The State of SciPy

Jarrod Millman (millman@berkeley.edu) – *University of California Berkeley, Berkeley, CA* USA
Travis Vaught (travis@enthought.com) – *Enthought, Austin, TX* USA

The annual SciPy conference provides a unique opportunity to reflect on the state of scientific programming in Python. In this paper, we will look back on where we have been, discuss where we are, and ask where we are going as a community.

Given the numerous people, projects, packages, and mailing lists that make up the growing SciPy community, it can be difficult to keep track of all the disparate developments. In fact, the annual SciPy conference is one of the few events that brings together the community in a concerted manner. So it is, perhaps, appropriate that we begin the conference proceedings with a paper titled "The State of SciPy". Our hope is we can help provide the context for the many interesting and more specific papers to follow. We also aim to promote a much more detailed discussion of the state of the project, community, and software stack, which will continue throughout the year.

The last year has seen a large number of exciting developments in our community. We have had numerous software releases, increased integration between projects, increased test coverage, and improved documentation. There has also been increased focus on improving release management and code review. While many of the papers in the proceedings describe the content of these developments, this paper attempts to focus on the view from 10,000 feet.

This paper is organized in three sections. First, we present a brief and selective historical overview. Second, we highlight some of the important developments from the last year. In particular, we cover the status of both NumPy and SciPy, community building events, and the larger ecosystem for scientific computing in Python. Finally, we raise the question of where the community is heading and what we should focus on during the coming year. The major goal of the last section is to provide some thoughts for a roadmap forward—to improve how the various projects fit together to create a more unified user environment.

In addition to this being the first year that we have published conference proceedings, it is also the first time we have had a formal presentation on the state of SciPy. It is our hope that these will both continue in future conferences.

Past: Where we have been

Before highlighting some of the communities' accomplishments this year, we briefly present a history of scientific computing in Python. Since almost the first release of Python, there has been interest in the scientific community for using Python. Python is an ideal choice for scientific programming; it is a mature, robust, widely-used, and open source language that is freely distributable for both academic and commercial use. It has a simple, expressive, and accessible syntax. It does not impose a single programming paradigm on scientists but allows one to code at many levels of sophistication, including Matlab style procedural programming familiar to many scientists. Python is available in an easily installable form for almost every software platform, including Windows, Macintosh, Linux, Solaris, FreeBSD and many others. It is therefore well suited to a heterogeneous computing environment. Python is also powerful enough to manage the complexity of large applications, supporting functional programming, object-oriented programming, generic programming, and metaprogramming. In contrast, commercial languages like Matlab and IDL, which also support a simple syntax, do not scale well to many complex programming tasks. Lastly, Python offers strong support for parallel computing. Because it is freely available, and installed by default on most Unix machines, Python is an excellent parallel computing client.

Using Python allows us to build on scientific programming technologies that have been under active development and use for over 10 years; while, at the same time, it allows us to use mixed language programming (primarily C, C++, FORTRAN, and Matlab) integrated under a unified Python interface. IPython (ipython.scipy.org) is the de facto standard interactive shell in the scientic computing community. It has many features for object introspection, system shell access, and its own special command system for adding functionality when working interactively. It is a very efficient environment both for Python code development and for exploration of problems using Python objects (in situations like data analysis). Furthermore, the IPython has support for interactive parallel computing. Matplotlib (matplotlib.sourceforge.net) is a tool for 2D plots and graphs, which has become the standard tool for scientific visualization in Python. NumPy (numpy.scipy.org) is a high-quality, fast, stable package for N-dimensional array mathematics. SciPy (scipy.org) is a collection of Python tools providing an assortment of basic scientific programming algorithms (e.g., statistics, optimization, signal processing, etc.). The combination of IPython, matplotlib, NumPy, and SciPy forms the basis of a Matlab-like environment that provides many of the strengths of Matlab (platform independence, simple syntax, high level algorithms, and visualization routines) without its limitations (proprietary, closed source with a weak object model and limited networking capabilities).

Here is a selective timeline:

- 1994 — Python Matrix object (Jim Fulton)

- 1995 — Numeric born (Jim Hugunin, Konrad Hinsen, Paul Dubois, David Ascher, Jim Fulton)
- 2000 — Numeric moves to sourceforge (Project registered as numpy)
- 2001 — SciPy born (Pearu Peterson, Travis Oliphant, Eric Jones)
- 2001 — IPython born (Fernando Perez)
- 2002 — SciPy '02 - Python for Scientific Computing Workshop
- 2003 — matplotlib born (John Hunter)
- 2003 — Numarray (Perry Greenfield, J. Todd Miller, Rick White, Paul Barrett)
- 2006 — NumPy 1.0 Released

Present: Where we are

With the release of NumPy 1.0 in 2006, the community had a new foundation layer for scientific computing built on the mature, stable Numeric codebase with all the advanced functionality and features developed in Numarray. Of course, the existing scientific software had to be ported to NumPy. While Travis Oliphant spent a considerable effort to ensure that this would be as simple as possible, it did take some time for all the various projects to convert.
This is the second conference for the community since the release of NumPy. At this point, most projects have adopted NumPy as their underlying numeric library.

