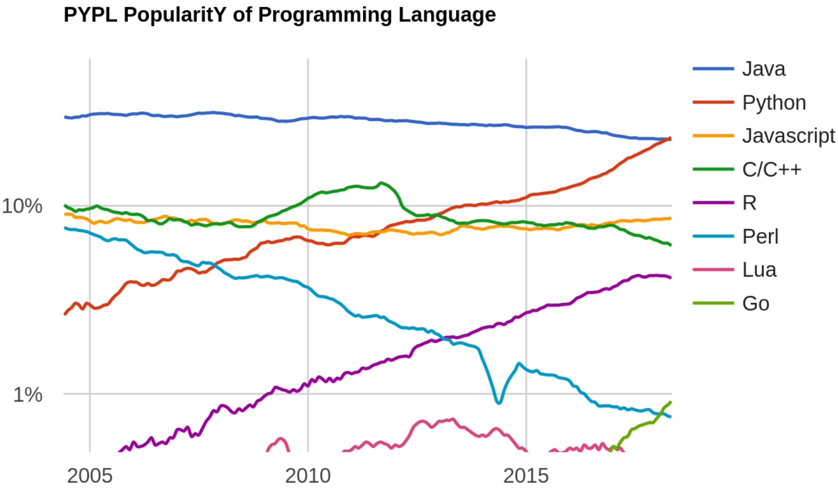
Python/Numpy for High-Performance Numerical Processing

Jim Pivarski

Princeton University

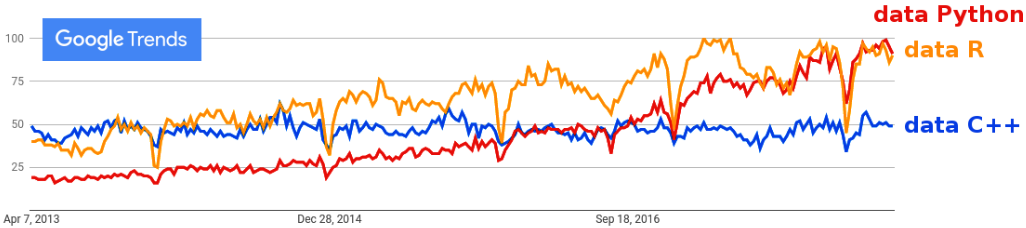
November 15, 2018

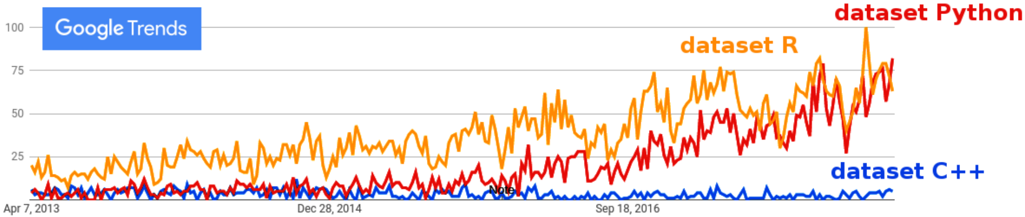
Why Python?



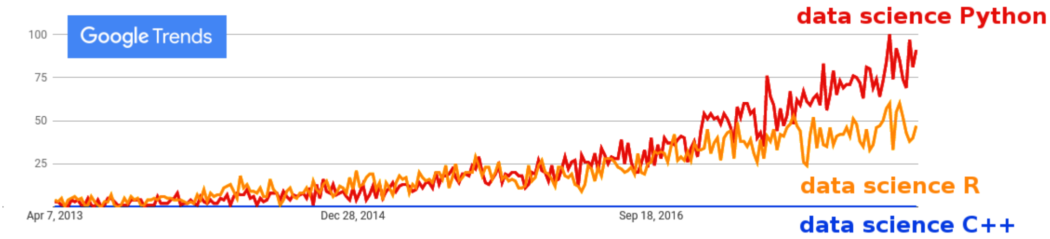
http://pypl.github.io/PYPL.html

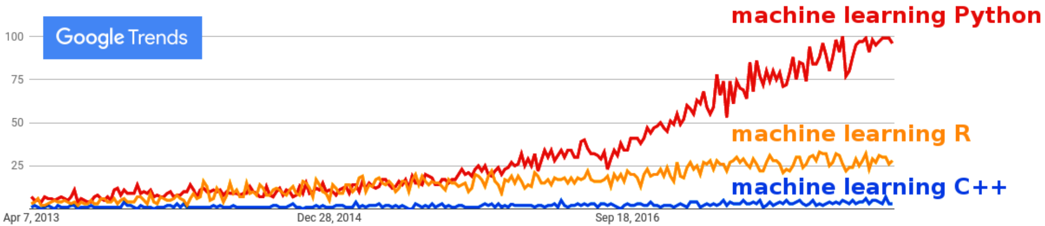
Why Python in science?



