R and performance

BSC-ES-CES training talks
18/05/2015

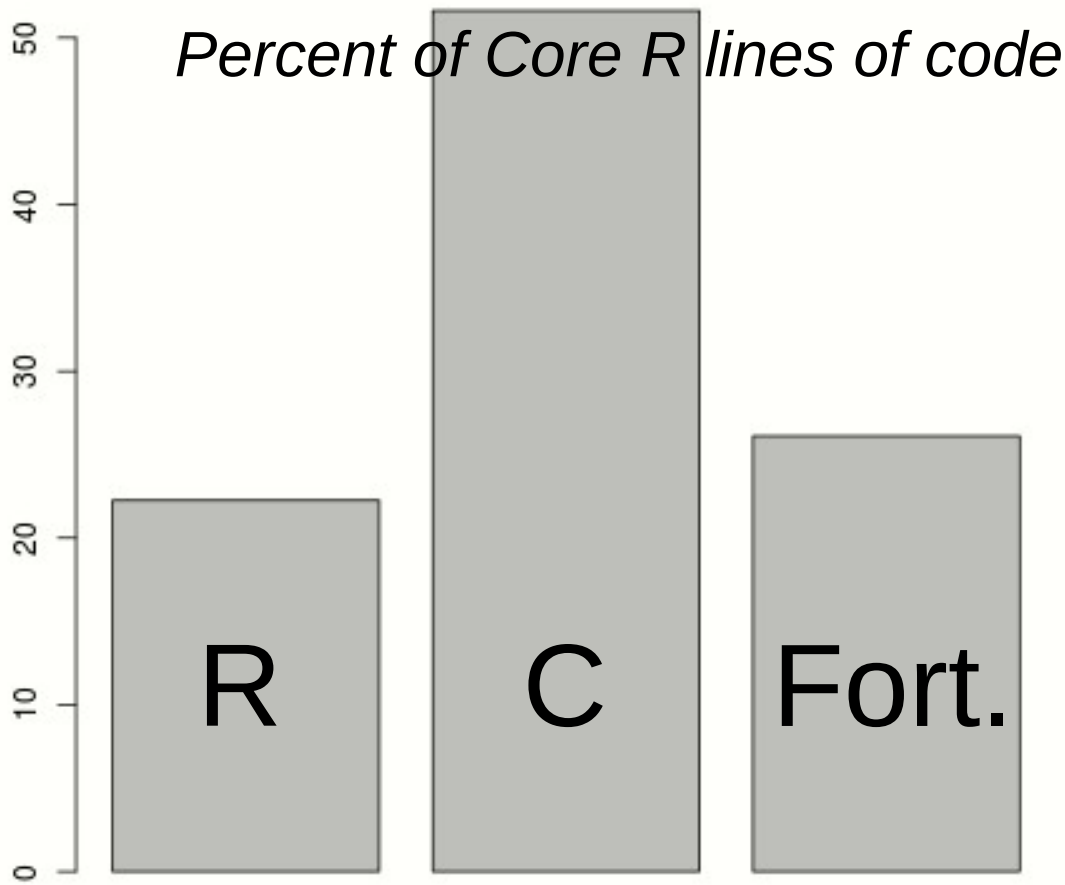
nicolau.manubens@ic3.cat

Outline

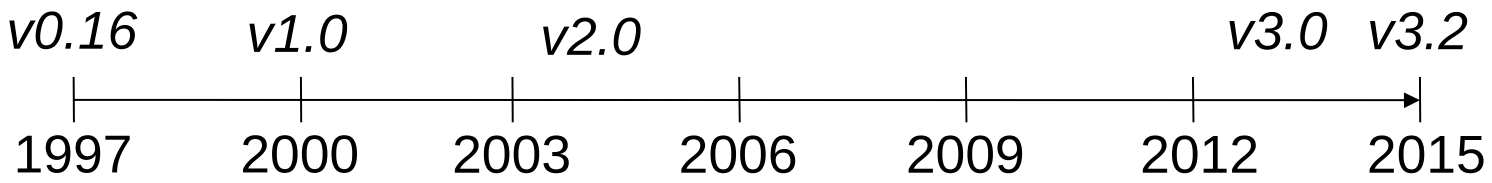
- About R
- Popularity
- Drawbacks
- Coding fast R
- Conclusions

About R

- Interpreted language
- Under GPL license
- Cross-platform
- Designed for statistics and graphics
- CRAN, package repo

