

Python: a view from the floating-point side

Fernando Pérez

<http://fperez.org>

Fernando.Perez@berkeley.edu

Helen Wills Neuroscience Institute, UC Berkeley

Sage Days 29,
Mathematics, U. Washington, Seattle.
March 24, 2011

Outline

- 1 Context
- 2 Scientific Computing
- 3 Core Scientific Tools
- 4 Growing rapidly

Outline

- 1 Context
- 2 Scientific Computing
- 3 Core Scientific Tools
- 4 Growing rapidly

Outline

- Changes in scientific computing - overview
- How did I get here?
- What challenges do we have today in applied mathematics?
 - Reuse old tools
 - Develop more complex algorithms - beyond just linear algebra.
 - Interface with external systems (hardware, sensors, networks, databases, etc).
 - Use hybrid and complex hardware: cpus, gpus, clusters.
 - Code is a run-time resource.
 - Create reproducible results and truly build upon each other's work.

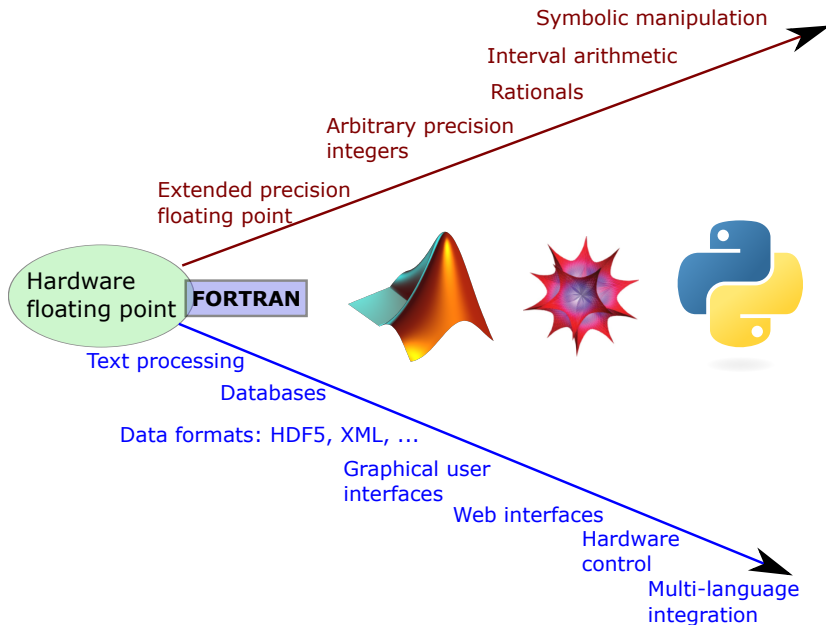
Significant changes in the science-computing relationship

- **High level** computational systems (Matlab, Mathematica, Python...)
- An avalanche of **experimental quantitative data**
 - Biology, genetics, neuroscience, astronomy, climate modeling...
 - All require algorithmic and computational tools
- **Drop in cost** of computing, storage and data transfer.
- **Internet**: a platform for
 - interaction among scientists
 - sharing of data and code
- **Open Source Software**
 - a similar development model to that of scientific production
 - viable alternatives to proprietary software

Outline

- 1 Context
- 2 Scientific Computing
- 3 Core Scientific Tools
- 4 Growing rapidly

Beyond (floating point) number crunching



A bit of history, à la Cremona

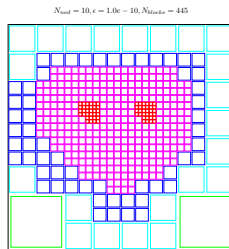
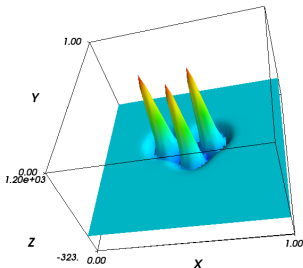
- High school in Colombia, the 80's
 - TI-99/4A, 16KB Basic with my Sony tape recorder.
 - A few home tutoring lessons on 'structured programming', promptly forgotten. Never did anything interesting.
- Engineering college, in Colombia:
 - Engineering: Pascal; my only formal computer course ever.
 - Control a wooden home-made robot in Pascal over a serial port.
- Switch to physics, plot fractals in TurboPascal, Hercules Mono graphics.
 - Program on paper, use mom's office PC on weekends.
 - Debug on paper. Think a lot away from the screen.
 - No idea about free software, or the internet (which I unplugged)
 - No sense of collaborative work.

- Undergraduate thesis: the electrostatic unrestricted 3-body problem.
 - Maple -> C -> gnuplot.
 - Code generation as a natural part of the problem
 - Multi-language integration.
- 1995: Teach computational physics for undergrads: C/Gnuplot on VAX talking to a 486PC running Linux.
 - A complete disaster.
 - Never again. Need different/better tools.
 - They need to be free, like the Linux I had. But for math.
- Graduate school: Mathematica, IDL.
- Thesis: lattice QCD (numerical Quantum Chromodynamics)
 - large open source C package (MILC), custom C code, Mathematica, IDL, bash, sed, awk, perl, gnuplot.
- Tail end of my PhD: perl->python.

