An overview of the R programming environment

Deepayan Sarkar

Software for Statistics

- Computing software is now essential for data analysis
  - Large datasets
  - Visualization
  - Simulation
  - Iterative methods
- Many softwares are available

- Today I will talk about one such software called R
  - Available as Free / Open Source Software
  - Very popular (both academia and industry)
  - Easy to try out on your own

Outline

- Installing and starting R
- Some examples
- A little bit of history
- This is not a tutorial! But I am happy to answer questions later.

Installing R

- R is most commonly used as a REPL (Read-Eval-Print-Loop)
- This is essentially the model used by a calculator:
    - Waits for user input
    - Evaluates and prints result
    - Waits for more input
- There are several different *interfaces* to do this
- R itself works on many platforms (Windows, Mac, UNIX, Linux)
- Some interfaces are platform-specific, some work on most
- R and the interface may need to be installed separately

Installing R

- Go to https://cran.r-project.org/ (or choose a mirror first)
- Follow instructions depending on your platform (probably Windows)
- This will install R, as well as a default graphical interface on Windows and Mac
- I recommend a different interface called R Studio that needs to be installed separately
- I personally use yet another interface called ESS which works with a general purpose editor called Emacs (download link for Windows)

Running R

- Once installed, you can start the appropriate interface (or R directly) to get something like this:

```
R version 3.5.1 (2018-07-02) -- "Feather Spray"
Copyright (C) 2018 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.

Loading required package: utils
>
```

- The > represents a *prompt* indicating that R is waiting for input.
- The difficult part is to learn what to do next

Before we start, an experiment!

Color combination: Is it **white & gold** or **blue & black** ? Let’s count!

Question: What proportion of population sees white & gold?

- Statistics uses data to make inferences
- Model:
  - Let p be the probability of seeing white & gold
  - Assume that individuals are independent
- Data:
  - Suppose X out of N sampled individuals see white & gold; e.g., $N = 45, X = 26.$
  - According to model, $X \sim Bin(N, p)$
- “Obvious” estimate of $p = X/N = 26/45 = 0.5778$
- But how is this estimate derived?

Generally useful method: maximum likelihood

- Likelihood function: probability of observed data as function of p

$$L(p) = P(X = 26) = \binom{45}{26} p^{26}(1-p)^{(45-26)}, p \in (0,1)$$

- Intuition: p that gives higher $L(p)$ is more “likely” to be correct
- Maximum likelihood estimate $\hat{p} = \arg\max L(p)$
- By differentiating

$$\log L(p) = c + 26 \log p + 19 \log(1-p)$$

we get

$$\frac{d}{dp} \log L(p) = \frac{26}{p} - \frac{19}{1-p} = 0 \implies 26(1-p) - 19p = 0 \implies p = \frac{26}{45}$$

How could we do this numerically?

- Pretend for the moment that we did not know how to do this.
- How could we arrive at the same solution numerically?
- Basic idea: Compute $L(p)$ for various values of p and find minimum.
- To do this in R, the most important thing to remember is that **R works like a calculator**:
  - The user types in an expression, R calculates the answer
  - The expression can involve numbers, variables, and functions
- For example:

```r
N = 45
x = 26
```

```r
p = 0.5
choose(N, x) * p^x * (1-p)^(N-x)
```

```
[1] 0.06930242
```

“Vectorized” computations

- One distinguishing feature of R is that it operates on “vectors”

```r
pvec = seq(0, 1, by = 0.01)
pvec
```

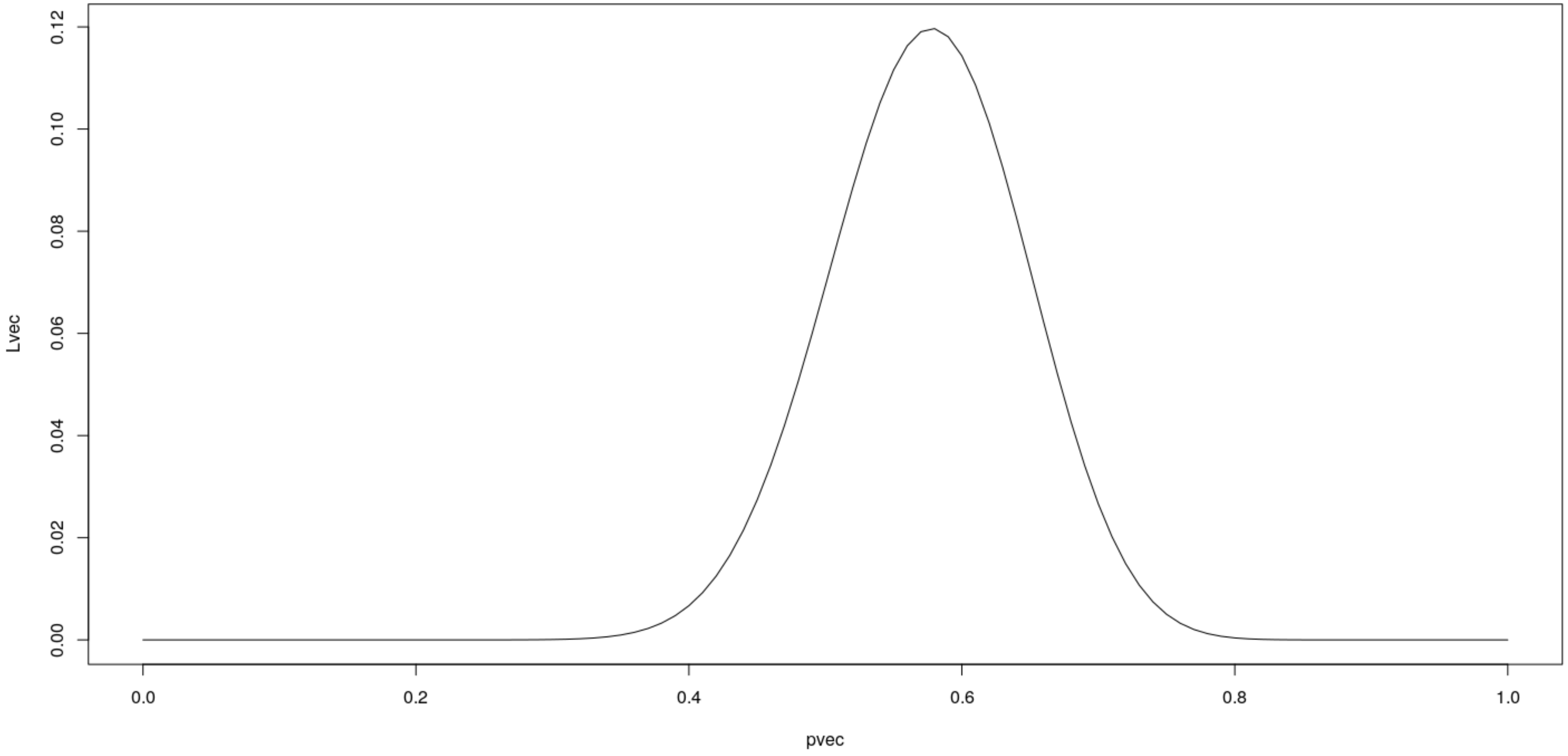
```
  [1] 0.00 0.01 0.02 0.03 0.04 0.05 0.06 0.07 0.08 0.09 0.10 0.11 0.12 0.13 0.14 0.15 0.16 0.17 0.18 0.19 0.20 0.21 0.22
 [24] 0.23 0.24 0.25 0.26 0.27 0.28 0.29 0.30 0.31 0.32 0.33 0.34 0.35 0.36 0.37 0.38 0.39 0.40 0.41 0.42 0.43 0.44 0.45
 [47] 0.46 0.47 0.48 0.49 0.50 0.51 0.52 0.53 0.54 0.55 0.56 0.57 0.58 0.59 0.60 0.61 0.62 0.63 0.64 0.65 0.66 0.67 0.68
 [70] 0.69 0.70 0.71 0.72 0.73 0.74 0.75 0.76 0.77 0.78 0.79 0.80 0.81 0.82 0.83 0.84 0.85 0.86 0.87 0.88 0.89 0.90 0.91
 [93] 0.92 0.93 0.94 0.95 0.96 0.97 0.98 0.99 1.00
```

```r
Lvec = choose(N, x) * pvec^x * (1-pvec)^(N-x)
Lvec
```

```
  [1] 0.000000e+00 2.014498e-40 1.114740e-32 3.474672e-28 5.056051e-25 1.371093e-22 1.283689e-20 5.765318e-19
  [9] 1.511495e-17 2.625366e-16 3.293866e-15 3.174813e-14 2.460262e-13 1.586687e-12 8.747777e-12 4.211439e-11
 [17] 1.801043e-10 6.938314e-10 2.435828e-09 7.868776e-09 2.358239e-08 6.602594e-08 1.737342e-07 4.318627e-07
 [25] 1.018706e-06 2.289299e-06 4.918220e-06 1.013189e-05 2.006894e-05 3.831376e-05 7.065023e-05 1.260767e-04
 [33] 2.181057e-04 3.663379e-04 5.982529e-04 9.510890e-04 1.473611e-03 2.227478e-03 3.287864e-03 4.742910e-03
 [41] 6.691627e-03 9.239888e-03 1.249429e-02 1.655390e-02 2.150009e-02 2.738512e-02 3.422026e-02 4.196469e-02
 [49] 5.051658e-02 5.970760e-02 6.930242e-02 7.900386e-02 8.846442e-02 9.730387e-02 1.051320e-01 1.115747e-01
 [57] 1.163022e-01 1.190543e-01 1.196637e-01 1.180712e-01 1.143327e-01 1.086179e-01 1.011977e-01 9.242411e-02
 [65] 8.270372e-02 7.246667e-02 6.213552e-02 5.209643e-02 4.267559e-02 3.412296e-02 2.660425e-02 2.020120e-02
 [73] 1.491921e-02 1.070050e-02 7.440747e-03 5.006696e-03 3.252859e-03 2.035570e-03 1.223457e-03 7.039944e-04
 [81] 3.863739e-04 2.013850e-04 9.918271e-05 4.588367e-05 1.979882e-05 7.901767e-06 2.887291e-06 9.539431e-07
 [89] 2.806024e-07 7.206085e-08 1.575446e-08 2.836495e-09 4.020606e-10 4.212284e-11 2.973693e-12 1.225581e-13
 [97] 2.318943e-15 1.283711e-17 7.560423e-21 1.877644e-26 0.000000e+00
```

