A Fresh Approach to Scientific Computing

Manuel Behrendt & Ludwig Böss

USM/LMU 12.02.2019

What is Julia?

It solves the two-language problem!

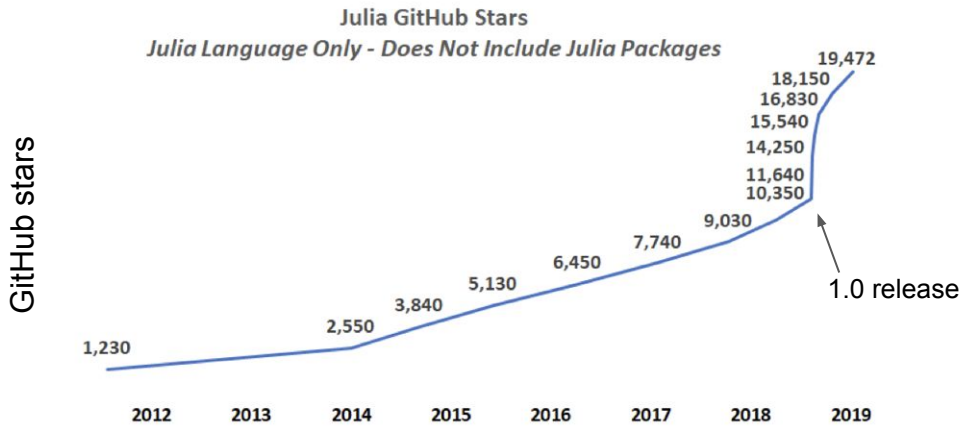
A single language that is good for scientific computing, machine learning, data mining, large-scale linear algebra, parallel computing.

- open source
- runs on multiple platforms (macOS, Linux, Windows, ARM processors)
- high-level syntax -> (easy to use/understand; fast code developing, more readability)
- dynamically-typed/optionally typed
- was designed from the beginning for high performance
- compiles to efficient native code (**J**ust-**I**n-**T**ime)
- multiple dispatch as a paradigm -> express many object-oriented and functional programming patterns.

Julia Growth Metrics

	Cumulative Total as of Jan 2018	Cumulative Total as of Jan 2019	Growth
Number of Julia Downloads Initiated	1.8 million	3.2 million	+78%
Julia Packages Available*	1,688	2,462	+46%
Number of News Articles Mentioning Julia	93	253	+172%
Julia Discourse Threads + Stack Overflow Questions	8,620	16,363	+90%
GitHub Stars for Julia Language (Not including Julia Packages)	9,626	19,472	+102%
Published Citations of Julia: A Fresh Approach to Numerical Computing (2017) and Julia: A Fast Dynamic Language for Technical Computing (2012)	613	1,048	+71%

Julia GitHub Stars: Julia GitHub stars have nearly doubled since the Aug 2018 release of Julia 1.0 and Julia is one of the top 10 languages on GitHub measured by stars and forks.



Credit: JuliaComputing newsletter 9.1.2019

(For 2018)

Top Machine Learning Languages on GitHub

1 Python
2 C++
3 JavaScript
4 Java
5 C#
6 Julia
7 Shell
8 R
9 TypeScript
10 Scala

GitHub

Top Machine Learning Projects on GitHub

1 tensorflow/tensorflow
2 scikit-learn/scikit-learn
3 explosion/spaCy
4 JuliaLang/julia
5 CMU-Perceptual-Computing-Lab/openpose
6 tensorflow/serving
7 thtrieu/darkflow
8 ageitgey/face_recognition
9 RasaHQ/rasa_nlu
10 tesseract-ocr/tesseract

GitHub

GitHub reports that Julia ranks #4 on the list of the top machine learning projects by contribution and #6 on the list of top machine learning languages on GitHub.

https://github.blog/2019-01-24-the-state-of-the-octoverse-machine-learning/

The Two Language Problem

Develop algorithms and **prototype** your program in a **dynamic language** like MATLAB or Python.

Rewrite all or parts in a **compiled** language like C++ or Fortran.

- Use **external libraries** e.g. Numpy written in a low-level language.
- Rewrite parts of the code to **use e.g. Cython or Numba** to make it more performant.

