Outline

1. Scientific Computing
   - Existing tools
   - Python?

2. Development in Python

3. OK, but does anyone use it?
   - EEG analysis for epilepsy
   - JPL: Mars mission data visualization
   - Multiwavelets for PDEs
   - NetworkX: complex networks
   - MayaVi: customizable data visualization
   - Sage

4. IPython: Interactive Python

5. Parallel and distributed computing

6. IPython in this context
1 Scientific Computing
   - Existing tools
   - Python?

2 Development in Python

3 OK, but does anyone use it?
   - EEG analysis for epilepsy
   - JPL: Mars mission data visualization
   - Multiwavelets for PDEs
   - NetworkX: complex networks
   - MayaVi: customizable data visualization
   - Sage

4 IPython: Interactive Python

5 Parallel and distributed computing

6 IPython in this context
Outline

1. Scientific Computing
   - Existing tools
   - Python?

2. Development in Python

3. OK, but does anyone use it?
   - EEG analysis for epilepsy
   - JPL: Mars mission data visualization
   - Multiwavelets for PDEs
   - NetworkX: complex networks
   - MayaVi: customizable data visualization
   - Sage

4. IPython: Interactive Python

5. Parallel and distributed computing

6. IPython in this context
Outline

1. Scientific Computing
   - Existing tools
   - Python?

2. Development in Python

3. OK, but does anyone use it?
   - EEG analysis for epilepsy
   - JPL: Mars mission data visualization
   - Multiwavelets for PDEs
   - NetworkX: complex networks
   - MayaVi: customizable data visualization
   - Sage

4. IPython: Interactive Python

5. Parallel and distributed computing

6. IPython in this context
1 Scientific Computing
   - Existing tools
   - Python?

2 Development in Python

3 OK, but does anyone use it?
   - EEG analysis for epilepsy
   - JPL: Mars mission data visualization
   - Multiwavelets for PDEs
   - NetworkX: complex networks
   - MayaVi: customizable data visualization
   - Sage

4 IPython: Interactive Python

5 Parallel and distributed computing

6 IPython in this context
1. Scientific Computing
   - Existing tools
   - Python?

2. Development in Python

3. OK, but does anyone use it?
   - EEG analysis for epilepsy
   - JPL: Mars mission data visualization
   - Multiwavelets for PDEs
   - NetworkX: complex networks
   - MayaVi: customizable data visualization
   - Sage

4. IPython: Interactive Python

5. Parallel and distributed computing

6. IPython in this context
1 Scientific Computing
   - Existing tools
   - Python?

2 Development in Python

3 OK, but does anyone use it?
   - EEG analysis for epilepsy
   - JPL: Mars mission data visualization
   - Multiwavelets for PDEs
   - NetworkX: complex networks
   - MayaVi: customizable data visualization
   - Sage

4 IPython: Interactive Python

5 Parallel and distributed computing

6 IPython in this context
Pure Fortran, C and C++

- Tools optimized for the CPU, not the developer.
- Low-level (data types and libraries).
- Difficult access to visualization, quick profiling, text processing, . . .
- No interactive facilities.

However!

- They deliver excellent performance.
- Millions of LOC in proven scientific codes.
- We need to *work with these tools*, not replace them!
Pure Fortran, C and C++

- Tools optimized for the CPU, not the developer.
- Low-level (data types and libraries).
- Difficult access to visualization, quick profiling, text processing, . . .
- No interactive facilities.

**However!**
- They deliver excellent performance.
- Millions of LOC in proven scientific codes.
- We need to *work with these tools*, not replace them!
Complementary high level tools

- Mathematica, Maple, Matlab, IDL: popular, for good reasons.
  - Interactivity, visualization, extensive libraries.
  - Unpleasant for large-scale programs and non-mathematical tasks.
  - Expensive, proprietary: lock-in.

