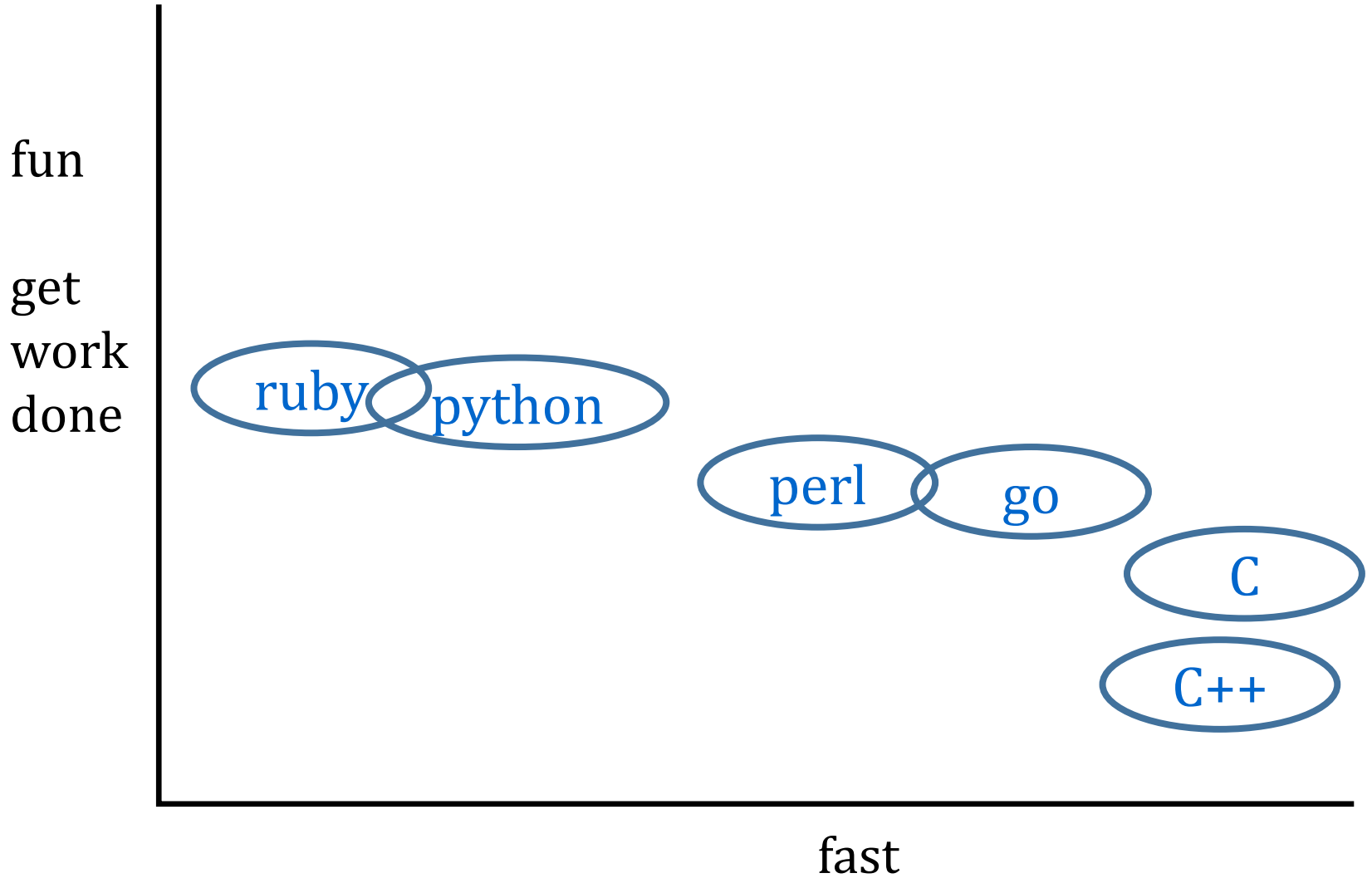
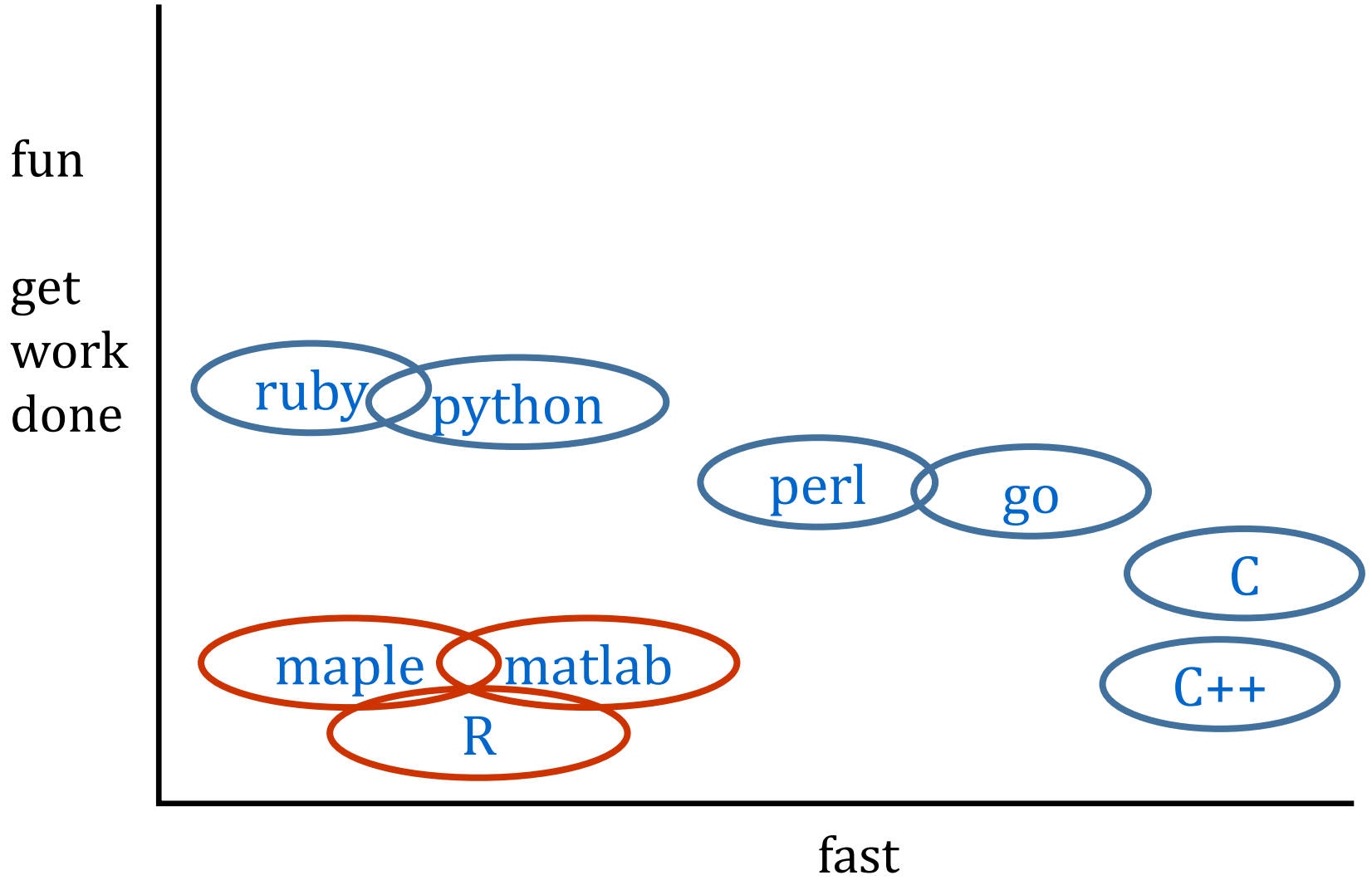
R and a bit of statistics

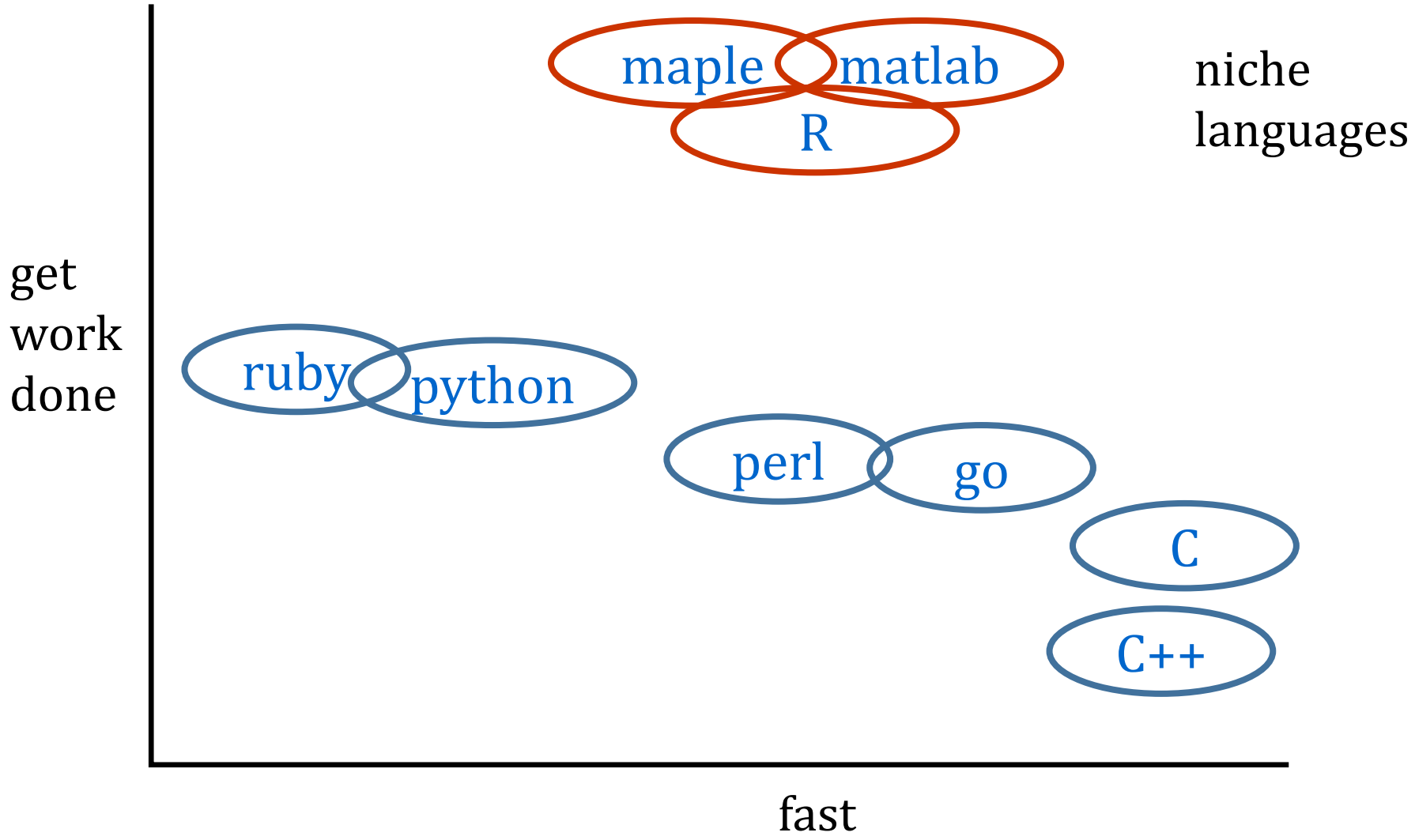
why ?



Needleman & Wunsch / parser for mmcif (X-ray) files



invert matrix, symbolic maths, curve fitting, plotting



R gives you

- data structures suited to vectors, matrices
  - with appropriate basic operations
- every imaginable distribution
- data fitting
- graphics

Horrible syntax `a[2]` `a[[2]]`

Starting and stopping

- `R, q()`

Will you get desperate and angry ?

- ask why are there google style guidelines ? a microsoft optimised release ?

Typical uses

Real data sets

- read, filter, plot, fit, look for correlations…

Playing

- generate a distribution, sample, plot, add noise, ..

Use

Lots of interfaces

- Rstudio – works on our teaching pool
- Rcmdr - works on our teaching pool
- eclipse
- emacs ESS mode

Code / Scripts / Interactive – not just interpreter

C	write, compile, run	compiler
Python	write, run	interpreter
R	play line by line + write, run	interpreter a bit more ..

interpreter, interactive ?

classic scripting		no programming
use an editor / write something	`R`	`Rcmdr, Rstudio`
complex	play with command line	Read +display data, like spreadsheet
run `Rscript`		do manipulations from GUI

Structural differences

C, C++, python

- cannot add arrays, what does "+" do in C++ ? Add ? Concatenate ?
- R: vectors, lists, matrices behave like vectors and matrices

```
> a = c(1, 2, 3)
> b = c(4, 5, 6)
> c = a+b
> c
[1] 5 7 9
```

Do not write `for (i in x) { ...`

Speed / Memory

Speed

`a <- b * c` or **`x %*% y`** or big vectors / matrices – very fast

- code is recognised, runs hand-crafted routines

```
> for (i in 1:length(a)) { a[i] = b[i] * c[i]}
```

as slow as python

Memory

- like python, perl, ..
  - garbage collected - no memory leaks
  - quite a bit of overhead
- sometimes lots of non-obvious memory – correlations, plots
- easy to make crazy inefficient constructs

self study

- install.packages('swirl') #download swirl package
- library(swirl) #load in swirl package
  - cheesy, but effective

Essential

`apropos('etwas')`

`?etwas`

- google R etwas
- Vorsicht – documentation is very formal
- Built-in data sets – often referred to in examples
- **`iris`**, **`mtcars`**, .. type **`data()`**

Packages

base R = what you get from compiling R distribution

- many popular extensions
- these lectures – base R (3.X oder 4.0)

Packages

- 10^5 R packages on CRAN – good quality control, well supported
  - 10^2 on our machines **`installed.packages()`**
- **`cran.r-project.org`** plotting, advanced statistics, machine learning
- compared to C libraries
  - your matrix implementation is different to mine – try using gsl

Technical…

Operators

`a <- 1` R people like **`<-`** do not write **`a = 1`**

`+ - * /` no surprises – binary operators work on vectors and matrices (element by element – not algebra)

logical operators

>, <, … |, &

Other operations handled by base functions (base = built-in)

- **`mean()`**, **`max()`**, **`median()`**, **`sum()`**, ..
  - if you are looking for this kind of common operation
    - look for a built-in – faster than the one you build

Data Structures

- scalars
- vectors
- matrices
- lists
- data frames

Scalars

- logical

```
> v <- TRUE
> v
[1] TRUE
> str(v)
 logi TRUE
```

- int
- numeric
- complex
- character

```
> v <- "TRUE" ; v ; str (v)
[1] "TRUE"
chr "TRUE"
```

  - types are automatically chosen

Vectors

first a little function, **`c()`**

- **`?c`** will tell you what it does **`The default method combines its arguments to form a vector..`**

```
> x <- c(1, 2, 3) ; str (x)
 num [1:3] 1 2 3
```

- indexing from 1 (not zero)
- all elements same type (all float, all int, all logical, ..)

