Matplotlib - Foundation of Visualization in Python

July 30, 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[3] users. Matplotlib is used across a wide spectrum of fields, including bio-medical imaging, microscopy, and genomics [1–11], and we expect this user base to continue to grow as Python is further 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 a primarily volunteer effort.

This proposal asks for 2.4 person years of developer effort for Matplotlib, laying the groundwork for the next 16 years by supporting:

a) Maintenance of the library, including labeling new and existing Issues and Pull Requests (PRs).
b) Developing a comprehensive plan to evolve the core architecture of Matplotlib.
c) Developing the tools, documentation, and community to foster a rich eco-system of domain-specific plotting tools built on Matplotlib.

[1] `http://matplotlib.org`

[2] `https://www.scipy.org/about.html`

[3] Estimated from `pypi` download numbers, `conda` download numbers, and the number of unique monthly visitors to the documentation website

1.1 Maintenance and Growth of Library and Community

Matplotlib is a community-driven project, but we have grown to the point where we need supported developers with the time to organize, plan, and make timely decisions. Currently, new Issues and Pull Requets (PRs) are being submitted faster than our volunteers can review them. For the past few years, we have closed about 43 Issues and merged 33 PRs month; and however every month about 51 new issues are opened and 40 PRs are proposed. We have accumulated about 1200 open issues and 300 open PRs. Among the former there may be critical bug reports or insightful feature requests and 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 or occasional contributors, and distracting for core developers.

To maintain Matplotlib's health we need to:

- Label and review current backlog of Issues and PRs in terms of topic, difficulty, and urgency. Promptly label and review newly opened Issues and PRs
- Fix critical bugs and regressions promptly.
- On-board new contributors to the project team; this is critical to sustaining and diversifying our developer community.
- Maintain backward compatibility. If we break the API, ensure it is intentional and well documented.
- Manage discussions about proposed enhancements, features, and API changes.

The requested support for developers is intended to complement and facilitate, not replace, the crucial work by volunteers. We aim to better co-ordinate and nurture their efforts, with the goal of growing and sustaining a diverse community of expert contributors, both volunteer and paid. We view this proposal as the start of a new phase for Matplotlib, in which we will need to find support from a variety of sources in the coming years.

1.2 Road-map and Architecture

Matplotlib needs sustained attention to update the library's architecture for the next decade. The current architecture[4] of Matplotlib was developed over 15 years ago [12] and has served us well, a testament to its initial design, but it does not reflect recent developments in software design, data

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

structures, and visualization. We will improve the consistency, composability, and discoverabilty of the API, taking better advantage of modern data structures. The ultimate goal is facilitating rich interactive domain-specific visualizations.

1.2.1 Homogenization of the Application Programming Interface (API)

The library has grown organically over time through the contributions of many people (over 900 individuals), and the code has accumulated many small inconsistencies in the API. Similar methods have different argument order, e.g., `ax.text(x, y, s)` vs `ax.annotation(s, (x, y))`, and some keyword 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 somewhere in our large user base. Our goal is to minimize breakage, but still unify the API. Taking into account **all** of the APIs, we will carefully consider which to leave as they are, which to deprecate, and which to replace.

1.2.2 API generalization

Currently Matplotlib has two main user-facing APIs: the `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". While `pyplot` is convenient for interactive terminal usage, the global state frequently produces surprising and undesired coupling when used by libraries. On the other hand the `OO` interface is more flexible and explicit but more verbose. The main name space for plotting methods in the `OO` API is methods on the `Axes` class, which leads to three issues with the API. First, 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. Second, some visualizations require putting `Artist`s on multiple `Axes`; this doesn't fit the model in which Artists are made via `Axes` methods. Lastly, there are over 250 methods on the `Axes`—all of the plotting methods and some additional `Axes` specific methods—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`(s) and return `Artist`s. These functions will use consistent naming and call conventions, as discussed in 1.2.1 and return the `Artist`s discussed in section 1.2.3. From the large pool of plotting functions, augmented in collaboration with downstream libraries, we will curate domain-specific name spaces to enable users to discover the functionality they need.