Postdoc work: Adaptive, multiwavelet PDE tools

Gregory Beylkin, Vani Cheruvu, FP. Applied Math, CU Boulder.

- Fast application of integral kernels. (Partial Differential Equations)
- Implementation went from 1 to 3 dimensions directly (*extremely* unusual).
- Very complex algorithm that goes beyond pure numerics.
- Very good performance, thanks to NumPy, F2PY and weave.
 - Dynamically generated C++ sources: code as a run-time resource.



Outline

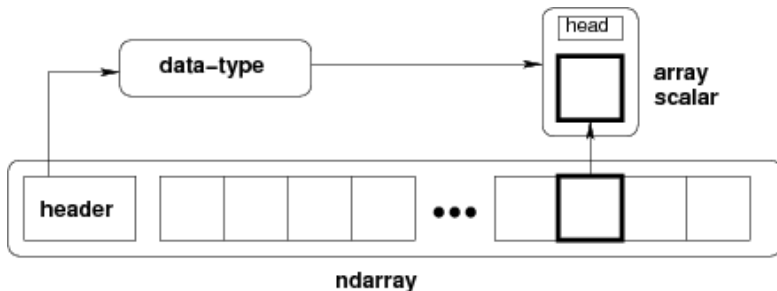
- 1 Context
- 2 Scientific Computing
- 3 Core Scientific Tools
- 4 Growing rapidly

NumPy: the foundation for array processing

- A flexible, efficient, multidimensional array object.
- Homogeneous elements
 - Supports all native types (ints, floats, etc).
 - Arbitrary user-defined types of fixed size.
 - Arbitrary Python objects can also be stored.
- Convenient syntax for high-level operations.
- Math library that operates on arrays.
- Basic scientific functionality:
 - Linear algebra
 - FFTs
 - Random number generation

NumPy: flexible arrays

- Array is a container of objects “of the same kind”: **homogeneous**.
- Concept of “kind” embodied in the data type, or **dtype**.
- Dtypes **can be user-defined** to be arbitrarily complex.
 - Structured arrays: internal structure
 - datarray (<https://github.com/fperez/datarray>): labeled geometry (think R DataFrames)



SciPy: numerical algorithms galore

- **linalg** : Linear algebra routines (including BLAS/LAPACK)
- **sparse** : Sparse Matrices (including UMFPACK, ARPACK,...)
- **fftpack** : Discrete Fourier Transform algorithms
- **cluster** : Vector Quantization / Kmeans
- **odr** : Orthogonal Distance Regression
- **special** : Special Functions (Airy, Bessel, etc).
- **stats** : Statistical Functions
- **optimize** : Optimization Tools
- **maxentropy** : Routines for fitting maximum entropy models
- **integrate** : Numerical Integration routines
- **ndimage** : n-dimensional image package
- **interpolate** : Interpolation Tools
- **signal** : Signal Processing Tools
- **io** : Data input and output
- **Lots more...**

Scikits: domain-specific toolkits

<http://scikits.appspot.com>

- **ann** Approximate Nearest Neighbor library wrapper for Numpy
- **audiolab** A python module to make noise from numpy arrays
- **bootstrap** Bootstrap Error-Estimation Scikit
- **bvp1lg** Boundary value problem (legacy) solvers for ODEs
- **bvp_solver** two-point boundary value problems
- **cuda** Python interface to GPU-powered libraries
- **datasmooth** Scikits data smoothing package
- **eartho** Earth Observation routines for SciPy
- **hydroclimpy** Environmental time series manipulation
- **image** Image processing routines for SciPy
- **learn** A set of python modules for machine learning and data mining
- **odes** ODE and differential algebraic equation solvers
- **optimization** A python module for numerical optimization
- **samplerate** A python module for high quality audio resampling
- **scattpy** Light Scattering methods for Python
- **sparse** Scikits sparse matrix package
- **statsmodels** Statistical computations and models for use with SciPy
- ... More that don't fit here ...

IPython: the soundbite edition

Getting all the power from interactive computing in Python

- ① A better Python shell: object introspection, system access, extensible 'magic' commands, ...
- ② A flexible, embeddable interpreter:
 - ① debugging, mix batch/interactive work.
 - ② build custom systems based on Python with new syntax, etc.
- ③ Data visualization and GUIs: Matplotlib, Mayavi, all GUIs toolkits.
- ④ A rich toolkit: terminal, Qt console, HTTP client.
- ⑤ High level (and interactive!) parallel computing interfaces.

