Introduction to R

–

A language and environment for statistical computing

david.kneis @ tu-dresden.de

ANSWER ITN course, Dresden, September 2016

Outline

About this course

- An appetizer, which might not suite everybody's taste
- A pragmatic introduction, that intentionally omits details

What is R?

- A language and environment for statistical computing
- A multi-purpose scripting language
- A free, cross-platform, open-source software
- A project with many contributors

https://www.r-project.org/

Reasons to learn R

- Process raw data into final results through scripts
  $\rightarrow$ Transparent, repeatable, re-usable

Reasons to learn R

- Process raw data into final results through scripts
  $\rightarrow$ Transparent, repeatable, re-usable
- Wide spectrum of (statistical) methods
  $\rightarrow$ Many available by default
  $\rightarrow$ $\approx$ 9000 add-on packages
  $\rightarrow$ Can code on your own

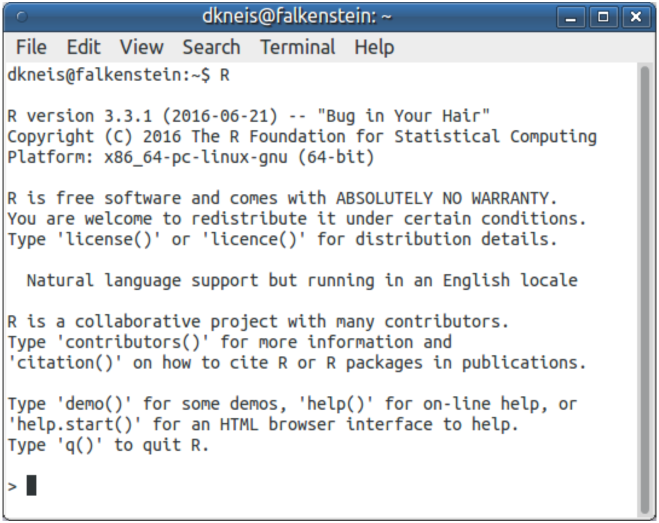
Reasons to learn R

- Process raw data into final results through scripts
  $\rightarrow$ Transparent, repeatable, re-usable
- Wide spectrum of (statistical) methods
  $\rightarrow$ Many available by default
  $\rightarrow$ $\approx$ 9000 add-on packages
  $\rightarrow$ Can code on your own
- Production of high-quality graphics

Reasons to learn R

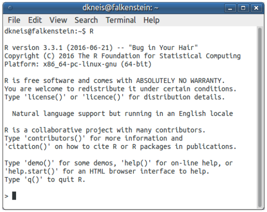
- Process raw data into final results through scripts
  $\rightarrow$ Transparent, repeatable, re-usable
- Wide spectrum of (statistical) methods
  $\rightarrow$ Many available by default
  $\rightarrow$ $\approx$ 9000 add-on packages
  $\rightarrow$ Can code on your own
- Production of high-quality graphics
- Large community
  $\rightarrow$ Web search brings up the answer to most questions

R terminal



```
dkneis@falkenstein:~$ R

R version 3.3.1 (2016-06-21) -- "Bug in Your Hair"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> 
```

R terminal: How to ...



```
dkneis@falkenstein:~$ R

R version 3.3.1 (2016-06-21) -- "Bug in Your Hair"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

Quit an R session

- Type `q()` + Enter

Get help on method known by name

- Type `?` followed by name
- Example: `?runif`
- Press `q` to exit help

Start help in web browser

- Type `help.start()`

Repeat entered commands

- Press $\boxed{\uparrow}$ key

Interrupt a computation

- Try Ctrl+C or ESC

Ways to interact with R

1. Enter commands at the R prompt
2. Put chain of commands in a text file (= 'script')
   – Copy & paste into terminal
   – Process file contents using R's `source()` function
3. Use dedicated development environment, e. g. Rstudio
4. Process R script as a batch job

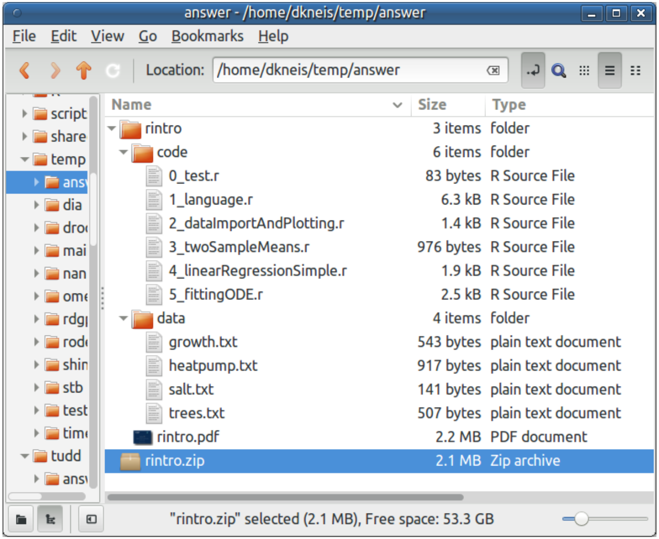
We will use approaches 1 & 2 during the course

Install files for this course

1. Open the link written on white board
2. Navigate to section '**R-related material**'
3. Download the '**Answer16**' course material for your operating system (blue colored link)
4. Save the archive '`rintro.zip`' into a folder of your choice
   e. g. `C:\Users\yourname\Documents`
   e. g. `/home/yourname/answer`
5. Unzip (right click in file browser)

If this fails, please request an off-line copy on USB stick.

Install files for this course


answer - /home/dkneis/temp/answer
File Edit View Go Bookmarks Help
Location: /home/dkneis/temp/answer
script
share
temp
ansv
dia
droi
mai
nan
ome
rdgi
rode
shin
stb
test
time
tudd
ansv

Name	Size	Type
rintro	3 items	folder
code	6 items	folder
0_test.r	83 bytes	R Source File
1_language.r	6.3 kB	R Source File
2_dataImportAndPlotting.r	1.4 kB	R Source File
3_twoSampleMeans.r	976 bytes	R Source File
4_linearRegressionSimple.r	1.9 kB	R Source File
5_fittingODE.r	2.5 kB	R Source File
data	4 items	folder
growth.txt	543 bytes	plain text document
heatpump.txt	917 bytes	plain text document
salt.txt	141 bytes	plain text document
trees.txt	507 bytes	plain text document
rintro.pdf	2.2 MB	PDF document
rintro.zip	2.1 MB	Zip archive

"rintro.zip" selected (2.1 MB), Free space: 53.3 GB

Running a script I: Copy & paste

1. Try to open '`0_test.r`' located in folder '`code`'
2. If the file doesn't open out-of-the-box, associate the file extension '`.r`' with your preferred editor
3. Copy & paste the file contents into the R terminal
   $\rightarrow$ Quick way is Ctrl+A, Ctrl+C, Ctrl+V

Running a script II

Type something like the following into the R terminal

