TEAM SQUAAD

Deliverable 5

Table of Contents

General

Testing

These are the general testing procedures which we did for all features. If any additional tests/testing were done, they are stated within the relevant section of the report.

Integration Testing

Similar to bug fixes, we tested that our feature was correctly integrated and did not produce any unexpected behaviour with existing features by running the pytest which came with the repository.

Acceptance Testing

For all our features, we decided to conduct acceptance testing by having a teammate who did not work on this feature use the feature. Our main focus points on acceptance testing were:

1) Does the feature name/location make sense?
2) Does this feature do what the user expects it to do (based off the associated documentation)?

Additionally, we listened to feedback of the interviewer during weekly interviews, and made any changes based off the feedback received.

Work Breakdown

In the initial week (week 0), our team spent our time determining features to implement as well as how to split our team to work on the features we selected.

Feature Request #4938

Feature Request #4938 on Github || Python 3

Description

Matplotlib contains a python implementation of the TeX language, called mathtext, that is triggered when a string contains two unescaped $ symbols. When two unescaped $ symbols are detected, the content between the two symbols is passed to mathtext and a rendered version of the text is return and substituted in place of the original text.

Our feature is to add an rcParam “mathtext.enabled” that defaults to True. When set to False, the text and mathtext parsers ignore mathtext blocks in strings and the original input is passed directly to the renderer.

Our change works by escaping all $ symbols in the input string before passing to any parsers. Both the text and mathtext renderers were updated to support this new rcParam

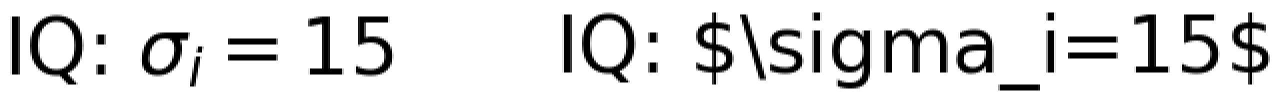
Here’s an example setting the x-axis label:

```
from matplotlib import rcParams
import matplotlib.pyplot as plt

rcParams['mathtext.enabled'] = False
fig, ax = plt.subplots()
ax.set_xlabel(r'IQ: $\sigma_i=15$')
plt.show()
```

Here’s an example making a png render of the equation:

```
from matplotlib import rcParams
import matplotlib.mathtext as mathtext

matplotlib.rcParams['mathtext.enabled'] = False
parser = mathtext.MathTextParser("Bitmap")
parser.to_png('test.png', r'IQ: $\sigma_i=15$')
```

Example Renders:


IQ: $\sigma_i = 15$

`IQ: $\sigma_i=15$`

Work Breakdown

Week 1

18 hours: Researching bugs and testing feasibility of implementing fixes

Week 2

10 hours: Tracing regular text parsing and rendering in various locations (title, axis, text, etc.)
10 hours: Tracing mathtext parsing and rendering in various locations (title, axis, text, etc.)
20 hours: Total

Week 3

3 hours: Implement new rcParam and add checks in parsers to prevent parsing mathtext when present
4 hours: Testing changes to ensure that rcParam prevents parsing in all use cases
6 hours: Researching unit tests using image comparison and debugging issues with image comparison
3 hours: Searching through documentation to find where updates to documentation need to made
16 hours: Total

Feature Request #10610

Feature Request #10610 on Github || for both Python 2 and 3

Description

The new method is named bar_2d_height(). The code works by utilizing the ax.bar() method in a loop to create every bar in the 2d matrix.

bar_2d_height(x, heights, width, colors, *args, **kwargs)

- The x-axis array and the array of heights must have the same length.
- The arrays inside the heights array must all be the same length
- Width can either be a single float, or an array of floats the same size as heights
- colors parameter should only be an array of the same size as heights
- For all of these arrays, both numpy arrays and normal arrays work

The input x is the x-axis locations where the user wants to place their bars and the heights is the 2d matrix of heights for the bars.

