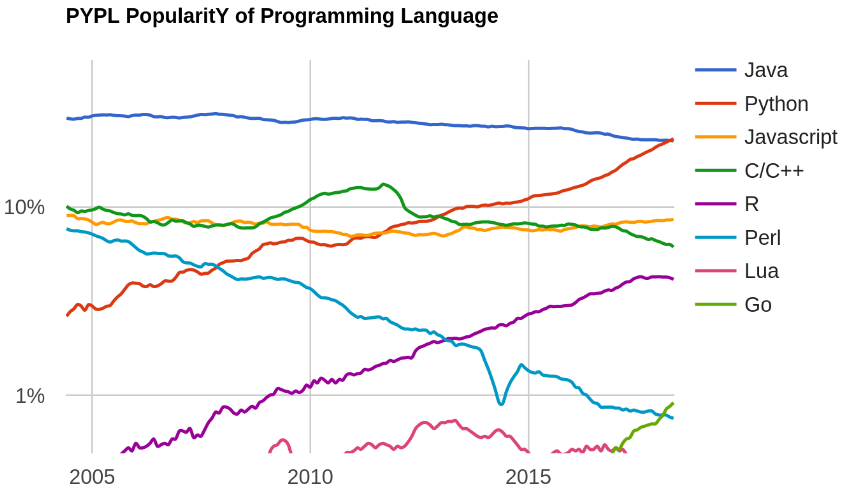
The Scientific Python Ecosystem

Jim Pivarski

Princeton University – DIANA-HEP

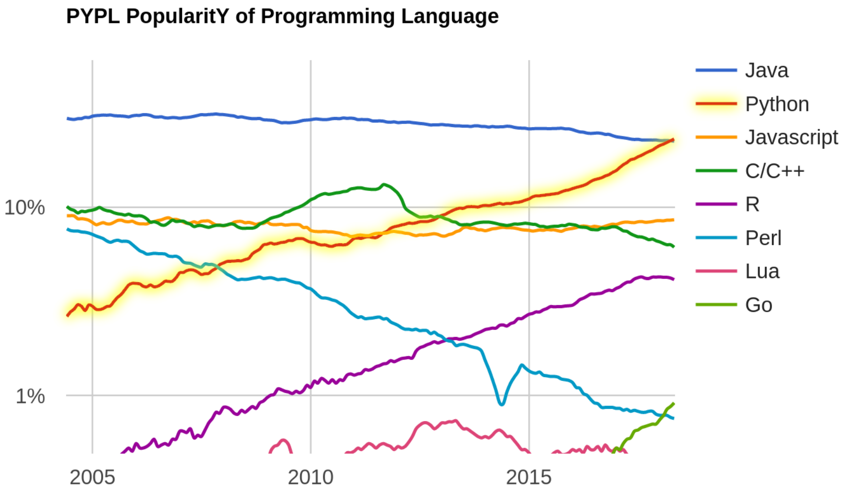
July 25, 2018

Why you can't ignore Python



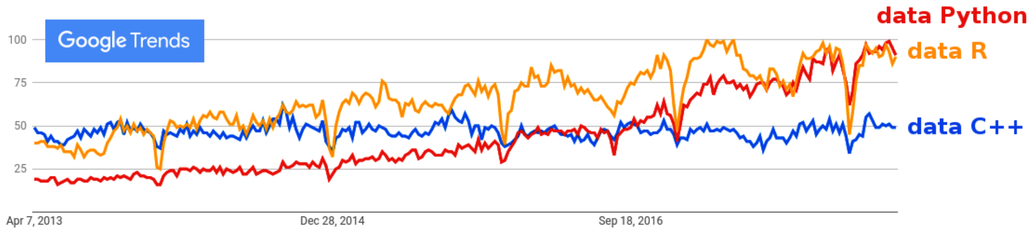
http://pypl.github.io/PYPL.html

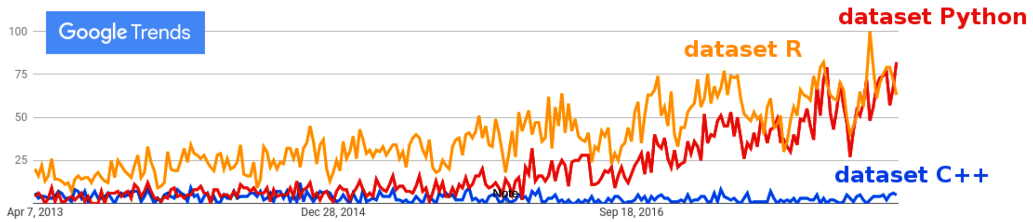
Why you can't ignore Python