```
setwd("C:/Users/Yourname/Documents/rintro/code")
getwd()
```

- Adapt the file path according to your work folder first
- On Windows: Replace any backslash '\' in path by '/'

Then, type the following to run the test script

```
source("0_test.r", echo=FALSE, print.eval=TRUE)
```

How to run code from this presentation

Option 1	Type yellow-shaded text into R terminal → Slow
Option 2	Copy yellow-shaded text from '`rintro.pdf`' → Selecting text may be a hassle
Option 3	Copy from respective '`.r`' file in folder '`code`'
Option 4	Run the respective '`.r`' file using `source()` → Insert `print()` and `stop()` as necessary

Outline

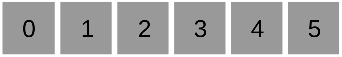
Ingredients of a script

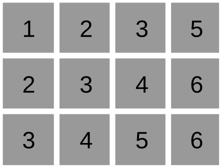
Variables	Store data under a name
Functions	Produce output (data) from some input
Control structures	Condition testing and iteration
Comments	Text following a # character

```r
x <- runif(1)  # assigns a random number to variable 'x'
if (x > 0.5)   # test a condition
  print(x)     # call to 'print' function
```

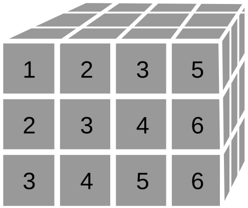
Variables

Name	Letter, followed by letters, digits, '_', period
Type	numeric, logical, character, list, factor (special values: NA, NaN, Inf, NULL)
Length	Vector of length 6
Dimension	





Matrix: 2 dimensions



3-dimensional array

Variables

Assignment to scalar variables

```
x <- 3.1415           # numeric
x <- "E. coli"        # character (overwrites old value)
x <- TRUE             # logical
```

```
x <- list(id=1, name="E. coli", rodShaped=TRUE)   # list
```

Info about a variable

```
print(x)     # print value
typeof(x)    # returns the type
is.list(x)   # type check (is.numeric, is.character, ...)
str(x)       # structure; useful for large lists
```

Variables

Vector constructors I: Frequently used

```
x <- seq(from=0, to=2, by=0.25)       # numeric sequence
x <- 1:5                              # integer sequence
x <- rep(0, times=3)                  # replication
```

Vector constructors II: Concatenation

```
x <- c("Athens", "Paris", "Rome")     # vector of strings
x <- c(red=255, green=0, blue=128)    # elements named
```

Some vector-related functions

```
length(x)                             # number of elements
names(x)                              # element names
```

Variables

Vector subsetting I: By position

```
x[1]                        # 1st element
x[1:2]                      # using a vector of positions
```

Vector subsetting II: By name

```
x["red"]                    # single element
x[c("blue","red")]          # using a vector of names
```

Vector subsetting III: By mask

```
x[x > 0]                    # logical mask
x[which(x > 0)]             # a more explicit alternative
```

Variables

Matrix constructors I: General method

```
x <- matrix(1:6, nrow=2, ncol=3, byrow=FALSE)
```

Matrix constructors II: Merging of column vectors

```
x <- cbind(temp= c(0, 10, 20),             # 1st column
           rate= c(0, 0.2, 0.4))           # 2nd column
```

Matrix constructors III: Merging of row vectors

```
x <- rbind(
  tap=   c(NO3= 0.5, NH4=  0, O2=12),     # 1st row
  river= c(NO3=   5, NH4=  1, O2=10),
  WWTP=  c(NO3=  10, NH4= 10, O2= 3))
```

Variables

Matrix attributes

```
ncol(x)             # number of columns
nrow(x)
colnames(x)         # column names
rownames(x)
```

Matrix subsetting

```
x[1, ]                   # 1st row (vector)
x[, 1]                   # 1st column (vector)
x[1, 1]                  # top left element (scalar)
x[1:2, 2:ncol(x)]        # sub-matrix
x[x[,"O2"] > 5, "O2"]  # subset by name / logical mask
```

Variables

Data frame

- A special type to store tabular data
- Use if columns differ in type (matrix won't fit then)
- A data frame is basically a list where ...
  - each element is a column vector
  - all those vectors are of common length

```
x <- data.frame(
  element= c("C", "Si", "N", "P"),
  group=   c(4, 4, 5, 5),
  mass=    c(12.01, 28.09, 14.01, 30.97)
)
```

Variables

Data frame treated as matrix

