Intro to the Julia programming language

Brendan O’Connor
CMU, Dec 2013

They have very good docs at: http://julialang.org/

I’m borrowing some slides from: http://julialang.org/blog/2013/03/julia-tutorial-MIT/
Julia

- A relatively new, open-source numeric programming language that’s both convenient and fast
- Version 0.2. Still in flux, especially libraries. But the basics are very usable.
- Lots of development momentum
Why Julia?

Dynamic languages are extremely popular for numerical work:

- Matlab, R, NumPy/SciPy, Mathematica, etc.
- very simple to learn and easy to do research in

However, all have a “split language” approach:

- high-level dynamic language for scripting low-level operations
- C/C++/Fortran for implementing fast low-level operations

Libraries in C — no productivity boost for library writers

Forces vectorization — sometimes a scalar loop is just better
“Gang of Forty”

Matlab Maple Mathematica SciPy SciLab IDL R
Octave S-PLUS SAS J APL Maxima Mathcad
Axiom Sage Lush Ch LabView O-Matrix PV-WAVE
Igor Pro OriginLab FreeMat Yorick GAUSS MuPad
Genius SciRuby Ox Stata JLab Magma Euler Rlab
Speakeasy GDL Nickle gretl ana Torch7
### Numeric programming environments

<table>
<thead>
<tr>
<th></th>
<th>Dynamic and math-y?</th>
<th>Fast?</th>
</tr>
</thead>
<tbody>
<tr>
<td>C/C++/Fortran/Java</td>
<td>−</td>
<td>+</td>
</tr>
<tr>
<td>Matlab</td>
<td>+</td>
<td>−</td>
</tr>
<tr>
<td>Num/SciPy</td>
<td>+</td>
<td>−</td>
</tr>
<tr>
<td>R</td>
<td>+</td>
<td>−</td>
</tr>
</tbody>
</table>


Tuesday, December 17, 13

- Dynamic vs Fast: the usual tradeoff
- PL quality: more subjective. can you define more than 1 function in a file? do you have a module system? do operators and functions work in a consistent way?
- Julia aims to have all of them
- Ecosystem
- To the extent there’s still a CS/Stats divide (or engineering/stats divide), you see it in R versus Matlab.
## Numeric programming environments

<table>
<thead>
<tr>
<th></th>
<th>Dynamic and math-y?</th>
<th>Fast?</th>
</tr>
</thead>
<tbody>
<tr>
<td>C/C++/Fortran/Java</td>
<td>−</td>
<td>+</td>
</tr>
<tr>
<td>Matlab</td>
<td>+</td>
<td>−</td>
</tr>
<tr>
<td>Num/SciPy</td>
<td>+</td>
<td>−</td>
</tr>
<tr>
<td>R</td>
<td>+</td>
<td>−</td>
</tr>
<tr>
<td>Julia</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>

Matlab-style syntax, REPL

Close to C speeds


Tuesday, December 17, 13

- Dynamic vs Fast: the usual tradeoff
- PL quality: more subjective. can you define more than 1 function in a file? do you have a module system? do operators and functions work in a consistent way?
- Julia aims to have all of them
- Ecosystem
- To the extent there’s still a CS/Stats divide (or engineering/stats divide), you see it in R versus Matlab.
### Numeric programming environments

<table>
<thead>
<tr>
<th>Core properties</th>
<th>Dynamic and math-y?</th>
<th>Fast?</th>
<th>PL quality?</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>C/C++/Fortran/Java</strong></td>
<td>−</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td><strong>Matlab</strong></td>
<td>+</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td><strong>Num/SciPy</strong></td>
<td>+</td>
<td>−</td>
<td>+</td>
</tr>
<tr>
<td><strong>R</strong></td>
<td>+</td>
<td>−</td>
<td>−</td>
</tr>
<tr>
<td><strong>Julia</strong></td>
<td>+</td>
<td>+</td>
<td>++</td>
</tr>
</tbody>
</table>

- **Matlab-style syntax, REPL**
- **Close to C speeds**
- **Optional static types**
- **Multiple dispatch**
- **Lisp-style macros**