NumPy and SciPy packages

For several months leading up to last year's conference, we were in the unfortunate position that the current releases of NumPy and SciPy were incompatible. At the conference we decided to resolve this by releasing NumPy 1.0.3.1 and SciPy 0.5.2.1. These releases included a few other minor fixes, but didn't include the bulk of the changes from the trunk. Since then we have had three releases of NumPy and one release of SciPy:

- SciPy 0.6.0 (September 2007)
- NumPy 1.0.4 (November 2007)
- NumPy 1.1.0 (May 2008)
- NumPy 1.1.1 (August 2008)

These releases featured a large number of features, speed-ups, bug-fixes, tests, and improved documentation.

- New masked arrays — *MaskedArray* now subclasses *ndarray*. The behavior of the new *MaskedArray* class reproduces the old one.
- Overhaul of IO Code — The NumPy/SciPy IO code is undergoing a major reworking. NumPy will provide basic IO code for handling NumPy arrays, while SciPy will house file readers and writers for third-party data formats (data, audio, video, image, etc.). NumPy also supports a new standard binary file format (.npy/.npz) for arrays/groups_of_arrays. This is the new default method of storing arrays; pickling arrays is discouraged.
- Better packaging — The win32 installer now solves the previously recurring problem of non-working atlas on different sets of CPU. The new installer simply checks which CPU it is on, and installs the appropriate NumPy accordingly (without atlas if the CPU is not supported). We also now provide an official Universal Mac binary.
- Improved test coverage — This year has seen a concerted focus on better test coverage as well as a push for test-driven development. An increasing number of developers are requesting patches or commits to include unit tests.
- Adopted Python Style Guide [PEP8] — For years the official naming convention for classes in NumPy and SciPy was lower_underscore_separated. Since the official Python convention used CapWords for classes as well as several SciPy-related projects (e.g., ETS, matplotlib), it was confusing and led to both standards being used in our codebase. Going forward, newly created classes should adhere to the Python naming convention. Obviously, we will have to keep some of the old class names around so that we don't needlessly break backward compatibility.

Version numbering. During the lead up to the 1.1.0 release, it became apparent that we needed to become more disciplined in our use of release version numbering. Our current release versioning uses three numbers, separated by periods **<major.minor.bugfix>**.[1] Development code for a new release appends an alphanumeric string to the upcoming release numbers, which designate that status (e.g., alpha, beta) of the development code. For example, here is a key to the current minor 1.2.x release series:

- 1.2.0dev5627 — development version 5627
- 1.2.0a1 - first alpha release
- 1.2.0b2 — second beta release
- 1.2.0rc1 — first release candidate
- 1.2.0 — first stable release
- 1.2.1 — first bug-fix release

According to this numbering scheme, the NumPy 2.0.0 release will be the first in this series to allow us to make more significant changes, which would require large-scale API breaks. The idea being that a major release might require people *rewriting* code where a minor release would require a no more than a small amount of *refactoring*. Bug-fix releases will not require any changes to code depending on our software.

Buildbot. Albert Strasheim and Stéfan van der Walt set up a buildbot for numpy shortly before last year's SciPy conference. The buildbot is an automated system for building and testing. This allows the developers and release manager to better track the ongoing evolution of the software.

Community Involvement

The level of community involvement in the project has seen solid growth over the last year. There were numerous coding sprints, training sessions, and conference events. We also held a number of documentation and bug-fix days. And Gaël Varoquaux set up a SciPy blog aggregator early in 2008, which currently has almost fifteen subscribers:

`http://planet.scipy.org/`

Sprints. Sprints have become popular coding activities among many open-source projects. As the name implies, sprints are essentially short, focused coding periods where project members work together in the same physical location. By bringing developers in same location for short-periods of time allows them to socialize, collaborate, and communicate more effectively than working together remotely. While the SciPy community has had sprints for a number of years, this year saw a marked increase. Here is a list of a few of them:

- August 2007 — SciPy 2007 post-conference sprint
- December 2007 — SciPy sprint at UC Berkeley
- February 2008 — SciPy/SAGE sprint at Enthought
- March 2008 — NIPY/IPython sprint in Paris
- April 2008 — SciPy sprint at UC Berkeley
- July 2008 — Mayavi sprint at Enthought

