TEAM SQUAAD

Deliverable 3

Table of Contents

Bug Report

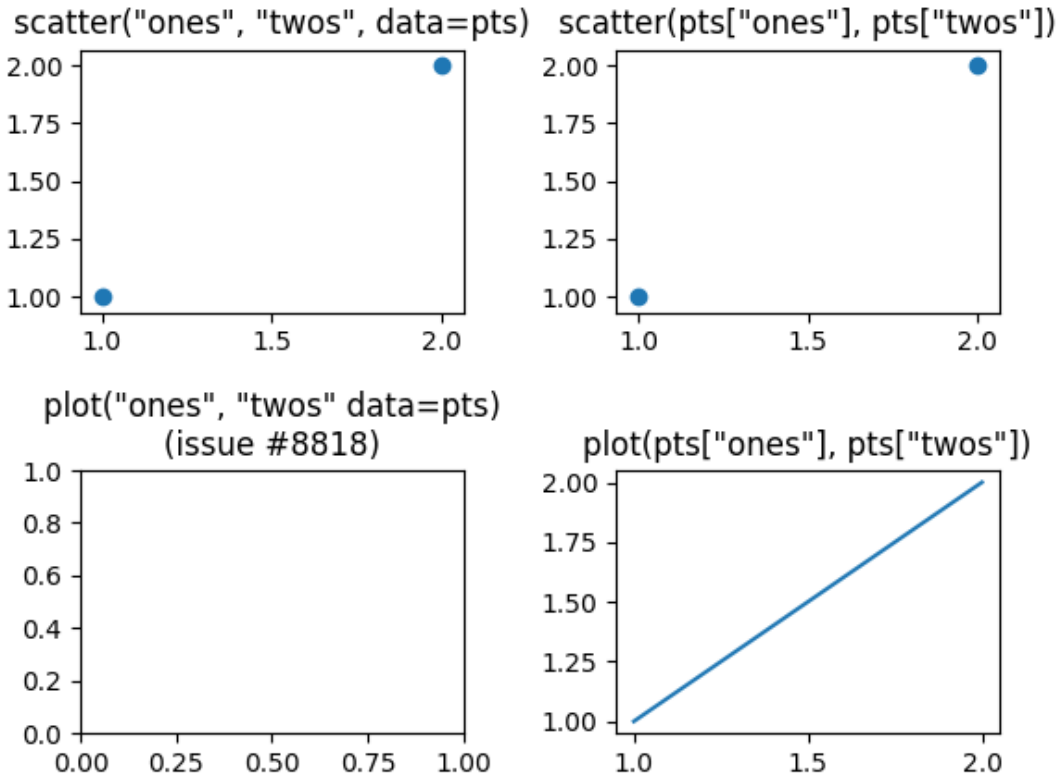
Issue #8818

Issue #8818 on GitHub || Occurs in Python 2 and 3

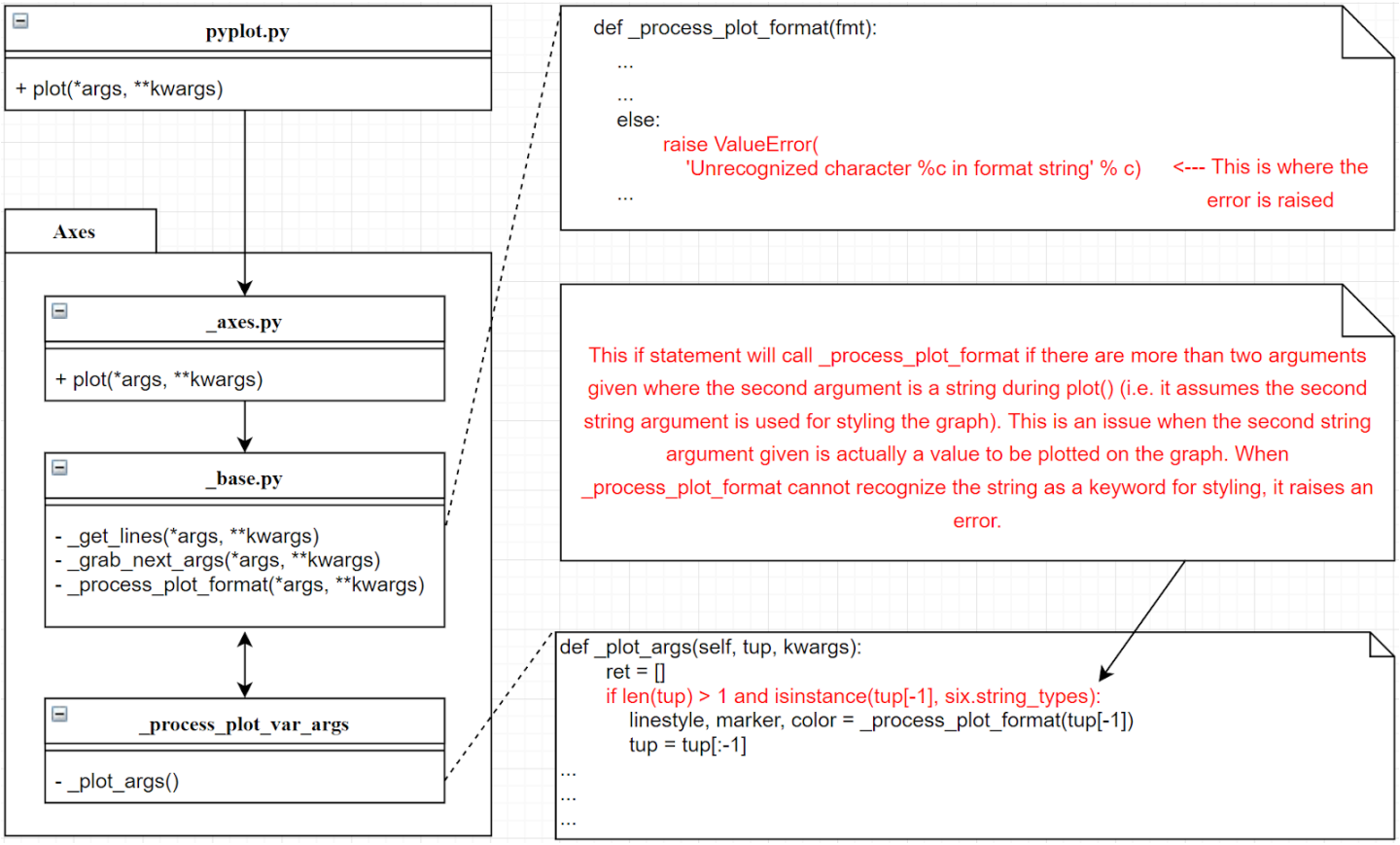
When making a structured array (e.g. one using Numpy with names/labels for the data) and plotting the data within the structured array by passing in the names and structured array using pyplot's plot function (*pyplot.plot*), the plot function will incorrectly interpret the second name argument as a colour specification argument, and throw a ValueError from trying to parse it.

The expected behaviour can be produced by passing in an indexed structured array instead of the names and array, however, this is different than pyplot's scatter plot function, which supports both methods of passing in and plotting structured array data.

The behaviour of scatter and plot with structured data as described above is plotted below in matplotlib (code can be found in on the team GitHub repository):



UML



Analysis

The scope of affected code to fix is fairly small (as mentioned above, this bug is mostly resultant from the *plot()* function treating any second string argument as a format string. This issue is caused by plot() having a unique kwarg parser that isn't used in the other methods (such as scatter()) that are mentioned in the issue. Improving the plot() parser should be straightforward and require us to check the other types of arguments before treating the second argument as a format string for the colour specification. Essentially, we must improve the current heuristic to only treat string arguments as format strings when evidence supports that the second string is not a name passed in with a structured array.

However, there will be a lot of work which needs to be done in order to test this fix. Since this fix will directly influence *plot()*; one of the main plotting functions in Matplotlib, thorough testing must be made in order to ensure that the fix does not affect all the possible ways of currently using *plot()*. However, we can take advantage of Matplotlib's existing test suite to ensure we don't cause any major issues.

