Experiencia de un bioinfoRmático con Julia

Diego Javier Zea, Noviembre de 2013

Julia

- Julia es un lenguaje dinámico de alto nivel.
- *Performance* cercana a **C**.
- Hace sencillo la utilización de bibliotecas en **C**.
- Incorpora bibliotecas de **C** y **Fortran** que son lo mejor en el estado del arte en álgebra lineal, generadores de números aleatorios, procesamiento de textos, *FFTs*...
- Es posible hacer metaprogramación a través de *macros*.
- Ejecución paralela (distribuida).
- Los tipos de datos y las funciones definidas por el usuario tienen las misma *performance* que los que vienen por defecto.

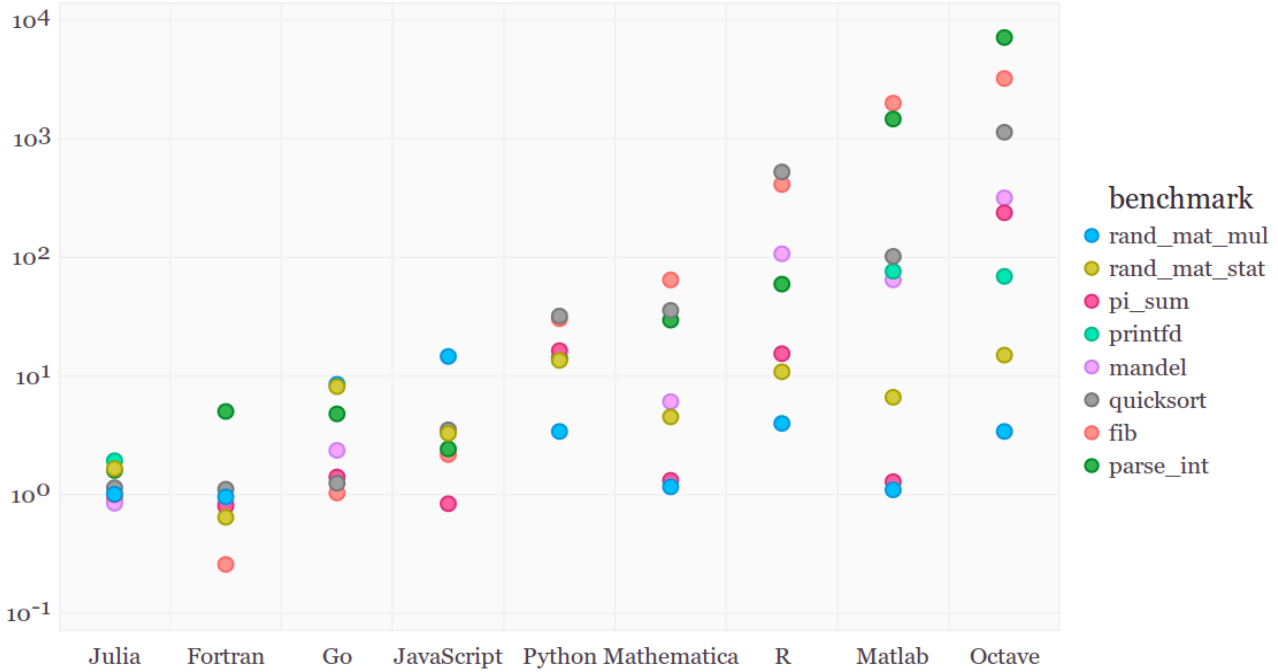
Que queremos...

“We are greedy: we want more.

We want a language that’s **open source**, with a liberal license. We want the **speed of C** with the **dynamism of Ruby**. We want a language that’s **homoiconic, with true macros like Lisp**, but with obvious, familiar **mathematical notation like Matlab**. We want something as usable for **general programming as Python**, a**s easy for statistics as R**, as natural for s**tring processing as Perl**, as powerful for **linear algebra as Matla**b, as good at **gluing programs together as the shel**l. Something that is dirt **simple to learn**, yet keeps the most serious hackers happy. We **want it interactive** and **we want it compiled**.

(Did we mention it should be as fast as C?)”

