Matplotlib - Foundation of Visualization in Python

July 28, 2019

1 Goals

Matplotlib (`http://matplotlib.org`) is the fundamental data visualization library for the scientific Python stack of tools, providing plotting functionality used by over a million[1] scientists in conjunction with other foundational tools like Numpy and SciPy (`https://www.scipy.org/about.html`). It is used across a wide spectrum of fields, including bio-medical imaging, microscopy, and genomics [CJL$^+$06, WAT18, vdWSNI$^+$14] [SIW$^+$11, GHWB09, HWSY12, KR12, CRDZ$^+$14, LRA$^+$14, JSC$^+$15, APE$^+$14], and we expect the user base to continue to grow as Python is adopted by more scientists 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 can be developed and maintained with a primarily volunteer effort.

This proposal supports 2.4 person years of developer time for Matplotlib, laying the groundwork for guiding the project through the next 16 years:

a) Maintenance of the library, including curating new and existing Issues and Pull Requests.
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.

1.1 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 Requests 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 latter are useful contributions and bug fixes that, possibly with additional attention and polish, could 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 do the following:

- Curate the current backlog of Issues and Pull Requests in terms of topic, difficulty, and urgency.
- Label and review newly opened Issues and Pull Requests promptly.

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

- 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 faciliate, 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[2] of Matplotlib was developed 15 years ago [Hun07]. That it is still in use is a testament to its initial design; but that design does not reflect recent developments in data 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 convenient for quick interactive usage, it frequently produces surprising and undesired coupling between in the code when used in libraries. On the other hand the `OO` interface is more explicit and flexible but more verbose. The main name space for plotting methods is 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" plots. Second, some visualizations require putting `Artists` 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.

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

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, that can be used by downstream libraries. 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 is a huge change in the API that will require dedicated developer time to carefully consider 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 turn the user's input into colored pixels. Currently, Matplotlib has a mix of "primitive" `Artist`s (e.g. lines, images, and text) and "composite" artists (e.g., the whole `Figure`).

We need to develop and apply a smarter, more uniform composite artist API. 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, 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

"Structured data" combines multiple data elements along with labels, metadata, and the relationships among the components into a single data structure. Such structures are not natively supported by Matplotlib. Instead, extracting data elements and reassembling them as arguments for Matplotlib plotting functions must be done at the level of user code. Further, 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 to write code that updates the data or style for interactive exploration, streaming, and animation.

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 `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

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 render to either raster or vector file formats, displaying the result in an interactive window and/or saving it to disk. There is currently no good way to "reopen" a Matplotlib `Figure` or export it to another plotting library, such as `bokeh`, `d3` or `QtCharts`. In addition, due to the way Matplotlib internals are implemented, it is difficult to take advantage of GPUs to accelerate drawing.

To address these problems we will investigate adding two additional export paths. One, at a high-level, will be suitable for subsequent re-input to Matplotlib and for interoperability with other high-level plotting libraries. The second, at a low scene-graph level, will be suitable for passing 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 exposed. Our goal is to make these libraries as thin and easy to write as possible. This means there will always be a need for domain-specific visualization libraries. Much of the domain-specific specialization is carried in the semantics of the structured data and the specific visualization needs of the domain.

To this end we will identify and engage with downstream libraries in the life sciences that are currently using Matplotlib for their visualization to identify their pain points and ensure that we are actually solving their problems. In particular we plan to engage with `scikit-learn`[3], `CellProfiler`[4], `scanpy`[4], `starfish`[4], `nipy`, and `scikit-image`[4]

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 whil others can take days to months of effort—but we believe that there is value at looking at the number of open Issues and Pull requests. We will reduce this number, ideally closing Issues and Pull Requests faster 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].

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.

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

[4]Currently funded by CZI

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

3 Work Plan

We request funding for the following:

- 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 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 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.

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.

4 Existing Support

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

References

[APE$^+$14] Alexandre Abraham, Fabian Pedregosa, Michael Eickenberg, Philippe Gervais, Andreas Mueller, Jean Kossaifi, Alexandre Gramfort, Bertrand Thirion, and Gael Varoquaux. Machine learning for neuroimaging with scikit-learn. *Frontiers in Neuroinformatics*, 8:14, 2014.

[CJL$^+$06] Anne E. Carpenter, Thouis R. Jones, Michael R. Lamprecht, Colin Clarke, In Han Kang, Ola Friman, David A. Guertin, Joo Han Chang, Robert A. Lindquist, Jason Moffat, Polina Golland, and David M. Sabatini. Cellprofiler: image analysis software for identifying and quantifying cell phenotypes. *Genome Biology*, 7(10):R100, Oct 2006.

[CRDZ$^+$14] Thomas M. Carlile, Maria F. Rojas-Duran, Boris Zinshteyn, Hakyung Shin, Kristen M. Bartoli, and Wendy V. Gilbert. Pseudouridine profiling reveals regulated mrna pseudouridylation in yeast and human cells. *Nature*, 515:143 EP –, Sep 2014.

[GHWB09] Ryan N. Gutenkunst, Ryan D. Hernandez, Scott H. Williamson, and Carlos D. Bustamante. Inferring the joint demographic history of multiple populations from multidimensional snp frequency data. *PLOS Genetics*, 5(10):1–11, 10 2009.

[Hun07] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[HWSY12] Tamar Hashimshony, Florian Wagner, Noa Sher, and Itai Yanai. Cel-seq: Single-cell rna-seq by multiplexed linear amplification. *Cell Reports*, 2(3):666 – 673, 2012.

[JSC$^+$15] Xiaolong Jiang, Shan Shen, Cathryn R. Cadwell, Philipp Berens, Fabian Sinz, Alexander S. Ecker, Saumil Patel, and Andreas S. Tolias. Principles of connectivity among morphologically defined cell types in adult neocortex. *Science*, 350(6264), 2015.

[KR12] Johannes Kster and Sven Rahmann. Snakemakea scalable bioinformatics workflow engine. *Bioinformatics*, 28(19):2520–2522, 08 2012.

[LRA$^+$14] Arthur Laganowsky, Eamonn Reading, Timothy M. Allison, Martin B. Ulmschneider, Matteo T. Degiacomi, Andrew J. Baldwin, and Carol V. Robinson. Membrane proteins bind lipids selectively to modulate their structure and function. *Nature*, 510:172 EP –, Jun 2014.

[SIW$^+$11] Nicola Segata, Jacques Izard, Levi Waldron, Dirk Gevers, Larisa Miropolsky, Wendy S. Garrett, and Curtis Huttenhower. Metagenomic biomarker discovery and explanation. *Genome Biology*, 12(6):R60, Jun 2011.

[vdWSNI$^+$14] Stfan van der Walt, Johannes L. Schnberger, Juan Nunez-Iglesias, Franois Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Gouillart, and Tony and Yu. scikit-image: image processing in python. *PeerJ*, 2:e453, June 2014.

[WAT18] F. Alexander Wolf, Philipp Angerer, and Fabian J. Theis. Scanpy: large-scale single-cell gene expression data analysis. *Genome Biology*, 19(1):15, Feb 2018.