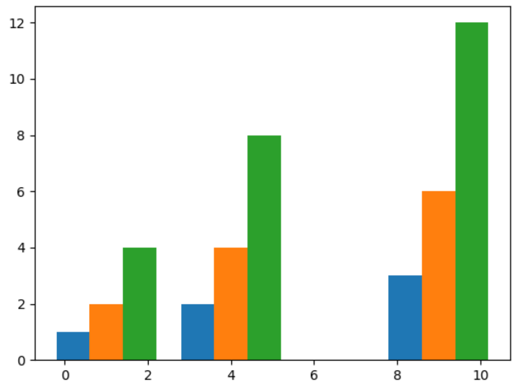
The code starts by doing some checks that ax.bar does not check. If the length of x, heights, colors or width(if it is a list) is not the same, or if heights has less dimensions than a 2d array, it will raise a Value error. The rest of the checks will be done by the ax.bar() method.
If color or width is null, they are set to default values.
An offset is set as the width of the bars, because the offset ensures that the bars created on the same x-axis at the same time will not overlap.
Then the code loops through all of the values in the height 2d array, and creates bars using the ax.bar() feature.

The code is in the same file as ax.bar, which is a part of the _axes.py file, so there is no new class or file that was created for this new method.

Sample Usage

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

x = np.array([1, 4, 9])
fig, ax = plt.subplots()
heights = np.array([[1,2,3],[2,4,6],[4,8,12]])
ax.bar_2d_height(x,heights,align='center')
```

Sample Output



Documentation

In the Matplotlib documentation for matplotlib.pyplot, we would add under “Functions”:

`Axes.bar_2d_height`	Create a bar graph using a 2D array for the height

In the function description page (from clicking the above link), we would have this page:

matplotlib.pyplot.bar_2d_height

```
matplotlib.pyplot.bar_2d_height(self, x, heights, width=None, colors=None,
                                *args, **kwargs)
```

Create a bar graph, passing in a 2d array for the height parameter.

Work Breakdown

Week 1

3 hours: Spent looking over bar to see if it was possible to make it accept a 2-D array for heights and making a graph similar to the one posted in the feature request. Didn’t seem practical as bar was for creating a single bar, so most of the parameters would apply to all the bars, for example same color, and not to mention much more time consuming.
3 hours: Spent looking over axe’s hist function to see how they implemented multibar function.
2 hours: Spent I testing or playing with axe’s hist function to understand how it works and if can make similar code to create a multibar from 2-D heights. In the end, we didn’t bother as histogram and bar purpose are somewhat different but we learned some stuff that would be useful.
2 hours: Spent coming up with a rough idea of how to implement multibar without editing axe’s bar function or something like hist. Ended up with idea to just make another function separate from bar function but uses bar function as might as well reuse functions then to implement another one that does the same thing.
10 hours: Total

Week 2

2 hours: Spent creating a rough idea of the function to implement multiple bar
1 hours: Spent testing out the rough function and fixing up a few bugs or unwanted results
2 hours: Spent attempting to add new parameters that would add new functionality to the multibar function like widths and colors i.e. array of color and array of width so that each bar can have their own properties.
2 hours: Spent testing out the new parameters widths and colors but ended up scrapping the part of the idea as it proved problematic and possibly unwanted. Kept colors though as it can be used as replacement for color, allowing the bars to have user chosen colors.
3 hours: Changing and adding new calculations and change to code such that it would move positions of the group of bar to be centered at the points. Even numbered group of bars would have the point be directly between the 2 middle bars, while odd number of bars will have the point be directly at the center of the middle bar in the group.
10 hours: Total

Week 3

2 hours: Testing changes to the offset, found bugs such as the offset calculation being wrong or off. Had to add new addition to handle array for width which meant another separate case.
2 hours: Figuring out the algorithm, ordering and case to implement array for width to be allowed, and adding the implementation.

2 hours: Spent testing the function to make sure it works properly with all the new changes including handling numpy array or list, or single value and etc. for the parameters, or the calculation for the offset such that the group of bars are centered correctly at the corresponding x-point.
1 hour: Spent refactoring the function, i.e. get rid of some redundancies such as multiple calls to bar when only one is needed, calling multiple create copy of x when only one is needed, etc.
1 hour: Spent creating comments and docstring for the function.
1 hour: Fixing indentation errors.
1 hour: Final testing to make sure it works.
10 hours: Total

Team Feature Idea

Suggested by team || for Python 3 (since Matplotlib only supports Python 3)

Description

Support for specifying a single color as a string to apply to all points in a scatter plot, when a structured data array is passed in.

