TWENTY 6IXERS

DELIVERABLE 2

Stefan Mitic
Kevin Bato
Chia Anamekwe
Zongda Wang
Byron Leung

TABLE OF CONTENTS

BUG #1

Issue #8236: Legend does not show ‘annotate’
Link: https://github.com/matplotlib/matplotlib/issues/8236

Description: Annotations are not shown on the legend when displayed on a plot. The main issue is that annotation is not a part of the list of artists that is handled by the legend handler. The method Axes.annotate() creates an instance of Text class. Normally, a text annotation does not show up on a legend of a plot since it is just a label. However, adding an arrow property on the annotate function creates a FancyArrowPatch which is not labelled on the legend due to the nature of how it is created.

Estimated Amount of Time: The bug description seems straightforward that the annotation does not show up on the legend. However, the bug is quite complex due to understanding the Legend class and legend handler. Since the annotate function was not intended to have a legend on a plot. The simple fix would have been set_label() on the FancyArrowPatch but this solution does not work. Judging by the complexity of the possible solution, the estimated time for this solution is 10 - 12 hours (5 - 6 days = 2 hours/day).

Possible Solution: Understand and plan how Legend class works will take most of the time since we need to understand how Legend class implements primitive artist objects (Line2D, Rectangle, …) and try to replicate the same effect on annotations. The setter and getter for labels inherited from Artist are primarily used for the legend on a plot. In addition to possibly fixing this issue, annotation must be included in the legend handler. This may require a new handler class for annotation in legend handler.

BUG #2 (FIXED)

Issue #13007: Inconsistent range handling for set_xticks
Link: https://github.com/matplotlib/matplotlib/issues/13007

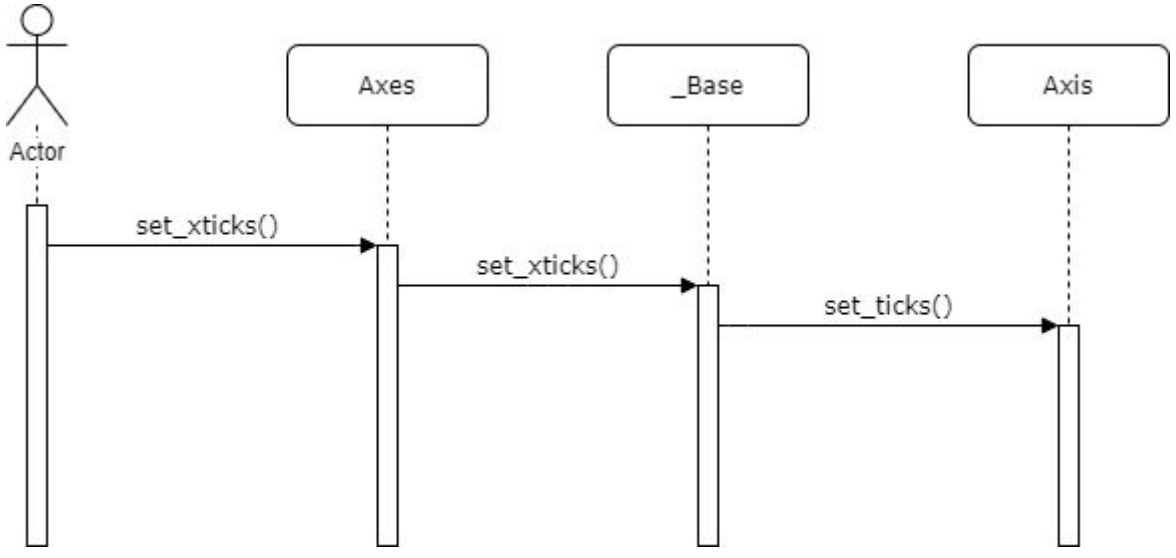
Description: When passing in a range for the method set_xticks() specifically for a list of length 1. The tick is not displayed on the x-axis but with a list of length greater than 1 then set_ticks() will set the ticks on the x-axis as intended.