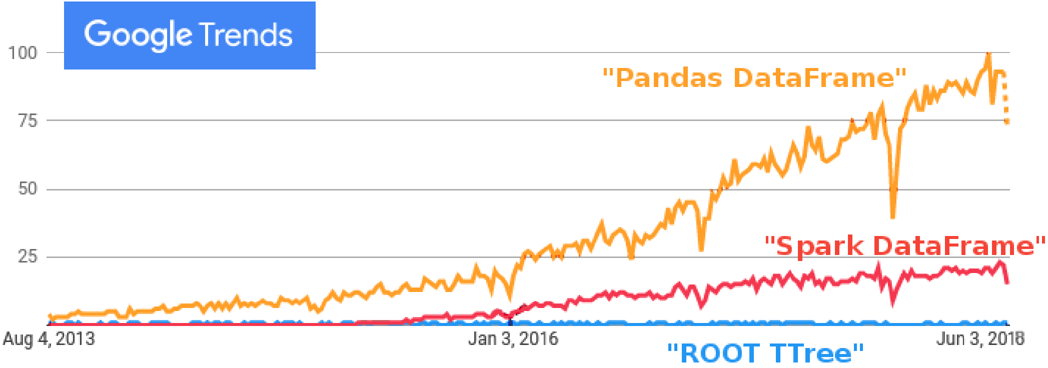


Why Python in science?





Why Python in science?



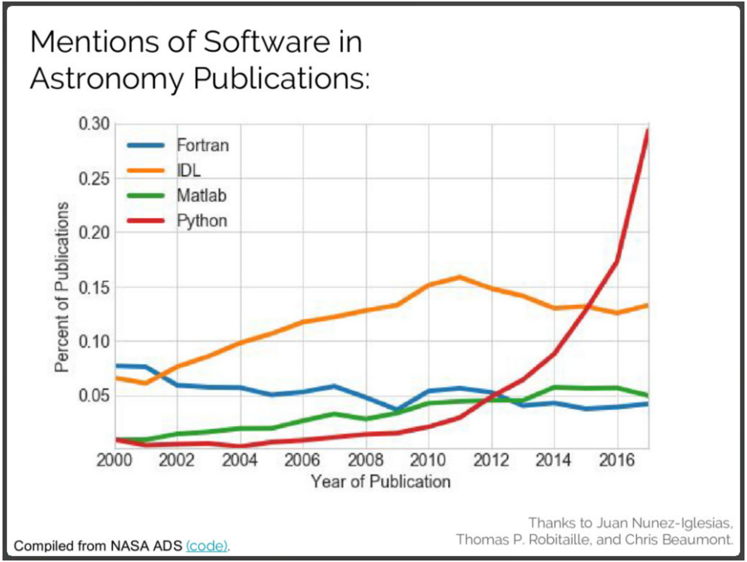
Why Python in science?

All of the machine 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 Python in science?

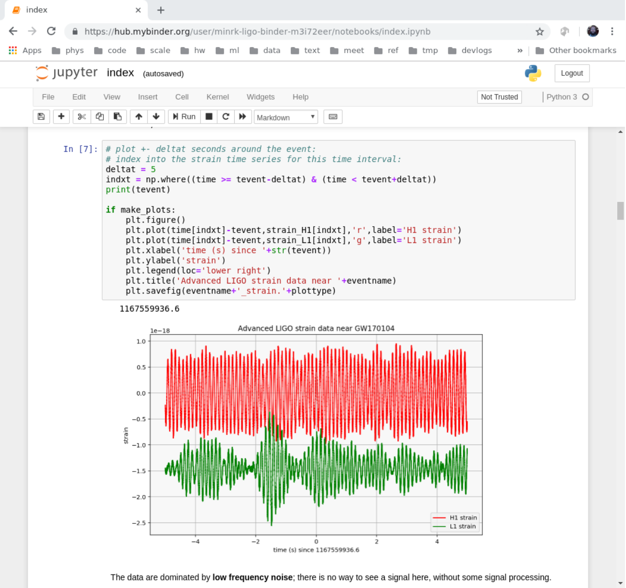


Mentions of Software in Astronomy Publications:

Compiled from NASA ADS (code).

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

Why Python in science?


index
https://hub.mybinder.org/user/minrk-ligo-binder-m3i72eer/notebooks/index.ipynb
Apps phys code scale hw ml data text meet ref tmp devlogs » Other bookmarks
jupyter index (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 ○
Run
Markdown
In [7]:

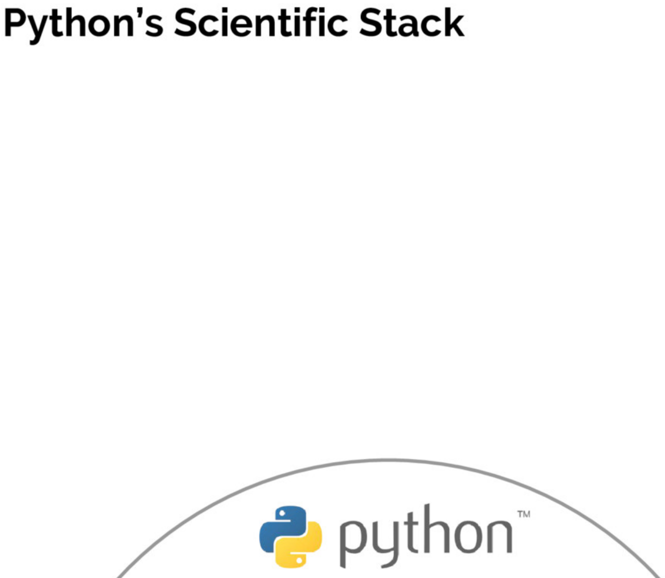
```
# plot +- deltat seconds around the event:
# index into the strain time series for this time interval:
deltat = 5
indxt = np.where((time >= tevent-deltat) & (time < tevent+deltat))
print(tevent)

if make_plots:
    plt.figure()
    plt.plot(time[indxt]-tevent,strain_H1[indxt],'r',label='H1 strain')
    plt.plot(time[indxt]-tevent,strain_L1[indxt],'g',label='L1 strain')
    plt.xlabel('time (s) since '+str(tevent))
    plt.ylabel('strain')
    plt.legend(loc='lower right')
    plt.title('Advanced LIGO strain data near '+eventname)
    plt.savefig(eventname+'_strain.'+plottype)
```

```
1167559936.6
```

