Extensible Software for Research

principles and an example in julia

Center for Lifespan Psychology, Max Planck Institute for Human Development

Maximilian Ernst

Contents

- Why should you care?
- How do you get there?

Research Software Personas

- research software engineers
- statistics researcher
- applied user

A day in the live of …

a statistics researcher

- work with a specific type of model
  - linear regression, deep learning, …
- have an idea
- test it
- make it available to applied researchers

Now we need software

- to test $\rightarrow$ prototype
- to make it available $\rightarrow$ deploy

What's the fastest way to get there?

existing software

It would be nice if they could extend existing software

But ...

- **understand** 1000s of lines of code
- make **changes**, possibly breaking stuff
- get maintainers to **adopt** their changes

these hurdles are often too high!

A day in the live of …

- to test: barebone, minimal reimplementation
  - waste of time
  - not well tested
  - hard to reproduce

- to deploy: put their code on github
  - bad user interface, no documentation
  - missing features
  - incompatible to existing software

My Experience

- from R $\rightarrow$ julia
- most R packages are very hard to extend
- most julia packages are very easy to extend

Culture

- care about extensibility
- developer documentation
- assume their code is read

Software Design

You need to be able to add new features...

- without understanding existing code
- without changing existing code
- syntactical requirements need to be clear and easy communicable!

Two modes of extension

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam nec interdum est, et suscipit elit. Aenean imperdiet augue sed arcu iaculis mollis. Aenean felis augue, fringilla ac diam non, dapibus commodo tellus. Donec laoreet a magna id vestibulum. Suspendisse sapien turpis, dictum sed scelerisque ac, malesuada ac augue. Integer id mattis ipsum. Fusce nec dui eu tellus elementum efficitur. Aenean iaculis lorem sem. Aenean eu placerat augue. Cras eget fermentum augue. Nullam in orci ut erat aliquet lacinia. Vivamus rhoncus, mauris vel pulvinar dapibus, tellus ante vestibulum lorem, sed tristique erat orci quis orci. Integer at laoreet neque, id lobortis turpis.

Quisque ultricies ultrices massa ut rhoncus. Sed finibus neque purus, sed ullamcorper lectus ultricies eu. Integer malesuada sem eget feugiat tincidunt. Aenean laoreet vulputate metus non iaculis. Nullam nec viverra purus, a elementum est. Mauris consequat nunc ut urna aliquam, a sollicitudin dui tincidunt.

Mauris et urna non justo faucibus cursus eu sit amet orci. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Proin at purus nisl. Quisque nec mi eget tellus vestibulum imperdiet. Praesent convallis dui urna, eu volutpat mi porta id. Cras euismod metus quam, a sollicitudin magna aliquet a. Ut placerat nunc ut leo viverra, ac bibendum est vehicula. Maecenas non finibus velit. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vivamus sit amet dolor lacinia, dignissim libero in, laoreet neque. Maecenas aliquet velit rhoncus, iaculis sem eget, eleifend velit.

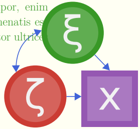
Vestibulum at sem felis. Suspendisse vitae euismod sapien, eget gravida massa. Nullam ac porttitor elit. Vestibulum in pharetra risus. Quisque condimentum porttitor massa, ut ornare turpis convallis sed. Nullam ut vestibulum sem. Aliquam eu risus. Sed commodo posuere ante. Nam vulputate sit amet mauris a aliquet. In tempor, enim ullamcorper auctor accumsan, odio felis sagittis erat, a venenatis est purus sed risus. Donec interdum dui at urna facilisis, porttitor ultrices lorem dapibus. Aenean scelerisque nisl at neque placerat.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam nec interdum est, et suscipit elit. Aenean imperdiet augue sed arcu iaculis mollis. Aenean felis augue, fringilla ac diam non, dapibus commodo tellus. Donec laoreet a magna id vestibulum. Suspendisse sapien turpis, dictum sed scelerisque ac, malesuada ac augue. Integer id mattis ipsum. Fusce nec dui eu tellus elementum efficitur. Aenean iaculis lorem sem. Aenean eu placerat augue. Cras eget fermentum augue. Nullam in orci ut erat aliquet lacinia. Vivamus rhoncus, mauris vel pulvinar dapibus, tellus ante vestibulum lorem, sed tristique erat orci quis orci. Integer at laoreet neque, id lobortis turpis.

Quisque ultricies ultrices massa ut rhoncus. Sed finibus neque purus, sed ullamcorper lectus ultricies eu. Integer malesuada sem eget feugiat tincidunt. Aenean laoreet vulputate metus non iaculis. Nullam nec viverra purus, a elementum est. Mauris consequat nunc ut urna aliquam, a sollicitudin dui tincidunt.

Mauris et urna non justo faucibus cursus eu sit amet orci. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Proin at purus nisl. Quisque nec mi eget tellus vestibulum imperdiet. Praesent convallis dui urna, eu volutpat mi porta id. Cras euismod metus quam, a sollicitudin magna aliquet a. Ut placerat nunc ut leo viverra, ac bibendum est vehicula. Maecenas non finibus velit. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Vivamus sit amet dolor lacinia, dignissim libero in, laoreet neque. Maecenas aliquet velit rhoncus, iaculis sem eget, eleifend velit.

Vestibulum at sem felis. Suspendisse vitae euismod sapien, eget gravida massa. Nullam ac porttitor elit. Vestibulum in pharetra risus. Quisque condimentum porttitor massa, ut ornare turpis convallis sed. Nullam ut vestibulum sem. Aliquam eu risus. Sed commodo posuere ante. Nam vulputate sit amet mauris a aliquet. In tempor, enim ullamcorper auctor accumsan, odio felis sagittis erat, a venenatis est purus sed risus. Donec interdum dui at urna facilisis, porttitor ultrices lorem dapibus. Aenean scelerisque nisl at neque placerat.



