Computational Economics

SciencesPo 2019

Florian Oswald

Today's Agenda

1. Logistics.

Today's Agenda

1. Logistics.
2. Why Economists Must Talk About Computing.

Today's Agenda

1. Logistics.
2. Why Economists Must Talk About Computing.
3. High Performance Computing Overview.

Today's Agenda

1. Logistics.
2. Why Economists Must Talk About Computing.
3. High Performance Computing Overview.
4. How to Choose a Programming Language.

Logistics and Course Structure

- Meetings: Every Monday except 25 Feb and 4 March

Logistics and Course Structure

- Meetings: Every Monday except 25 Feb and 4 March
- Communication: Slack

Logistics and Course Structure

- Meetings: Every Monday except 25 Feb and 4 March
- Communication: Slack
- Materials: All online via my website

Logistics and Course Structure

- Meetings: Every Monday except 25 Feb and 4 March
- Communication: Slack
- Materials: All online via my website
- Grades: 60% homeworks, 40% term project

Logistics and Course Structure

- Meetings: Every Monday except 25 Feb and 4 March
- Communication: Slack
- Materials: All online via my website
- Grades: 60% homeworks, 40% term project
- Term Project: Develop CourseMatch for SciencesPo

Economists and Computation I

This set of slides borrows heavily from Jesus Fernandez-Villaverde's lectures. Thanks Jesus!

Computation has become an important tool in Economics:

1. Macro: Solution of DSGE models, forecasting models, ...
2. Micro: Agent-based models, games, life-cycle models, high-dimensional fixed effects models ...
3. Econometrics: Simulation-based estimators and large datasets, ...
4. Trade and spatial economics: multi-country-firm-type models, with dynamics, ...
5. Finance: Asset Pricing, Value at Risk models, ...

Economists and Computation II

- Ken Judd: Computation often *complements*, rather than substitutes, theory.

> If theory shows that some partial derivative of interest is *positive*, computation can tell us *how positive*.

Economists and Computation II

- Ken Judd: Computation often *complements*, rather than substitutes, theory.

> If theory shows that some partial derivative of interest is *positive*, computation can tell us *how positive*.

- Economics is not different from many other fields.
    - Computational Biology (`R` Bioconductor)
    - Computational Chemistry
    - Physics, Engineering, Applied Maths
    - Comparative Literature
    - Astronomy

Economists and Computation II

- Ken Judd: Computation often *complements*, rather than substitutes, theory.

> If theory shows that some partial derivative of interest is *positive*, computation can tell us *how positive*.

- Economics is not different from many other fields.
  - Computational Biology (`R` Bioconductor)
  - Computational Chemistry
  - Physics, Engineering, Applied Maths
  - Comparative Literature
  - Astronomy

- Importance of Computation as a tool will likely further increase.

What Does This Mean for You?

1. You (will) spend a considerable amount of your time writing code.

What Does This Mean for You?

1. You (will) spend a considerable amount of your time writing code.

2. You (will) collaborate with coauthors and colleagues on code.

What Does This Mean for You?

1. You (will) spend a considerable amount of your time writing code.
2. You (will) collaborate with coauthors and colleagues on code.
3. You (will) read and evaluate papers that use computational methods

What Does This Mean for You?

1. You (will) spend a considerable amount of your time writing code.
2. You (will) collaborate with coauthors and colleagues on code.
3. You (will) read and evaluate papers that use computational methods
4. (Hopefully you will be supplied with the paper's code for your evaluation.)

What Does This Mean for You?

1. You (will) spend a considerable amount of your time writing code.
2. You (will) collaborate with coauthors and colleagues on code.
3. You (will) read and evaluate papers that use computational methods
4. (Hopefully you will be supplied with the paper's code for your evaluation.)
5. For all practical purposes, **you are a research software engineer**. 👷‍♀️ 👷🏽‍♂️
   Carefully choose the best methods for software engineering at any time.

Gentzkow and Shapiro

> Here is a good rule of thumb: If you are trying to solve a problem, and there are multi-billion dollar firms whose entire business model depends on solving the same problem, and there are whole courses at your university devoted to how to solve that problem, you might want to figure out what the experts do and see if you can't learn something from it.

