An Introduction to the Julia Programming Language

Tyler Ransom

Duke University, Social Science Research Institute

December 30, 2016
1. What is Julia?

2. Syntax

3. Coding Tips

4. Data Analysis

5. Optimization

6. Examples
What is Julia?
What is Julia?

Overview

- An attractive computing alternative to Matlab
- Hybrid of scripted languages (like Matlab, Python, and R) and compiled languages (C++, FORTRAN).
- Offers a lot of the speed gains associated with compiled languages without having to deal with the lower-level programming components that make the compiled languages painful.
- How this is possible: Julia scripts are compiled “Just In Time” (JIT)
What is Julia?

Syntax   Coding Tips   Data Analysis   Optimization   Examples

Selling points

- **Speed**
  - Two macro economists (Aruoba and Fernández-Villaverde, 2014) ran speed tests on 10 different computing languages
  - Found that Julia was 10x faster than Matlab for their (simple) problem and only 2x slower than C++ and FORTRAN.

- **Ease of switching from Matlab**
  - Syntax is nearly identical

- **High quality optimization**
  - You can make use of quality optimizers (which out-of-the-box Matlab is lacking).
1. What is Julia?

2. Syntax

3. Coding Tips

4. Data Analysis

5. Optimization

6. Examples
Syntax differences vs. Matlab

- Use “[]” to index matrices (so $X(:,1)$ in Matlab should be $X[:,1]$)
- `ones(N)` produces an $N \times 1$ vector in Julia as opposed to an $N \times N$ matrix
- Element-wise operators need to explicitly have “.” in them (e.g. $X.==1$)
- Julia by default does not print output to the screen, so no semicolons are required
  - User needs to explicitly show output by using `println()` which is Julia’s version of `disp()`
In Matlab, functions are created as “function [out1,...,outK] = fname(in1,...,inL)”

In Julia, put function outputs at the end of the file with a return call:

- “function fname(in1,...,inL)
  
  return out1,...,outK
  end”

  (Always explicitly end functions)

- When calling the function in Julia, use “out1,...,outK = fname(in1,...,inL)” instead of “[out1,...,outK] = fname(in1,...,inL)” as in Matlab
Outline

1. What is Julia?

2. Syntax

3. Coding Tips

4. Data Analysis

5. Optimization

6. Examples
General coding tips for Julia

- Put everything in a function
  - If you are used to scripting, then convert scripts to functions with no inputs [example later]
  - Reason: variables defined outside of a function are assumed to be global in scope and require more resources from the software

- Explicitly specify types
  - Because Julia does JIT compiling, what “type” an object is matters a lot (e.g. integer vs. floating point number).
  - e.g. 1e5 vs 100000
  - Scripting languages do this conversion automatically for the user at the cost of efficiency
Vectors in Julia

- Unlike Matlab, a vector in Julia is not the same as a 1-column matrix
- e.g. `rand(N, 1)` creates a 50x1 (2-dimensional) array, whereas `rand(N)` creates a 50-element vector.
- This matters for built-in functions that expect a vector
- Column vectors are created with commas (e.g. `z=[1, 5, 6, 8, 9]`)
- Row vectors created with spaces (e.g. `z=[1 5 6 8 9]`)
- Note: In Matlab, both commas and spaces create row vectors, while semicolons create column vectors
<table>
<thead>
<tr>
<th>Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. What is Julia?</td>
</tr>
<tr>
<td>2. Syntax</td>
</tr>
<tr>
<td>3. Coding Tips</td>
</tr>
<tr>
<td>4. Data Analysis</td>
</tr>
<tr>
<td>5. Optimization</td>
</tr>
<tr>
<td>6. Examples</td>
</tr>
</tbody>
</table>
Data frames

- Just like in R and Python, Julia has data frames, which allow for heterogeneous-typed data to coexist in the same object
- Similar to Stata in that missing values are handled appropriately
- Trivia: Matlab also has a similar Dataset class, but I’ve never used it or seen it used
Many out-of-the-box estimation routines exist for Data frames in Julia (just as with R and Python).

All are lagging behind Stata in terms of user-friendliness.
1. What is Julia?
2. Syntax
3. Coding Tips
4. Data Analysis
5. Optimization
6. Examples
Julia for Mathematical Programming (JuMP)

- JuMP is a modeling language for Julia that takes advantage of Julia’s strengths
  - Julia has *syntactic macros* which allow code to produce code on its own (*metaprogramming*)
- Features of JuMP:
  - interfaces seamlessly with many industry-grade solvers
  - can be used to solve linear programming, nonlinear programming, and many other types of problems (including constrained optimization)
  - automatically differentiates the objective function (*not* numerical differentiation), resulting in speed gains
  - user-friendly model construction: user simply writes the objective function and any constraints in mathematical notation; JuMP then translates this into binaries at compile time
JuMP example (classical normal MLE)

Likelihood function:

$$\max_{\beta, \sigma} \frac{N}{2} \ln \left( \frac{1}{\sqrt{2\pi}\sigma^2} \right) - \frac{1}{2\sigma^2} \sum_{i=1}^{N} \left( y_i - \sum_{k=1}^{K} x_{ik}\beta_k \right)^2$$