```
ncol(x)        # also 'nrow(x)'
colnames(x)    # also 'rownames(x)'
x[ ,3]         # last column, access by index
x[ ,ncol(x)]   #   as above
x[ ,"mass"]    # last column, access by name
x[1, ]         # first row, returns a list!
```

Data frame treated as a list

```
names(x)       # same as 'colnames(x)'
x$element      # extract column
```

Variables

Summary of important data structures

`vector`	Holds multiple values of **equal** type
	$\rightarrow$ scalars are vectors of length 1
`matrix`	like above, but arranged in 2 dimensions
`array`	like above, used for $>$ 2 dimensions
`list`	Holds values that **differ** in type
`data.frame`	List type, dedicated to tabular data

Operators

Numeric (`+ - * / ^`)

```
x <- 1:10
x * 2       # each element multiplied with scalar
x * x       # operands of same length: works element-wise
```

Matrix multiplication

```
x <- matrix(1:6, ncol=3)
y <- matrix(1:6, ncol=2, byrow=TRUE)
x %*% y
```

Operators

Comparisons (`> < >= <= == !=`)

```
x <= 5      # comparison for each element
!(x == 5)   # negation, same as 'x != 5'
```

Logical AND / OR

```
x <- c(TRUE, FALSE)
y <- c(FALSE, TRUE)
x & y                 # element wise AND
x | y                 # element wise OR
```

String manipulation

Concatenation

```
paste("use", "R", sep="")      # 'sep' specifies the glue
paste("matrix is of size",     # auto-converts numbers
   nrow(x), "x", ncol(x))
```

Substrings

```
substring("function", 1, 3)
```

Pattern matching and substitution

```
?regexpr
```

Flow control

Conditional execution I: Pure `if`

```
x <- runif(1)           # single random number, range [0,1]
if (x > 0.5) {          # logical statement
  print("greater")      # executed if TRUE
}                       # { } define a block of statements
```

Conditional execution II: Multiple branches

```
if (x > 0.5) {
  print("greater")
} else if (x < 0.5) {  # optional alternative condition(s)
  print("less")
} else {                # if all conditions are FALSE
  print("equal")        # very unlikely to be printed ever
}
```

Flow control

Iterating over elements of a vector

```
x <- 1:8
for (i in x) {       # i cycles through the elements of x
  print(2 ^ i)
}
```

Repeated execution

```
x <- 1
while (x > 0.1) {   # (non)-exit condition
  print(x)
  x <- runif(1)
}
```

Functions

Definition

`foo`: Given name, `x`: Formal argument(s), if any

```
foo <- function(x) {   # single argument function
  lowest <- min(x)     # processing of arguments
  highest <- max(x)    # + use of local variables
  c(lowest, highest)   # return value
}                      # closes body
```

Function call

Value of actual argument `y` passed to formal arg. `x`

```
y <- rnorm(n=100, mean=0, sd=1)    # sample from N(0,1)
foo(y)                             # same as range(y)
```

Functions

- Do **not** reference variables from the calling environment inside a function. Pass their values via dedicated formal arguments!
- This makes code safer and more re-usable.

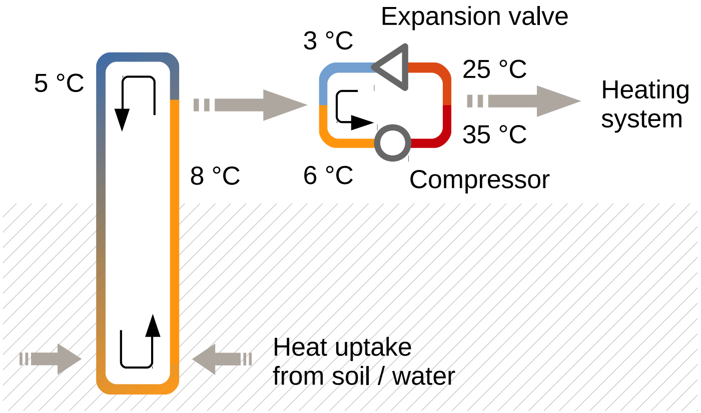
```
bad <- function (x) { x * a }  # origin of 'a' not obvious
a <- 2
bad(3)
```

- Do not ignore this advice until you learned R's **scoping rules**.

Outline

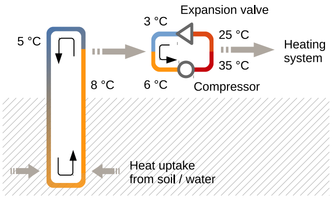
Example data

Heat pump



Example data

Example data



Recorded variables

`date`	YYYY-MM-DD
`heat`	Heat extracted from ground (*kWh*)
`power`	Electric power consumed by heat pump (compressor, hydraulic pumps) and all other domestic appliances (*kWh*)

Data import

```
?read.table
```

Reading delimited text

```
dat <- read.table(file="../data/heatpump.txt",
  sep="\t", header=TRUE)                          # TAB-delim.
print(dat[1:3, ], row.names=FALSE)                # top rows
```

```
      date heat power
2015-02-04   45    16
2015-02-06   45    16
2015-02-09   42    15
```

- One can also read from ”clipboard” or pass an URL

Basic inspection

```
# number of records
nrow(dat)
```

```
[1] 35
```

```
# sum over rows
colSums(dat[,2:3])
```

```
 heat power
 1081   391
```

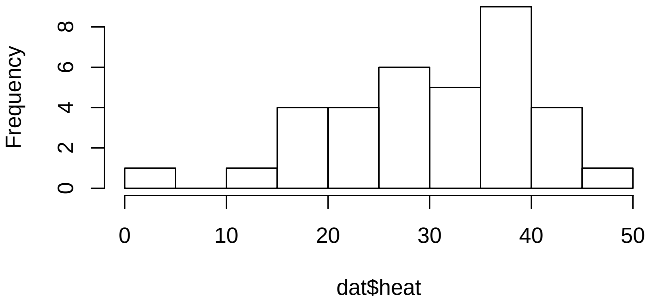
```
# data range of a vector
range(dat$power)
```