Alta *performance* de Julia



Performance comparada a **C** ($y=1$) en escala logarítmica. **Julia** 0.2 en este *benchmark* esta entre 0.85 y 1.66 veces C. **R 3.0.2** varía entre 3.98 y 524.29 veces C. **Python 2.7.3** varía entre 3.41 y 31.98 veces C. Es gráfico realizado con **Gadfly**

http://julialang.org/benchmarks/

R mirando a **Julia**

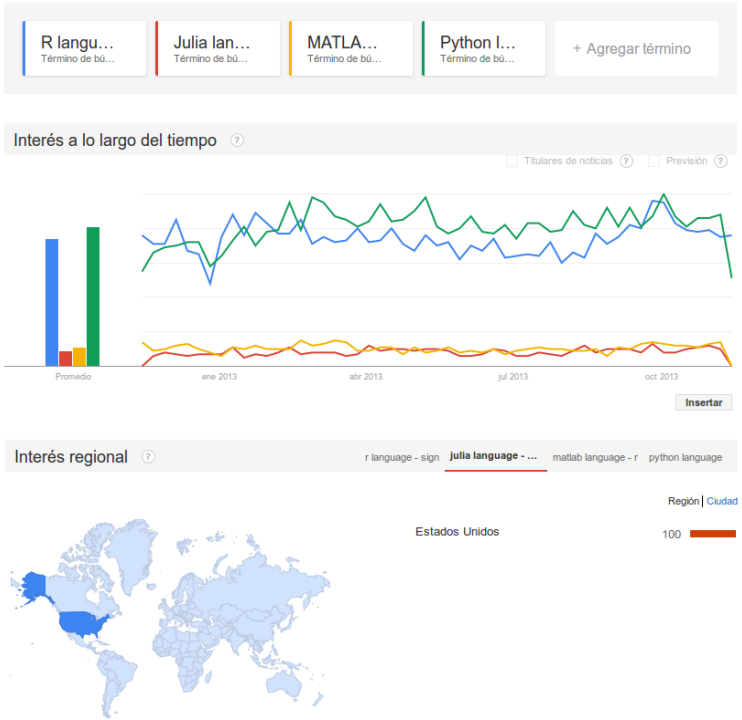
1. Top 100 R posts of 2012

R-bloggers' success is thanks to the content submitted by the over 400 R bloggers who have joined r-bloggers. The R community currently has around **245 active R bloggers** (links to the blogs are clearly visible in the right navigation bar on the R-bloggers homepage). In the past year, these bloggers wrote around 3200 posts about R!

Here is a **list of the top visited posts** on the site in 2012 (you can see the number of page views in parentheses):

1. Select operations on R data frames (42,742)
2. Julia, I Love You (22,405)
3. R at 12,000 Cores (22,584)
4. An R programmer looks at Julia (17,172)

R mirando a **Julia**



Un 4.38% de los **paquetes**

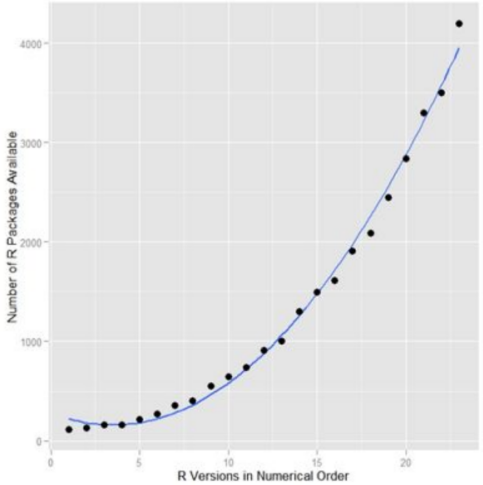
November 08, 2013

CRAN now has 5000 R packages

Prof. Ripley today announced on the r-devel mailing list that CRAN now has it's 5000th R package:

> Package 'quint' brought the number of packages on CRAN (for all platforms: some are Windows-only or non-Windows only) to 5000 a few minutes ago: see http://cran.r-project.org/web/packages/index.html.

That's quite a milestone! The number of CRAN packages has been increasing rapidly recently, as the chart below from The Popularity of Data Analysis Software shows. There were about 4200 pacakges available when R 2.15.2 was released on October 26, 2012.



Desde Agosto de 2012, **Julia** fue acumulando **219 paquetes** en METADATA.jl gracias a la contribución de más de 140 personas.

PyCall: Llenando la brecha

PyCall.jl es un paquete de **Julia,** creado por Steven Johnson, que permite llamar funciones de **Python** o utilizar sus módulos. Permite, entre otras cosas, usar **BioPython** desde Julia.

```
using PyCall

@pyimport Bio.Seq as PySeq

@pyimport Bio.Alphabet as PyAlphabet

function pyocre(seq::ASCIIString)
    PySeq.Seq(seq, PyAlphabet.generic_dna)[:find]("TAA") != -1

end
```