Plotting is very easy

```
plot(x = pvec, y = Lvec, type = "l")
```



Functions

- Functions can be used to encapsulate repetitive computations
- Like mathematical functions, R function also take arguments as input and “returns” an output

```
L = function(p) choose(N, x) * p^x * (1-p)^(N-x)
L(0.5)
```

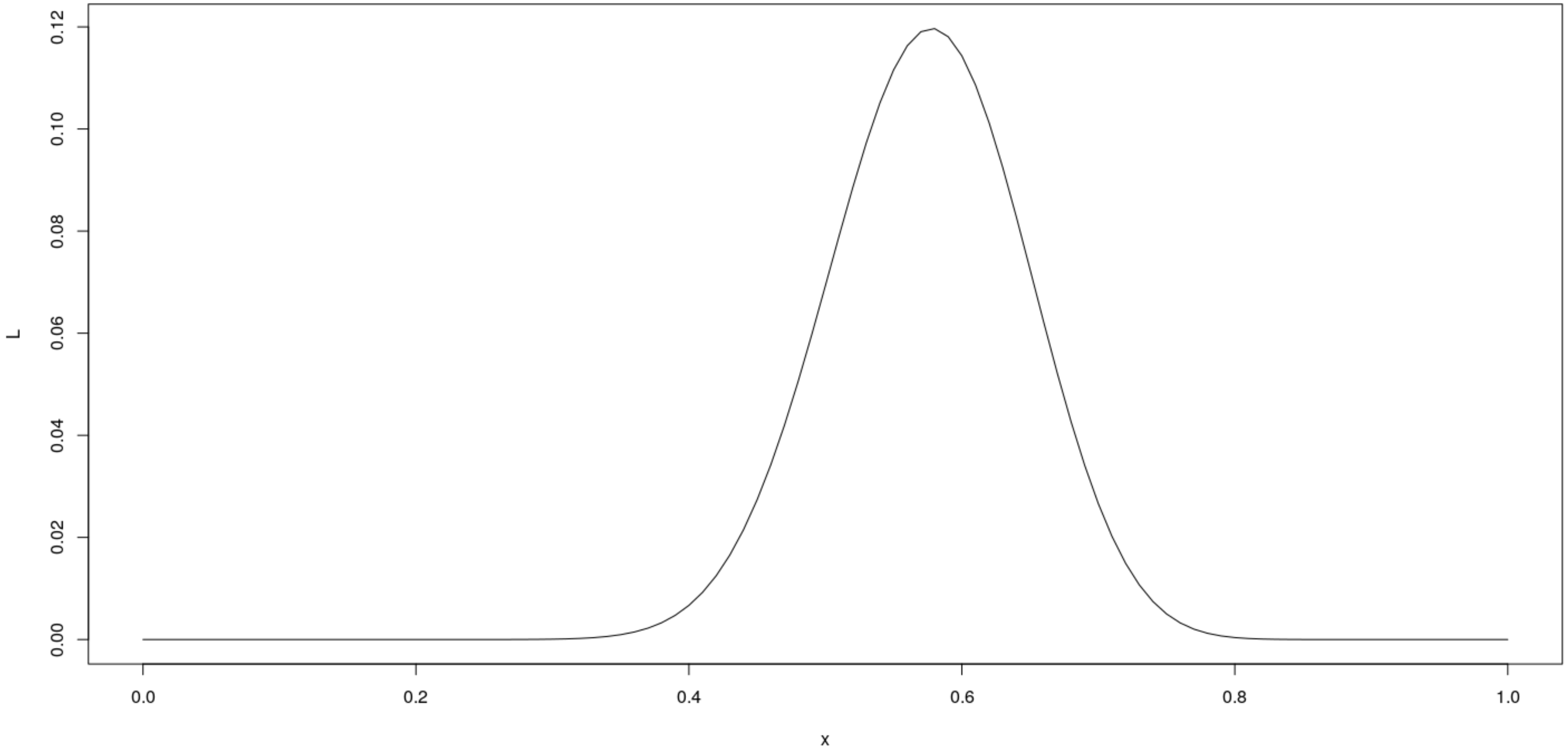
```
[1] 0.06930242
```

```
L(x/N)
```

```
[1] 0.1197183
```