- Another common approach: the ‘command pipeline’
  - FORTRAN, C, C++ programs ...
  - driven by Bash/awk/sed/Perl scripts ...
  - which feed them input and send the output ...
  - to visualization/analysis programs.
Complementary high level tools

- Mathematica, Maple, Matlab, IDL: popular, for good reasons.
  - Interactivity, visualization, extensive libraries.
  - Unpleasant for large-scale programs and non-mathematical tasks.
  - Expensive, proprietary: lock-in.

- Another common approach: the ‘command pipeline’
  - FORTRAN, C, C++ programs ...
  - driven by Bash/awk/sed/Perl scripts ...
  - which feed them input and send the output ...
  - to visualization/analysis programs.
Python: high level language

- **Free** (BSD license), highly portable (Linux, OSX, Windows, lots...).
- **Interactive** interpreter provided.
- Extremely readable syntax ("executable pseudo-code").
- **Simple**: non-professional programmers can use it effectively.
- Clean object oriented model, but not mandatory.
- Strongly but **dynamically** typed.
- Rich built-in types: lists, sets, dictionaries (hash tables), strings, ...
- Very comprehensive standard library (**batteries included**)
- Standard libraries for IDL/Matlab-like arrays (**Numpy**)
- Easy to **wrap** existing C, C++ and FORTRAN codes.
Outline

1. Scientific Computing
   - Existing tools
   - Python?

2. Development in Python

3. OK, but does anyone use it?
   - EEG analysis for epilepsy
   - JPL: Mars mission data visualization
   - Multiwavelets for PDEs
   - NetworkX: complex networks
   - MayaVi: customizable data visualization
   - Sage

4. IPython: Interactive Python

5. Parallel and distributed computing

6. IPython in this context
Python is a general programming language

---

**It’s not a bug, it’s a feature!**

- Get others to develop the non-scientific tools you may need.
- Networking, text processing, XML parsing, databases, etc. . .
- Integrated support for automated testing.
- Excellent documentation tools
- Supports all major GUI toolkits.

---

**There are still rough edges**

- Installation, deployment: much harder than needed.
- No good, single-point of entry integrated environment/help
- Documentation too scattered.
- Your favorite/critical tool may well not exist yet.
Python is a general programming language

It’s not a bug, it’s a feature!

- Get others to develop the non-scientific tools you may need.
- Networking, text processing, XML parsing, databases, etc.
- Integrated support for automated testing.
- Excellent documentation tools
- Supports all major GUI toolkits.

There are still rough edges

- Installation, deployment: much harder than needed.
- No good, single-point of entry integrated environment/help
- Documentation too scattered.
- Your favorite/critical tool may well not exist yet.
Algorithmic scaling: from quicksort to PDE solvers

def qsort(L):
    if len(L) <= 1: return L
    return qsort([lt for lt in L[1:] if lt < L[0]]) + [L[0]] + qsort([ge for ge in L[1:] if ge >= L[0]])

Use cases

- Teaching.
- Research.
- Commercial applications (many tools BSD licensed).
- Local codes or web services.
Algorithmic scaling: from quicksort to PDE solvers

```python
def qsort(L):
    if len(L) <= 1:  return L
    return qsort([lt for lt in L[1:] if lt < L[0]]) + [L[0]] + qsort([ge for ge in L[1:] if ge >= L[0]])
```

Use cases

- Teaching.
- Research.
- Commercial applications (many tools BSD licensed).
- Local codes or web services.
Development Model

... scalable

**Code size/complexity**

- Interactive experimentation...
- Small, self-contained scripts...
- Or million-lines projects.
- From occasional/novice to full-time use (try that with C++).

**Performance:** “premature optimization is the root of all evil”

... but incremental optimization isn’t so bad. Some tools:

- Cython: ‘static python’ compiled to C
- Scipy.weave: inline C/C++ in Python source code.
- Numpy.f2py: Fortran access with full array support.
... scalable

**Code size/complexity**
- Interactive experimentation...
- Small, self-contained scripts...
- Or million-lines projects.
- From occasional/novice to full-time use (try that with C++).