- where do they come from ?
- **`vector()`** ? **`as.vector()`** coming
- data you read in – extract vectors (columns, rows)

Vectors - Accessing elements

accessing elements..

- **`x[1]`** first element
- **`x[-4]`** everything except fourth element
- **`x[2:4]`**
- **`x[:4]`** elements, 1, 2, 3, 4
- logical versions – compact filtering

```
> x <- c(1, 2, 3, 4, 5) ; x[x>=3]
[1] 3 4 5
```

matrices

```
m <- matrix(x, nrow = 3, ncol = 4)
```

but more often from

- a data set
- from a calculation like a correlation matrix
- putting vectors together

```
> x <- c(4, 5, 6) ; y <- c(7, 8, 9)
> m <- cbind(x, y) ; m
     x y
[1,] 4 7
[2,] 5 8
[3,] 6 9
> n <- rbind(x, y); n
  [,1] [,2] [,3]
x    4    5    6
y    7    8    9
```

matrix access

- `m[2, 3]` an element
- `m[, 3]` third column (vector)
- `m[1,]` first row
- logical access – perverse but works

```
x <- c(4, 5, 6) ; y <- c(5, 8, 9)
> m <- cbind(x, y) ; m
     x y
[1,] 4 5
[2,] 5 8
[3,] 6 9
> m[m==5]
[1] 5 5
```

lists

- not vectors
- mixed types

```
> a <- 'a word'; b <- 1.0; c <- TRUE; d <- c(1, 2, 3)
> l <- list(a, b, c, d) ; str (l)
List of 4
 $ : chr "a word"
 $ : num 1
 $ : logi TRUE
 $ : num [1:3] 1 2 3
```

- group things that are related, but different
- often used for control, functions

elements in a list

```
> l <- list(x = 1:5, y = c('a', 'b'))  ; l
$x
[1] 1 2 3 4 5
$y                    # There are two things in l, both are vectors
[1] "a" "b"
```

access

```
l[2]
$y
[1] "a" "b"
```

- there are double and single brackets `[]`, `[[]]`

```
> str (l[[1]])
 int [1:5] 1 2 3 4 5   # elements from [[..]]
> str (l[1])
List of 1
 $ x: int [1:5] 1 2 3 4 5 # a new list from [..]
```

data frames

```
$ cat data.txt
andrew mary
3 4
5 6
7 8
```

Very foreign to C, other languages

- hash + array ? more generally hash + more general type

Very natural for data

```
> df <- read.table("data.txt", header = TRUE)
> df
  andrew mary
1      3    4
2      5    6
3      7    8
> df$mary
[1] 4 6 8
> str (df$mary)
 int [1:3] 4 6 8    # a vector
```

data frames

```
$ cat data.txt
andrew mary
3 4
5 6
7 8
```

If you must, get to rows,

```
> df[2,]
  andrew mary
2  5     6   # a data frame
```

You like scalars, vectors, matrices

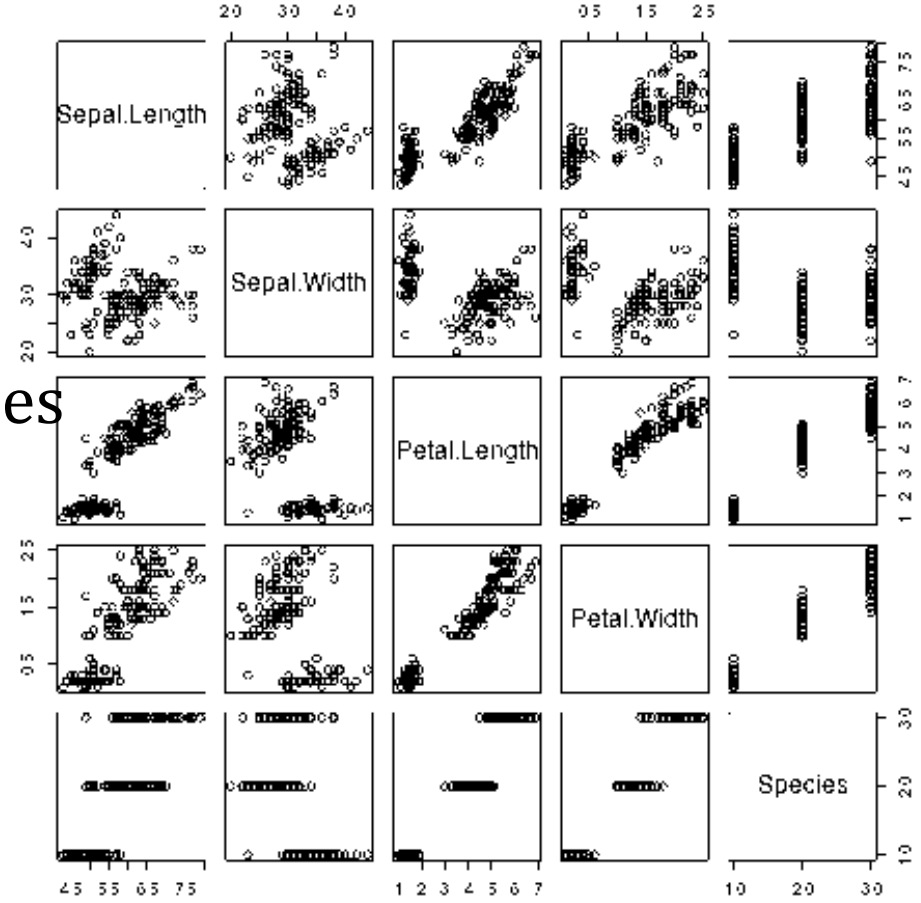
- can you avoid data frames ? No
- ¾ of the time, **`df$a`**, **`df$b`** will do

Why must I learn data frames ?

- result of **`read.table()`** and friends
- R has remarkable defaults…

```
> iris    #  data set that comes with R – properties of some flowers
    Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
1            5.1         3.5          1.4         0.2     setosa
2            4.9         3.0          1.4         0.2     setosa
…
> pairs (iris)
```

- eats a data frame,
- plots all possible pairs
- no tricks – default plot, default axis scales
- tells me
  - who is correlated with who
  - I should replot without species ?



language syntax - for, while, if – no surprises

```
for (i in 1:4) {          #  Vorsicht  not for (i=0; i<4; i++)
     i <- i + 10
     print (i)
}
-------------------------------------
i <- 0
while (i < 5){
     print(i)
     i <- i + 1
}
---------
if (i > 3) {
    print('Yes')
} else {
    print('No')
}
```

functions

```
junk$ Rscript z.r
answer is  18
```

- pass by value
- very often operate on whole vectors

```
$ cat z.r
addup <- function (x) {
     s <- 0
     for (i in x) {
          s = i + s
     }
     return (s)
}

b <- c(5, 6, 7)
t <- addup (b)
cat ("answer is ", t, "\n")
```

Built in functions

- many × 10^2
- expected maths – trigonometry, logarithms – easy, act on vectors
- type manipulation **`as.vector`**, **`cbind`**, **`rbind`**, .. – foreign and varied
- plotting
- printing (ugly)
- data in / out
- character / text manipulation

- what is the syntax like ? How to read the manual pages ?

function parameters

```
?log
Usage:
      log(x, base = exp(1))
   ...
```

- log() takes two arguments, $\log(x, b)$ so $\log(x, 10)$ is $\log_{10} x$
- there is a default for the second argument so **`log(x)`** is really $\ln x$

- a horrible, but important example

```
?read.table
```

```
read.table(file, header = FALSE, sep = "", quote = "\"'",
      dec = ".", numerals = c("allow.loss", "warn.loss", "no.loss"),
      row.names, col.names, as.is = !stringsAsFactors,
      na.strings = "NA", colClasses = NA, nrows = -1,
      skip = 0, check.names = TRUE, fill = !blank.lines.skip,
      strip.white = FALSE, blank.lines.skip = TRUE,
      comment.char = "#",
      allowEscapes = FALSE, flush = FALSE,
      stringsAsFactors = default.stringsAsFactors(),
      fileEncoding = "", encoding = "unknown", text, skipNul = FALSE)
```

- `read.table('fname')` will often work
- `read.table('fname', skip = 3)` to jump over the first three lines
- …

Plotting

Just the main points

- base R – very clever
  - packages to make it more beautiful **`library(ggplot2)`**, **`library(tidyverse)`**
- types ?
  - lines, points, boxes
  - histogramming
  - box +whiskers
  - contours

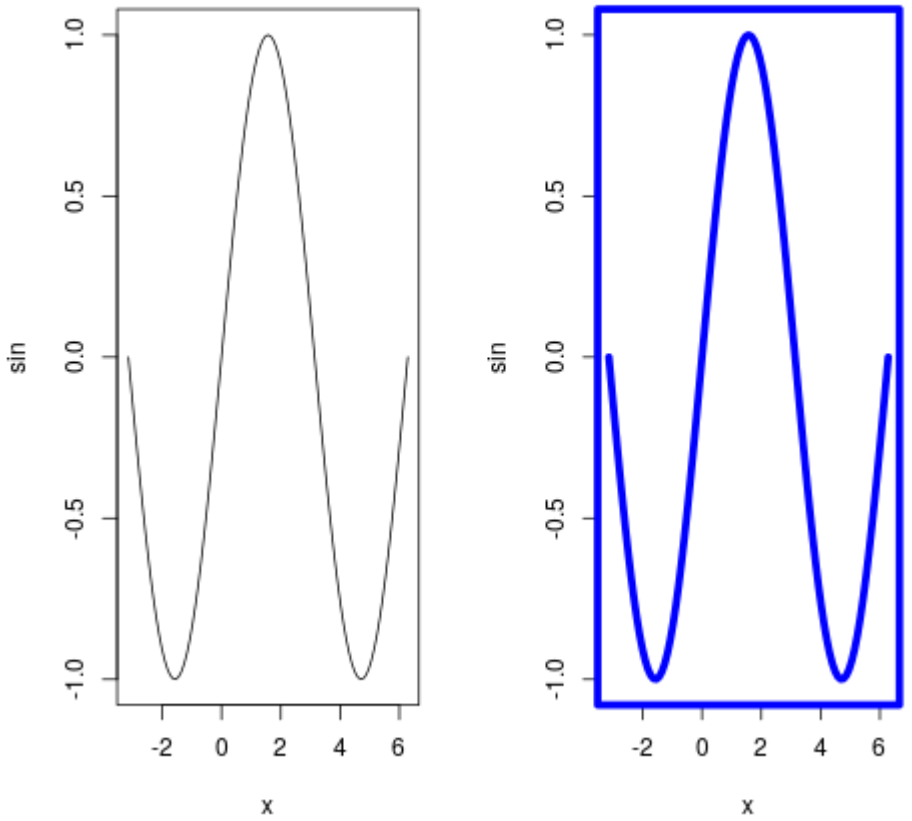
What are the surprises ?

plot complications parameter

Syntax is not pretty – so many parameters

- call **`par()`** and then plot

```
par (mfcol=c(1,2))
plot (sin, -pi, 2*pi)
par (col = "blue", lwd = 5)
plot (sin, -pi, 2*pi)
```



devices

- do not ever send me a screen dump
- interactive R – no surprises
- usually want output as pdf, png, svg

```
png (file='x.png')
plot (sin, -pi, 2*pi)
dev.off()
```

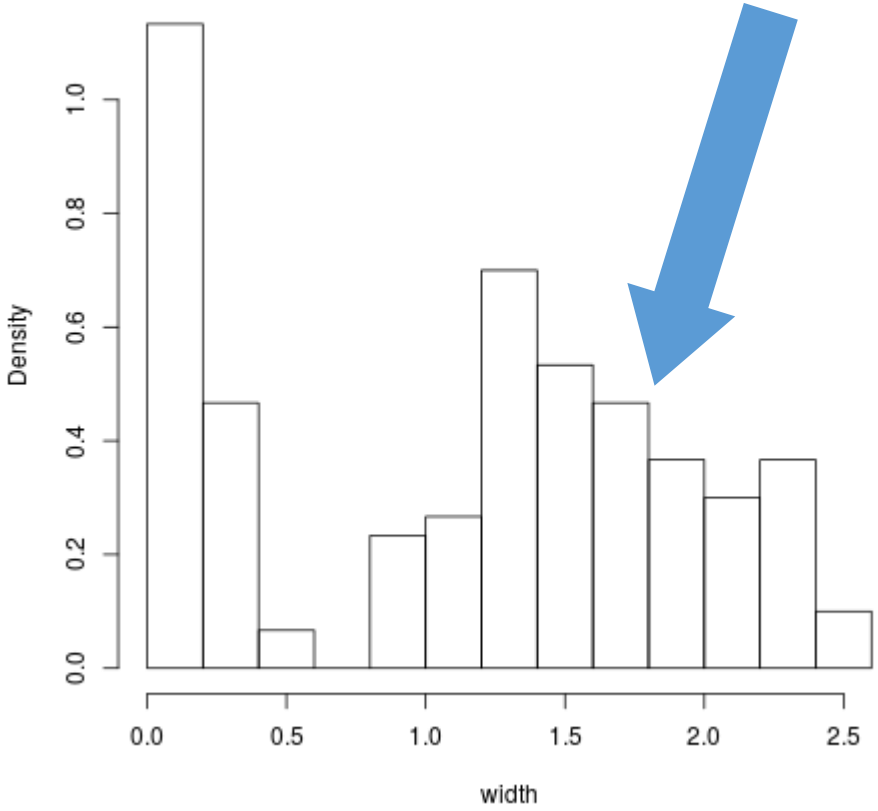
Many

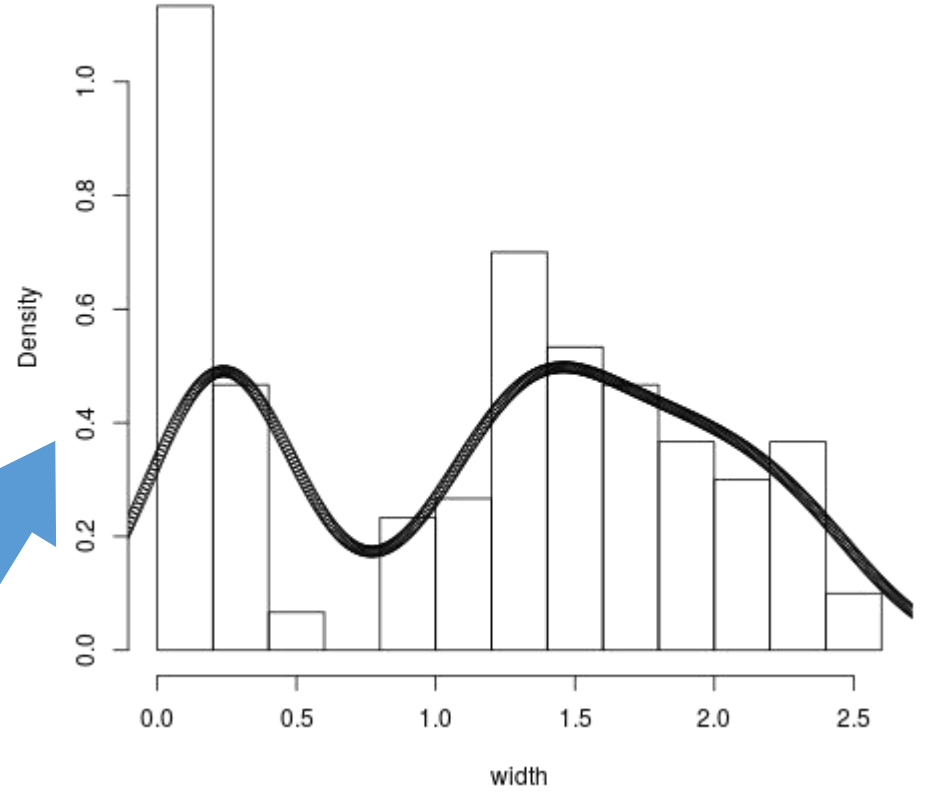
- options
- devices

Plots are often layered

iris is just a test data set

```
hist (iris$Petal.Width, freq=FALSE, main="", xlab="width")
```






```
hist (iris$Petal.Width, freq=FALSE, main="", xlab="width")
points(density (iris$Petal.Width))
```

Enough syntax

Next part

- some statistics, fitting, ..