Un **Bio** para Julia

```
1  seq = "GATTACA"
2
3  using PyCall
4  @pyimport Bio.Seq as PySeq
5  @pyimport Bio.Alphabet as PyAlphabet
6  function pyocre(seq::ASCIIString)
7      PySeq.Seq(seq, PyAlphabet.generic_dna)[:find]("TAA") != -1
8  end
9  pyocre( seq )
10
11 using BioSeq
12 jlocre(seq::ASCIIString) = search(nucleotide(seq),"TAA").start != 0
13 jlocre( seq )
14
15 using Benchmark
16 py() = pyocre( "ACTG"[rand(1:4,100)] )
17 jl() = jlocre( "ACTG"[rand(1:4,100)] )
18 compare([py,jl],100)
19
```

Un **Bio** para Julia

```
julia> seq = "GATTACA"
"GATTACA"

julia> using PyCall

julia> @pyimport Bio.Seq as PySeq

julia> @pyimport Bio.Alphabet as PyAlphabet

julia> function pyocre(seq::ASCIIString)
         PySeq.Seq(seq, PyAlphabet.generic_dna)[:find]("TAA") != -1
       end
pyocre (generic function with 1 method)

julia> pyocre( seq )
false

julia> using BioSeq

julia> jlocre(seq::ASCIIString) = search(nucleotide(seq),"TAA").start != 0
jlocre (generic function with 1 method)

julia> jlocre( seq )
false

julia> using Benchmark

julia> py() = pyocre( "ACTG"[rand(1:4,100)] );

julia> jl() = jlocre( "ACTG"[rand(1:4,100)] );

julia> compare([py,jl],100)
2x4 DataFrame:
       Function     Elapsed Relative Replications
[1,]       "py"  0.00835356  23.2473          100
[2,]       "jl" 0.000359334      1.0          100
```

BioSeq.jl es paquete para trabajar con secuencias biológicas en *Julia*. Otorgando funcionalidad similar a *Seq* de *BioPython* sin perder *performance*.

Un **Bio** para Julia

FastaIO.jl

Inspirado en **kseq.h** para ser eficiente, permite pasear y escribir archivos *Fasta* (incluso comprimidos) desde *Julia*.

Phylogenetics.jl

Inspirado en el paquete **ape** de **R**, contiene herramientas para representar y manipular árboles filogenéticos.

Julia mira a R

Julia para el análisis estadístico de datos

DataFrames.jl

DataFrames no sólo introduce una estructura de datos tabulares como los *Data Frames* de **R**, sino que también introduce el tipo de datos **NA** y **Formula**.

```
julia> run(`cat CAF1M_YERPE.txt`)
Source  Domain  Start   End
sig_p   n/a     1       23
Pfam A  PapD_N  36      171
disorder        n/a     135     137
Pfam A  PapD_C  190     253
julia> using DataFrames

julia> data = readtable("CAF1M_YERPE.txt", separator='\t', nastrings=["n/a"])
4x4 DataFrame:
             Source   Domain Start End
[1,]       "sig_p"       NA     1  23
[2,]      "Pfam A" "PapD_N"    36 171
[3,]    "disorder"       NA   135 137
[4,]      "Pfam A" "PapD_C"   190 253


julia> data = data[ :( ! isna(Domain) ) , : ]
2x4 DataFrame:
          Source   Domain Start End
[1,]    "Pfam A" "PapD_N"    36 171
[2,]    "Pfam A" "PapD_C"   190 253


julia> Formula( :( Start ~ End ) )
Formula: Start ~ End

julia>
```

Datasets de **R** y modelos lineales

```
julia> using RDatasets

julia> form = data("datasets", "Formaldehyde")
6x2 DataFrame:
        carb optden
[1,]     0.1  0.086
[2,]     0.3  0.269
[3,]     0.5  0.446
[4,]     0.6  0.538
[5,]     0.7  0.626
[6,]     0.9  0.782


julia> using GLM

julia> lineal = lm( :( optden ~ carb ), form )

Formula: optden ~ carb

Coefficients:

2x4 DataFrame:
          Estimate  Std.Error  t value   Pr(>|t|)
[1,]    0.00508571 0.00783368 0.649211   0.551595
[2,]      0.876286  0.0135345  64.7444 3.40919e-7
```

RDatasets.jl contiene datasets de diferentes paquetes de **R**.

GLM.jl contiene funciones para evaluar modelos lineales y utiliza el paquete Distributions.jl para los modelos lineales generalizados.

La lista es más larga

- Paquetes para graficar datos (Winston.jl, Gadfly.jl, etc.)
- Clustering.jl
- Otros paquetes estadísticos (Stats.jl, MCMC.jl, RobustStats.jl, MixedModels.jl, etc.)
- Modulos para finanza cuantitaiva (https://github.com/JuliaQuant)
- Linkear Julia con R (Rif.jl) o R con Julia (jl4R)

¡¡¡Muchas Gracias!!!