```
[1]  4 16
```

```
summary(dat$heat)              # summary for numeric data

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   3.00   24.00   32.00   30.89   38.00   46.00
```

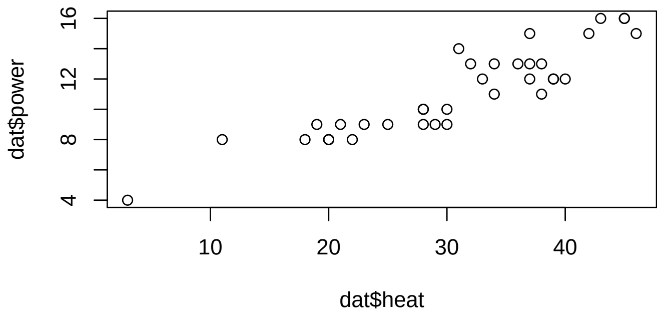
Histogram

```r
hist(dat$heat, main="")
```



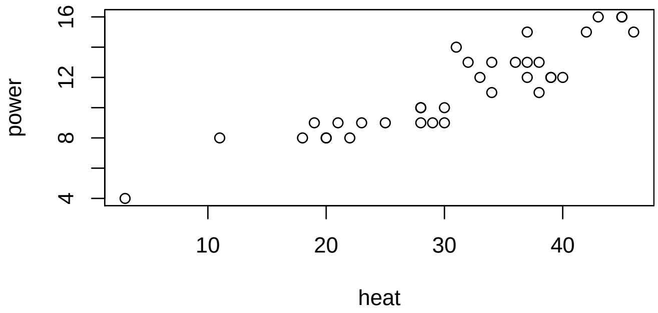
Scatter plot

```
plot(x=dat$heat, y=dat$power)
```



Simplified access to columns

```
with(dat, {
  plot(x=heat, y=power)
})
```



Adding a derived column

```r
dat$year <- as.numeric(substr(dat$date,1,4))
print(dat[1:3, ], row.names=FALSE)
```

```
      date heat power year
2015-02-04   45    16 2015
2015-02-06   45    16 2015
2015-02-09   42    15 2015
```

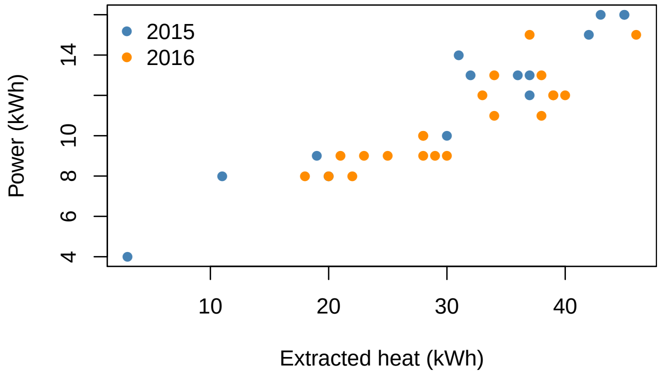
Labels, colors, legend, ...

```r
# Function to assign colors to years; vector-valued
f <- function(x) {
  col <- rep("black", length(x))
  col[x == 2015] <- "steelblue"
  col[x == 2016] <- "darkorange"
  col
}
```

```r
plot(x=dat$heat, y=dat$power, pch=16, col=f(dat$year),
  xlab="Extracted heat (kWh)", ylab="Power (kWh)")
legend("topleft", bty="n", pch=16, col=f(range(dat$year)),
  legend=range(dat$year))
```

Labels, colors, legend, ...

Labels, colors, legend, ...



Important methods for plotting

```
?plot                # primary plot method
```

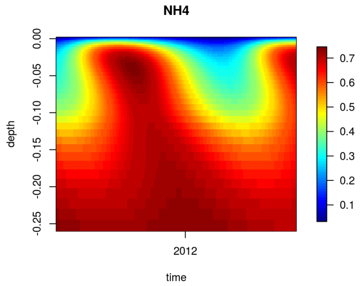
```
?barplot             # often used high-level methods
?boxplot
```

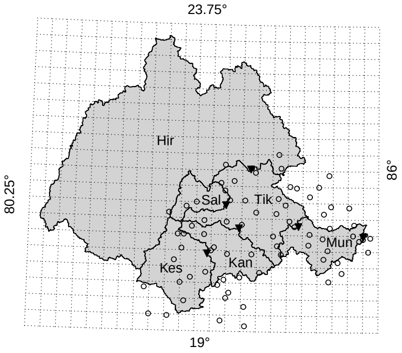
```
?points              # add data to existing plot
?lines
?polygon
```

```
?legend              # legend
?text                # text labels; also 'mtext'
?par                 # fine-tune settings
```

More fancy plot examples

Simulated dynamics of NH_4^+ (mmol L^{-1}) in a lake sediment





River basin with remote sensing grid

Outline

Example data

Height of trees with different exposure to sunlight

Data import

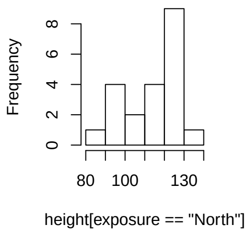
```
trees <- read.table(file="../data/trees.txt",
  sep="\t", header=TRUE)
print(trees[1:3, ], row.names=FALSE)          # rows 1-3
```

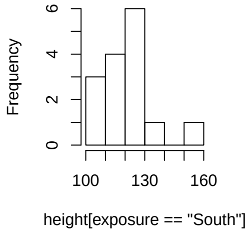
```
exposure height
   North    135
   North    118
   North    121
```

`exposure`	'North' or 'South', interpreted as factors
`height`	Plant height (cm)

Histograms and means

