Julia for research

Tomas Fiers
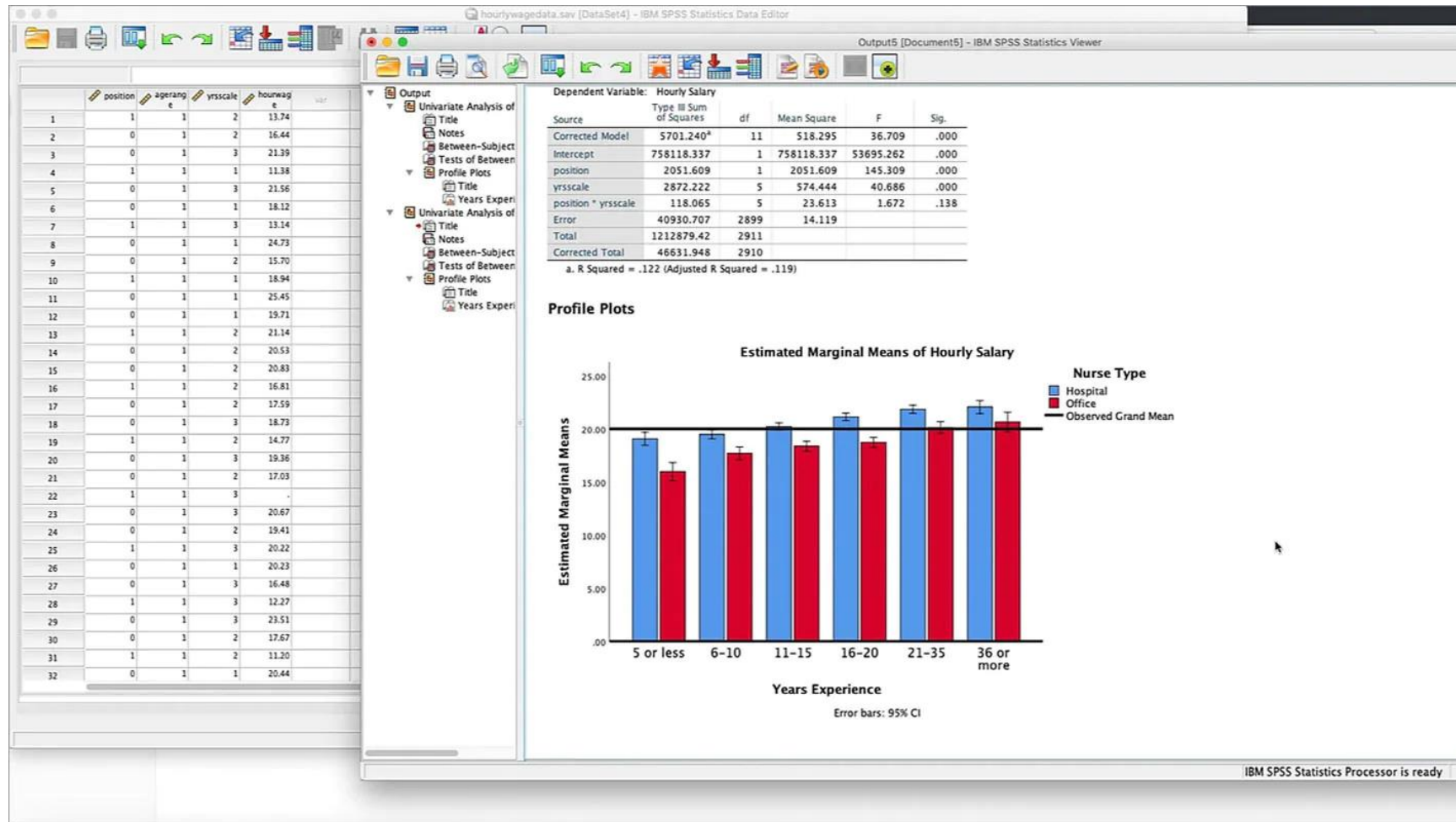

Lunchtime data club at the School of Psychology

Dec 7, 2022

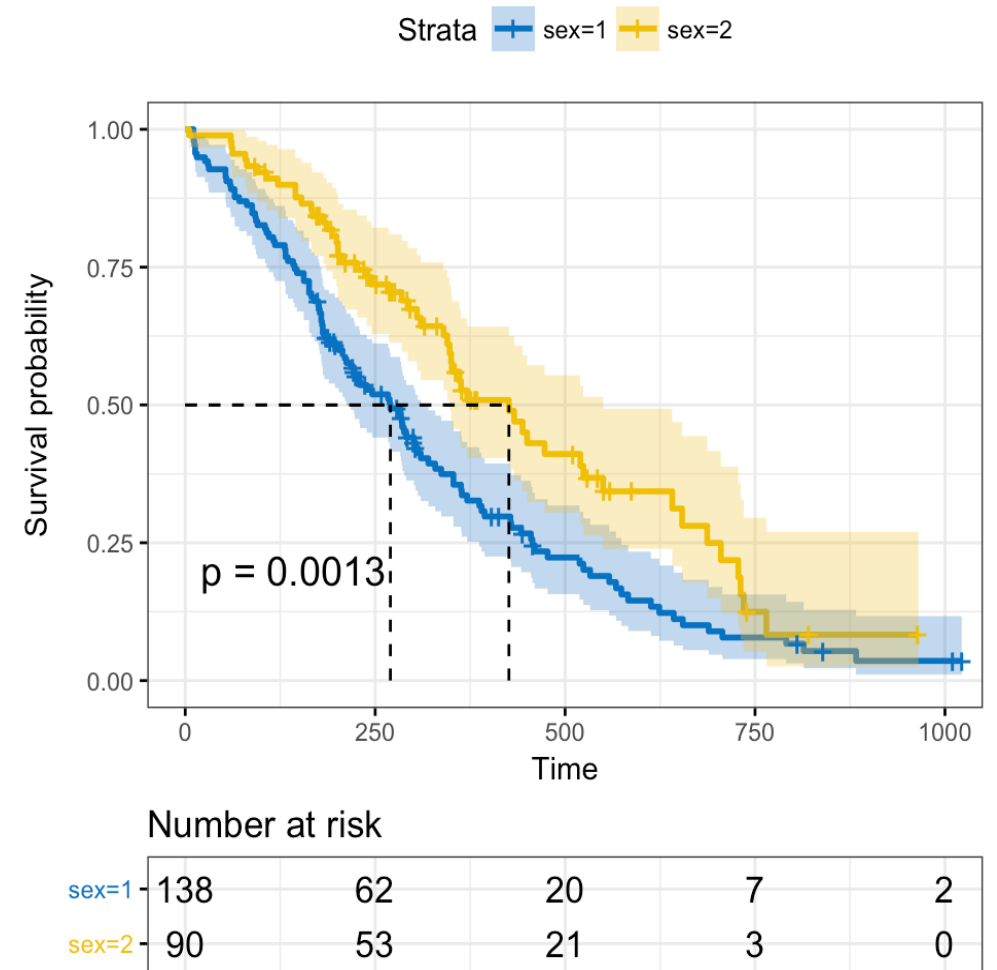If you're using Excel, SPSS, Stata, JASP, …

Why learn a programming language?

Why program?

- Automate analyses
  → less error-prone

- Customize
  - Special plots
  - Tweak analyses

- Run simulations!