Tuesday, December 17, 13

- Dynamic vs Fast: the usual tradeoff
- PL quality: more subjective. Can you define more than 1 function in a file? Do you have a module system? Do operators and functions work in a consistent way?
- Julia aims to have all of them
- Ecosystem
- To the extent there’s still a CS/Stats divide (or engineering/stats divide), you see it in R versus Matlab.
### Numeric programming environments

<table>
<thead>
<tr>
<th>Core properties</th>
<th>Ecosystem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic</td>
<td>Open-source</td>
</tr>
<tr>
<td>C/C++/Fortran/Java</td>
<td>-</td>
</tr>
<tr>
<td>Matlab</td>
<td>+</td>
</tr>
<tr>
<td>Num/SciPy</td>
<td>+</td>
</tr>
<tr>
<td>R</td>
<td>+</td>
</tr>
<tr>
<td>Julia</td>
<td>+</td>
</tr>
</tbody>
</table>

**Matlab-style syntax, REPL**

**Close to C speeds**

**Optional static types**

**Multiple dispatch**

**Lisp-style macros**

---


---

Tuesday, December 17, 13

- Dynamic vs Fast: the usual tradeoff
- PL quality: more subjective. can you define more than 1 function in a file? do you have a module system? do operators and functions work in a consistent way?
- Julia aims to have all of them
- Ecosystem
- To the extent there’s still a CS/Stats divide (or engineering/stats divide), you see it in R versus Matlab.
### Big 3 usual numeric languages

Languages that Google has spent a zillion dollars to make fast.

<table>
<thead>
<tr>
<th></th>
<th>Fortran</th>
<th>Julia</th>
<th>Python</th>
<th>R</th>
<th>Matlab R2012a</th>
<th>Octave 3.6.4</th>
<th>Mathematica 8.0</th>
<th>JavaScript V8 3.7.12.22</th>
<th>Go</th>
</tr>
</thead>
<tbody>
<tr>
<td>fib</td>
<td>0.26</td>
<td>0.91</td>
<td>30.37</td>
<td>411.36</td>
<td>1992.00</td>
<td>3211.81</td>
<td>64.46</td>
<td>2.18</td>
<td>1.03</td>
</tr>
<tr>
<td>parse_int</td>
<td>5.03</td>
<td>1.60</td>
<td>13.95</td>
<td>59.40</td>
<td>1463.16</td>
<td>7109.85</td>
<td>29.54</td>
<td>2.43</td>
<td>4.79</td>
</tr>
<tr>
<td>quicksort</td>
<td>1.11</td>
<td>1.14</td>
<td>31.98</td>
<td>524.29</td>
<td>101.84</td>
<td>1132.04</td>
<td>35.74</td>
<td>3.51</td>
<td>1.25</td>
</tr>
<tr>
<td>mandel</td>
<td>0.86</td>
<td>0.85</td>
<td>14.19</td>
<td>106.97</td>
<td>64.58</td>
<td>316.95</td>
<td>6.07</td>
<td>3.49</td>
<td>2.36</td>
</tr>
<tr>
<td>pi_sum</td>
<td>0.80</td>
<td>1.00</td>
<td>16.33</td>
<td>15.42</td>
<td>1.29</td>
<td>237.41</td>
<td>1.32</td>
<td>0.84</td>
<td>1.41</td>
</tr>
<tr>
<td>rand_mat_stat</td>
<td>0.64</td>
<td>1.66</td>
<td>13.52</td>
<td>10.84</td>
<td>6.61</td>
<td>14.98</td>
<td>4.52</td>
<td>3.28</td>
<td>8.12</td>
</tr>
<tr>
<td>rand_mat_mul</td>
<td>0.96</td>
<td>1.01</td>
<td>3.41</td>
<td>3.98</td>
<td>1.10</td>
<td>3.41</td>
<td>1.16</td>
<td>14.60</td>
<td>8.51</td>
</tr>
</tbody>
</table>

**Figure:** benchmark times relative to C (smaller is better, C performance = 1.0).
Why is it fast?

• Language design and smart use of LLVM
• [notebook]
• Don’t have to vectorize everything!
  • Matlab/R/NumPy have taught us wrong
    • And it’s a bad paradigm for structured cases, e.g. in NLP
  • e.g. Wasteful temporary allocations
    a+b+c+d
Other stuff

- Multiple dispatch
- Parallelism
- Metaprogramming (homoiconic, macros...)
- Calling C
Community

