Python for Scientists and Engineers

During the past decade, Python (an interpreted, high-level programming language) has arguably become the de facto standard for exploratory, interactive, and computation-driven scientific research. This issue discusses Python's advantages for scientific research and presents several of the core Python libraries and tools used in this domain. Although the issue's articles are self-contained, they nicely complement those in *CiSE*'s May/June 2007 special issue, "Python: Batteries Included."[1]

In addition to the technical advantages described in this issue, one of Python's most compelling assets as a platform for scientific computing is the SciPy community. The SciPy community is a well-established and growing group of scientists, engineers, and researchers using, extending, and promoting Python's use for scientific research.

1521-9615/11/$26.00 © 2011 IEEE
Copublished by the IEEE CS and the AIP

K. Jarrod Millman
University of California, Berkeley
Michael Aivazis
California Institute of Technology

Scientific Computing for Python: A Short History

Although Python wasn't specifically designed to meet the computational needs of the scientific community, it quickly attracted the interest of scientists and engineers. Despite its expressive syntax and a rich collection of built-in data types (such as strings, lists, dictionaries), it became clear that, to provide the necessary framework for scientific computing, Python needed to provide an array type for numerical computing.

In 1995, the Python community formed the matrix-sig (http://mail.python.org/pipermail/matrix-sig), a special interest group focused on creating a new array data type. Jim Hugunin, then an MIT graduate student, developed a C-extension module called *Numeric*, based on Jim Fulton's matrix object released the year before and incorporating many ideas from the matrix-sig. In June 1997, Hugunin announced that he was leaving the project to focus on Jython, an implementation of Python using Java. After Hugunin left, Paul Dubois took over as the lead Numeric developer.

During these early years, there was considerable interaction between the standard and scientific Python communities. In fact, Guido van Rossum, Python's Benevolent Dictator For Life (BDFL), was an active member of the matrix-sig. This close interaction resulted in Python gaining new features and syntax specifically needed by the scientific Python community. While there were miscellaneous changes, such as the addition of complex numbers, many changes focused on providing a more succinct and easier to read syntax for array manipulation. For instance, the parenthesis around tuples were made optional so that array elements could be accessed through, for example, `a[0,1]` instead of `a[(0,1)]`. The slice syntax gained a step argument—`a[::2]` instead of just `a[:]`, for example—and an ellipsis operator, which is useful when dealing with multidimensional data structures.

Over the next five years, a relatively small but committed community of scientists and engineers using Python for its computing needs slowly formed around Numeric. This community continued to improve Numeric and began developing and sharing additional packages for scientific computing.

By 2000, there was a growing number of extension modules and increasing interest in creating a complete environment for scientific computing in Python. Over the next three years, several things happened that greatly increased Python's usefulness for scientific computing. Travis Oliphant, Eric Jones, and Pearu Peterson merged code they'd written and called the resulting package *SciPy*. The newly created package provided a standard collection of common numerical operations on top of the Numeric array data structure. Fernando Pérez released the first version of IPython, an enhanced interactive shell widely used in the scientific community. John Hunter released the first version of matplotlib, the standard 2D plotting library for scientific computing.

However, while Numeric had proven useful as a foundation for these new packages, its code base had become difficult to extend and development had slowed. To address this problem, Perry Greenfield, Todd Miller, and Rick White at the Space Telescope Science Institute in Baltimore, Maryland, developed a new array package for Python, called *numarray*, which pioneered many useful features. Unfortunately, the division between Numeric and numarray fractured the community for several years. This division was breached in 2006, when Travis Oliphant released NumPy, a significant rewrite of Numeric incorporating the most useful features of numarray. Since then, the SciPy community has rapidly grown and the basic stack of tools has steadily improved and expanded.

Python for Mathematicians

Although Python has been used for serious numerical computing since the mid '90s, Python has only in the last few years become popular for symbolic computing. To get a feeling for this important emerging direction, let's take a quick look at three popular projects for mathematical and symbolic computing: sympy, mpmath, and Sage.

SymPy is a computer algebra system written in pure Python. Figure 1 shows a simple SymPy session to give you an idea of what SymPy provides. After importing a few things from SymPy, we declare one symbol `x` using the `var` function. We can then use `x` symbolically by using either procedural (`integrate`) or object-oriented (`diff`) styles.

The mpmath library provides multiprecision floating-point arithmetic. Besides arbitrary-precision real and complex floating-point number types, mpmath has functions for infinite series and products, integrals, derivatives, limits, nonlinear equations, ordinary differential equations, special functions, function approximation, and linear algebra. As a simple demonstration, Figure 2 shows how we can evaluate π to 50 digits using the Gaussian integral:

$$\int_{-\infty}^{\infty} e^{-x^2} dx = \sqrt{\pi}\,.$$

Sage is an open source mathematical software system that bundles several open source packages and provides a uniform Python-based interface. It covers a range of mathematical domains including linear algebra, calculus, number theory, cryptography, commutative algebra, group theory, combinatorics, graph theory, and many more. While NumPy, SciPy, matplotlib, and several other libraries provide a numerical computing environment similar to Matlab, Sage is more similar to tools like Mathematica, Maple, or Magma.

