Python & Scientific Computing
Leading the charge for open source, high-level tools

Fernando Pérez
<Fernando.Perez@berkeley.edu>

Helen Wills Neuroscience Institute, U.C. Berkeley

Google
November 13, 2008
Let me ask you a question first

How can I distinguish $f$ and $y$ without calling them?

In [1]: def f():
       .......:     return 1
In [2]: def y():
       .......:     yield 1

# They are both callable:
In [3]: callable(f), callable(y)
Out[3]: (True, True)

# Both functions:
In [4]: type(f), type(y)
Out[4]: (<type 'function'>, <type 'function'>)

# One returns a generator:
In [5]: type(f()), type(y())
Out[5]: (<type 'int'>, <type 'generator'>)

# Can this be known about $y$ without calling it?

Answer (thanks Alex!):
Outline

1. Scientific Computing
   - General context
   - Existing tools
   - Python?

2. Numpy and Scipy

3. OK, but does anyone use it?
   - IPython
   - EEG analysis for epilepsy
   - JPL: Mars mission data visualization
   - PMV: structural bioinformatics
   - MayaVi: customizable data visualization
   - Multiwavelets for PDEs
   - Sympy
   - Sage

4. Community

5. Where to now?
Inherently **exploratory**
- Rarely have up-front requirements.
- **Compute-think-compute** cycle.

**Very high level...**
- Yet performance can matter a lot (**can**, not necessarily **does**)

**Scientists are (supposedly) experts...**
- at things other than programming
- C++? Hah, funny.

**We know little about software engineering**
- ... and many don’t even care at all!
- (Code rarely gives you grants or tenure).
A bit of history

**Fortran**
- Workhorse of the field.
- Well known quantity.
- Excellent optimization support.

**C**
- $y = x^n$: Civilized exponentiation.
- $c_i = \sum_j a_{ij} b_j$: Proper multidimensional arrays.
- $z = x + iy$: Complex numbers.
- Nobody needs that stuff!

**C++**
- We (scientists) shouldn’t be allowed near a C++ compiler.
- No **standardized** support for numerical needs.
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!
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?**
  - Very popular, and for good reasons!
    - Interactivity, visualization, extensive libraries.
    - Unpleasant for large-scale programs and non-mathematical tasks.
    - Expensive, proprietary: lock-in.

- **Open source counterparts:**
  - GNU **Octave** (Matlab clone)
  - **Maxima**: symbolics.
  - They use **gnuplot** for visualization.

- **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?
  Very popular, and for good reasons!
  - Interactivity, visualization, extensive libraries.
  - Unpleasant for large-scale programs and non-mathematical tasks.
  - Expensive, proprietary: lock-in.

- Open source counterparts:
  - GNU Octave (Matlab clone)
  - Maxima: symbolics.
  - They use gnuplot for visualization.

- 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 in this context

- **Free** (BSD license): Speech and Beer.
- **Portable**.
- **Interactive** interpreter provided.
- **Simple**: even scientists can master it.
- Clean object oriented model, but **not mandatory**.
- Rich built-in types: lists, sets, dictionaries (hash tables), strings, ...
- Very comprehensive standard library (**batteries included**)
- Support for IDL/Matlab-like **arrays** (NumPy)
- Easy to **wrap** existing C, C++ and FORTRAN codes.
- Flexible:
  - High level...
  - Low level... **as needed**!
Outline

1. Scientific Computing
   - General context
   - Existing tools
   - Python?

2. Numpy and Scipy

3. OK, but does anyone use it?
   - IPython
   - EEG analysis for epilepsy
   - JPL: Mars mission data visualization
   - PMV: structural bioinformatics
   - MayaVi: customizable data visualization
   - Multiwavelets for PDEs
   - Sympy
   - Sage

4. Community

5. Where to now?
Is Python good enough for scientific computing?

Efficient numerical processing?

- Compiler-specific numbers needed: ints (32, 64-bit), floats, etc.
- Homogeneous arrays of such elements.
- Easy arithmetic on entire arrays that is efficient.
- Comprehensive math library to operate on arrays.
- Common linear algebra support.

Needed for:

- Mathematics
- Image processing
- Data analysis
- ... Just about anything remotely connected to scientific computing.
A tiny bit of history

1990s-2004: Numeric
- Started by Jim Hugunin, MIT grad student.
- Developed by many: national labs, academia, industry.
- Fast and light C code.
- Difficult to maintain and extend.

