Introduction to Julia:
Why are we doing this to you?

(Spring 2020)

Steven G. Johnson, MIT Applied Math

MIT classes 18.06, 18.S191, 18.S096,
18.303, 18.330, 18.08[56],
18.335, 18.337, …

these slides, cheatsheets, links:

https://github.com/mitmath/julia-mit

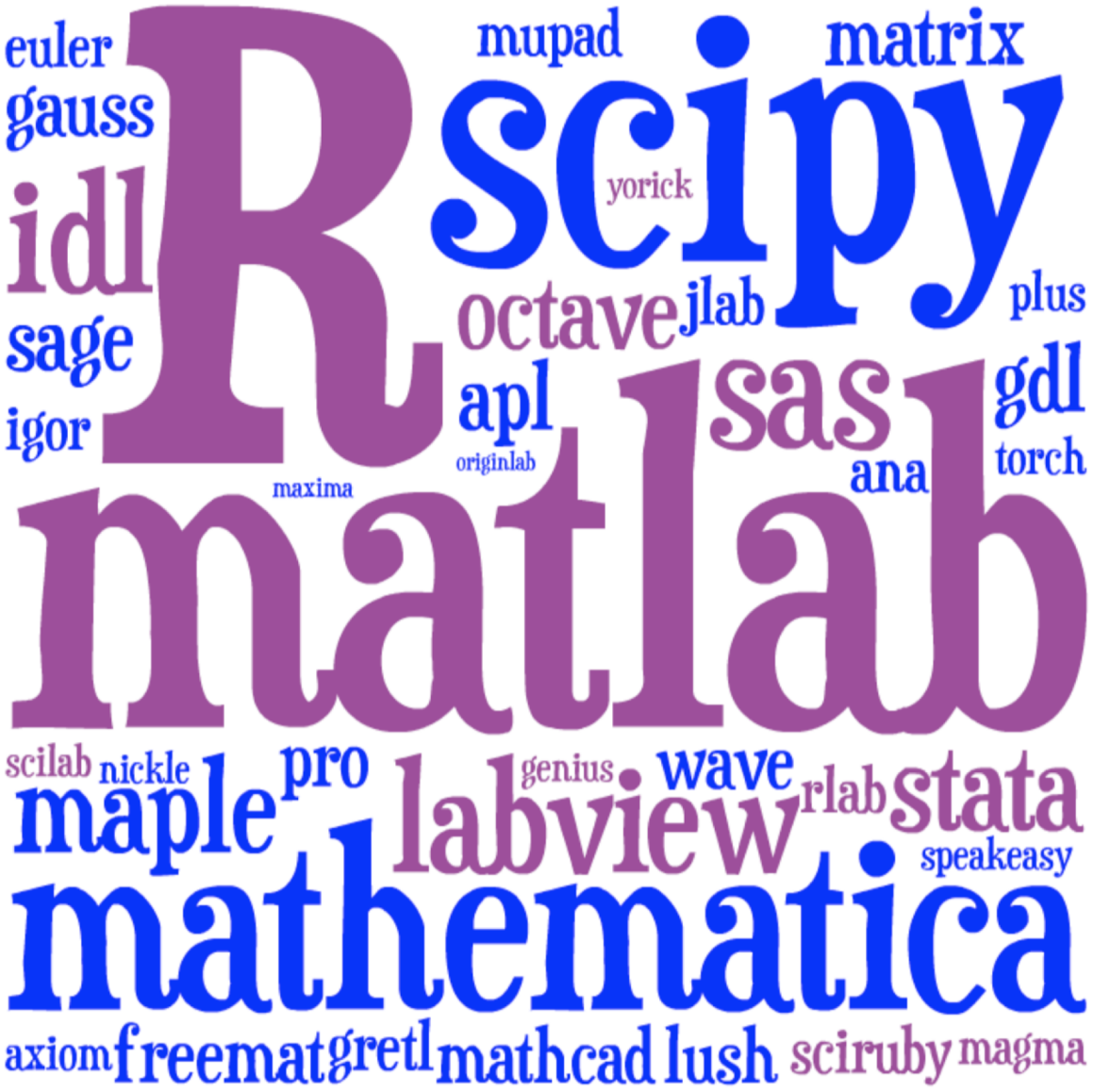
What language for teaching scientific computing?

For the most part, these are not hard-core programming courses,
and we only need little "throw-away" scripts and toy numerical experiments.

Almost any high-level, interactive (dynamic) language with easy facilities for linear algebra ($Ax=b$, $Ax=\lambda x$), plotting, mathematical functions, and working with large arrays of data would be fine.