```
pts = np.array([(1, 5), (2, 2),(3, 3), (4, 4)],
   dtype=[("ones", float), ("twos", float)])

plt.scatter("ones", "twos", data=pts, c=["r"]) #works
plt.scatter("ones", "twos", data=pts, c="r")   #fails
```

After investigating the feature, we decided that it was not a good idea to implement for several reasons:

- Not backwards compatible, users may be expecting the code to crash when an invalid data key is given.
- Easy workaround, simply have to put the string inside a list.
- Implementing the feature would require adding try/catch code to the _preprocess_data decorator. This decorator is used by major axes functions, such scatter, bar, and imshow. Modifying the decorator has the potential to change the expected behaviour of a large portion of the code.

Overall, this feature requires substantial changes and introduces problems that overshadow any potential benefits.

Work Breakdown

4 Hours - Finding the relevant area of code to modify
4 Hours - Finding how the _preprocess_data decorator works
10 Hours - Analyzing possible implementations
18 Hours total

Feature Request #326

Feature Request #326 on Github || for Python 3 (since Matplotlib only supports Python 3)

Description

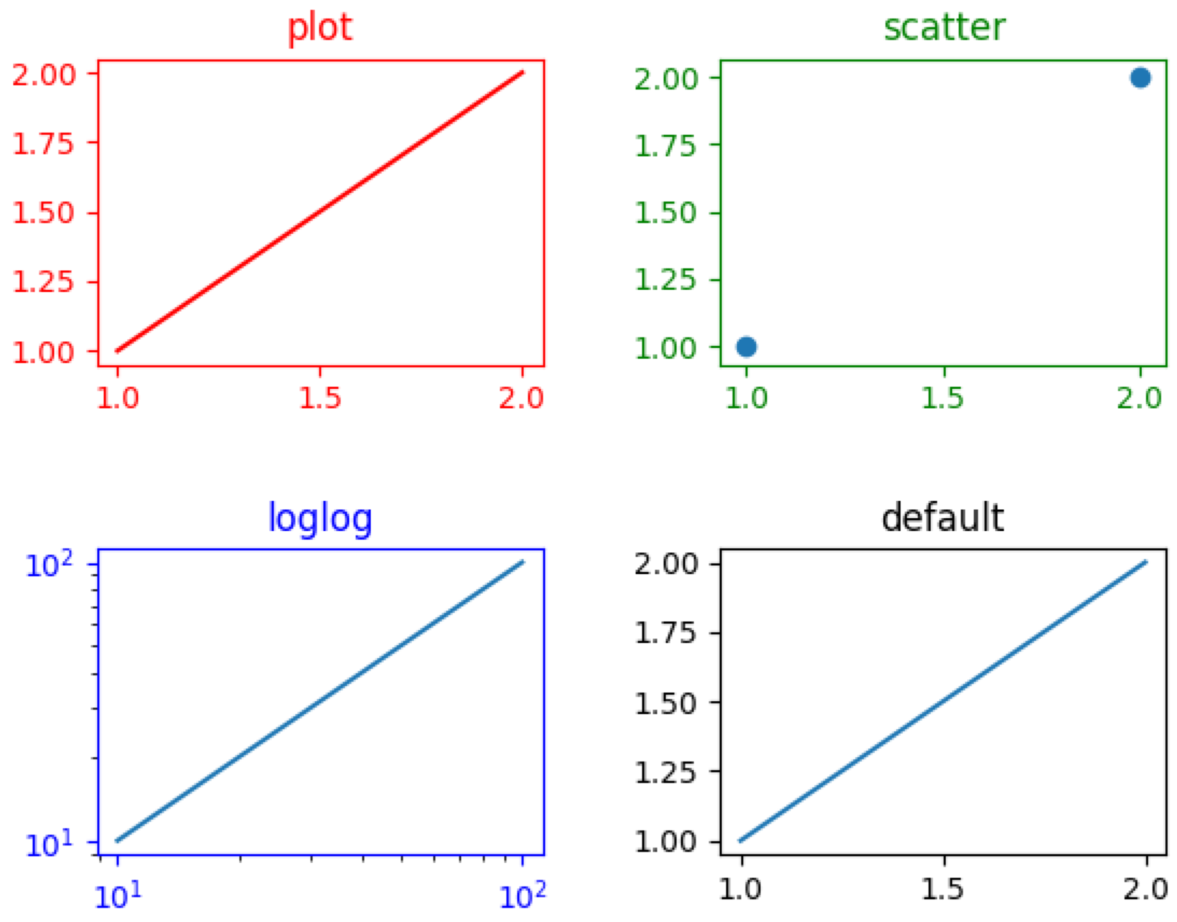
Add functionality to support the color setting of axis, labels, ticks, and title with a single action (unlike currently, where the user needs to manually set the colour of each of these elements individually).

Sample Usage

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

fig = plt.figure()

# structured array
pts = np.array([(1, 1), (2, 2)],
               dtype=[("ones", float), ("twos", float)])


# this creates a plot
plt.subplot(2, 2, 1)
plt.plot("ones", "twos", "r-", data=pts, color='red')
plt.title('plot')
plt.set_axes_color('r')

# this creates a scatterplot
plt.subplot(2, 2, 2)
plt.scatter(pts["ones"], pts["twos"])
plt.title('scatter')
plt.set_axes_color('g')

# this creates a loglog
plt.subplot(2, 2, 3)
plt.loglog((10,100), (10,100))
plt.title('loglog')
plt.set_axes_color('b')
# this creates a default plot
```

```
plt.subplot(2, 2, 4)

plt.plot(pts["ones"], pts["twos"])
plt.title('default')

plt.subplots_adjust(hspace=0.6, wspace=0.4)

plt.show()
```