2004: Numarray
- Perry Greenfield and team (Hubble Space Telescope)
- Clean codebase, new ideas.
- Many new features and documentation.
- Some performance issues lingered

2005-future: NumPy: unification effort
- Led by Travis Oliphant, with Perry’s full support.
- Best of Numeric and Numarray, into a new codebase
- The whole community rallied behind the effort.

Today: only NumPy for any new code!
NumPy: key ideas

- A flexible, efficient, **multidimensional array** object
- **Homogeneous** elements (almost)
  - Supports all native types (ints, floats, etc)
  - Arbitrary user-defined types of fixed size
  - Arbitrary Python objects can also be stored

- High-level syntax
  - \( c=a+b \rightarrow c_{ij} = a_{ij} + b_{ij} \ \forall ij \)

- **Math library** that operates on arrays
  - \( y=\sin(x) \rightarrow y_{ijk} = \sin(x_{ijk}) \ \forall ijk \)
  - You can define your own `ufuncs` (universal functions)

- Basic scientific functionality
  - Linear algebra
  - FFTs
  - Random number generation
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.

Image credit: T. Oliphant, Enthought Inc.
```
>>> a[0,3:5]
array([3,4])

>>> a[4:,4:]
array([[44, 45],
       [54, 55]])

>>> a[:,2]
array([2, 22, 52])

>>> a[2::2,::2]
array([[20, 22, 24],
       [40, 42, 44]])
```

Slicing does not create copies of the array’s contents
```
>>> a[(0,1,2,3,4),(1,2,3,4,5)]
array([ 1, 12, 23, 34, 45])

>>> a[3:, [0, 2, 5]]
array([[30, 32, 35],
       [40, 42, 45],
       [50, 52, 55]])

>>> mask = array([1,0,1,0,0,1], dtype=bool)
>>> a[mask, 2]
array([2,22,52])
```

Unlike slicing, fancy indexing creates copies instead of views into original arrays.
NumPy: broadcasting

Image credit: E. Jones, Enthought Inc.
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
Using today’s scientific codes 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 today’s scientific codes 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
   - General context
   - Existing tools
   - Python?

2. Numpy and Scipy

3. OK, but does anyone use it?
   - IPython
   - EEG analysis for epilepsy
   - JPL: Mars mission data visualization
   - PMV: structural bioinformatics
   - MayaVi: customizable data visualization
   - Multiwavelets for PDEs
   - Sympy
   - Sage

4. Community

5. Where to now?
IPython: efficient interactive workflows
Matlab/IDL-style

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'$J_0$');
In [12]: axhline(0, color='green', label='nolegend_');
In [13]: title(r'Verify $J_0(x) = \frac{1}{x} \int_{0}^{\pi} \cos(x \sin \phi) \, d\phi$');
In [14]: xlabel('x');
In [15]: legend();
In [16]: imshow(numpy.random.randn((32, 32)))
IPython: GUI integration
Control Wx, Qt, GTK applications

```
bgranger@pinch> ipython --no-ipythonython
Python 2.4.3 (#1, Apr 7 2005, 10:54:33)
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.

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)
```
Think of Python as ‘the CPU’.
IPython abstracts them over the network.
Use interactively or not.
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
From: Name Elided <nameelided@jpl.nasa.gov>
Date: Oct 2, 2007 7:15 PM
Subject: Fwd: matplotlib bug numbers
To: John Hunter <jdh2358@gmail.com>

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 important 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:
Summary of the current navigation team activities and modelling (data used, event times etc):
PMV: the Python Molecule Viewer
Michel F. Sanner, Molecular Biology Department, The Scripps Research Institute.
FluidLab: a MayaVi based CFD visualization tool
K. Julien, M. Franklin, P. Schmitt, B. Barrow, F.P.
Fast application of integral kernels for PDEs.
Complex algorithm: beyond pure numerics.
Fortran, C, run-time generated C++, Cython.
High performance tool with exploratory interactive interface.
These commands were executed:
```python
>>> from __future__ import division
>>> from sympy import *
>>> x, y, z = symbols('xyz')
>>> k, m, n = symbols('k m n', integer=True)
>>> f = Function('f')
```