Examples Programming in R

From C programming to R style

Goal

- two football teams with different averages
- how often does the better team win ?
  - poisson processes

Ingredients / Plan

Some ingredients / the plan

- poisson processes / exponential distribution / time between events
- how to code it
  - naïve – time-based simulation
  - changing distributions
  - C programmer version
  - using R features

Taylor expansion of ln x

Will need (soon)

$$\lim_{n\to\infty}\left(1+\frac{T}{N}\right)^N$$

First I want to know about $\ln(x+1)$

Remember Taylor expansion for some a

$$f(x)=f(a)+f'(a)(x-a)+\frac{f''(a)}{2!}(x-a)^2+\frac{f'''(a)}{3!}(x-a)^3+\cdots$$

for logarithms

$$\ln(x)=\ln a+\frac{x-a}{a}-\frac{(x-a)^2}{2a^2}+\frac{(x-a)^3}{3a^3}+\cdots$$

why ? do not forget $\frac{d}{dx}\ln x=\frac{1}{x}$

from previous slide

$$\ln(x) = \ln a + \frac{x-a}{a} - \frac{(x-a)^2}{2a^2} + \frac{(x-a)^3}{3a^3} + \cdots$$

so

$$\ln(x+1) = \ln(a) + \frac{x+1-a}{a} - \frac{(x+1-a)^2}{2a^2} + \frac{(x+1-a)^3}{3a^2} - \cdots$$

let me set $a = 1$

$$\ln(x+1) = \ln(1) + x - \frac{x^2}{2} + \frac{x^3}{3} - \cdots = 0 + x - \frac{x^2}{2} + \frac{x^3}{3} - \cdots$$

what happens as $x \to 0$?

$$\lim_{x \to 0} (\ln(x+1)) = x$$

will need this later

Uniformly distributed events

Decay of a particle / chemistry

- $\mathrm{A} \rightarrow \mathrm{B} + \mathrm{C}$ long term average is clear $A(t) = A_0 e^{-\lambda t}$
- intuitively
  - in some time ΔT I can talk about the probability of a breakdown
  - if the decay rate λ is high, the probability is higher
  - say time between breakdown events is $\tau = \lambda^{-1}$

We rarely look at individual molecules ($\Delta T \gg \tau$)

- when do we see individual events ?
  - football game (and Geiger counters, ion channels)

Non-Uniformly distributed events

Football

- long term average is clear (1300 goals in 1000 games)
- short term ? very uncertain – no goals, 5 goals are possible
- order of magnitude..
  - $\tau = \frac{T}{N} = \frac{90}{2} = 45$ min (for about two goals scored)

Other systems in biology / chemistry ?

- ion channels in nerves open / close spontaneously (rare, but easy to measure)
- few copies of DNA repressor per cell
  - DNA + protein → (DNA-protein) rare event – hard to see
  - classical chemical kinetics is not helpful

Distribution for these events

Derivation

- Start from average over long T
- divide into $N \times \Delta T$
- get limit as $N \to \infty$ and $\Delta T \to 0$

Nomenclature

- rate $\lambda = {}^{1}/_{\tau}$ the average time between goals / channel opening / ..

Average number of goals in ΔT ? $P(\Delta T) = \lambda \Delta T$

Probability of no goal in some ΔT is $P_0(\Delta T) = (1 - \lambda \Delta T)$

longer time with many ΔT

$$P_0(T) \approx (1-\lambda\Delta T)^N \quad \text{or} \left(1-\frac{\lambda T}{N}\right)^N$$

Result from earlier ... $\lim_{x\to 0} \ln(1+x) = x$

$$P_0(T) \approx \left(1-\frac{\lambda T}{N}\right)^N$$

$$= \lim_{N\to\infty}\left(1-\frac{\lambda T}{N}\right)^N$$

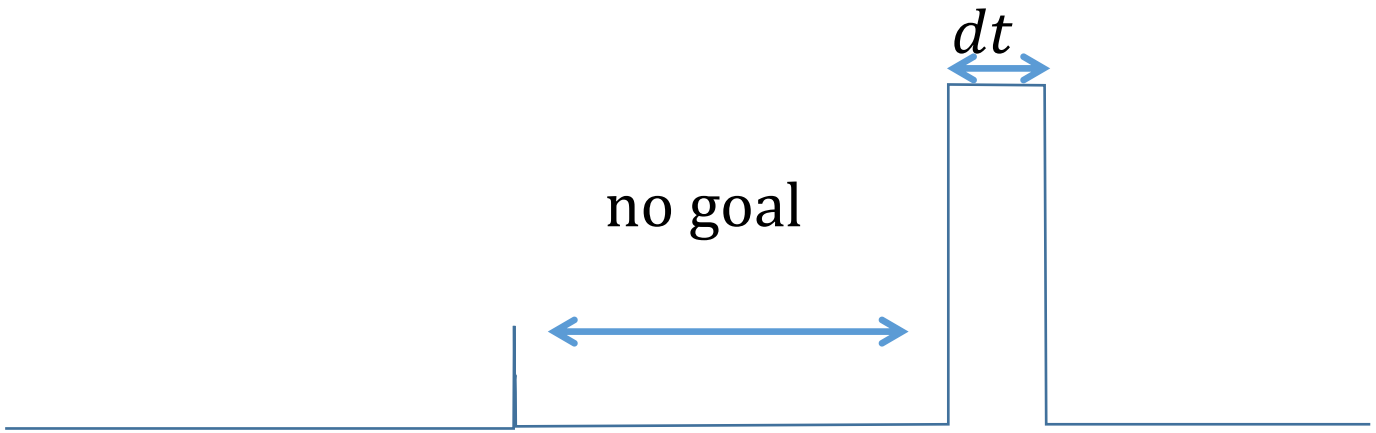
$$= \exp\left(\ln \lim_{N\to\infty}\left(1-\frac{\lambda T}{N}\right)^N\right) = \exp\left(N \lim_{N\to\infty} \ln\left(1-\frac{\lambda T}{N}\right)\right)$$

$$= \exp\left(N\frac{-\lambda T}{N}\right)$$

$$= e^{-\lambda T}$$

The exponential distribution

- probability for no goal $P_0(T) = e^{-\lambda T}$
- check intuition
- exponential distribution – time between events



$I_1 dt$ = probability of no goal in t × probability of goal in dt

$$I_1 dt = P_0(t)\ \lambda\, dt = \lambda e^{-\lambda T} dt$$

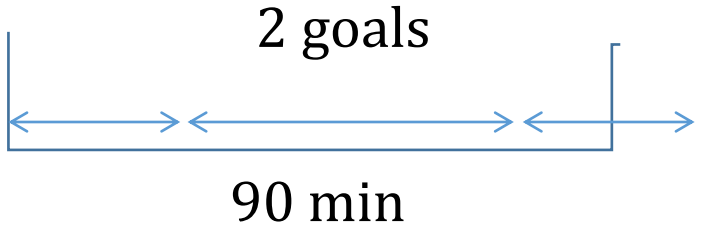
$I_1 = \lambda e^{-\lambda T}$ exponential distribution

Possibility – use exponential distribution

Naïve and inefficient

total rate $\lambda_0 = \lambda_1 + \lambda_2$
set up counters n_1 and n_2
Set up tmp_1 and tmp_2
while $(t < T_{game})$

$n_1 \mathrel{+}= tmp_1; n_2 \mathrel{+}= tmp_2$
$tmp_1 = tmp_2 = 0$
Δt = random_from exponential (λ_0)
decide_who_gets_goal (random based on $\frac{\lambda_1}{\lambda_1+\lambda_2}$)
increment tmp_1 or tmp_2



we can do much better

expected number of goals in t

Start with binomial distribution

- probability of success in one try is p
- I have n tries
- what is the probability of seeing k successes ?

$$P(k|n,p) = \binom{n}{k} p^k (1-p)^{n-k}$$

Remember $\binom{n}{k} = \frac{n!}{k!(n-k)!}$

When do you see this ?

Probability of seeing $k = 5$ heads from $n = 10$ coin tosses with $p = {}^1/_2$

- rate per unit time game λ
- some rules
  - events (goals) are independent
  - events are rare – probability of one in short time t is λt
  - you never see two events in a very short time

- take unit time and divide by n (trials)
- probability in one of n units is $p = {}^{\lambda}/_{n}$
- we are interesting in case of very small p in any one δt

original name binomial $P(k|n,p) = \frac{n!}{k!(n-k)!} p^k (1-p)^{n-k}$

write as $P(X = x) = \frac{n!}{x!(n-x)!} p^x (1-p)^{n-x}$ remember $p = \frac{\lambda}{n}$

consider limit

$$\lim_{n\to\infty} \frac{n!}{x!(n-x)!} \left(\frac{\lambda}{n}\right)^x \left(1-\frac{\lambda}{n}\right)^{n-x} = \lim_{n\to\infty} \frac{n(n-1)\cdots(n-x+1)}{n^x} \frac{\lambda^x}{x!} \left(1-\frac{\lambda}{n}\right)^n \left(1-\frac{\lambda}{n}\right)^{-x}$$

from binomial to poisson

$$\lim_{n\to\infty} \frac{n!}{x!\,(n-x)!}\left(\frac{\lambda}{n}\right)^{x}\left(1-\frac{\lambda}{n}\right)^{n-x} = \lim_{n\to\infty} \frac{n(n-1)\cdots(n-x+1)}{n^{x}}\frac{\lambda^{x}}{x!}\left(1-\frac{\lambda}{n}\right)^{n}\left(1-\frac{\lambda}{n}\right)^{-x}$$

$$\lim_{n\to\infty} \frac{n(n-1)\cdots(n-x+1)}{n^{x}} = \lim_{n\to\infty}\left[\frac{n}{n}\left(1-\frac{1}{n}\right)\cdots\left(1-\frac{x-1}{n}\right)\right] = 1$$

$$\lim_{n\to\infty}\left(1-\frac{\lambda}{n}\right)^{n} = e^{-\lambda} \text{ and } \lim_{n\to\infty}\left(1-\frac{\lambda}{n}\right)^{-x} = 1$$

$$\lim_{n\to\infty} \frac{n!}{x!\,(n-x)!}\left(\frac{\lambda}{n}\right)^{x}\left(1-\frac{\lambda}{n}\right)^{n-x} = \frac{\lambda^{x}e^{-\lambda}}{x!} = P(X=x)$$

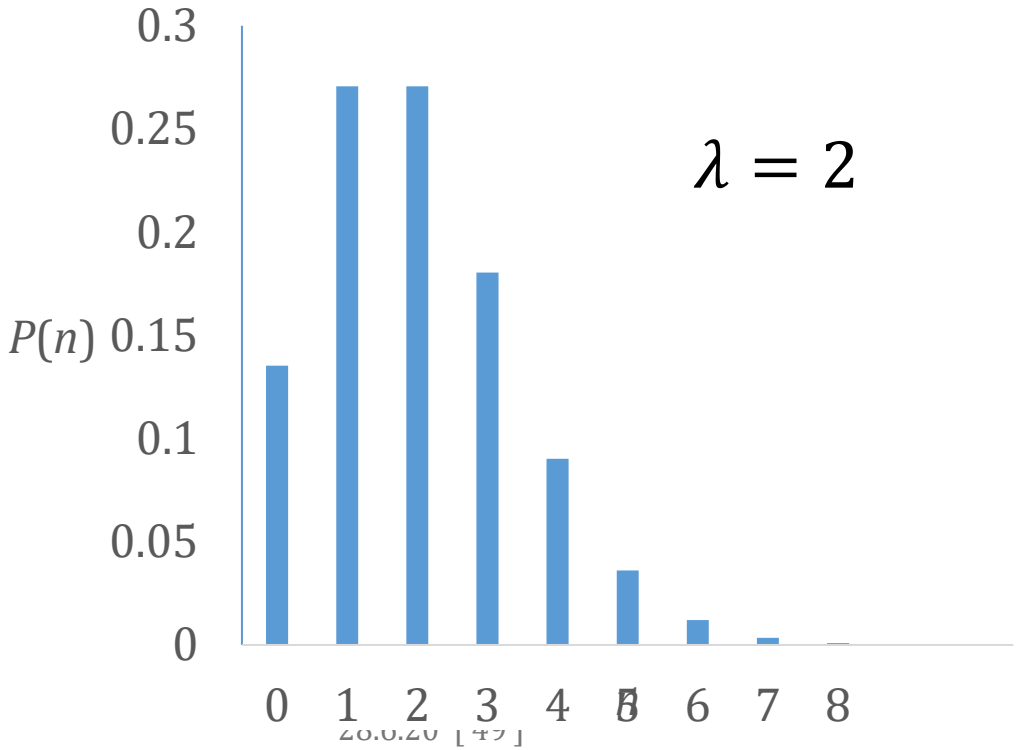
Poisson distribution

$$P(X = x) = \frac{\lambda^x e^{-\lambda}}{x!}$$

From average rate of events (λ)

I can calculate the probability of seeing some number x events

Change simulation strategy



simulation strategy

- look up rate of goals for team 1 (λ_1) and 2 (λ_2)
- say Pois(λ) is a random number drawn from poisson distribution
- a game is

  $n_1 = \text{Pois}(\lambda_1)$ and $n_2 = \text{Pois}(\lambda_2)$
  if $(n_1 > n_2)$ team 1 wins
  elseif $(n_2 > n_1)$ team 2 wins
  else draw