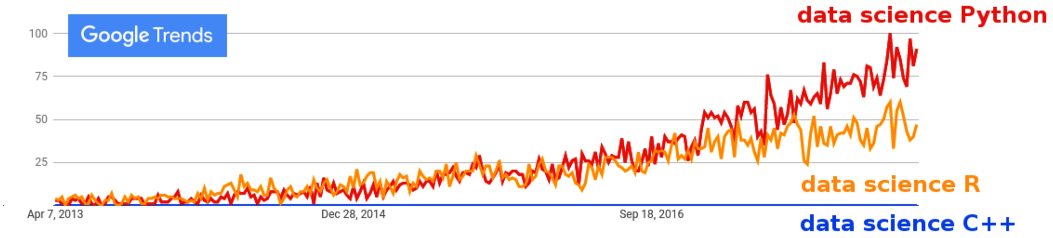
http://pypl.github.io/PYPL.html

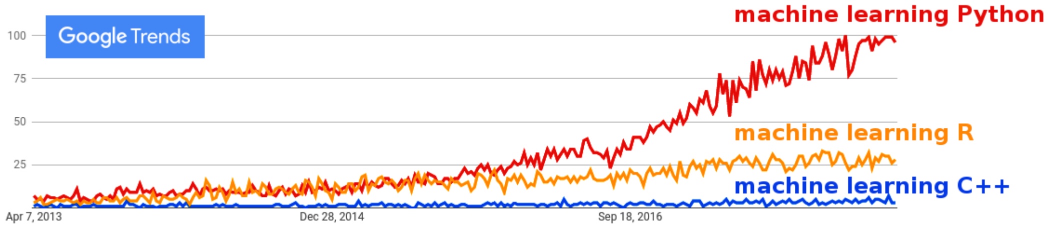
Why you can't ignore Python





Why you can't ignore Python





Why you can't ignore Python

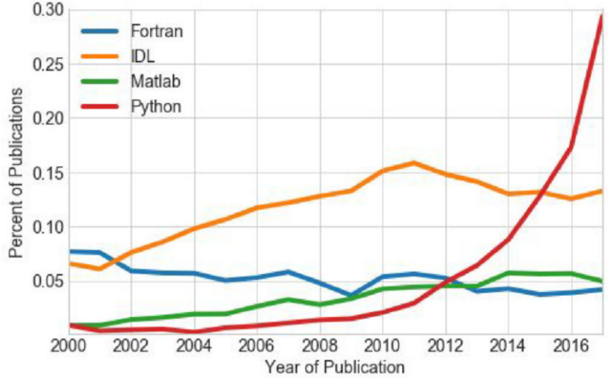
All of the deep learning libraries I could find either have a Python interface or are primarily/exclusively Python.