**Performance: “premature optimization is the root of all evil”**
... but incremental optimization isn’t so bad. Some tools:
- Cython: ‘static python’ compiled to C
- Scipy.weave: inline C/C++ in Python source code.
- Numpy.f2py: Fortran access with full array support.
Using your codes from today with Python

Code too monolithic but with good core pieces?
- Break into a library core and control layers.
- Wrap the libraries and expose them to Python.
- Use Python for control or use interactively.

For existing libraries in C/C++/Fortran
- Provide Python bindings for them.
- It will make it much easier for others to test/use them.
- The network economy benefits all. In this case, $O(n^2)$ is good!
Using your codes from today with Python

- Code too monolithic but with good core pieces?
  - Break into a library core and control layers.
  - Wrap the libraries and expose them to Python.
  - Use Python for control or use interactively.

- For existing libraries in C/C++/Fortran
  - Provide Python bindings for them.
  - It will make it much easier for others to test/use them.
  - The network economy benefits all. In this case, $O(n^2)$ is good!
Outline

1. Scientific Computing
   - Existing tools
   - Python?

2. Development in Python

3. OK, but does anyone use it?
   - EEG analysis for epilepsy
   - JPL: Mars mission data visualization
   - Multiwavelets for PDEs
   - NetworkX: complex networks
   - MayaVi: customizable data visualization
   - Sage

4. IPython: Interactive Python

5. Parallel and distributed computing

6. IPython in this context
Data analysis for epilepsy surgery
Isolating the origin of drug-resistant epileptic seizures which require surgery.

John Hunter, Department of Pediatric Neurology, University of Chicago.
Electrode location in 3D, combined with MRI data
Correlation analysis of seizure data
Final location of epileptic foci for surgery
One of my lead developers mentioned that they had sent a bug to you about the annotations feature of MatPlotLib. Would you be able to let me know what the timeline is to resolve that bug? The reason is that the feature is needed for the Phoenix project and their arrival at Mars will be in March sometime, but they are doing their testing in the coming few months. This annotation feature is used on reports that present the analysis of the trajectory to the navigation team and it shows up on our schedule. It would really help me to know approximately when it could be resolved.

B-plane plots are used to show the trajectory of a spacecraft with respect to the target body (specifically perpendicular to the incoming asymptote of the spacecraft trajectory) and we plot them with the y-axis inverted. The plot is used heavily in flight operations so it is important to our customers.

In addition, we have what is called a thundering heard plot where many different trajectory solutions (determined from different measurement sources) are plotted together. The annotations are import there so we can see which plot corresponds to each source of data. I hope it helps to know how your code will be used in spacecraft navigation.

Thanks for all your efforts.
Expected communication power levels between an orbiting spacecraft and a lander as it goes through the atmosphere:
Multiresolution algorithms for integral operators

G. Beylkin, V. Cheruvu, FP (U. Colorado), M. Mohlenkamp (Ohio U).

- Fast application of integral kernels for PDEs.
- Complex algorithm that goes beyond pure numerics.
- Fortran, dynamically generated C++, python, 2d vis, 3d vis...
NetworkX: tools for complex networks
Aric Hagberg, Pieter Swart et. al., Los Alamos Theory Division
MayaVi: high-level, interactive VTK for 3D visualization
Sage: open source mathematics
Led by William Stein, U. Washington Mathematics (Berkeley alumnus)
Who else?

Academia

- **PyRAF**: Astronomical data analysis (Hubble Space Telescope).
- **PyTrilinos**: Parallel solvers (Sandia National Labs).
- **CDAT**: Climate Data Analysis Tools (Lawrence Livermore).
- **FiPy/OOF2**: Finite Volume codes for materials science (NIST).
- **SECANT**: Computational curriculum (Purdue, NSF funded)

Industry

- **InteractiveSupercomputing.com**: Python interface to their proprietary backend.
- **Numenta**: pattern recognition algorithms (Jeff Hawkins-Palm)
- **Google (employs Guido van Rossum)**: AppEngine, YouTube.

