Python
A Scalable Tool for Scientific and Mathematical Computing

Fernando Pérez
<Fernando.Perez@berkeley.edu>

Helen Wills Neuroscience Institute, U.C. Berkeley

SIAM Meeting, San Diego
July 9, 2008

- **Part I (MS 60)**
  Today 10:30 AM - 12:30 PM, Room: Town & Country

- **Part II (MS 73)**
  Today 4:00 PM - 6:00 PM, Room: Town & Country

- **Part III (MS 81)**
  Tomorrow 10:30 AM - 12:30 PM, Room: San Diego
1. **Scientific Computing**
   - Existing tools
   - Python?

2. Development in Python

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

1 Scientific Computing
   - Existing tools
   - Python?

2 Development in Python

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

1. Scientific Computing
   - Existing tools
   - Python?

2. Development in Python

3. OK, but does anyone use it?
   - EEG analysis for epilepsy
   - Multiwavelets for PDEs
   - JPL: Mars mission data visualization
   - PMV: structural bioinformatics
   - MayaVi: customizable data visualization
   - Sage
   - IPython
1. **Scientific Computing**
   - Existing tools
   - Python?

2. Development in Python

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

- **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.
- 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
   - Multiwavelets for PDEs
   - JPL: Mars mission data visualization
   - PMV: structural bioinformatics
   - MayaVi: customizable data visualization
   - Sage
   - IPython
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

```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.
Python: ‘Scalable’
What do I mean by that?

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.
... 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 (MS 73, today at 4:00)
- 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 (MS 73, today at 4:00)
- 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!
Writing a new code?

- Avoid whole programs in C/Fortran: **you are optimizing globally!**
  - Prototype the problem in Python (get the algorithm right first)
    - using fast libraries.
  - **Profile** the hotspots.
  - Optimize **only those**, if needed.

- The result may be production-ready: **less throw-away prototypes**.
  - Make your code available as a library for interactive use.
  - Integrate plotting, visualization, logging, ..., into your objects.
Writing a new code?

- Avoid whole programs in C/Fortran: you are optimizing globally!
  - Prototype the problem in Python (get the algorithm right first)
    - using fast libraries.
  - Profile the hotspots.
  - Optimize only those, if needed.

- The result may be production-ready: less throw-away prototypes.
  - Make your code available as a library for interactive use.
  - Integrate plotting, visualization, logging, ..., into your objects.
Outline

1. Scientific Computing
   - Existing tools
   - Python?

2. Development in Python

3. OK, but does anyone use it?
   - EEG analysis for epilepsy
   - Multiwavelets for PDEs
   - JPL: Mars mission data visualization
   - PMV: structural bioinformatics
   - MayaVi: customizable data visualization
   - Sage
   - IPython
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
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.
- See MS91 for continuation of this work in quantum mechanics.
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:
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.
Sage: open source mathematics

```python
show(graphs.CubeGraph(5).plot3d())

show(graphs.CubeGraph(6).plot3d())
```

```python
show(plot(sin(x^2)+x, -pi, pi, hue=0.7, thickness=3))
```
IPython: Matlab/IDL-like interactive use

```
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*pi-y) + cos(3*pi+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)
In [9]:
```
Think of Python as ‘the CPU’.
IPython abstracts them over the network.
Use interactively or not.
Who else?

Academia

- **PyRAF**: Astronomical data analysis (Hubble Space Telescope).
- **PyTrilinos**: Parallel solvers (Sandia National Lab’s - MS 81).
- **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)
- **Google/YouTube**: Internally, new AppEngine web service platform.

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 Lab’s - MS 81).
- **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)
- **Google/YouTube:** Internally, new AppEngine web service platform.

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 Lab's - MS 81).
- **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)
- **Google/YouTube**: Internally, new AppEngine web service platform.

**Lots more**

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

What’s in it for you?
- Hopefully a very useful tool.
- Provide Python bindings to your own research codes.
  - ... and we all benefit from the resulting network effect.
- Help an existing tool improve (there’s plenty to do!)
Python is rapidly growing in scientific computing.
Collaborative effort: by scientists, for scientists.
An open stack supports truly reproducible computational research.

What’s in it for you?

- Hopefully a very useful tool.
- Provide Python bindings to your own research codes.
  - ... and we all benefit from the resulting network effect.
- Help an existing tool improve (there’s plenty to do!)
If you are really interested

EuroSciPy Conference:
Leipzig, July 26-27

SciPy Conference:
Caltech, August 19-24 2008
http://conference.scipy.org
We’re done!
Writing a new code?

- Avoid whole programs in C/Fortran: you are optimizing globally!
  - Prototype the problem in Python (get the algorithm right first)
    - using fast libraries.
  - Profile the hotspots.
  - Rewrite those with low-level tools.

- The result may be production-ready: less throw-away prototypes.
  - Make your code available as a library for interactive use.
  - Integrate plotting, visualization, logging, ..., into your objects.
Writing a new code?

- Avoid whole programs in C/Fortran: you are optimizing globally!
  - Prototype the problem in Python (get the algorithm right first)
    - using fast libraries.
  - Profile the hotspots.
  - Rewrite those with low-level tools.

- The result may be production-ready: less throw-away prototypes.
  - Make your code available as a library for interactive use.
  - Integrate plotting, visualization, logging, ..., into your objects.