scikit learn PYTORCH Keras TensorFlow Caffe2 GLUON

Chainer Microsoft CNTK Lasagne ONNX Cesium dmlc XGBoost

Why you can't ignore Python

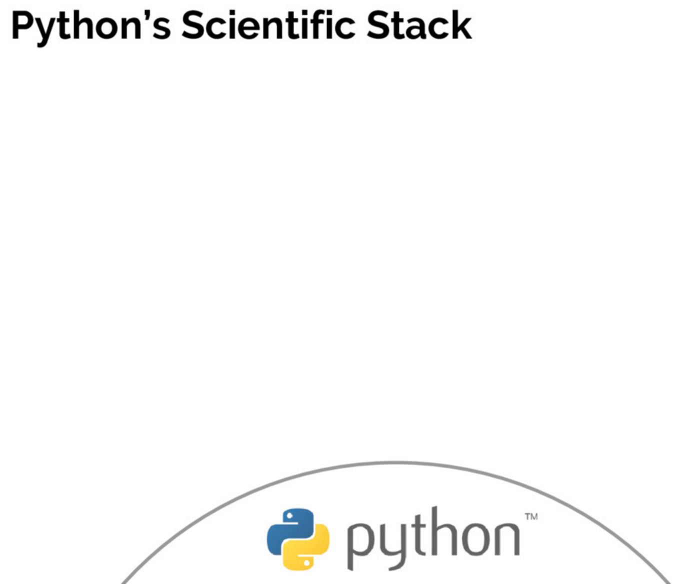
Mentions of Software in Astronomy Publications:

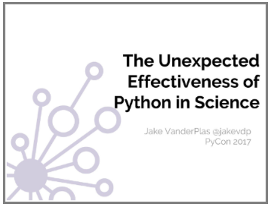


Compiled from NASA ADS (code).

Thanks to Juan Nunez-Iglesias, Thomas P. Robitaille, and Chris Beaumont.

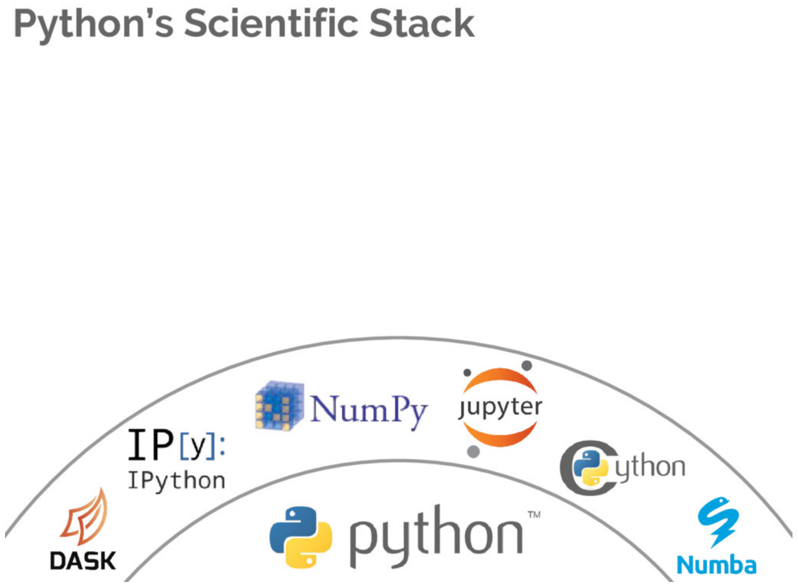
More significant: what has grown around Python