Functions can be plotted directly

```
plot(L, from = 0, to = 1)
```



…and they can be numerically “optimized”

```r
optimize(L, interval = c(0, 1), maximum = TRUE)
```

```
$maximum
[1] 0.5777774

$objective
[1] 0.1197183
```

- Compare with

```r
x / N
```

```
[1] 0.5777778
```

```r
L(x / N)
```

```
[1] 0.1197183
```

A more complicated example

- Suppose $X_1, X_2, \ldots, X_n \sim Bin(N, p)$, and are independent
- Instead of observing each X_i, we only get to know $M = \max(X_1, X_2, \ldots, X_n)$
- What is the maximum likelihood estimate of p? (N and n are known, $M = m$ is observed)

A more complicated example

To compute likelihood, we need p.m.f. of M :

$$P(M \le m) = P(X_1 \le m, \ldots, X_n \le m) = \left[\sum_{x=0}^{m} \binom{N}{x} p^x (1-p)^{(N-x)}\right]^n$$

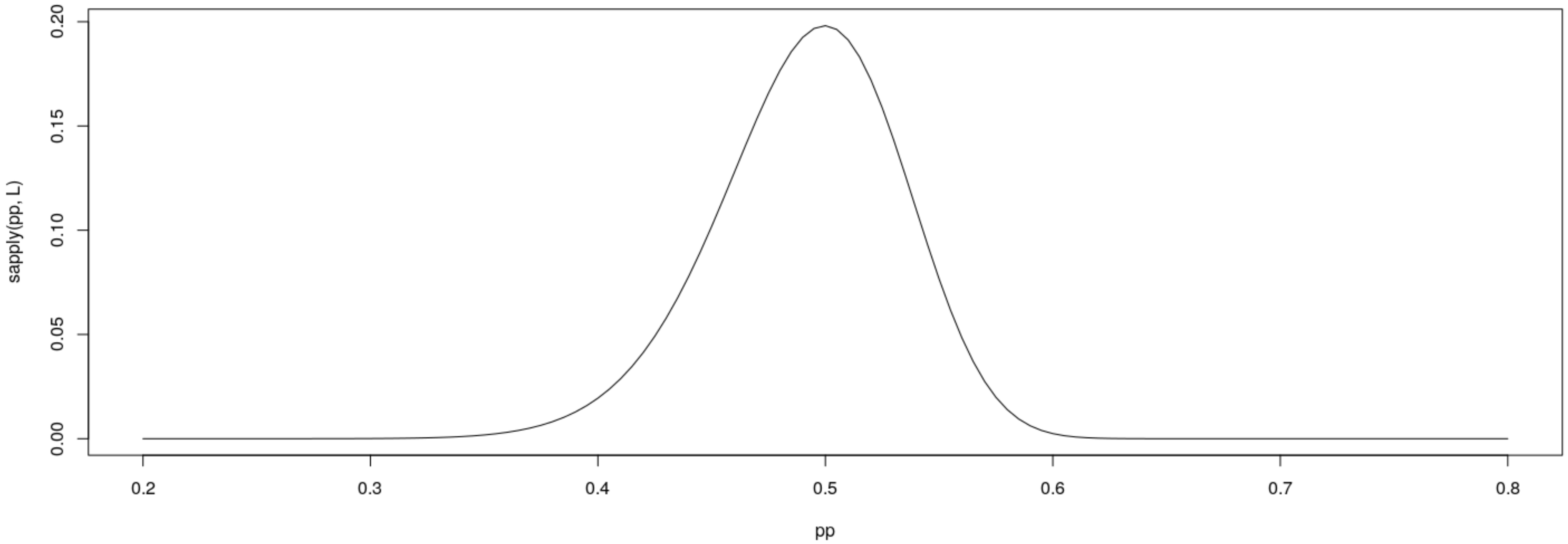
and

$$P(M = m) = P(M \le m) - P(M \le m-1)$$

In R,

```r
n = 10
N = 50
M = 30
F <- function(p, m)
{
    x = seq(0, m)
    (sum(choose(N, x) * p^x * (1-p)^(N-x)))^n
}
L = function(p)
{
    F(p, M) - F(p, M-1)
}
```

Maximum Likelihood estimate



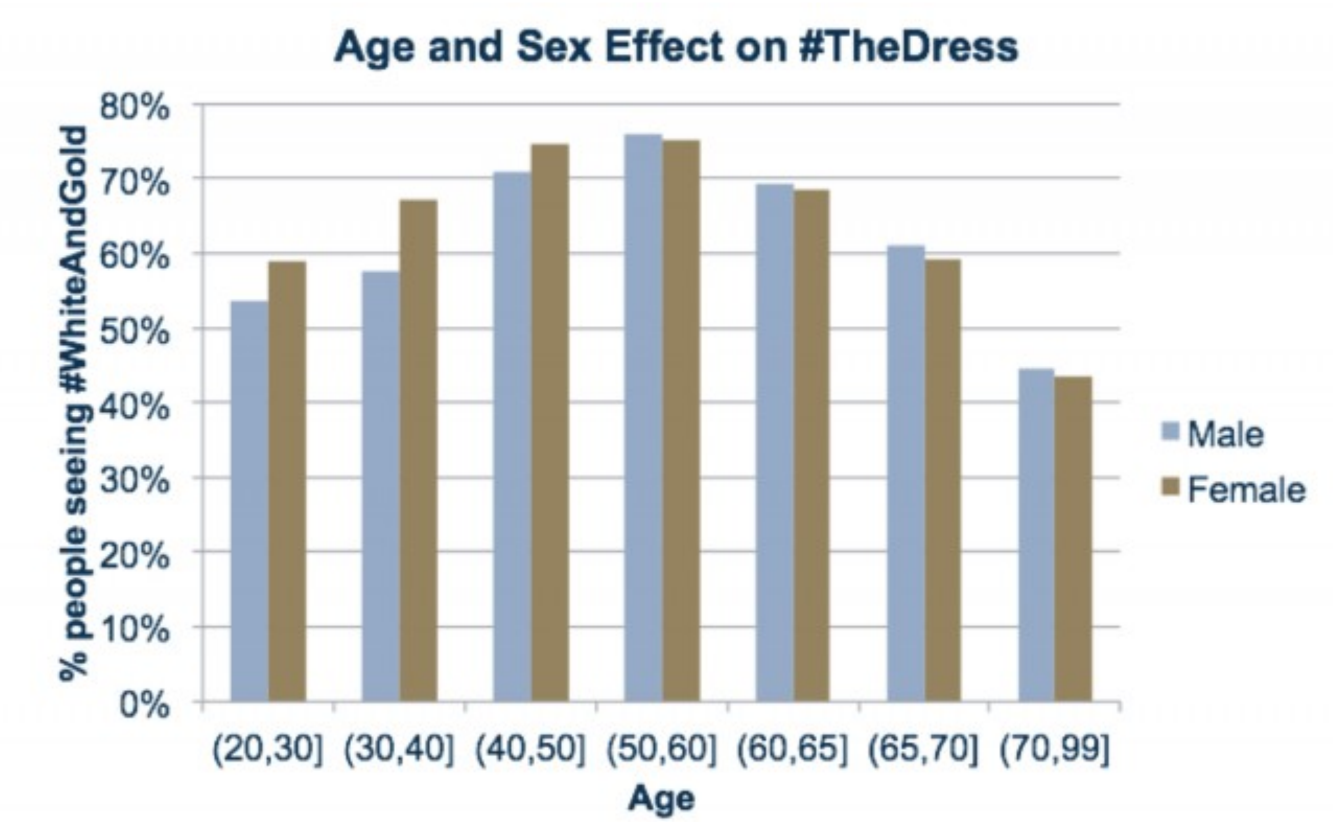
```r
optimize(L, interval = c(0, 1), maximum = TRUE)
```

```
$maximum
[1] 0.4996703

$objective
[1] 0.1981222
```

