Deliverable 4

Team 28 - Big Software Energy 2.0
Muhammad Farooq, Jadin Luong, Vincent Teng, Aster Wu
April 4th, 2020

User Guide

OVERVIEW

The following feature that has been implemented grants users the ability to toggle the legend of the plot they have constructed by simply clicking the newly created “Toggle Legend” button at the navigation toolbar of the figure.

INSTALLATION

Before you are able to use this new feature, you are required to install the new version of matplotlib that contains this feature. Please ensure that you have GitHub installed on your system before continuing. To install the new version, follow the steps below:

1. *Open a Command Prompt*

2. *Direct/move to a folder you want to save the new matplotlib source code to*

3. *Run the following command:* ***git clone -b d4 https://github.com/CSCD01/matplotlib-team28.git***

4. *After the branch has been copied successfully run the following command:* ***cd matplotlib-team28***

5. *To install the new matplotlib source, run the command:* ***python -mpip install .***

HOW TO USE THE NEW FEATURE

1. ***Create a figure with your desired plots.***

In order to work with the legend of your plots, you must formulate a figure and show the created figure. Open an IDE of your choice that can compile python files and develop a script to display plots of your choice on a figure. Below is a code example that creates and shows the figure that plots a linear and quadratic function.

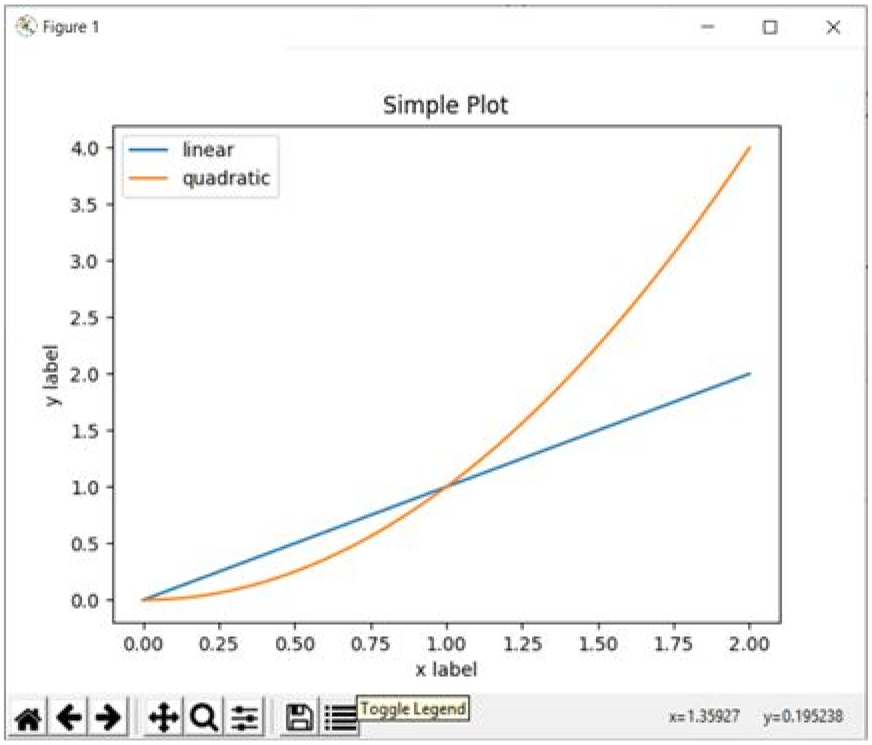
```
import matplotlib.pyplot as plt
import matplotlib as mpl
import numpy as np

x = np.linspace(0, 2, 100)
```

```
plt.plot(x, x, label='linear')
plt.plot(x, x**2, label='quadratic')

plt.xlabel('x label')
plt.ylabel('y label')

plt.title("Simple Plot")

plt.legend()

plt.show()
```

Note: Users must enable the legend to be shown in the figure, the above script does the following by performing “plt.legend()”. If the users decide not to show the legend within their figure at all, they will still be able to click on the “Toggle Legend” button but nothing will happen.

