Python for Scientific Computing at CSE09 *

Fernando Pérez
University of California, Berkeley

Hans Petter Langtangen
Simula Research Laboratory

Randy LeVeque
University of Washington, Seattle

January 26, 2010

For the SIAM CSE09 meeting, we organized the 3-part minisymposium *Python for Scientific Computing*, where a mix of speakers from universities, government research laboratories and industry showed why they chose the Open Source Python programming language and how they use it in their research and teaching. The sessions were well attended, as was a similar 3-part minisymposium at the annual meeting last summer.

A flexible tool

Over the last few years, Python has experienced tremendous growth as the tool of choice for high-level scientific computing. It offers an effective mix of interactive and exploratory development, direct access to libraries for many different tasks, and interfaces with high-performance numerical libraries in Fortran, C or C++.

Scientists coming from computational environments such as Matlab^{TM} or IDL^{TM} will find that Python, with a few free add-on packages, provides basic capabilities similar to these systems: numerical arrays with syntactic support for arithmetic and mathematical operations, a comprehensive library of common algorithms (linear algebra, FFTs, numerical integration, optimization, special functions, etc.), interactive control of data visualization, publication-quality plotting and modules to interface with codes written in numerous other programming languages. But what is attracting many scientists to the language is a combination of flexibility, expressive power and development possibilities that is unmatched by commercial tools.

Python was originally designed as a general-purpose language with an emphasis on a very clear and readable syntax, high-level constructs that did not impede access to low-level resources, robust error handling and portability. It ships with a comprehensive standard library that covers many common tasks, from text processing to network protocols or database access. In addition, free projects exist that add support for most common computational needs, as well as providing bindings to use a wide collection of Fortran, C and C++ libraries. It is used today extensively in industry by companies like Google (who employs Python's creator, Guido van Rossum), but also by most federal research agencies in the USA, as was illustrated by the presence of speakers from several National Laboratories and NIST. Python offers zero licensing cost and hence no license manager hassles (important considerations when you need to run parallel codes on hundreds or thousands of processors or use cloud computing services), as well as being extremely portable: it can be run equally on a cell phone as on the nation's largest supercomputers.

1
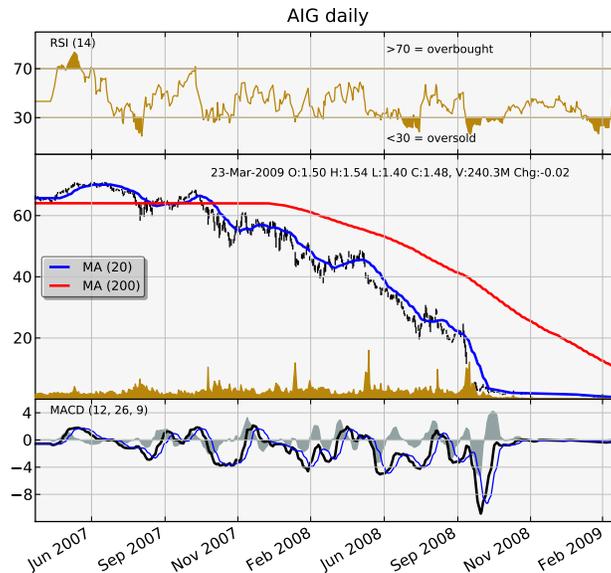
Figure 1: AIG's recent stock performance, elegantly summarized by matplotlib.

A mix of topics, from research to education

The talks covered both general purpose tools and domain-specific projects (all materials are available online[1]). A group of speakers focused on interactive, exploratory computing and data visualization: Fernando Pérez from UC Berkeley discussed the *IPython* system of components for interactive computing, and Brian Granger from Cal Poly San Luis Obispo covered IPython's applications for high-level parallel computing as well as the design of distributed data structures. John Hunter from Tradelink, the lead developer of the *matplotlib* plotting package, presented an overview and hands-on demonstration of the project, whose goal is to provide exceptionally high quality two-dimensional plots (see Figure 1). Hank Childs from Lawrence Livermore National Laboratory followed this up with a discussion of *VisIt*, a package for the analysis and visualization of large-scale three-dimensional data sets such as those produced using adaptive mesh refinement on massively parallel machines. While VisIt has its own GUI interface, it also has a Python interface that makes it particularly convenient for incorporating in other projects. Figure 2 shows a visualization done with VisIt, which uses the high-quality VTK 3D graphics library, combining multiple rendering options for displaying elevation data of Mount St. Helens. Figure 3 shows a visualization done with *Mayavi2*, a related Python tool based on the same toolkit as VisIt, the high-quality VTK 3D graphics library.