“The Dress” revisited

- What factors determine perceived color? (From 23andme.com)



Simulation: birthday problem

- R can be used to simulate random events
- Example: how likely is a common birthday in a group of 20 people?

```
N = 20
days = sample(365, N, rep = TRUE)
days
```

```
 [1] 276 278 126 350 125 165 105  15  55 266 216  71  97 335 267 317  15 345 197 192
```

```
length(unique(days))
```

```
[1] 19
```

Law of Large Numbers

- With enough replications, sample proportion should converge to probability

```r
haveCommon = function()
{
    days = sample(365, N, rep = TRUE)
    length(unique(days)) < N
}
haveCommon()
```

```
[1] FALSE
```

```r
haveCommon()
```

```
[1] FALSE
```

```r
haveCommon()
```

```
[1] FALSE
```

```r
haveCommon()
```

```
[1] FALSE
```

Law of Large Numbers

- With enough replications, sample proportion should converge to probability
- Do this sytematically:

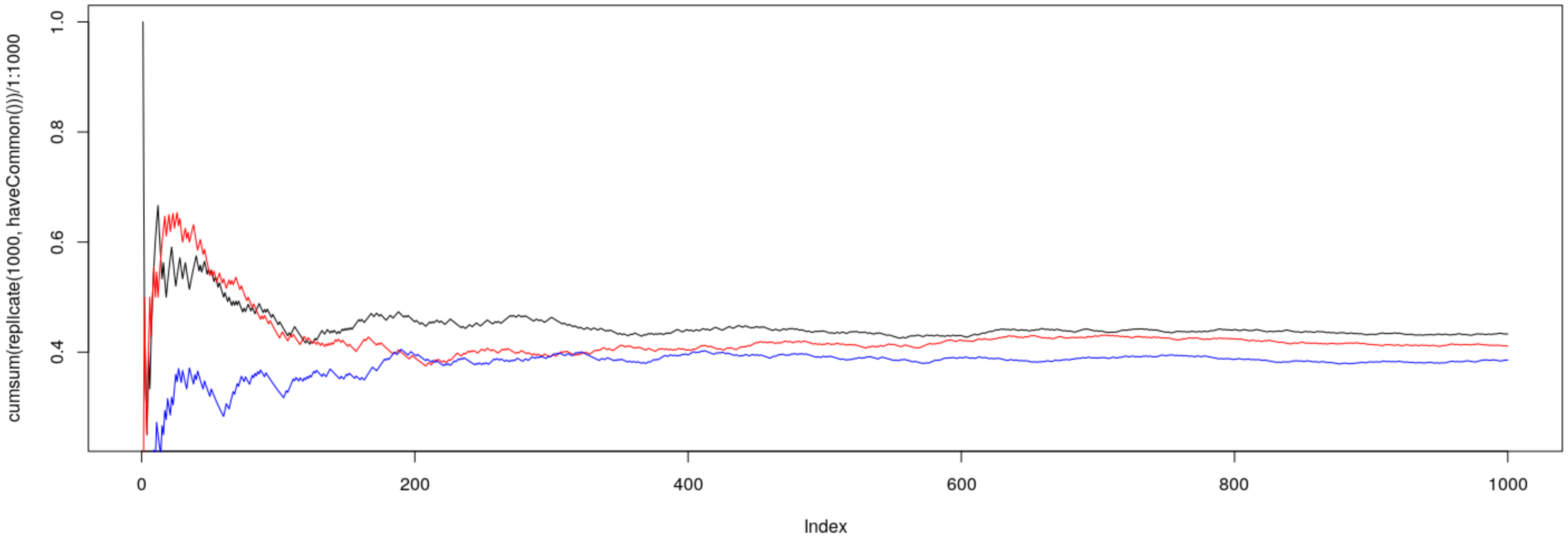
```r
replicate(100, haveCommon())
```

```
  [1]  TRUE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE  TRUE  TRUE FALSE
 [20] FALSE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE  TRUE  TRUE  TRUE FALSE
 [39]  TRUE FALSE  TRUE FALSE FALSE  TRUE FALSE  TRUE  TRUE FALSE FALSE  TRUE  TRUE FALSE  TRUE  TRUE FALSE FALSE FALSE
 [58] FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE FALSE  TRUE FALSE  TRUE  TRUE FALSE
 [77] FALSE FALSE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE  TRUE
 [96] FALSE  TRUE FALSE  TRUE  TRUE
```

Law of Large Numbers

- With enough replications, sample proportion should converge to probability

```
plot(cumsum(replicate(1000, haveCommon())) / 1:1000, type = "l")
lines(cumsum(replicate(1000, haveCommon())) / 1:1000, col = "red")
lines(cumsum(replicate(1000, haveCommon())) / 1:1000, col = "blue")
```



A more serious example: climate change

Show 10 entries Search:

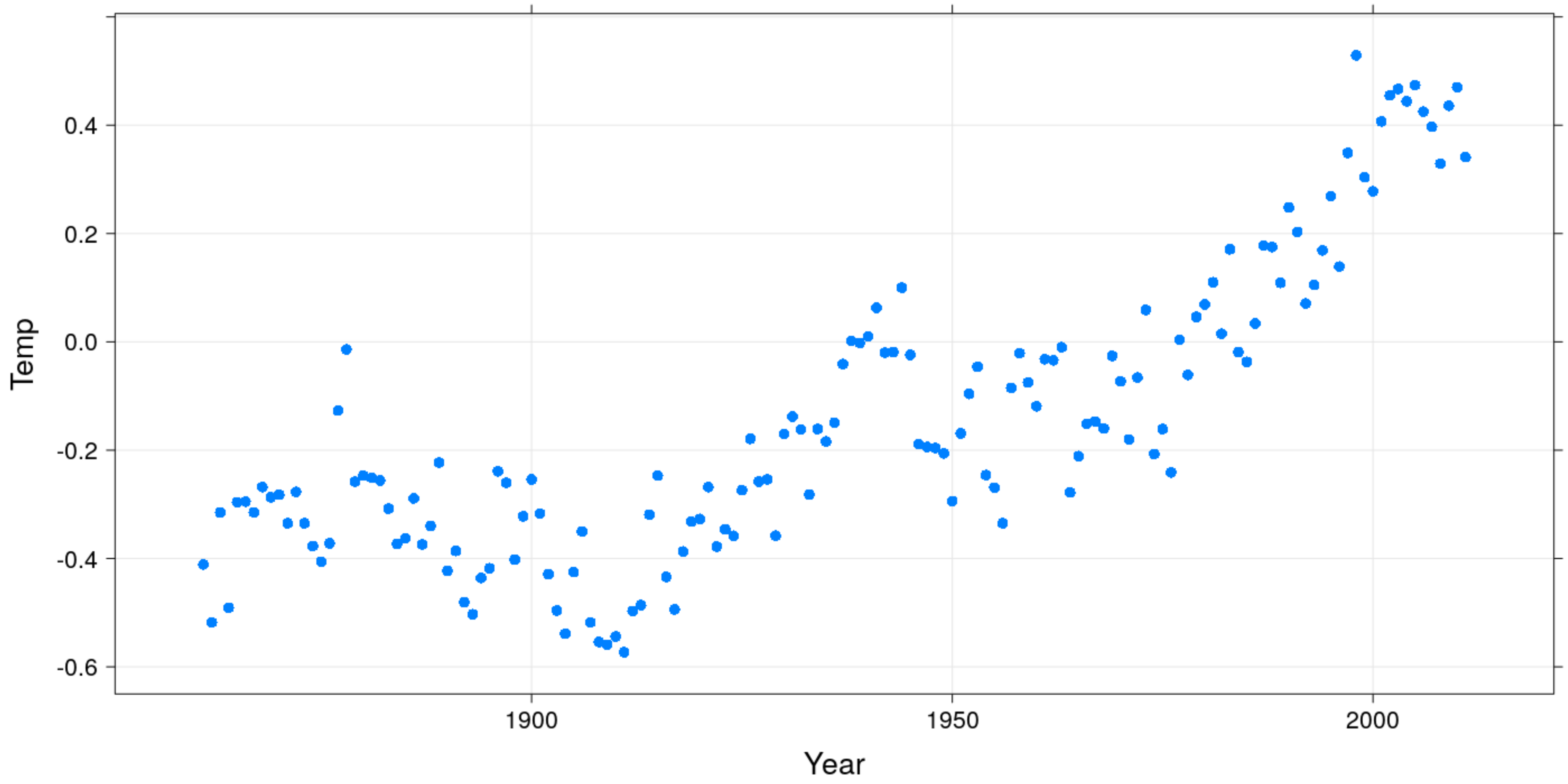
Year	Temp	CO2	CH4	NO2
1861	-0.411	286.5	838.2	288.9
1862	-0.518	286.6	839.6	288.9
1863	-0.315	286.8	840.9	289.0
1864	-0.491	287.0	842.3	289.1
1865	-0.296	287.2	843.8	289.1
1866	-0.295	287.4	845.5	289.2
1867	-0.315	287.6	847.1	289.3
1868	-0.268	287.8	848.6	289.3
1869	-0.287	288.0	850.2	289.4
1870	-0.282	288.2	851.8	289.5

Showing 1 to 10 of 151 entries

Previous 1 2 3 4 5 … 16 Next

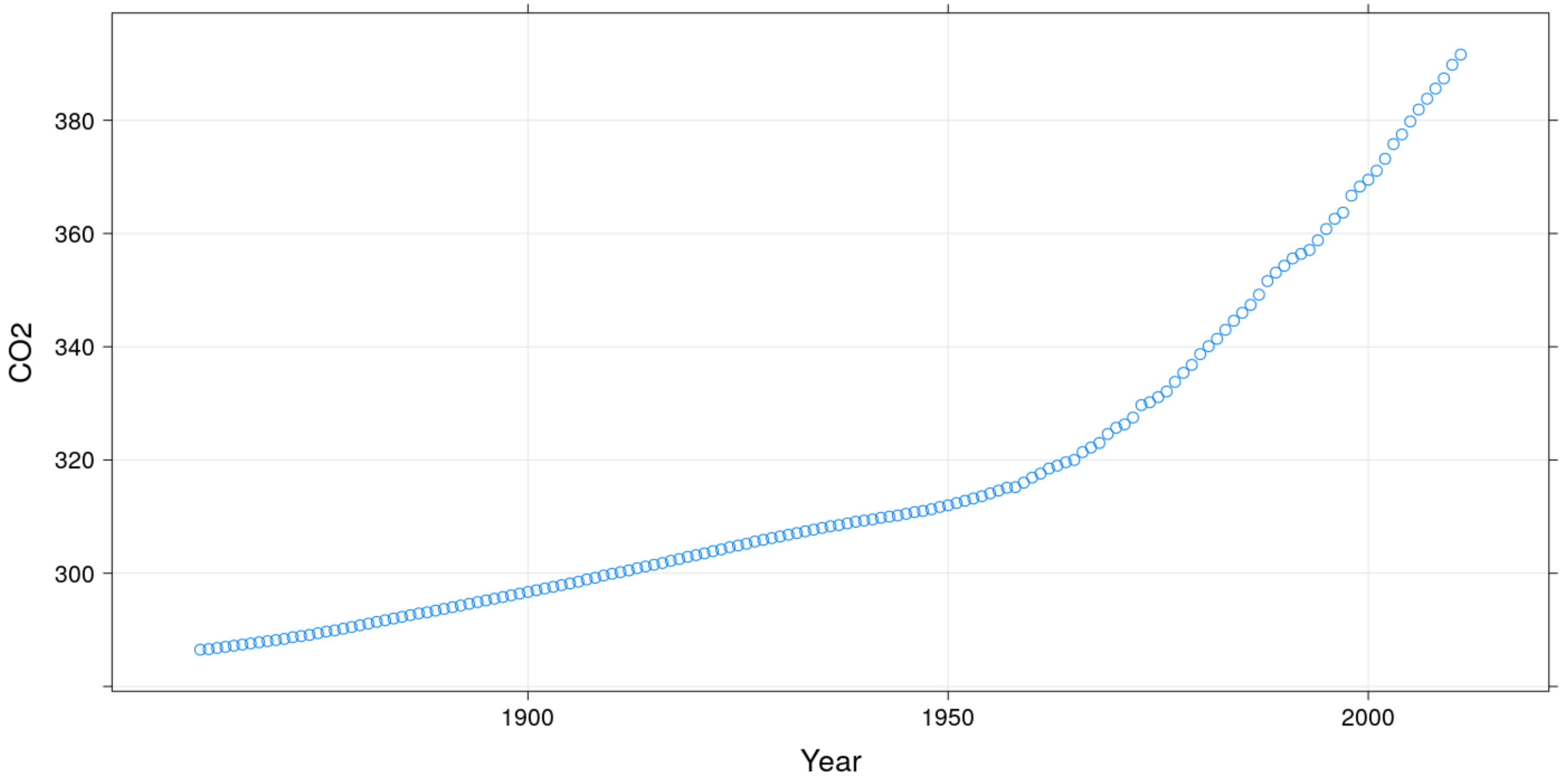
Change in temperature (global average deviation) since 1851

```
library(lattice)
xyplot(Temp ~ Year, data = globalTemp, grid = TRUE, pch = 16)
```



