The Numpy Ecosystem

Jim Pivarski

Princeton University

November 15, 2018

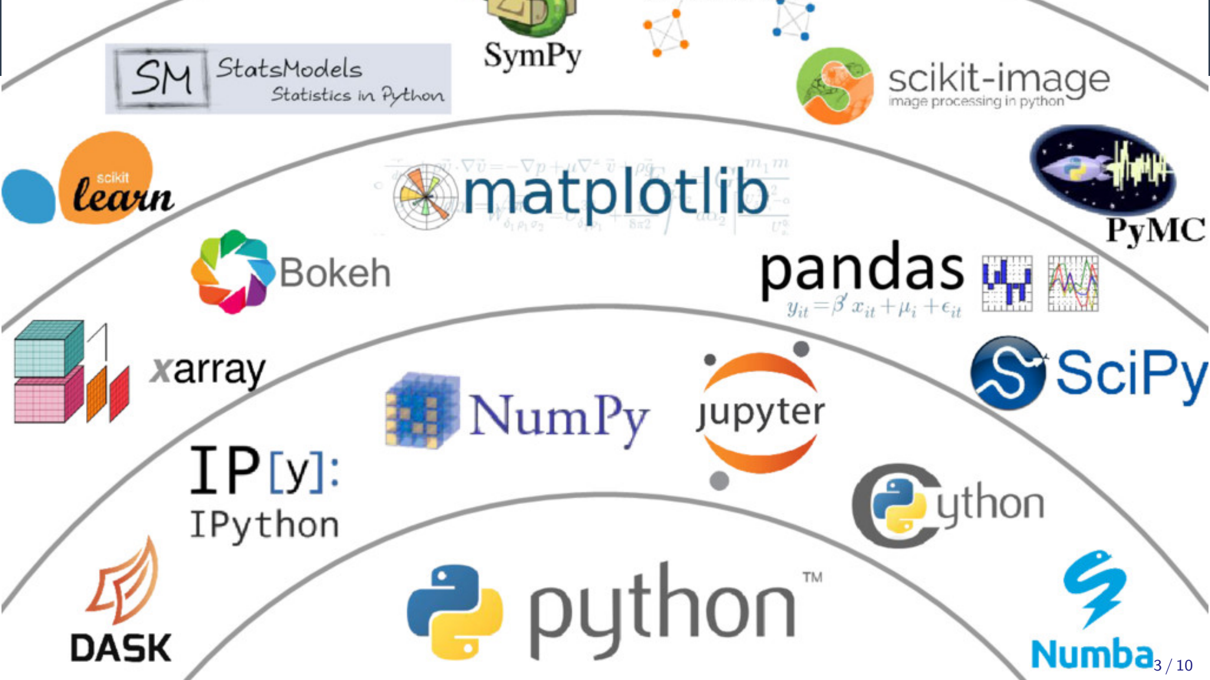
This afternoon

This morning, we focused on just one library— Numpy— and worked on putting its slicing interfaces together to achieve things you'd normally need for loops for.

This afternoon

This morning, we focused on just one library— Numpy— and worked on putting its slicing interfaces together to achieve things you'd normally need for loops for.

This afternoon, we switch to. . .


StatsModels
Statistics in Python
SymPy
scikit-image
image processing in python
scikit learn
matplotlib
PyMC
Bokeh
pandas
$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$
xarray
SciPy
NumPy
jupyter
IP[y]:
IPython
Cython
DASK
python™
Numba

Specific topics

Statistics tools

- **Pandas:** a central component, becoming as important as Numpy itself.

Specific topics

Statistics tools

- **Pandas:** a central component, becoming as important as Numpy itself.

Other than that, you're on your own. Statistical software are as varied as your domains.

Specific topics

Statistics tools

- **Pandas:** a central component, becoming as important as Numpy itself.

Other than that, you're on your own. Statistical software are as varied as your domains.

Speeding up code

- **Dask:** parallel processing; Multiple Instructions on Multiple Data (MIMD).
- **Numba:** compile a limited subset of Python, as-is, to C-like speeds.
- **Cython:** compile any Python code, but you have to modify it to make it fast.
- **CuPy:** run any Numpy operations on a GPU.
- **Numba-GPU:** compile limited Python for the GPU.
- **PyCUDA:** interface with raw CUDA through Numpy arrays.
- **ctypes:** cast pointers as Numpy arrays and run code in shared library (`*.so`) files.

Speeding up code

Fast software is not like a fast runner, who has some superior intrinsic ability.
All run at the same rate, but some have more hurdles on the track than others.