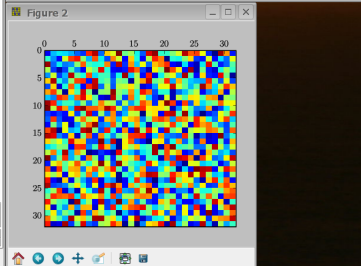
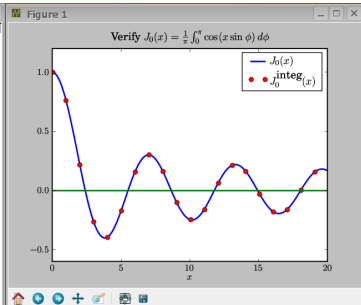
IPython: Matlab/IDL-like interactive use

```
fperez@longs:/home/fperez - Shell - Konsole
longs[-]> ipython -pylab
Python 2.4.3 (#2, Apr 27 2006, 14:43:58)
Type "copyright", "credits" or "license" for more information.

IPython 0.7.3.svn -- An enhanced Interactive Python.
? -> Introduction to IPython's features.
%magic -> Information about IPython's 'magic' % functions.
help -> Python's own help system.
object? -> Details about 'object'. ?object also works, ?? prints more.

Welcome to pylab, a matplotlib-based Python environment.
For more information, type 'help(pylab)'.

In [1]: import math, numpy
In [2]: from scipy.integrate import quad
In [3]: from scipy.special import j0
In [4]: def j0i(x):
...:     """Integral form of J_0(x)"""
...:     def integrand(phi):
...:         return math.cos(x*math.sin(phi))
...:     return (1.0/math.pi)*quad(integrand,0,math.pi)[0]
In [5]: x = numpy.linspace(0,20,200) # sample grid: 200 points between 0 and 20
In [6]: y = j0i(x) # sample J0 at all values of x
In [7]: x1 = x[::10] # subsample the original grid every 10th point
In [8]: y1 = map(j0i,x1) # evaluate the integral form at all points in x1
In [9]: # Make a plot with these values (the ; suppresses output)
In [10]: plot(x,y,label=r'$J_0(x)$');
In [11]: plot(x1,y1,'ro',label=r'$J_0\{\text{int}\}(x)$');
In [12]: axhline(0,color='green',label='nolegend');
In [13]: title(r'Verify $J_0(x)=\frac{1}{\pi}\int_0^\pi\cos(x\sin\phi)d\phi$');
In [14]: xlabel('$x$');
In [15]: legend();
In [16]: matshow(numpy.random.random((32,32)))
Out[16]: <matplotlib.figure.Figure instance at 0x4630042c>
```



Some projects using IPython

Scientific

- **Sage**: open source mathematics.
- **PyRAF**: Space Telescope Science Institute
- **CASA**: National Radio Astronomy Observatory.
- **Ganga**: CERN.
- **PyMAD**: neutron spectrometer, Institut Laue Langevin.
- **Sardana**: European Synchrotron Radiation Facility.
- **ASCEND**: engineering modeling (Carnegie Mellon).
- **JModelica**: dynamical systems.
- Denver Aerosol Sources and Health (**DASH**), CU Boulder.
- **PyIMSL** Studio, by Visual Numerics.
- **Trilinos**: Sandia National Lab.
- **Pymerase**: microarray gene expression.

Web/Other

- **Visual Studio 2010**: MS.
- **Django** web.
- **Turbo Gears** web.
- **Pylons** web framework
- **Zope** and **Plone** CMS.
- Axon Shell, BBC **Kamaelia**.
- **Schevo** database.
- **Pitz**: distributed task/bug tracking.
- **iVR** (interactive Virtual Reality).
- **Movable Python** (portable Python environment).
- ...

IPython: a REPL (Read/Eval/Print Loop)

Core idea: manage a namespace

- Read: take user input.
- Eval: execute code.
- Print: provide output.
- Add support for data transfer...

...and interactive and parallel work start looking very similar.

These steps can happen in multiple processes:

- Read: user environment
- Eval: kernel process
- Print: user environment

IPython: a REPL (Read/Eval/Print Loop)

Core idea: manage a namespace

- Read: take user input.
- Eval: execute code.
- Print: provide output.
- Add support for data transfer...

...and interactive and parallel work start looking very similar.

These steps can happen in multiple processes:

- Read: user environment
- Eval: kernel process
- Print: user environment

IPython: a REPL (Read/Eval/Print Loop)

Core idea: manage a namespace

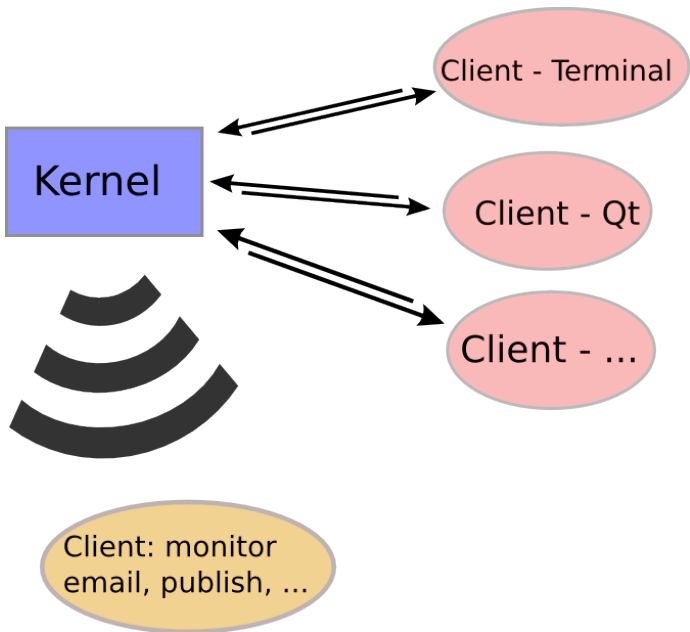
- Read: take user input.
- Eval: execute code.
- Print: provide output.
- Add support for data transfer...