- Gentzkow and Shapiro: Code and Data
- GSLab-econ
- RA-manual
- Whatever you do: don't reinvent the wheel. ☸️

HPC Overview

(High Performance Computing)

HPC

(This is a high-level overview. We will go very practial with HPC later on.)

HPC

(This is a high-level overview. We will go very practial with HPC later on.)

- HPC methods help us solve complex computational problems.

HPC

(This is a high-level overview. We will go very practial with HPC later on.)

- HPC methods help us solve complex computational problems.
- The computational complexity may arise purely from the *scale* of the problem.

HPC

(This is a high-level overview. We will go very practial with HPC later on.)

- HPC methods help us solve complex computational problems.
- The computational complexity may arise purely from the *scale* of the problem.
- Some Examples:
    - Any (dynamic programming) model with a large state space.
    - DSGE models with many shocks.
    - Structural Estimation.
    - Handling large datasets.

Parallel Processing

- Many HPC solutions *parallelize* computation somehow.

Parallel Processing

- Many HPC solutions *parallelize* computation somehow.
- Not all solutions require a traditional computing cluster at your university.

Parallel Processing

- Many HPC solutions *parallelize* computation somehow.
- Not all solutions require a traditional computing cluster at your university.
- Examples:

1. Multicore CPUs (in single computer, or connected in a cluster)
2. Many-core coprocessors
3. GPUs (Graphical processing Units)
4. TPUs (tensor processing units) - Google's GPU
5. ...

Parallel Processing

- Many HPC solutions *parallelize* computation somehow.
- Not all solutions require a traditional computing cluster at your university.
- Examples:

1. Multicore CPUs (in single computer, or connected in a cluster)
2. Many-core coprocessors
3. GPUs (Graphical processing Units)
4. TPUs (tensor processing units) - Google's GPU
5. ...

- You can rent all of those machines in the cloud quite cheaply.

Example: Amazon Web Services (AWS) Costs

- Here are the prices for spot instances in the Paris region (per hour):

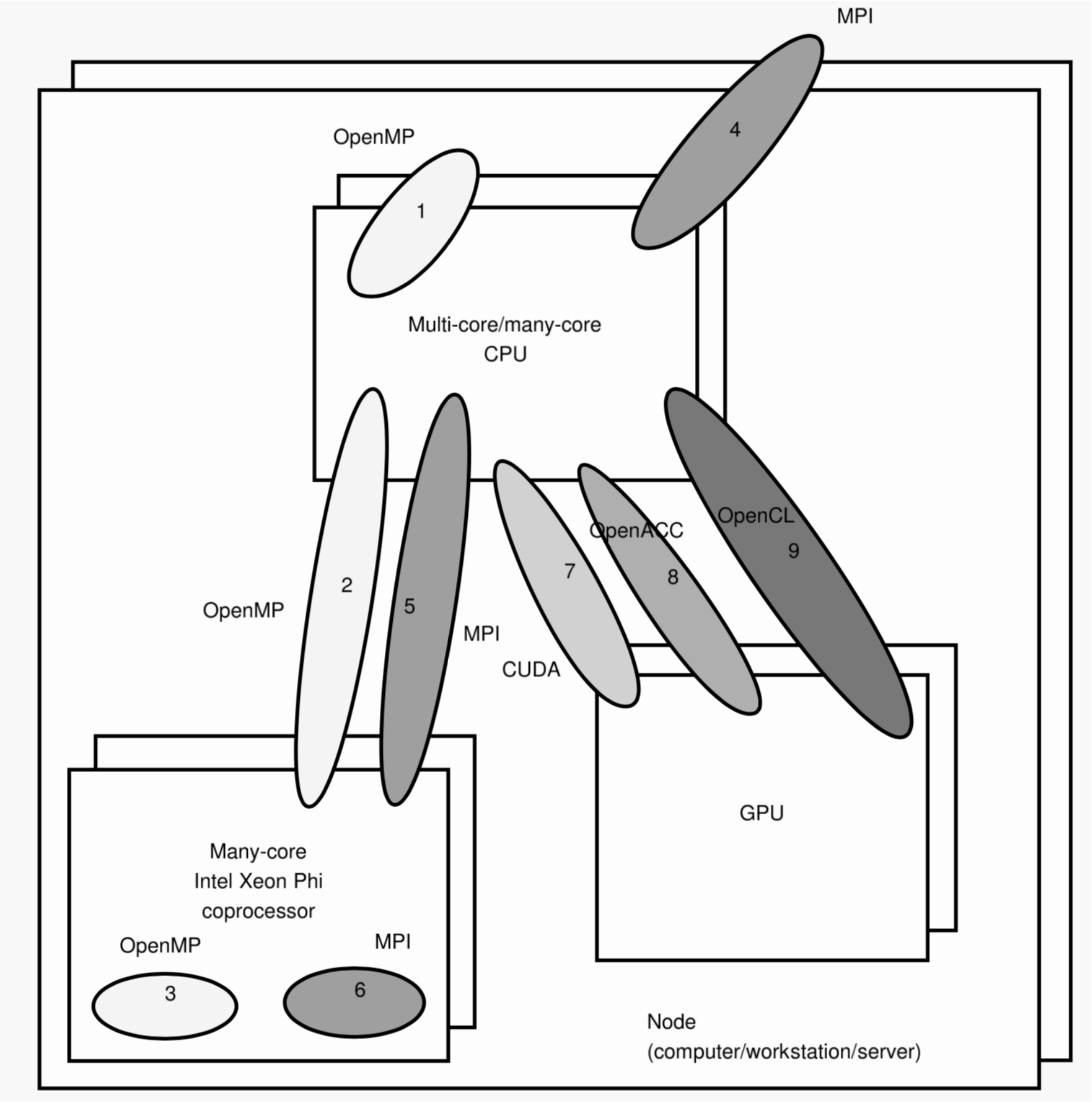
type	unix	windows
t2.micro	$0.004	$0.0086
t2.small	$0.0079	$0.0171
t2.medium	$0.0158	$0.0338
t2.large	$0.0317	$0.0597
t2.xlarge	$0.0634	$0.1044
t2.2xlarge	$0.1267	$0.1887
t3.nano	$0.0059	$0.0105

