IntroJulia

October 5, 2018

1 Introduction to Julia and JuliaCall

2 Structure of the talk

- What do we have before Julia? What is the situation in R/Python/MATLAB?
- Introduction to Julia
- Some Julia packages and examples
- More advanced topics in Julia
- Introduction to JuliaCall and other related packages

3 Solution 1: write code in C/Cpp

- R is slow, how about writing functions in C/Cpp and then call it from R?

3.1 Pros

- Faster functions.

3.2 Cons

- Need to learn a language that is (very) different from R, and is quite low level (C) or quite complicated (cpp).
- It doesn't compose well.

3.3 An example from R itself

R itself already uses lots of functions in C, but is still slow....

R: `sin(sqrt(a))` ## a is some vector

1. R: evaluate `sqrt(a)` -> call C function
2. C: create a new vector -> do the calculation -> give the result `b = sqrt(a)` back to R
3. R: evaluate `sin(b)` -> call C function
4. C: create a new vector -> do the calculation -> give the result `c = sin(b)` back to R.

This is a *sub*optimal.

Why not 1. R: evaluate `sqrt(a)` -> call C function 2. C: create a new vector -> do the calculation -> give the result `b = sin(sqrt(a))` back to R?

If we want to achive this, we need to write a C function called `sin_sqrt` and then use it from R. And maybe we also need to write `sqrt_sin`, `exp_sin`, `sin_exp`, `cos_sin`, `sin_cos` ....

4 Solution 2: Just In Time (JIT) Compiling

- Functions need to be executed will be compiled first.
- JIT is widely used, in Julia, in your browser (Google V8), in MATLAB, in Python (numba), and in R.

4.1 MATLAB and Python's JIT

4.1.1 Pros

- Fast when they work.
- Being optimized for much longer than Julia.

4.1.2 Cons

- It's limited.
- It doesn't compose well.

5 Julia's advantage

Instead of the case in R/Python/MATLAB, where you have the language first, and then consider the optimization (JIT), Julia's design is to make JIT easier.

- Not limited: users can define their own types, and Julia will try to do JIT for users.
- Compose well: if Julia find further optimization opportunities, it will do it automatically.

5.1 An example

Julia: `sin.(sqrt.(a))` ## a is some vector

1. create a new vector `b`
2. for each element `x` in `a` do the calculation of `sin(sqrt(x))`
3. stores the result in `b`

6 Julia

6.1 Official website

https://julialang.org/

6.2 IDEs

- Juno in Atom
- Visual Studio Code
- Jupyter
- Julia Box

6.3 Version choice

- Convex optimization: Convex.jl
- Optimization: Optim.jl, JuMP.jl
- Differential equations: DifferentialEquations.jl
- Automatic differentiation: ForwardDiff.jl, ReverseDiff.jl
- Mixed Effects Models: MixedModels.jl
- Artificial Intelligence: Flux.jl, TensorFlow.jl

In [1]:
```
## using Pkg; Pkg.add("JuMP"); Pkg.add("Ipopt")
using JuMP
using Ipopt

p = 3
m = Model(solver = IpoptSolver())
@variable(m, x[1:p])
@objective(m, Min, sum(x.*x))
status = solve(m)
getvalue(x[1:p])
```

```
******************************************************************************
This program contains Ipopt, a library for large-scale nonlinear optimization.
 Ipopt is released as open source code under the Eclipse Public License (EPL).
         For more information visit http://projects.coin-or.org/Ipopt
******************************************************************************

This is Ipopt version 3.12.8, running with linear solver mumps.
NOTE: Other linear solvers might be more efficient (see Ipopt documentation).

Number of nonzeros in equality constraint Jacobian...:        0
Number of nonzeros in inequality constraint Jacobian.:        0
Number of nonzeros in Lagrangian Hessian.............:        3

Total number of variables............................:        3
                     variables with only lower bounds:        0
```

```
                variables with lower and upper bounds:        0
                     variables with only upper bounds:        0
Total number of equality constraints...:        0
Total number of inequality constraints...:        0
        inequality constraints with only lower bounds:        0
   inequality constraints with lower and upper bounds:        0
        inequality constraints with only upper bounds:        0

iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
   0  0.0000000e+00 0.00e+00 0.00e+00  -1.0 0.00e+00    -  0.00e+00 0.00e+00   0

Number of Iterations....: 0

                                   (scaled)                 (unscaled)
Objective...:   0.0000000000000000e+00    0.0000000000000000e+00
Dual infeasibility...:   0.0000000000000000e+00    0.0000000000000000e+00
Constraint violation...:   0.0000000000000000e+00    0.0000000000000000e+00
Complementarity...:   0.0000000000000000e+00    0.0000000000000000e+00
Overall NLP error...:   0.0000000000000000e+00    0.0000000000000000e+00


Number of objective function evaluations             = 1
Number of objective gradient evaluations             = 1
Number of equality constraint evaluations            = 0
Number of inequality constraint evaluations          = 0
Number of equality constraint Jacobian evaluations   = 0
Number of inequality constraint Jacobian evaluations = 0
Number of Lagrangian Hessian evaluations             = 0
Total CPU secs in IPOPT (w/o function evaluations)   =      0.158
Total CPU secs in NLP function evaluations           =      0.045

EXIT: Optimal Solution Found.
```

Out[1]:

```
3-element Array{Float64,1}:
 0.0
 0.0
 0.0
```

In [2]:

```
## using Pkg; Pkg.add("ForwardDiff")
using ForwardDiff

function NewtMin(f, x0, eps)
    fgrad = x-> ForwardDiff.gradient(f, x)
    fhess = x-> ForwardDiff.hessian(f, x)
    oldval = f(x0)
    newx = x0 - fhess(x0)\fgrad(x0)
    newval = f(newx)
```

```
    while abs(newval - oldval) > eps
        oldval = newval
        newx = newx - fhess(newx)\fgrad(newx)
        newval = f(newx)
    end
    return newx
end

f(x) = sum(x.^2)

NewtMin(f, [1., 1., 1.], 1e-6)
```

Out[2]:
```
3-element Array{Float64,1}:
 0.0
 0.0
 0.0
```

In [3]:
```
## using Pkg; Pkg.add("Optim")
using Optim

r = Optim.optimize(f, [1., 1., 1.])
Optim.minimizer(r)
```

Out[3]:
```
3-element Array{Float64,1}:
 -5.0823878310249317e-5
  9.262030970117879e-6
 -1.2828696727829803e-5
```

7 Advanced: multiple dispatch

- Similar to S3 system in R: `print.lm`, `anova.lm`, etc, but works on multiple arguments, and is **fast**!
- A simple example:

  julia> function f(x::Integer) x+1 end

  ## f (generic function with 1 method)

  julia> function f(x::AbstractFloat) 3 * x end

  ## f (generic function with 2 methods)

8 More advanced

- `@code_llvm`
- `@code_warn_type`
- `@trace` from Traceur.jl

8.1 Some more advanced and comprehensive introductions

- https://github.com/johnfgibson/whyjulia
- http://ucidatascienceinitiative.github.io/IntroToJulia/Html/WhyJulia

9 JuliaCall

9.1 Website

- https://github.com/Non-Contradiction/JuliaCall
- https://cran.r-project.org/web/packages/JuliaCall/index.html
- If you find JuliaCall useful, please consider giving it a STAR on Github ^_^

10 Questions?

11 Thank you