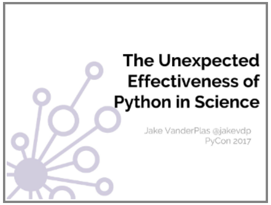

Python's Scientific Stack


The Unexpected Effectiveness of Python in Science

Jake VanderPlas @jakevdp
PyCon 2017

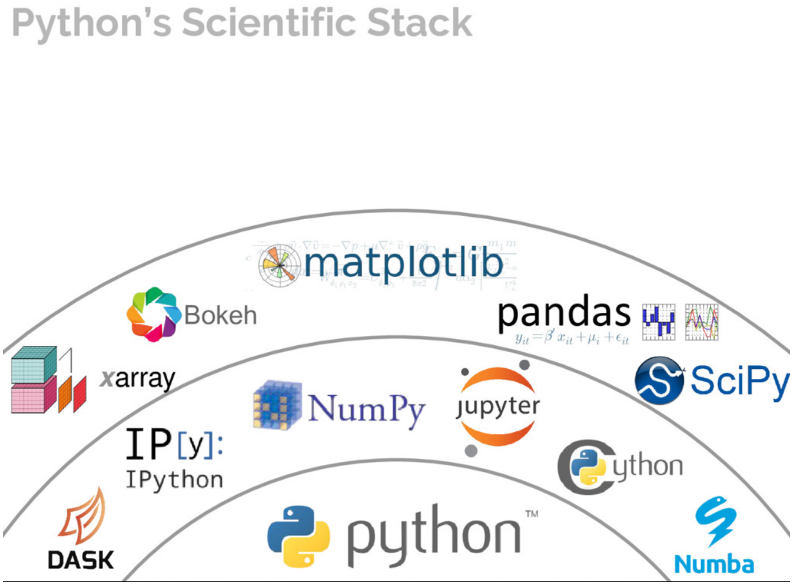
More significant: what has grown around Python

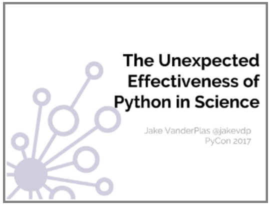



The Unexpected Effectiveness of Python in Science

Jake VanderPlas @jakevdp
PyCon 2017

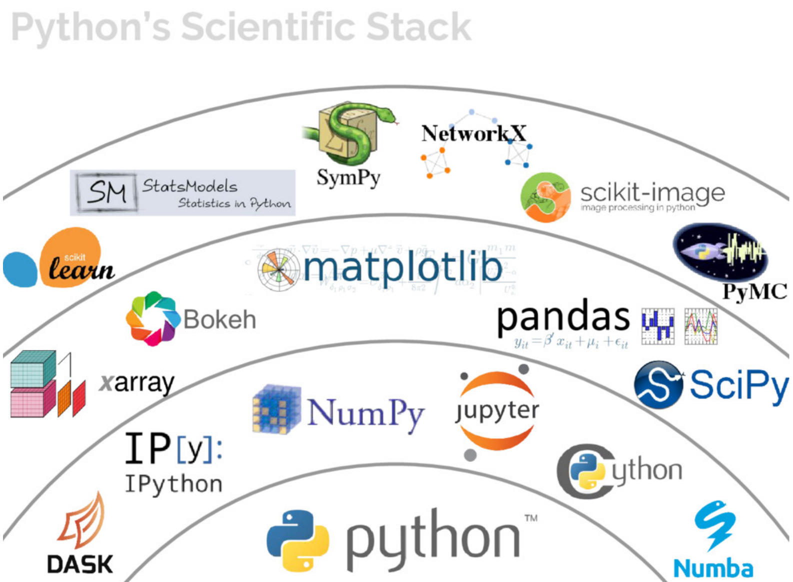
More significant: what has grown around Python

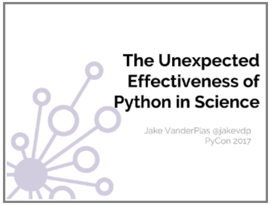



The Unexpected Effectiveness of Python in Science

Jake VanderPlas @jakevdp
PyCon 2017

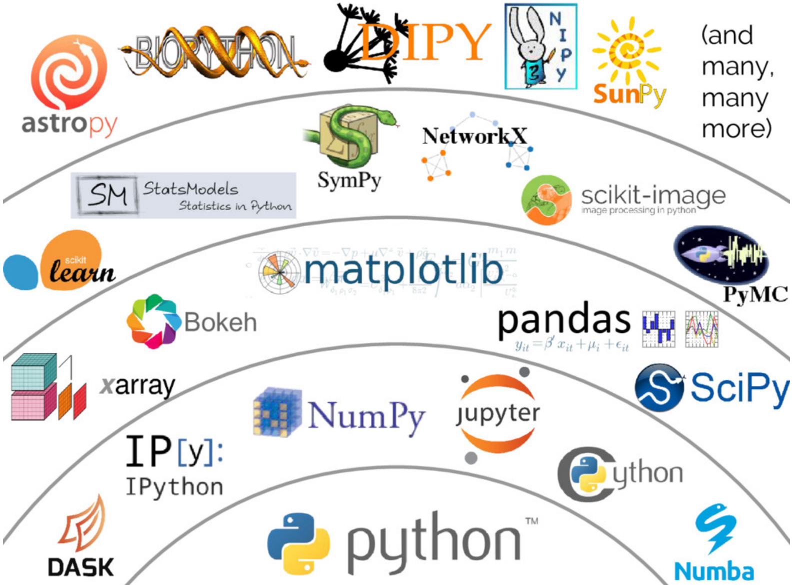
More significant: what has grown around Python