Documentation can be found at http://sympy.org/

```python
In [1]: integrate(sin(x))
Out[1]: -cos(x)

In [2]: integrate(sin(x)/x)
Out[2]:
\[
\int \frac{\sin(x)}{x} \, dx
\]

In [3]: limit(sin(x)/x, x, 0)
Out[3]: 1

In [4]: expand(((x+y)**3/(x-y)))
Out[4]:
\[
\frac{x^3}{x-y} + \frac{3x^2y}{x-y} + \frac{3xy^2}{x-y} + \frac{y^3}{x-y}
\]

In [5]: (sin(x)/x).series(x, 0, 5)
Out[5]:
\[
1 - \frac{x}{6} + \frac{x^4}{120} + 0(x^5)
\]
```

In [6]:
Sage: open source mathematics
Python: Batteries Included
Who else?

Academia

- **PyRAF**: Astronomical data analysis (Hubble Space Telescope).
- **PETSc4py**: Parallel PDE solvers (Argonne National Lab).
- **PyTrilinos**: Parallel solvers (Sandia National Lab).
- **DANSE**: Spallation Neutron Source (ORNL/Caltech).
- **CDAT**: Climate Data Analysis Tools (Lawrence Livermore).
- **OOF2**: Finite Element Analysis of Microstructures (NIST).

Industry

- **InteractiveSupercomputing.com**: Python interface to their proprietary backend.
- **Numenta**: pattern recognition algorithms (Jeff Hawkins-Palm)

Lots more

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

Academia

- **PyRAF**: Astronomical data analysis (Hubble Space Telescope).
- **PETSc4py**: Parallel PDE solvers (Argonne National Lab).
- **PyTrilinos**: Parallel solvers (Sandia National Lab).
- **DANSE**: Spallation Neutron Source (ORNL/Caltech).
- **CDAT**: Climate Data Analysis Tools (Lawrence Livermore).
- **OOF2**: Finite Element Analysis of Microstructures (NIST).

Industry

- **InteractiveSupercomputing.com**: Python interface to their proprietary backend.
- **Numenta**: pattern recognition algorithms (Jeff Hawkins-Palm)

Lots more

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

Academia

- **PyRAF**: Astronomical data analysis (Hubble Space Telescope).
- **PETSc4py**: Parallel PDE solvers (Argonne National Lab).
- **PyTrilinos**: Parallel solvers (Sandia National Lab).
- **DANSE**: Spallation Neutron Source (ORNL/Caltech).
- **CDAT**: Climate Data Analysis Tools (Lawrence Livermore).
- **OOF2**: Finite Element Analysis of Microstructures (NIST).

Industry

- **InteractiveSupercomputing.com**: Python interface to their proprietary backend.
- **Numenta**: pattern recognition algorithms (Jeff Hawkins-Palm)

Lots more

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

1. Scientific Computing
   - General context
   - Existing tools
   - Python?

2. Numpy and Scipy

3. OK, but does anyone use it?
   - IPython
   - EEG analysis for epilepsy
   - JPL: Mars mission data visualization
   - PMV: structural bioinformatics
   - MayaVi: customizable data visualization
   - Multiwavelets for PDEs
   - Sympy
   - Sage

4. Community

5. Where to now?
Community growth: www.scipy.org

<table>
<thead>
<tr>
<th>Rank</th>
<th>Country</th>
<th>Visits</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>United States</td>
<td>252,208</td>
<td>38.12%</td>
</tr>
<tr>
<td>2.</td>
<td>Germany</td>
<td>59,515</td>
<td>9.00%</td>
</tr>
<tr>
<td>3.</td>
<td>United Kingdom</td>
<td>41,063</td>
<td>6.21%</td>
</tr>
<tr>
<td>4.</td>
<td>France</td>
<td>40,354</td>
<td>6.10%</td>
</tr>
<tr>
<td>5.</td>
<td>Canada</td>
<td>24,688</td>
<td>3.73%</td>
</tr>
<tr>
<td>6.</td>
<td>Japan</td>
<td>19,378</td>
<td>2.93%</td>
</tr>
<tr>
<td>7.</td>
<td>Italy</td>
<td>15,151</td>
<td>2.29%</td>
</tr>
<tr>
<td>8.</td>
<td>Australia</td>
<td>13,591</td>
<td>2.05%</td>
</tr>
<tr>
<td>9.</td>
<td>China</td>
<td>13,452</td>
<td>2.03%</td>
</tr>
<tr>
<td>10.</td>
<td>Netherlands</td>
<td>11,799</td>
<td>1.78%</td>
</tr>
</tbody>
</table>
Community: SIAM 2008 summer meeting