Why julia?

In Julia, everything has a type:

```
a = 1.0
typeof(a) # Float64

b = ”hello”
typeof(b) # String
```

Why julia?

You can define your own types:

```julia
struct ClockTime{T}
    time::T
end

my_time = ClockTime(5.0) # ClockTime{Float64}(5.0)
my_time.time # 5.0
```

Why julia?

A function is a collection of methods:

```
typeof(1.0) # Float64
typeof(1) # Int64

@code_llvm 6.0*7.0
@code_llvm 6*7

methods(*)
```

Why julia?

We can write our own addition:

```julia
import Base: +

function +(x::ClockTime{T}, y::ClockTime{T}) where{T}
    return ClockTime((x.time + y.time) % T(24))
end

my_time = ClockTime(11.2)
your_time = ClockTime(18.4)

our_time = my_time + your_time
```

Goal achieved

We have just written extensible code.

I have memory problems, and I only care about full hours.

```
my_time = ClockTime(UInt8(5))
your_time = ClockTime(UInt8(8))

our_time = my_time + your_time
```

Why julia?

I want to multiply sparse matrices of clock times

```julia
using SparseArrays

import Base: zero, *

function *(x::T, y::ClockTime{T}) where T
    return ClockTime((x * y.time) % T(24))
end

zero(x::ClockTime{T}) where T = ClockTime(zero(T))
zero(::Type{ClockTime{T}}) where T = ClockTime(zero(T))
```

Why julia?

Let’s define some matrices!

```
a = zeros(ClockTime{Float64}, 20, 20)

a[1,1] = ClockTime(5.0)
a[1,2] = ClockTime(11.673)
a[6,9] = ClockTime(17.23)
a[16,4] = ClockTime(20.87)

a_sparse = sparse(a)

b = zeros(20, 20)
b[3,9] = 1
b[6,9] = sqrt(2)
b[19,1] = π
b[4,5] = e

b_sparse = sparse(b)
```

Why julia?

```
using BenchmarkTools

@benchmark b_sparse*a_sparse

BenchmarkTools.Trial: 10000 samples with 199 evaluations.
 Range (min … max):  422.864 ns …  12.295 μs
 Time  (median):     453.877 ns
 Time  (mean ± σ):   566.627 ns ± 648.451 ns
Memory estimate: 2.22 KiB

@benchmark b*a

BenchmarkTools.Trial: 10000 samples with 1 evaluation.
 Range (min … max):  56.683 μs …  1.323 ms
 Time  (median):     57.111 μs
 Time  (mean ± σ):   59.411 μs ± 18.922 μs
 Memory estimate: 23.75 KiB
```

A few days in a methods researchers life

- Do you have any ideas why this does not converge?
- Staring puzzled at the theory (should work?!).
- Staring very puzzled at the implementation in C++.
- Rinse and repeat for a couple of days and researchers.

An hour in our life

- Look at the formula: $ridge(x, \lambda) = \lambda \sum_{j=1}^{p} x^2$
- Implement in Julia: `ridge(x, λ) = λ * sum(x.^2))`
- add 30 lines of API (formal requirements)
- Enjoy.

Two hours in our life

- Original simulation takes weeks on a dedicated workstation.
- Original simulation freezes our cluster due to poor parallelization.
- Simulation in Julia takes 2 hours on my laptop.

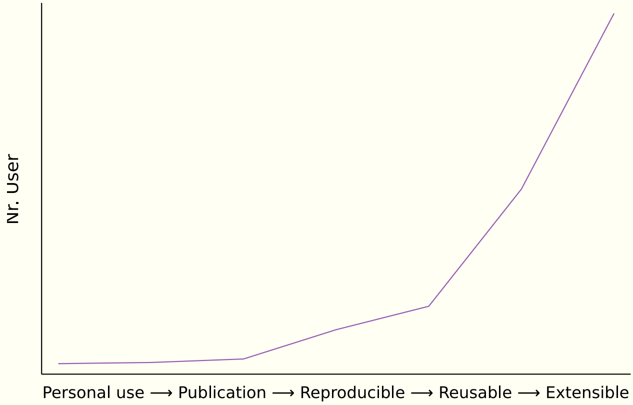
Why?

- Some investments in extensibility
- division of labor:
  - optimizing linear algebra is done by Intel
  - numerical optimization is done by dedicated experts
  - differentiation is automated
- modern infrastructure

But why?

convenience

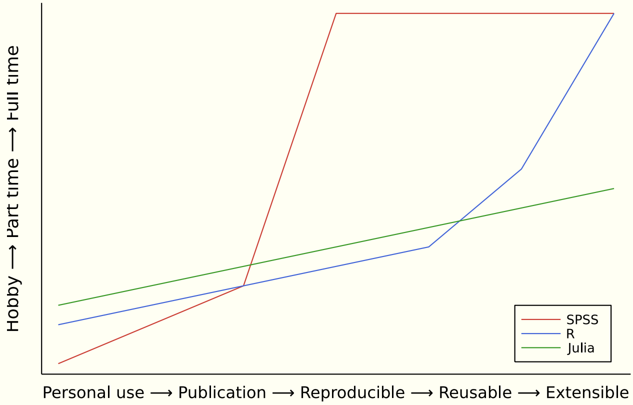
The leverage of convenience



How to improve convenience?

1. Extensible Software
2. Dokumentation
3. User Interface

Time is limited



Dokumentation

- Dokumentation for users
- Dokumentation for contributors/developers

User Interface

- Frictionless
- Connected to prior knowledge