Another group of talks covered tools with a focus on performance: Andreas Klöckner, from Brown University, showed how to easily access the capabilities of modern high-performance graphics cards for numerical computing with his *PyCuda* library, while Pearu Peterson from the Institute of Cybernetics at Estonia's Tallinn University of Technology presented his research on the algorithmic and data structure problems involved in designing *sympycore*, a fast library for symbolic computing in Python. A presentation by Lisandro Dalcín from the Argentinean CIMEC research laboratory and Brian Granger covered the *mpi4py* library that permits the development of MPI codes in Python. mpi4py allows both the calling of MPI primitives in pure Python and accessing C or Fortran MPI codes (even both in the same process) directly, often with minimal performance loss. Tony Drummond from Lawrence Berkeley National Laboratory

---

[1]URL: https://cirl.berkeley.edu/fperez/cse09
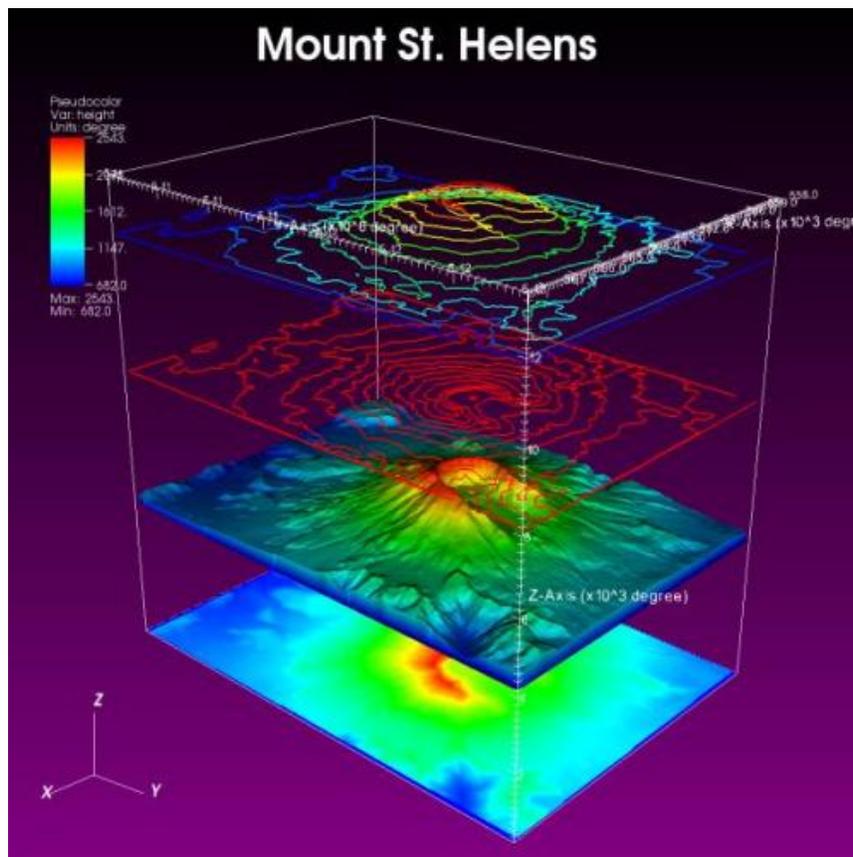
Figure 2: Topographic visualization of Mount St. Helens performed with Lawrence Livermore Lab's VisIt software.
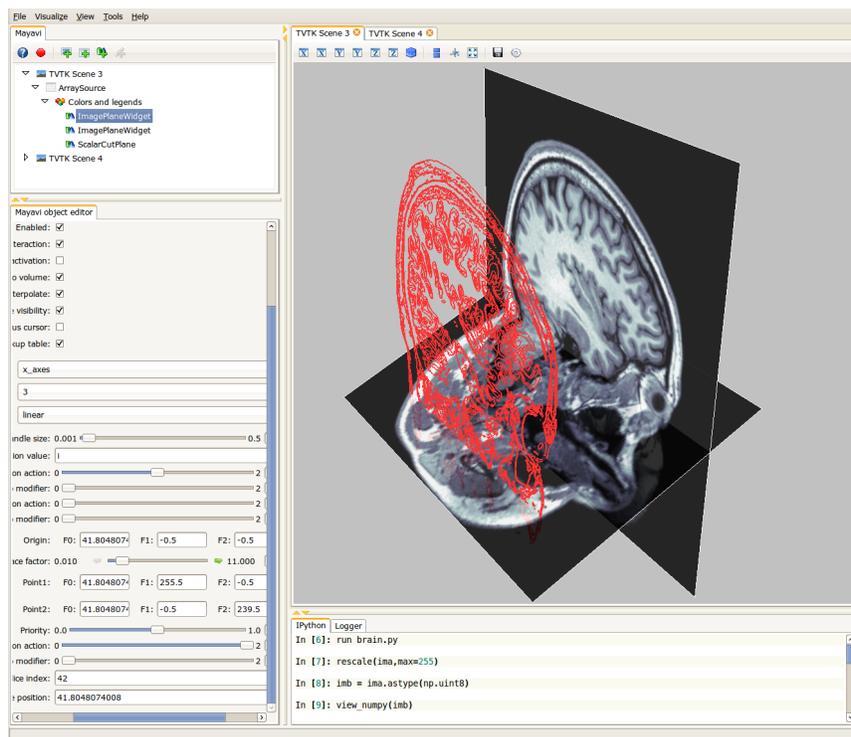
Figure 3: 3D interactive data visualization with Mayavi2 and IPython.