Example: Amazon Web Services (AWS) Costs

- Here are the prices for spot instances in the Paris region (per hour):

type	unix	windows
t2.micro	$0.004	$0.0086
t2.small	$0.0079	$0.0171
t2.medium	$0.0158	$0.0338
t2.large	$0.0317	$0.0597
t2.xlarge	$0.0634	$0.1044
t2.2xlarge	$0.1267	$0.1887
t3.nano	$0.0059	$0.0105

- So, compute *time* is relatively cheap. 💵
- More at this short presentation


MPI
OpenMP
1
4
Multi-core/many-core
CPU
OpenMP
2
5
MPI
CUDA
7
OpenACC
8
OpenCL
9
GPU
Many-core
Intel Xeon Phi
coprocessor
OpenMP
MPI
3
6
Node
(computer/workstation/server)

Speed and Total Time

- You will hear a lot of talk about the *speed* of some application, or the time it takes to complete.

Speed and Total Time

- You will hear a lot of talk about the *speed* of some application, or the time it takes to complete.
- All else equal, we prefer applications that run faster. ⏱️

Speed and Total Time

- You will hear a lot of talk about the *speed* of some application, or the time it takes to complete.
- All else equal, we prefer applications that run faster. ⏱️
- However, **your time** (developer) is more expensive than computing time.

Speed and Total Time

- You will hear a lot of talk about the *speed* of some application, or the time it takes to complete.
- All else equal, we prefer applications that run faster. ⏱️
- However, **your time** (developer) is more expensive than computing time.
- `Total Time = Development Time + Run Time`.

Speed and Total Time

- You will hear a lot of talk about the *speed* of some application, or the time it takes to complete.
- All else equal, we prefer applications that run faster. ⏱️
- However, **your time** (developer) is more expensive than computing time.
- `Total Time = Development Time + Run Time`.
- Proper Coding is **key**. We want to be fast in the **development** phase.

Speed and Total Time

- You will hear a lot of talk about the *speed* of some application, or the time it takes to complete.
- All else equal, we prefer applications that run faster. ⏱️
- However, **your time** (developer) is more expensive than computing time.
- `Total Time = Development Time + Run Time`.
- Proper Coding is **key**. We want to be fast in the **development** phase.
- Code Quality. Failing Fast. Exploration. Test-Driven Development.