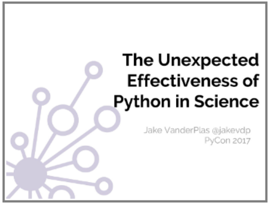
Advanced LIGO strain data near GW170104
1e−18
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
strain
−4
−2
0
2
4
time (s) since 1167559936.6
H1 strain
L1 strain

The data are dominated by **low frequency noise**; there is no way to see a signal here, without some signal processing.

Stealing from Jake VanderPlas's *Unexpected Effectiveness* talk



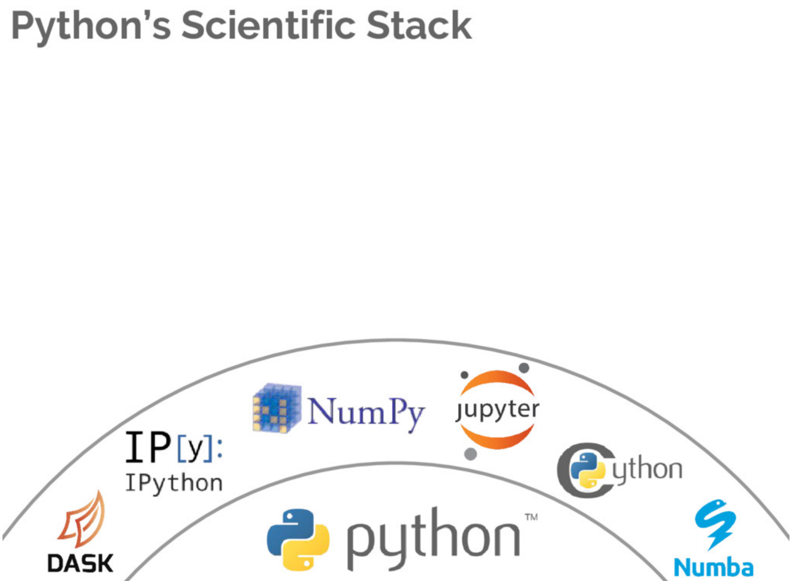
Python's Scientific Stack



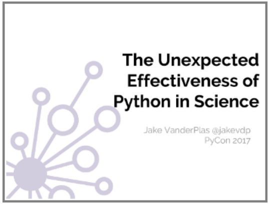
The Unexpected Effectiveness of Python in Science

Jake VanderPlas @jakevdp
PyCon 2017

Stealing from Jake VanderPlas's *Unexpected Effectiveness* talk



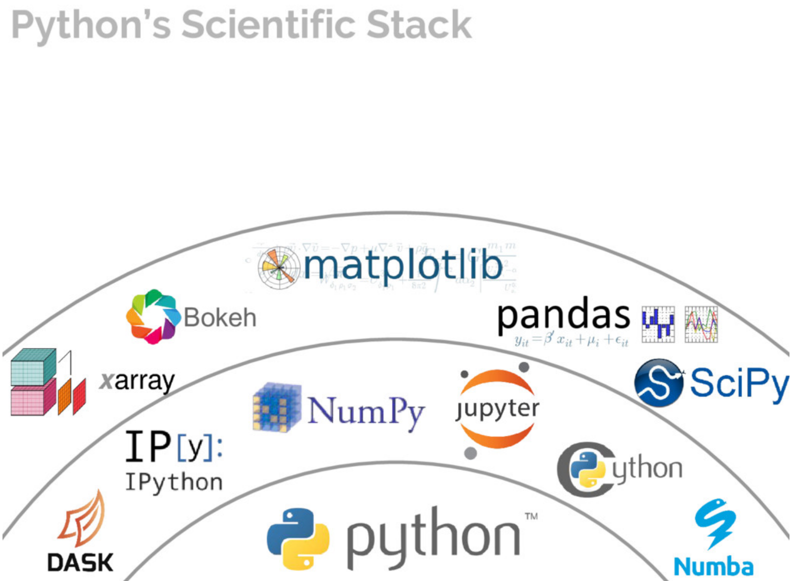
Python's Scientific Stack



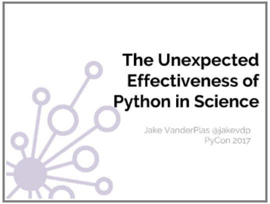
The Unexpected Effectiveness of Python in Science