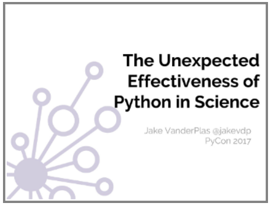



The Unexpected Effectiveness of Python in Science

Jake VanderPlas @jakevdp
PyCon 2017

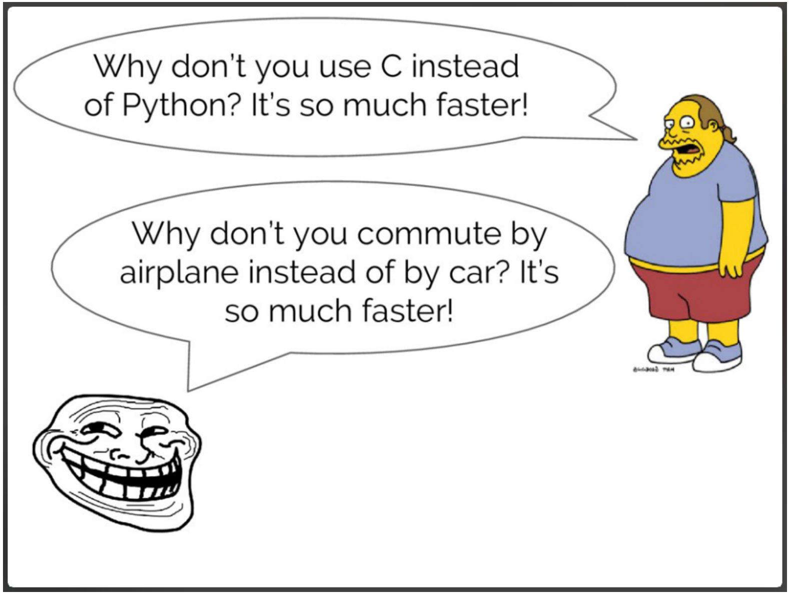
More significant: what has grown around Python

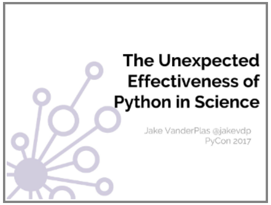



The Unexpected Effectiveness of Python in Science

Jake VanderPlas @jakevdp
PyCon 2017

Also stolen from that talk. . .


Why don't you use C instead of Python? It's so much faster!
Why don't you commute by airplane instead of by car? It's so much faster!


The Unexpected Effectiveness of Python in Science

Jake VanderPlas @jakevdp
PyCon 2017

“Python is a slow language”

Quibble: A *language* is neither fast nor slow. (But CPython is pretty slow.)

"Python is a slow language"

Quibble: A *language* is neither fast nor slow. (But CPython is pretty slow.)

Why Python is slow

“Python is a slow language”

Quibble: A *language* is neither fast nor slow. (But CPython is pretty slow.)

Why Python is slow

- Virtual machine between Python bytecode and the physical machine.

“Python is a slow language”

Quibble: A *language* is neither fast nor slow. (But CPython is pretty slow.)

Why Python is slow

- Virtual machine between Python bytecode and the physical machine.
- Type-checking during execution, *every time* in a tight for loop.

"Python is a slow language"

Quibble: A *language* is neither fast nor slow. (But CPython is pretty slow.)

