and a bit of statistics
why?

fun
get
work
done

python
ruby
perl
go
C
C++

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

fun
get
work
done

fun
get
work
done

python
ruby

perl
go

C
C++

fast

maple
matlab
R

R

18/03/2019 [ 3 ]
invert matrix, symbolic maths, curve fitting, plotting

get work done

fast

ruby, python
perl, go
C, C++

maple, matlab, R

niche languages
R gives you

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

Horrible syntax

Starting and stopping
- \texttt{R}, \texttt{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 – needs root – good for personal use
- Rcmdr (works on our teaching pool), eclipse
- emacs ESS mode

Code / Scripts / Interactive – not just interpreter

<table>
<thead>
<tr>
<th>Language</th>
<th>Features</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>write, compile, run</td>
<td>compiler</td>
</tr>
<tr>
<td>Python</td>
<td>write, run</td>
<td>interpreter</td>
</tr>
<tr>
<td>R</td>
<td>play line by line + write, run</td>
<td>interpreter</td>
</tr>
</tbody>
</table>

a bit more ..
### interpreter, interactive?

<table>
<thead>
<tr>
<th>classic scripting</th>
<th>no programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>use an editor / write something complex</td>
<td>R</td>
</tr>
<tr>
<td>run Rscript</td>
<td>Rcmdr, Rstudio</td>
</tr>
<tr>
<td>play with command line</td>
<td>Read +display data, like spreadsheet</td>
</tr>
<tr>
<td></td>
<td>do manipulations from GUI</td>
</tr>
</tbody>
</table>
Structural differences

C, C++, python

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

```r
> 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

- \( a \leftarrow b \times c \) or \( x \%\% y \) or big vectors / matrices – very fast
  - code is recognised, runs hand-crafted routines
  - `for (i in 1:length(a)) { c[i] = a[i] * b[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 two lectures – base R

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

Technical...
Operators

- \( a = 1 \) or \( a \leftarrow 1 \)  
  R people like \( \leftarrow \)
- \(+ - * /\) 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)
- \( \text{mean}() , \max() , \text{median}() , \text{sum}() , \ldots \)
  - 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, \texttt{c()}

- \texttt{?c} will tell you what it does The default method combines its arguments to form a vector.

\begin{verbatim}
> x <- c(1, 2, 3); str(x)
  num [1:3] 1 2 3
\end{verbatim}

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

- where do they come from?
- \texttt{vector()} \texttt{?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
- \( x[2:4] \)
- \( x[1:4] \) elements, 1, 2, 3, 4

• logical versions – compact filtering

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

\[ m \leftarrow \text{matrix}(x, \text{nrow} = 3, \text{ncol} = 4) \]

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

\[
\begin{align*}
> & x \leftarrow c(4, 5, 6) ; y \leftarrow c(7, 8, 9) \\
> & m \leftarrow \text{cbind}(x, y) ; m \\
> & x \ y \\
[1,] & 4 \ 7 \\
[2,] & 5 \ 8 \\
[3,] & 6 \ 9 \\
> & n \leftarrow \text{rbind}(x, y) ; n \\
> & [,1] \ [,2] \ [,3] \\
& x \ 4 \ 5 \ 6 \\
& y \ 7 \ 8 \ 9
\end{align*}
\]
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

```r
> 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
[1] "a" "b"

# There are two things in l, both are vectors

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

- very foreign to C, other languages
  - hash + array? more generally hash + more general type
- very natural for data

```r
> 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
```

$ cat data.txt
andrew mary
3 4
5 6
7 8
• If you must, get to rows, 

```r
> 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

<table>
<thead>
<tr>
<th></th>
<th>Sepal.Length</th>
<th>Sepal.Width</th>
<th>Petal.Length</th>
<th>Petal.Width</th>
<th>Species</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5.1</td>
<td>3.5</td>
<td>1.4</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>2</td>
<td>4.9</td>
<td>3.0</td>
<td>1.4</td>
<td>0.2</td>
<td>setosa</td>
</tr>
</tbody>
</table>

> 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

```r
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')
}
```
junk$ Rscript z.r
answer is 18

• pass by value
• very often operate on whole vectors

$ cat f.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 $\times 10^2$
- 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:

\[
\log(x, \text{base} = \exp(1))
\]

...  

• \log() takes two arguments, \log(x, b) so \log(x, 10) is \log_{10} x
• there is a default for the second argument so \text{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?
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")

points(density (iris$Petal.Width))
Enough syntax

Next week

• some statistics, fitting, ..