Matplotlib Solves the Riddle of the Sphinx

Michael Droettboom (`mdboom@gmail.com`) – *Space Telescope Science Institute,* USA

This paper shares our experience converting matplotlib's documentation to use Sphinx and will hopefully encourage other projects to do so. Matplotlib's documentation serves as a good test case, because it includes both narrative text and API docstrings, and makes use of automatically plotted figures and mathematical expressions.

Introduction

Sphinx [Bra08] is the official documentation tool for future versions of Python and uses reStructuredText [Goo06] as its markup language. A number of projects in the scientific Python community (including IPython and NumPy) have also converged on Sphinx as a documentation tool. This standardization, along with the ease-of-use of reStructuredText, should encourage more people to contribute to documentation efforts.

History

Before moving to Sphinx, matplotlib's [Hun08] documentation toolchain was a homegrown system consisting of:

- HTML pages written with the YAPTU templating utility [Mar01], and a large set of custom functions for automatically generating lists of methods, FAQ entries, generating screenshots etc.

- Various documents written directly in LaTeX, for which only PDF was generated.

- pydoc [Yee01] API documentation, only in HTML.

- A set of scripts to build everything.

Moving all of these separate formats and silos of information into a single Sphinx-based build provides a number of advantages over the old approach:

- We can generate printable (PDF) and on-line (HTML) documentation from the same source.

- All documentation is in a single format, reStructuredText, and in plain-text files or docstrings. Therefore, there is less need to copy-paste-and-reformat information in multiple places and risk diverging.

- There are no errors related to manually editing HTML or LaTeX syntax, and therefore the barrier to new contributers is lower.

- The output is more attractive, since the Sphinx developers have HTML/CSS skills that we lack. Also, the docstrings now contain rich formatting, which improves readability over pydoc's raw monospaced text. (See Figures at the end of this paper).

- The resulting content is searchable, indexed and cross-referenced.

Perhaps most importantly, by moving to a standard toolchain, we are able to share our improvements and experiences, and benefit from the contributions of others.

Built-in features

Search, index and cross-referencing

Sphinx includes a search engine that runs completely on the client-side. It does not require any features of a web server beyond serving static web pages. This also means that the search engine works with a locally-installed documentation tree.

Sphinx also generates an index page. While docstrings are automatically added to the index, manually indexing important keywords is inherently labor-intensive so matplotlib hasn't made use of it yet. However, this is a problem we'd like to solve in the long term, since many of the questions on the mailing list arise from not being able to find information that is already documented.

autodoc

Unlike tools like pydoc and epydoc [Lop08], Sphinx isn't primarily a tool for fully-automatic API and code documentation. Instead, its focus is on narrative documentation, meant to be read in a particular order. This difference in bias is not accidental. Georg Brandl, the author of Sphinx, wrote[1]:

> One of Sphinx' goals is to coax people into writing good docs, and that unfortunately involves writing in many instances :) This is not to say that API docs don't have their value; but when I look at a new library's documentation and only see autogenerated API docs, I'm not feeling encouraged.

However, Sphinx does provide special directives to extract and insert docstrings into documentation, collectively called the `autodoc` extension. For example, one can do the following:

```
.. automodule:: matplotlib.pyplot
   :members:
   :show-inheritance:
```

This creates an entry for each class, function, etc. in the `matplotlib.pyplot` module.

There are a number of useful features in epydoc that aren't currently supported by Sphinx including:

---

[1]In a message on the `sphinx-dev` mailing list on August 4, 2008: http://groups.google.com/group/sphinx-dev/msg/9d173107f7050e63

- Linking directly to the source code.

- Hierarchical tables of modules, classes, methods etc. (Though documented objects are inserted into an alphabetized master index.) This shortcoming is partially addressed by the inheritance diagram extension.

- A summary table with only the first line of each docstring, that links to the complete versions.