...and interactive and parallel work start looking very similar.

These steps can happen in multiple processes:

- Read: user environment
- Eval: kernel process
- Print: user environment

More complex interactive uses?



A messaging protocol

Direct communication

- Execute code ('eval')
- Object information
- Complete
- History
- Connect

Broadcasting

- Functional execution:
 - Python inputs
 - Python outputs
 - Python errors
- Side effects:
 - Streams (stdout, stderr, etc)
 - Display data: plots, other payloads

The socket library that acts as a concurrency framework

- Pure C++ library.
- Python bindings in [Cython](#) (Brian Granger, Min RK). Python 2.5-3.2.
- Python bindings run messaging in native threads - [no GIL](#)
- Abstractions are at the [message delivery](#) level, not the raw-bytes level.
- Socket types encapsulate [messaging patterns](#).
- Open source (LGPL), actively developed.

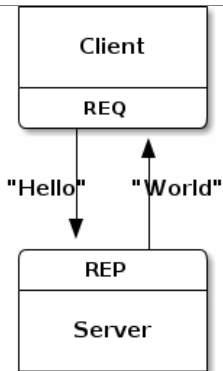


Figure 1 – Request-Reply

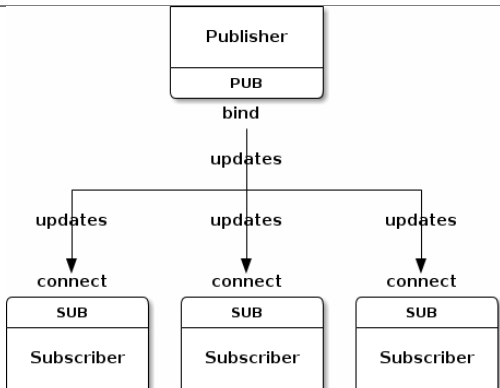
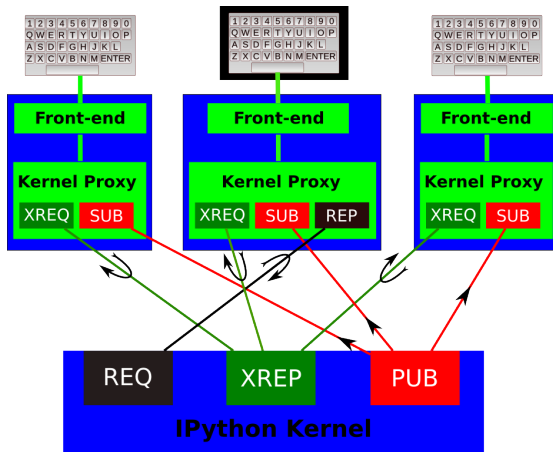


Figure 4 – Publish-Subscribe

Image credit: official ØMQ documentation

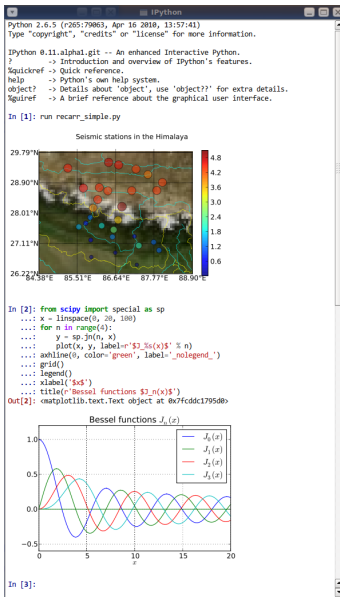
Interactive IPython on ØMQ



- - Kernel raw_input
- - Requests to kernel
- - Kernel output broadcast
- ↩ - Request/Reply direction

Back to the clients: a rich Qt Console

Enthought: sponsorship, Evan Patterson.



Feels like a console, runs like a GUI

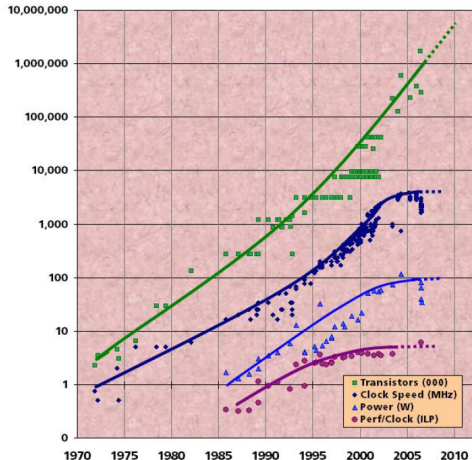
- Inline and floating images
- Syntax highlighting, full multiline editing
- Session saving
 - HTML (with PNG or SVG)
 - PDF/printing
- Help viewer
- %magics, !system access, IPython...
- Detach/reattach support

A little detour

Python and parallel computing

Parallel computing: why should we care?