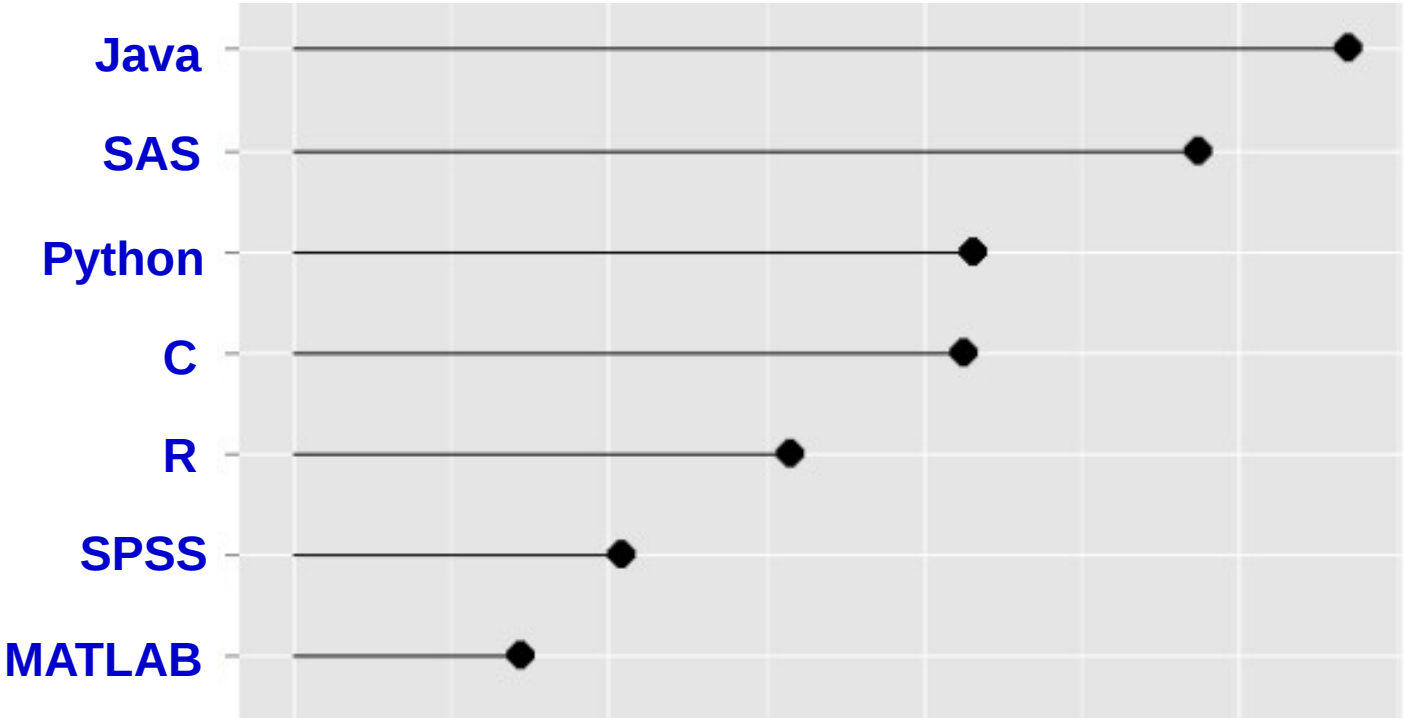


Origin (R. Ihaka, R. Gentelman)