Why Python is slow

- Virtual machine between Python bytecode and the physical machine.
- Type-checking during execution, *every time* in a tight for loop.
- Data are bloated and distributed in memory (cache line efficiency, pointer chasing).

“Python is a slow language”

Quibble: A *language* is neither fast nor slow. (But CPython is pretty slow.)

Why Python is slow

- Virtual machine between Python bytecode and the physical machine.
- Type-checking during execution, *every time* in a tight for loop.
- Data are bloated and distributed in memory (cache line efficiency, pointer chasing).
- Global Interpreter Lock (GIL): parallel processing is not parallel.

“Python is a slow language”

Quibble: A *language* is neither fast nor slow. (But CPython is pretty slow.)

Why Python is slow

- Virtual machine between Python bytecode and the physical machine.
- Type-checking during execution, *every time* in a tight for loop.
- Data are bloated and distributed in memory (cache line efficiency, pointer chasing).
- Global Interpreter Lock (GIL): parallel processing is not parallel.

Why Python is nevertheless good for science

“Python is a slow language”

Quibble: A *language* is neither fast nor slow. (But CPython is pretty slow.)

Why Python is slow

- Virtual machine between Python bytecode and the physical machine.
- Type-checking during execution, *every time* in a tight for loop.
- Data are bloated and distributed in memory (cache line efficiency, pointer chasing).
- Global Interpreter Lock (GIL): parallel processing is not parallel.

Why Python is nevertheless good for science

- Python usually just *drives* compiled code (good C, C++, Fortran bindings).

“Python is a slow language”

Quibble: A *language* is neither fast nor slow. (But CPython is pretty slow.)

Why Python is slow

- Virtual machine between Python bytecode and the physical machine.
- Type-checking during execution, *every time* in a tight for loop.
- Data are bloated and distributed in memory (cache line efficiency, pointer chasing).
- Global Interpreter Lock (GIL): parallel processing is not parallel.

Why Python is nevertheless good for science

- Python usually just *drives* compiled code (good C, C++, Fortran bindings).
- Many *standard* precompiled libraries, which also release the GIL.

“Python is a slow language”

Quibble: A *language* is neither fast nor slow. (But CPython is pretty slow.)

Why Python is slow

- Virtual machine between Python bytecode and the physical machine.
- Type-checking during execution, *every time* in a tight for loop.
- Data are bloated and distributed in memory (cache line efficiency, pointer chasing).
- Global Interpreter Lock (GIL): parallel processing is not parallel.

Why Python is nevertheless good for science

- Python usually just *drives* compiled code (good C, C++, Fortran bindings).
- Many *standard* precompiled libraries, which also release the GIL.
- **You start thinking in terms of “slow control” and “fast math.”**

“Python is a slow language”

Quibble: A *language* is neither fast nor slow. (But CPython is pretty slow.)

Why Python is slow

- Virtual machine between Python bytecode and the physical machine.
- Type-checking during execution, *every time* in a tight for loop.
- Data are bloated and distributed in memory (cache line efficiency, pointer chasing).
- Global Interpreter Lock (GIL): parallel processing is not parallel.

Why Python is nevertheless good for science

- Python usually just *drives* compiled code (good C, C++, Fortran bindings).
- Many *standard* precompiled libraries, which also release the GIL.
- **You start thinking in terms of “slow control” and “fast math.”**
- **Computational performance is not the same as user productivity!**

Numpy was key to Python developing a scientific ecosystem

1994	**Python** 1.0 released.
1995	First array package: **Numeric** (a.k.a. Numerical, Numerical Python, NumPy).
2001	Diverse scientific codebases merged into **SciPy**.
2003	**Matplotlib**
2003	Numeric was limited; **numarray** appeared as a competitor with more features (memory-mapped files, alignment, record arrays).
2005	Two packages were incompatible; could not integrate numarray-based code into SciPy. Travis Oliphant merged the codebases as **Numpy**.
2008	**Pandas**
2010	**Scikit-Learn**
2011	**AstroPy**
2012	**Anaconda**
2014	**Jupyter**
2015	**Keras**