Although the current issue doesn't provide an in-depth discussion of the growing importance of Python for mathematical and symbolic computing, the July/August 2012 issue of *CiSE* will focus on that topic.

In This Issue

We begin this issue with "Python: An Ecosystem for Scientific Computing," by Fernando Pérez, Brian E. Granger, and John D. Hunter. Today's scientific codes require not only raw numerical performance and ease of use, but often need to support network protocols, Web- and database-driven applications, and sophisticated graphical interfaces, among other things. This overview argues that Python augmented with a stack of tools developed specifically for scientific computing forms a highly productive environment for modern scientific computing.

The next two articles focus on improving the efficiency of Python code. NumPy and Cython provide complimentary approaches to balancing the needs of raw performance while retaining Python's ease of use. In their article, "The NumPy Array: A Structure for Efficient Numerical Computation," Stéfan van der Walt, S. Chris Colbert, and Gaël Varoquaux describe how NumPy provides a high-level multidimensional array structure, that also allows fine-grained control over performance and memory-management. In "Cython: The Best of Both Worlds," Stefan Behnel, Robert Bradshaw, Craig Citro, Lisandro Dalcin, Dag Sverre Seljebotn, and Kurt Smith discuss this popular tool for creating Python extension modules in C, C++, and Fortran.

```
>>> from sympy import var, sin, integrate, pi
>>> var('x')
x
>>> sin(x)
sin(x)
>>> sin(x).diff(x)
cos(x)
>>> integrate(sin(x), x)
-cos(x)
>>> integrate(sin(x), (x, 0, pi))
2
```

Figure 1. Demonstrating the symbolic computing capabilities of SymPy from the Python interpreter. In this example, we show how to create and use symbolic variables with SymPy functions to find the derivative and integral of `sin(x)`.

The final article, "Mayavi: 3D Visualization of Scientific Data," by Prabhu Ramachandran and Gaël Varoquaux introduces Mayavi, a 3D scientific visualization package for Python. Mayavi provides several interfaces that let scientists develop simple scripts to visualize their data; to load and explore their data with a full-blown interactive, graphical application; and to assemble their own custom applications from Mayavi widgets.

We hope you enjoy this special issue and try the tools presented. We also strongly recommend the 2007 issue's articles, which are still highly relevant for readers interested in learning more about Python's use in scientific computing. Finally, we encourage you to attend one of the annual SciPy conferences, which include tutorials and talks. The 10th

```
>>> from mpmath import mp, quad, inf, exp
>>> mp.dps = 50         # set precision
>>> mp.pretty = True    # nice output formatting
>>> quad(lambda x: exp(-x**2), [-inf,inf])**2
3.1415926535897932384626433832795028841971693993751
```

Figure 2. Using mpmath interactively from the Python interpreter to calculate the first 50 digits of π using the Gaussian integral.

US SciPy conference takes place this summer in Austin, Texas, from 11–16 July. In addition to the US conference, the 4th European SciPy conference will be held 25–28 August in Paris. Although the date and location hasn't been finalized, the 3rd SciPy India conference will take place in December. We're also planning the first SciPy conference in Japan this year. Please visit http://conference.scipy.org to register for conferences, view calls for papers, and find additional information.

Reference

1. Paul Dubois, "Python: Batteries Included," *Computing in Science and Eng.*, vol. 9, no. 3, 2007, pp. 7–9.

***K. Jarrod Millman** is a researcher at the University of California, Berkeley's Brain Imaging Center, where he helped found the Neuroimaging in Python (NIPY) project. He is on the SciPy steering committee and a contributor to both the NumPy and SciPy projects. His research interests include reproducible research, functional brain imaging, informatics, configuration management, and computer security. Millman has a BA in mathematics and computer science from Cornell University. Contact him at millman@berkeley.edu.*

***Michael Aivazis** is principal scientist at the Caltech Center for Advanced Computational Research, where his research focuses on the design and implementation of Pyre, a comprehensive, Python-based component framework for high-performance scientific computing. He is also a coprincipal investigator at the Caltech Predictive Science Academic Alliance Program Center, where he leads the effort to construct and integrate large-scale massively parallel multiphysics simulation codes in a large-scale, global optimization framework. In addition, he leads the effort to produce the next generation of solvers for the center, focusing on scalable parallel algorithms for meshing, contact, fracture, and fragmentation. His research interests include software engineering and techniques for object-oriented programming. Contact him at aivazis@cacr.caltech.edu.*