Conferences. In addition to the SciPy 2008 conference, SciPy has had a major presence in several other conferences this year:

- PyCon 2008 — Travis Oliphant and Eric Jones taught a tutorial session titled "Introduction to NumPy" and another titled "Tools for Scientific Computing in Python". While, John Hunter taught a session titled "Python plotting with matplotlib and pylab".
- 2008 SIAM annual meeting — Fernando Perez and Randy LeVeque organized a 3-part minisymposium entitled "Python and Sage: Open Source Scientific Computing", which were extremely well received and chosen for the conference highlights page.
- EuroSciPy 2008 — The first ever EuroSciPy conference was held in Leipzig, Germany on Saturday and Sunday July 26-27, 2008. Travis Oliphant delivered the keynote talk on the history of NumPy/SciPy. There were about 45 attendees.

The Larger Ecosystem

The NumPy and SciPy packages form the basis for a much larger collection of projects and tools for scientific computing in Python. While many of the core developers from this larger ecosystem will be discussing recent project developments during the conference, we want to selectively highlight some of the exciting developments in the larger community.

Major Releases. In addition to releases of NumPy and SciPy, there have been a number of important releases.

- matplotlib 0.98 — Matplotlib is a core component of the stack, and the 0.98 release contains a rewrite of the transforms code. It also features mathtext without the need of LaTeX installed, and a 500-pages-long user-guide.
- ETS 3 — The Enthought Tool Suite (ETS) is a collection of components developed by Enthought and their partners. The cornerstone on which these tools rest is the Traits package, which provides explicit type declarations in Python; its features include initialization, validation, delegation, notification, and visualization of typed attributes.
- Mayavi 2 — This Mayavi release is a very important one, as Mayavi 2 now implements all the features of the original Mayavi 1 application. In addition, it is a reusable library, useful as a 3D visualization component in the SciPy ecosystem.
- IPython 0.9 — This release of IPython marks the integration of the parallel computing code with the core IPython interactive shell.

Distribution. While the quantity and quality of the many scientific Python package has been one the strengths of our community, a mechanism to easily and simply install all these packages has been a weakness. While package distribution is an area that still needs improvement, the situation has greatly improved this year. For years the major Linux distributions have provided official packages of the core scientific Python projects including NumPy, SciPy, matplotlib, and IPython. Also starting with version 10.5

[1] The NumPy 1.0.3.1 and SciPy 0.5.2.1 releases being an aberration from this numbering scheme. In retrospect, those releases should have been numbered 1.0.4 and 0.5.3 respectively.

"Leopard", Mac OS X ships with NumPy 1.0.1 pre-installed. Also, as mentioned above, the NumPy and SciPy projects have worked to provide better binary installers for Windows and Mac. This year has also seen a number of exciting efforts to provide a one stop answer to scientific Python software distribution:

- Python Package Index — While there are still many issues involving the wide-spread adoption of setuptools, an increasing number of projects are providing binary eggs on the Python Packaging Index. This means an increasing number of scientists and engineers can easily install scientific Python packages using *easy_install*.
- Python(x,y) — www.pythonxy.com
- EPD — Enthought Python Distribution, www.enthought.com/epd
- Sage — Sage is a Python-based system which aims at providing an open source, free alternative to existing proprietary mathematical software and does so by integrating multiple open source projects, as well as providing its own native functionality in many areas. It includes by default many scientific packages including NumPy, SciPy, matplotlib, and IPython.

Cool New Tools. There were also several useful tools:

- Sphinx — Documentation-generation tool.
- Cython — New Pyrex: mixing statically-typed, compiled code with dynamically-typed code, with a Python-like syntax.

Future: Where are we going?

What will the future hold? - Improved release management. - More regular releases. - Clear policy on API and ABI changes. - Better unification of the existing projects.
In the broader view, our ecosystem would benefit from more project cohesion and common branding, as well as an IDE (integrated development environment), an end-user application that would serve as an entry point to the different technologies.

NumPy and SciPy packages

NumPy 1.2 and SciPy 0.7 are on track to be released by the end of this month. This is the first synchronous release since NumPy last August. SciPy 0.7 will require NumPy 1.2 and both releases require Python 2.4 or greater and feature:

- Sphinx-based documentation — This summer saw the first NumPy Documentation Marathon, during which many thousands of lines of documentation were written. In addition, a web framework was developed which allows the community to contribute docstrings in a wiki-like fashion, without needing access to the source repository. The new reference guide, which is based on these contributions, was built using the popular Sphinx tool. While the documentation coverage is now better than ever, there is still a lot of work to be done, and we encourage interested parties to register and contribute further.
- Guide to NumPy — Travis Oliphant released his "Guide to NumPy" for free and checked it into the trunk. Work has already begun to convert it to the ReST format used by Sphinx.
- Nose-based testing framework — The NumPy test framework now relies on the nose testing framework version 0.10 or later.