Lots more

- [http://www.scipy.org/Topical_Software](http://www.scipy.org/Topical_Software)
Who else?

Academia

- **PyRAF**: Astronomical data analysis (Hubble Space Telescope).
- **PyTrilinos**: Parallel solvers (Sandia National Labs).
- **CDAT**: Climate Data Analysis Tools (Lawrence Livermore).
- **FiPy/OOF2**: Finite Volume codes for materials science (NIST).
- **SECANT**: Computational curriculum (Purdue, NSF funded)

Industry

- **InteractiveSupercomputing.com**: Python interface to their proprietary backend.
- **Numenta**: pattern recognition algorithms (Jeff Hawkins-Palm)
- **Google (employs Guido van Rossum)**: AppEngine, YouTube.

Lots more

- [http://www.scipy.org/Topical_Software](http://www.scipy.org/Topical_Software)
Who else?

Academia

- **PyRAF**: Astronomical data analysis (Hubble Space Telescope).
- **PyTrilinos**: Parallel solvers (Sandia National Labs).
- **CDAT**: Climate Data Analysis Tools (Lawrence Livermore).
- **FiPy/OOF2**: Finite Volume codes for materials science (NIST).
- **SECANT**: Computational curriculum (Purdue, NSF funded)

Industry

- **InteractiveSupercomputing.com**: Python interface to their proprietary backend.
- **Numenta**: pattern recognition algorithms (Jeff Hawkins-Palm)
- **Google (employs Guido van Rossum)**: AppEngine, YouTube.

**Lots more**

- [http://www.scipy.org/Topical_Software](http://www.scipy.org/Topical_Software)
Outline

1. Scientific Computing
   - Existing tools
   - Python?

2. Development in Python

3. OK, but does anyone use it?
   - EEG analysis for epilepsy
   - JPL: Mars mission data visualization
   - Multiwavelets for PDEs
   - NetworkX: complex networks
   - MayaVi: customizable data visualization
   - Sage

4. IPython: Interactive Python

5. Parallel and distributed computing

6. IPython in this context
1. A better Python shell:
   - object introspection, system access, ‘magic’ command system, ...

2. An embeddable interpreter: mix batch and interactive work.

3. A flexible component: tweak to taste for your project.

4. An interactive component plug into GUIs, browsers, etc.

5. High level distributed/parallel computing.
Cast of Characters

- **Brian Granger** - Physics, Cal Poly San Luis Obispo
- Ville Vainio - CS, Tampere University of Technology, Finland
- **Min Ragan-Kelley** - Applied Science and Technology, UC Berkeley
- Gael Varoquaux - Neurospin (Orsay, France)
- Robert Kern - Enthought
- Stefan van der Walt - Applied Math, U. Stellenbosch, South Africa
- Jorgen Stenarson - Sweden
- Onrej Certik - Physics, U. Nevada Reno
- Laurent Dufrechou - France
- Vivian De Smedt - Belgium
- Darren Dale - Astronomy, Cornell
- *Many more I am forgetting...*
Quick overview

History

- Started in late 2001, ‘just one afternoon’...
- It seems to have filled a need.
- Organic community uptake, mostly ‘spare time’.
- Messy code, good functionality
- Many contributions from outsiders

Tools

- Project hosting (site, wiki, mailing lists): http://ipython.scipy.org (courtesy of Enthought)
- Code hosting and bug tracking: http://launchpad.net/ipython (Canonical)
- Distributed version control: Bazaar.
IPython: components