Post-process and **reduce** huge **data** sets from your simulation in a **low-level language**.

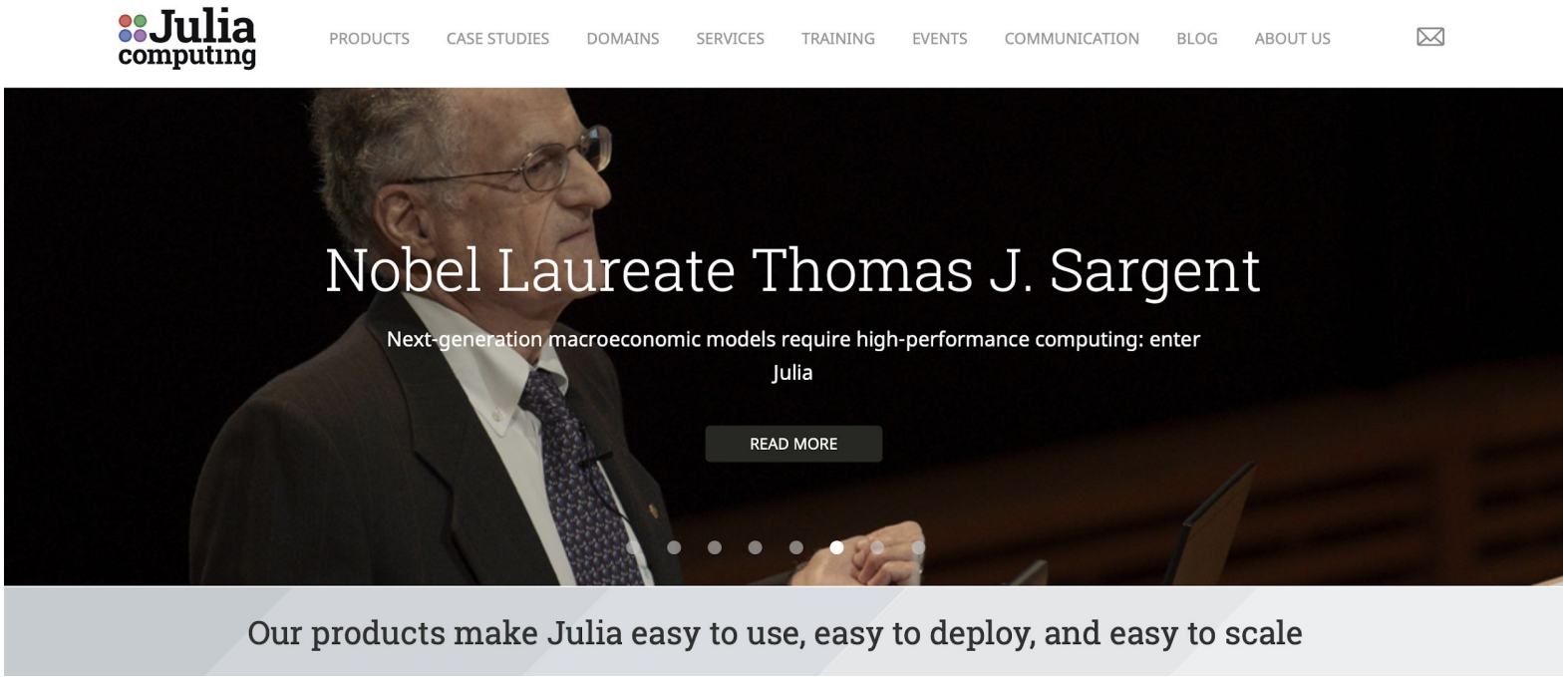
Analyze further and **plot** data in a **high-level** language like Python or R.