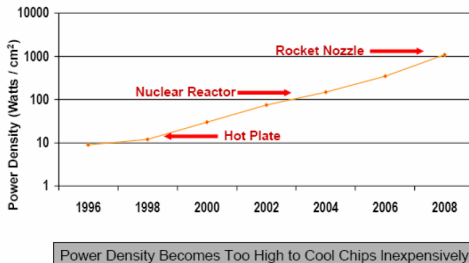
Because reality looks like this:



Sources: Intel, Microsoft (Sutter), Stanford (Olukotun, Hammond) & Berkeley (Yelick)

We can't escape thermodynamics

Moore's Law Extrapolation: Power Density for Leading Edge Microprocessors



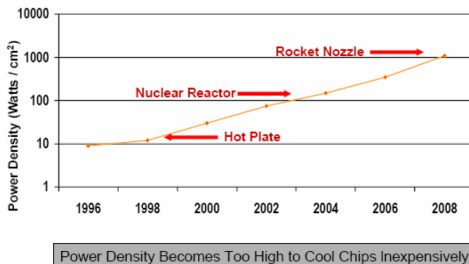
Sources: Shekhar Borkar, Intel Corp & Kathy Yelick, UC Berkeley

The vendor's solutions

- Multicore chips: everywhere (soon in your phone)
- Graphics cards: hundreds of specialized processors per card.
- High-density clusters: SiCortex (> 5000 processors in a cabinet).

We can't escape thermodynamics

Moore's Law Extrapolation: Power Density for Leading Edge Microprocessors



Sources: Shekhar Borkar, Intel Corp & Kathy Yelick, UC Berkeley

The vendor's solutions

- Multicore chips: everywhere (soon in your phone)
- Graphics cards: hundreds of specialized processors per card.
- High-density clusters: SiCortex (> 5000 processors in a cabinet).

The infamous Global Interpreter Lock in CPython

Only one thread can modify Python state/variables at a time

- Historical reasons, simplicity of implementation
- All attempts at removing it have **failed**
 - 2x loss of performance is not acceptable
- Threads only good for i/o bound tasks.
- Mostly **useless for CPU-bound ones**.
- Can operate on pre-allocated arrays, but:
 - code must be in C/C++/Fortran/Cython
 - be **very careful** with locking if code is not atomic at Python level

The best possible reference on the GIL: David Beazley's work

<http://www.dabeaz.com/GIL>

The infamous Global Interpreter Lock in CPython

Only one thread can modify Python state/variables at a time

- Historical reasons, simplicity of implementation
- All attempts at removing it have **failed**
 - 2x loss of performance is not acceptable
- Threads only good for i/o bound tasks.
- Mostly **useless for CPU-bound ones**.
- Can operate on pre-allocated arrays, but:
 - code must be in C/C++/Fortran/Cython
 - be **very careful** with locking if code is not atomic at Python level

The best possible reference on the GIL: David Beazley's work

<http://www.dabeaz.com/GIL>

Parallelism in Python

- **In-process (mind the GIL)**

- [Data parallelism](#) with threaded libraries
- Numpy/Scipy can use a [threaded ATLAS](#)
- [Numexpr](#): a 'numpy VM'
- [Theano](#): a library that thinks it's a compiler
- GPU-based solutions: [PyCuda](#)/[PyOpenCL](#), [scikits.cuda](#).
- Hand-written threaded code...

- **Out-of-process**

- The [multiprocessing](#) module
- Python [futures](#): coming in Python 3.2.
- Communicating Sequential Processes, ParallelPython, ... many more
- [IPython](#)

Parallelism in Python

• In-process (mind the GIL)

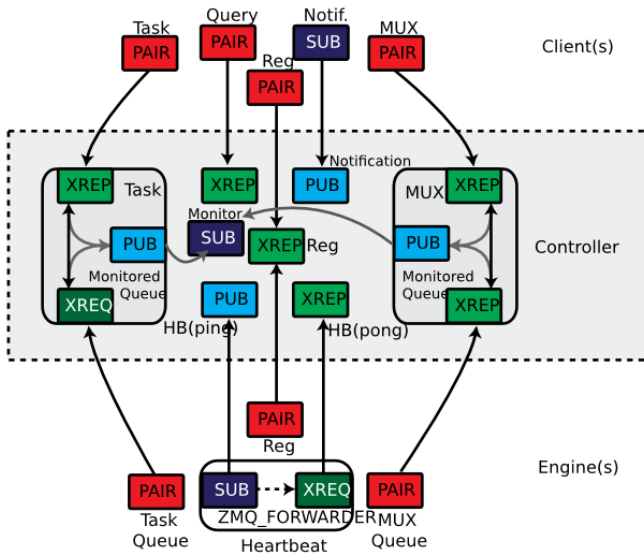
- [Data parallelism](#) with threaded libraries
- Numpy/Scipy can use a [threaded ATLAS](#)
- [Numexpr](#): a 'numpy VM'
- [Theano](#): a library that thinks it's a compiler
- GPU-based solutions: [PyCuda](#)/[PyOpenCL](#), [scikits.cuda](#).
- Hand-written threaded code...