- repeat many times to get probabilities

First approach C style

C programmer's version of football

```
game <- function (mu_1, mu_2)  {
    team1_result <- rpois(1, mu_1)
    team2_result <- rpois(1, mu_2)

    if (team1_result > team2_result) {
        result <- 1
    } else if (team2_result > team1_result) {
        result <- 2
    } else {
        result <- 0
    }
    return (result)
}
```

rpois random number from Poisson distribution

to run the game..

```
result <- c()
for (i in 1:n_games) {
     result <- c(result, game(team1_mu, team2_mu))
}
w1 = length(result[result==1]); w2 = length (result[result==2])
draw = n_games - (w1 + w2)
cat ("team 1", w1, w1/n_games * 100, "%\n")
cat ("team 2", w2, w2/n_games * 100, "%\n")
cat ("draw  ", draw, draw/n_games * 100, "%\n")
```

fancy indexing
select elements
where results is 2

from 100 000 games

```
team 1 28630 28.6 %
team 2 43422 43.4 %
draw   27948 27.9 %
```

- took 10 ½ s can do much better

- games are independent events
- use vectors in R

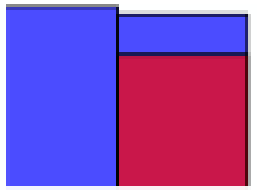
```
team1_mu <- 1.0    # Average number goals per match
team2_mu <- 1.3
n_games <- 100000  # How many games to play

team1 <- rpois(n_games, team1_mu)
team2 <- rpois(n_games, team2_mu)
```

generate 100 000 results in one go
team1 and 2 are long vectors

```
w1 <- sum (team1 > team2)
w2 <- sum (team2 > team1)
draws <- n_games - (w1 + w2)
```

sum over vectors of logicals
team2 > team1 is a long logical vector

From 10 ½ s to 0.13 s (including printing results)
Plot results…

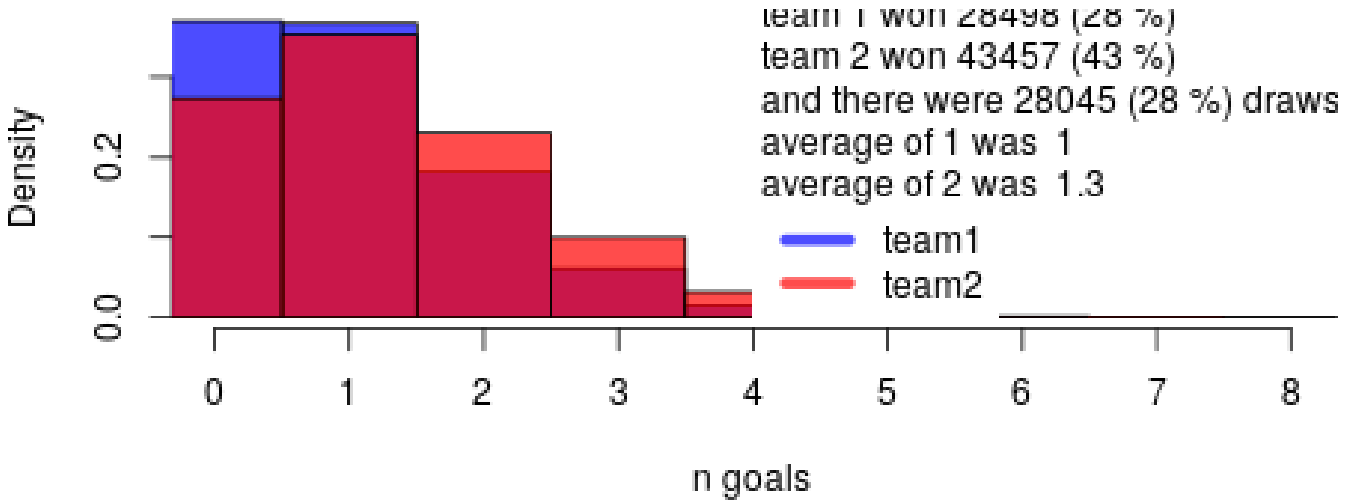


- build up text, put in "a", define colour1, colour2, setup breaks

```
xlim=c(0,9)
hist (team1, breaks=breaks, probability=T, main="",
   col=colour1,xlab= "n goals", ylab = "frequency", xlim=xlim)
hist (team2, breaks=breaks, probability=T, main="",
col=colour2, add=T)
legend(x=3, y=0.25, legend=c("team1", "team2"),
   col=c(colour1, colour2), lwd=5, box.lty=0)
text(a, x=4, y=0.3, adj=0)
```

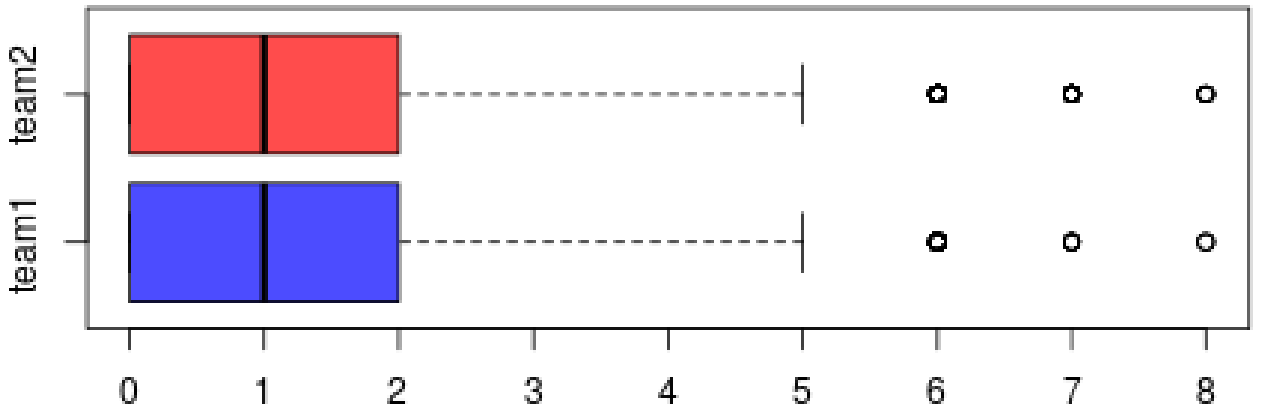
- can make the plot clearer
  - box and whiskers

- remember the scores are in team1 and team2
- add a command for two rows and boxplot



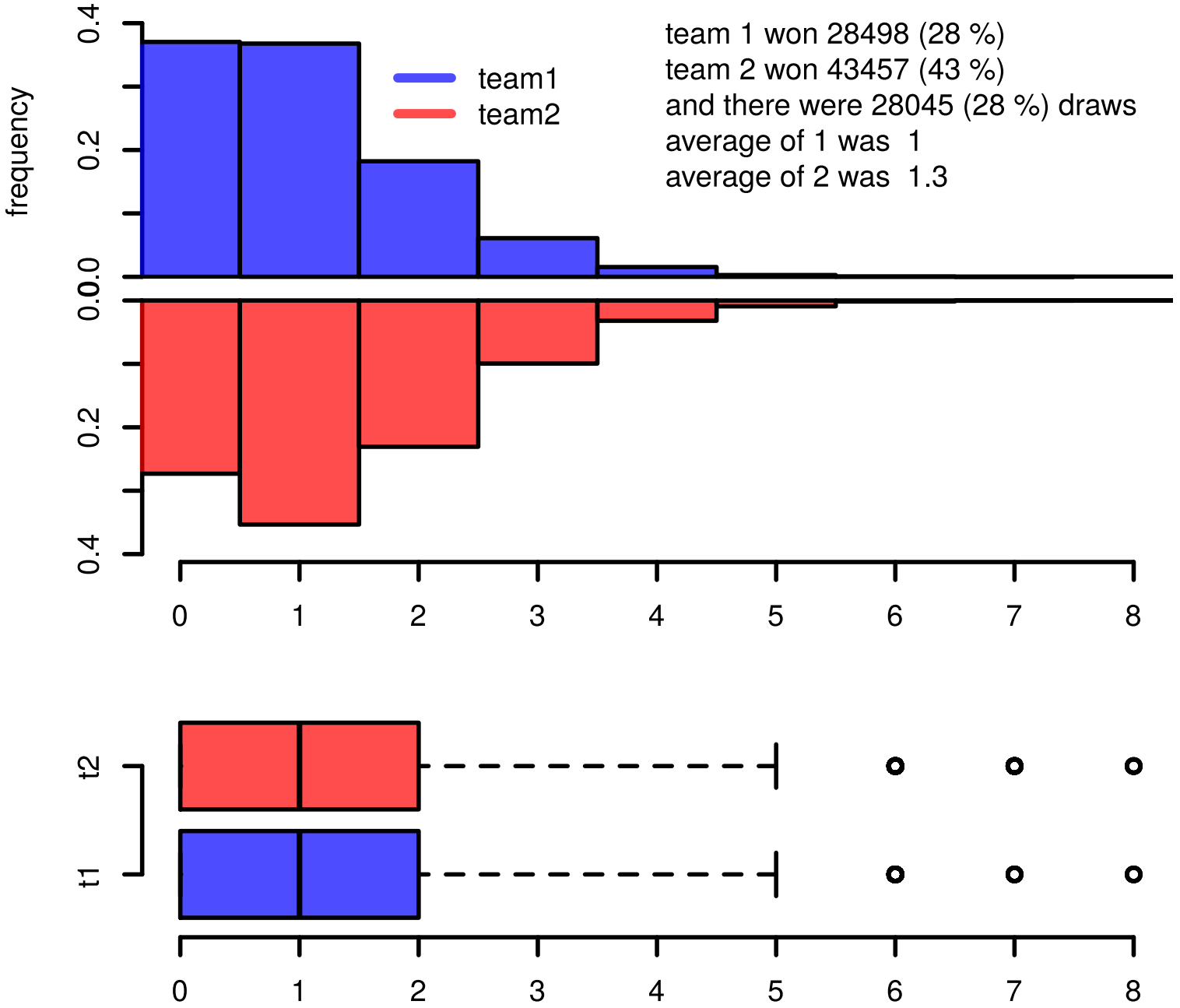

```
boxplot (df, horizontal=T, ylim=c(0,8), col=c(colour1, colour2))
```

- line shows median
- half box is 25 %
- dots for outliers



- histogram is not so clear

Flip limits on second histogram – no editing of data



put code in gitlab

can we make football better ?

Instead of one time unit $\mu = \lambda$ make games n times longer so

$$\mu = \lambda n$$

Put 100 000 games into a function

```
oneround <- function (mu1, mu2, n_games) {
    team1 <- rpois(n_games, mu1)
    team2 <- rpois(n_games, mu2)
    w1 <- sum (team1 > team2)
    w2 <- sum (team2 > team1)
    draw <- n_games - (w1 + w2)
    return (c(w1, w2, draw))
}
```

and call this for different values of μ_1, μ_2 scaled by n

more compact

How often does the better team win ?

```
oneround <- function (mu1, mu2, n_games) {
    team1 <- rpois(n_games, mu1)
    team2 <- rpois(n_games, mu2)
    w2 <- sum (team2 > team1)
}
```

just collect results for team 2

play repeatedly

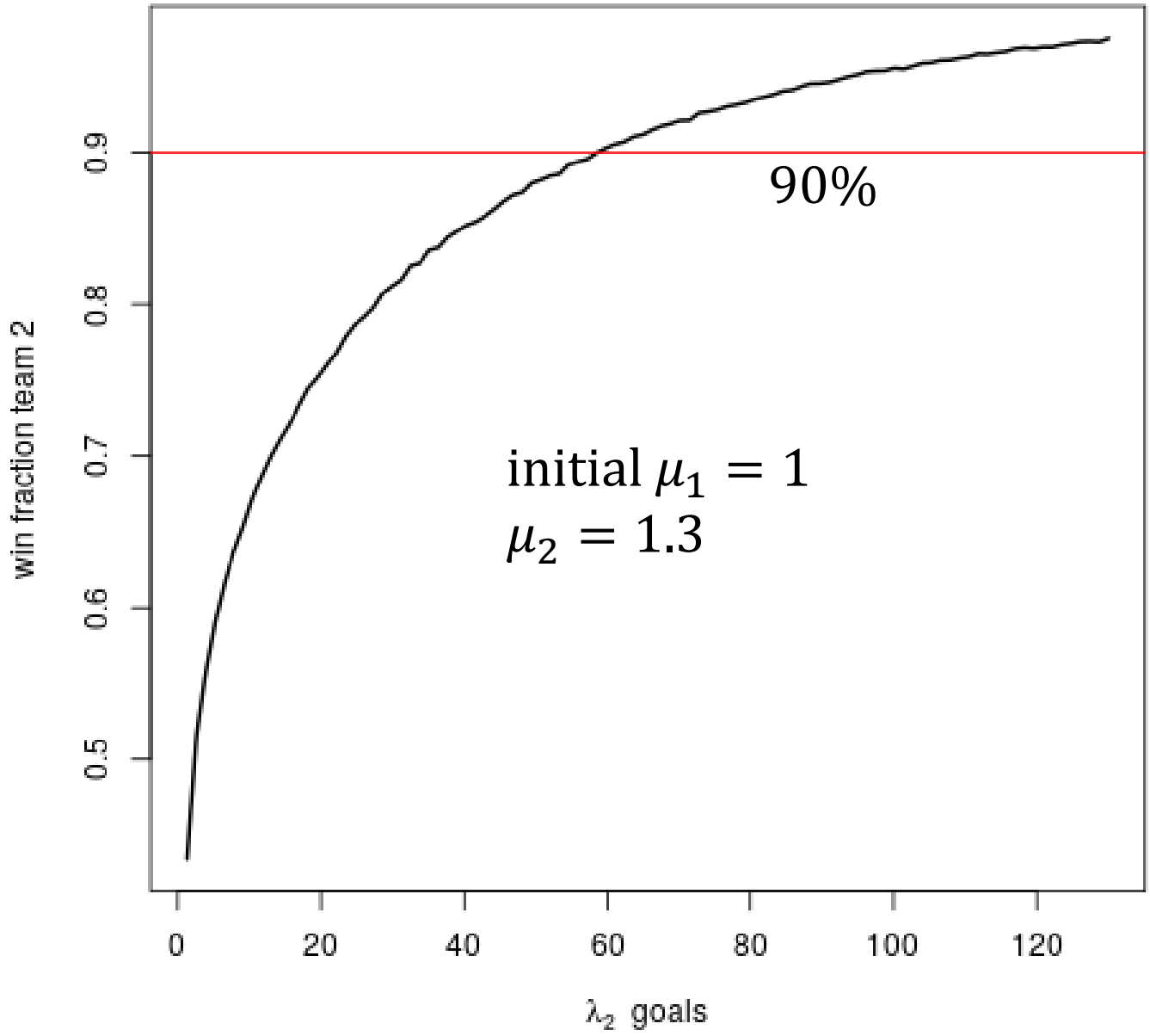
```
mult <- seq(from = 1, to = 100, by = 1)
wins <- c()
for (m in mult) {
    r <- oneround (team1_mu * m, team2_mu * m, n_games)
    wins = c(wins, r)
}
```

plot the results

If game are 10 times longer better team wins 62 %

If football games are about 60 fold longer, results are interesting

Note: team 2 does not win more than 50% for short games



what has one seen ?