-> need to master several languages -> programming effort, complex to implement and deploy

-> portability problems -> http://www.stochasticlifestyle.com/why-numba-and-cython-are-not-substitutes-for-julia/

The Julia lab embraces openness and the solving of human problems. Today, the Julia Lab collaborates with a variety of researchers on real-world problems and applications, while simultaneously working on the core language and its ecosystem.

https://julia.mit.edu/



- Custom design and development
- Building wrappers for open source or commercial software
- Developing integrations with third party software
- Developing packages that can be open source or proprietary
- Software design
- Design reviews
- Code reviews
- Performance engineering, scalability

https://juliacomputing.com

JULIA USERS, PARTNERS AND EMPLOYERS HIRING JULIA PROGRAMMERS

https://juliacomputing.com

WHAT JULIA USERS ARE SAYING

Julia is a great tool. We like Julia. We are very excited about Julia because our models are complicated. It's easy to write the problem down, but it's hard to solve it – especially if our model is high dimensional. That's why we need Julia. Figuring out how to solve these problems requires some creativity. This is a walking advertisement for Julia.

NOBEL LAUREATE THOMAS J. SARGENT

WHAT JULIA USERS ARE SAYING

The Julia code is therefore more than 100 times faster than the equivalent Python code. Multiple dispatch with function calls gives Julia extremely efficient code that is practically superior to any high-level language. Faster code in Julia can be achieved without any tricks like vectorization or outsourcing to C extensions. By contrast, such tricks are often necessary to speed up Python or R code.

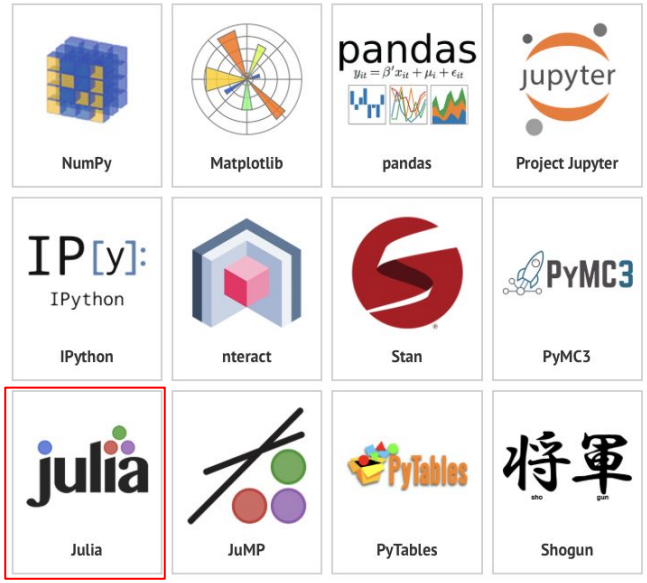
PROFESSOR MARK VOGELSBERGER, THEORETICAL ASTROPHYSICIST, MIT

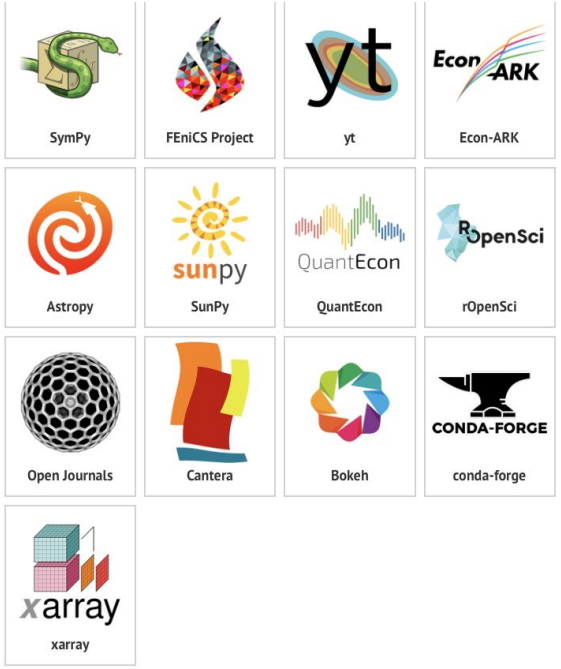
https://juliacomputing.com

NUMFOCUS

OPEN CODE = BETTER SCIENCE

The mission of NumFOCUS is to promote sustainable high-level programming languages, open code development, and reproducible scientific research. We accomplish this mission through our educational programs and events as well as through fiscal sponsorship of open source scientific computing projects. We aim to increase collaboration and communication within the data science and scientific computing community.