Estimated Amount of Time: Understanding how set_xticks() work will take some time and seeing any potential risks it has with other classes, if any changes are made. For a bug like this, it will take some time to understand the inner workings of Axes and Axis class. Also, taking into account the risk of altering other classes. The estimated time for this is roughly 4 hours (2 days = 2 hours/day).

Reason for Fix: The bug is not difficult but considering the risk of altering the y-axis as well since both x and y axis share the same behavior when setting tick marks on the plot. This means that set_yticks() will have the same problem of having inconsistent range along the y-axis.

Solution: After tracing through the function call, Axes.set_xticks() is inherited from _Base class. The _Base.set_xticks() then calls Axis.set_ticks() as shown on the sequence diagram. The Axis.set_ticks() function sets the viewing interval of the plot whether it is on the x or y axis.

UML Sequence Diagram



Modified Files:

- lib/matplotlib/axis.py

Technical Commentary: The solution was changing the conditional from 1 to 0 on Axis.set_ticks() such that the function can handle a list length of 1 and retain its normal functionality. We added a set of image comparison tests because this is more suited for testing as it is a graphical bug. The tests consist mainly of changing the length of the tick list and setting one tick amount to have a large or smaller scale.

Unit Test:
Terminal directory matplotlib execute:
$ python3 -m pytest --pyargs matplotlib.tests.test_xticks

Acceptance Test:
When executing the code below, the first two plots from left to right are as intended and the third plot will have a tick mark of 2 on the x-axis.
Use the following code:

```
import matplotlib.pyplot as plt

fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(9, 3))
for ax in [ax1, ax2, ax3]:
    ax.plot([1, 2])

ax2.set_xticks([1, 2])
ax3.set_xticks([2])
plt.show()
```

BUG #3 (FIXED)

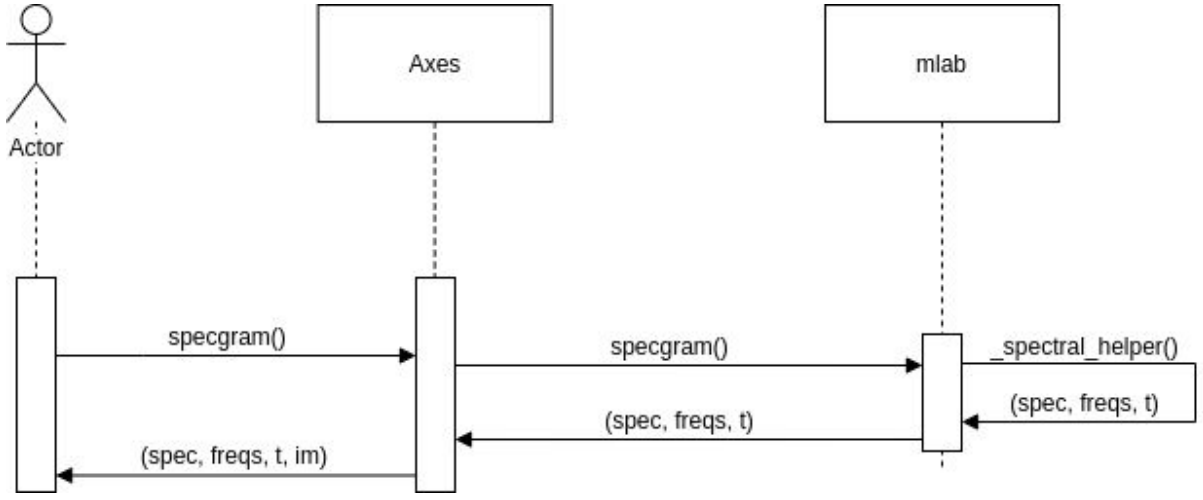
Issue #9100: docstring of parameter Fs of matplotlib.pyplot.specgram() inconsistent with default value
Link: https://github.com/matplotlib/matplotlib/issues/9100

Description: When creating a specgram using pyplot.specgram(), an error is thrown as the parameter Fs is of type None and it is being used in an equation within _axes.py::specgram(). However, the docstring of specgram indicates that the default value of Fs is set to 2 meaning if Fs is not specified during the function call then it should set it to 2 and not throw an error.