Did I cheat in scripts ? not much

- some code for placing plots on page, setting random seed, histogram breaks

Football results are close to meaningless

R programming

- ugly but very powerful – basic poisson competition less than 10 lines
- graphics easy, but syntax horrible
- use built-in functions
- work with vectors not scalars

more serious R

Real statistician

- would have looked up poisson race

R – these lectures too short

- much serious statistics
- interesting fitting in the Übung (general non-linear, arbitrary function)
- most R users would
  - work in rcmdr, rstudio, ..
  - used a higher level graphics library (ggplot2, lattice)

Fitting

You are given

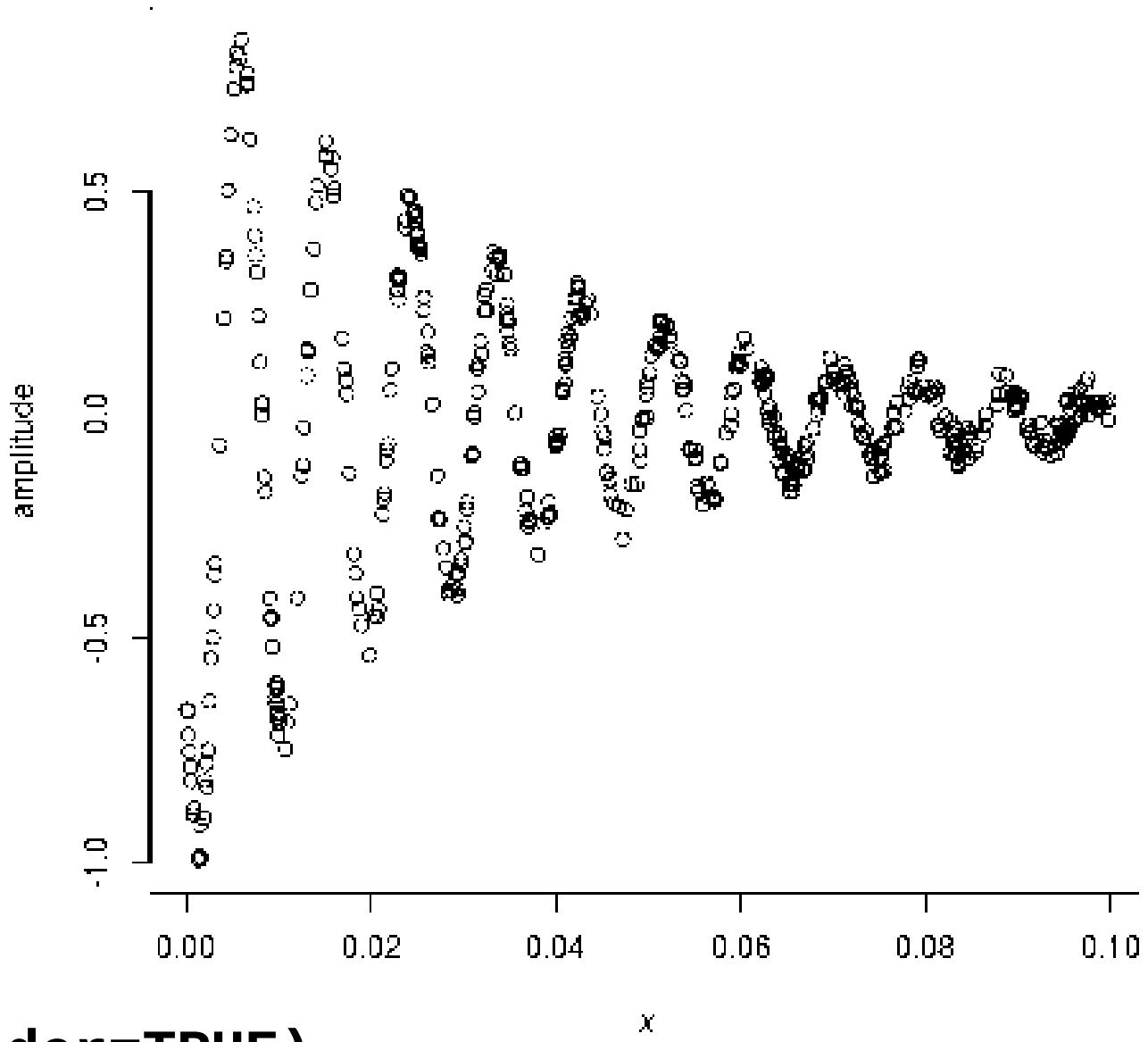
```
x y
0.09991595 0.031080097
0.09982738 -0.012845276
0.09946064 0.026970036
[ .. 500 .. lines]
```

and asked to make sense of it.

Start with a plot

Hint that it is of form
$y = e^{-\beta x} \sin(2\pi\omega x + \varphi)$
for some
β decay rate
ω frequency
φ phase

- there is noise
- points are not evenly spaced




```
d <- read.table (datafile, header=TRUE)
plot (d$x, d$y, xlab = "x", ylab="amplitude")
```

Code up the likely function

```
sinexp <- function (x, phase, freq, decay) {
     v <- sin(2 * pi * x * freq + phase)
     v <- v * exp(-decay * x)
}
```

- this function acts on a vector (**x**)
- returns a vector (**v**)

Have we got the right form ?

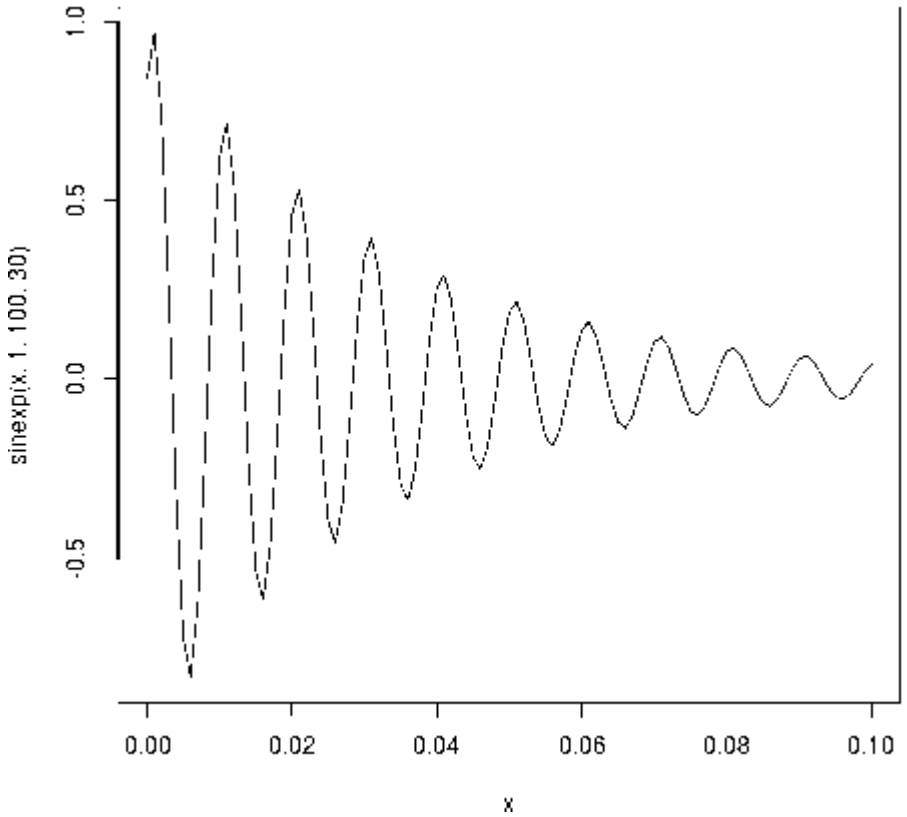
Check if our function looks sensible

```
curve (sinexp(x, 1, 100, 30), from = 0, to = 0.1)
```

- **`sinexp()`** seems to be possible

Note

- **`curve()`** has a default name of x
- it takes guessing / experience to get sensible values
- these values can also be used as starting points for fitting



non-linear least-squares fitting

- **`?nls`**
- gives you about 330 lines of help **`?nls.control`** gives more
- what works here ? defaults including Gauss-Newton method

We are asking R to move around in 3-parameter space, β, ω, φ

```
nlmod <- nls (y~sinexp(x, phase, freq, decay), data = d,
              start = list(phase=3, freq=150, decay=30))
```

You may not have seen the **~** in R – used to define models

reading results

Results are stored in nlmod

```
> nlmod
Nonlinear regression model
  model: y ~ sinexp(x, phase, freq, decay)
   data: d
  phase    freq   decay
  3.801 108.846  30.922
 residual sum-of-squares: 0.342

Number of iterations to convergence: 10
Achieved convergence tolerance: 2.912e-06
```

accessing elements is a bore, but you can say **`coef(nlmod)[phase]`**

```
> coef(nlmod)
phase  freq decay
3.8 108.8  30.9
```

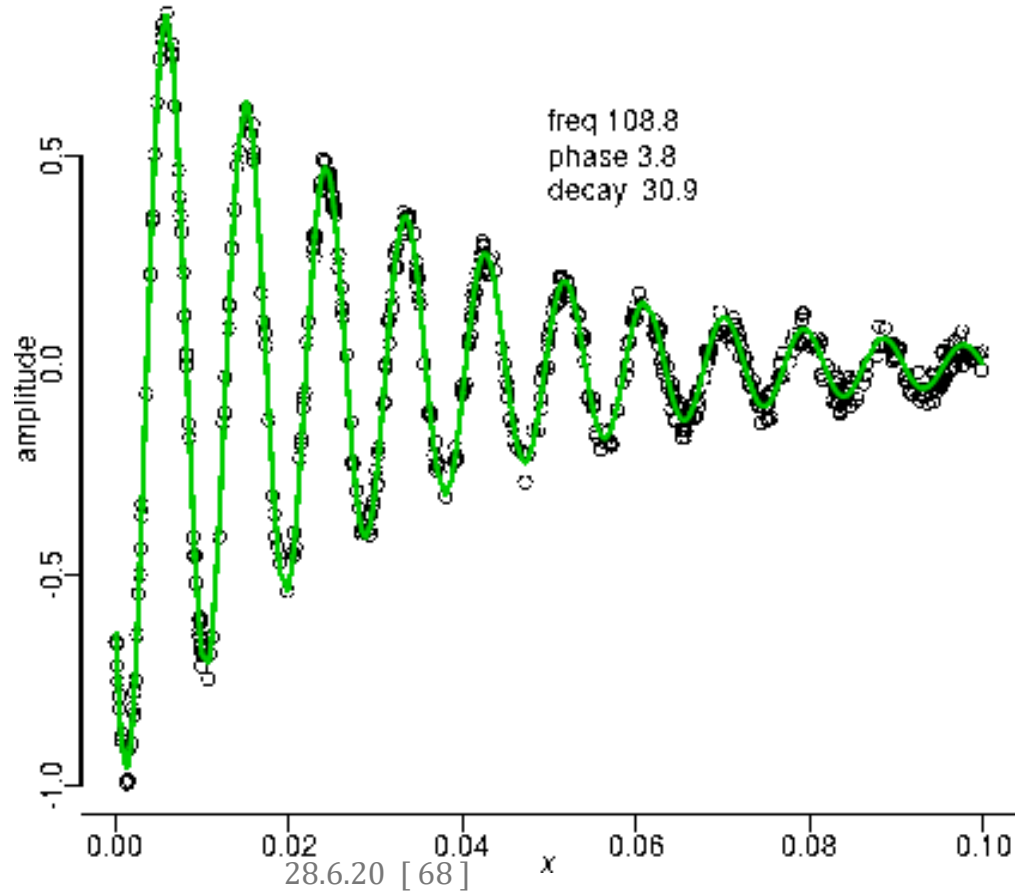
note **`coef()`** is a function call

see why **`coef(nlmod)[phase]`** works ?

to see if you really reproduce data

use the predict() function

```
plot (d$x, d$y, xlab = expression (italic(x)), ylab="amplitude")
pred=predict (nlmod)
lines(d$x, predict(nlmod), col = 3, lwd = 3)
```



Summary

- R syntax is as ugly as last week
- numerical functions remarkably
  - concise
  - simple

Last lecture

Are you sad or happy ?

- these lectures
  - little classic statistics (no t-test, anova, χ^2)
    - emphasis on programming
  - base R (deliberate)
  - no front end (rstudio, rcmdr, jupyter, …)

- what one could do
  - more regression, clustering, exploratory graphics
- more details ? text editing, file handling
- examples
  - not from biochemistry or physics

Bayes rule simulations

- Do we need to know serious statistics ?
- sometimes there is an analytical answer
  - but we do not know it
- R makes it easy to avoid knowing serious statistics
  - brute force / dumb simulations – extract properties of observations and count

- simulating and Bayes rule

Typical Bayes question

- This cancer is present in 1 in 1000 people ($P(C) = 10^{-3}$)
- Diagnostic tells you if you have a cancer
  - 5 % false positive
  - no false negatives

You test positive

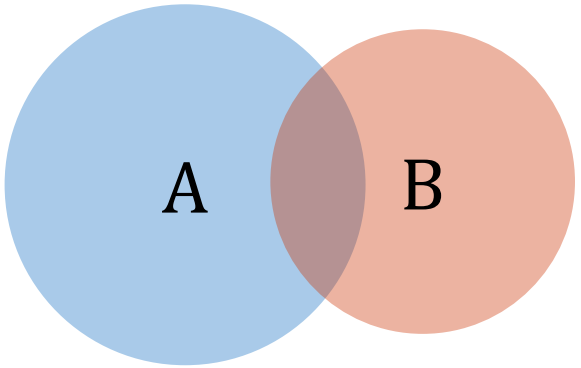
- what is the probability that you have cancer ?

Philosophy

- prior knowledge ? $P(C) = 10^{-3}$ modified by additional data

Bayes Rule

Intersection of A and B, $P(A \cap B)$



$$P(A \cap B) = P(B|A)P(A)$$

similarly

$$P(B \cap A) = P(A|B)P(B)$$

but $B \cup A$ is just an intersection, so $P(B \cap A) = P(A \cap B)$

$$P(B)P(A|B) = P(A)P(B|A)$$

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

Back to cancer question

cancer application

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

Call C cancer, and D a positive diagnostic so

$$P(C|D) = \frac{P(C)P(D|C)}{P(D)}$$

$P(C)$ is the probability of cancer overall, 1 in 1000 = 10^{-3}

$P(D|C)$ is 1

$P(D)$ more complicated – probability of a positive test in the general population

- contribution from sick + contribution from healthy .. $P(H) = 1 - 10^{-5}$

$$P(D) = P(D|C)P(C) \quad + \quad P(D|H)P(H)$$
$$= 1 \times 10^{-3} \quad + 0.05 \times (1 - 10^{-3}) = 10^{-3} + 0.05 \times 0.999$$

Cancer example

$$P(C|D) = \frac{P(C)P(D|C)}{P(D)}$$

$P(C) = 10^{-3}$

$P(D|C) = 1$ (no false negatives)
we had false positive of 0.05

$P(D)$ = healthy that are given positive (false positives) + cancerous that are given positive

$$= 0.999 \times 0.05 + 0.001 \times 1$$

$$P(C|D) = \frac{10^{-3} \times 1}{0.999 \times 0.05 + 0.001 \times 1}$$

$$\approx 0.02$$

You tested positive, but there is only a 2% chance of cancer

What would change if we also had false negatives ?