- For fun

An example custom plot

Choosing a programming language
as a researcher

	First release	Free & open?	Online community
		(hackable, "own your code running environment")	(→ Learning resources & documentation polish)
• Main choices:			
R	1995 / 1976 (S)	Yes	Huge
Python	1991	Yes	Huge
• Others			
Julia	2012	Yes	Medium
Matlab	1979	No	Large

• ..is also choosing a community

Victor 🇺🇦
@vzverovich

Replying to @lefticus

Julia is Matlab without users

3:28 AM · Feb 21, 2022

Guillaume Dalle
@giomdal

Matlab is Julia without open source contributors 🤷

Victor 🇺🇦 @vzverovich · Feb 21
Replying to @lefticus
Julia is Matlab without users

8:59 AM · Mar 20, 2022

Julia syntax

```julia
# Simulate a simple neuron for input current `I`
# and return when the neuron fires its first spike
function first_spike(I, Δt, τ)
    N = length(I)
    v = 0
    for i in 1:N
        v += Δt * (I[i] - v) / τ    # Euler integration
        if v > 2                     # Spike!
            return time = i * Δt
        end
    end
    return nothing
end
```

In Python / R / Matlab:
"Avoid for-loops"
"Write vectorized code"

Compilation. Your code ⤳ the CPU

- If **one line** of Julia code corresponds
  to just a few CPU instructions

- ..then the same line in base Python / R / Matlab will often
  correspond to an order of magnitude **more instructions** *

  - ..That's why the code that does the 'real' numeric work in these
    languages is actually written in C / C++
    NumPy, PyTorch, Tensorflow, dplyr, …: all have their core written
    in a different language

  - ..That's why, to have your code run fast, you're discouraged from
    writing for-loops for numeric code ..

  - .. and instead use the provided library functions
    e.g. np.where(…)

  - Python is often used as "glue-code" (see next slide)

  - If you want a custom numeric algorithm that's not provided by
    the libraries, you need to learn C / C++
    The "two languages-problem"

* Matlab added JIT compilation in 2015
(but it's rather opaque)

Python can have JIT compilation via the fantastic
Numba package. (But you can only use base
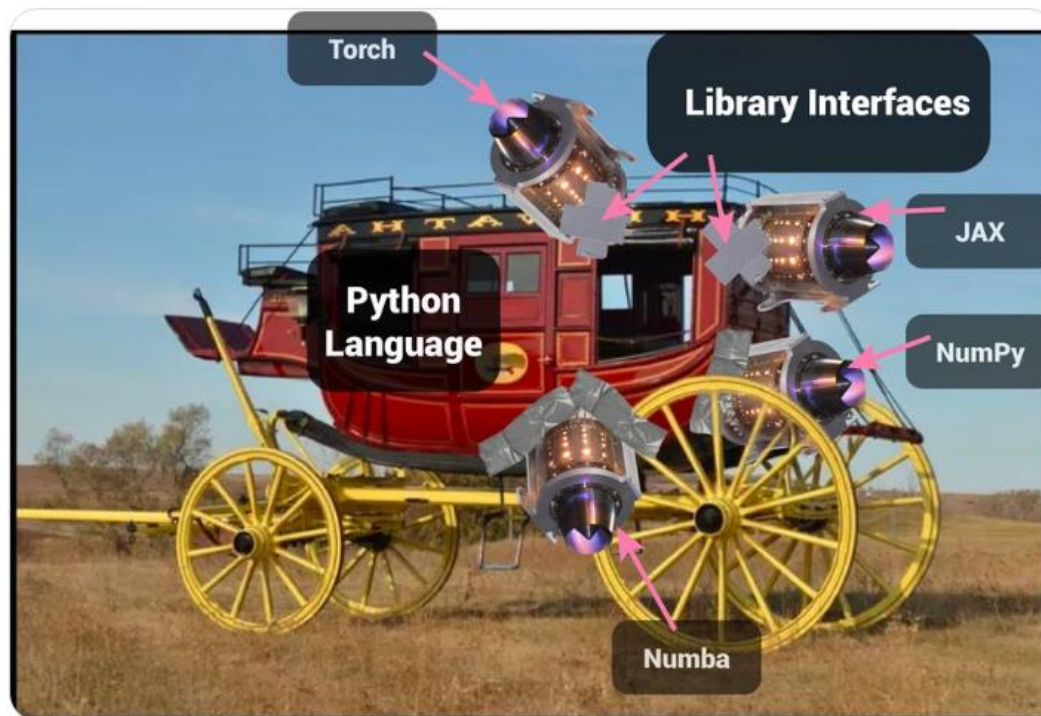Python with Numba, not arbitrary other packages).