Estimated Amount of Time: This bug initially does not seem to big of a change as it requires probably having an if statement to check if the Fs value is not of NoneType. However, from the discussion board presented on the issue page, someone did propose such a solution which was rejected. It will take a bit more time to figure out which methods are being invoked by specgram for Axes. The estimated amount of time spent on this bug is expected to be around 6 hours (3 days = 2 hours/day).

Reason for Fix: This bug does not seem too complex based on the problem as it only needs to check so that Fs is not of type None and prevent the division of Fs as a NoneType. However, the only risk with this bug is that a previous implementation was actually closed because setting Fs to 2 directly was duplicating code. Therefore, finding another approach to the problem will be more difficult and internally testing specgram without image comparisons might require some time to comprehend the implementation.

UML Sequence Diagram:



Possible Solution: Based on some of the feedback gotten from a previous attempt at implementing a bug fix which was adding an if condition which checks if Fs is of type None and sets it to 2 if it is. This solution works, however, we should not be setting Fs to 2 within _axes.py. Instead, use the return values from the function call to mlab.py which within that invokes a helper function that sets Fs to 2. So the solution is to set Fs to the corresponding frequency which was returned after the function call to mlab.py.

Modified files:

- lib/matplotlib/axes/_axes.py
- lib/matplotlib/tests/test_axes.py

Technical Commentary: For _axes.py, a condition was added to check if Fs is of type None. If it is, then I set Fs to be the value of the last frequency returned by mlab.specgram() as mlab.__spectral__helper() sets Fs to 2 if its value is of type None which follows with the documentation. So instead of repeating code, I used the output from calling mlab functions and setting the value correctly to avoid any errors. I added a test in test_axes.py called test_specgram_fs_non which checks for an exception while passing Fs to be None and checks if xmin and xmax within specgram are set accordingly as those values are calculated from Fs.

Unit Test:
Terminal directory matplotlib execute:
$ python3 -m pytest -s --pyargs matplotlib.tests.test_axes::test_specgram_fs_none

Acceptance Test:
Initialize a specgram from pyplot of matplotlib without setting Fs. When executed, there should not be any error and the default value should be 2.
Use the following code:

```
import matplotlib.pyplot as plt
import numpy as np
plt.specgram(np.ones(300))
plt.show()
```

BUG #4 (FIXED)