```
par(mfrow=c(1,2))                                  # 2 plots/row
with(trees, {
  hist(height[exposure=="North"], main="")
  hist(height[exposure=="South"], main="")
  tapply(height, exposure, mean)                   # Group means
})
```

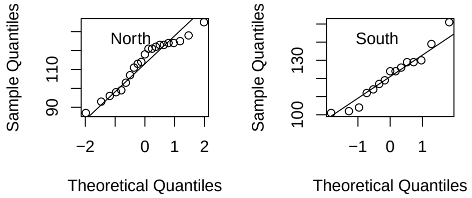




```
   North    South
113.5714 121.4000
```

Visual check for normality

```
par(mfrow=c(1,2))                                    # 2 plots/row
with(trees, {
  for (group in unique(exposure)) {
    qqnorm(height[exposure == group], main="")
    qqline(height[exposure == group])
    legend("topleft", bty="n", legend=group)
  }
})
```



Concepts of testing

Null hypothesis, H_0	Hypothesis you try to reject, typically one of – sample *a* drawn from same population as *b* – variable *a* is unrelated to variable *b*
Alt. hypothesis, H_a	Experimental hypothesis; Opposite of H_0
Significance level, α	Tolerated probability of incorrect rejection of H_0; often 0.05
p-**value**	Probability of getting the observed (or more extreme) result if H_0 was true.

- If $p < \alpha$ we reject H_0. There is 'reasonable evidence' for H_a.
- No significance: Either H_0 is true <u>or sample size is too small</u>.
- Significant <u>does not necessarily mean relevant</u>.

Concepts of testing

Probabilities of correct decisions and errors

True hypothesis

Accepted hypothesis	H_0	H_a
H_0	$1-\alpha$	β
H_a	α	$1-\beta$

- Probability α ('false positive') equals chosen level of signif.
- Probability β ('false negative') depends on α, sample size, type of test
- Can only minimize α or β, $\rightarrow$ Problem-specific choice
- Analysis of 'power' ($1-\beta$) is important aspect of study design

Performing the test

```
?t.test
```

Without checking / assuming equal variance

```
t.test(height ~ exposure, data=trees)
```

```
Welch Two Sample t-test

data:  height by exposure
t = -1.7106, df = 29.341, p-value = 0.0977
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -17.183696   1.526553
sample estimates:
mean in group North mean in group South
           113.5714            121.4000
```

Interpretation of *p*-value

```
Welch Two Sample t-test

data:  height by exposure
t = -1.7106, df = 29.341, p-value = 0.0977
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -17.183696   1.526553
sample estimates:
mean in group North mean in group South
           113.5714            121.4000
```

- $p > 0.05$: We can't reject H_0 at the 5% level.
- Note that the 95% CI (refers to the difference in means) includes the value of zero

Power analysis

Power $(1 - \beta)$ is the probability of detecting an existing effect

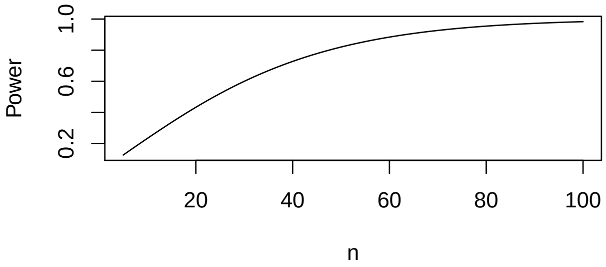
```
?power.t.test
```

Typical applications

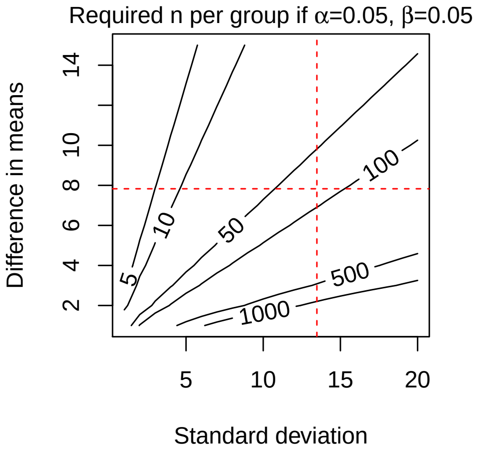
- How large is $1 - \beta$ for a particular test problem?
- What is the required sample size *n* to achieve a certain power if α is set and characteristics of the samples are known?

Power analysis

```
with(trees, {
  x <- power.t.test(n=seq(from=5, to=100, by=1),
    delta= diff(tapply(height, exposure, mean)),
    sd= mean(tapply(height, exposure, sd)),
    sig.level=0.05, power=NULL)
  plot(x$n, x$power, type="l", xlab="n", ylab="Power")
})
```



Power analysis



Red lines refer to tree data

Power analysis

Code to produce the contour plot on previous slide

```r
delta_mu= seq(1, 15, 1)  # Assumed differences in means
sdev= seq(1, 20, 1)      # Assumed standard deviations
alpha= 0.05              # Level of significance
beta= 0.05               # Desired beta error (1-power)
```

```r
# Empty matrix to store required sample size
n <- matrix(NA, ncol=length(delta_mu), nrow=length(sdev),
  dimnames=list(sdev, delta_mu))
```

Power analysis