In addition, SciPy 0.7 includes a whole host of new features and improvements including:

- Major sparse matrices improvements —
- Sandbox removed — The sandbox was originally intended to be a staging ground for packages that were undergoing rapid development during the port of SciPy to NumPy. It was also a place where broken code could live. It was never intended to stay in the trunk this long and was finally removed.
- New packages and modules — Several new packages and modules have been added including constants, radial basis functions, hierarchical clustering, and distance measures.

Python 3.0. Python 2.6 and 3.0 should be released before the end of the year. Python 3.0 is a new major release that breaks backward compatibility with 2.x. The 2.6 release is provided to ease forward compatibility. We will need to keep supporting Python 2.x for the at least the near future. If needed, once released we will provide bug-fix releases to the current releases of NumPy and SciPy to ensure that they run on Python 2.6. We don't currently have a time-frame for Python 3.0 support, but we would like to have a Python 3.0 compatible releases before next year's SciPy conference. The [PEP3000] recommends:

1. You should have excellent unit tests with close to full coverage.
2. Port your project to Python 2.6.
3. Turn on the Py3k warnings mode.
4. Test and edit until no warnings remain.
5. Use the 2to3 tool to convert this source code to 3.0 syntax. **Do not manually edit the output!**
6. Test the converted source code under 3.0.
7. If problems are found, make corrections to the **2.6** version of the source code and go back to step 3.
8. When it's time to release, release separate 2.6 and 3.0 tarballs (or whatever archive form you use for releases).

Release Management

While we were able to greatly improve the quality of our releases this year, the release process was a much less than ideal. Features and API-changes continually creep in immediately before releases. Release schedules repeatedly slipped. And the last SciPy release is out of date compared to the trunk. Obviously these problems are not unique to our project. Software development is tricky and requires balancing many different factors and a large distributed project like ours has the added complexity of coordinating the activity of many different people. During the last year, we had several thoughtful conversations about how to improve the situation. A major opportunity for our project this year will be trying to improve the quality of our release management. In particular, we will need to revisit issues involving release management, version control, and code review.

Time Based Releases. Determining when and how to make a new release is a difficult problem for software development projects. While there are many ways to decide when to release code, it is common to think in terms of feature- and time-based releases:

[Feature-based]

A release cycle under this model is driven by deciding what features will be in the next release. Once all the features are complete, the code is stabilized and finally a release is made. Obviously this makes it relatively easy to predict what features will be in the next release, but extremely difficult to determine when the release will occur.

[Time-based]

A release cycle under this model is driven by deciding when the next release will be. This, of course, makes predicting when the release will be out extremely easy, but makes it difficult to know exactly what features will be included in the release.

Over the last several years, many large, distributed, open-source projects have moved to time-based release management. There has been a fair amount of interest among the SciPy development community to move in this direction as well. Time-based releases are increasingly seen as an antidote to the issues associated with more feature driven development in distributed, volunteer development projects (e.g., lack of planning, continual release delays, out of date software, bug reports against old code, frustration among developers and users). Time-based releases also allows "a more controlled development and release process in projects which have little control of their contributors and therefore contributes to the quality of the output" [Mic07]. It also moves the 'release when it's ready' policy down to the level of specific features rather than holding the entire code base hostage.

A major objective of time-based releases is regular releases with less time between releases. This rapid pace of regular releases, in turn, enables more efficient developer coordination, better short and long term planning, and more timely user feedback, which is more easily incorporated into the development process. Time-based releases promote more incremental development, while they discourage large-scale modifications that exceed the time constraints of the release cycle.

An essential feature of moving to time-based releases is determining the length of the release cycle. With a few notable exceptions (e.g., Linux kernel), most projects have followed the GNOME 6-month release cycle, originally proposed by Havoc Pennington [Pen02]. In order to ensure that the project will succeed at meeting a 6-month time frame requires introducing new policies and infrastructure to support the new release strategy. And the control mechanisms established by those policies and infrastructure have to be enforced.