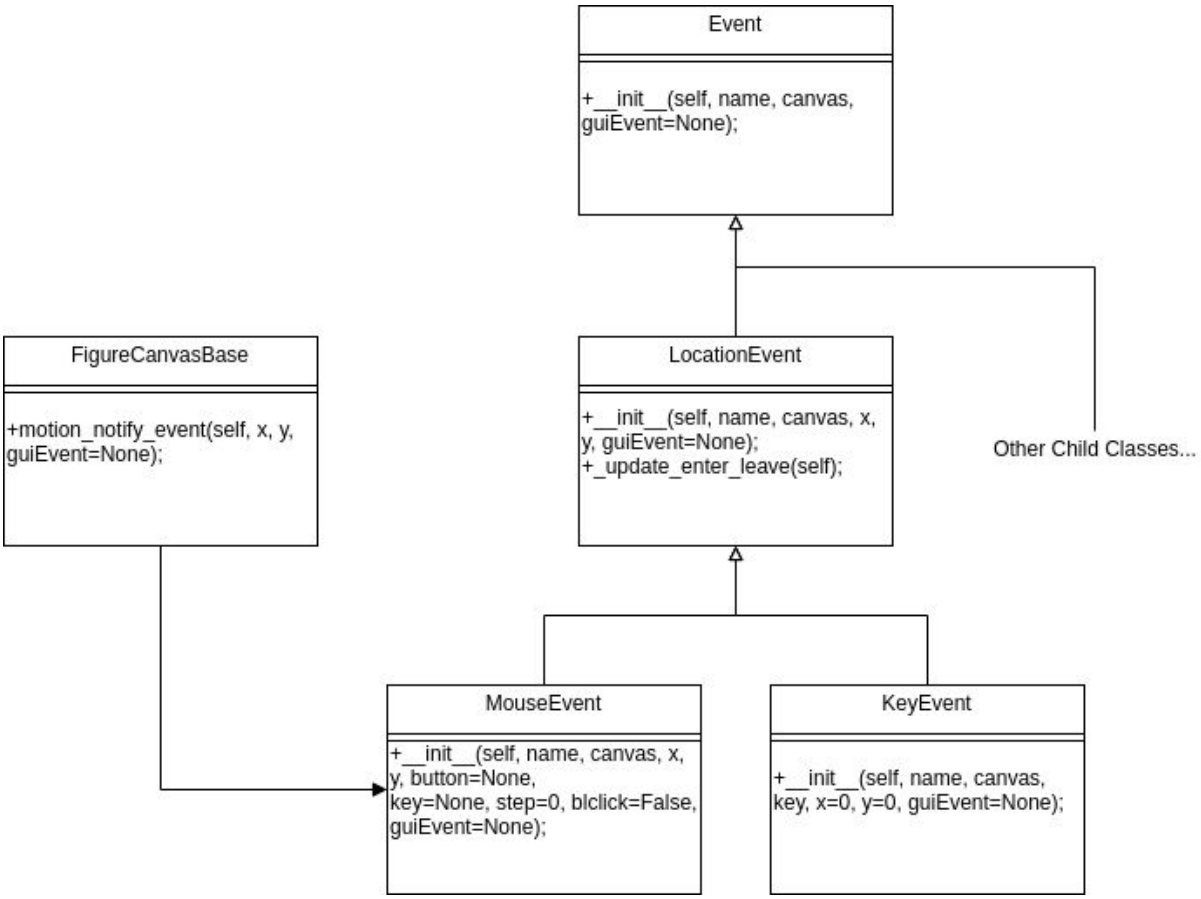
Issue #15139: All subclasses of LocationEvent could be used in cbook.callbacks before being fully initialized
Link: https://github.com/matplotlib/matplotlib/issues/15139

Description: When using callback for a MouseEvent in FigureCanvasBase, an event object can be passed to callbacks.process for axes_enter_event in LocationEvent before it has been fully initialized. This results in an error for MouseEvent as the object has no attribute button.

Estimated Amount of Time: The bug will take a lot more time as it requires research in order to figure out the best approach to solving the problem. The bug focuses on the inheritance of the object types for the two types of LocationEvents: MouseEvent and KeyEvent. The problem occurs when some callback process is invoked with MouseEvent rather than LocationEvent and MouseEvent does not finish initializing a MouseButton. As this bug seems to be complex it will take around 8 hours (4 days = 2 hours/day).

Reason for Fix: This bug requires a lot more information on the inheritance and structure of FigrureCanvasBase and its relationships with LocationEvent. The reason I chose this issue was there was a well documented report of it on github and the user provided a great way to replicate the problem. The anticipated risk is not having completed the fix as it does involve a lot of integrate parts between the different types of objects as well as callback processes.

UML Class Diagram:



Possible Solution: In the MouseEvent object, before initializing LocationEvent which calls a callback.process of axes_enter_event, first initialize the MouseButton for MouseEvent and then initialize LocationEvent. This way the MouseEvent can have a defined MouseButton before the callback.process is invoked. As for the concern that axes_enter_event is not called with LocationEvent, it can be argued that MouseEvent is a LocationEvent as it is its subclass.