- Python and Sage: Open Source Scientific Computing
- 3-part minisymposium, very well attended
- Selected for annual highlights

This talk was part of a session on the programming languages Python and Sage. Perez said that Python is free, simple, and extremely portable, with rich built-in data types such as sets and lists. He gave many examples to show the power and flexibility of Python, including examples from computational fluid dynamics, brain surgery (at right), and a Mars mission. While Python is a general programming language and not a mathematical language (which Perez counts as "a feature, not a bug"), Sage is geared toward mathematics. Sage began in 2006 and is now up to five million lines of open-source code. Although Sage is based on Python, each can be used within the framework of the other. (Image: Final location of epileptic foci for surgery by John Hunter, Ph.D., Pediatric Neurology, University of Chicago.)
7th Annual Python in Science Conference

Pasadena, CA, August 19-24, 2008: the most in-depth, practical Python scientific programming event.

The SciPy conference, 7th Python in Science conference, was held in Pasadena this summer. The conference is now over. You can join us next year.

**Procedings:** The conference proceedings are available.

This conference provides a unique opportunity to learn and affect what is happening in the realm of scientific computing with Python. Attendees have the opportunity to review the available tools and how they apply to specific problems. By providing a forum for developers to share their Python expertise with the wider commercial, academic, and research communities, this conference fosters collaboration and facilitates the sharing of software components, techniques and a vision for high level language use in scientific computing.
Community: various events

- **Sage/Scipy joint sprints**
  - March 2008 @ Enthought in Austin, TX
  - November 2008 @ UT Austin.

- **Py4Science@Berkeley:**
  - https://cirl.berkeley.edu/view/Py4Science.
  - Meet every 2 weeks, hands-on, science-focused.

- **Annual SciPy conference:** August or September.

- **SIAM CSE09:** March 2-6 2009, Miami.
  - Minisymposium about Python in computational sciences.
  - Org: Randy LeVeque (U. Washington), Hans-Petter Langtangen (Simula, Oslo), FP (UC Berkeley).

- **Sage Days@Berkeley:** MSRI, March 2009, Algebraic Geometry
  - http://www.msri.org/calendar/workshops/WorkshopInfo/502/show_workshop
  - Sage holds many ‘Sage Days’! See their site.
Outline

1. Scientific Computing
   - General context
   - Existing tools
   - Python?

2. Numpy and Scipy
3. OK, but does anyone use it?
   - IPython
   - EEG analysis for epilepsy
   - JPL: Mars mission data visualization
   - PMV: structural bioinformatics
   - MayaVi: customizable data visualization
   - Multiwavelets for PDEs
   - Sympy
   - Sage

4. Community

5. Where to now?
PEP 225 feedback: new operators in Python
Summary: +1

- Most needed: multiplication.
  - `c = a * b` : element-wise \( c_{ij} = a_{ij} b_{ij} \)
  - `c = a ~* b` : matrix product \( c_{ij} = \sum_k a_{ik} b_{kj} \)

- Also exponentiation, logical, and perhaps division (for Matlab’s \( A \setminus B \))

- NumPy’s matrix class: the source of no end of trouble.

- This would make our lives easier.

https://cirl.berkeley.edu/fperez/static/numpy-pep225
Challenges, weaknesses?

- **Optimization of bottlenecks.**
  - `numpy.f2py`: Fortran
  - `scipy.weave`: inline C/C++, including arrays (Blitz++)
  - **Cython**! Very exciting development.

- **Documentation**: `sphinx`
  - Excellent, searchable, uniform docs.
  - IPython, Matplotlib, Numpy, Scipy, Sympy, Sage: all switched to it.
  - Thanks!

- **Testing**: `nose` is really helping.
  - Lightweight, no-hassle test writing.
  - Proper generative test support.
  - Closer support in python (like sphinx) would be welcome.

- **Build, install, deploy.**
  - The neverending nightmare.
Challenges, weaknesses?

