width, weight, data=potatoes) +
    geom_line(aes(width, fv), col="red")
```



Index

List of Corrections