Modified Files:

- lib/matplotlib/backend_bases.py

Technical Commentary: After trying to implement a solution that refactors the inheritance of properties for python's 'self' use case, this solution would require a lot of refactoring and was not seen as the best approach. Instead I considered the MouseEvent to be a LocationEvent because of the inheritance of properties. By moving the initialization of LocationEvent after MouseButton was set in MouseEvent object, it avoided any error with callbacks.

Acceptance Test:
To test the solution, hover over the graph with your cursor. The callback function will be invoked printing to standard output indicating a MouseEvent followed with True for button and printing motify_notify_event. It should not indicate an error.
Using the following code:

```
import matplotlib.pyplot as plt
def mycallback (event):
 print (type(event), hasattr (event, 'button'))
 print (event)

plt.gca().figure.canvas.mpl_connect ('axes_enter_event', mycallback)
plt.gca().figure.canvas.mpl_connect ('key_press_event', mycallback)

plt.show()
```

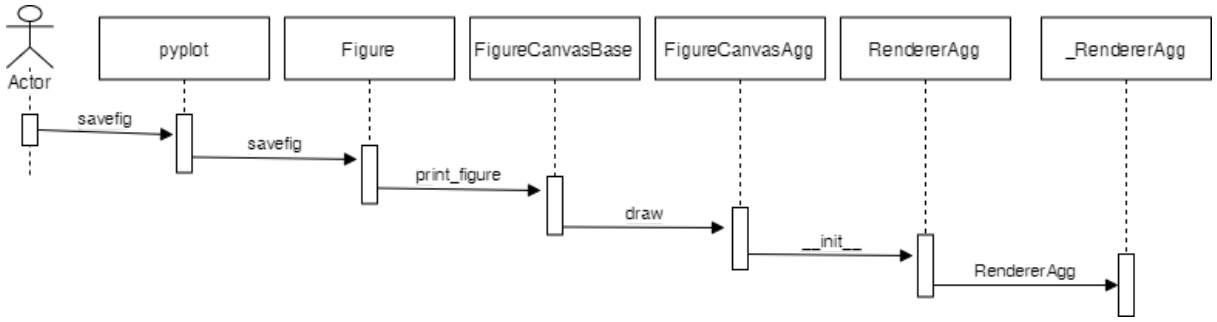
BUG #5

Issue #14842: Segment fault when set dpi too high
Link: https://github.com/matplotlib/matplotlib/issues/14842

Description: When the dpi for a figure is set too high, then python will crash with a segmentation fault error.

Estimated Amount of Time: Since the bug replication occurs in the pyplot scripting layer, there are many layers and code that could have caused this bug. Hence, there is a high degree of uncertainty in regards to the amount of time that it may take to fix this error. The issue posted on github does not include a lot of detail and does not help with pinpointing where the bug may have occurred, so a thorough investigation of each layer of the function call that leads to the error may be required. The estimated time for this fix is around 4 - 6 hours (2 - 3 days = 2 hours/day).

UML Sequence Diagram:



Possible solution: After investigating the contents of each function that leads to the segmentation fault, we eventually determined that the error does not occur within the python code of matplotlib and is instead caused by a limitation in the AGG C++ backend that matplotlib uses to render image files. When matplotlib attempts to save a figure as an image, the backend will attempt to create the image using the AGG library. To do so, matplotlib must create a “pixel buffer” that AGG will use to store an image in its entirety in pixel format. When the width * height * dpi^2 * 4 of a figure is greater 2^{32} - 1, or the maximum unsigned integer that can be stored in C++, then matplotlib will attempt to create a buffer with a length greater than what is allowed in C++ and a segmentation fault will occur. Since it is infeasible to remove this limitation from the AGG library, a possible solution would be to output an error or warning notifying the user of this limitation when he or she tries to save a figure using the AGG backend with width, height, and dpi values that are too large.