Numpy was key to Python developing a scientific ecosystem

1994	**Python** 1.0 released.
1995	First array package: **Numeric** (a.k.a. Numerical, Numerical Python, NumPy).
2001	Diverse scientific codebases merged into **SciPy**.
2003	**Matplotlib**
2003	Numeric was limited; **numarray** appeared as a competitor with more features (memory-mapped files, alignment, record arrays).
2005	Two packages were incompatible; could not integrate numarray-based code into SciPy. Travis Oliphant merged the codebases as **Numpy**.
2008	**Pandas**
2010	**Scikit-Learn**
2011	**AstroPy**
2012	**Anaconda**
2014	**Jupyter**
2015	**Keras**

The scientific Python ecosystem could have failed before it started if the Numeric-numarray split hadn't been resolved!

The Numpythonic mindset

Although you can write Python `for` loops over Numpy arrays, you don't reap the benefit unless you express your calculation in Numpy ufuncs (universal functions).

```
pz = numpy.empty(len(pt))
for i in range(len(pt)):
    pz[i] = pt[i]*numpy.sinh(eta[i])
```

vs

```
pz = pt * numpy.sinh(eta)
```

$\mathcal{O}(N)$ Python bytecode instructions, type-checks, interpreter locks.

$\mathcal{O}(1)$ Python bytecode instructions, type-checks, interpreter locks.

$\mathcal{O}(N)$ statically typed, probably vectorized native bytecode operations on contiguous memory.

The Numpythonic mindset

Although you can write Python `for` loops over Numpy arrays, you don't reap the benefit unless you express your calculation in Numpy ufuncs (universal functions).

```python
pz = numpy.empty(len(pt))
for i in range(len(pt)):
    pz[i] = pt[i]*numpy.sinh(eta[i])
```

vs

```python
pz = pt * numpy.sinh(eta)
```

$\mathcal{O}(N)$ Python bytecode instructions, type-checks, interpreter locks.

$\mathcal{O}(1)$ Python bytecode instructions, type-checks, interpreter locks.

$\mathcal{O}(N)$ statically typed, probably vectorized native bytecode operations on contiguous memory.

In other words, a Single (Python) Instruction on Multiple Data.

The Numpythonic mindset

Although you can write Python `for` loops over Numpy arrays, you don't reap the benefit unless you express your calculation in Numpy ufuncs (universal functions).

```
pz = numpy.empty(len(pt))
for i in range(len(pt)):
    pz[i] = pt[i]*numpy.sinh(eta[i])
```

vs

```
pz = pt * numpy.sinh(eta)
```

$\mathcal{O}(N)$ Python bytecode instructions, type-checks, interpreter locks.

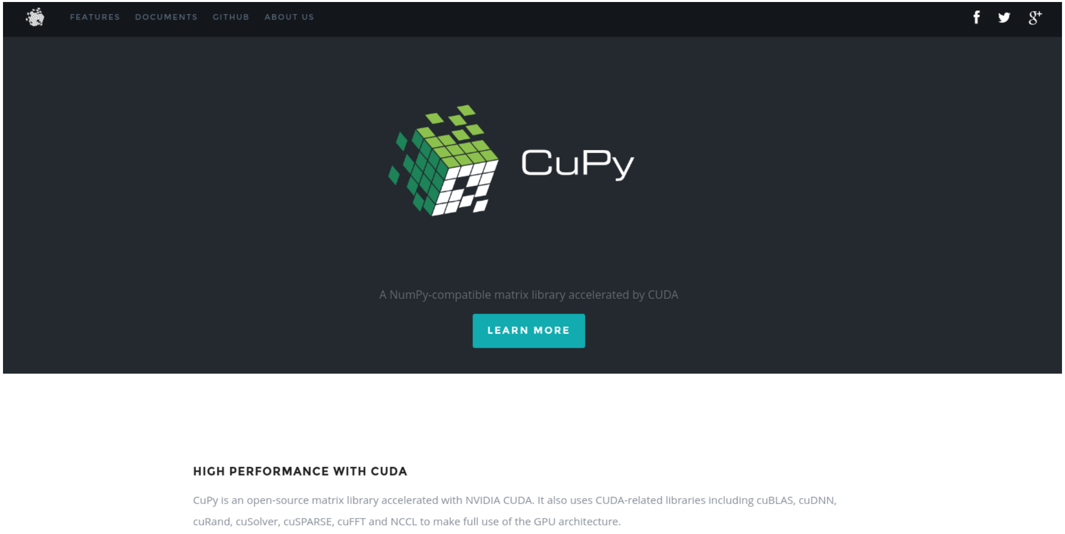
$\mathcal{O}(1)$ Python bytecode instructions, type-checks, interpreter locks.

