Python for Scientists and Engineers

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

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

In addition to the technical advantages described in this issue, one of Python's most compelling assets is the SciPy community. The SciPy community is a well-established and growing group of scientists, engineers, and researchers using, extending, and promoting its use for scientific research. Our community has formed around two main software packages, NumPy and SciPy. NumPy provides a multi-dimensional array object, a powerful data structure that has become the standard representation of numerical data in Python. SciPy provides additional functionality for NumPy arrays including toolboxes for IO, linear algebra, statistics, optimization, integration, interpolation, Fourier transforms, special functions, sparse matrices, image and signal processing, maximum entropy models, and clustering.

A short history of scientific computing for Python

While 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 (e.g., strings, lists, dictionaries), it became clear that in order to provide the necessary framework for scientific computing Python needed to provide an array type for numerical computing.

In order to address this need, the Python community formed a special interest

[1] Paul Dubois. Python: Batteries Included, Computing in Science and Engineering, vol. 9, no. 3, May/June 2007.

group (matrix-sig) [2] in 1995 focused on creating a new array data type. Jim Hugunin, a 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, Jim announced that he was leaving the project to focus on Jython, an implementation of Python using Java. After Jim left, Paul Dubois took over as the lead NumPy developer.

During these early years, there was a lot of 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 a number of miscellaneous changes such as the addition of complex numbers, many of these 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 like this *a[0,1]* instead of *a[(0,1)]*. The slice syntax gained a step argument (e.g., *a[::2]* instead of just *a[:]*) and an ellipsis operator, which is very useful when dealing with multi-dimensional data structures.

For the next five years, a relatively small, but committed community of scientists and engineers using Python for their computing needs slowly formed around *numeric*. This community continued to improve *numeric* and began developing additional packages (e.g., FFT, special functions, statistics, integration, optimization) 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 the usefulness of Python for scientific computing. Travis Oliphant, Eric Jones, and Pearu Peterson merged code they had written and called the resulting package SciPy. The newly created package provided a standard collection of common numerical operations (e.g., special functions, optimization, genetic algorithms) on top of the numeric array data structure. Fernando Perez 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. From these new packages, the SciPy community was born.

However, while *numeric* had proven useful as a foundation for these new packages its codebase 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 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 a number of years. This division was breached in 2006, when Travis Oliphant released *NumPy*, which is a significant rewrite of *numeric* incorporating the useful features pioneered by

[2] http://mail.python.org/pipermail/matrix-sig/

numarray. Since then, the SciPy community has rapidly grown and the basic stack of tools has steadily improved and expanded.

Python for mathematicians

While Python has been used for serious numerical computing since the mid-90s, Python has only in last few years become popular for symbolic computing. Let's take a quick look at three of the most popular projects for mathematical and symbolic computing: sympy, mpmath, and Sage.

SymPy is a computer algebra system written in pure Python. To give you an idea of what SymPy provides, let's consider a simple SymPy session:

```
>>> 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
```

After importing a few things from SymPy, we declare one symbol `x`, using the `var` function. Now we can use `x` symbolically by using either procedural (`integrate`) or object oriented (`diff`) styles.

The `mpmath` library provides multi-precision 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. Since Python's built-in integers are inefficient at high precision, `mpmath` uses GMP/MPIR integers when available and uses Cython (described in this issue) to speed up computational primitives. As a simple demonstration, $\left(\int_{-\infty}^{\infty} e^{-x^2} dx\right)^2 = \pi$ may be evaluated accurately to 50 digits as follows:

```
>>> 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
```

Sage is an open source mathematical software system, which bundles several

open source packages and provides a uniform Python-based interface. It covers a huge 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.

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

In this issue

We begin this issue with an overview of the Python ecosystem for scientific computing. 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, sophisticated graphical interfaces, among other things. The 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. NumPy provides a multi-dimensional array structure as well as several operations on arrays. Cython is a popular tool for creating Python extension modules in C, C++, and Fortran.

The final article introduces a 3D scientific visualization package for Python called Mayavi. Mayavi provides several interfaces allowing scientists to develop simple scripts to visualize their data, to load and explore their data with a full-blown interactive, graphical application, as well as assembling their own custom applications from Mayavi widgets.

Next steps

We hope you enjoy this special issue and try the tools presented. For more information, the articles in the 2007 issue are still highly relevant and strongly recommended for readers interested in learning more about the use of Python in scientific computing. We also encourage you to attend one of the annual SciPy conferences, which included tutorials and talks. The tenth US SciPy conference takes place this summer in Austin, TX from July 11 to July 16, 2011. In addition to the US conference, the 4th European SciPy conference will be held from August 25th to the 28th in Paris, France. While the date and location

hasn't been finalized, the third SciPy India conference will be held sometime in December 2011. We are also planning the first SciPy conference in Japan this year. Please visit http://conference.scipy.org to find registration, call for papers, as well as additional information.

K. Jarrod Millman is on the SciPy steering committee and a contributor to both the NumPy and SciPy projects. He is a researcher at UC Berkeley's Brain Imaging Center, where he helped found the Neuroimaging in Python (NIPY) project. 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 Brain Imaging Center, 10 Giannini Hall, Berkeley, CA 94720; millman@berkeley.edu.

Michael Aivazis is a Co-PI at the Caltech Center for the Dynamic Response of Materials, where he is leading the effort to construct and integrate large scale massively parallel multi-physics simulation codes. His primary responsibility is the design and implementation of pyre, the Center's integration platform and problem solving environment. He is also leading the effort to produce the next generation of solvers to be used by the center, with focus currently 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 CACR 158-79, Caltech, Pasadena, CA 91125; aivazis@cacr.caltech.edu; www.cacr.caltech.edu.