NERSC Astronomy

Parallel Supercomputing for Astronomy

Researchers use Julia on a NERSC supercomputer (650,000 cores) to speed astronomical image analysis 1,000x, catalog 188 million astronomical objects in 15 minutes and achieve peak performance of 1.5 petaflops per second

https://juliacomputing.com/case-studies/

Classes using Julia for teaching

Julia is now being used in several universities and online courses. If you know of other classes using Julia for teaching, please consi
this list.

- AGH University of Science and Technology, Poland
  - Signal processing in medical diagnostic systems (Tomasz Pieciak), Spring 2015
- Arizona State University
  - MAT 423, Numerical Analysis (Prof. Clemens Heitzinger), Fall 2014
- Azad University, Science and Research Branch
  - CE 3820, Modeling and Evaluation (Dr. Arman Shokrollahi), Fall 2014
- Brown University
  - CSCI 1810, Computational Molecular Biology (Prof. Benjamin J. Raphael), Fall 2014
- Budapest University of Technology and Economics
  - [Applications of Differential Equations and Vector Analysis for Engineers II.] (Brigitta Szilágyi)
- City University of New York
  - MTH 229, Calculus Computer Laboratory (Prof. John Verzani), Spring 2014. Also see the MTH 229 Projects page.
- Cornell University
  - CS 5220, Applications of Parallel Computers (Prof. David Bindel), Spring 2014
- École Polytechnique Fédérale de Lausanne [CIVIL 557] Decision-aid methodologies in transportation (Mor Kaspi, Virginie Lurk
- Einaudi Institute for Economics and Finance, Rome
  - Econometrics of DSGE Models (Giuseppe Ragusa)
- Emory University
  - MATH 346, Introduction to Optimization Theory (Prof. Lars Ruthotto), Spring 2015
  - MATH 516, Numerical Analysis II (Prof. Lars Ruthotto), Spring 2015
- Federal Rural University of Rio de Janeiro (UFRRJ)
  - TM429, Introduction to Recommender Systems (Prof. Filipe Braida), Fall 2016, Spring 2017
- Federal University of Alagoas (*Universidade Federal de Alagoas*, UFAL)
  - COMP272, Distributed Systems (Prof. André Lage-Freitas): 2015, 2016, and 2017
- Federal University of Paraná (*Universidade Federal do Paraná*, UFPR)
  - CM103, Mathematics Laboratory (Prof. Abel Soares Siqueira): 2016, 2017, and 2018
  - CM106, Nonlinear Optimization (Prof. Abel Soares Siqueira): 2018
- Federal University of Uberlândia, Institute of Physics
  - GFM050, Física Computacional (Prof. Gerson J. Ferreira), Fall 2016
- Hadsel High School, Stokmarknes, Nordland, Norway
  - AppINLA, Modern Applications of Numerical Linear Algebra (Prof. Ivan Slapnicar), June 2016
- Iowa State University
  - STAT 590F, Topics in Statistical Computing: Julia Seminar (Prof. Heike Hofmann), Fall 2014
- Luiss University Rome, Department of Economics and Finance
  - Econometric Theory (Giuseppe Ragusa)
- Massachusetts Institute of Technology (MIT)
  - 6.251/15.081, Introduction to Mathematical Programming (Prof. Dimitris J. Bertsimas), Fall 2015
  - 18.06, Linear Algebra: Fall 2015, Dr. Alex Townsend; Fall 2014, Prof. Alexander Postnikov; Fall 2013, Prof. Alan Edelman

https://julialang.org/learning/

Small performance comparison

(Lenovo Thinkpad Edge E540), a 64-bit system with 16 GB of RAM running Linux Mint 19 and GCC 7.3.0

Purpose: Compare simple code written in three languages

When each function is called, the six parameters are large arrays with 1M of elements.