presented the *PyACTS* project that provides Python interfaces to the ACTS collection of high-performance codes (Aztec, Hypre, PETSc, SLEPc, ScaLAPACK, SUNDIALS, SuperLU, TAO and OPT++). Similar efforts of this type include *PyTrilinos* and *petsc4py* for using Trilinos and PETSc from Python. Jon Guyer from NIST showed the architecture of the *FiPy* project, a finite volume PDE solver developed by his group that exploits Python tools like NumPy, SciPy, matplotlib, and PyTrilinos to successfully tackle materials science problems – with an easy to use syntax for model description. Then, Aric Hagberg from Los Alamos National Laboratory presented the *NetworkX* project, a library of algorithms for studying complex networks that illustrates Python's strengths well: an algorithmic core with rich functionality coupled to multiple visualization libraries that provide alternative means of looking at networks. These visualization systems may be written in other languages, but via Python, they can all be used to render the results from NetworkX's with a unified interface.

But Python is not only a research tool: the day was bracketed by two presentations illustrating its many benefits for scientific computing education. Hans Petter Langtangen from Norway's Simula Research Laboratory described how the University of Oslo has successfully implemented a major reform of computational science teaching using Python as its foundation. Their students learn Python and numerical methods in the very first semester and apply these tools in a range of science courses across the University. Towards the end of the day, Joe Harrington from the University of Central Florida described an attempt in 2007 to replace IDL with Python in his course on Astronomical Data Analysis, with poor results because of documentation issues. He responded by organizing and funding the SciPy Documentation Project during the summer of 2008. Results were dramatic: students in his fall 2008 class learned more and spent less time than students in the IDL-based class.

Python's use in education is growing and its impact can be seen at both ends of the spectrum: while in the United States the NSF funds the SECANT project to develop a Python-based curriculum and workshops for interdisciplinary computational science education, the One Laptop Per Child project provides children in difficult conditions with laptops loaded with Python-based software they can inspect, learn from and modify.

Open tools and reproducible research

At the CSE conference SIAM emphasized the growing acknowledgment that computational research must be truly reproducible. All attendees to the conference received in their registration packet the January issue of *Computing in Science and Engineering*, a special issue focused on Reproducible Research. Python is an excellent platform on which to build a work flow where every step can be validated and reproduced by anyone: its open source nature and zero cost means that there are no barriers for anyone to use it, and all of its internals are open to inspection. Using Open Source tools to build our computing foundation facilitates this core principle of the scientific process in our discipline: every project discussed in this minisymposium is freely available for all to download, use, verify and improve upon. We hope the community of scientists building tools in this manner will continue to grow; the Python projects for scientific computing are rapidly maturing, becoming better integrated and documented, more powerful and easier to use. Join us!

Readers interested in learning more about these tools can visit the SciPy website, which hosts some of these tools and contains links and information to many more related projects.

We would like to thank the speakers of the minisymposium for their participation as well as for their careful reading and comments on the drafts of this article.