JIT compilation

- If **one line** of Julia code corresponds to just a few CPU instructions
- ..then the same line in base Python / R / Matlab[*] will often correspond to an order of magnitude **more CPU instructions**

- Why is this?
    - The same line of code (say, $z = x + y$) does different things, based on the **type** of x and y
        - If they're integers (8 + 3), use the `leaq` CPU instruction
        - If one is a float (8 + 3.3), call `convert` and use the floating point processor unit
        - If they're both *plots*, call subroutines, to compose the plots together into a bigger figure
        - …
    - Python, R, and Matlab need to check the types of x and y *every time the line is run,* and then call the appropriate subroutines
        - Hence all these extra CPU instructions

    - Julia will *infer* the types of x and y
        - When? <u>The first time that the function which contains our code is called</u>
        - It does this (i.e. does 'type inference') based on the arguments that the function was called with (more specifically, their types), and by analyzing the function source code
    - I then compiles a fast version of the function This is just-in-time (JIT) compilation

Data analysis in Julia

- DataFrames.jl
  - Tidyverse's dplyr & Python's Pandas equivalent
    - Better API than Pandas, imho
  - In the very capable hands of Bogumił Kamiński
    - Check out his tutorials: github.com/bkamins/Julia-DataFrames-Tutorial

- \<missing\> datatype is built-in in Julia
  - distinct from \<nothing\>

- I plot using Python's matplotlib 😄
  - Via PyPlot.jl
  - There's also Makie.jl
  - ..and Gadfly.jl, which is ggplot-inspired

Julia likes

- Unicode variable names & operators

```
izh() = begin
    # Conductance-based synaptic current
    I_syn = g_e*(v-E_e) + g_i*(v-E_i)
    # Izhikevich 2D system
    Δ.v = (k*(v-v_l)*(v-v_t) - u - I_syn) / C   # Membrane potential
    Δ.u = a*(b*(v-v_r) - u)                      # Adaptation current
    # Synaptic conductance decay
    # (g_e is sum over all exc synapses)
    Δ.g_e = -g_e / τ
    Δ.g_i = -g_i / τ
end
has_spiked() = (v ≥ v_s)
on_self_spike() = begin
    v = v_r
    u += Δu
end
```

Julia likes

- Community
  - Discourse forum & Slack
  - Scientists
  - Contribute to ecosystem (open source, build upon others)
- As close-to-the-metal as you like
  - Look under the hood
  - Understand why something is slow / fast
- "structs and functions" design style
  - Versus when you're designing software in Python, you tend to use more OOP (inheritance)

- Inspectability
  - `@edit` to jump to source code of anything… amazing
  - `@code_native` to see cpu instructions
  - `?` for documentation
  - …
- Dependency management
  - Project.toml
  - Manifest.toml
- Macro's
  - Lisp-like. 'Code as data'

Julia annoyances

- **Package startup time 💀 ("time-to-first-plot")**
  - Language developers are working hard this year to improve this
- No winning plotting package yet
- `name.<tab>` completion (API discovery) not as good as Python
- Getting floats to print with lower precision is way more difficult than it should be for new users
- Traits / interfaces (lack of)
- Error handling is underdeveloped / under-practiced ("→ silent fails & crashes")
- See also:
  - yuri.is/not-julia
  - danluu.com/julialang
  - viralinstruction.com/posts/badjulia
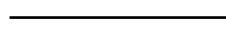
"Julia has a correctness problem"

- (i.e. there's nasty hidden bugs everywhere)
- Not true for Base Julia:
  - every line there is pored over by many language developers
  - automatic test coverage is very comprehensive
- For other people's packages:
  - Not a problem in my experience.
  - But you have to inspect the packages that you use, if they're not in Julia Base; and make a value judgement about their quality
  - A lot of Julia packages are of *very* high quality in my experience
    - Except for the lack of error checking (of inputs and outputs)
      - Julia doesn't hold your hand:
        you gotta know what you're doing mathematically / numerically / statistically

Why did I switch to Julia?