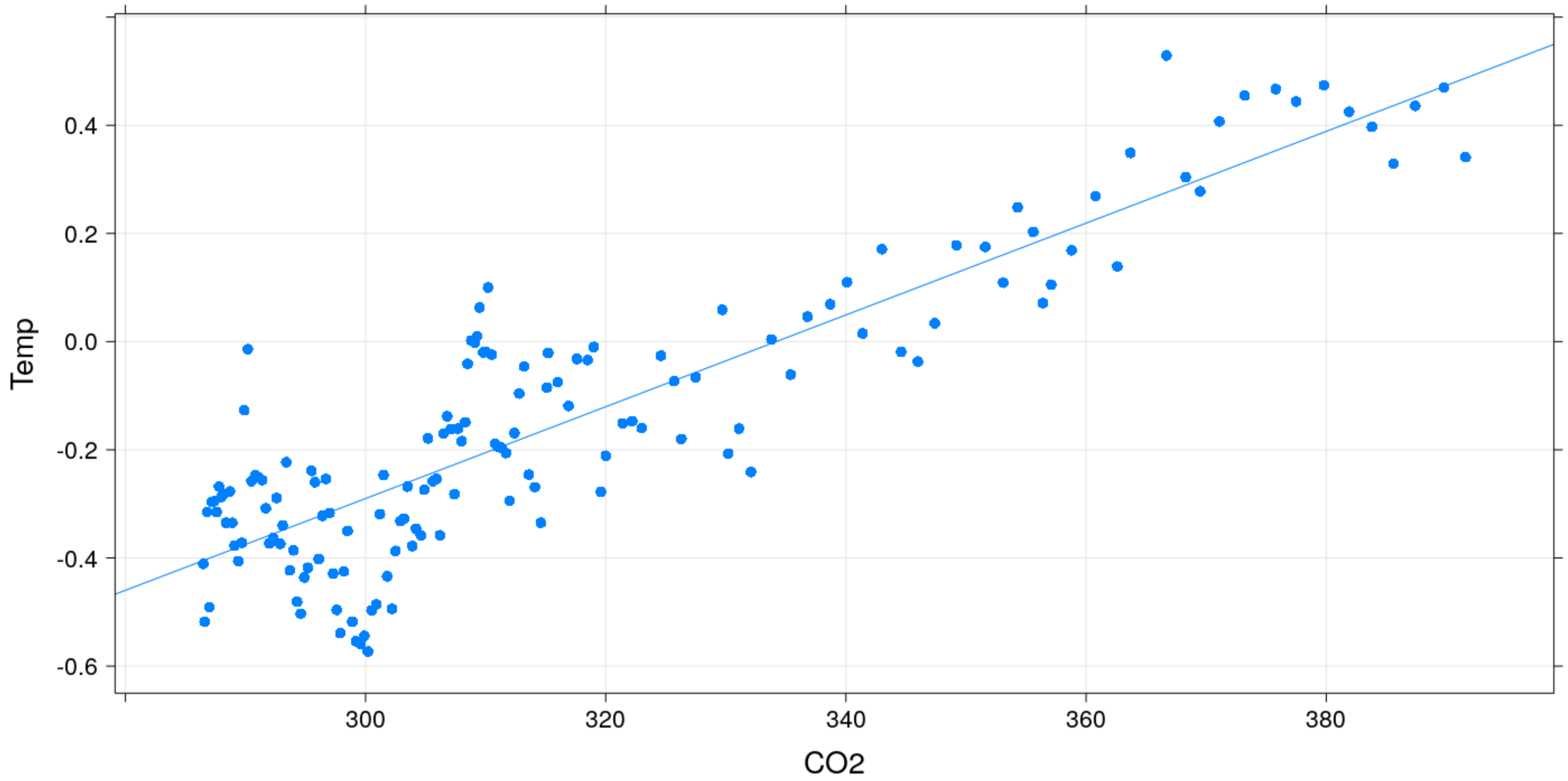
Change in atmospheric carbon dioxide

```
xyplot(CO2 ~ Year, data = globalTemp, grid = TRUE)
```



Does change in CO_2 explain temperature rise?

```
xyplot(Temp ~ CO2, data = globalTemp, pch = 16, grid = TRUE, type = c("p", "r")) # include OLS regression line
```



Fitting the “least squares” regression model

```
fm = lm(Temp ~ 1 + CO2, data = globalTemp)   # lm() fits linear models
coef(fm)                                     # coefficients of line minimizing least squared errors
```

```
 (Intercept)          CO2
-2.836082117  0.008486628
```

We can confirm using a general optimizer:

```
SSE = function(beta)
{
    with(globalTemp,     sum((Temp - beta[1] - beta[2] * CO2)^2))
}
optim(c(0, 0), fn = SSE)
```

```
$par
[1] -2.836176636  0.008486886

$value
[1] 2.210994

$counts
function gradient
      93       NA

$convergence
[1] 0

$message
NULL
```

Fitting the regression model

- The least squares problem has an exact solution (equivalent to solving a set of linear equations)
- `lm()` directly computes this exact solution
- It also gives more statistically relevant information (such as error estimates and hypothesis tests)

```r
summary(fm)
```

```

Call:
lm(formula = Temp ~ 1 + CO2, data = globalTemp)

Residuals:
     Min       1Q   Median       3Q      Max 
-0.28460 -0.09004 -0.00101  0.08616  0.35926 

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -2.8360821  0.1145766  -24.75   <2e-16
CO2          0.0084866  0.0003602   23.56   <2e-16

Residual standard error: 0.1218 on 149 degrees of freedom
Multiple R-squared:  0.7884,    Adjusted R-squared:  0.787 
F-statistic: 555.1 on 1 and 149 DF,  p-value: < 2.2e-16
```

Changing the model-fitting criteria

- But suppose we wanted to minimize *sum of absolute errors* instead of sum of squares
- No closed form solution any more, but general optimizer will still work:

```
SAE = function(beta)
{
    with(globalTemp,
         sum(abs(Temp - beta[1] - beta[2] * CO2)))
}
opt = optim(c(0, 0), fn = SAE)
opt
```

```
$par
[1] -2.832090898  0.008471257

$value
[1] 14.5602

$counts
function gradient
     123       NA

$convergence
[1] 0

$message
NULL
```

Changing the model-fitting criteria

- Compare with least squares line

```
coef(fm) # least squared errors
```

```
 (Intercept)          CO2 
-2.836082117  0.008486628 
```