In brief, here is a partial list of issues we will need to address in order to successfully move to time-based release schedule:

- Branching — In order to be able to release on schedule requires that the mainline of development (the trunk) is extremely stable. This requires that a significant amount of work being conducted on branches.
- Reviewing — Another important way to improve the quality of project and keep the trunk in shape is to require peer code review and consensus among the core developers on which branches are ready to be merged.
- Testing — A full test suite is also essential for being able to regularly release code.
- Reverting — Sticking to release schedule requires occasionally reverting commits.
- Postponing — It also requires postponing branch merges until the branch is ready for release.
- Releasing — Since time-based release management relies on a regular releases, the cost of making a release needs to be minimized. In particular, we need to make it much, much easier to create the packages, post the binaries, create the release notes, and send out the announcements.

An important concern when using time-based releases in open-source projects is this very ability to work relatively "privately" on code and then easily contribute it back to the trunk when it is finally ready.

Given how easy it is for developers to create their own private branches and independently work on them using revision control for as long as they wish, it is important to provide social incentives to encourage regular collaboration and interaction with the other members of the project. Working in the open is a core value of the open-source development model. Next we will look at two mechanisms for improving developer

Proposals. Currently there is no obvious mechanism for getting new features accepted into NumPy or SciPy. Anyone with commit access to the trunk may simply start adding new features. Often, the person developing the new feature, will run the feature by the list. While this has served us to this point, the lack of a formal mechanism for feature proposals is less than ideal.

Python has addressed this general issue by requiring proposals for new features conform to a standard design document called a Python Enhancement Proposal (or "PEP"). During the last year, several feature proposals were written following this model:

- A Simple File Format for NumPy Arrays [NEP1]
- Implementing date/time types in NumPy
- Matrix Indexing
- Runtime Optimization
- Solvers Proposal

Code Review. We also lack a formal mechanism for code review. Developers simply commit code directly to the trunk. Recently there has been some interest in leveraging the Rietveld Code Review Tool developed by Guido van Rossum [Ros08]. Rietveld provides web-based code review for subversion projects. Another option would be to use bzr and the code review functionality integrated with Launchpad.

Code review provides a mechanism for validating design and implementation of patches and/or branches. It also increases consistency in design and coding style.

Distributed Version Control. While subversion provides support for branching and merging, it is not its best feature. The difficult of branch tracking and merging under subversion is pronounced enough that most subversion users shy away from it. Since there is a clear advantage to leverage branch development with time-based releases, we will want to consider using version control mechanisms that provide better branching support.

Distributed Version Control Systems (DVCS), unlike centralized systems such as subversion, have no technical concept of a central repository where everyone in the project pulls and pushes changes. Under DVCS every developer typically works on his own local repository or branch. Since everyone has their own branch, the mechanism of code sharing is merging. Given that with DVCS, branching and merging are essential activities, they are extremely well-supported. This makes working on branches and then merging the code in the trunk only once it is ready extremely simple and easy. DVCS also have the advantage that they can be used off-line. Since anyone can create their own branch from the project trunk, it potentially lowers the barrier to project participation. This has the potential to create a greater culture of meritocracy than traditional central version control systems, which require potential project contributors to acquire "committer" status before gaining the ability to commit code changes to the repository. Finally, DVCS makes it much easier to do private work—allowing you to use revision control for preliminary work that you may not want to publish.

Proposed Release Schedule. Assuming we (1) agree to move to a time-based release, (2) figure out how to continuously keep the trunk in releasable condition, and (3) reduce the manual effort required to make both stable and development releases, we should be able to increase the frequencies of our releases considerably.

	Devel	Stable
Oct	1.3.0b1	1.2.1
Nov	1.3.0rc1	
Nov		1.3.0
Jan	1.4.0a1	1.3.1
Mar	1.4.0b1	1.3.2
Apr	1.4.0rc1	
May	1.4.0rc1	1.3.3

subsystem maintainers, release conference calls, irc meetings, synchronized releases of NumPy and SciPy.

Getting involved

- Documentation
- Bug-fixes
- Testing
- Code contributions
- Active Mailing list participation
- Start a local SciPy group
- Code sprints
- Documentation/Bug-fix Days
- Web design

Acknowledgements

- For reviewing... Stéfan van der Walt
- The whole community...

References

[PEP8] http://www.python.org/dev/peps/pep-0008/

[Mic07] Michlmayr, M. (2007). Quality Improvement in Volunteer Free and Open Source Software Projects: Exploring the Impact of Release Management. PhD dissertation, University of Cambridge.

[Pen02] http://mail.gnome.org/archives/gnome-hackers/2002-June/msg00041.html

[NEP1] http://svn.scipy.org/svn/numpy/trunk/numpy/doc/npy-format.txt

[Ros08] http://code.google.com/appengine/articles/rietveld.html

[PEP3000] http://www.python.org/dev/peps/pep-3000/