Resources

- HPC Carpentry
- Victor Eijkhout
- Inside HPC
- A curriculum
- Jesus Fernandez-Villaverde's homepage

Programming Languages

How To Choose?

Choosing a Language

- You are an economics grad student and need to choose your *weapon of choice*.

Choosing a Language

- You are an economics grad student and need to choose your *weapon of choice*.
- You will use more than one language. (This is a good thing.)

Choosing a Language

- You are an economics grad student and need to choose your *weapon of choice*.
- You will use more than one language. (This is a good thing.)
- All Languages have pros and cons. I will give some opinionated advice.

Choosing a Language

- You are an economics grad student and need to choose your *weapon of choice*.
- You will use more than one language. (This is a good thing.)
- All Languages have pros and cons. I will give some opinionated advice.
- Then I will force you to use a certain language to complete homeworks 😜

Taxonomy of Languages

Fortran

CUDA

C

OpenCL

Low-level languages for max performance

R

Mathematica

Matlab

Python

High-level languages for max productivity

The Pros/Cons Rundown

Hello, World?

Hello, World?

- It's good custom to first print `hello world` when introducing a language. It's fun, but pretty uninformative. Like, *what's the performance of printing "hello, world"?*

Hello, World?

- It's good custom to first print `hello world` when introducing a language. It's fun, but pretty uninformative. Like, *what's the performance of printing "hello, world"?*

- Instead, we will show in each language how to implement the function `sum over an array` of values `a`:

  $$\text{sum}(a) = \sum_{i=1}^{n} a_i$$

  where n is the number of elements in a

- Later, we will then *benchmark* each language to see tradeoff between high- and low level languages.

`C/C++`

- The world pretty much runs on `C++`. Most of the apps on your phone do. Many flight computers computers do. 🛩

`C/C++`

- The world pretty much runs on `C++`. Most of the apps on your phone do. Many flight computers computers do. 🛩️
- If you now some `C++` and some `Unix` you know a lot already.

`C/C++`

- The world pretty much runs on `C++`. Most of the apps on your phone do. Many flight computers computers do. ✈️
- If you now some `C++` and some `Unix` you know a lot already.
- Developed at Bell Labs in 1980s.

C/C++

- The world pretty much runs on `C++`. Most of the apps on your phone do. Many flight computers computers do. 🛩
- If you now some `C++` and some `Unix` you know a lot already.
- Developed at Bell Labs in 1980s.
- All `C` programs are valid `C++`, not other way around.

```cpp
//sumvec.cpp
#include <iostream>
#include <vector>
int main(){
    std::vector<int> x;
    for (int i=1;i<5;i++){
        x.push_back(i);
    }
    int sum = 0;
    for (std::vector<int>::iterator i=x.begin();i!=x.end();i++){
        sum += *i;
    }
    std::cout << "sum is " << sum << std::endl;
}
//compile
g++ sumvec.cpp -o sum.x
```

C sum

- Defining the function:

```c
#include <stddef.h>
double c_sum(size_t n, double *X) {
    double s = 0.0;
    for (size_t i = 0; i < n; ++i) {
        s += X[i];
    }
    return s;
}
```

- place in `main()`, compile and run as above.

C++: Pros and Cons

Pros

- Very versatile.
- Continuously Evolving. C++2017 standard is current.
- Very performant.
- Excellent open source compilers.
- Very stable and widely used.
- Large community.

C++: Pros and Cons

Pros

- Very versatile.
- Continuously Evolving. C++2017 standard is current.
- Very performant.
- Excellent open source compilers.
- Very stable and widely used.
- Large community.

Cons

- Hard to learn. Pointers, Classes, OOP in general.
- Some find it hard to work with Compiled languages.
- It's easy to overcomplicate things for novices.

Python

* *The* general purpose language out there. Swiss Army Knife. All Terrain. 🚙

Python

- *The* general purpose language out there. Swiss Army Knife. All Terrain. 🚙
- Open Source

Python

- *The* general purpose language out there. Swiss Army Knife. All Terrain. 🚙
- Open Source
- Designed by Guido von Rossum.

Python

- *The* general purpose language out there. Swiss Army Knife. All Terrain. 🚙
- Open Source
- Designed by Guido von Rossum.
- Elegant, intuitive, full OOP support (Classes etc).