Jake VanderPlas @jakevdp
PyCon 2017

Stealing from Jake VanderPlas's *Unexpected Effectiveness* talk



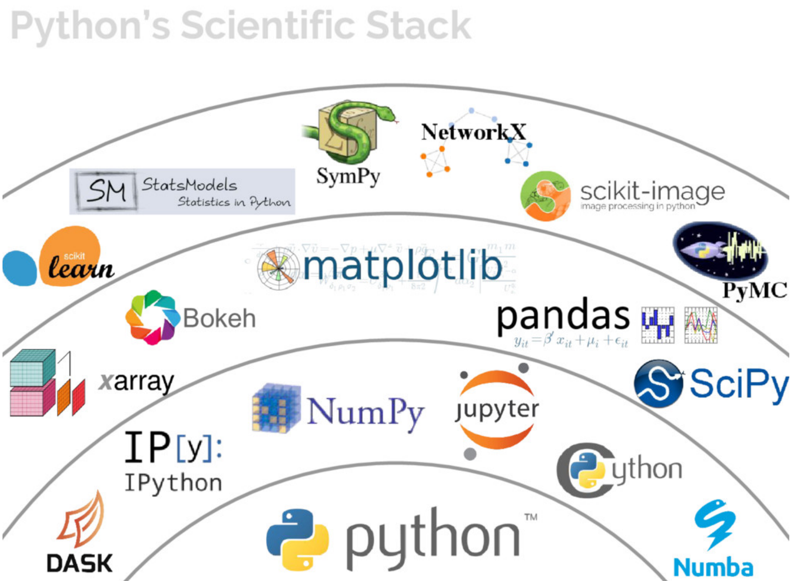
Python's Scientific Stack



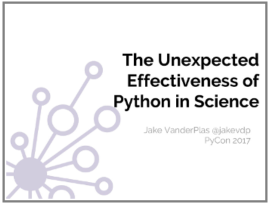
The Unexpected Effectiveness of Python in Science

Jake VanderPlas @jakevdp
PyCon 2017

Stealing from Jake VanderPlas's *Unexpected Effectiveness* talk



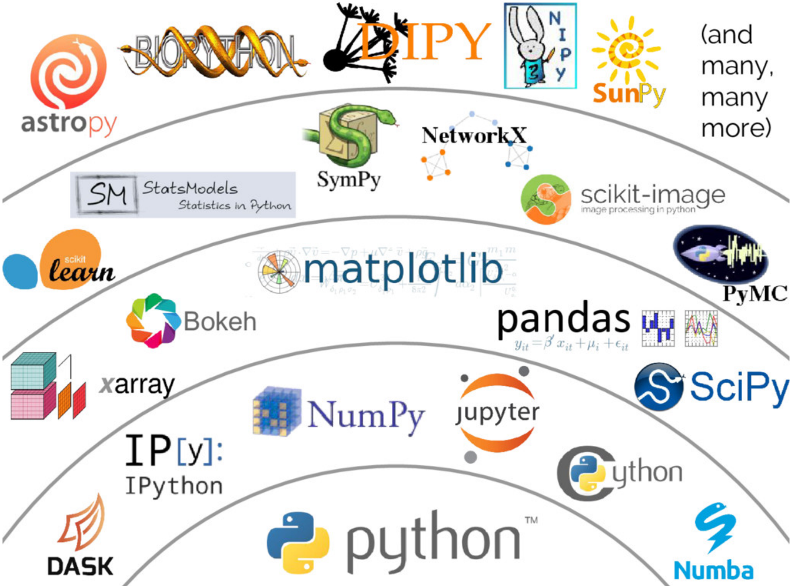
Python's Scientific Stack

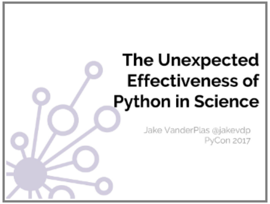


The Unexpected Effectiveness of Python in Science

Jake VanderPlas @jakevdp
PyCon 2017

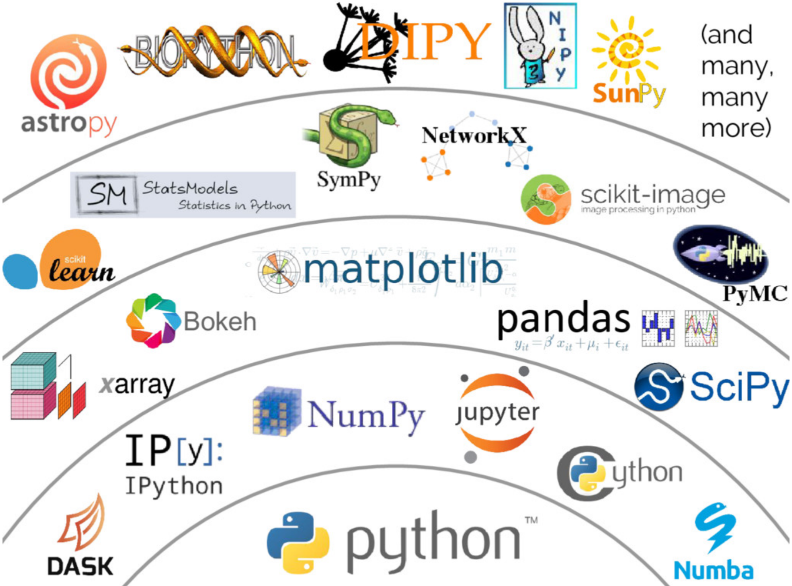
Stealing from Jake VanderPlas's *Unexpected Effectiveness* talk