In the matplotlib documentation, this last shortcoming is painfully felt by the `pyplot` module, where over one hundred methods are documented at length. There is currently no way to easily browse what methods are available.

Note that Sphinx development progresses rather quickly, and some or all of these shortcomings may be resolved very soon.

Extended features

As Sphinx is written in Python, it is quite easy to write extensions. Extensions can:

- add new builders that, for example, support new output formats or perform actions on the parsed document trees.

- add code triggered by certain events during the build process.

- add new reStructuredText roles and directives, extending the markup. (This is primarily a feature of docutils, but Sphinx makes it easy to include these extensions in your configuration).

Most of the extensions built for matplotlib are of this latter type.

The matplotlib developers have created a number of Sphinx extensions that may be generally useful to the Scientific Python community. Where applicable, these features have been submitted upstream for inclusion in future versions of Sphinx.

Automatically generated plots

Any matplotlib plot can be automatically rendered and included in the documentation. The HTML version of the documentation includes a PNG bitmap and links to a number of other formats, including the source code of the plot. The PDF version of the documentation includes a fully-scalable version of the plot that prints in high quality.

This functionality is very useful for the matplotlib docs, as we can now easily include figures that demonstrate various methods. For example, the following reStructuredText directive inserts a plot generated from an external Python script directly into the document:

```
.. plot:: ../mpl_examples/xcorr_demo.py
```

See Figures for a screenshot of the result.

Inheritance diagrams

Given a list of classes or modules, inheritance diagrams can be drawn using the graph layout tool graphviz [Gan06]. The nodes in the graph are hyperlinked to the rest of the documentation, so clicking on a class name brings the user to the documentation for that class.

The reStructuredText directive to produce an inheritance diagram looks like:

```
.. inheritance-diagram:: matplotlib.patches
                         matplotlib.lines
                         matplotlib.text
   :parts: 2
```

which produces:
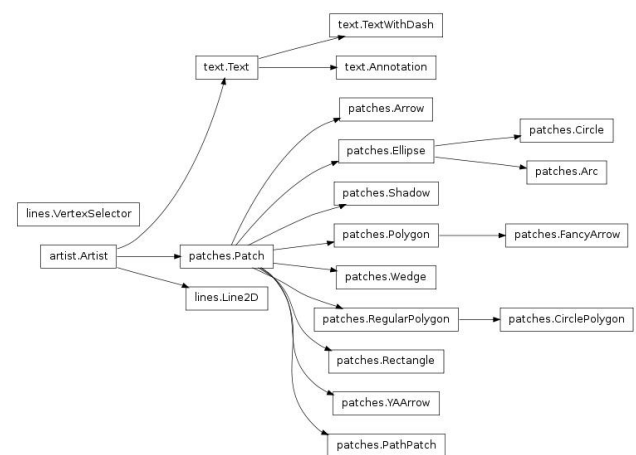
Mathematical expressions

Matplotlib has built-in rendering for mathematical expressions that does not rely on external tools such as LaTeX, and this feature is used to embed math directly in the Sphinx HTML output.

This rendering engine was recently rewritten by porting a large subset of the TeXmath layout algorithm [Knu86] to Python[2]. As a result, it supports a number of new features:

- radicals, eg., $\sqrt[3]{x}$

- nested expressions, eg., $\sqrt{\frac{x+\frac{1}{3}}{x+1}}$

- wide accents, eg., $\widehat{xyz}$

- large delimiters, eg., $\left(\frac{\delta x}{\delta y}\right)\lfloor z \rfloor$

- support for the STIX math fonts [STI08], giving access to many more symbols than even TeX itself, and a more modern-looking sans-serif math mode.