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

1.2.3 Rich, Semantic `Artists`

`Artist`s are the "middle layer" of Matplotlib that encode user-intent, style, and data. To update the style or data users interact with the `Artist` objects, and `Artist`s know how to translate that intent on to the rendered output.

We need to embrace composite artists to provide a smarter, more uniform `Artist` API. Currently, Matplotlib has a mix of "primitive" `Artist`s (e.g. lines, images, and text) and "composite" artists (e.g., the whole `Figure`). Presently 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 style, they must adjust each `Rectangle` independently. Instead, the functions discussed in 1.2.2 should return objects with a uniform interface for updating their data and style, making interactive visualizations easier for all users to develop.

1.2.4 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. These lead to inconsistencies across the library and makes it difficult for users to write code that updates the data or style for interactive exploration and animation.

"Structured data"–combining multiple pieces of (possibly heterogeneous) data along with labels and metadata into single data structure–is revolutionizing science, but is not fully exploited by Matplotlib. Instead, extracting data elements and reassembling them as arguments for Matplotlib plotting functions must be done at the level of user code.

We will develop a data access API such that each `Artist` will have a `Data` object responsible for actually holding the data. This model will support the base Matplotlib library and, more importantly, will be the technical underpinning to handle, exploit, and update structured data in a coherent fashion. By removing the 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 streaming or interactively;
- use of alternative data sources such as database queries or analytic functions

This will decouple the development of the data sources from the `Artists`. Downstream libraries will be able to provide sophisticated data sources to the `Artists` in the core library or sophisticated `Artists` that use the data sources from core.

1.3 Coordination with downstream projects

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 carried in the semantics of the structured data and the specific visualization needs of the domain. These vary widely between domains; no library can meet all of the needs simultaneously so there will always be a need for domain-specific visualization libraries. Our goal is to make these libraries as easy to write as possible. We will develop a set of guidelines for the APIs so that downstream libraries "feel" similar and inter-operate well, as well as tools and templates to make the process of developing a new library painless.

We will engage with libraries in the life sciences that are currently using Matplotlib for their visualization to ensure that we are addressing their problems and to collaborate on prototyping a end-to-end solution. In particular we plan to engage with `scikit-learn`[5], `CellProfiler`[6], `scanpy`[6], `starfish`[6], `nipy`, and `scikit-image`[6]

2 Expected outcomes, success evaluation and metrics

2.1 Issue and PR curation

Quantitatively evaluating maintenance work can be tricky—some Issues or PRs can 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, closing Issues and PRs faster than they are opened until a reasonable equilibrium is reached. With the introduction

[5] Also applying for Essential Open Source Software for Science

[6] Currently funded by CZI

of paid developers, NumPy has had success in reversing the ever increasing trends in the number of Issues and Pull Requests[7].

We will evaluate and label every open Issue and Pull Request to assign an action, a priority, and an estimated difficulty. Once the backlog is handled, we will aim to have all new Issues and Pull Requests labeled within 7 days of being opened.

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.

To validate the design, we will develop end-to-end prototypes targeting one or more of the life-science libraries discussed above to validate the design.

3 Work Plan

We request funding for the following:

- Fund Thomas Caswell's position at 40%. Caswell is currently 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-contributor to Matplotlib for three years. She has helped lead outreach efforts such as Matplotlib's Google Summer of Code participation and is the author of a major recent feature set (support for string-categorical values). 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 her PhD thesis. Aizenman will take the lead on the data model design and new-contributor on-boarding.
- 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 Scientific and Python conferences (such as SciPy or PyCon) and for in-person meetings if required.

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

We want to use this dedicated effort to leverage and empower the Matplotlib developer community. In terms of direct work on the code base, 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.

4 Existing Support

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

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

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

[8] 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.

[9] 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.

[10] 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.

[11] 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.

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