And there are lots of choices…

Lots of choices for interactive math…



[image: Viral Shah]

Just pick the most popular?
Matlab or *Python* or *R*?

We must often use another a language for our "real work".

Traditional HL computing languages hit a performance wall in "real" work … eventually force you to C, Cython, Fortran, Numba…

A new programming language?

Viral Shah

Jeff Bezanson

Alan Edelman

Stefan Karpinski

[MIT]

julialang.org

[40+ developers with 100+ commits,
4000+ external packages, 7th JuliaCon in 2020]

[begun 2009, “0.1” in 2013, ~50k commits,
1.0 release in 2018, 1.5 in Aug. 2020]

As high-level and interactive as Matlab or Python+IPython,
as general-purpose as Python,
as productive for technical work as Matlab or Python+SciPy,
but as **fast as C**.

Lots of performance discussions, benchmarks, tutorials online…

Let’s just look at
one simple example.

Generating Vandermonde matrices

given x = [α_1, α_2, …], generate:

$$V = \begin{bmatrix} 1 & \alpha_1 & \alpha_1^2 & \dots & \alpha_1^{n-1} \\ 1 & \alpha_2 & \alpha_2^2 & \dots & \alpha_2^{n-1} \\ 1 & \alpha_3 & \alpha_3^2 & \dots & \alpha_3^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha_m & \alpha_m^2 & \dots & \alpha_m^{n-1} \end{bmatrix}$$

NumPy (`numpy.vander`): *[follow links]*

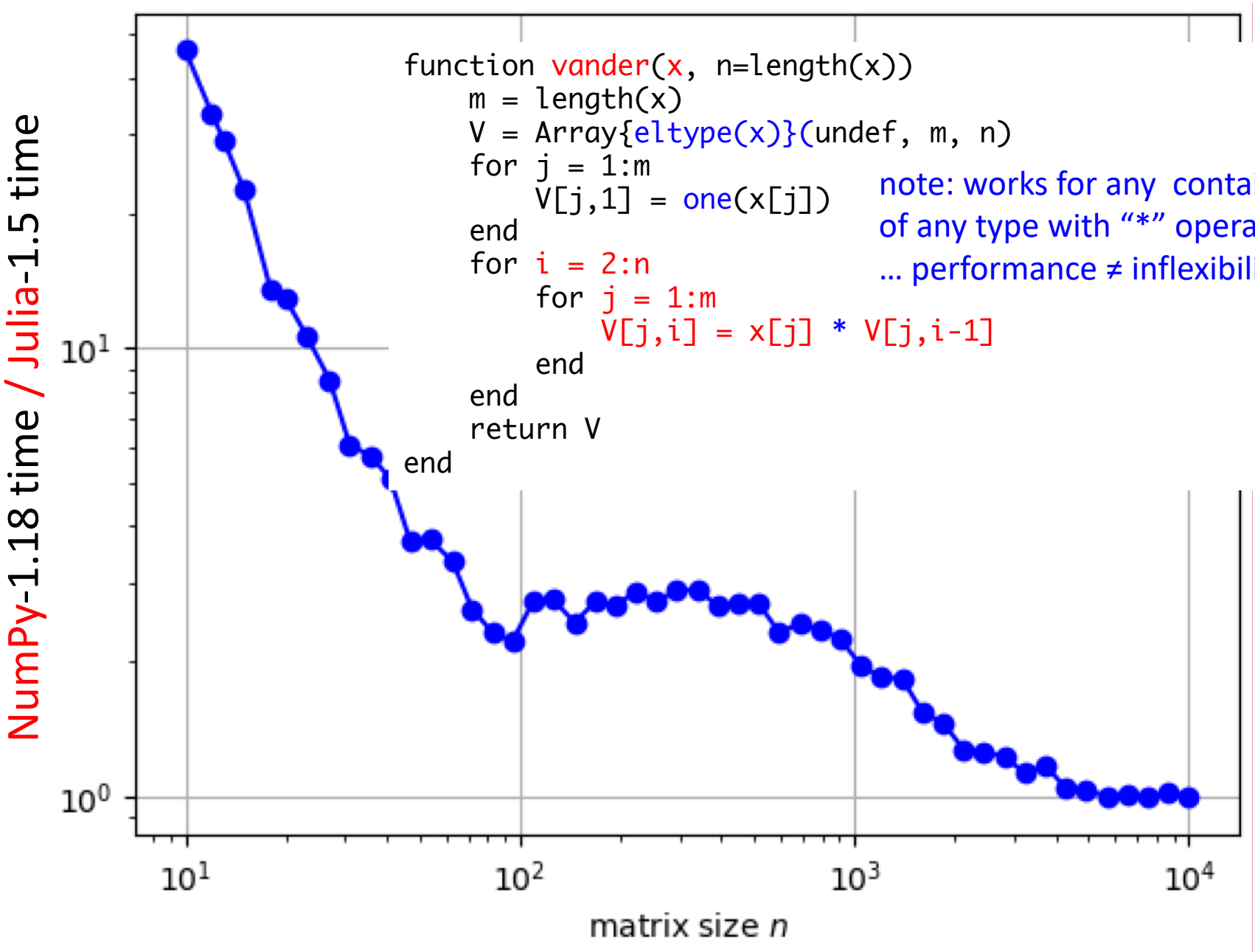
Python code …wraps C code
… wraps generated C code