- Interactive app - **terminal**
- **Embeddable** shell (dbg, exploration, control)
- **Widget** (Wx, GTK, Tk, Qt)
- Network-aware computational engine
  - **Interactive distributed and parallel** computing
  - High-level interfaces for this
  - Complement, not replace MPI
- Node for **visual programming** driven HPC - IPVision
Matlab-like interactive usage
Matplotlib: high quality plotting. Scipy: numerical algorithms.

```
In [1]: import math, numpy
In [2]: from scipy.integrate import quad
In [3]: from scipy.special import j0
In [4]: def j0j(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 = j0(x) # sample J0 at all values of x
In [7]: x1 = x[:-10] # subsample the original grid every 10th point
In [8]: y1 = map(j0j,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'$\int_0^\alpha \cos(x \sin \phi) \, \mathrm{d} \phi$';
In [12]: axhline(0,color='green', label='nolegend_1');
In [13]: title(r'$\int_0^\alpha \cos(x \sin \phi) \, \mathrm{d} \phi$');
In [14]: xtitle('5x5');
In [15]: legend();
In [16]: matshow(numpy.random.random((32,32)))
Out[16]: <matplotlib.figure.Figure instance at 0x4630042c>
```
Multiple GUI systems

VTK with WxWidgets

In [1]: from enthought.tvtk.tools import mlab
In [2]: from scipy import *
In [3]: def f(x, y):
   ...
   return sin(x+y) + sin(2*x-y) + cos(3*x+4*y)
   ...
In [4]: x = linspace(-5.0, 5.0, 200)
In [5]: y = linspace(-5.0, 5.0, 200)
In [6]: fig = mlab.figure()
In [7]: surf = mlab.SurfRegular(x, y, f)
In [8]: fig.add(surf)
Outline

1. Scientific Computing
   - Existing tools
   - Python?

2. Development in Python

3. OK, but does anyone use it?
   - EEG analysis for epilepsy
   - JPL: Mars mission data visualization
   - Multiwavelets for PDEs
   - NetworkX: complex networks
   - MayaVi: customizable data visualization
   - Sage

4. IPython: Interactive Python

5. Parallel and distributed computing

6. IPython in this context
Multiple implementations of the Virtual Machine:

- **CPython**: pure C, ‘reference’
- **IronPython**: .NET
- **Jython**: Java

Their threading behaviors differ, I’ll focus on CPython

Native threads supported, but of **limited use**.

**Global interpreter lock (GIL)**: only **one** python thread.

- Threads only good for i/o bound tasks.
- Mostly useless for CPU-bound ones.

**No language-specific primitives** for parallelism.
Tools and projects
http://wiki.python.org/moin/ParallelProcessing

- Built-in `threading` module
- Built-in `multiprocessing` module: similar API to threads.
- `pyro`: remote proxy objects
- Google’s `Unladen Swallow`:
  - LLVM-based optimizations to CPython
  - Long-term outlook: full JIT and GIL removal
- `ParallelPython`: SMP and clusters, high level interfaces
- `mpi4py`: excellent MPI bindings for Python
- `IPython`: what are we doing differently?
Outline

1. Scientific Computing
   - Existing tools
   - Python?

2. Development in Python

3. OK, but does anyone use it?
   - EEG analysis for epilepsy
   - JPL: Mars mission data visualization
   - Multiwavelets for PDEs
   - NetworkX: complex networks
   - MayaVi: customizable data visualization
   - Sage

4. IPython: Interactive Python

5. Parallel and distributed computing

6. IPython in this context
Python: static vs dynamic types

Dynamically typed languages are great for interactive use

- Minimal typing, maximal effect
- Sketches of code are enough to get execution
- Great for data exploration and algorithm development
- No need to babysit the compiler

For the purposes of this discussion:

Scientific computing ⇔ exploratory computing

Can you type templated C++ off the top of your head at a shell?
We want

- High-level benefits of Python
- Maintain good interactive, exploratory workflows
- Easy debugging
- Impromptu visualization
- Collaboration
- Exceptions?
  - Asynchronous ones (signals)?

But we also want...