2. Compile and run the script

After constructing your script, compile your script and run it. This will open a new window that represents the figure that displays your desired plots in a graph. The image below represents the figure of the example script created in step 1.

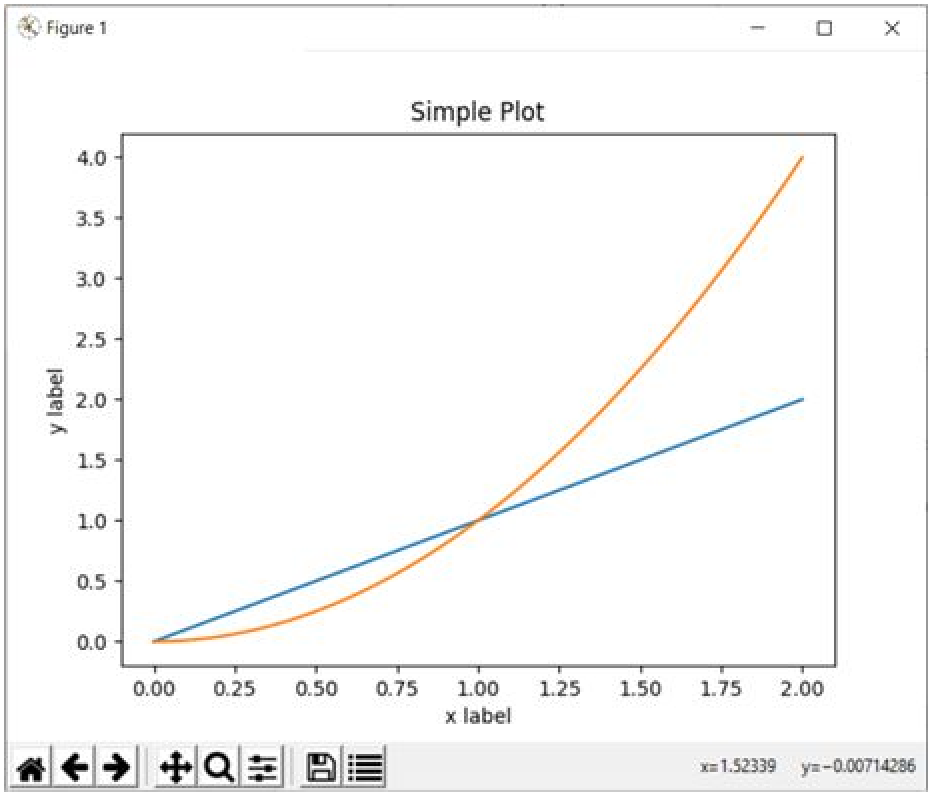


3. Toggling the legend On/Off

On this version of matplotlib, your constructed figure's navigation tool bar contains a new button that allows you to toggle the legend which can be seen at the top left of the plot.

a. Toggling the legend off

Suppose the current state of the legend in your figure is exactly as shown in the image in step 2. To hide the legend simply clicked the Toggle Legend button. Afterwards, the legend should be hidden and look something similar to the following image.



b. Toggling the legend On

Similar to the steps in 3.a, you can simply display the legend again by clicking the Toggle Legend button again