- Optimization of bottlenecks.
  - `numpy.f2py`: Fortran
  - `scipy.weave`: inline C/C++, including arrays (Blitz++)
  - **Cython**! Very exciting development.

- Documentation: **sphinx**
  - Excellent, searchable, uniform docs.
  - IPython, Matplotlib, Numpy, Scipy, Sympy, Sage: all switched to it.
  - Thanks!

- Testing: **nose** is really helping.
  - Lightweight, no-hassle test writing.
  - Proper generative test support.
  - Closer support in python (like sphinx) would be welcome.

- Build, install, deploy.
  - The neverending nightmare.
Optimization of bottlenecks.

- `numpy.f2py`: Fortran
- `scipy.weave`: inline C/C++, including arrays (Blitz++)
- `Cython`! Very exciting development.

Documentation: `sphinx`

- Excellent, searchable, uniform docs.
- IPython, Matplotlib, Numpy, Scipy, Sympy, Sage: all switched to it.
- Thanks!

Testing: `nose` is really helping.

- Lightweight, no-hassle test writing.
- Proper generative test support.
- Closer support in python (like sphinx) would be welcome.

Build, install, deploy.

- The neverending nightmare.
Challenges, weaknesses?

- Optimization of bottlenecks.
  - `numpy.f2py`: Fortran
  - `scipy.weave`: inline C/C++, including arrays (Blitz++)
  - **Cython**! Very exciting development.

- Documentation: **sphinx**
  - Excellent, searchable, uniform docs.
  - IPython, Matplotlib, Numpy, Scipy, Sympy, Sage: all switched to it.
  - Thanks!

- Testing: **nose** is really helping.
  - Lightweight, no-hassle test writing.
  - Proper generative test support.
  - Closer support in python (like sphinx) would be welcome.

- Build, install, deploy.
  - **The neverending nightmare.**
Distutils, you say?
OK, we’re brave, we go in...
Image credit: http://xkcd.com/246

AND OVER THERE WE HAVE THE LABYRINTH GUARDS. ONE ALWAYS LIES, ONE ALWAYS TELLS THE TRUTH, AND ONE STABS PEOPLE WHO ASK TRICKY QUESTIONS.
Well, there’s setuptools, right? It’s called easy_install, so it should be easy...
Installation and deployment

Probably the most problematic issue for wider acceptance of Python in scientific computing at large.

- We (scientists) don’t know how to solve this...
- Though the brave ones do help:
  - Dave Peterson (Enthought)
  - Gaël Varoquaux (Neuroimaging, Neurospin, Paris)
  - David Cournapeau (Signal Processing, U. Kyoto)

- Some tools
  - `numpy.distutils` (Pearu Peterson and others)
  - `numscons` (D. Cournapeau)

- Progress in self-contained distributions
  - Enthought Python Distribution (EPD).
  - Python(x,y).
  - Sage.

- A good solution on this front would **really** help us.
- I know it’s a hard problem and a can of worms...
Python is **really growing** in scientific computing.

Collaborative effort: *by scientists, for scientists.*

An open stack supports truly **reproducible computational research.**

**Come on board!**

- Provide Python bindings to your own research codes.
- Help an existing tool improve *(there’s plenty to do!)*
- Just a few places you can help:
  
  - [http://www.scipy.org](http://www.scipy.org)
  - [http://ipython.scipy.org](http://ipython.scipy.org)
  - [http://matplotlib.sf.net](http://matplotlib.sf.net)
  - [http://sympy.org](http://sympy.org)
  - [http://sagemath.org](http://sagemath.org)

Or find a project that interests you at

[http://www.scipy.org/Topical_Software](http://www.scipy.org/Topical_Software)
Python is **really growing** in scientific computing.
Collaborative effort: **by scientists, for scientists.**
An open stack supports truly **reproducible computational research.**

**Come on board!**
- Provide Python bindings to your own research codes.
- Help an existing tool improve *(there’s plenty to do!)*
- Just a few places you can help:

  http://www.scipy.org
  http://ipython.scipy.org
  http://matplotlib.sf.net
  http://sympy.org
  http://sagemath.org

Or find a project that interests you at

http://www.scipy.org/Topical_Software
That’s it, folks.
Any questions?

Thank You!