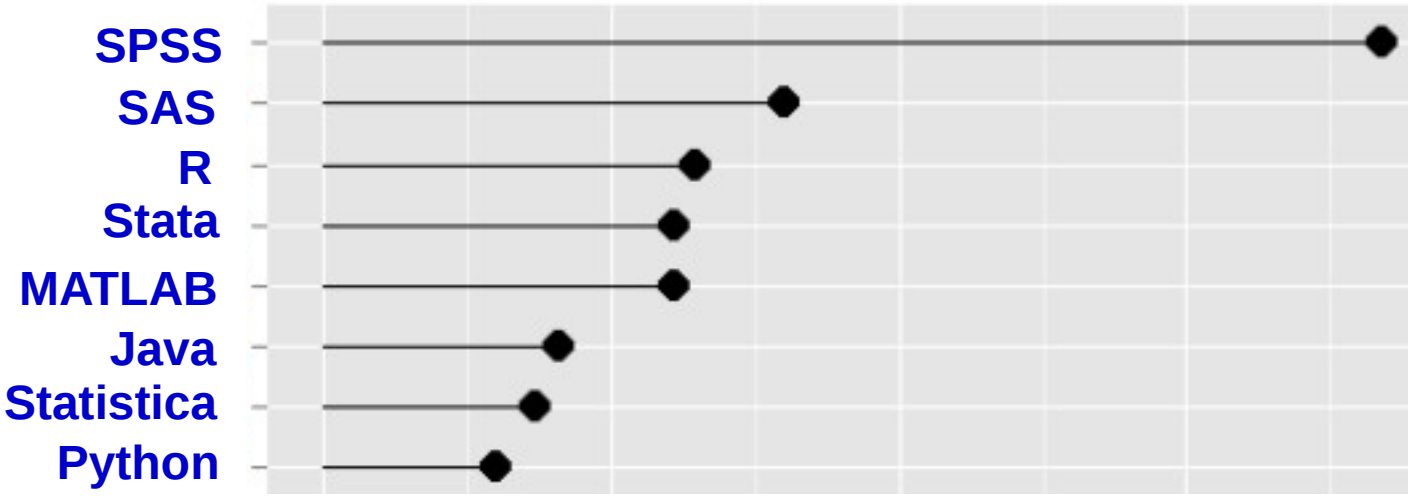
Popularity

- One of most widely used programming language for Data Analytics



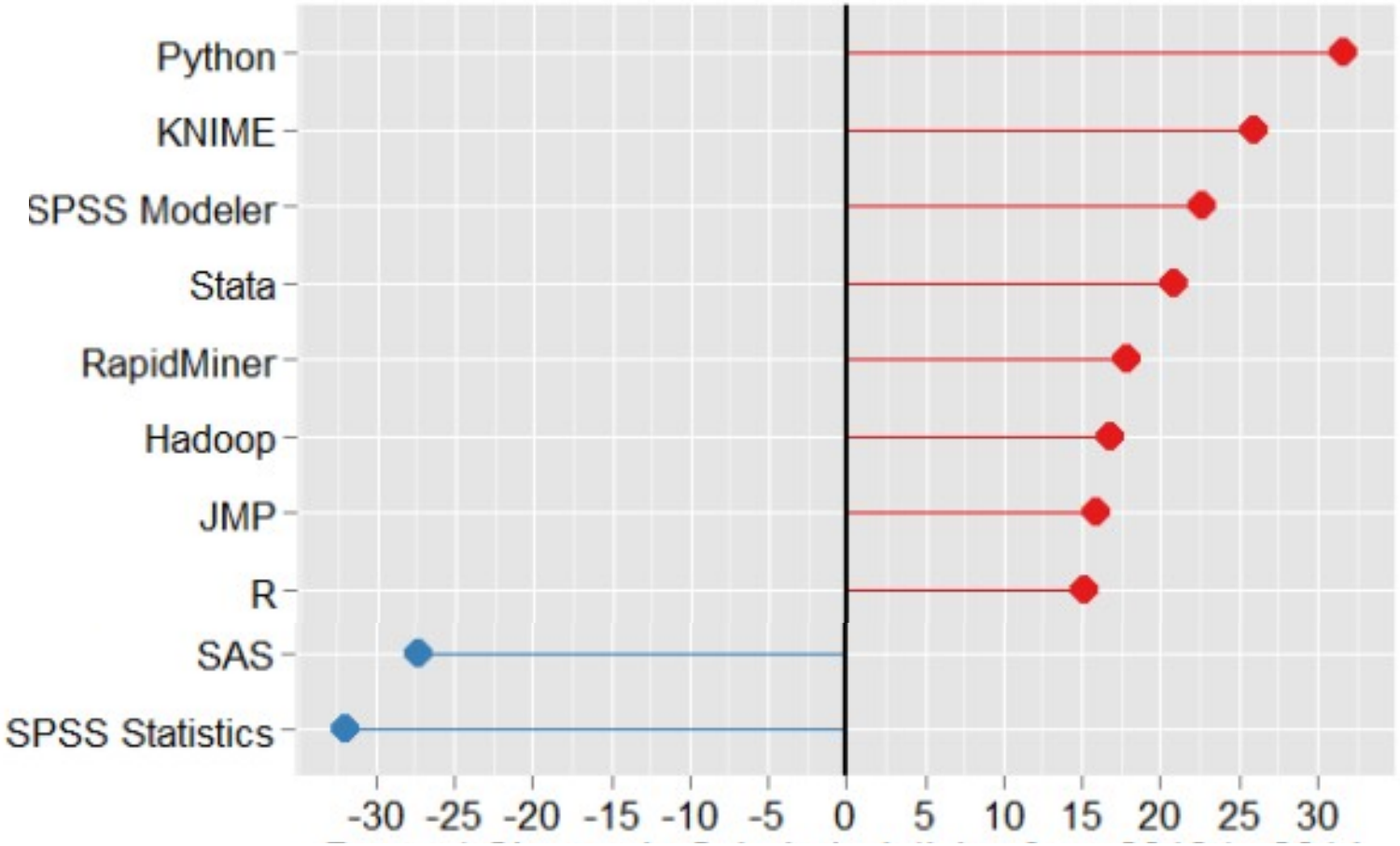
Number of Analytics jobs in indeed.com (1 cell = 1 k)

