Matplotlib - Foundation of Scientific Visualization in Python

July 31, 2019

1 Goals

Matplotlib[1] is the fundamental data visualization library for the scientific Python Ecosystem, used in conjunction with other foundational tools like NumPy and SciPy [2] by over a million[1] users. Matplotlib is used across a wide spectrum of fields, including bio-medical imaging, microscopy, and genomics [3–13], and we expect this user base to grow as Python continues to be adopted in the life sciences. Matplotlib has been actively developed and maintained by a vibrant, primarily volunteer, community over the last 16 years; However, given the scale, scope, and importance of the project, we are at the limit of what we can sustainably maintain and develop with mostly volunteer effort.

To strengthen the library and lay the groundwork for future versions, this proposal asks for 2.4 person years of developer effort to support:

a) Fostering community, documentation, and tools to grow a rich ecosystem of domain-specific libraries built using Matplotlib
b) Maintaining the library and contributor base
c) Developing a comprehensive plan to evolve the core architecture

1.1 Fostering downstream projects

We will engage with libraries in the life sciences that are using Matplotlib for their visualization to ensure that we are addressing their problems and to collaborate on prototyping an end-to-end solution, including data, plot representations, and a high-level user API. In particular we plan to engage with `scikit-learn`[2], `CellProfiler`[3], `scanpy`[3], `starfish`[3], `nipy`, and `scikit-image`[3] [3–5]

Our goal is to make these libraries and ones like them easier to write. The most common visualizations in a domain need to be fluid for the end-practitioners, with the "obvious" customization options exposed. Much of the domain-specific specialization is carried in the structure, semantics and assumptions of the data, and in the standard visualizations of the domain. These specializations can vary widely, in contradictory ways, between domains. Because no high-level API can simultaneously satisfy all of the visualization needs, there will always be a need for domain-specific visualization libraries.

Through maintaining the code base and developing the architecture for the next version, we will generate API guidelines so that downstream libraries inter-operate well and "feel" similar. By building out community, documentation, and tools, we will provide downstream library developers resources for getting the most out of building their tools on top of Matplotlib.

[1] Estimated from `pypi` and `conda` download numbers and unique monthly visitors to the documentation website
[2] Also applying for Essential Open Source Software for Science
[3] Currently funded by CZI

1.2 Maintenance and Growth of Library and Community

Matplotlib is a community-driven project, but we have grown to the point where we need developers with the time to organize, plan, and make decisions. Issues and PRs are submitted faster than our volunteers can review them; for the past few years we have resolved approximately 40 Issues and 30 PRs a month, but every month about 50 new Issues are opened and 40 PRs are proposed.

We have accumulated about 1200 open Issues and 300 open PRs due to this imbalance. There may be critical bug reports or insightful feature requests among the former, while among the latter are useful contributions or bug fixes that would improve Matplotlib for direct users and downstream packages. The backlog is discouraging for new and occasional contributors and distracting for core developers.

To maintain Matplotlib's health we need to:

- Fix critical bugs and regressions
- Triage backlog of Issues and PRs in terms of topic, difficulty, and urgency and promptly triage newly opened Issues and PRs
- Maintain backward compatibility and extensively document intentional changes
- On-board new contributors to sustain and diversify developer team
- Manage discussions about proposed enhancements, features, and API changes

The requested support for developers is intended to complement and facilitate, not replace, crucial volunteer work. We aim to better co-ordinate and nurture their efforts, with the goal of growing and sustaining a diverse community of volunteer and paid expert contributors.

1.3 Road-map and Architecture

Matplotlib needs sustained attention to update the library's architecture for the next decade. The architecture[4] of Matplotlib was developed over 15 years ago [1] and has served us well, but it does not reflect recent developments in software design, data structures, and visualization. We aim to take advantage of these developments to improve the consistency, composability, and discoverabilty of the API. The ultimate goal is facilitating rich interactive domain-specific visualizations.

1.3.1 Homogenization of the Application Programming Interface (API)

The library has grown organically over time through the contributions of over 900 individuals, accumulating many small inconsistencies in the API. For example, similar methods have different argument order, e.g., `ax.text(x, y, s)` vs `ax.annotation(s, (x, y))`, and some keyword (named) arguments can be singular or plural, e.g., `color` vs `colors`. These subtle issues add friction for users, but are hard to fix without breaking existing code for someone in our large user base. Our goal is to minimize breakage, while still unifying the API. Taking into account **all** of the public function signatures, we will carefully consider which to leave as they are, which to deprecate, and which to replace.