`python sum`

- we just use the built-in `sum`:

```
a = [1,2,3,4,5]
sum(a)
```

- that's it!

Python Pros/Cons

Pros

- Console: good for exploration.
- Many useful libraries (NumPy, SciPy, Pandas, matplotlib)
- Easy Unit Testing
- Very Performant String Manipulation
- Large community

Python Pros/Cons

Pros

- Console: good for exploration.
- Many useful libraries (NumPy, SciPy, Pandas, matplotlib)
- Easy Unit Testing
- Very Performant String Manipulation
- Large community

Cons

- Slow.
- Version `2.7` or `3.6`? Huge problem.
- High performance routes use annoted Python code. Numba: JIT compiler,Pypy: JIT compiler, Cython: compile to C++
- All feel like a Ferrari Engine on a Daihatsu.

R

- High level open source language for statistical computing.

`R`

- High level open source language for statistical computing.
- Ross Ihaka and Robert Gentleman developed `R` as a successor to John Chambers' `S`

`R`

- High level open source language for statistical computing.
- Ross Ihaka and Robert Gentleman developed `R` as a successor to John Chambers' `S`
- `R` went open source quickly via http://cran.r-project.org/

`R`

- High level open source language for statistical computing.
- Ross Ihaka and Robert Gentleman developed `R` as a successor to John Chambers' `S`
- `R` went open source quickly via http://cran.r-project.org/
- Tremendously rich add-on packages environment.

`R`

- High level open source language for statistical computing.
- Ross Ihaka and Robert Gentleman developed `R` as a successor to John Chambers' `S`
- `R` went open source quickly via http://cran.r-project.org/
- Tremendously rich add-on packages environment.
- Has basic OOP support via `S4` classes and methods.

`R sum`

- we just use the built-in `sum`:

```r
a = 1:5
sum(a)
```

- that's (again) it!

`R` Pros/Cons

Pros

- Excellent IDE Rstudio
- Thousands of high quality packages. `tidyverse` is a sensation in itself.
- *Many many* econometrics-related packages.
- Wide community.
- `Rcpp` is good to connect to `C++`
- Unbeatable for **spatial data**: the `sf` package.
- Very good for data processing.

`R` Pros/Cons

Pros

- Excellent IDE Rstudio
- Thousands of high quality packages. `tidyverse` is a sensation in itself.
- *Many many* econometrics-related packages.
- Wide community.
- `Rcpp` is good to connect to `C++`
- Unbeatable for **spatial data**: the `sf` package.
- Very good for data processing.

Cons

- Base `R` is slow.
- Not a modern language. Some quite arcane behaviours.
- `Rcpp` means you end up writing `C++` code.
- Not straightforward to write performant low-level code close to the math (i.e.: loops)

Fortran

- The Grandfather of all languages 👴🏾

Fortran

- The Grandfather of all languages 👴🏽
- FORTRAN (Formula Translation) was developed in 1957.

Fortran

- The Grandfather of all languages 👴🏽
- FORTRAN (Formula Translation) was developed in 1957.
- Still used by many economists.

Fortran

- The Grandfather of all languages 👴🏾
- FORTRAN (Formula Translation) was developed in 1957.
- Still used by many economists.
- Still used for many scientific problems (nuclear bombs design, wheather forecasts, phyical experiments, etc) 💣 🌦

- create a text file `test.f90`:

```
program sumit
    implicit none
    double precision a(5)   ! allocate an array with 5 slots
    a = (/1,2,3,4,5/)       ! fill with values
    print *,"result is ", sum(a)
end program sumit
```

- compile it and run it:

```
$ gfortran test.f90 -o test
$ ./test
 result is    15.000000000000000
```

FORTRAN Pros/Cons

Pros

- Relatively easy to learn.
- Good array support built in.
- Good parallelization support via MPI.
- Fast.

`FORTRAN` Pros/Cons

Pros

- Relatively easy to learn.
- Good array support built in.
- Good parallelization support via MPI.
- Fast.

Cons

- Different compilers implement different standards (Intel Fortran vs `GFORTRAN` array constructor, e.g.)
- Small user community.
- Not very many tutorials.
- Not faster than C++ (used to be true).
- Hard to automatize unit testing via e.g. pfunit.
- Language is very bare-bone.
- Hard to process data.

