University of Toronto – Scarborough
CSCD01 – Engineering Large Software Systems

Deliverable 03 – Report

Prepared by ***GoonSquad*** Team

Swarnajyoti Datta
Nikki L. Quibin
Junil Patel
Beiyang Liu
Laine London
Hajoon Choi
Leo Li

Date: February 16, 2018

Table of Contents

Bug #1: Legend Annotate

Bug/Issue: Legend does not show ‘annotate’ #8236

Code Reproduction for Bug: /bug_snippets/legend_annotate.py

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

# create a range of numbers as the x axis and the function for the y axis
x = np.arange(0.0, 15.0, 0.01)
y = np.sin(0.3*np.pi*x)

# create a new figure with only one graph
fig = plt.figure()
ax = fig.add_subplot(111)

# plot the function
ax.plot(x, y, lw=2, label="sin(x)")

# create annotations to indicate a wave's length and amplitude
# notice the label param to indicate name in legend
ax.annotate("",
            xy=(1.6, 1.03),
            xytext=(8.4, 1.03),
            arrowprops={'arrowstyle':'<->'},
            label="wavelength")
ax.annotate("",
            xy=(8.4, 0),
            xytext=(8.35, 1.0),
            arrowprops={'arrowstyle':'<->', 'ls':'dashed'},
            label="amplitude")