```
# Compute cells
for (icol in 1:ncol(n)) {     # iterate over columns
  for (irow in 1:nrow(n)) {   # iterate over rows
    n[irow,icol] <- power.t.test(n=NULL,
      delta=as.numeric(colnames(n)[icol]),
      sd=as.numeric(rownames(n)[irow]),
      sig.level=alpha, power=1-beta)$n
  }
}
```

Comparison of two samples' means

Power analysis

```
# Create contour plot from matrix (log scale)
levels= log10(c(5,10,50,100,500,1000))
contour(x=as.numeric(rownames(n)),
  y=as.numeric(colnames(n)), z=log10(n),
  xlab="Standard deviation", ylab="Difference in means",
  levels=levels, labels=10^levels, labcex=1)
mtext(substitute(paste("Required n per group if ",
  alpha,"=",a,", ",beta,"=",b), list(a=alpha, b=beta)))
```

```
# Highlight required sample size for tree problem
abline(h=with(trees, diff(tapply(height, exposure,
  mean))), col="red", lty=2)
abline(v=with(trees, mean(tapply(height, exposure,
  sd))), col="red", lty=2)
```

Outline

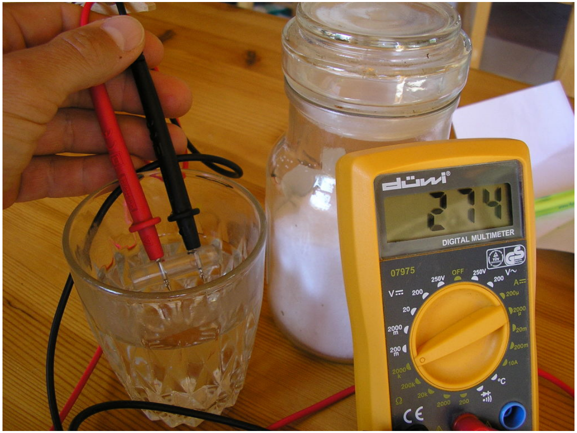
Structure of linear models

Linear model $y = a + b_1 \cdot x_1 + b_2 \cdot x_2 \ldots + b_n \cdot x_n + \epsilon$

y	Dependent variable; response
x	Independent variables; predictors
a	Intercept
b	Coefficients for predictors; slopes
ϵ	Random error; residuals

Simple case $y = a + b \cdot x + \epsilon$

Example data



Electric conductance vs. salt concentration

Data import / preparation

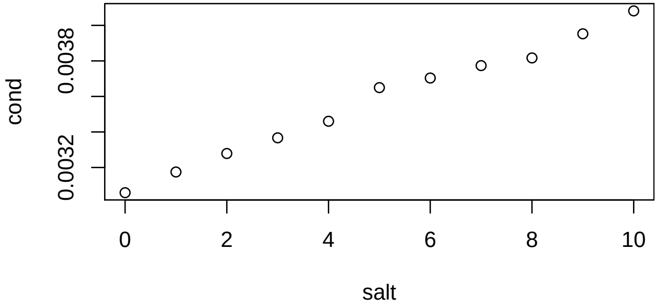
```
dat <- read.table(file="../data/salt.txt",
  sep="\t", header=TRUE)
dat$cond <- 1 / dat$resist                  # get conductance
print(dat[1:3, ], row.names=FALSE)          # show top rows
```

```
salt resist        cond
   0    327 0.003058104
   1    315 0.003174603
   2    305 0.003278689
```

`salt`	Dissolved amount of NaCl (unspecified unit)
`resist`	Measured electrical resistance (Ω)
`cond`	Electrical conductance (S)

Visual check for linearity

```
with(dat, plot(x=salt, y=cond))
```



Estimate coefficients (OLS)

```r
fit <- lm(cond ~ salt, data=dat)
summary(fit)
```

```
Call:
lm(formula = cond ~ salt, data = dat)

Residuals:
       Min         1Q     Median         3Q        Max
-5.472e-05 -1.648e-05 -3.245e-06  6.836e-06  7.541e-05

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 3.079e-03  1.956e-05  157.44  < 2e-16 ***
salt        9.909e-05  3.305e-06   29.98  2.5e-10 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 3.467e-05 on 9 degrees of freedom
Multiple R-squared:  0.9901, Adjusted R-squared:  0.989
F-statistic: 898.8 on 1 and 9 DF,  p-value: 2.499e-10
```

Interpretation

```
              Estimate Std. Error t value  Pr(>|t|)
(Intercept) 0.0030790  1.956e-05  157.40 8.556e-17
salt        0.0000991  3.305e-06   29.98 2.499e-10
```

Interpretation of coefficients

- Conductance of tap water is $\approx$ 3 mS
- Conductance increases by $\approx$ 0.1 mS per unit of added NaCl

Interpretation of *p*-values

- Null hypothesis (H_0): Regression coefficients are zero.
- *p*-value: Probability of observing the data if H_0 was true.

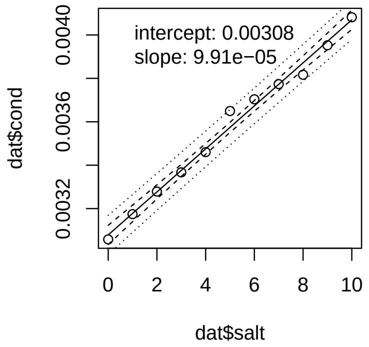
Plotting regression results

```
plot(dat$salt, dat$cond)                         # scatter
lines(dat$salt, predict(fit, interval="none"))  # add line
```

```
fitted <- signif(coef(fit), 3)                   # extract
names(fitted) <- c("intercept", "slope")         #   coeff.
legend("topleft", bty="n", legend=paste(names(fitted),
  fitted, sep=": "))
```