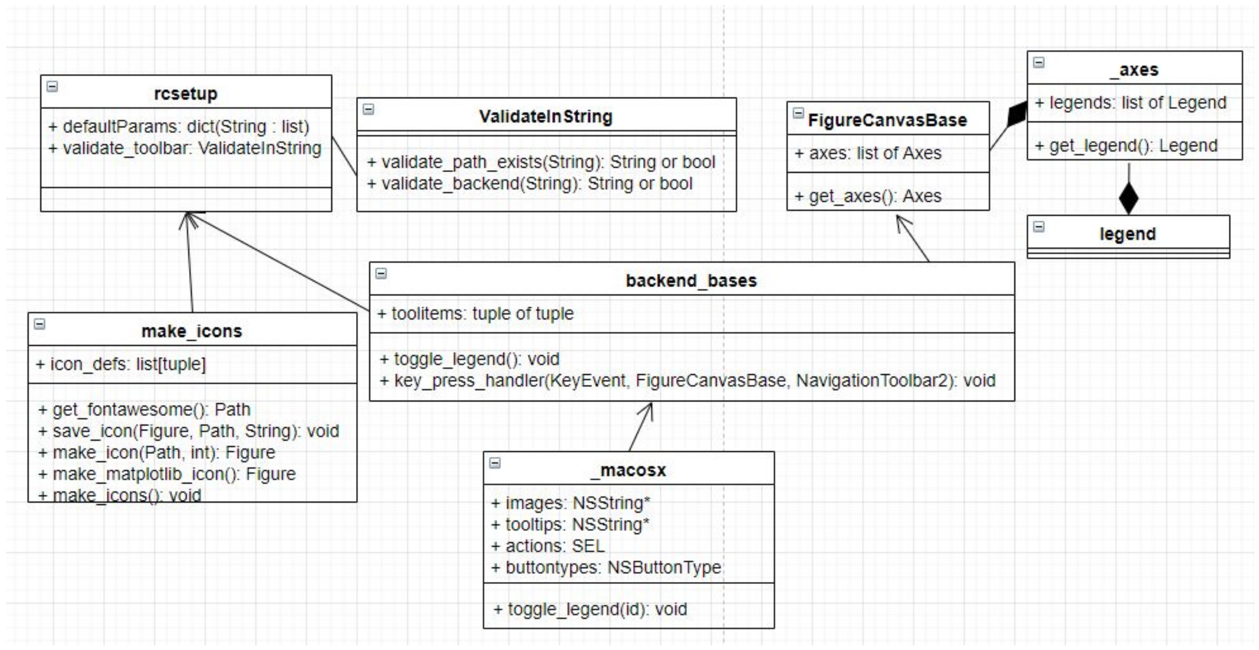
Code Design

Our code can be found in the d4 branch of our fork here.

Changes to the code were made in the following files:

- rcsetup.py
- backend_bases.py
- make_icons.py
- _macosx.m

These files dealt with the backend layer of matplotlib though there was interaction with the artist layer. The links in this section will take you to code snippets we added to implement the feature.


rcsetup
+ defaultParams: dict(String : list)
+ validate_toolbar: ValidateInString
ValidateInString
+ validate_path_exists(String): String or bool
+ validate_backend(String): String or bool
FigureCanvasBase
+ axes: list of Axes
+ get_axes(): Axes
_axes
+ legends: list of Legend
+ get_legend(): Legend
legend
backend_bases
+ toolitems: tuple of tuple
+ toggle_legend(): void
+ key_press_handler(KeyEvent, FigureCanvasBase, NavigationToolbar2): void
make_icons
+ icon_defs: list[tuple]
+ get_fontawesome(): Path
+ save_icon(Figure, Path, String): void
+ make_icon(Path, int): Figure
+ make_matplotlib_icon(): Figure
+ make_icons(): void
_macosx
+ images: NSString*
+ tooltips: NSString*
+ actions: SEL
+ buttontypes: NSButtonType
+ toggle_legend(id): void

UML Diagram of Relevant Components

The file rcsetup.py is used to validate parameters for our new methods as well as provide a keymapping for the toggle_legend method.

The file backend_bases.py is where the toolbar was created and is where the toggling with the legend occurs. There already exists a tuple that holds all of the buttons for the toolbar. To implement our feature, we added the toggle_legend element to the tuple. In addition, to stay consistent with the other buttons on the toolbar, we added a keymapping for toggling the legend. We created a method called toggle_legend(self) that interacts with the FigureCanvas to first grab the axes then grab the legend within the axes. From there, the methods set_visible(bool) and get_visible() are called to toggle the legend.

The make_icons.py file is used to create the icons for the button. Within the file, there is a list of tuples that hold the name of the button initialized in backend_bases.py as well as the unicode for the icon from FontAwesome. This file is responsible for creating all of the image files for all backends. This is the location of the newly added code snippet responsible for creating the image files for toggle_legend().