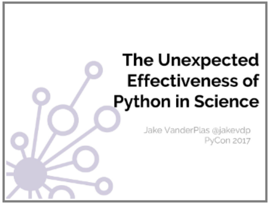



The Unexpected Effectiveness of Python in Science

Jake VanderPlas @jakevdp
PyCon 2017

Stealing from Jake VanderPlas's *Unexpected Effectiveness* talk



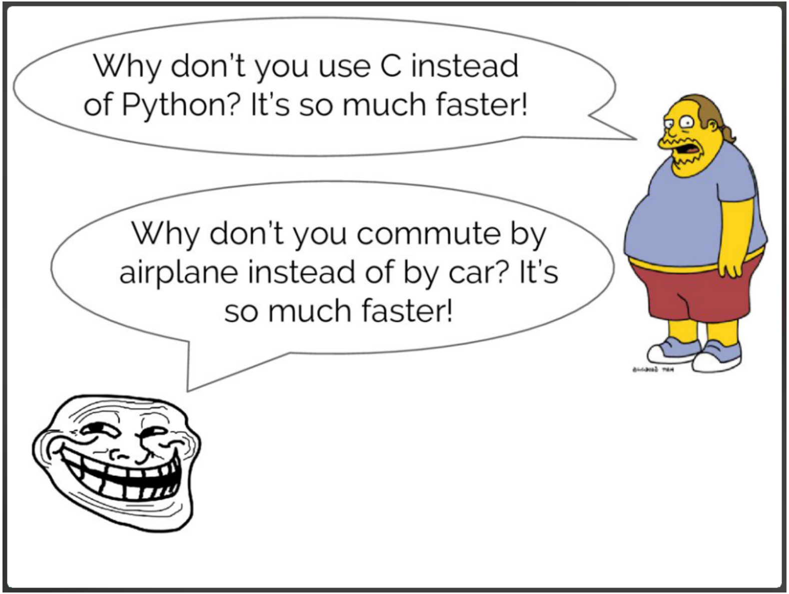


The Unexpected Effectiveness of Python in Science

Jake VanderPlas @jakevdp
PyCon 2017

If you're used to writing your own code, searching for tools is eye-opening: you learn what's unique about what you do and what isn't.

Stealing again from Jake VanderPlas


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!

Why not indeed?

In science, we often have to scale up analyses to large datasets.

Why not indeed?

In science, we often have to scale up analyses to large datasets.

10% faster doesn't mean much, but the difference between "five minutes" and "overnight" is life-changing.

Why not indeed?

In science, we often have to scale up analyses to large datasets.

10% faster doesn't mean much, but the difference between "five minutes" and "overnight" is life-changing.

That's the scale we're talking about between C and Python.

Why not indeed?

In science, we often have to scale up analyses to large datasets.

10% faster doesn't mean much, but the difference between
“five minutes” and “overnight” is life-changing.

That's the scale we're talking about between C and Python.

But we also need the interactivity of a dynamic language to *develop* the analysis.
(“If we knew what we were doing, it wouldn't be called research.”)

Metaphor time!

Drive to the airport by car, then take a plane.

Small-scale *project organization* in Python, ignoring performance entirely.

Run over *big data* in compiled code, tuning performance until it no longer matters.

Python is a good glue language: my thesis workflow in 2006

Earlier steps in Mathematica, Perl, GnuPlot, Emacs Lisp, PAW, and ROOT

Fit function

FORTRAN

```
 DOUBLE PRECISION FUNCTION GBWKF(RM,GAM,WSPREAD,HC)
        IMPLICIT NONE
c  implicit real (A-H,O-Z), integer (I-N)

C   Observed shape of resonance peak, starting with Breit-Wigner
C   Normalized per unit W, MeV^-1
C   F_KF(x,s) convoluted with unit-area Breit-Wigner, and with beam
resolution
C    unit-area Gaussian
C   If GAM<1 keV, the Breit-Wigner is replaced with a delta function
C   BW(w) = (GAM/2*pi)/[(w-M)^2+GAM^2/4] (unit integral over w)
C   Gaussian(w) = exp{-.5*[(w-M)^2/WSPREAD^2]}/(2*pi*WSPREAD) (unit int...)

        INTEGER   MKBIN, MTBIN, JTBIN
        REAL*8    ROOT2, PI, RTWOPI, RTWOBPI, PMAX, TINY

 REAL*8    FKFT(MKBIN)               ! F_KF(x,s) (Kuraev-Fadin eq.28)
 REAL*8    HA(MKBIN)                   ! WA-RM of centroid WA bin
 REAL*8    DWA(MKBIN)                  ! bin width in WA-RM
 REAL*8    HT(0:MTBIN)
 REAL*8    DWT(0:MTBIN)                ! WT-RM bin center
 REAL*8    BWKF(-MTBIN:MTBIN)          ! result of BW-F_KF convolution
...
 DO 11 ITBIN=-MTBIN,MTBIN            ! sum over WT's
   H=HT(IABS(ITBIN))                 ! WT-RM
   IF(ITBIN.LT.0)H=-H
   SIGS=(H-HC)/(ROOT2*WSPREAD)       ! sqrt of exponent
   IF((ITBIN.EQ.MTBIN).AND.(SIGS.LT.2.))GO TO 12 ! running out of bins?
   IF(SIGS.GT.3.)GO TO 13            ! Gaussian getting negligible?
   P=SIGS**2                         ! exponent
   IF(P.LT.PMAX)THEN
     IF(DWT(IABS(ITBIN)).GT..2*WSPREAD)GO TO 12  ! binning too coarse?
            SUM=SUM+EXP(-P)*BWKF(ITBIN)*DWT(IABS(ITBIN))
   ENDIF
 11 CONTINUE
 13 GBWKF=SUM/(RTWOPI*WSPREAD)          ! convolution
C        1/(sqrt(2*pi)*WSPREAD) normalizes Gaussian
 RETURN
 END
```