The following figure shows a complex fictional mathematical expression rendered using the three supported font sets, Computer Modern, STIX and STIX sans serif.

$$W^{3\beta}_{\delta_1\rho_1\sigma_2} = U^{3\beta}_{\delta_1\rho_1} + \sqrt{\frac{1}{8\pi 2}} \int^{\alpha_2}_{\alpha_2} d\alpha_2' \left[\frac{U^{2\beta}_{\delta_1\rho_1} - \alpha_2' U^{1\beta}_{\rho_1\sigma_2}}{\mathbb{A}^{0\beta}_{\rho_1\sigma_2}} \right]$$

$$W^{3\beta}_{\delta_1\rho_1\sigma_2} = U^{3\beta}_{\delta_1\rho_1} + \sqrt{\frac{1}{8\pi 2}} \int^{\alpha_2}_{\alpha_2} d\alpha_2' \left[\frac{U^{2\beta}_{\delta_1\rho_1} - \alpha_2' U^{1\beta}_{\rho_1\sigma_2}}{\mathbb{A}^{0\beta}_{\rho_1\sigma_2}} \right]$$

$$W^{3\beta}_{\delta_1\rho_1\sigma_2} = U^{3\beta}_{\delta_1\rho_1} + \sqrt{\frac{1}{8\pi 2}} \int^{\alpha_2}_{\alpha_2} d\alpha_2' \left[\frac{U^{2\beta}_{\delta_1\rho_1} - \alpha_2' U^{1\beta}_{\rho_1\sigma_2}}{\mathbb{A}^{0\beta}_{\rho_1\sigma_2}} \right]$$

The use of this extension in the matplotlib documentation is primarily a way to test for regressions in our own math rendering engine. However, it is also useful for generating math expressions on platforms that lack a LaTeX installation, particularly on Microsoft Windows and Apple OS-X machines, where LaTeX is harder to install and configure.

There are also other options for rendering math expressions in Sphinx, such as mathpng.py[3], which uses LaTeX to perform the rendering. There are plans to add two new math extensions to Sphinx itself in a future version: one will use jsmath [Cer07] to render math using JavaScript in the browser, and the other will use LaTeX and dvipng for rendering.

Syntax-highlighting of IPython sessions

Sphinx on its own only knows how to syntax-highlight the output of the standard `python` console. For matplotlib's documentation, we created a custom docutils formatting directive and pygments [Bra08b] lexer to color some of the extra features of the `ipython` console.

```
In [156]: fig = plt.figure()
In [157]: ax1 = fig.add_subplot(211)
In [158]: ax2 = fig.add_axes([0.1, 0.1, 0.7, 0.3])
In [159]: ax1
Out[159]: <matplotlib.axes.Subplot instance at 0xd54b26c>
In [160]: print fig.axes
[<matplotlib.axes.Subplot instance at 0xd54b26c>, <matplotlib.axes.Axes instance at 0xd3f0b2c>]
```

Framework

These new extensions are part of a complete turnkey framework for building Sphinx documentation geared specifically to Scientific Python applications. The framework is available as a subproject in matplotlib's source code repository[4] and can be used as a starting point for other projects using Sphinx.

This template is still in its early stages, but we hope it can grow into a project of its own. It could become a repository for the best ideas from other Sphinx-using projects and act as a sort of incubator for future features in Sphinx proper. This may include the web-based documentation editor currently being used by the Numpy project.

Future directions

intersphinx

Sphinx recently added "intersphinx" functionality, which allows one set of documentation to reference methods and classes etc. in another set. This opens up some very nice possibilities once a critical mass of Scientific Python tools standardize on Sphinx. For instance, the histogram plotting functionality in matplotlib could reference the underlying methods in Numpy, or related methods in Scipy, allowing the user to easily learn about all the options available without risk of duplicating information in multiple places.

Acknowledgments

---

[2]The license for TeX allows this, as long as we don't call it "TeX".
[3]https://trac.fysik.dtu.dk/projects/ase/browser/trunk/doc/mathpng.py
[4]http://matplotlib.svn.sourceforge.net/viewvc/matplotlib/trunk/py4science/examples/sphinx_template/

Figures

```
matplotlib.pyplot.acorr(*args, **kwargs)
```
   call signature:

```
acorr(x, normed=False, detrend=mlab.detrend_none, usevlines=False,
    maxlags=None, **kwargs)
```

Plot the autocorrelation of *x*. If *normed = True*, normalize the data but the autocorrelation at 0-th lag. *x* is detrended by the *detrend* callable (default no normalization).