1.3.2 API generalization

Matplotlib has two main user-facing APIs, `pyplot` API and the `Object Oriented` (`OO`) API. The `pyplot` API closely follows MATLAB and is built around the concept of a global "current Axes" drawing surface. Although `pyplot` can be convenient for interactive terminal usage, it is easy to lose track of which Axes is the "current Axes"; particularly when library functions, where code is typically hidden from the user, are also using `pyplot`. This uncertainty as to which Axes is the

[4] `https://www.aosabook.org/en/matplotlib.html`

"current Axes" frequently results in data being plotted to wrong and surprising places. On the other hand, the `OO` interface is more flexible and explicit but verbose.

The main namespace for plotting methods in the `OO` API is methods on the `Axes` class, which leads to three issues:

1. Third party domain-specific packages can never feel "First Class" as their plotting functions will not be implemented as `Axes` methods like the "built in" ones.

2. Some visualizations require plotting to multiple `Axes`es, which doesn't fit the model in which plots are made via `Axes` methods.

3. There are over 250 methods on the `Axes`—all of the plotting and some additional `Axes`—which makes it extremely hard to use tab-completion to search for a method.

To address these issues, we will move to a primary API in which top-level functions take in data, style, and `Axes`(es) and return `Artist`s. These functions will use consistent naming and call conventions, as discussed in 1.3.1 and return the `Artist`s discussed in section 1.3.3. From the large pool of plotting functions, augmented in collaboration with downstream libraries, we will curate domain-specific namespaces to facilitate discoverability.

This large-scale refactoring requires dedicated developer time to carefully consider and engage with our users about the consequences.

1.3.3 Rich, Semantic `Artists`

`Artist`s are the "middle layer" of Matplotlib that encodes a plot's type, look, and data. `Artist`s know how to translate user specifications to rendered output, and users interact with the `Artist` objects to update the data or aesthetics.

We need to embrace composite artists to provide a smarter, more uniform `Artist` API. Matplotlib has a mix of "primitive" `Artist`s (e.g., lines, images, and text) and "composite" artists (e.g., the whole `Axes`). The mapping between the user API and the `Artist`s can be one-to-one, but often one user call will generate many decoupled primitive `Artist`s. For example, `hist` displays a histogram, notionally a single entity to the user, but returns a list of independent `Rectangle` `Artist`s (one per bin). If the user wants to update the data or the aesthetics, they must adjust each `Rectangle` independently. Instead, the functions discussed in 1.3.2 should return objects with a uniform update interface, making interactive visualizations easier to develop.

1.3.4 Data Model

Matplotlib needs a consistent way for `Artist`s to access their underlying data. Currently, each plotting method and `Artist` handles sanitizing and storing data independently. Hence, some common functionality, such as handling data with attached units (e.g., degrees Celsius, dates), is scattered throughout the code base. This leads to inconsistencies across the library and makes it difficult for users to write code that updates interactive explorations and animations. Matplotlib also cannot exploit "Structured data"–combining multiple pieces of (possibly heterogeneous) data with labels and metadata into single data structure; instead users must extract data elements and reassemble them as arguments to Matplotlib plotting functions.

We will develop a data access API such that each `Artist` will have a single `Data` object responsible for holding the data. This model will be the technical underpinning to handle, exploit, and update structured data in a coherent fashion. By removing direct data storage from the `Artist`s and defining an API for data sources we will enable:

- native consumption of structured data
- smart down-sampling of plotted data based on view limits
- seamless updating of the underlying data, either interactively or via streams
- use of alternative data sources such as database queries or analytic functions

This will decouple the implementation of the data sources from the `Artist`s. Downstream libraries will be able to provide sophisticated data sources to the core library `Artists`, build complex `Artist`s that use those sources, and create specialized forms of the data model.

2 Expected outcomes, success evaluation and metrics

2.1 Maintenance

Quantitatively evaluating maintenance work can be tricky—some Issues or PRs take minutes to review while others can take days to months of effort—but we believe that there is value at looking at the number of open Issues and PRs. We will reduce this number by closing Issues and PRs at a faster rate than they are opened until a reasonable equilibrium is reached. With the introduction of paid developers, NumPy has had success in reversing the ever increasing trends in the number of Issues and Pull Requests[5]. Once the backlog is handled, we aim to have all new Issues and Pull Requests labeled within 7 days of being opened.

Improved contributor onboarding should yield an increase in the percentage of new contributor PRs that get resolved, a decrease in the amount of time it takes for resolution, and an increase the number of PRs submitted by non core developers.

2.2 Road-map and Architecture

We will write a white paper with a road map documenting the proposed design, critical use-cases, and requirements for the data model and API overhaul.

3 Downstream Libraries

To validate the design, we will develop end-to-end prototypes targeting one or more of the life-science libraries discussed in 1.1 and qualitatively assess downstream library developers engagement with these prototypes.