SEAL-MINUIT

C++

```
#include "Minuit/MnMinos.h"
#include "Minuit/FunctionMinimum.h"
#include "Minuit/FCNBase.h"
#include "Minuit/MnFunctionCross.h"
#include "Minuit/MnCross.h"
#include "Minuit/MinosError.h"

std::pair<double,double> MnMinos::operator()(unsigned int par, unsigned
int maxcalls) const {
  MinosError mnerr = minos(par, maxcalls);
  return mnerr();
}

double MnMinos::lower(unsigned int par, unsigned int maxcalls) const {

  MnUserParameterState upar = theMinimum.userState();
  double err = theMinimum.userState().error(par);

  MnCross aopt = loval(par, maxcalls);

  double lower = aopt.isValid() ? -1.*err*(1.+ aopt.value()) :
(aopt.atLimit() ? upar.parameter(par).lowerLimit() : upar.value(par));

  return lower;
}
```

PyMinuit

C API

```
// g++ minuit.cpp -I/nfs/cleo3/Offline/rel/current/other_sources/pyth
include/python2.4/ -I/cdat/daf9/mccann/software/src/minuit/Minuit-1_5_2 /
cdat/daf9/mccann/software/src/minuit/Minuit-1_5_2/src/*.o -shared -o
_minuit.so
// g++ minuit.cpp -I/usr/include/python2.3 -I/root/src/Minuit-1_5_2/ /
root/src/Minuit-1_5_2/src/*.o -shared -o _minuit.so

#include <Python.h>
#include "Minuit/MnUserParameters.h"
#include "Minuit/MnMigrad.h"
#include "Minuit/MnMinimize.h"
...
PyObject* dominos(PyObject *self, PyObject *args)
{
   // parameter list:
   PyObject *p_fcn;                      // objective function p_fcn
   int npar;                             // number of parameters in p_fcn
   double up;                            // 1 for chi^2, 0.5 for loglike
   PyObject *p_min;                      // the minimum you previously found

   if (!PyArg_ParseTuple(args, "OidOiiOOiOO", &p_fcn, &npar, &up, &p_min,
&maxcalls, &strategy, &p_dolower, &p_doupper, &parnum, &p_grad,
&p_checkgrad)) {
      PyErr_SetString(PyExc_TypeError, "calling format must be: FCN(f),
npar(i), up(d), minimum(FunctionMinimum), maxcalls(i or 0), strategy(i),
dolower(b), doupper(b), parnum(i), gradient(f or None), checkgrad(b or
None)");
      return NULL;
   }
```

Fitting script

Python

```
# get_runs has been given a thorough look-over: it is correct (7 Oct
# 2005) (get_runs contains all corrections, from numbers of events to
# real, live cross-section.)

from minuit import *
execfile("/home/mccann/antithesis/utilities.py")
import gbwkf
import gbwkftau
...
def dofitgauss(h):
  def gauss(m, s, x): return exp(-(x-m)**2/2./s**2)/sqrt(2.*pi)/s
  def fitgauss(m,s):
    c = 0.
    for x in h.data:
      c += -log(gauss(m, s, x))
    return c
  m = Minuit(fitgauss, start=[0., 1.], up=0.5)
  m.migrad()
  m.minos([0,1])
  err0 = (m.minos_errors[0][1] - m.minos_errors[0][0])/2.
  err1 = (m.minos_errors[1][1] - m.minos_errors[1][0])/2.
  return m.values[0], err0, m.values[1], err1, lambda x:
0.1*extraarea*len(h.data)*gauss(m.values[0], m.values[1], x)
```

(pickle)

GNU plotutils

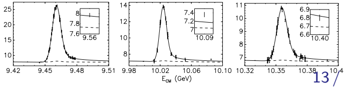
biggles

Numeric

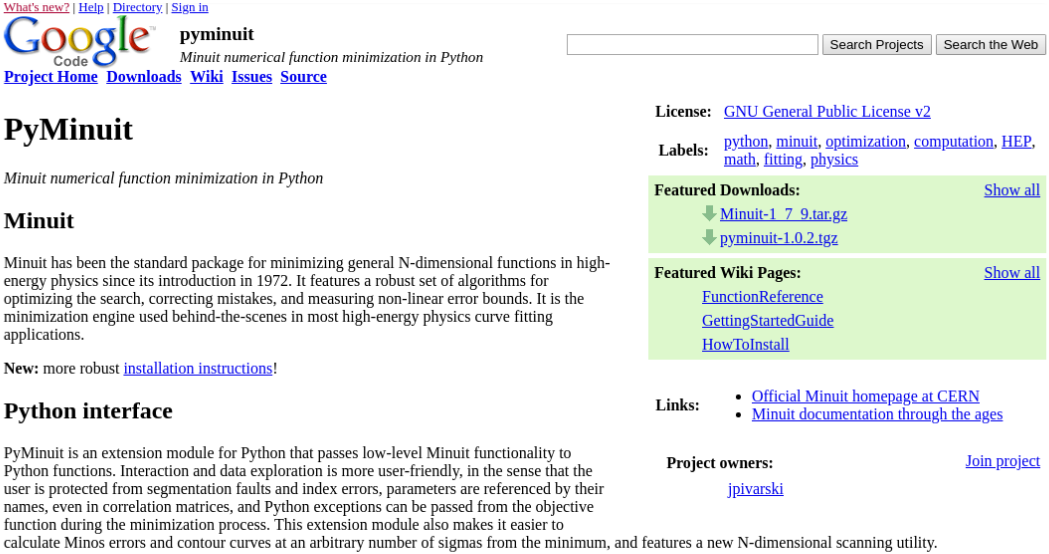
Plotting script

Python

```
from math import *
import biggles, Numeric, cPickle as pickle
import gbwkf
import gbwkftau

allthat = pickle.load(file("/home/mccann/antithesis/novemberdata.p"))
u1runs = allthat["u1runs"]
u2runs = allthat["u2runs"]
u3runs = allthat["u3runs"]
...
q = biggles.FramedPlot()
adddata(q, [None], u2data["high"], 0.)
addfunc(q, thefunc, 10080., 10090.)
addfunc(q, thefunc_bkgnd, 10080., 10090., linetype="dashed")
```