Data are plotted as `plot(lags, c, **kwargs)`

Return value is a tuple (*lags, c, line*) where:

- *lags* are a length 2*maxlags+1 lag vector

- *c* is the 2*maxlags+1 auto correlation vector

- *line* is a `Line2D` instance
    returned by `plot()`

The default *linestyle* is None and the default *marker* is `'o'`, though these can be overridden with keyword args. The cross correlation is performed with `numpy.correlate()` with *mode* = 2.

If *usevlines* is *True*, `vlines()` rather than `plot()` is used to draw vertical lines from the origin to the acorr. Otherwise, the plot style is determined by the kwargs, which are `Line2D` properties. The return value is a tuple (*lags, c, linecol, b*) where

- *linecol* is the `LineCollection`
- *b* is the *x*-axis.

maxlags is a positive integer detailing the number of lags to show. The default value of *None* will return all `(2*len(x)-1)` lags.
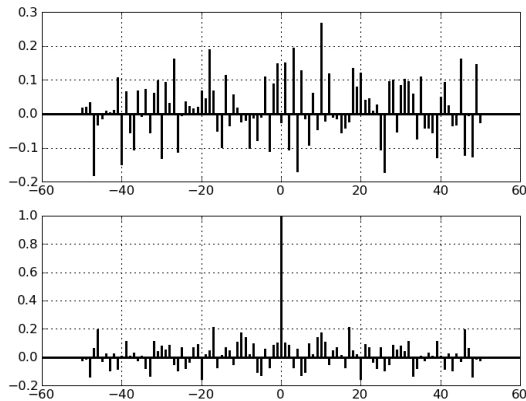
See the respective `plot()` or `vlines()` functions for documentation on valid kwargs.

Example:

`xcorr()` above, and `acorr()` below.

Example:

[source code, png, pdf]

Additional kwargs: hold = [True|False] overrides default hold state

The HTML output of the `acorr` *docstring.*

References

[Bra08]   G. Brandl. 2008. *Sphinx: Python Documentation Generator.* http://sphinx.pocoo.org/

[Bra08b]   G. Brandl. 2008. *Pygments: Python syntax highlighter.* http://pygments.org

[Cer07]   D. P. Cervone. 2007. *jsMath: A Method of Including Mathematics in Web Pages.* http://jsmath.sourceforge.net/

[Gan06]   E. Gansner, E. Koustsofios, and S. North. 2006. *Drawing graphs with dot.* http://www.graphviz.org/Documentation/dotguide.pdf

[Goo06]   D. Goodger. 2006. *reStructuredText: Markup Syntax and Parser Component of Docutils.* http://docutils.sourceforge.net/rst.html

[Hun08]   J. Hunter, et al. 2008. *matplotlib: Python 2D plotting library.* http://matplotlib.sourceforge.net/

[Knu86]   D. E. Knuth. 1986. *Computers and Typesetting, Volume B: TeX: The Program.* Reading, MA: Addison-Wesley.

[Lop08]   E. Loper. 2008. *Epydoc: Automatic API Documentation Generation for Python.* http://epydoc.sourceforge.net/

[Mar01]   A. Martelli. 2001. *Recipe 52305: Yet Another Python Templating Utility (YAPTU).* From Python Cookbook. *<http://code.activestate.com/recipes/52305/>*

[STI08]   STI Pub Companies. 2008. *STIX Font Set Project.* http://www.stixfonts.org

[Yee01]   K.-P. Yee. 2001. *pydoc: Python documentation generator and online help system.* http://lfw.org/python/pydoc.html & http://www.python.org/doc/lib/module-pydoc.html.