- Scale: serial, multicore, clusters, supercomputers...
- Play well with MPI
- OpenMP... (Cython?)
IPython in this context

IPython? Wasn’t that a shell?

- Brian Granger: Cal Poly San Luis Obispo (then at Santa Clara U).
- Min Ragan-Kelley: Santa Clara undergrad, now UCB grad in AS&T
- Brian: Python bindings for Apple’s XGrid system
- The abstractions needed were the same as those to make IPython useful in GUIs, browsers, embedded in other projects, etc.
- We joined forces around 2003
  - They lead the charge on much of the parallel work.
- Min’s senior thesis: early prototype
Goals of IPython in this context

- Parallel computing: **fully interactive**
  - development, debugging, testing, execution, monitoring,...
- Easy things should be **easy**, difficult things **possible**
- Make parallel computing **collaborative**
- More dynamic model for **load balancing** and **fault tolerance**
- Seamless integration with other tools: plotting/visualization, system shell.
- Also want to keep the benefits of traditional approaches:
  - Should integrate with threads/MPI if appropriate
  - Should be easy to integrate compiled code and libraries
- Support many types of parallelism
Network-aware IPython

The "IPython VM"

- Kernel Controller
  - Non-blocking
  - World-visible

Private link

Kernel Engine

MAY BLOCK!

Twisted

Networks
IPython’s architecture
What does IPython offer here?

- **Easy reuse** and distribution of existing serial (‘normal’) codes.
- High-level abstractions for ‘embarrassingly parallel’ problems.
  - Direct execution of code over the network: multiengine interface.
  - Out-of-the box *task farming* tools: task interface.
- Task farming system is “low-latency” *(not in the Myrinet sense...)*
  - can be integrated into more complex codes.
- Implement any approach to parallelism you want:
  - Synchronous or asynchronous execution of code on nodes.
  - Task farming.
  - Traditional Message Passing (MPI).
  - Integrate hybrid codes.
  - BYO.
- **Actively developed.**
Use cases

Multicore

- Run controller, engines, client on the same system.
- Zero setup
- Can continue using an active IPython session

$ ipcluster local -n 4

Cluster with MPI: existing C/Fortran MPI code

- Wrap for python use (mpi4py)
- Start controller and engines with MPI/PBS support
- Connect to Controller from IPython session and use interactively

$ ipcluster mpirun -n 64 --mpi=mpi4py
$ ipcluster pbs -n 64 --pbs-script=myscript.sh
### Use cases

**Multicore**
- Run controller, engines, client on the same system.
- Zero setup
- Can continue using an active IPython session

```
$ ipcluster local -n 4
```

**Cluster with MPI: existing C/Fortran MPI code**
- Wrap for python use (mpi4py)
- Start controller and engines with MPI/PBS support
- Connect to Controller from IPython session and use interactively

```
$ ipcluster mpirun -n 64 --mpi=mpi4py
$ ipcluster pbs -n 64 --pbs-script=myscript.sh
```
Interactive distributed factorization of symbolic polynomials

In [1]: import sympy

In [2]: def factorit(n):
   ...:     x = sympy.var('x')
   ...:     return sympy.factor(x**n-1,x)
   ...:

In [3]: factorit(5)
Out[3]: -(1 - x)*(1 + x + x**2 + x**3 + x**4)

In [4]: from IPython.kernel import client
In [5]: mec = client.MultiEngineClient()
In [6]: mec.execute('import sympy')
In [7]: f = mec.map(factorit, range(100, 110))

In [11]: f[0]
Out[11]: -(1 + x)*(1 + x**2)*(1 - x)*(1 + x + x**2 + x**3 + x**4)*(1 - x + x**2 - x**3 + x**4)*(1 + x**5 + x**10 + x**15 + x**20)*(1 - x**5 + x**10 - x**15 + x**20)*(1 - x**10 + x**20 - x**30 + x**40)*(1 - x**2 + x**4 - x**6 + x**8)