Which got me involved in open source (PyMinuit is now “iminuit”)



pyminuit

Minuit numerical function minimization in Python

PyMinuit

Minuit numerical function minimization in Python

Minuit

Minuit has been the standard package for minimizing general N-dimensional functions in high-energy physics since its introduction in 1972. It features a robust set of algorithms for optimizing the search, correcting mistakes, and measuring non-linear error bounds. It is the minimization engine used behind-the-scenes in most high-energy physics curve fitting applications.

New: more robust installation instructions!

Python interface

PyMinuit is an extension module for Python that passes low-level Minuit functionality to Python functions. Interaction and data exploration is more user-friendly, in the sense that the user is protected from segmentation faults and index errors, parameters are referenced by their names, even in correlation matrices, and Python exceptions can be passed from the objective function during the minimization process. This extension module also makes it easier to calculate Minos errors and contour curves at an arbitrary number of sigmas from the minimum, and features a new N-dimensional scanning utility.

License: GNU General Public License v2

Labels: python, minuit, optimization, computation, HEP, math, fitting, physics

Featured Downloads: Show all

- Minuit-1_7_9.tar.gz
- pyminuit-1.0.2.tgz

Featured Wiki Pages: Show all

- FunctionReference
- GettingStartedGuide
- HowToInstall

Links:

- Official Minuit homepage at CERN
- Minuit documentation through the ages

Project owners: Join project

jpivarski

The key to ecosystem development was a common array library

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 key to ecosystem development was a common array library

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!

Numpy is high-level, array-at-a-time math

```
>>> import numpy
>>> a = numpy.arange(12)
>>> a
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
>>> a.shape = (3, 4)
>>> a
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
>>> a.sum(axis=0)
array([12, 15, 18, 21])
>>> a.min(axis=1)
array([0, 4, 8])
>>> a**2
array([[  0,   1,   4,   9],
       [ 16,  25,  36,  49],
       [ 64,  81, 100, 121]])
>>> numpy.sqrt(a)
array([[0.        , 1.        , 1.41421356, 1.73205081],
       [2.        , 2.23606798, 2.44948974, 2.64575131],
       [2.82842712, 3.        , 3.16227766, 3.31662479]])
```

NumPy

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 universal functions (ufuncs).