$\mathcal{O}(N)$ statically typed, probably vectorized native bytecode operations on contiguous memory.

In other words, a Single (Python) Instruction on Multiple Data.

The same code reorganization that speeds up Python with Numpy would speed up C++ with CUDA or SIMD.

The Numpythonic mindset: Numpy API in CUDA

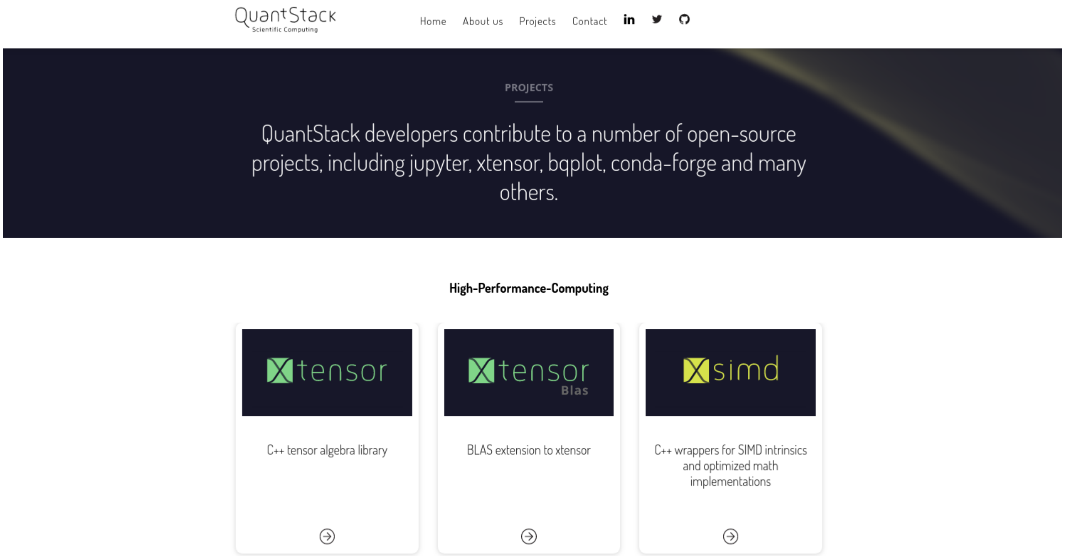


A NumPy-compatible matrix library accelerated by CUDA

HIGH PERFORMANCE WITH CUDA

CuPy is an open-source matrix library accelerated with NVIDIA CUDA. It also uses CUDA-related libraries including cuBLAS, cuDNN, cuRand, cuSolver, cuSPARSE, cuFFT and NCCL to make full use of the GPU architecture.

The Numpythonic mindset: Numpy API in C++


QuantStack
Scientific Computing

PROJECTS

QuantStack developers contribute to a number of open-source projects, including jupyter, xtensor, bqplot, conda-forge and many others.

High-Performance-Computing

C++ tensor algebra library

BLAS extension to xtensor

C++ wrappers for SIMD intrinsics and optimized math implementations

Switch to the notebook now or install packages.

```
# install Miniconda...
conda install pip
conda install numpy
conda install pandas

# optional...
conda install numba
conda install cython
conda install scipy

# if you have a GPU and CUDA...
conda install -c lukepfister pycuda
```