# set the bounds and show the legend
ax.set_ylim(-1.5, 1.5)
ax.legend()

plt.grid()
plt.show()
```

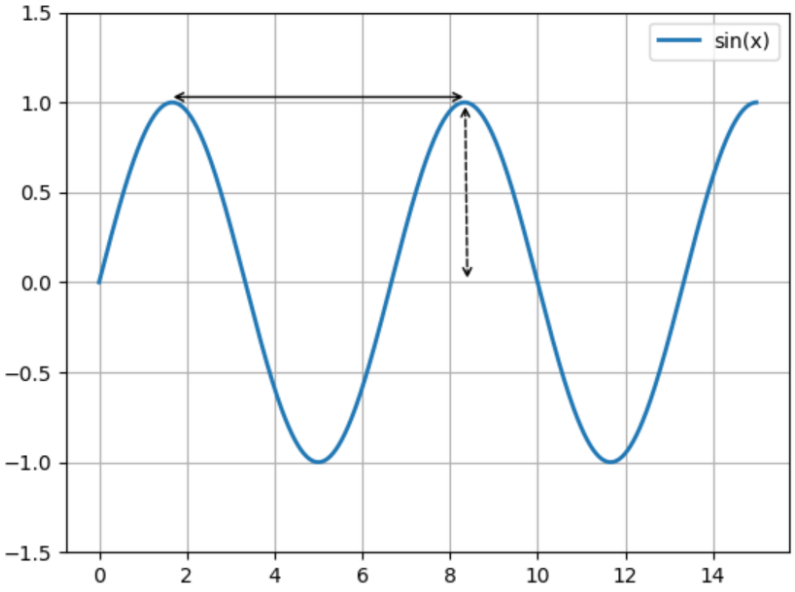

Figure 1: Code Snippet and Bug Outcome

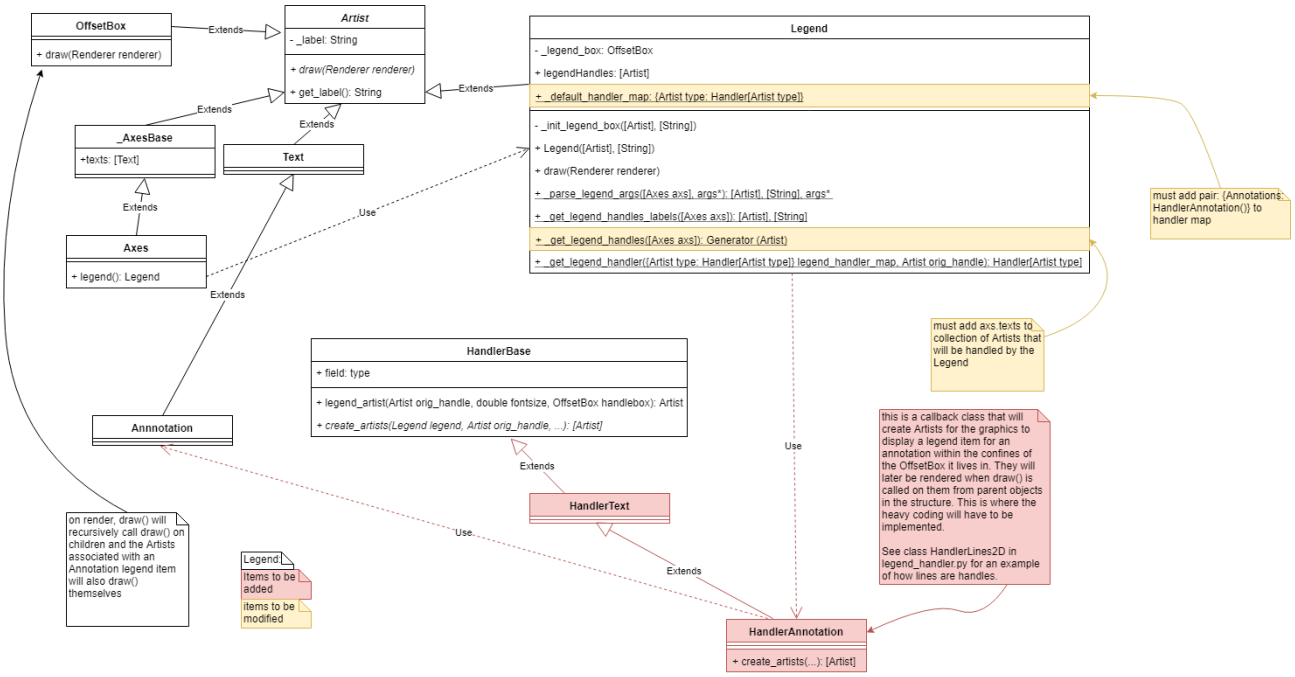

Figure 2: Legend Annotate Bug Class Diagram

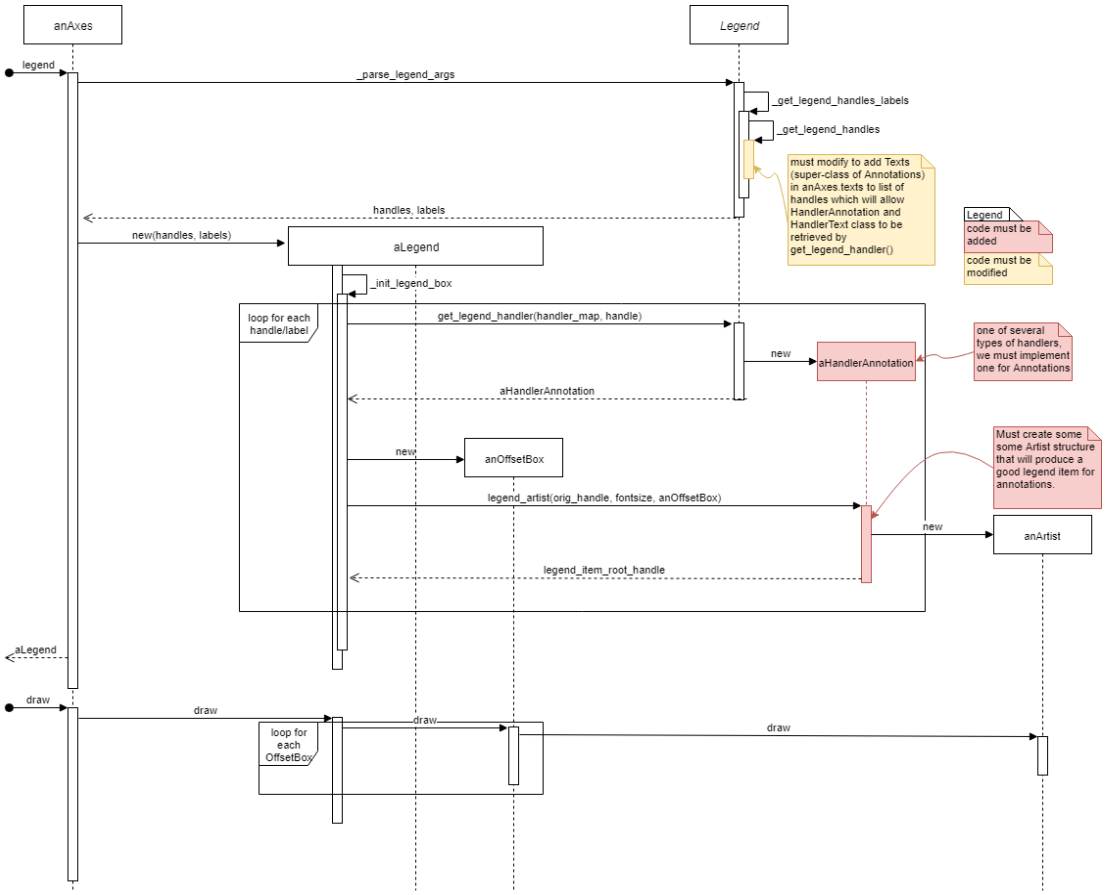

Figure 3: Legend Annotate Bug Sequence Diagram

Description:

Legend items for annotations are currently not operational despite inputs currently being legal for them. Boxing the bi-directional arrows above into a legend to denote amplitude and wavelength is an example of a desirable use case.

When the legend() method on an Axes object gets called, the program eventually adds items to a list of handles to be inserted into the Legend for that Axes (see legend.py:1308, 1313). Annotations, stored in the Axes field texts (as Annotations are a sub-class of the Text class), are not currently added to this list. Additionally, the handler to construct the legend items for Annotations and Texts do not currently exist in the file legend_handler.py (and are subsequently not mapped in legend.py:805).

Approach to Solution:

While we know the exact missing components, fixing this issue will require a sizable amount of effort. The bulk of the job will be in coding the handler to create an adequate construction of legend items for Annotations. There is very little documentation concerning legend items, so we will be required to reverse-engineer the handlers for the other objects like lines. Additionally, this is an issue that will require some decision-making as to how the design of the final legend item should look like.

The exact work necessary is indicated in the UML's. Yellow portions will require modifications to accommodate the addition of an extra type of legend item. Red portions will require creation from scratch. Note that there is no difference to how the program currently operates when it comes to using other items in legends (the handlers in red are already implemented in those cases), so they were cut out for brevity and to focus attention on what needs to be implemented to get Annotations functional with legends. The sequence diagram consists of two separate operations: commanding the Axes to generate a legend and later actually rendering the result via draw(). Both are called from higher-level operations or via direct advanced users. The legend() command in the given scenario assumes no arguments (in which case it legends every available item to the Axes of interest).

The good news is that this is a mostly self-contained issue. Any further bugs or errors because of attempting to work on this issue are likely to be local to the chain of operations regarding the legend.

The estimated hours needed to fully fix the bug is around 13-18 hours including creating tests or validating that the code conforms to the standards set up by the matplotlib team.

The steps needed to fix the issue are as follows:

1) Study the way legend items are constructed in the other handlers and learn the techniques required to position and create the necessary components.
2) Code the handler, add it to the default handler map in legend.py (line 805), and allow Annotations to be returned by the generator Legend._get_legend_handles().
3) In addition to standard suggested testing, test that legend items for other types of components still work and display properly even when mixed in with annotations.
4) Re-evaluate the design decisions made for the structure of the new legend items.

Bug #2: Bbox Tight Legend

Bug/Issue: Legend is not present in the generated image if I use “tight” for bbox_inches #10194

Code Reproduction for Bug: /bug_snippets/bbox_tight_missing_legend.py

