Matplotlib - Foundation of Visualization in Python

July 28, 2019

1 Goals

Matplotlib[1] is the fundamental data visualization library for the scientific Python ecosystem, used by over a million[2] users in conjunction with other foundational tools like Numpy and SciPy[3]. 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. There are many downstream packages that build on Matplotlib to implement domain-specific plotting tools.

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 sustainable maintain and develop with 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 curating 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] Estimated from `pypi` download numbers, `conda` download numbers, and the number of unique monthly visitors to the documentation website

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

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 PRs are being submitted faster than our volunteers can review them. Over the past few years we have merged PRs and closed Issues at about [X per month], but about [Y] new issues and PRs are opened monthly; currently we have 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:

- Curate the current backlog of Issues and PRs in terms of topic, difficulty, and urgency.
- Label and review newly opened Issues and PRs promptly.
- 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 do break API, ensure it is intentional and well documented.
- Manage discussions about proposed enhancements, features, and breaking 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 design the library's architecture for the next decade. The current architecture[4] of Matplotlib was developed 15 years ago [12]. That it is still in use is a testament to its initial design; but that design does not reflect recent developments in data

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

structures, software design, and visualization. Matplotlib does not natively know how to exploit structured (e.g. `pandas` or `xarray`) or streaming data. Some of the design choices about data structures and name space organization are becoming unwelcome constraints as Matplotlib scales to more domains. While there are many downstream domain-specific libraries that are built on top of Matplotlib, interoperability between them is problematic.

1.2.1 Homogenization of the Application Programming Interface (API)

The library has grown organically over time through the contributions of many people (approx. 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 the `pyplot` API is convenient for interactive usage, the global state frequently produces surprising and undesired coupling when used in 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). During this refactoring we will use consistent naming and call conventions, as discussed in 1.2.1. These functions will return the rich composites discussed in section 1.2.3

and consume the data objects discussed in section 1.2.4. From the large pool of plotting functions we will curate domain-specific name spaces, which can be augmented by downstream libraries to enable users to discover the functionality they need.

This large-scale refactoring requires dedicated developer time to carefully consider the consequences and trade offs.

1.2.3 Rich, Semantic `Artists`

`Artists` 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 `Artists` know how to translate that intent on to the rendered image.

Currently, Matplotlib has a mix of "primitive" `Artists` (e.g. lines, images, and text) and "composite" artists (e.g., the whole `Figure`). The mapping between the user API and the `Artists` can be one-to-one, but often one user call will generate many `Artists`. For example, `hist` displays a histogram, but returns a list of independent `Rectangle Artists` (one per bin). If the user wants to update the data or the style, they must adjust each `Rectangle` independently. While technically possible, this makes developing interactive figures and animations is significantly harder than it should be. The functions discussed in 1.2.2 should return objects that have a uniform interface to update their data and style to make interactive visualization development accessible to all users.

1.2.4 Data Model

"Structured data" combines multiple pieces of, possibly heterogeneous, data along with labels and metadata into single data structure. Structured data is is not natively supported by Matplotlib, users must split the structures up and pass the components individually into plotting methods, destroying the structure. Further, each plotting method and `Artists` handle sanitizing and storing data independently. Common functionality, such as handling data with attached units (e.g., degrees Celsius, dates), is scattered throughout the code base and each `Artist` stores its data slightly differently. This leads to inconsistencies across library and makes it difficult to uniformly update the data or style for interactive exploration, streaming, and animation use cases.

We will re-organize the internal data representation in Matplotlib to a model appropriate for the base Matplotlib library and, more importantly, to be the technical underpinning to handle, exploit, and update structured data in a coherent fashion. By removing the direct data storage from the `Artists` 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 update the underlying data, either streaming or interactively;
- non-materialized data sources such as database queries or analytic functions

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

1.2.5 Additional Export Methods

Matplotlib `Figure`s can be render to either raster or vector file formats. From there it is displayed to an interactive window or saved to disk and be used like any other image file. There is currently no good way to "reopen" a Matplotlib `Figure` or export it to another plotting library, such as `bokeh`, `d3` or `QtCharts`. Due to the way Matplotlib internals are implemented, it is difficult to take advantage GPUs to accelerate drawing.

To address these problems we will investigate adding two additional export paths. One at a high-level, suitable for a Matplotlib-specific file format and interoperability with other high-level plotting libraries, and one at a low level scene-graph level, suitable to pass to a GPU.

1.3 Coordination with downstream projects

The most common visualizations in a domain need to be one or two simple lines of code for the end-practitioners with the "obvious" customization options surfaced. However, across domains the structure of the user data and the set of "obvious" assumptions vary wildly. It is impossible to have one API that serves all users equally so there will always be a need for domain specific visualization libraries. These libraries can make assumptions about the structure of the data and the standard visualizations made within that field. Our goal is to make these libraries as thin and easy to write as possible.

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

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

[6] Currently funded by CZI

2 Expected outcomes, success evaluation and metrics

2.1 Issue and PR curation

Quantitatively evaluating maintenance work can be tricky. Not all Issues and PRs are equivalent, some can take minutes to resolve where as others can take days to months of effort to resolve.

Never-the-less, we believe the net number of open Issues and PRs per month is a meaningful number. Currently Issues and PRs are opened at a greater than we can resolve them. If we are successful that rate at which Issues and PRs accumulate will be reduced and ideally we will reduce the size of the backlog. [7].

We will evaluate and label every open Issue and PR determining: assigning an action, a priority, and an estimated difficulty. Once that is done, we will aim to have all new Issues and PRs labeled within 7 days of being opened.

2.2 Road-map and Architecture

We will write a white paper and 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.

3 Work Plan

The funds will be paid to:

- 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.
- Fund Hannah Aizenman's position for 12 months. Aizenman has been a core-contributor Matplotlib for three years 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 City College of New York. Her work on the architecture of Matplotlib will be the basis of

[7] NumPy has had success in reducing their backlog with full-time developers `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.

- Fund 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.

We want to use this dedicated effort to leverage and empower the Matplotlib developer community. In terms of direct work on the code base an equal amount of time will be spent mentoring and reviewing code from community members rather than directly implementing features or fixing bugs.

Part of this work is to develop the project Road Map.

All of the design work will be done in public with input from the community.

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.