Hurdles, from smallest to largest

1. Unnecessary or repeated arithmetic
2. Arithmetic in separate instructions that could be in the same instruction (vectorization)
3. Transcendental functions or division
4. Unnecessary or nonsequential memory access; cache swapping
5. Virtual machine indirection
6. Boxing numbers as objects
7. Type checking at runtime
8. Unnecessary or nonsequential disk/network access
9. Wacky stuff

Hurdles, from smallest to largest

1. Unnecessary or repeated arithmetic
2. Arithmetic in separate instructions that could be in the same instruction (vectorization)
3. Transcendental functions or division
4. Unnecessary or nonsequential memory access; cache swapping
5. Virtual machine indirection
6. Boxing numbers as objects
7. Type checking at runtime
8. Unnecessary or nonsequential disk/network access
9. Wacky stuff

Compilation optimizes away most of #1, #2, and #4.

Hurdles, from smallest to largest

1. Unnecessary or repeated arithmetic
2. Arithmetic in separate instructions that could be in the same instruction (vectorization)
3. Transcendental functions or division
4. Unnecessary or nonsequential memory access; cache swapping
5. Virtual machine indirection
6. Boxing numbers as objects
7. Type checking at runtime
8. Unnecessary or nonsequential disk/network access
9. Wacky stuff

Compilation optimizes away most of #1, #2, and #4.

GPUs focus on #2 and #4 (by putting memory close to processing).

Hurdles, from smallest to largest

1. Unnecessary or repeated arithmetic
2. Arithmetic in separate instructions that could be in the same instruction (vectorization)
3. Transcendental functions or division
4. Unnecessary or nonsequential memory access; cache swapping
5. Virtual machine indirection
6. Boxing numbers as objects
7. Type checking at runtime
8. Unnecessary or nonsequential disk/network access
9. Wacky stuff

Compilation optimizes away most of #1, #2, and #4.

GPUs focus on #2 and #4 (by putting memory close to processing).

Python is guilty of #4, #5, #6, and #7 (Java only #4, #5, and half of #6).

Optimization is about trade-offs

We're here because we like the productivity Python gives us in exchange for #4, #5, #6, and #7.

Optimization is about trade-offs

We're here because we like the productivity Python gives us in exchange for #4, #5, #6, and #7.

Ideally, we'd like a library that makes Python code fast without modification.

- I don't know how much speedup I'll get until I apply it; but that costs effort.
- If I've applied it and I don't like it, I want to easily remove it.

Optimization is about trade-offs

We're here because we like the productivity Python gives us in exchange for #4, #5, #6, and #7.

Ideally, we'd like a library that makes Python code fast without modification.

- I don't know how much speedup I'll get until I apply it; but that costs effort.
- If I've applied it and I don't like it, I want to easily remove it.

If we had such a thing, though, when would we ever *not* use it?

Optimization is about trade-offs

We're here because we like the productivity Python gives us in exchange for #4, #5, #6, and #7.

Ideally, we'd like a library that makes Python code fast without modification.

- I don't know how much speedup I'll get until I apply it; but that costs effort.
- If I've applied it and I don't like it, I want to easily remove it.

If we had such a thing, though, when would we ever *not* use it?

Example: PyPy, a reimplementation of Python with just-in-time (JIT) compilation. If it works, we'd only use that. It doesn't yet work with all extension modules, though.

Horizontal and vertical scaling

Horizontal: split up task and distribute among parallel workers.

Vertical: use hardware more effectively by removing hurdles.

Horizontal and vertical scaling

Horizontal: split up task and distribute among parallel workers.

Oddly, this speedup is rarely proportional to the number of workers, even when work is independent, due to bookkeeping overhead and shipping data.

Vertical: use hardware more effectively by removing hurdles.

Horizontal and vertical scaling

Horizontal: split up task and distribute among parallel workers.

Oddly, this speedup is rarely proportional to the number of workers, even when work is independent, due to bookkeeping overhead and shipping data.

Vertical: use hardware more effectively by removing hurdles.

Plateaus as you get close to optimum. More effort yields diminishing returns.

Why cover Pandas in an afternoon about performance?

Pandas is about simplifying data analysis, and it does so by translating the array programming style from Numpy to domain concepts: timestamps, categorical data, relational data, etc.

Why cover Pandas in an afternoon about performance?

Pandas is about simplifying data analysis, and it does so by translating the array programming style from Numpy to domain concepts: timestamps, categorical data, relational data, etc.

It's like a spreadsheet that uses Numpy arrays instead of graphical cells.

Why cover Pandas in an afternoon about performance?

pandas

$y_i t = \beta' x_{it} + \mu_i + \epsilon_{it}$

Pandas is about simplifying data analysis, and it does so by translating the array programming style from Numpy to domain concepts: timestamps, categorical data, relational data, etc.

It's like a spreadsheet that uses Numpy arrays instead of graphical cells.

It's not as fast as Numpy or the other accelerators I'll show, but it benefits from the conciseness of the same Numpythonic mindset.

So without further ado...