- Many developers, active mailing lists & responsive github issues

- Package system (200+ currently)
R-style data analysis

• Plotting
  • Gadfly (ggplot grammar of graphics-style): http://dcjones.github.io/Gadfly.jl/
  • PyPlot: interface to Python’s matplotlib

• DataFrames http://juliastats.github.io/DataFrames.jl/
  • Split-combine-apply, missing values, etc.
Statistics - a few libraries

- In-progress overview: https://github.com/JuliaStats/Roadmap.jl/issues/1
- **Distributions**: sampling, moments, MLE, conjugate updates
- **GLM**: linear mixed-effects regressions models
- **MCMC**
## Optimization juliaopt.org

- **JuMP** - An algebraic modeling language for optimization problems
- **Optim.jl** - Implementations of standard algorithms in pure Julia
- Interfaces to external solvers

<table>
<thead>
<tr>
<th>Solver</th>
<th>Julia</th>
<th>JuMP</th>
<th>LP</th>
<th>MILP</th>
<th>QCQP</th>
<th>MIQCQP</th>
<th>SDP</th>
<th>NLP</th>
<th>MINLP</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>COIN Cbc/Clp</td>
<td>Cbc.jl/Clp.jl</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GNU GLPK</td>
<td>GLPK.jl</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>IP Callbacks</td>
</tr>
<tr>
<td>Gurobi</td>
<td>Gurobi.jl</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>IP Callbacks</td>
</tr>
<tr>
<td>Ipopt</td>
<td>Ipopt.jl</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NLopt</td>
<td>NLopt.jl</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
JuMP library

using JuMP

m = Model()
@defVar(m, 0 <= x <= 2 )
@defVar(m, 0 <= y <= 30 )

@setObjective(m, Max, 5x + 3*y )
@addConstraint(m, 1x + 5y <= 3.0 )
solve(m)

• Calls out to external solvers
• Macros and metaprogramming make it easier to
develop specialized languages
MCMC library

- Metaprogramming gives expression parsing, supports autodiff for Hamiltonian MC

```plaintext
ex = quote
  vars ~ Normal(0, 1.0)
  prob = 1 / (1. + exp(- X * vars))
  Y ~ Bernoulli(prob)
end

m = model(ex, vars=zeros(nbeta), gradient=true)

# run random walk metropolis
mcchain01 = run(m * RWM(0.05) * SerialMC(1000:10000))

# run Hamiltonian Monte-Carlo
mcchain02 = run(m * HMC(2, 0.1) * SerialMC(1000:10000))
```

---

Tuesday, December 17, 13

this is L2-regularized logreg demo of quoting...

:(x + y * z)
dump:(x + y * z)
<table>
<thead>
<tr>
<th></th>
<th>Core properties</th>
<th>Ecosystem</th>
</tr>
</thead>
<tbody>
<tr>
<td>C/C++/Fortran/Java</td>
<td>−</td>
<td>+</td>
</tr>
<tr>
<td>Matlab</td>
<td>+</td>
<td>−</td>
</tr>
<tr>
<td>Num/SciPy</td>
<td>+</td>
<td>−</td>
</tr>
<tr>
<td>R</td>
<td>+</td>
<td>−</td>
</tr>
<tr>
<td>Julia</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>

The core language and standard library work very well right now.

The greater ecosystem of libraries is not yet mature, but advancing (frighteningly) rapidly. Comparability to R will surely take years.

Tuesday, December 17, 13

- Dynamic vs Fast: the usual tradeoff
- PL quality: more subjective. Can you define more than 1 function in a file? Do you have a module system? Do operators and functions work in a consistent way?
- Julia aims to have all of them
- Ecosystem
- To the extent there’s still a CS/Stats divide (or engineering/stats divide), you see it in R versus Matlab.
Links

• http://julialang.org/

• http://julialang.org/blog/2013/03/julia-tutorial-MIT/

• http://datacommunitydc.org/blog/2013/07/a-julia-meta-tutorial/

• Some stuff here I literally found last night from mailing list discussions
  https://groups.google.com/forum/#!forum/julia-stats
  https://groups.google.com/forum/#!forum/julia-users
  https://groups.google.com/forum/#!forum/julia-dev