```
x <- data.frame(salt=pretty(dat$salt, 100))
confLim <- predict(fit, newdata=x, interval="conf")
predLim <- predict(fit, newdata=x, interval="pred")
for (lim in c("lwr", "upr")) {
  lines(x$salt, confLim[,lim], lty=2)
  lines(x$salt, predLim[,lim], lty=3)
}
```

Plotting regression results



Prediction limits (dotted)
For a specific X, the true Y is within limits with $p \geq 1 - \alpha$.

Confidence limits (dashed)
Regression line is within the limits with $p \geq 1 - \alpha$.

Default α is 0.05.

Confidence limits of coefficients

Standard method

```
confint(fit)
```

```
                   2.5 %       97.5 %
(Intercept) 3.034516e-03 0.0031229905
salt        9.161782e-05 0.0001065728
```

Simple bootstrap for comparison

```
library(boot)
f <- function(x, i) {
  coef(lm(x$cond[i] ~ x$salt[i], data=x))}
cf <- boot(dat, f, R=1000)$t                       # 1000 x
t(apply(cf, 2, quantile, probs=c(0.025, 0.975)))
```

```
             2.5%        97.5%
[1,] 3.061697e-03 0.0031182366
[2,] 9.317326e-05 0.0001040553
```

Assumptions and how to verify

1. Linear relation between predictor(s) and response
   $\rightarrow$ Inspect scatter plot

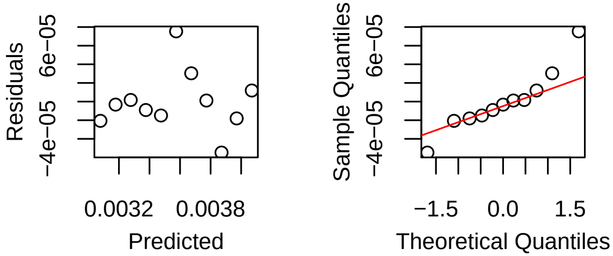
2. Independence of residuals, esp. in time-series data
   $\rightarrow$ Understand how data were produced
   $\rightarrow$ Check auto-correlation

3. Constant variance of residuals (Homoscedasticity)
   $\rightarrow$ Plot residuals over predicted values
   $\rightarrow$ Try transformation if necessary

4. Gaussian residuals
   $\rightarrow$ Check Q-Q-Plot

Inspection of residuals

```
par(mfrow=c(1,2))                          # 2 plots/row
plot(predict(fit), residuals(fit),         # Homoscedasticity?
  xlab="Predicted", ylab="Residuals")
qqnorm(fit$residuals, main="")             # Normal QQ plot
qqline(fit$residuals, col="red")           # Through quartiles
```

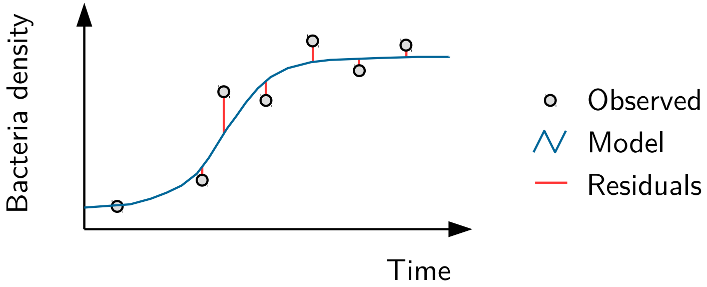


Beyond simple linear regression

- Multiple linear regression
  $\rightarrow$ use `lm` with more than one predictor
- Generalized linear models
  $\rightarrow$ e. g. for non-numeric predictors or response
  $\rightarrow$ see `glm`
- Non-linear fitting
  $\rightarrow$ see `nls`, `nlm`, `optim`

Outline

Example problem



Objective: Find model parameters that minimize residuals

Data import

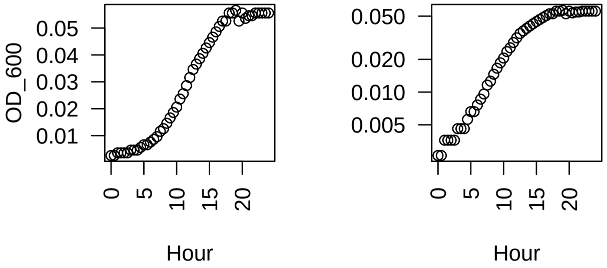
```
d <- read.table(file="../data/growth.txt",
  sep="\t", header=TRUE)
print(d[1:3, ], row.names=FALSE)
```

```
time    dens
 0.0 0.00261
 0.5 0.00261
 1.0 0.00361
```

`time`	Hours after start of experiment
`dens`	Optical density at 600 nm – indicator of *E. coli* concentration in liquid culture – relation is linear within observed range

Plot dynamics

```
par(mfrow=c(1,2))
plot(d$time, d$dens, xlab="Hour", ylab="OD_600", las=2)
plot(d$time, d$dens, log="y", xlab="Hour", ylab="", las=2)
```



Log-scale plot suggests delayed growth in early stage (lag phase)

Proposed differential eqn.

$$\begin{aligned}
\tfrac{d}{dt}Y_a &= g \cdot Y_a \cdot \left(1 - \frac{Y_a + Y_i}{K}\right) + w \cdot Y_i \\
\tfrac{d}{dt}Y_i &= - w \cdot Y_i
\end{aligned}$$

Growth — Resource limitation — 'Wakeup'

State variables

Y_a	Conc. of active cells
Y_i	Conc. of inactive cells

Parameters

g	Max. growth rate (h^{-1})
w	Wakeup rate (h^{-1})
K	Carrying capacity

Note: Measured optical density reflects $Y_a(t) + Y_i(t)$

Methods I: Numerical integration

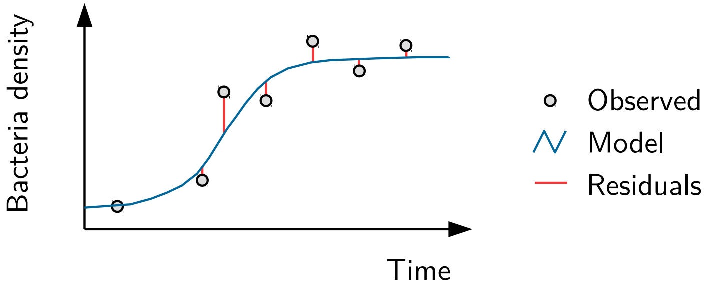
Numerical integration is required to obtain the dynamics $Y_a(t)$ and $Y_i(t)$ from the given differential equations.