Sample Output



Implementation Process

For our feature, we had initially tried to implement out feature as a **kwarg passed into the matplotlib.pyplot.plot() function. However, on further investigation of the matplotlib library, we realized that there were many plotting functions (plot, scatter, hist, hist2...etc.) and it would not be good design nor good support to have code within plot() repeated within each of the plotting functions.

Hence, after some thorough investigation of the library, we decided that it would be better to implement the feature as a function which a user can call on a plot - similar to setting the title of a plot, which is also done on multiple different plots.

Testing

Acceptance Testing

In the first week, the user feedback was that it did not make sense to pass the feature into the plotting function as a **kwarg, and hence for week two we worked on refactoring the feature into a function. In the second week, user feedback was positive for both criteria, and hence we deemed that the feature has passed user acceptance testing.

If we create a pull request on Github, we will consider that as the final round of user acceptance testing (with the contributors and owners of Matplotlib). Additionally, we are not too confident on our name of the new feature, and will most likely seek further naming advice if we do create a pull request.

Unit Testing

To unittest our code, we wanted to assure that the feature worked with all different types of plot. We tried our best to go through all different types of graphs and test our feature, but clearly that was not feasible with a time constraint. Hence, we selected 3 types of graphs and tested our feature.

If we had more resources and time, it would have been wise to write a script iterating through all types of graphs and testing the feature and checking that the valid colours were set for the different elements of the feature (axis, title, ticks, etc.).

Documentation

In the Matplotlib documentation for matplotlib.pyplot, we would add under “Functions”:

`Axes.set_axes_color`	Set the color of the axis, ticks, labels, and title.

In the function description page (from clicking the above link), we would have this page:

matplotlib.pyplot.set_axis_color

```
matplotlib.pyplot.set_axes_color(c)
```

Set the color of the axis, ticks, labels, and title of this plot to c, where c is any matplotlib color.

Work Breakdown

Week 1

4 hours: Investigate and trace the colour setting of axes (x and y), ticks, title, and labels
2 hours: Investigating relationship of axes and axis
3 hours: Determine the scope/availability of color setting functions within various files
3 hours: Investigate how **kwargs work and where they are parsed for all plotting functions
3 hours: Investigate the SubplotAxis factory in _subplots.py and how that may affect feature
2 hours: Discussion of putting in feature as a **kwarg and implementation
2 hours: Setting up basic demo and sanity testing as preparation for week 1 interview
19 hours: Total

Week 2

3 hours: Looking through Matplotlib documentation in attempt to find all graphs
3 hours: Looking through Matplotlib repository in attempt to find all graphs
3 hours: Investigating boilerplate.py and how that affects pyplot.py
6 hours: Tracing through all plotting functions in boilerplate.py to check if they use axes or axis
2 hours: Discussion of putting feature as a function, and good location to insert it
3 hours: Refactoring feature from week 1 into a function, writing new unitttest for feature
20 hours: Total