Factory example

Three factories, A is small and makes bad parts, 30 times worse than C

factory	production	defect rate
A	0.05	0.60
B	0.25	0.10
C	0.70	0.02

You get a defective part

What is the probability that it came from factory A ?

Define some terms

- $P(A), P(B), P(C)$ probability that part came from A, B or C
- $P(d)$ probability that part is defective
- Bayes rule $P(A|d) = \frac{P(d|A)\,P(A)}{P(d)}$ what is $P(d)$?

factory	production	defect rate	contribution
A	0.05	0.60	0.030
B	0.25	0.10	0.025
C	0.70	0.02	0.014
			0.069

Probability of a defective part is $P(d) = 0.069$

factory	production	defect rate	contribution
A	0.05	0.60	0.030
B	0.25	0.10	0.025
C	0.70	0.02	0.014
			0.069

Bayes rule $P(A|d) = \frac{P(d|A)\,P(A)}{P(d)} = \frac{0.6 \cdot 0.05}{0.069} \approx 0.44$

Bayes over distributions

Previously $P(A|B) = \frac{P(A)P(B|A)}{P(B)}$ Write in more specific way,

- H some hypothesis, D some data $P(H|D) = \frac{P(H)P(D|H)}{P(D)}$

Reconsider factory example: hypothesis is broken part came from factory A

- with no additional information probability probability of factory A, $P(H)$=0.05
- given some data, $P(D|H) = 0.6$
  our expectations (posterior probability) can be updated (from 0.05 to 0.44)

factory	production	defect rate
A	0.05	0.60
B	0.25	0.10
C	0.70	0.02

estimating money in circulation

collect 1000 notes

label

distribute to people

sample 1000 notes

from 1000 notes 20 are labelled

How many notes are in use ? …

$$\frac{1000}{20} \cdot 1000 = \frac{10^5}{2} = 50\,000 \text{ notes ?}$$

Uncertainty

50 000 is intuitive, but

- is 45 000 also possible ? How certain are you ?

Back to Bayes

$$P(H|D) = \frac{P(H)P(D|H)}{P(D)}$$

$P(H)$ our hypothesis is naïve: $n_{notes} > 1000$, not very informative

$P(D)$ treat as some constant – common in Bayes calculations

$P(D|H)$ a hypothesis H would be that $n_{notes} = 50\,000$ or $55\,000$ or …

- Data D … we sample 1000 notes and find 20 marked
- the mission is to calculate $P(D|\,H)$ for various values of n_{notes}

$P(D|H)$ for various values of n_{notes}

For 40 000, 42 000, … 100 000 how often do I see 20 from 1000 marked notes ?

Make a pool of notes

```
pool_of_notes <- rep (c(1, 0), c(n_marked, n_notes - n_marked))
```

- a marked note is 1, unmarked is 0
- `c(n_marked, n_notes - n_marked))` like `c(1000, 40000)`
  - means `pool..` is a long array 1000 × 1, 40000 × 0

A single sample is

```
single_sample <- function (pool_of_notes, n_resampled) {
      samp <- sample (pool_of_notes, n_resampled)
      return (sum(samp))# note the return – number of marked notes
}
```

```
sample_notes <- function (n_notes, n_marked, n_trials, n_resampled) {
 pool_of_notes <- rep (c(1, 0), c(n_marked, n_notes - n_marked))
 return(replicate(n_trials,single_sample(pool_of_notes,n_resampled)))
}
```

Summarise till now

- generate the complete currency using available knowledge
- take a sample from this pool

- only for a specified size of the pool (a guess / hypothesis)

Next step

- set up a loop to try out different hypotheses

we want to repeat this for different hypotheses

```
n_found <- 20

h <- seq(20000, 120000, 5000)
for (i in h) {
          a <- sample_notes(i, n_marked, n_trials, n_resampled)
          t <- length(a[a == n_found])
          success <- c(success, t)
 }
```

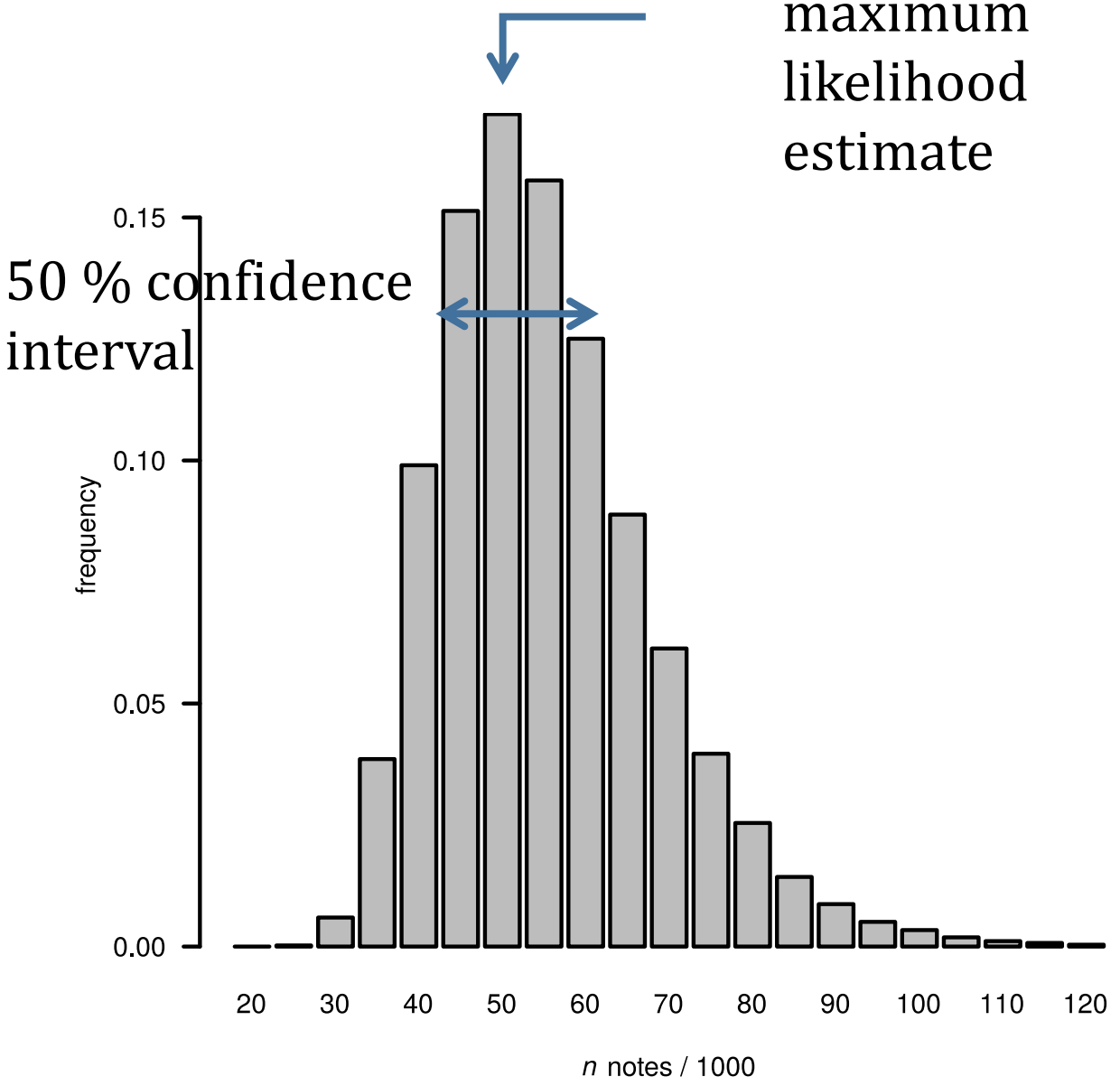
plot out **`success`** vector…

results of simulation

What has one gained ?

- uncertainty
- 50 000 is your best guess, but the estimate is not very good

How hard was it ?



```
single_sample <- function (pool_of_notes, n_resampled) {
     samp <- sample (pool_of_notes, n_resampled)
     return (sum(samp))
}

sample_notes <- function (n_notes, n_marked, n_trials, n_resampled) {
     pool_of_notes <- rep (c(1, 0), c(n_marked, n_notes - n_marked))
     return (replicate (n_trials, single_sample(pool_of_notes, n_resampled)))
}

# ------------ main -----------------
mymain <- function () {
     n_marked    <- 1000   # Originally marked this many notes
     n_resampled <- 1000   # We sampled this many notes
     n_found     <- 20     # and found this many marked
     n_trials    <- 10000  # How many times do we repeat sampling
     h <- seq(20000, 120000, 5000) # An array of hypotheses, values to try out
     set.seed (37)         # I like determinism
     hard_copy = F

     success <- vector()
     for (i in h) {
          a <- sample_notes(i, n_marked, n_trials, n_resampled)
          t <- length(a[ (a == n_found])
          success <- c(success, t)
     }

     success = success / sum (success)
     barplot (success, names = h/1000, xlab='notes/1000', ylab='frequency', axes=TRUE, las=1)
}
mymain ()
```

code that does something ?

a dozen lines

Adding information to model

started with $P(H|D) = \frac{P(H)P(D|H)}{P(D)}$ and said all $P(H)$ are possible

Zentralbank printed 70 000 notes so $P(H) = 0$ if $H > 70\,000$

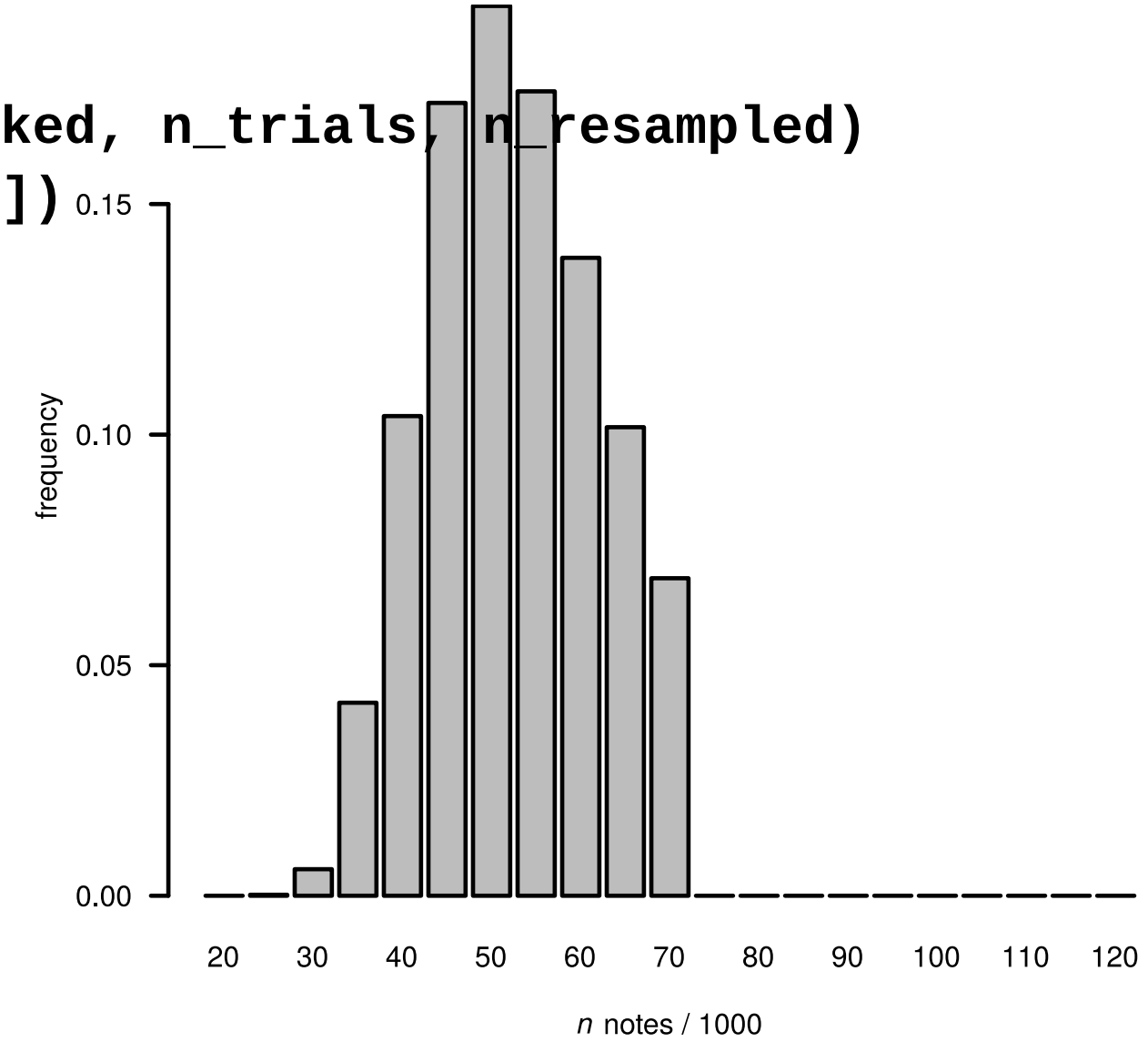
```
h <- seq(20000, 120000, 5000)
for (i in h) {
          a <- sample_notes(i, n_marked, n_trials, n_resampled)
          t <- length(a[a == n_found]) # n_found = the 20 marked notes
          success <- c(success, t)
 }
```

Change and plot

only 70 000 notes printed

```
for (i in h) {
    if (i <= 70000) {
        a <- sample_notes(i, n_marked, n_trials, n_resampled)
        t <- length(a[a == n_found])
    else {
        t <- 0
    }
    success <- c(success, t)
 }
```