- 🎄 [Advent of Code](#) :) (2021)

- Physical units in neuron simulations:

- I could just keep using:
  - my Jupyter notebook workflow
  - my Matplotlib experience

```
parameters = (
    # Izhikevich neuron
    C   =   100      * pF
    k   =     0.7    * (nS/mV)
    vₗ  = -  60      * mV
    vₜ  = -  40      * mV
    a   =     0.03   / ms
    b   = -   2      * nS
    vₛ  =    35      * mV
    vᵣ  = -  50      * mV
    Δu  =   100      * pA
    # Synapses
    Eₑ  =     0 * mV
    Eᵢ  = -80 * mV
    τ   =     7 * ms
    # Inputs
    Nₑ  = 40
    Nᵢ  = 10
    N   = Nₑ + Nᵢ
    Δgₑ = 60nS / Nₑ
    Δgᵢ = 60nS / Nᵢ
    # Integration
    Δt  = 0.1ms
    T   = 10seconds
)
```

Julia tips

- Code must be type-inferable ("type-stable")
  - Put everything in (small) functions
  - If using globals: `const`, or typed

- Read the manual
  - Especially the "Performance tips" section, if you're wondering why your code is not as fast as promised. Also:
- **Ask questions on the forum**
  - discourse.julialang.org
  - People are very eager to help, and the community managers do a great job
- Use Revise.jl (Use all of Tim Holy's packages actually).
  - This minizes nr. of times you have to restart the Julia session (re: time-to-first-X problem)
  - Plus:
  - If using VS Code, there's a plugin for Julia. Also: the JuliaMono font :) ⟶
  - On Windows, use the Julia REPL in the Windows Terminal
  - Checkout startup.jl

- Don't load unnecessary packages
  - Julia *Base* has no real latency (time-to-first-X) problem. It's loading many packages that gets you
    - Especially packages that have many dependencies themselves (looking at you SciML ecosystem :P)
  - Do you really need this package? Can you just implement it yourself / copy the relevant part?

- Learn by doing
  - Like by doing some Advent of Code puzzles!

```julia
# Code excerpt from the
# JuliaMono homepage.
# Original by Zygmunt Szpak

⊗ = kron
N = length(𝒟[1])
ℳ, ℳ′ = 𝒟
Λ₁, Λ₂ = 𝒞
e₁ = @SMatrix [1.0; 0.0; 0.0]
e₂ = @SMatrix [0.0; 1.0; 0.0]
for n = 1:N
    index = SVector(1,2)
    Λₙ[1:2,1:2] .=  Λ₁[n][index,i
    Λₙ[3:4,3:4] .=  Λ₂[n][index,i
    m     = hom(ℳ[n])
    m′    = hom(ℳ′[n])
    Uₙ    = (m ⊗ m′)
    ∂ₓuₙ = [(e₁ ⊗ m′) (e₂ ⊗ m′)
    Bₙ    = ∂ₓuₙ * Λₙ * ∂ₓuₙ'
    Σₙ    = θ' * Bₙ * θ
    Σₙ⁻¹  = inv(Σₙ)
    …
end
```

Should you use julia ?

- Do you 'just' need data analysis, automation, and pretty, customized plots?
  - Then, no

- **Or do you also write custom numeric algorithms / simulations?**
  - **Then, yes :)**
  - ..Unless you already know Matlab and don't have the time
  - ..Plus, Python and R have huge ecosystems of packages that might already do your custom thing
    - Also, Python has **Numba** for JIT-optimization of hot inner loops (numba.pydata.org). That might be enough

Code sharing, git, GitHub

- tfiers.github.io/phd
    - made with JupyterBook
    - auto-built and -published with GitHub Actions on GitHub Pages

- github.com/schluppeck/ng-data-club
    - /presentations subdirectory, by ISO8601 date+slug