type-generic at high-level, but
low level limited to small set of types.

Writing fast code "in" Python or Matlab = mining the standard library for pre-written functions (implemented in C or Fortran).

If the problem doesn't "vectorize" into built-in functions,
if you have to write your own inner loops … sucks for you.

Vandermonde matrices: Julia vs NumPy




```
function vander(x, n=length(x))
    m = length(x)
    V = Array{eltype(x)}(undef, m, n)
    for j = 1:m
        V[j,1] = one(x[j])
    end
    for i = 2:n
        for j = 1:m
            V[j,i] = x[j] * V[j,i-1]
        end
    end
    return V
end
```

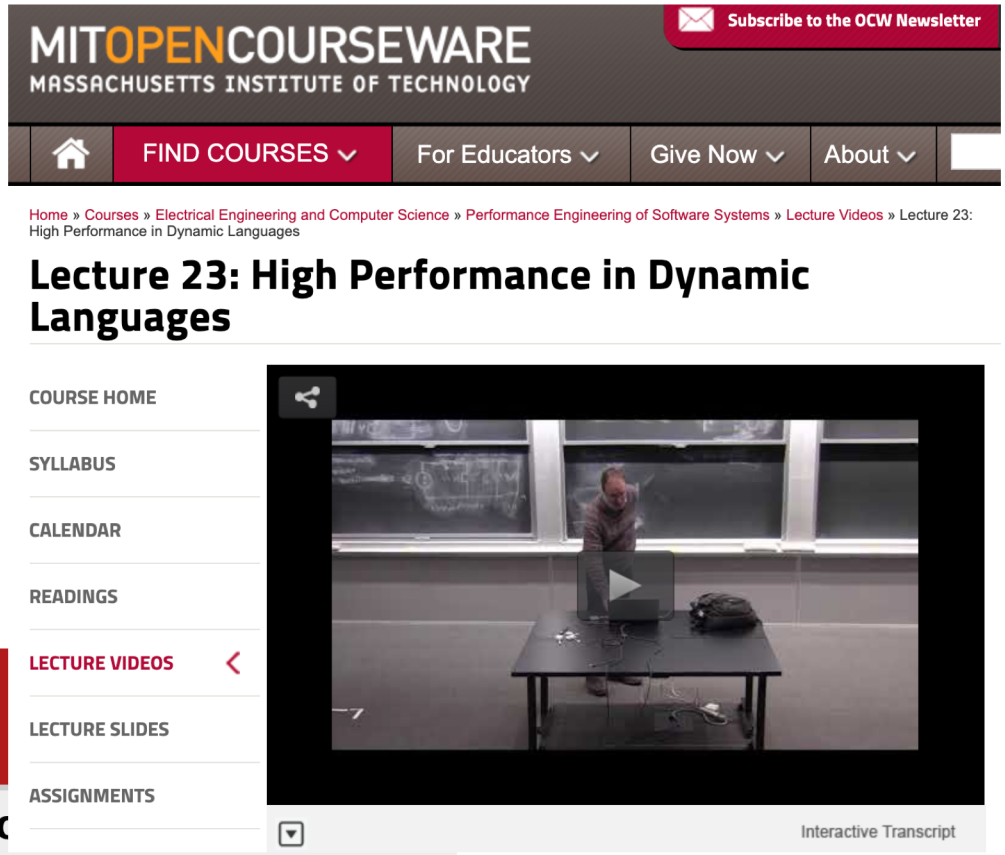
note: works for any container of any type with "*" operation

… performance ≠ inflexibility

Why is Julia fast? A topic for another day.

See my 6.172 “guest lecture” from 2018:

https://bit.ly/2QUrgB4



The original paper:

arXiv.org > cs > arXiv:1411.1607

Computer Science > Mathematical Sc

[Submitted on 6 Nov 2014 (v1), last revised 19 Jul 2015 (this version, v4)]

Julia: A Fresh Approach to Numerical Computing

Jeff Bezanson, Alan Edelman, Stefan Karpinski, Viral B. Shah

But I don't "need" performance!