```
1  import matplotlib.pyplot as plt
2  import numpy as np
3  import matplotlib as mpl
4
5  x = np.arange(-10, 10, 0.5)
6  y1 = np.tan(x)
7  y2 = np.sin(x)
8
9  fig, ax1 = plt.subplots(ncols=1, nrows=1, figsize=(10, 6))
10
11 ax1.plot(x, y1, label='tan')
12 ax1.plot(x, y2, label='sin')
13
14 handles, labels = ax1.get_legend_handles_labels()
15 # following code does not work
16 legend = fig.legend(handles, labels, bbox_to_anchor=(1, 1))
17 plt.savefig('test.png', bbox_extra_artists=[legend], bbox_inches='tight')
18 plt.show()
19
```

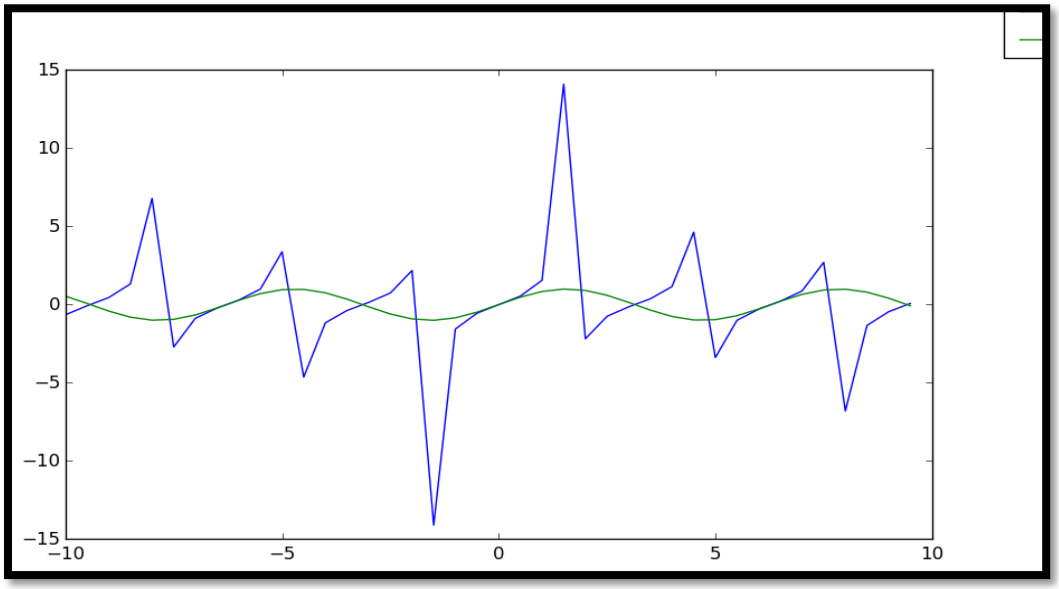

Figure 4: Code Snippet and Bug Outcome

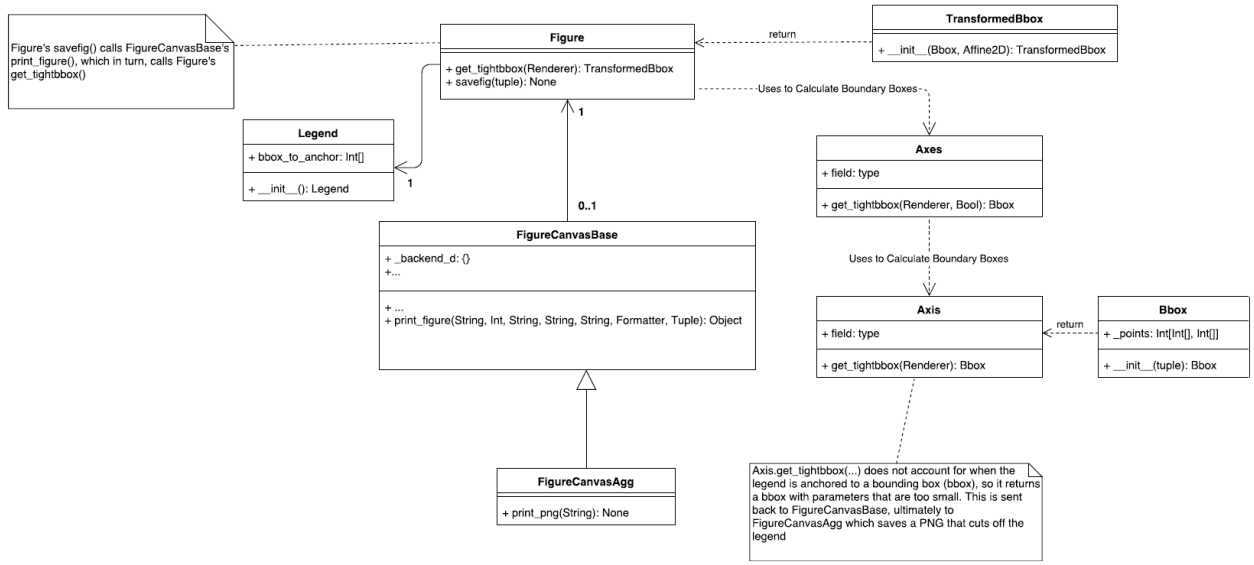

Figure 5: Bbox Tight Legend Bug Class Diagram

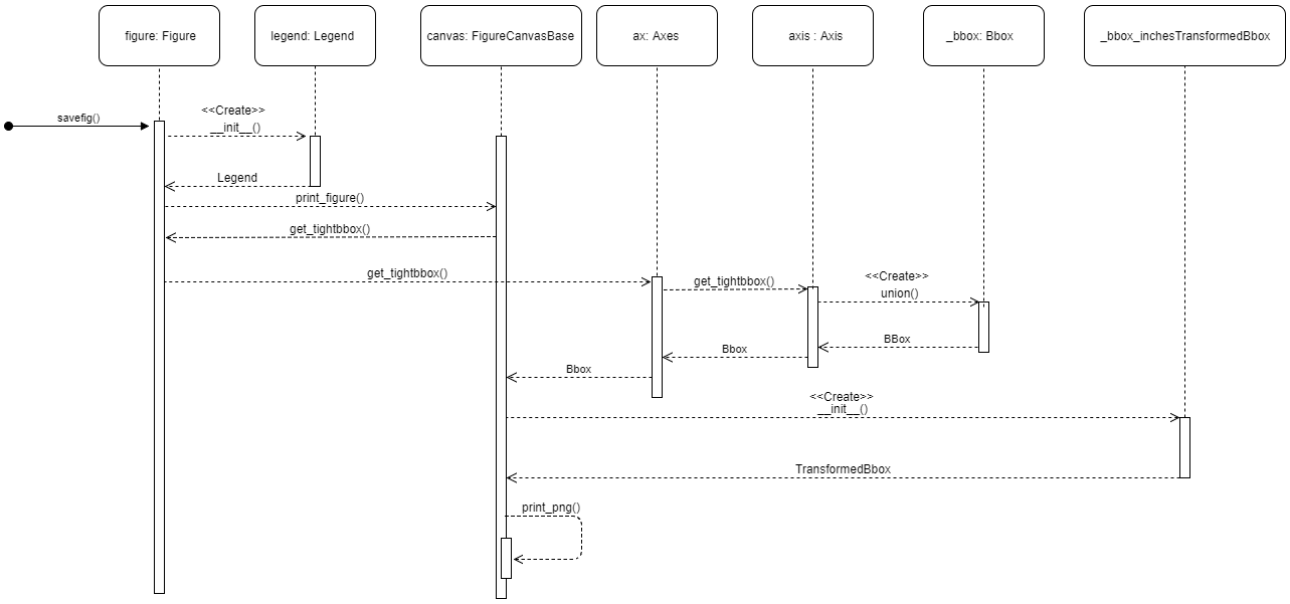

Figure 6: Bbox Tight Legend Bug Sequence Diagram

Description:

As shown in figure 4, when bbox_anchor is used along with the “tight” property of bbox_inches, the created legend gets cut off in the outputted file. The “tight” property enables users to reduce the size of the whitespace in the outputted figure but in the process, the figure’s legend is omitted. This results in a figure that lacks important information. This rendering is also inconsistent with using the loc property, wherein the legend is not omitted in the outputted file, and a detailed figure is produced.

Approach to Solution:

Coming up with an elegant solution will require a little bit of effort, as we will be required to look over several files within the code base. We will first need to figure out how the bbox module works with the legend module, and then figure out a clever way to attach the legend to the plot. Right now there is a disconnect between bbox and legend, so to determine this connection and to fix the bug it will require about 8-10 hours of work, with testing and validating taking another 3-4 hours.

The steps needed to fix the issue are as follows:

1) Understand the connection between legend, axis, bbox
2) Find way to detect legend properties from axes, axis without changing drastic changes such as adding new arguments to existing methods
3) Account for legend size and location within bbox calculations
4) Ensure that changes do not affect larger codebase

Relevant files are:

backend_bases.py

figure.py

transforms.py

tight_bbox.py

legend.py

The files that will be affected are those listed above. Everything else should be unaffected due to bbox and legend being isolated components of a figure. Regardless, the only code affected will be those that handle the creation of a legend and alignment of the legend on the outputted file.

Bug #3: Logscale

Bug/Issue: Minor ticks on log-scale colorbar are not cleared #8358

Code Reproduction for Bug: /bug_snippets/log_scale.py

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  from matplotlib.colors import LogNorm
4
5  data = np.array([[0.1, 0.3],
6                   [0.6, 1]])
7
8  plt.rcParams['ytick.minor.visible'] = True
9
10 plt.pcolormesh(data, norm=LogNorm())
11 plt.colorbar()
```