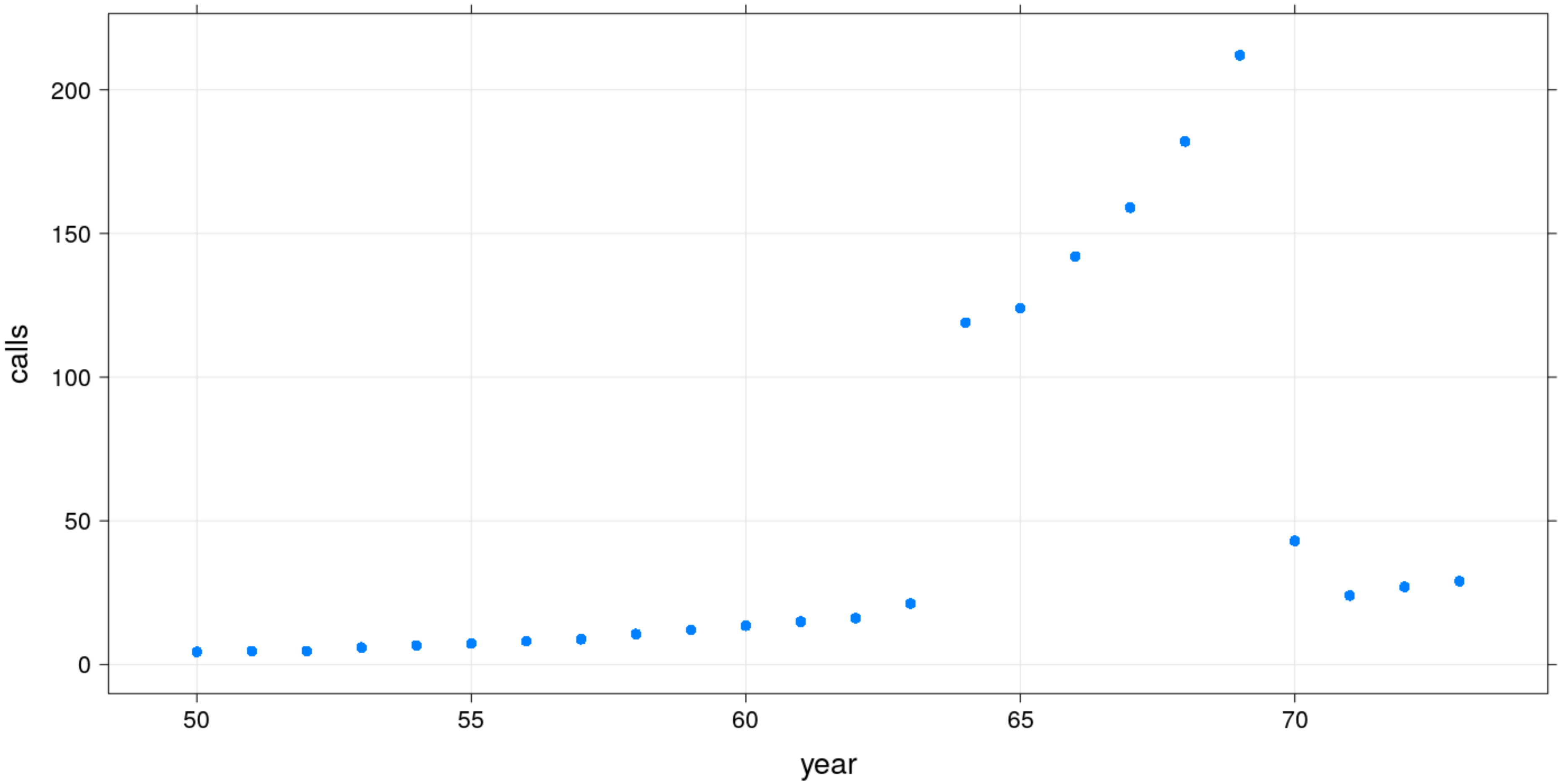
```
opt$par  # least absolute errors
```

```
[1] -2.832090898  0.008471257
```

- The two lines are virtually identical in this case
- This is not always true

Another example: number of phone calls per year in Belgium

```r
data(phones, package = "MASS")
xyplot(calls ~ year, data = phones, grid = TRUE, pch = 16)
```



Another example: number of phone calls per year in Belgium

```
fm2 <- lm(calls ~ year, data = phones)
SAE = function(beta)
{
    with(phones,
         sum(abs(calls - beta[1] - beta[2] * year)))
}
opt = optim(c(0, 0), fn = SAE)
```

```
coef(fm2) # least squared errors
```

```
(Intercept)        year 
-260.059246    5.041478 
```

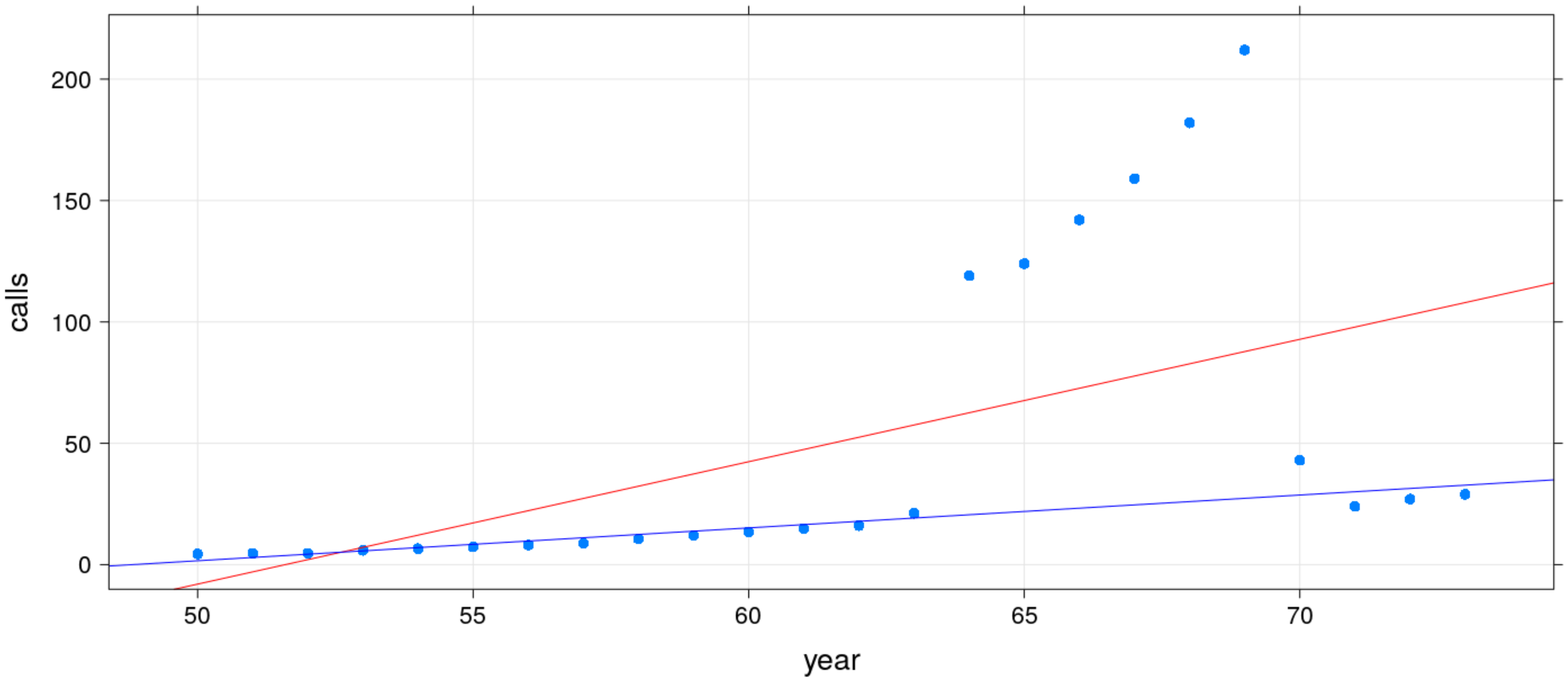
```
opt$par  # least absolute errors
```

```
[1] -66.053297   1.353735
```

- The two lines are quite different
- The second line is an example of *robust regression*

Another example: number of phone calls per year in Belgium

```
xyplot(calls ~ year, data = phones, grid = TRUE, pch = 16,
       panel = function(x, y, ...) {
           panel.xyplot(x, y, ...)
           panel.abline(fm2, col = "red") # least squared errors
           panel.abline(opt$par, col = "blue") # least absolute errors
       })
```



Take-home message

- Conventional statistical learning focuses on problems that can be “solved” analytically
- Numerical solutions are also valid solutions… but potentially difficult to obtain
- R makes it *easy* to obtain numerical solutions and compare with traditional solutions

A very brief history of R

What is R?