Since the button we made didn't show up for MacOSX systems with the changes made in backend_bases.py, we needed to edit the file for the MacOSX backend. To accomplish this, we made changes in _macosx.m. In the file, there already exists an array that holds the current buttons in the toolbar. To make room for the new toggle legend button, the size of the arrays were increased by 1 and info for the method call was stored inside the newly created slot. After, we duplicated the method for clicking the pan and zoom buttons to create the method for the toggle legend button. Upon this code change, the feature appears for MacOSX backend users.

There were many differences from our plan in this deliverable compared to deliverable 3. For instance, our initial plan involved us editing all backend files in order to implement the button. In reality, we only needed to edit 4 backend files: backend_bases.py, _macosx.m, rcparams.py, and make_icons.py. We did not realize we would need to edit rcparams.py and make_icons.py initially but after further investigation during our implementation process we realized they were essential to our feature.

For the actual implementation process, we believed that we would need to toggle the legend through manipulation of it's alpha values. However upon further inspection of the legend.py file, we realized that there is a method called set_visible(bool) that will easily toggle it for us. We also believed that the functionality of the button should be in the _axes.py file but realized we should place the functionality inside of backend_bases.py for consistency.

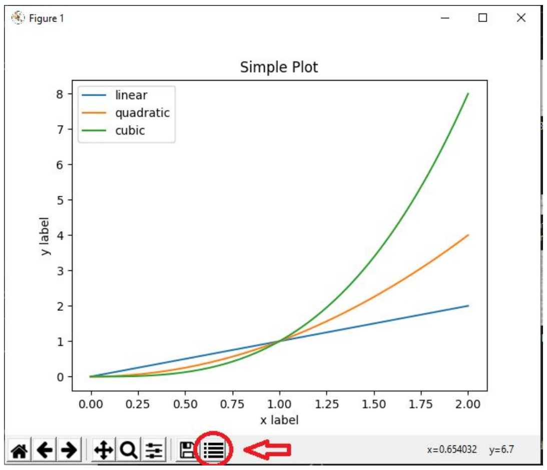
Acceptance Tests

1. Create a random figure/plot. For the sake of this feature, create a figure that plots more than one function. In the following example, the figure contains the plot of a linear, quadratic and cubic function. We will run this file using python3 **filename**.py.

```
import matplotlib.pyplot as plt
import matplotlib as mpl
import numpy as np


x = np.linspace(0, 2, 100)
```

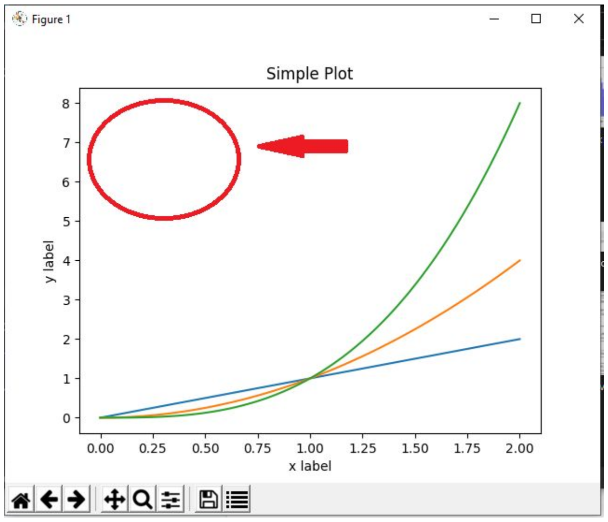
```
plt.plot(x, x, label='linear')
plt.plot(x, x**2, label='quadratic')
plt.plot(x, x**3, label='cubic')

plt.xlabel('x label')
plt.ylabel('y label')

plt.title("Simple Plot")

plt.legend()

plt.show()
```

2. Ensure that there is a toggle button to show/hide the legend of the figure.



3. Ensure that clicking the toggle button for the legend will actually show or hide the legend. Image below shows the legend being hidden after toggling the legend off.

(Alternatively, pressing the key “t” should have the same effect)



4. Confirm that upon clicking the toggle button again for the legend (or pressing the key "t" on your keyboard) that the legend is now visible again.