For lots of problems, especially "toy" problems in courses, Matlab/Python performance is good enough.

But if use those languages for all of your "easy" problems, then you won't be prepared to switch when you hit a hard problem. When you **need** performance, it is too late.

You don't want to learn a new language at the same time that you are solving your first truly difficult computational problem.

Just vectorize your code?

= rely on mature external libraries,
operating on large blocks of data,
for performance-critical code

Good advice! But…

- Someone has to write those libraries.
- Eventually that person will be you.
  — some problems are impossible or just very awkward to vectorize.

But everyone else is using Matlab/Python/R/…

Julia is still a young, niche language. That imposes real costs — lack of familiarity, rough edges, continual language changes. These are real obstacles.

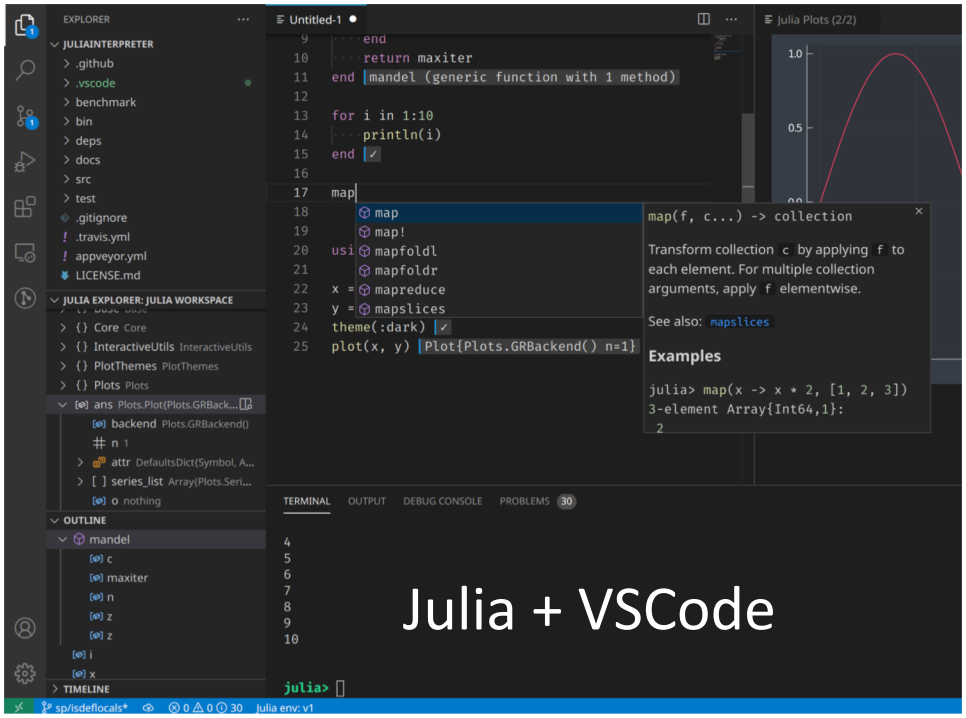
But it also gives you advantages that Matlab/Python users don't have.

But I lose access to all the libraries available for other languages?

Very easy to call C/Fortran libraries from Julia, and also to call Python…

Julia Coding Environments

Of course, full-featured “IDE” coding environments are available with VSCode, Atom, not to mention Emacs, Vim, …



and Julia has the usual features for organizing large programs into files, data structures, modules…

+ integrated hypertext documentation support…

+ built-in github-integrated package manager for distributing & installing packages and dependencies…

… but today we’re only looking at interactive snippets of code:
Julia as a “glorified calculator.”

Julia Interactive-Computing Environments

Jupyter notebooks:
code+text+
equations+images+…

+ Julia backend
= IJulia

Pluto.jl

Enter cell code...

Fourier sine series

It is a remarkable fact that the sine functions $\sin(n\pi x)$ are **orthogonal** under the "dot" product:

$$\int_0^1 \sin(m\pi x)\sin(n\pi x)\,dx = \begin{cases} 0 & m \neq n \\ \frac{1}{2} & m = n \end{cases}.$$

Let's plot a few of these functions:

In [2]:

```
x = linspace(0,1,200)
for n = 1:4; plot(x, sin.(n*π*x), "-"); end
legend([L"\sin(\pi x)", L"\sin(2\pi x)", L"\sin(3\pi x)",
title("orthogonal sine functions")
```