JuMP implementation:

```julia
modelname = Model(solver=IpoptSolver(tol=1e-8))
@defVar(modelname, b[i=1:K], start = bAns[i])
@defVar(modelname, s >=0.0, start = sigAns)
@setNLObjective(modelname, Max, (N/2)*log(1/(2*pi*s^2))-
sum{(Y[i]-sum{X[i,k]*b[k],k=1:K})^2, i=1:N}/(2s^2))
solve(modelname)
```

That’s it!
Constrained MLE with JuMP

Add the constraint(s) after defining variables, but before defining the objective function.

```julia
modelname = Model(solver=IpoptSolver(tol=1e-8))
@defVar(modelname, b[i=1:K], start = bAns[i])
@defVar(modelname, s >=0.0, start = sigAns)
@addConstraint(modelname, b[15] == 0)
@setNLOobjective(modelname, Max, (N/2)*log(1/(2*pi*s^2)) - sum{(Y[i]-sum{X[i,k]*b[k],k=1:K})^2, i=1:N}/(2s^2))
solve(modelname)
```
Can do maximization or minimization just by indicating “min” or “max” in the objective function definition

Can do extremely large scale optimization problems (Lubin and Dunning, 2013)

Can return gradient and hessian (though not as seamlessly as fminunc)

Can easily implement Mathematical Programming with Equilibrium Constraints (MPEC, see Su and Judd, 2012)
  - This method is orders of magnitude more efficient than nested fixed-point (NFXP)
  - the only equilibrium that needs to be solved exactly is the one associated with the final estimate of structural parameters
Because JuMP does constrained optimization, the hessian it returns is the hessian of the *Lagrangian*, not the hessian of the *objective*.

In this sense, JuMP most closely corresponds to `fmincon` in Matlab, as opposed to `fminunc`.

Additionally, no matrix multiplication in JuMP at the moment, requiring additional summation operators in objective function.
Outline

1. What is Julia?
2. Syntax
3. Coding Tips
4. Data Analysis
5. Optimization
6. Examples
Example script

An example script to generate data and estimate parameters of a likelihood function (called “JuMPestimation.jl”)

```julia
# declare packages that you will use (similar to LaTeX preamble)
using JuMP, Ipopt
# compile functions you will be using later
include("datagen.jl")
include("jumpMLE.jl")
# evaluate the functions referenced above in the -include- statements.
# -@time- is equivalent to tic/toc in Matlab
@time X,Y,bAns,sigAns,n = datagen()
@time bOpt,sOpt = jumpMLE(Y,X,[bAns,sigAns])
```
function datagen()
## Generate data for a linear model to test optimization
srand(1234)
N = convert(Int64,1e4)
T = 5
n = convert(Int64,N*T)

# generate the Xs
X = [ones(N*T,1) 5+3*randn(N*T,1) rand(N*T,1)
    2.5+2*randn(N*T,1) 15+3*randn(N*T,1) .7-.1*randn(N*T,1)
    5+3*randn(N*T,1) rand(N*T,1) 2.5+2*randn(N*T,1)
    15+3*randn(N*T,1) .7-.1*randn(N*T,1) 5+3*randn(N*T,1)
    rand(N*T,1) 2.5+2*randn(N*T,1) 15+3*randn(N*T,1)
    .7-.1*randn(N*T,1)]
# generate the betas (X coefficients)
bAns = [ 2.15, 0.10, 0.50, 0.10, 0.75, 1.2, 0.10, 0.50, 0.10, 0.75, 1.2, 0.10, 0.50, 0.10, 0.75, 1.2 ]

# generate the std dev of the errors
sigAns = 0.3

# generate the Ys from the Xs, betas, and error draws
draw = 0 + sigAns*randn(N*T,1)
Y = X*bAns+draw

# return generated data so that other functions (below) have access
terun X,Y,bAns,sigAns,n
end
function jumpMLE(Y,X,startval)
  myMLE = Model(solver=IpoptSolver(tol=1e-8))
  @defVar(myMLE, b[i=1:size(X,2)], start = startval[i])
  @defVar(myMLE, s >=0.0, start = startval[end])
  # Write your objective function
  @setNLObjective(myMLE, Max, (n/2)*log(1/(2*pi*s^2)) - 
                   sum{(Y[i]-sum{X[i,k]*b[k], k=1:size(X,2)})^2, i=1:size(X,1)}/(2s^2))
  # Solve the objective function
  status = solve(myMLE)
  # Save estimates
  bOpt = getValue(b[:])
  sOpt = getValue(s)
  return bOpt,sOpt
end
To execute the script, just type the following at the command line:

```
include("JuMPestimation.jl")
```
Should I switch?

- It’s hard to switch computing languages
- It’s even harder to convince co-authors to switch languages
- Julia Pros
  - Speed
  - Similarity to Matlab
  - High-quality optimization
  - Open source
- Julia Cons
  - Cost of switching
  - Still a very young language
  - Still under major development
- I believe that Julia is the language of the future and will soon have a sizable market share in economics