- max likelihood unchanged
- median shifted left
- can adjust our error estimates…



cumulative probabilities

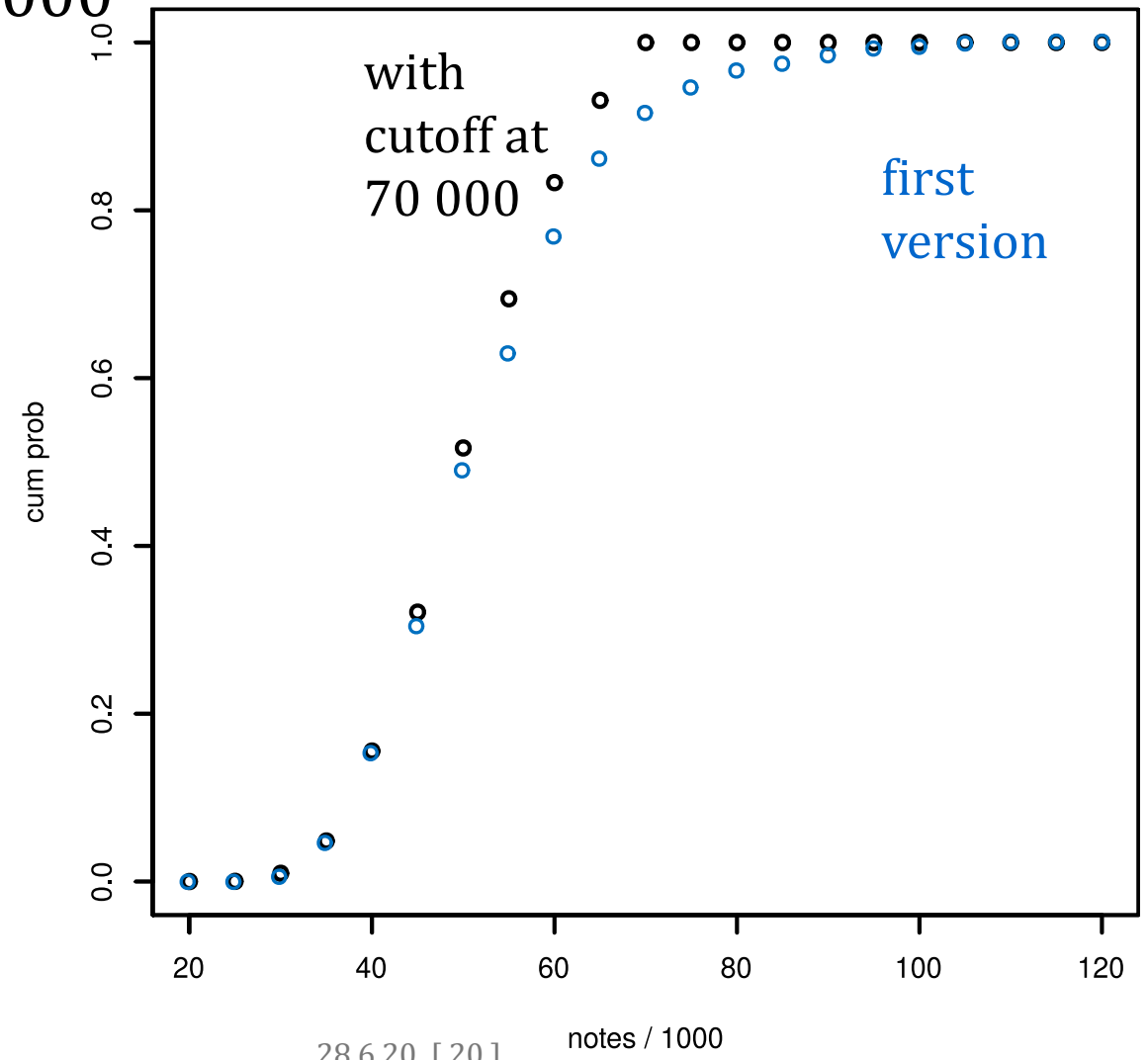
previously

- ≈ 75% chance that number of notes < 60 000
- now ≈ about 85 %

formal term for adjusting $P(H)$

- more informative "prior" information

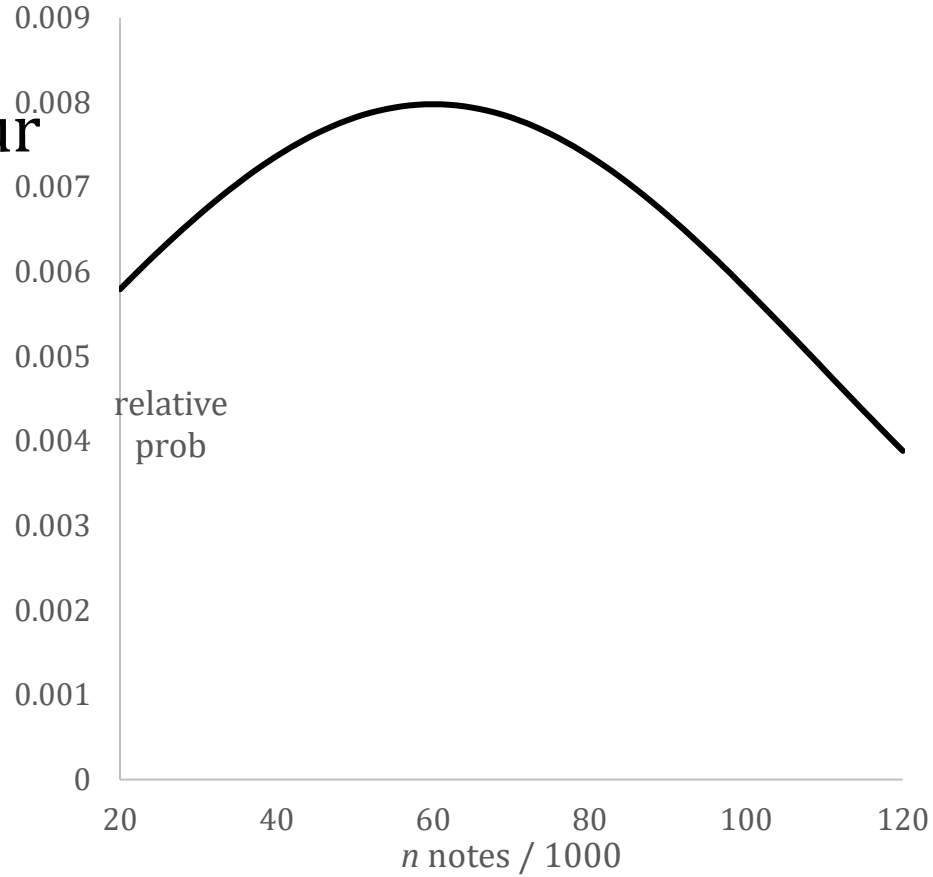
Can we make a better prior distribution ?



better prior distribution

Ask 200 bankers how many notes are in circulation and sum their estimates

- discard guesses > 70 000
- average guess is 60 000
- bankers average estimate is not the same as our max likelihood estimate (50 000)



Include the information very vaguely

- normal distribution with
- μ=60 000
- σ = 50 000 (σ : distrust in bankers)

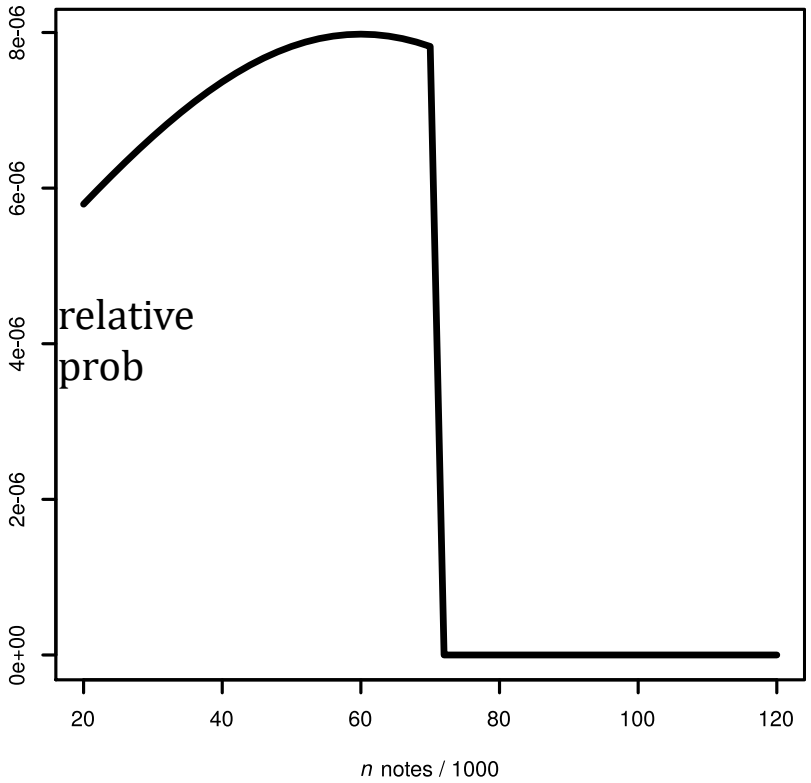
More: we have to put in the cutoff of 70 000

better prior distribution

Ask 200 bankers how many notes are in circulation and sum their estimates

- discard guesses > 70 000
- average guess is 60 000 before chopping

```
prior <- function(x, mean, sd) {
     if (x > 70000) {
          return (0)
     } else {
          return (dnorm (x, mean, sd))
     }
}
```



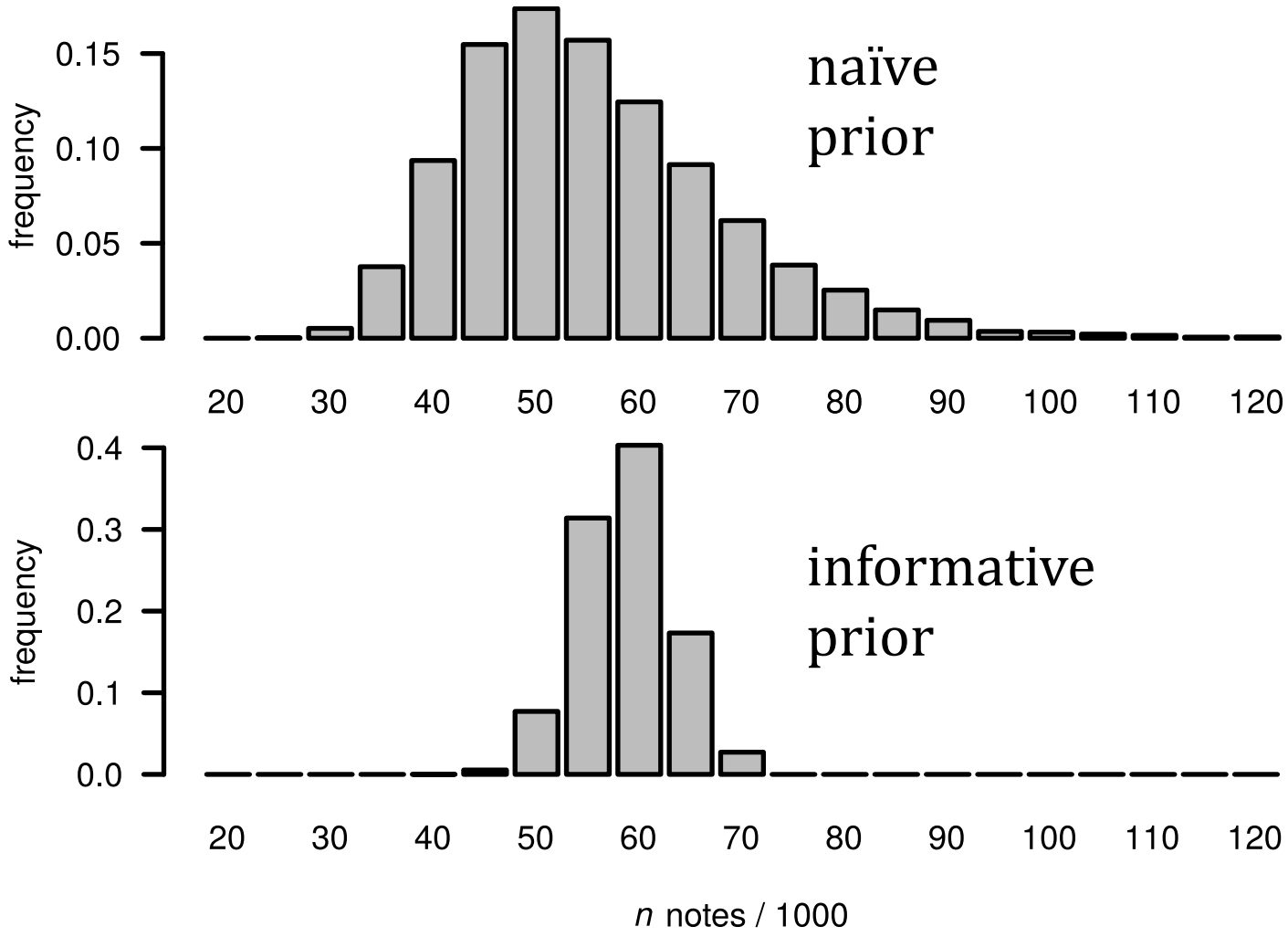
normal distribution

60 000

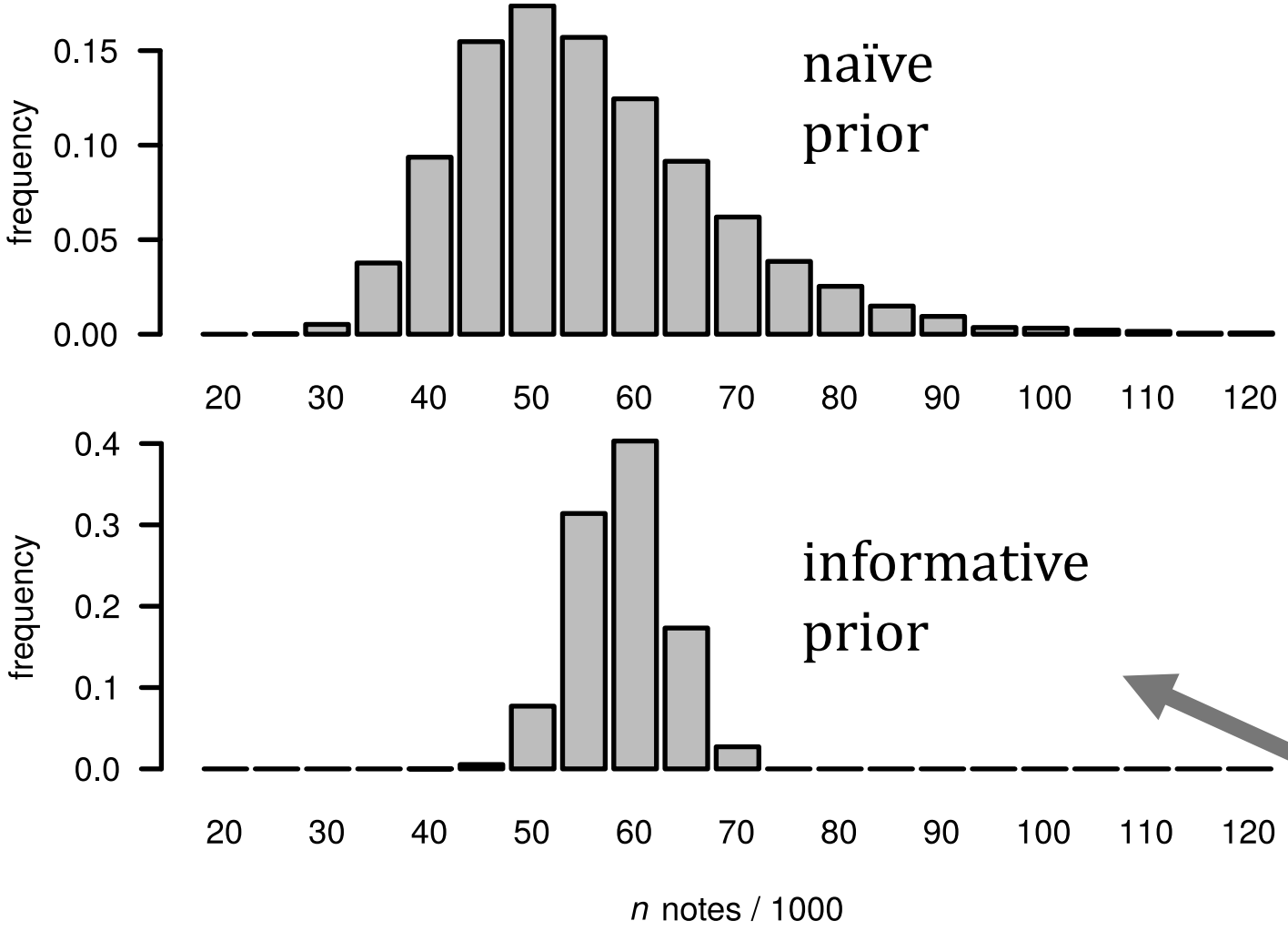
50 000

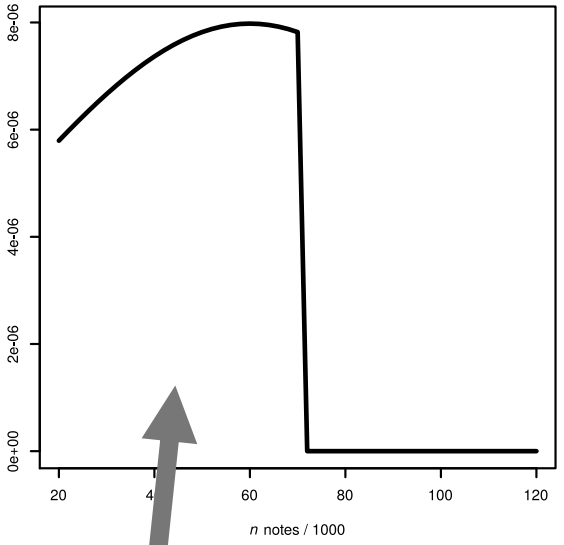
affect of prior information

- Maximum likelihood estimate is now 60 000 notes
- mean shifted right
- uncertainty much smaller