• Out-of-process

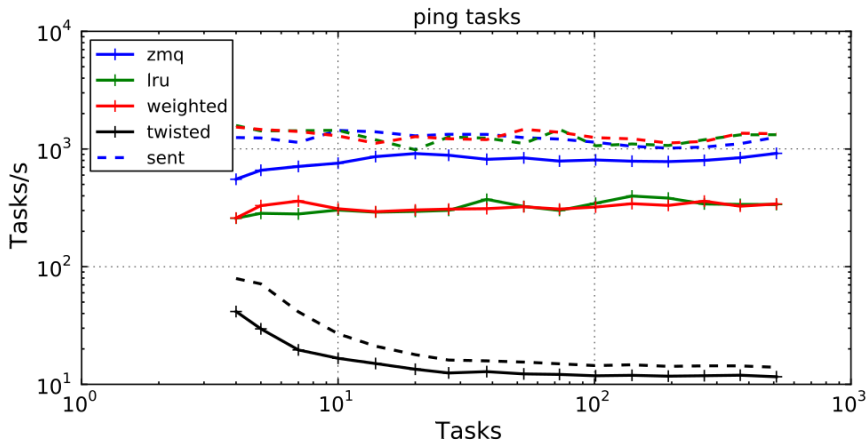
- The [multiprocessing](#) module
- Python [futures](#): coming in Python 3.2.
- Communicating Sequential Processes, ParallelPython, ... many more
- [IPython](#)

IPython for parallel computing

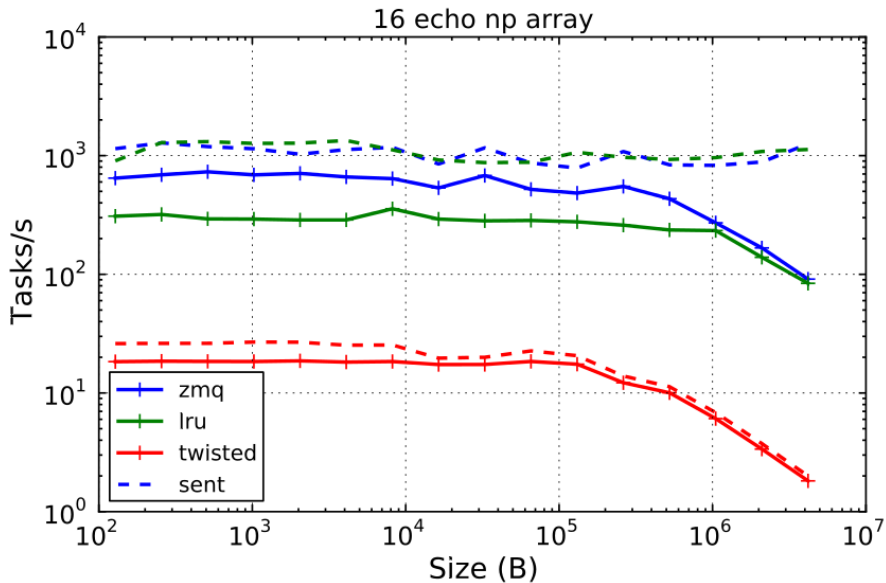
With Brian Granger (Cal Poly San Luis Obispo), Min Ragan-Kelley (Berkeley)



Phenomenal task latency



...and throughput



Multiple usage patterns

- Direct interface: explicit (and flexible) control of where things run.
 - Choice of blocking behavior up to the user.
- Task interface: load-balanced (with flexible scheduling policies)
- Data push/pull, scatter/gather.
- Decorators that encapsulate many common patterns
- Informative exception propagation
- Explicit node-to-node communication:
 - MPI-style tasks
 - ... without all the pain of MPI.

Neat trick: DAG dependencies

A simple DAG example

```
In [2]: G = random_dag(32,128)
In [3]: jobs = {}

# in reality, each job would presumably be different
# randomwait is just a function that sleeps for a random interval
In [4]: for node in G:
...:     jobs[node] = randomwait

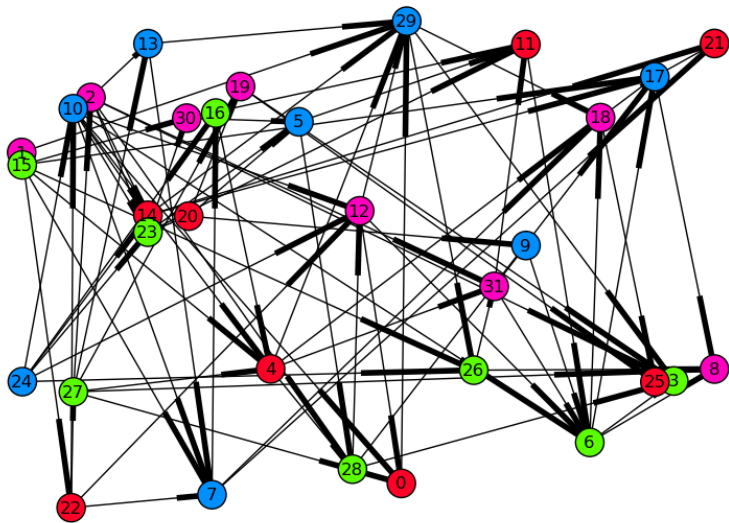
In [5]: c = client.Client()

In [6]: results = {}

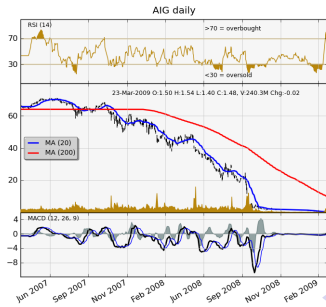
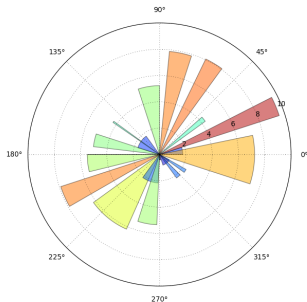
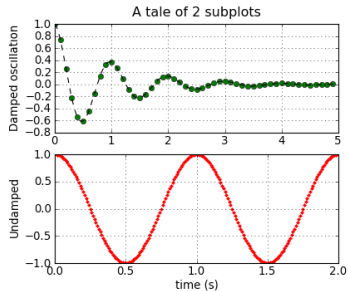
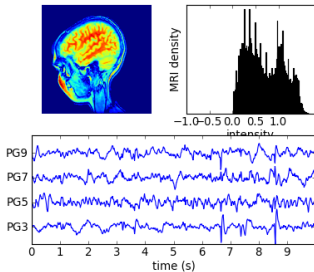
In [7]: for node in G.topological_sort():
...:     # get list of AsyncResult objects from nodes
...:     # leading into this one as dependencies
...:     deps = [ results[n] for n in G.predecessors(node) ]
...:     # submit and store AsyncResult object
...:     results[node] = client.apply(jobs[node], after=deps,
        block=False)

In [8]: [ r.get() for r in results.values() ]
```