Victims of Speed vs Productivity Tradeoff?

- We have seen high-level langs (R, python etc) are good for productivity: Iterate fast on `try, fail, repeat`

Victims of Speed vs Productivity Tradeoff?

- We have seen high-level langs (R, python etc) are good for productivity: Iterate fast on `try, fail, repeat`
- We have seen that low level languages *run* fast, but are worse for productivity.

Victims of Speed vs Productivity Tradeoff?

- We have seen high-level langs (R, python etc) are good for productivity: Iterate fast on `try, fail, repeat`
- We have seen that low level languages *run* fast, but are worse for productivity.
- So we cannot have both speed and productivity.

Victims of Speed vs Productivity Tradeoff?

- We have seen high-level langs (R, python etc) are good for productivity: Iterate fast on `try, fail, repeat`
- We have seen that low level languages *run* fast, but are worse for productivity.
- So we cannot have both speed and productivity.
- Or can we?


YES WE CAN

Why We Created Julia

> We are power Matlab users. Some of us are Lisp hackers. Some are Pythonistas, others Rubyists, still others Perl hackers. There are those of us who used Mathematica before we could grow facial hair. There are those who still can't grow facial hair. We've generated more R plots than any sane person should. C is our desert island programming language.
> We are greedy: we want more.
> We want a language that's open source, with a liberal license. We want the speed of C with the dynamism of Ruby. We want a language that's homoiconic, with true macros like Lisp, but with obvious, familiar mathematical notation like Matlab. We want something as usable for general programming as Python, as easy for statistics as R, as natural for string processing as Perl, as powerful for linear algebra as Matlab, as good at gluing programs together as the shell. Something that is dirt simple to learn, yet keeps the most serious hackers happy. We want it interactive and we want it compiled.

Julia

- Modern language

Julia

- Modern language
- Built for high performance and parallel: LLVM-JIT

Julia

- Modern language
- Built for high performance and parallel: LLVM-JIT
- Dynamically typed: good interactive use

Julia

- Modern language
- Built for high performance and parallel: LLVM-JIT
- Dynamically typed: good interactive use
- Rich data type system