Replace MultiEngineClient by TaskClient to get load balancing.
Vision: visual programming
Michel Sanner, Scripps Institute, La Jolla.
Vision
A generic framework for visual programming
IPVision: distributed computing
Brian Granger, Michel Sanner, Jose Unpingco, Ananth Devulapalli [Ohio Supercomputing Center/OSU], Min Ragan-Kelley
IPVision

- Easy access to distributed computing for non-experts
- Vision allows any python library to be accessed (after wrapping)
- Users get to use all their cores...
Some technical notes

- Networking: **Twisted**
  - High-level interfaces: no need to learn Twisted.
- RPC: Twisted’s **foolscap**
- **Security**: foolscap supports SSL (**pyOpenSSL**) and a capabilities model.
  - Review/improvements welcome, we’re not security experts!
- MPI support is there, use **mpi4py** bindings.
- Integration with queuing systems, better process control coming...
Interactive + parallel = toxic combination

- Many contradictory design situations
- Performance metrics often must be rethought
  - Latency: human latency!
- This is not MPI on Myrinet or a Cray backplane...
  - 16 core computer: less than $5K...
  - people will want to use those 16 cores under their desk.
- Cultural/technical challenges in shared facilities
  - Problems with one-at-a-time queuing schedulers
  - Using interactive queues?
    - Overcommitting: catastrophical on memory collisions
    - Undercommitting: terribly wasteful
- ??? We’re far from done with this problem...
Generic scientific computing in Python

- Rapidly growing
  - ~10 Google Summer of Code Scientific Python projects.
  - SIAM 2008 annual meeting: 12-talk minisymposium
  - SIAM 2009 CSE meeting: 12-talk minisymposium
- Collaborative effort: by scientists, for scientists.
- An open stack supports truly reproducible computational research.

Parallel work

- Plenty of opportunities: open platform for new work.
- In IPython, we want contributors!
- UCB neuroscience: we’ll be using these tools.

UC Berkeley

- Strong contingent in Neuroscience (multiple labs)
- Astronomy: Josh Bloom
- More, I’m sure... come to the meetings!

Talk to us!
Wrapup

Generic scientific computing in Python

- Rapidly growing
  - ~10 Google Summer of Code Scientific Python projects.
  - SIAM 2008 annual meeting: 12-talk minisymposium
  - SIAM 2009 CSE meeting: 12-talk minisymposium

- Collaborative effort: by scientists, for scientists.
- An open stack supports truly reproducible computational research.

Parallel work

- Plenty of opportunities: open platform for new work.
- In IPython, we want contributors!
- UCB neuroscience: we’ll be using these tools.

UC Berkeley

- Strong contingent in Neuroscience (multiple labs)
- Astronomy: Josh Bloom
- More, I’m sure... come to the meetings!

Talk to us!
Wrapup

Generic scientific computing in Python
- Rapidly growing
  - \(\sim 10\) Google Summer of Code Scientific Python projects.
  - SIAM 2008 annual meeting: 12-talk minisymposium
  - SIAM 2009 CSE meeting: 12-talk minisymposium
- Collaborative effort: by scientists, for scientists.
- An open stack supports truly reproducible computational research.

Parallel work
- Plenty of opportunities: open platform for new work.
- In IPython, we want contributors!
- UCB neuroscience: we’ll be using these tools.

UC Berkeley
- Strong contingent in Neuroscience (multiple labs)
- Astronomy: Josh Bloom
- More, I’m sure... come to the meetings!

Talk to us!
Special acknowledgments

- **Brian Granger** - Physics, Cal Poly San Luis Obispo.
- **Min Ragan-Kelley** - Applied Science and Technology, UC Berkeley.
If you are interested

Berkeley Py4Science meetings:
Wednesdays 4pm, Evans 508-20

https://cirl.berkeley.edu/view/Py4Science

SciPy Conference:
Caltech, August 18-23 2009

http://conference.scipy.org

Thank you for your patience!
Any questions?