From its own website:

> *R is a free software environment for statistical computing and graphics.*

> *It is a GNU project which is similar to the S language and environment which was developed at Bell Laboratories (formerly AT&T, now Lucent Technologies) by John Chambers and colleagues. R can be considered as a different implementation of S.*

The origins of S

- Developed at Bell Labs (statistics research department) 1970s onwards
- Primary goals
    - Interactivity: Exploratory Data Analysis vs batch mode
    - Flexibility: Novel vs routine methodology
    - Practical: For actual use, not (just) academic research

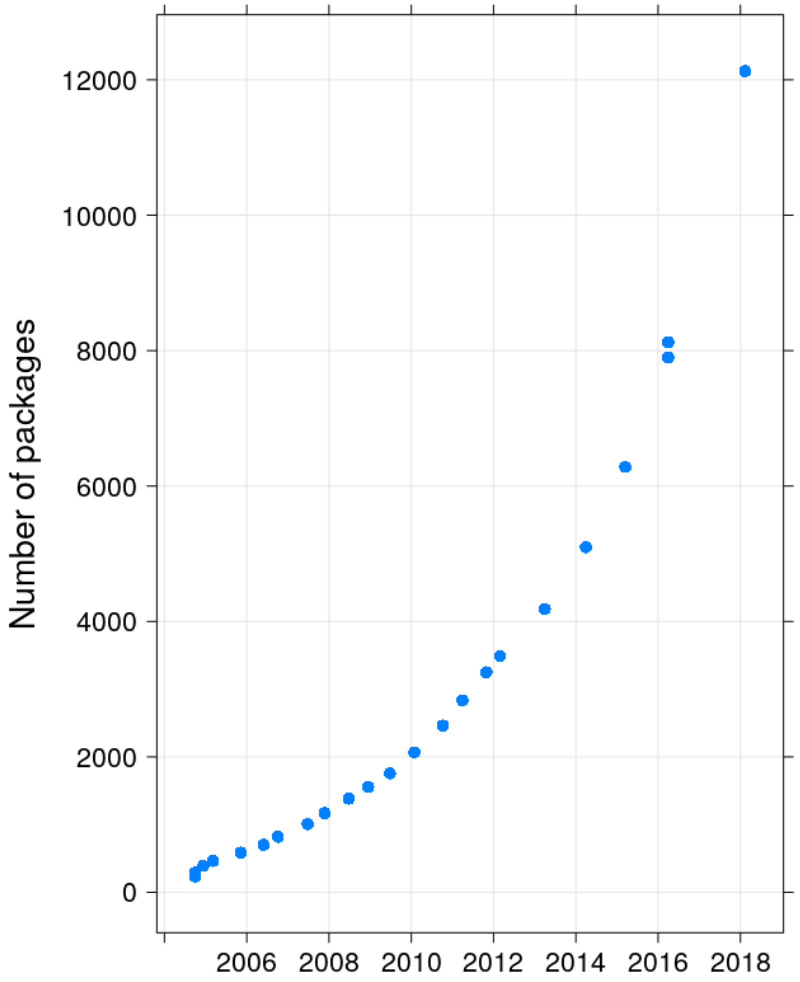
The origins of R

- Early 1990s: Started as teaching tool by Robert Gentleman & Ross Ihaka at the University of Auckland
- 1995: Convinced by Martin Maechler to release as Free Software (GPL)
- 2000: Version 1.0 released

Has since far surpassed S in popularity

One metric of popularity: Number of R packages on CRAN

- Contributed add-on packages can be submitted to "CRAN"
- The number of R packages on CRAN has grown exponentially
- Who are the creators of these packages?
    - Some are commercial entities
    - However, most are individuals like us
- The growth of R has been driven by this *community* of R enthusiasts



Why the success? The user’s perspective

- R is designed for data analysis
  - Basic data structures are vectors
  - Large collection of statistical functions
  - Advanced statistical graphics capabilities
- R “base” comes with a large suite of statistical modeling and graphics functions
- If these are not enough, more than 10000 add-on packages are available

- This is enough for the vast majority of R users who use it as a statistical toolbox
- Of course, *learning* to use these tools still requires a significant effort

The developer’s perspective

- Some R users eventually become R *developers*
- John Chambers, *Programming with Data*:

> *S can be, and is, used in a “non-programming” style, exploiting quick interaction and graphics to look at data. This use often leads to a desire to customize what you are doing, and* ***S encourages you to slide into programming, perhaps without noticing****.*

R is a full programming language

A silly example: generate Fibonacci sequence

```r
fibonacci <- function(n) {
    if (n < 2)
        x <- seq(length = n) - 1
    else {
        x <- c(0, 1)
        while (length(x) < n) {
            x <- c(x, sum(tail(x, 2)))
        }
    }
    x
}
fib20 <- fibonacci(20)
fib20
```

```
 [1]    0    1    1    2    3    5    8   13   21   34   55   89  144  233  377  610  987 1597 2584 4181
```

It is easy to call Fortran / C / C++ for efficiency

File `fib.cpp`:

```
#include <Rcpp.h>
using namespace Rcpp;

// [[Rcpp::export]]
NumericVector fibonacci_cpp(int n)
{
    NumericVector x(n);
    x[0] = 0; x[1] = 1;
    for (int i = 2; i < n; i++) x[i] = x[i-1] + x[i-2];
    return x;
}
```

Compile and call (using the `Rcpp` package):

```
Rcpp::sourceCpp("fib.cpp")
fibonacci_cpp(20)
```

```
 [1]    0    1    1    2    3    5    8   13   21   34   55   89  144  233  377  610  987 1597 2584 4181
```

It is easy to call Fortran / C / C++ for efficiency

- The result can be further analyzed using standard R tools

```
x <- fibonacci_cpp(100)
```

- The ratio of successive Fibonacci numbers converge to the “golden ratio”

$$\frac{a}{b} = \frac{a+b}{a} = \varphi = \frac{1+\sqrt{5}}{2} = 1.6180339887$$

```
r <- x[-1] / x[-100]
print(r[1:40], digits = 12)
```

```
 [1]           Inf 1.00000000000 2.00000000000 1.50000000000 1.66666666667 1.60000000000 1.62500000000 1.61538461538
 [9] 1.61904761905 1.61764705882 1.61818181818 1.61797752809 1.61805555556 1.61802575107 1.61803713528 1.61803278689
[17] 1.61803444782 1.61803381340 1.61803405573 1.61803396317 1.61803399852 1.61803398502 1.61803399018 1.61803398821
[25] 1.61803398896 1.61803398867 1.61803398878 1.61803398874 1.61803398875 1.61803398875 1.61803398875 1.61803398875
[33] 1.61803398875 1.61803398875 1.61803398875 1.61803398875 1.61803398875 1.61803398875 1.61803398875 1.61803398875
```

Strengths of R in a nutshell: flexibility and extensibility

- Powerful built-in tools
- Programming language
- Compiled code for efficiency

According to Hal Varian (quoted in a New York Times article)

> *The great beauty of R is that you can modify it to do all sorts of things, And you have a lot of prepackaged stuff that's already available, so you're standing on the shoulders of giants.*

Parting comments: reproducible documents

- Creating reports / presentations with numerical analysis is usually a two-step process:
    - Do the analysis using a computational software
    - Write report in a word processor, copy-pasting results
- R makes it very convenient to write “literate documents” that contain both analsyis code and report text
- Basic idea:
    - Start with source text file containing code+text
    - Transform file by running code and embedding results
    - Produces another text file (LaTeX, HTML, markdown)
    - Processed further using standard tools
- Example: this presentation is created from this source file (R Markdown) using knitr and pandoc
- As the source format is markdown, output could also be PDF instead of HTML