Julia code

```
f(r, x1, x2, x3, x4, x5, x6, x7, x8) = @. r = x1 + x2
g(r, x1, x2, x3, x4, x5, x6, x7, x8) = @. r = x1 + x2 - x3
h(r, x1, x2, x3, x4, x5, x6, x7, x8) = @. r = x1 + x2 - x3 + x4
i(r, x1, x2, x3, x4, x5, x6, x7, x8) = @. r = x1 + x2 - x3 + x4 - x5
j(r, x1, x2, x3, x4, x5, x6, x7, x8) = @. r = x1 + x2 - x3 + x4 - x5 + x6
k(r, x1, x2, x3, x4, x5, x6, x7, x8) = @. r = x1 + x2 - x3 + x4 - x5 + x6 - x7
l(r, x1, x2, x3, x4, x5, x6, x7, x8) = @. r = x1 + x2 - x3 + x4 - x5 + x6 - x7 + x8
```

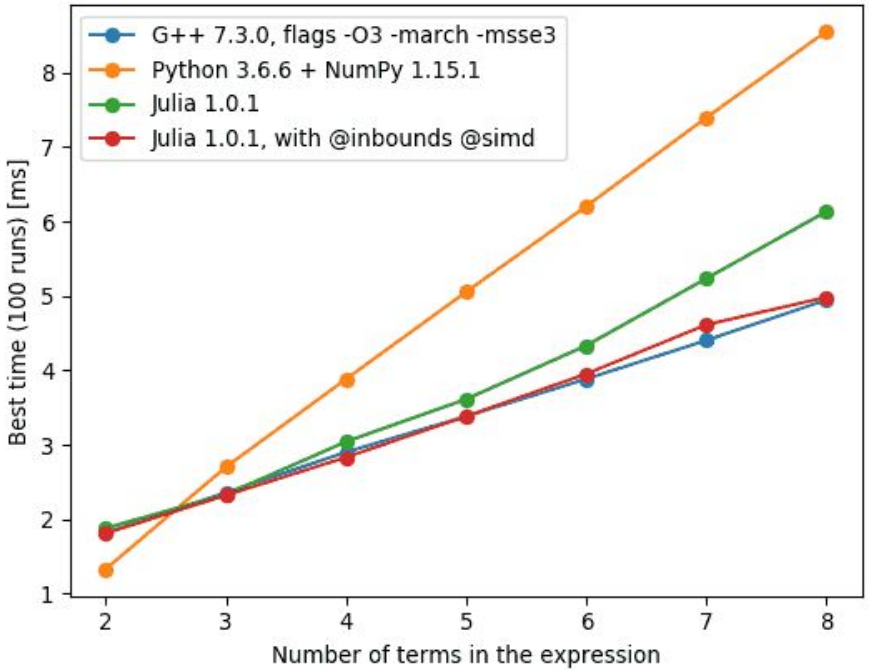
Python code

```
def f(r, x1, x2, x3, x4, x5, x6, x7, x8):
    r = x1 + x2

def g(r, x1, x2, x3, x4, x5, x6, x7, x8):
    r = x1 + x2 - x3

def h(r, x1, x2, x3, x4, x5, x6, x7, x8):
    r = x1 + x2 - x3 + x4
```

etc...

C++

```
void f(Vect &result, const Vect &x1, const Vect &x2, const Vect &x3,
       const Vect &x4, const Vect &x5, const Vect &x6, const Vect &x7,
       const Vect &x8) {
  for (size_t i = 0; i < x1.size(); ++i) {
    result[i] = x1[i] + x2[i];
  }
}

void g(Vect &result, const Vect &x1, const Vect &x2, const Vect &x3,
       const Vect &x4, const Vect &x5, const Vect &x6, const Vect &x7,
       const Vect &x8) {
  for (size_t i = 0; i < x1.size(); ++i) {
    result[i] = x1[i] + x2[i] - x3[i];
  }
}
```



Maurizio Tomasi:
https://github.com/ziotom78/python-julia-c-

Syntax Presentation

Presentation: Use JUNO remotely

Pros

- Simple syntax
- Very fast if one function called several times
- Modern features (meta-programming, build-in parallel computing, missing values…)
- Directly call C, Fortran, Python, R...
- Package manager: Fast/easy installing
- Editors/IDEs support: Juno, Visual Studio, Jupyter, Vim, Emacs, Sublime,...

Cons

- Slower execution for interactive programming, since first function call = JIT compiling.
- Less packages available as for other languages (Python, C++,..), but maybe enough for you.
- Plotting packages are still lacking, but promising.

Interesting Packages

Over 2,400 registered packages (Feb 2019)

Julia Astro

Community Astronomy and Astrophysics packages for Julia

Packages

Packages are separated by functionality and Julia's declarative package manager takes care of resolving dependencies. You get just the functionality you need, and smaller packages lead to more rapid development.