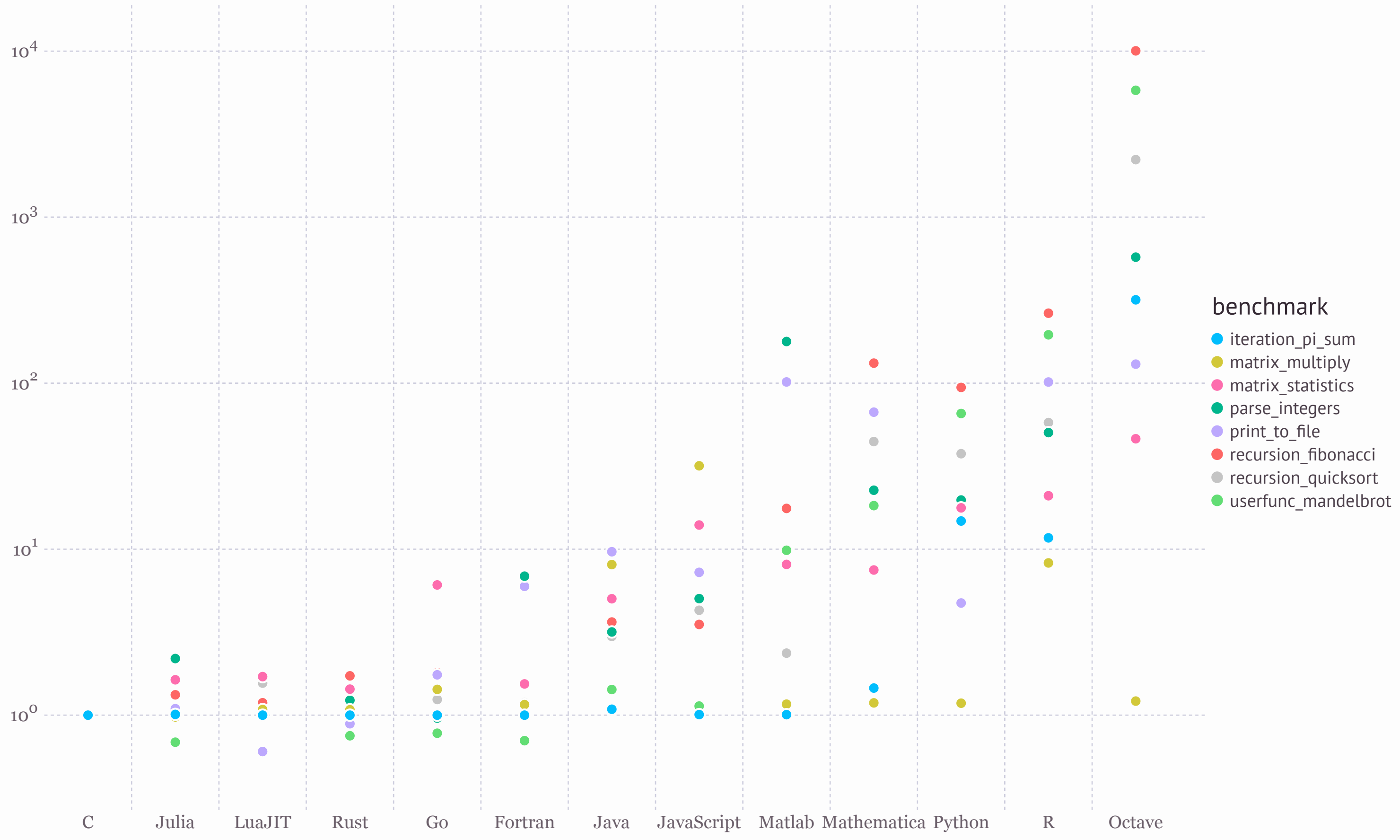
Julia

- Modern language
- Built for high performance and parallel: LLVM-JIT
- Dynamically typed: good interactive use
- Rich data type system
- Many Packages.

Julia

- Modern language
- Built for high performance and parallel: LLVM-JIT
- Dynamically typed: good interactive use
- Rich data type system
- Many Packages.
- Good Interoperability with other languages.

Benchmarks



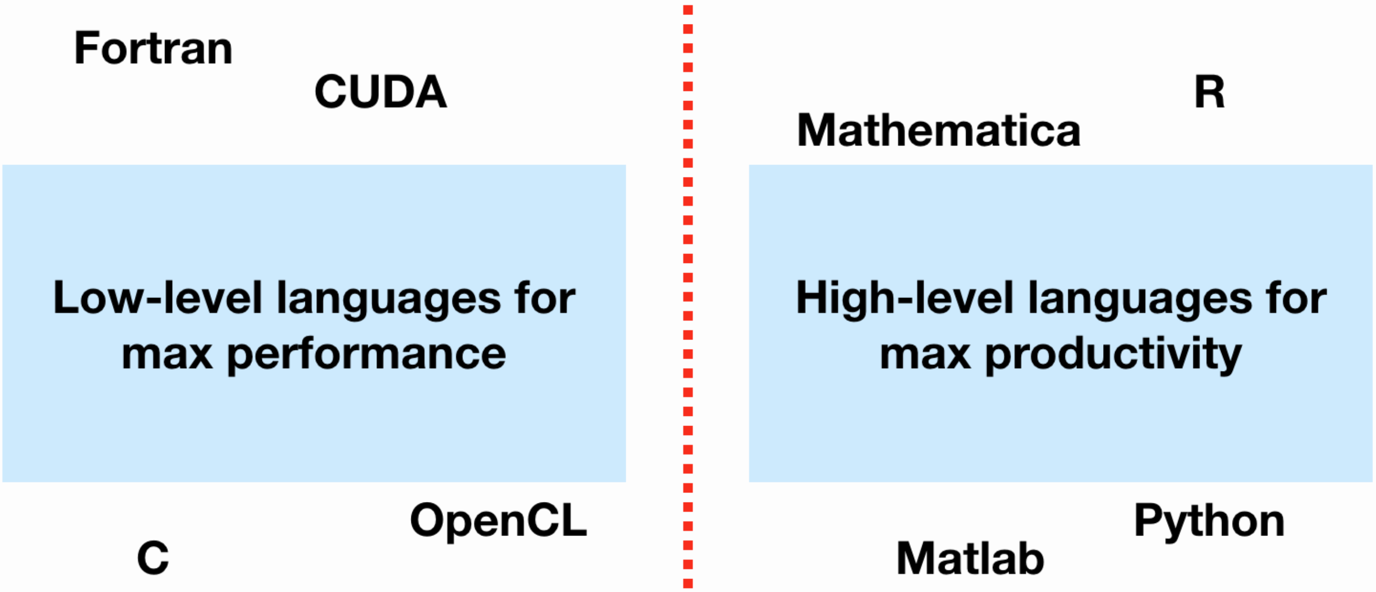
10^4
10^3
10^2
10^1
10^0
C
Julia
LuaJIT
Rust
Go
Fortran
Java
JavaScript
Matlab
Mathematica
Python
R
Octave
benchmark
iteration_pi_sum
matrix_multiply
matrix_statistics
parse_integers
print_to_file
recursion_fibonacci
recursion_quicksort
userfunc_mandelbrot