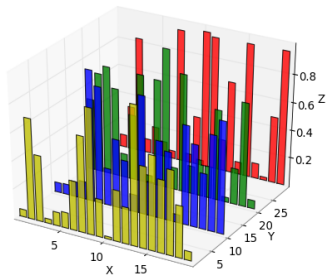
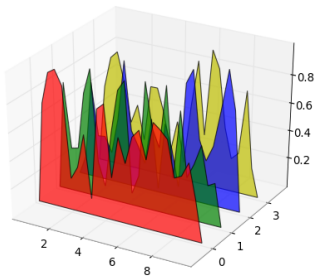
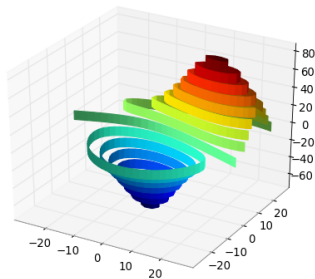
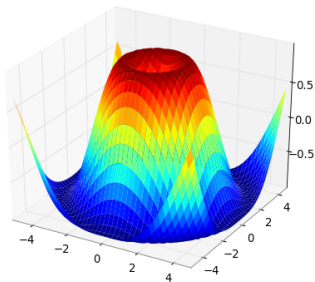

DAG dependencies - validation



Matplotlib: 2d plotting



Matplotlib: 3d plotting



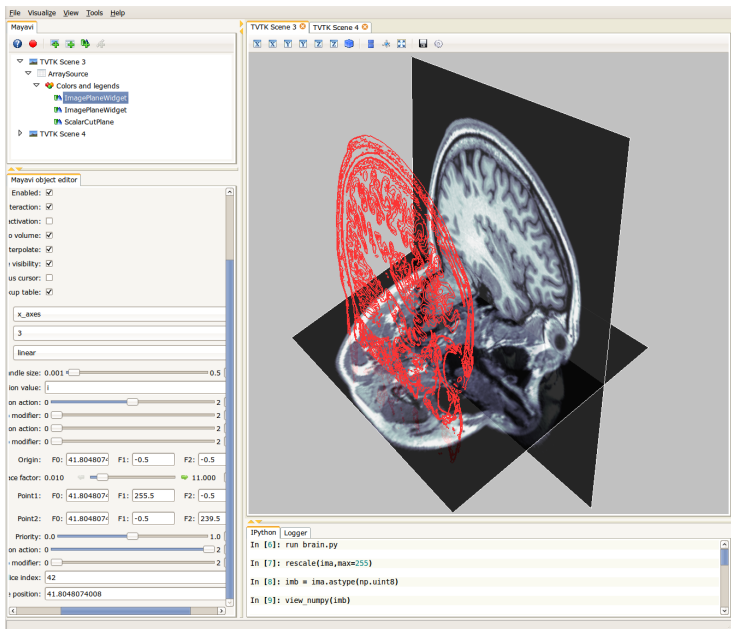
- Great quality plots on disk (png, pdf, etc): what Sage uses
- Familiar, high-level API (to those who know matlab).
- Local plotting with multiple GUI toolkits
- Interactive data navigation in plot windows
- GUI toolkit-independent event handling
- \LaTeX support, without the \LaTeX dependency
- Embeddable in GUI apps with any toolkit.