4 Work Plan

We request funding for the following:

- Fund Thomas Caswell's position at 40%. Caswell is the Lead Developer of Matplotlib and an Associate Computational Scientist at Brookhaven National Laboratory. His long-term experience, API design expertise, and project leadership are critical to the success of the work in this proposal. He will work on all aspects of the proposal.
- Hannah Aizenman's position for 12 months. Aizenman has been a core developer to Matplotlib for three years. She has helped lead outreach efforts and is the author of a recent feature set. She is a PhD candidate in Computer Science, studying visualization at The Graduate Center, CUNY. Her work on the architecture of Matplotlib will be the basis of

[5] https://github.com/seberg/numpy_talk_plots/blob/master/plots_used_in_talk/issues_prs_delta.pdf

her PhD thesis. Aizenman will take the lead on the data model design and new-contributor on-boarding.
- Summer support for Michael Grossberg, Aizenman's PhD advisor.
- 12 months of a yet-to-be identified software engineer to support all aspects of the proposal but focusing on maintenance, prototyping, and engaging down-stream libraries.
- Travel to key conferences (such as SciPy or PyCon) and for in-person meetings

We want to use this dedicated effort to leverage and empower the Matplotlib developer community. In terms of direct work, equal amounts of time will be spent mentoring and reviewing code from community members and directly implementing features or fixing bugs. All of the design work will be done in public with input from the community. Part of this work is to develop the project Road Map.

5 Existing Support

Thomas Caswell has 4hrs/wk from Brookhaven National Lab to work on Matplotlib.

References

[1] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[2] E. Jones, T. Oliphant, P. Peterson, *et al.*, "SciPy: Open source scientific tools for Python," 2001–. [Online; accessed ¡today¿].

[3] A. E. Carpenter, T. R. Jones, M. R. Lamprecht, C. Clarke, I. H. Kang, O. Friman, D. A. Guertin, J. H. Chang, R. A. Lindquist, J. Moffat, P. Golland, and D. M. Sabatini, "Cellprofiler: image analysis software for identifying and quantifying cell phenotypes," *Genome Biology*, vol. 7, p. R100, Oct 2006.

[4] F. A. Wolf, P. Angerer, and F. J. Theis, "Scanpy: large-scale single-cell gene expression data analysis," *Genome Biology*, vol. 19, p. 15, Feb 2018.

[5] S. van der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, and T. a. Yu, "scikit-image: image processing in python," *PeerJ*, vol. 2, p. e453, June 2014.

[6] N. Segata, J. Izard, L. Waldron, D. Gevers, L. Miropolsky, W. S. Garrett, and C. Huttenhower, "Metagenomic biomarker discovery and explanation," *Genome Biology*, vol. 12, p. R60, Jun 2011.

[7] R. N. Gutenkunst, R. D. Hernandez, S. H. Williamson, and C. D. Bustamante, "Inferring the joint demographic history of multiple populations from multidimensional snp frequency data," *PLOS Genetics*, vol. 5, pp. 1–11, 10 2009.

[8] T. Hashimshony, F. Wagner, N. Sher, and I. Yanai, "Cel-seq: Single-cell rna-seq by multiplexed linear amplification," *Cell Reports*, vol. 2, no. 3, pp. 666 – 673, 2012.

[9] J. Köster and S. Rahmann, "Snakemake—a scalable bioinformatics workflow engine," *Bioinformatics*, vol. 28, pp. 2520–2522, 08 2012.

[10] T. M. Carlile, M. F. Rojas-Duran, B. Zinshteyn, H. Shin, K. M. Bartoli, and W. V. Gilbert, "Pseudouridine profiling reveals regulated mrna pseudouridylation in yeast and human cells," *Nature*, vol. 515, pp. 143 EP –, Sep 2014.

[11] A. Laganowsky, E. Reading, T. M. Allison, M. B. Ulmschneider, M. T. Degiacomi, A. J. Baldwin, and C. V. Robinson, "Membrane proteins bind lipids selectively to modulate their structure and function," *Nature*, vol. 510, pp. 172 EP –, Jun 2014.

[12] X. Jiang, S. Shen, C. R. Cadwell, P. Berens, F. Sinz, A. S. Ecker, S. Patel, and A. S. Tolias, "Principles of connectivity among morphologically defined cell types in adult neocortex," *Science*, vol. 350, no. 6264, 2015.

[13] A. Abraham, F. Pedregosa, M. Eickenberg, P. Gervais, A. Mueller, J. Kossaifi, A. Gramfort, B. Thirion, and G. Varoquaux, "Machine learning for neuroimaging with scikit-learn," *Frontiers in Neuroinformatics*, vol. 8, p. 14, 2014.