Acceptance Tests:
If the bug is fixed, the following code should not crash and should instead output an appropriate error or warning.

```
import matplotlib.pyplot as plt
import matplotlib as mpl
mpl.use('Agg')

if __name__ == "__main__":
   plt.figure(figsize=[12, 9], dpi=5000)
   plt.plot(range(0,4), range(0,4))
   plt.savefig('a.png')
   plt.clf()
   plt.close()
```

BUG #6

Issue #14233 [Feature Request]: Allow setting default AutoMinorLocator
Link: https://github.com/matplotlib/matplotlib/issues/14233

Description: Currently, when displaying minor ticks on plot, AutoMinorLocator defaults to interval of 4 or 5, and no way to explicitly set interval. Added feature would be to allow interval to be specified.

Estimated Amount of Time: This is a relatively simple feature and appears quite easy to implement. The AutoMinorLocator class is easy to understand, but the issue may be in understanding the interactions with other classes, and what other elements of the project the change would affect. Allowing time to understand all potential affected classes, the estimated time for this feature would be around 3-5 hours (1 - 2 days = 3 hours/day).

Possible Solution: The straightforward solution would be to add a default argument set to None. If a value is specified, this value will be used to set the interval, and if it is not, the function will follow the same logic as before and default to 4 or 5. Most of the time will be spent understanding how values other than 4 or 5 affect the program.

BUG #7

Issue #14527: log scale messed up in histograms when sharing axes

Link: https://github.com/matplotlib/matplotlib/issues/14527

Description:
When constructing a histogram graph, setting y scale to log when having multiple data sets and if the last data set is empty, the graph constructed will have a negative log of y-axis which isn't correct, as it should be positive integers as counts. The ticker class should be responsible for this bug

Estimated Amount of Time:
Fixing the bug requires a good amount of understanding on how LogLocator works, especially how it interacts with datasets, as we know the problem occurs when the last data set is empty, we need to know how the ticks are selected considering given data. Following both trails of data and the details of "application" of the log scale. Estimated time will be 6 hours total, 1-2 days (3 hours /day)

Possible Solution: tick_values are generated by function Loglocator.tick_values, however the key problem may not be here, as the bug does not occur in any log scale graphs, it only happens when handling multiple data sets and when the last one is empty. It's most likely an error during iteration when attempting handling multiple dataset and forgetting the edge case of the last one could be empty. Solution could go from a simple edge case consideration to a redesign of abstraction of data in multiple dataset being refactored to be better applied by loglocator and generation of vmin and vmax.

SOFTWARE PROCESS

Our software process for this deliverable is a modified version of the V-shaped model that we outlined in the previous deliverable. For each issue, we went through four phases including requirements specification, design, implementation, and verification.

Requirements

The requirements specification involves specifying the issue from matplotlib's GitHub repository which gives us an outline and hopefully a clear understanding of the issue that needs to be addressed. If the GitHub issue description is not sufficient, we investigated the issue ourselves to ensure that we have a clear understanding of the specific problem that needs to be solved. An acceptance test of either our own design or taken from the GitHub issue description itself should be specified so it can be used in the future to definitively verify that we have satisfied the requirements.

Design

Then, in the design phase, we investigated matplotlib's source code to determine the relationships and method calls between the possible relevant classes in an attempt to gain an understanding of what the issue may be caused by and how it may be solved. We used UML class or sequence diagrams to concretely lay out and visualize these relationships for our own use. At this point we would have sufficient information and understanding of the bug to be able to conclude the design phase with a description of a possible implementation of a solution to the issue.

Implementation

The implementation phase is simply the phase in which we attempt to implement the possible solution detailed in the design phase. Thanks to the previous two phases there is a lot less trial and error involved in this phase and it was easier to come up with an alternative solution even when the previously planned solution does not go exactly as planned. We committed these changes to a feature branch of our fork of the matplotlib repo.

Verification

Finally, in the verification phase we started with verifying that our implementation passes the unit tests and acceptance tests created in the each phase, and then we verified that our implemented solution follows the documentation outlined by matplotlib's guide for contributing developers. Before merging the solution to the master branch of our repo we assigned members of the group as code reviewers. Once their code had been approved, it would be squashed and merged into the master branch.