AstroLib
Collection of generic astronomical and astrophysical functions
- Translation of many IDL AstroLib procedures

Docs

Cosmology
Distances in the Universe
- ΛCDM and w0-wa cosmologies
- Open, closed, flat variants

DustExtinction
Dust extinction laws & maps
- CCM (1989) and O'Donnell (1994) dust laws
- SFD (1998) galactic dust map

ERFA
Low-level ERFA wrapper
- Wrapper for erfa C library
- Time system conversions

EarthOrientation
Earth orientation parameters
- Calculate Earth orientation parameters from IERS tables

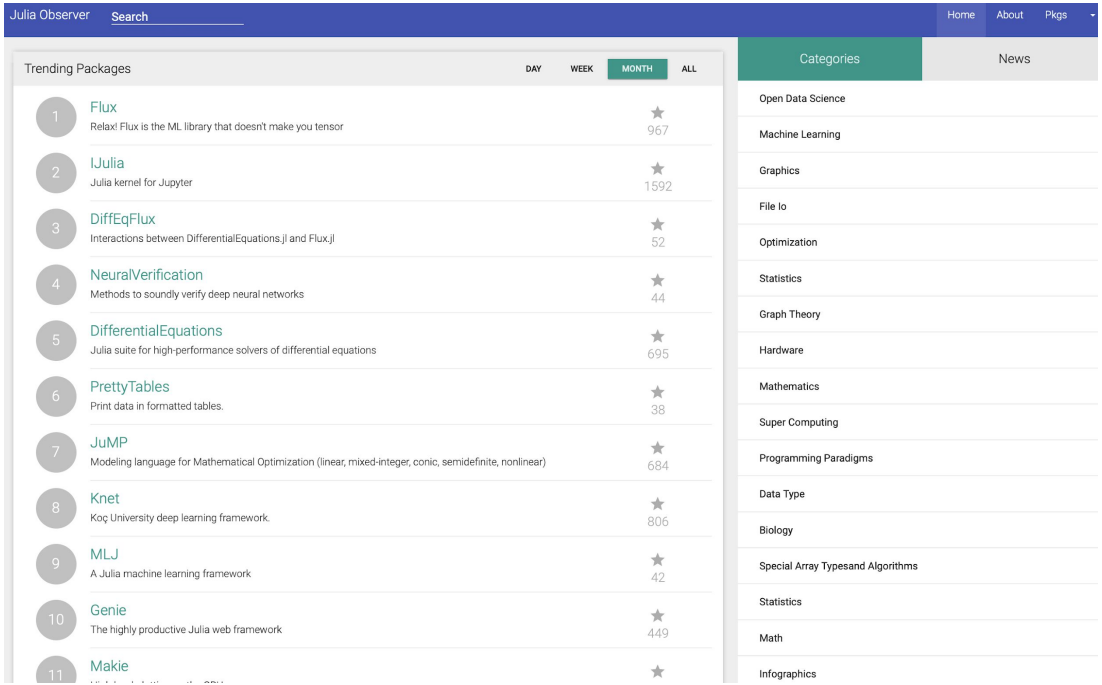
Docs

FITSIO
Reading and writing FITS files
- Image and table extensions
- Based on cfitsio C library

Docs

WCS
World Coordinate System transformations

http://juliaastro.github.io



https://juliaobserver.com

Machine learning presentation:

- Knet.jl

- Flux.jl

- Tensorflow.jl (Wrapper for the entire C-Library)

- ScikitLearn.jl (Wrapper for the Python Library)

Useful Links/Resources

Julia/Python/Matlab syntax comparison: https://cheatsheets.quantecon.org

Code, learning material, videos etc. : https://julialang.org
https://en.wikibooks.org/wiki/Introducing_Julia

Find packages: https://juliaobserver.com

Julia-Pro and Julia case studies: https://juliacomputing.com/

Talks and conferences: https://www.youtube.com/user/JuliaLanguage

Astro-Libraries: http://juliaastro.github.io

Other performance tests: https://julialang.org
https://modelingguru.nasa.gov/docs/DOC-2676

OOP in Julia: https://stackoverflow.com/questions/33755737/julia-oop-or-not