```
install.packages("deSolve")           # install ODE solvers
library("deSolve")                    # load solvers
```

A **function returning the derivatives** must be passed to solver

```
model <- function(time, y, p) {       # deSolve compliant
  list(c( dYi = -p["w"] * y["Yi"],    # y: state vector
          dYa = p["w"] * y["Yi"] +    # p: param. vector
                p["g"] * y["Ya"] *
                (1 - (y["Ya"] + y["Yi"]) / p["K"])
  ))
}
```

Methods II: Optimization



- Optimization means to minimize an **objective function**.
- Here, the objective function computes the sum of squared residuals for a given set of model parameters.

Methods II: Optimization

Objective function

```
objFunc <- function(p, obs) {
  # get Ya(t) and Yi(t) for current param. (integration)
  sim <- deSolve::ode(y=c(Yi=p[["Yi0"]], Ya=p[["Ya0"]]),
    time=obs$time, func=model,
    parms=c(w=p[["w"]], g=p[["g"]], K=p[["K"]]))
  # compare Ya(t) + Yi(t) to observations
  sum((obs$dens - apply(sim[,2:3], 1, sum))^2)
}
```

- Vector of parameters is 1^{st} arg., for compliance with optimizer
- Initial values of Y_a, Y_i estimated along with actual parameters

Methods II: Typical optimization issues

Discontinuities	Objective function returns invalid result for certain parameter values, e. g. zero-division $\rightarrow$ Need to use box-constraints
Non-identifyability	– Few data with insufficient variability – Colinearity of parameters
Local minima	Solution is sensitive to initial guess $\rightarrow$ Needs clever estimate or stochastic search
Scaling required	If parameters are of different magnitude

Initial guess

```
guess <- c(Yi0=0.001, Ya0=0.001, w=0.1, g=0.3, K=0.06)
```

- K can be estimated from plot (density at saturation)
- Sum $Y_i + Y_a$ visible in plot as well (assume, e. g., 50:50)
- Reasonable first guess for w might be $\frac{1}{2}g$
- g can be estimated from linear part of log-scale plot

$$Y(t_1) = Y(t_0) \cdot e^{g(t_1 - t_0)} \qquad \rightarrow \qquad g = \frac{ln(Y(t_1)) - ln(Y(t_0))}{t_1 - t_0}$$

Estimating the parameters

Optimization with box-constraints

```
fit <- optim(par=guess, fn=objFunc, gr=NULL, obs=d,
  method="L-BFGS-B",
  lower=c(Yi0=0, Ya0=0, w=0, g=0, K=0.04),
  upper=c(Yi0=Inf, Ya0=Inf, w=Inf, g=Inf, K=0.08),
  control=list(parscale=guess))
```

Essential check if used in a script

```
if (fit$convergence != 0)
  stop("fitting failed")
```

Result

```r
print(fit)
```

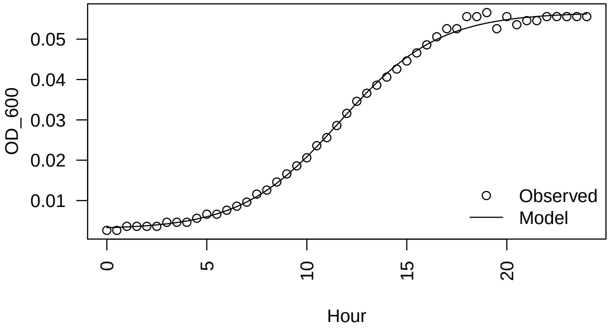
```
$par
      Yi0       Ya0         w         g         K 
0.0029460 0.0003339 0.0239800 0.4198000 0.0566100 

$value
[1] 3.855324e-05

$counts
function gradient 
      49       49 

$convergence
[1] 0
```

Result

Re-run model with best-fit parameters for plotting

```
par(mfrow=c(1,1))
with(as.list(fit$par), {
  int <- deSolve::ode(y=c(Yi=Yi0, Ya=Ya0),
    time=d$time, func=model, parms=c(w=w, g=g, K=K))
  plot(d$time, d$dens, xlab="Hour", ylab="OD_600", las=2)
  lines(int[,1], apply(int[,2:3], 1, sum))
  legend("bottomright", bty="n", pch=c(1,NA),
    lty=c(NA,1), legend=c("Observed", "Model"))
})
```

Result



Need to fit growth rates?

Consider the R package written by Thomas Petzoldt

https://cran.r-project.org/package=growthrates

Outline

General recommendations

- Keep data organized
  $\rightarrow$ learn the basics of data base design
  $\rightarrow$ always request and store meta data
- Increase efficiency and transparency by writing scripts
  $\rightarrow$ R is just one option
  $\rightarrow$ Use functions to facilitate re-use and debugging
- Plot your data, preferably in multiple ways
- Understand the meaning and limitations of p-values
- Only trust in transparent statistics

The ANSWER project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 675530.