Popularity



Number of Scholarly articles in 2014 (1 cell = 10 k)

Popularity



Percent change in Scholarly articles from 2013 to 2014

Drawbacks

- Known for being slow
  - Extremely dynamic (interpreted)
  - Implicit memory management
- Known for having issues with large data
  - Allocates active data in main memory

There are some ways around

mean_23: **21.225 s**

Coding fast R

- Compilation
- Vectorisation
- Wrapping languages
- Parallelisation
- Other solutions

Study case:
«Calculate the mean over the 2nd and 3rd dimensions of a 4-dimensional array»

```
mean_23 <- function(a) {
  b <- array(dim = c(dim(a)[1], dim(a)[4]))
  n <- dim(a)[2] * dim(a)[3]
  for (i in 1:dim(a)[1]) {
    for (j in 1:dim(a)[4]) {
      layer_sum <- 0
      for (k in 1:dim(a)[2]) {
        for (l in 1:dim(a)[3]) {
          layer_sum <- layer_sum + a[i, k, l, j]
        }
      }
      b[i, j] <- layer_sum / n
    }
  }
  return(b)
}
```

Average elapsed time:
21.225 s

mean_23: **21.225 s**
mean_23_comp: **10.120 s**

Coding fast R

- Compilation
  - Function compilation

```
mean_23_comp <- compiler::cmpfun(mean_23)
```

Average elapsed time:
10.120 s

mean_23: **21.225 s**
mean_23_comp: **10.120 s**
mean_23_vec: **0.414 s**
mean_23_hyper_vec: **0.126 s**

Coding fast R

- Vectorisation
  - *«Using R basic optimized functions»*

```
mean_23_vec <- function(a) {
  apply(a, c(1, 4), mean)
}
```

Average elapsed time:
0.414 s

  - *Or even highly optimized functions*

```
mean_23_hyper_vec <- function(a) {
  rowMeans(aperm(a, c(1, 4, 2, 3)), dims = 2)
}
```

Average elapsed time:
0.126 s

mean_23: **21.225 s**
mean_23_comp: **10.120 s**
mean_23_vec: **0.414 s**
mean_23_hyper_vec: **0.126 s**
mean_23_fort: **0.138 s**

Coding fast R

- Wrapping other languages
  - Cpp, C, Fortran

```
mean_23_fort <- function(a) {
  dyn.load('mean_23_fort.so')
  result <- .Fortran("mean_23_fort", ...)$b
  return(array(result$b, dim = c(...)))
}
```

```
      subroutine mean_23_fort(ndims,dims,la,a,lb,b)
      integer ndims, dims(0:ndims-1), la, lb, x, y
      double precision a(0:(la-1)), b(0:(lb-1)), sum
      do 20 i = 0, (dims(0)-1)
        do 15 j = 0, (dims(3)-1)
          sum = 0
          do 10 k = 0, (dims(1)-1)
            do 5 l = 0, (dims(0)-1)
              x = i*dims(1)*dims(2)*dims(3)
              y = k*dims(2)*dims(3)
              sum = sum + a(x + y + l*dims(3) + j)
 5          continue
 10       continue
          b(i*dims(3) + j) = sum/lb
 15     continue
 20   continue
      end
```

Average elapsed time:
0.138 s

mean_23: **21.225 s**
mean_23_comp: **10.120 s**
mean_23_vec: **0.414 s**
mean_23_hyper_vec: **0.126 s**
mean_23_fort: **0.138 s**
mean_23_par: **7.734 s**

Coding fast R

- Parallelisation
  - Lots of packages to run on multi-core, multi-processor, GPU and clusters.

```
mean_23_par <- function(a) {
  n_cores <- 4
  split_iterations <- seq(1, dim(a)[1], length.out = ncores)
  for (i in length(split_iterations)) {
    jobs <- mcparallel({mean_23(
                a[split_iterations[i]:min(split_iterations[i + 1], dim(a)[1]), , , ]
              )})
  }
  result <- mccollect(jobs)
  ...
}
```

Average elapsed time:
7.734 s

Coding fast R

- Other solutions
  - Avoid memory duplications
  - Recompile R with special compilers
  - Recompile or install new algebra libraries (BLAS → ATLAS, OpenBLAS)
  - Keep R up-to-date
  - Install R versions tuned for speed
  - Look for already existing solutions

Conclusions

- Always **look for existing functions** that do your work (can be complex, many contributions)
- **Otherwise** can always achieve good performance **wrapping Fortran or C**
- Moving completely to other language means losing “shareability”. **Wrapping only computing-intensive functions seems a good deal**.

Thank you for your attention!

nicolau.manubens@ic3.cat