Which problem does Julia want to solve?

- Julia cofounder Stefan Karpinski talks about **the 2 languages problem**

Which problem does Julia want to solve?

- Julia cofounder Stefan Karpinski talks about **the 2 languages problem**
- **Key: Wall in scientific software stack creates a social barrier.** Developer and User are different people. (Basically: who knows C++/fortran?)



Comparing Programming Languages in Economics

- S. Boragoan Aruoba and Jesus Fernandez-Villaverde compare performance of languages on an Industry Standard Growth Model.

Comparing Programming Languages in Economics

- S. Boragoan Aruoba and Jesus Fernandez-Villaverde compare performance of languages on an Industry Standard Growth Model.
- Conceptually difficult to write *identical* code in different languages. Authors do a good job. (How many *tricks* is one allowed to use before an implementation is no longer comparable? Wouldn't users use all available tricks?)

Comparing Programming Languages in Economics

- S. Boragoan Aruoba and Jesus Fernandez-Villaverde compare performance of languages on an Industry Standard Growth Model.
- Conceptually difficult to write *identical* code in different languages. Authors do a good job. (How many *tricks* is one allowed to use before an implementation is no longer comparable? Wouldn't users use all available tricks?)
- I'll show you the results from an updated version of the paper

Results

	Mac		
Language	Version/Compiler	Time	Rel. Time
C++	GCC-7.3.0	1.60	1.00
	Intel C++ 18.0.2	1.67	1.04
	Clang 5.1	1.64	1.03
Fortran	GCC-7.3.0	1.61	1.01
	Intel Fortran 18.0.2	1.74	1.09
Java	9.04	3.20	2.00
Julia	0.7.0	2.35	1.47
	0.7.0, fast	2.14	1.34
Matlab	2018a	4.80	3.00
Python	CPython 2.7.14	145.27	90.79
	CPython 3.6.4	166.75	104.22
R	3.4.3	57.06	35.66
Mathematica	11.3.0, base	1634.94	1021.84
Matlab, Mex	2018a	2.01	1.26
Rcpp	3.4.3	6.60	4.13
Python	Numba 0.37.9	2.31	1.44
	Cython	2.13	1.33
Mathematica	11.3.0, idiomatic	4.42	2.76

Not Mentioned

There are many other languages out there. We have not mentioned

- Matlab: widely used in economics, but clearly inferior to julia (Expensive + license block)
- Stata: very popular in microeconometrics, old design, commercial software.
- Java
- Rust
- Scala
- Go

Final (Opinionated) Advice

Final (Opinionated) Advice

- It's a good idea to learn a bit of `C++`. Much is based upon it, so you will see things differently.

Final (Opinionated) Advice

- It's a good idea to learn a bit of `C++`. Much is based upon it, so you will see things differently.
- Avoid learning `FORTRAN` unless you need to use legacy code.

Final (Opinionated) Advice

- It's a good idea to learn a bit of `C++`. Much is based upon it, so you will see things differently.
- Avoid learning `FORTRAN` unless you need to use legacy code.
- Learn `julia` and `R`.

Final (Opinionated) Advice

- It's a good idea to learn a bit of `C++`. Much is based upon it, so you will see things differently.
- Avoid learning `FORTRAN` unless you need to use legacy code.
- Learn `julia` and `R`.
- Stata is dominated by `R`.