affect of prior information

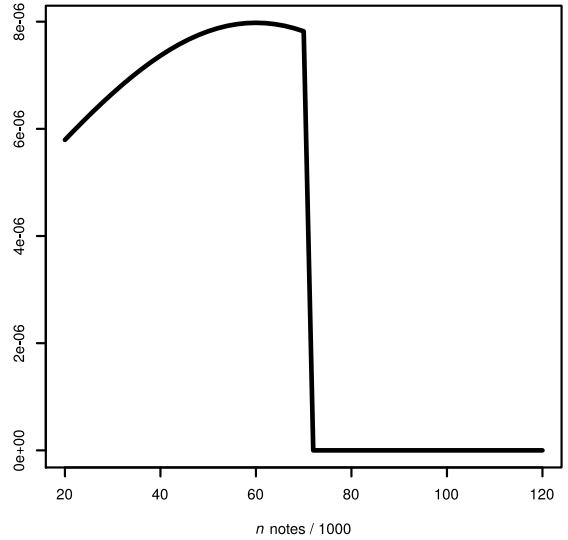




from
simulation

$$P(H|D) = \frac{P(H)P(D|H)}{P(D)}$$

ad hoc ?



Completely changed my estimate

- limit of 70 000 OK
- distribution based on banker's estimates ?

What if I do not trust bankers so much ?

- increase σ (**`sd`**) in the prior function

Philosophical question with Bayes

- how do you encode prior knowledge ?

```
prior <- function(x, mean, sd) {
     if (x > 70000) {
          return (0)
     } else {
          return (dnorm (x, mean, sd))
     }
}
```

prior information summary

	hypothesis	prior probability $P(H)$	posterior $P(H\|D)$
cancer	you have cancer	10^{-3}	0.02
factory	component from factory A	0.05	0.44
money	many hypotheses	flat distribution	more narrow distribution

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

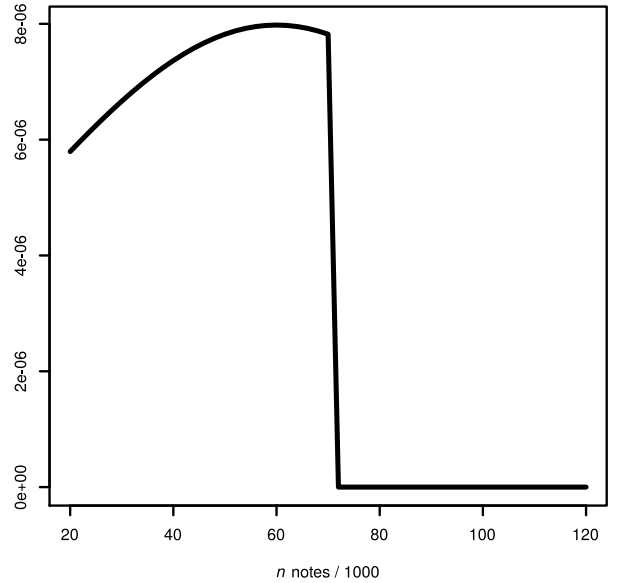
When I add knowledge, I am usually changing $P(A)$

what one sees (statistically)

Bayesian rules and prior information

- sampling
  - lets you avoid understanding statistics
  - lets you combine probabilities that would be hard analytically (truncated gaussian)
  - lets you estimate errors / mean / percentiles

- formal description and rule for including
  - well defined contributions
  - possibly *ad hoc* ideas (bankers expert opinion)



what one sees (programming)

Long loops are avoided

```
pool <- rep (c(1, 0), c(n_marked, n_notes - n_marked))
```

- repeat 1 and 0 according to n_marked and (n_notes−n_marked)

```
samp <- sample (pool_of_notes, n_resampled)
```

- take n_resampled entries from an array pool_of_notes

```
replicate (n_trials, single_sample(pool_of_notes, n_resampl))
```

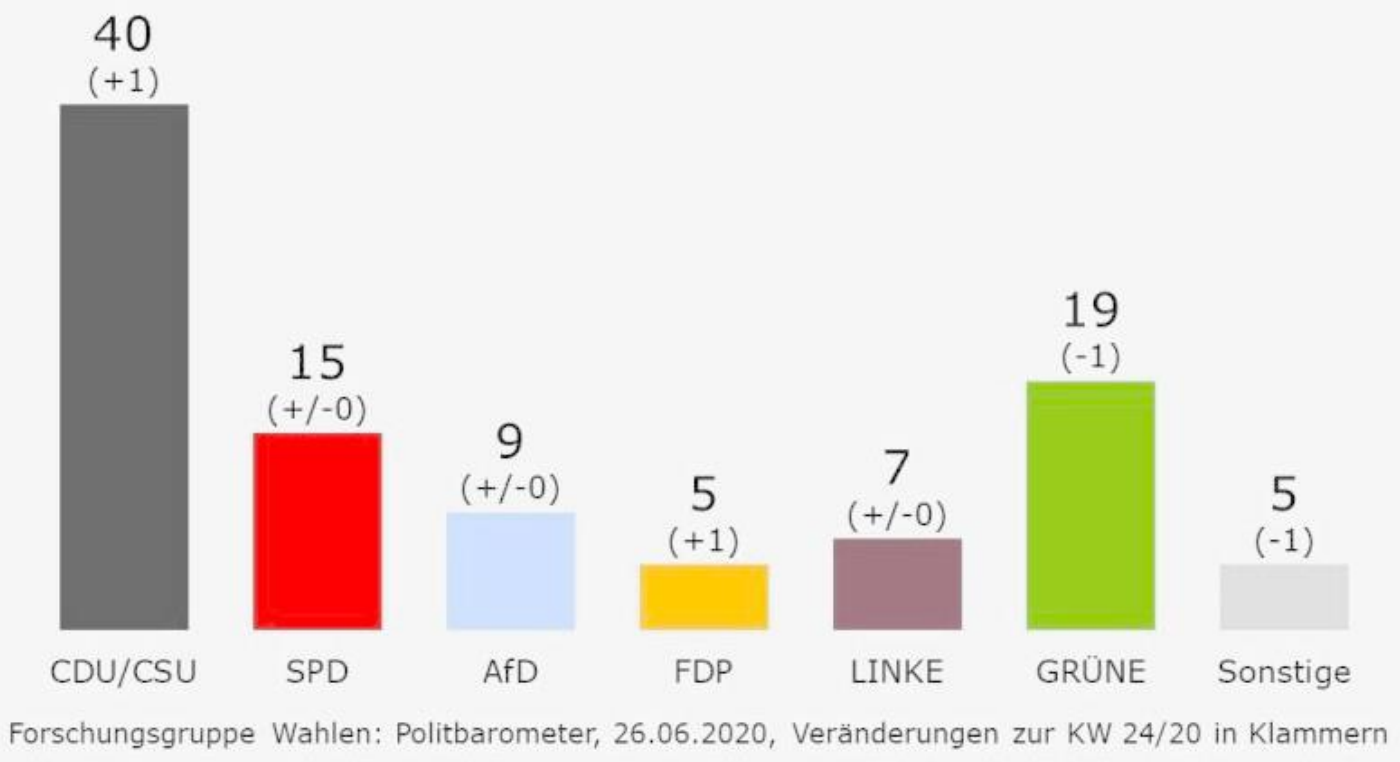
- run single_sample for n_trials and return a vector/matrix with results

Example code (used for this lecture) in gitlab

Sampling for election error

The Übung

- Problem ? Real … Answer ? very brute force, but easy



≈ 1250 telefonisch Befragte

can you tell the difference between 7 and 9 % ?

(hidden Folium)

how hard is it ?

sample from populations

one survey repeated n times

plot

repeat with different numbers of people

```
parties   <- c('green', 'spd', 'csu', 'afd', 'others')
true_prob <- c(0.21,    0.16,  0.29,  0.11 ) # One element shorter than parties

n_people_sets <- c(500, 1000, 2000)
n_sampling    <- 20000  # How often do we repeat the sampling
svg_or_screen <- 'svg_' # If set to svg, we make svg output

set.seed(1637)          # I like determinism

# The last entry in the probabilities is for 'others'
true_prob = c(true_prob, 1-sum(true_prob))
names(true_prob) <- parties

# onesamp does one sampling of the parties. It does not need the
# names of the parties.
onesamp <- function (n_people, true_prob) {
     nparty <- length(true_prob)
     s <- sample (1:nparty, n_people, replace=TRUE, true_prob)
     rslt <- tabulate (s, nbins=nparty)
     return (rslt)
}

# do_survey does a full calculation for a given survey size
do_survey <- function (n_people, n_sampling) {
     ms <- replicate (n_sampling, onesamp(n_people, true_prob))
     rownames(ms) <- parties
     ms <- ms / n_people

     fmt1 <- "%6s  %4.3f   %4.3f    %4.3f - %4.3f\n"
     cat ("Data collected from ", n_people, " people and with ", n_sampling, " resampling\n")
     cat (" party expcted median     (95% limits)\n")
     for (i in 1:length(parties)) {
          cat(sprintf (fmt1, parties[i], true_prob[i], median(ms[i,]),
                       quantile(ms[i,], c(0.025)), quantile(ms[i,],c(0.975))))
     }
#    From here on, we just plot. We could plot the matrix directly, but it
     ms <- data.frame(t(ms))       # is easier to control the order of output
     if (svg_or_screen == 'svg') { # after conversion to data frame
          svg(filename=paste("election_", n_people, ".svg", sep=''))
     }
     t <- paste (n_people, " people")
     boxplot(ms, main=t, boxwex=0.6, ylab='', cex=0.8, ylim=c(0,0.38), las=1)
     mtext('frequency ',side=2, las=1)
     if (svg_or_screen == 'svg') { dev.off()}
}

#----------------- main -------------------------------
discard <- lapply (n_people_sets, do_survey, n_sampling=n_sampling)
```

sampling

- Earlier – ugly – make a pile of notes and draw randomly from it
- More elegant - use sample to get results from array of possibilities with known probabilities

Planning: in all code keep an order of parties (green, SPD, CSU, ..)

- names, probabilities, results, …

```
parties   <- c('green', 'spd', 'csu', 'afd', 'others')
true_prob <- c(0.19,    0.15,  0.40,  0.09,  0.24)
```

The sampling

```
s <- sample (1:nparty, n_people, replace=TRUE, true_prob)
```

sampling

The sampling

```
s <- sample (1:nparty, n_people, replace=TRUE, true_prob)
```

a vector 1, 2, ..	500 people or 1000 or …	the array of probabilities

- **`sample()`** will keep picking numbers from **`1:nparty`** with correct probability
- what will **`s`** be ?

- how to count the contents of **`s`** ?

counting integers in an array

`s` looks like [1,2,5, 3,…] some hundred times

```
rslt <- tabulate(s, nbins=nparty)
```

tabulate function

- walks along **`s`** and putting numbers into bins
- returns a vector with the number of time we had 1, 2, 3, …

Easy to do one sample with n people

- repeat many times ?

repeatedly calling a function

```
ms <- replicate (n_sampling, onesamp(n_people, true_prob))
```

how many times ?

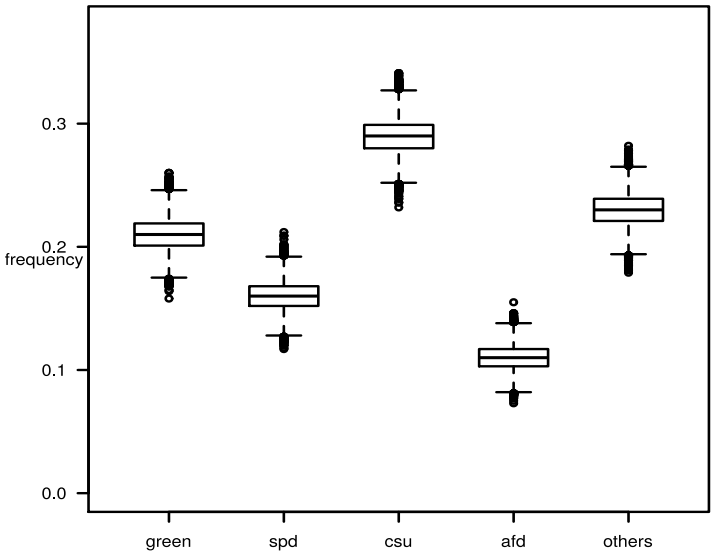
a function I write

- what is in **`ms`** ?
- **`replicate()`** calls some function many times and puts all the results into a vector, matrix, …
- here: a matrix with **`n_party`** rows

What did I do to make handling easier ?

- put name on matrix rows **`rownames(ms) <- parties`**

Finished by calling **`boxplot()`** on **`ms`**



summarise

```
sample (1:nparty, n_people, replace=TRUE, true_prob)
```

gives me a long array, representing the sampled people

`tabulate()` counts them
`replicate()` calls my sampling function many times

- compare with other languages

summarise

`sample (1:nparty, n_people, replace=TRUE, true_prob)`

- implicitly set up a loop over n_people
  - each time doing a biased selection from nparty

`tabulate(x, n)`

- implicitly walks along **`x`**, keeping track of each **`n`**
- **`rslts <- replicate(n, f())`**

- simple loop, but gathers results nicely

finishing up

- get everything to work
- put this into a function which takes n_people as argument

- call this function with several values of n_people (500, 1000, 2000)