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

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

vs

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

$\mathcal{O}(1)$ 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 universal functions (ufuncs).

```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.

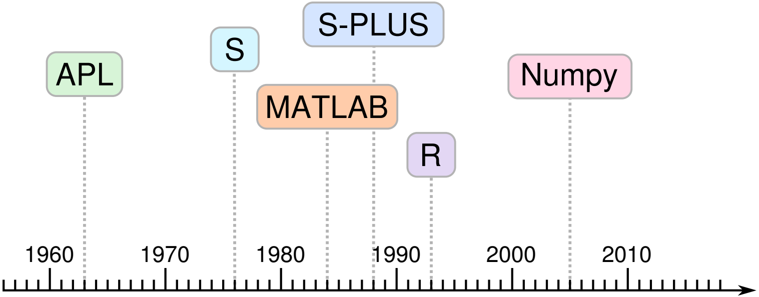
$\mathcal{O}(N)$ statically typed, probably vectorized native bytecode operations on contiguous memory.

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

Conceptually similar to SIMD, the program flow of GPUs.

This is not new

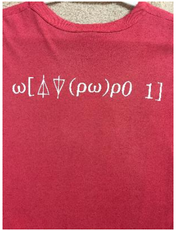
APL, “A Programming Language” introduced the idea of single commands having sweeping effects across large arrays.



All members of the APL family are intended for interactive data analysis.
Numpy, however, is a library in a general-purpose language, not a language in itself.

APL

APL pioneered conciseness;
discovered the mistake of being too concise.



Conway's Game of Life was one line of code:

$$\texttt{life} \leftarrow \{\uparrow 1\ \ \omega \vee . \wedge 3\ \ 4 = +/, {}^{-}1\ \ 0\ \ 1 \circ .\Theta {}^{-}1\ \ 0\ \ 1 \circ .\Phi \subset \omega\}$$

"Map" was implicit, "reduce" was a slash, functions were symbols. For example:

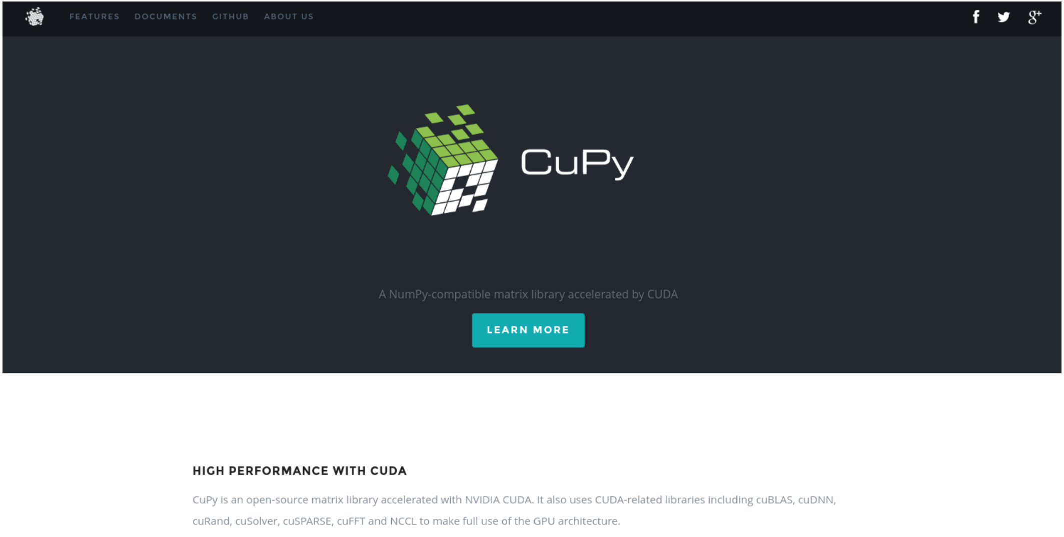
APL	Numpy
$\texttt{m} \leftarrow +/(3 + \iota 4)$	`m = (numpy.arange(4) + 3).sum()`

Numpythonic mindset: GPU and vectorization

As an array abstraction, Numpy presents a high-level way for users to think about vectorization.

Vectorization is key to using GPUs and modern CPUs efficiently.

Numpythonic mindset: GPU and vectorization

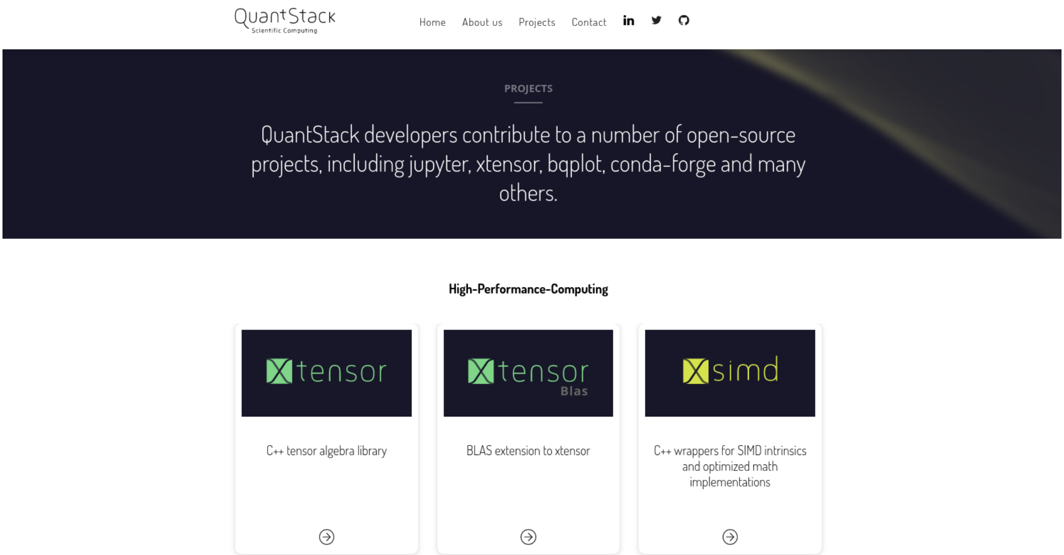


A NumPy-compatible matrix library accelerated by CUDA

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.

Numpythonic mindset: GPU and vectorization



PROJECTS

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

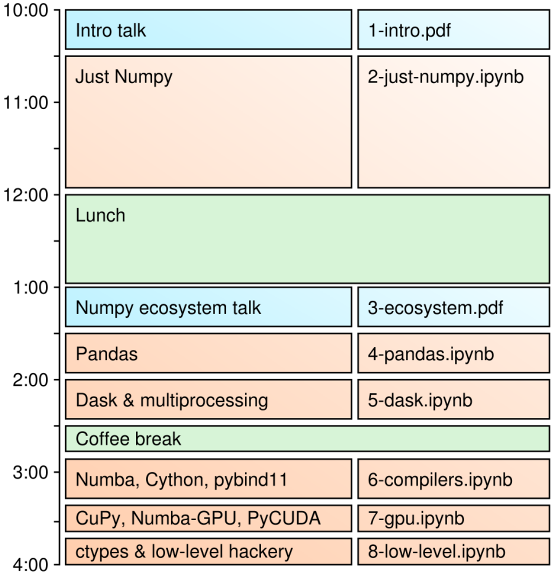
High-Performance-Computing

C++ tensor algebra library

BLAS extension to xtensor

C++ wrappers for SIMD intrinsics and optimized math implementations

Plan for the day



Intro talk	1-intro.pdf
Just Numpy	2-just-numpy.ipynb
Lunch	
Numpy ecosystem talk	3-ecosystem.pdf
Pandas	4-pandas.ipynb
Dask & multiprocessing	5-dask.ipynb
Coffee break	
Numba, Cython, pybind11	6-compilers.ipynb
CuPy, Numba-GPU, PyCUDA	7-gpu.ipynb
ctypes & low-level hackery	8-low-level.ipynb

Skills-based Numpy tutorial with a couple of exercises in the morning: how to think in SIMD.

Overview of libraries in the afternoon: where to look for solutions to your problems.