Unit Tests

Prerequisites:

Pytest 5.3.5
Python 3.6+

We used a similar process to what was done for deliverable 3 to unit tests our new feature. There already existed unit testing files in the Matplotlib repository. These files are separated by what component of Matlplotlib they test (backend, widget, etc.,). We created test cases within the relevant file and emulated what's already been done for widgets/keyboard press tests.

Given that feature had to be developed differently between Mac and PCs, we considered separate tests for the platforms (one targeting Python and one Objective-C). Seeing as there were no Mac specific test tools in the repository, we opted to focus on the Windows unit tests.

The unit tests can be found here.

We have three tests in total:

The first test test_no_legend() checks the behaviour of clicking the toggle legend button when no legend is set for the plot. The expected behaviour is that the legend property of the plot should still be None and no errors should be thrown upon pressing the button.

The second test test_legend_on() checks the behaviour of clicking the toggle legend button when the legend is set to visible for the plot. The expected behaviour is the legend of the plot should become hidden and the visibility of the legend should be False.

The third test test_legend_off() checks the behaviour of clicking the toggle legend button when the legend is set to hidden for the plot. The expected behaviour is the legend of the plot should become visible and the visibility of the legend should be True.

Software Development Process

We used a modified version of Kanban as our software development process. Our Kanban board can be found here: https://trello.com/b/yzFiKfpQ/d01-matplotlib

In total we had five columns: To Do, In Progress, Pending Review, Review, and Done. The "To-Do" column was where we placed the tasks that we may want to complete. The "In Progress" column was where we placed the tasks that we were currently working on. The "Pending Review" column was where we placed the tasks that were finished but want another member of the team to review. The "Review" column was where we placed the tasks that we are currently reviewing. The "Done" column was where we placed the finished and reviewed tasks.

We created separate tasks for investigation of issues, implementation of issues, testing of issues, documentation of issues, and tasks for working on the deliverables.

Over the course of the project, we made a few modifications to our process due to the nature of the project as well as the COVID-19 pandemic. As there weren't always enough issues for everyone to have one, we used pair programming and assigned two members per issue for tasks involving development. The idea of pair programming became more attractive after COVID-19 forced the team into isolation at home. Since we were unable to bounce ideas off of everyone in the team during our in-person team meetings, pairing up allowed us to flexibly meet and work together. For other tasks, each member of the team was able to work on them independently.

The Kanban board helped us stay focused on our tasks and ensured that no single person was overburdened and no team member was not contributing. As Trello sends us email updates whenever something is updated, the Kanban board allows us to receive real-time updates and track all project processes through the board and not through face-to-face communication or Slack. The addition of the "Pending Review" column allowed team members to notify each other without direct communication when an issue was ready to be reviewed.

Using Kanban also helps us integrate project development into our schedules with our other courses seamlessly. We are able to pick up new tasks as we find the time, and easily add new tasks when they come up. If we realized we missed something, we would just add the task to the board. The time saved compared to the planning overhead for sprints in Scrum and Waterfall was significant and allowed us to focus on fixing the bugs and testing rather than planning.

That is not to say we didn't have any meetings as a team. At the first meeting for every deliverable, we made sure we understood the deliverable, broke the deliverable down into simple tasks, distributed the tasks, and started to work on them together if we had time. After that, we had weekly standup meetings to identify bottlenecks, potential improvements, and to summarize what we did since we last met. This allowed us to see where everyone is at and ensured that everyone would be prepared for the TA interviews.

One of the issues we faced was that Trello does not have built-in WIP restrictions which placed the responsibility on us to implement these restrictions. In other words, Trello does not physically limit the amount of cards in a column. Although this did not cause any issues overall when it came to workloads and scope focus, in the next deliverable we need to find a better way to enforce the WIP limits we agreed on. In other words, 4 issues max in “In Progress” for the team and 1 issue max in “In Progress” and 1 issue max “In Review” for each team member. There was no limit to the number of issues in “Pending Review” for each person as no work is being done on those tasks.