MayaVi: sophisticated data visualization

- Free, easy to use scientific data visualizer.
- Heavy lifting of OpenGL-based rendering: VTK (a C++ library).
- A very good example of how to properly use Python:
 - A standalone GUI program...
 - also a library
 - Python: flexibility.
 - C++: performance (hardware-accelerated OpenGL)

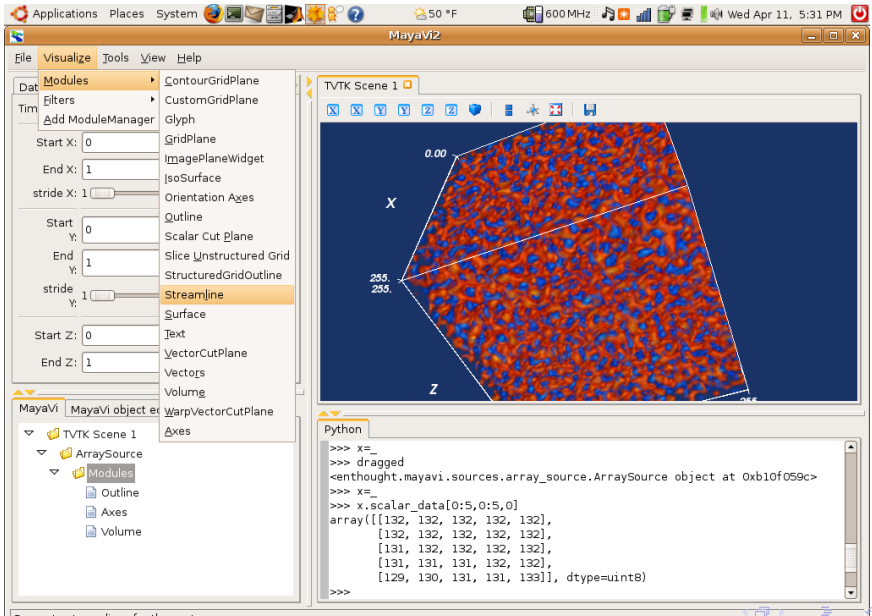
The punchline: fully programmable visualization, with builtin access to all kinds of numerical (and other) libraries from within the viz tool.

Mayavi: 3d visualization (VTK)



FluidLab: a MayaVi based CFD visualization tool

K. Julien, P. Schmitt (now NCAR), B. Barrow, F. Pérez (App. Math, CU).



Sympy: symbolic and multiprecision computing

```
Terminal
In [14]: dsolve(f(x).diff(x, x) + f(x), f(x))
Out[14]: f(x) = C1·sin(x) + C2·cos(x)

In [15]: zeta(4, x)
Out[15]: ζ(4, x)

In [16]: zeta(4, 1)
Out[16]:

$$\frac{\pi^4}{90}$$


In [17]: Ylm(2, 1, theta, phi)
Out[17]:

$$\frac{-\sqrt{30} \cdot \cos(\theta) \cdot e^{i \cdot \phi} \cdot \sin(\theta)}{4 \cdot \sqrt{\pi}}$$


In [18]: integrate(sin(x)**3)
Out[18]:

$$-\cos(x) + \frac{\cos^3(x)}{3}$$


In [19]: integrate(exp(-x**2)*sin(x))
Out[19]:

$$\int e^{-x^2} \cdot \sin(x) \, dx$$


In [20]:
```


Outline

- 1 Context
- 2 Scientific Computing
- 3 Core Scientific Tools
- 4 Growing rapidly

Workshops and Conferences

- Python Applied to Computational Chemistry and Molecular Modelling (June 2010, Barcelona), neuroscience (Trento, 2010; St. Andrews 2011), ...
- US Scipy Conference: 2001-today.
 - **SciPy 2011: Austin, TX. July 11-16..**
- EuroScipy: since 2008.
 - **EuroScipy 2011: Paris, August 25-28.**
- Scipy India: since 2009.
 - **Scipy India 2011: December.**
- Scipy Japan 2011
 - **Being planned...**
- At SIAM conferences (annual 2008 in San Diego, CSE 2009 in Miami)
 - CSE 2011 in Reno: standing room only, 5 sessions, many talks.
- Supercomputing'09: Python sessions, extremely well attended!
- **Sage days: 29 workshops and counting...**

Education

- Fossee India (multi-million US \$ investment in Python-based educational software, course materials and training across all of India).
- SECANT: Science Education in Computational Thinking (NSF funded)
- **Sage workshops: lots of students.**
- MIT 6.0X series: now in Python.
- UC Berkeley: Python bootcamp/graduate course - Josh Bloom (Astronomy).

Computing in Science and Engineering (IEEE/AIP)

- Special issue in 2007: one of the most popular ever
- **Special issue in 2011. Just came out!**

US Federal Labs

- LBL, LLNL, Los Alamos, Sandia, Oak Ridge, ...
- NASA (JPL, Hubble Space Telescope, ...)
- NIST
- NCAR
- NOAA

Industry

- Enthought (Austin, TX). Numpy, Scipy, Mayavi, scipy conference.
- The Python Academy: Germany, Euroscipy
- Visual Numerics: PyIMSL Studio
(IMSL+Python+ipython/numpy/scipy/matplotlib)
- Google,
- Industrial Light and Magic,
- Disney,
- Financial world, ...

Lots more

- http://www.scipy.org/Topical_Software

Technical considerations

- A flexible, modern language
- A good tool for today's numerical/scientific computing problems
- Excellent for problems where adaptive code/algorithms are needed.
- With a healthy ecosystem of interesting projects.

Social considerations

- Developed for and by researchers
- Do black boxes (matlab, Mathematica, etc) belong in research?
- Cost: an investment in open tools is an investment in students and researchers, not in paying for software licenses.

There's a lot to do still
We hope many of you will contribute!

Thank you!

Questions?