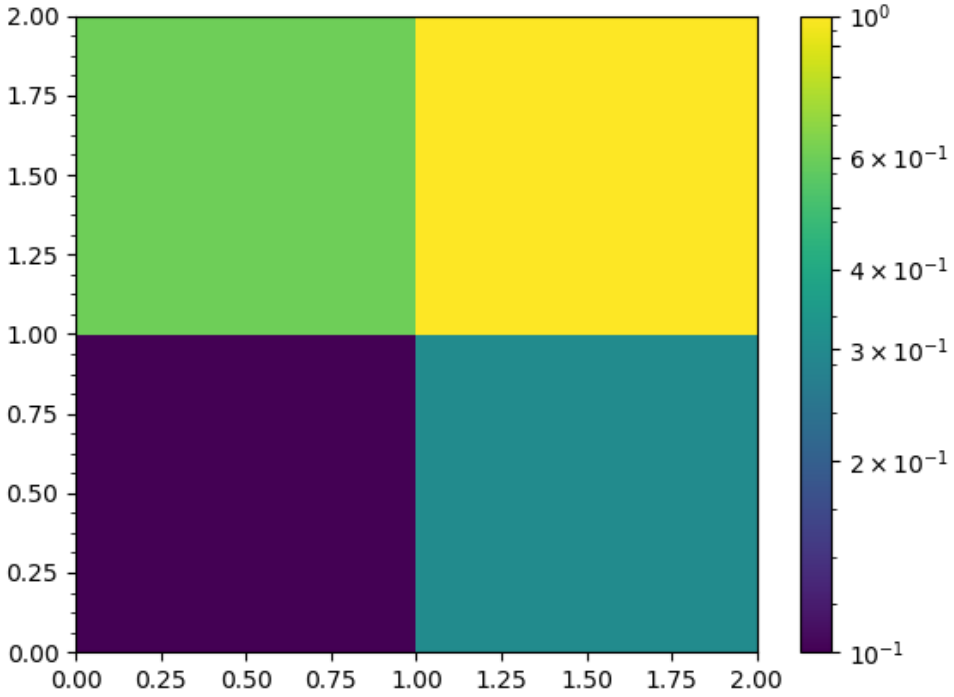

Figure 7: Code Snippet and Bug Outcome

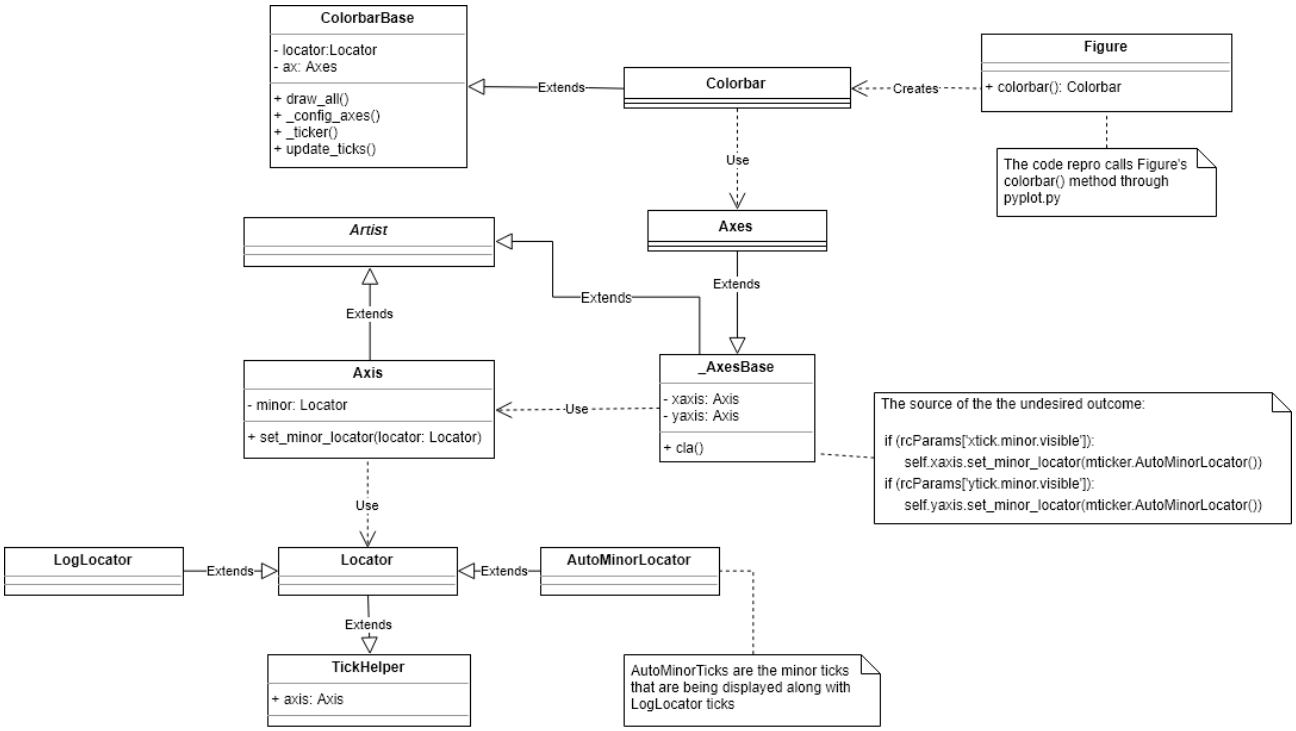

Figure 9: Logscale Bug Class Diagram

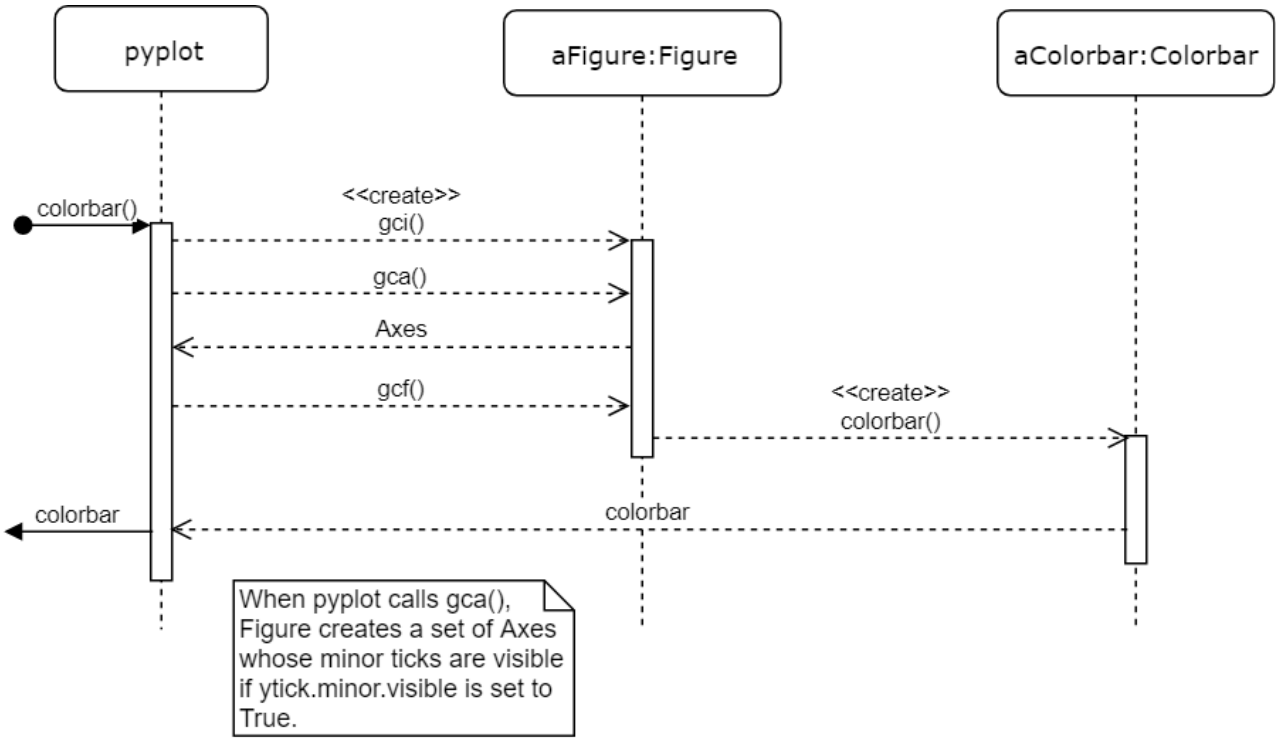

Figure 9: Logscale Bug Sequence Diagram

Description:

If the minor ticks on the colorbar are enabled while plotting a quadrilateral mesh using pcolor or pcolormesh with a logarithmic colorbar scale, some undesired linear scale minor ticks will be displayed alongside the desired logarithmic scale ticks.

In more detail, when the Axes object is created, it calls cla() to clear the current axes. However, when the xtick.minor.visible and/or ytick.minor.visible dictionary entries in rcParams are set to true, that method updates the minor locator to AutoMinorLocator, which is an undesired result.

Approach to Solution:

Given that this is a very specific issue caused by enabling the minor ticks and using a logarithmic scale on a colorbar, it will be a very difficult task to fix this bug without creating more issues in the process. Although the source of the bug begins in axes/_base.py (lines 1050-1054), we believe that changing any line of code in cla() may break other code that depend on it. So we will have to devote a lot of time to examining the consequences of potential fixes. Due to the nature of the bug, we estimate that it will take around 12 hours to fix, and another 4 hours dedicated to testing and validation and ensuring code consistency with matplotlib.

The steps needed to fix the issue are as follows:

1) Take a more in-depth look at the relevant methods and try to understand the design decisions behind the code responsible for the bug.
2) Determine ways to prevent AutoMinorLocator ticks from appearing on logarithmic scales.
3) Study the effects of these methods and how they interact with other parts of the code.
4) Decide which solution creates the least impact outside of fixing the bug and is the most consistent with matplotlib standards and implement it.