Introduction to Julia

Taylor Pospisil

2015-04-23

Motivation: The Two Language Problem

R/Python is too slow

- How do you get fast R code? Use Rcpp/Cython
- But it's easy to work with and you can work in the REPL
- Lots of libraries means someone has already done the work for you

C is too slow

- As fast as you're going to get.
- But don't forget about the programmer's time. It's not easy, and bugs are not easy to find.

Julia: A happy medium

- Best of both worlds: REPL with close-to-C speed

Concrete Example: Dynamic Programming

In the game of Risk, we want to calculate the probability of victory $p(A, D)$ attacking with A armies when the defender has D armies. Thus, we can use the following recursion

$$p(A, D) = \sum_{l=0}^{A} t(l, \min(A, 3), \min(D, 2)) \times p(A - l, D - (n - l))$$

Development

- Still a work-in-progress (current release = v0.37)
- Entire project on github.com/JuliaLang
- Package system (547 total) :
  - Gadfly (ggplot-like graphics)
  - PyCall, RCall
  - DataFrames
- Documentation available

Quick Technical Overview

JIT compilation

Code is compiled upon first execution

Type System

- The usual types included (ints, floats, arrays, dicts, sets, etc)
- User-defined types have the same performance as "built-ins"

Multiple Dispatch

Function calls are dispatched based upon the types of the arguments

Optimizing for Performance

Vectorization

Unlike with R, vectorization is not your first step

Type Stability

Very important, as type inference allows compilation of efficient code

The Usual

- Avoid memory allocation in tight loops
- Use the profiler
- Avoid pre-mature optimization and nothing makes bad algorithms fast

When is Julia > Matlab

Julia > Matlab

- Julia > R > Matlab
- Julia is open-source and free
- Julia has easy-to-read syntax
- Julia has good support for linear algebra

Matlab > Julia

- ??

Pick Julia when you don't have a license or don't rely upon existing Matlab code

When is Julia > Python

Julia > Python

- No python2/python3 issues
- No numpy issues; numerical computation isn't an add-on for Julia

Julia < Python

- Python has many prexisting libraries (beautifulsoup, sklearn, nltk)
- Python has a much bigger user/dev base

Pick Julia when you have do something original and numerical

When is Julia > R

Julia > R

- Julia is better suited for general purpose tasks
- Julia is faster for iteration (MCMC)

R > Julia

- R has existing packages
- R has better graphics (for now)
- Everyone knows R

Julia > R when you want to do something original and computationally intensive (but without needing C)

Takeaway

- Julia shows promise from a performance standpoint
- Julia is open-source and free
- Packages aren't there (yet?), but might use RCall or PyCall
- If you need C-like performance but R-like ease of development, give Julia a try!