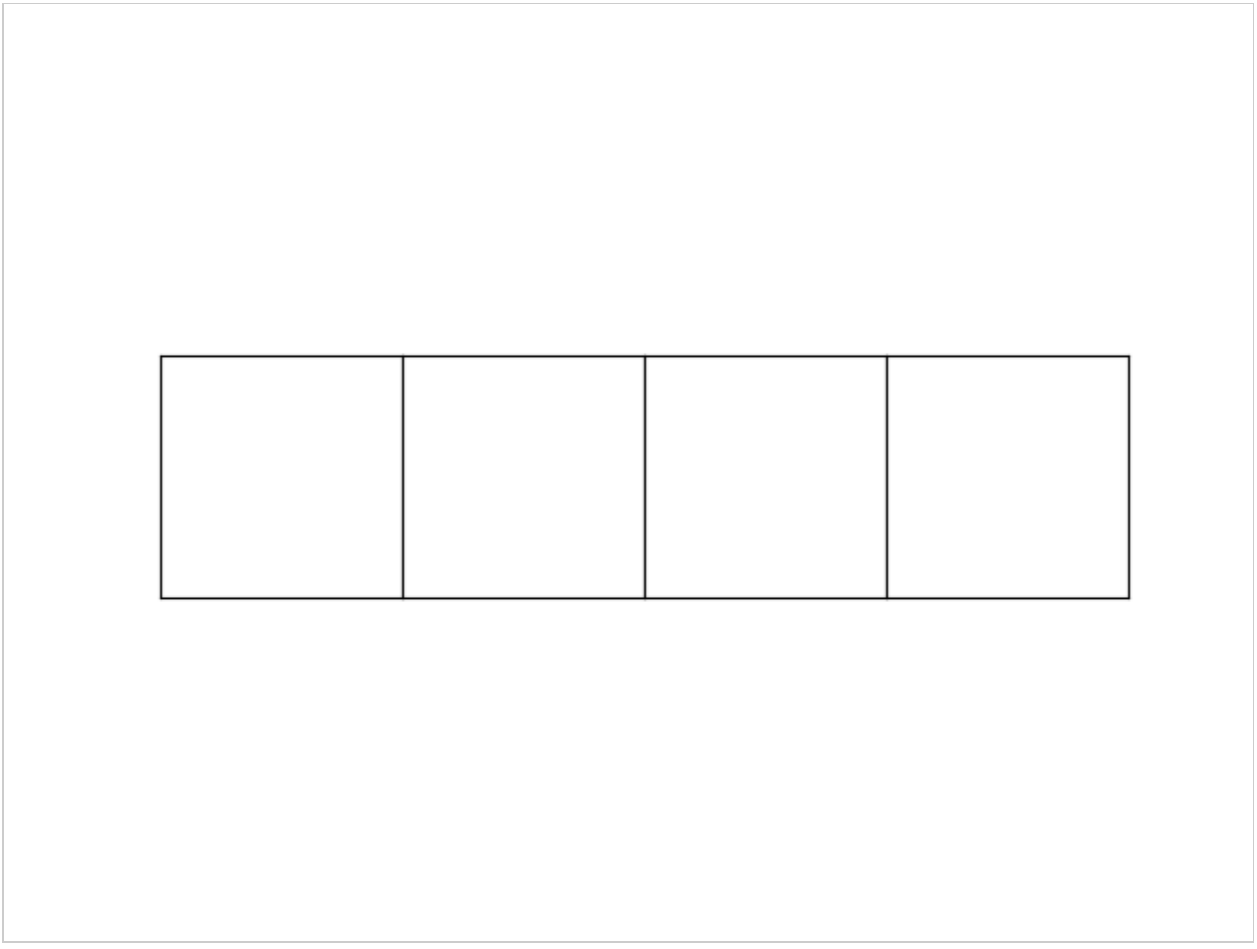
We've allocated 25 story points for D3. We're assigning this issue 13 points because we believe implementing a fix and testing it thoroughly will take about half of the time allotted for D3. Those 13 story points are broken down into more low-level stories, such as 3 story points being allocated to further investigating the plot() functions kwarg parser to ensure we don't damage existing functionality. We've also assigned 5 points to developing the improved heuristic for determining the correct treatment of string arguments, and 5 points to testing our changes to ensure we haven't removed or changed existing functionality.

Issue #9462

Issue #9462 on GitHub || Occurs in Python 2 and 3 on Ubuntu

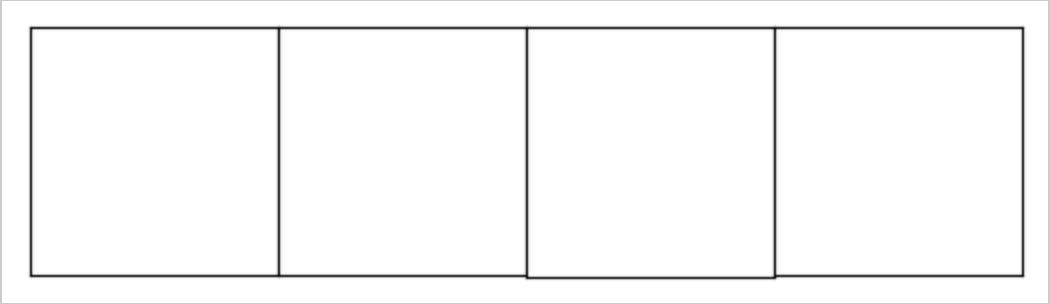
When subplots which have their ticks, grids, and labels removed and are plotted side by side tightly with no space between, the subplots will have misaligned bottom edges in a rendered PNG image. The problem stems from using the *bbox_inches='tight'* parameter when rendering the image, which only removes any extra white space around the figure without rearranging anything after rendering. Through testing, we discovered that this issue start occuring when there are 4 or more subplots side by side.

We demonstrate the behaviour of rendering without the *bbox_inches='tight'* parameter and show that the subplots are rendered as expected, with top and bottom edges aligned:



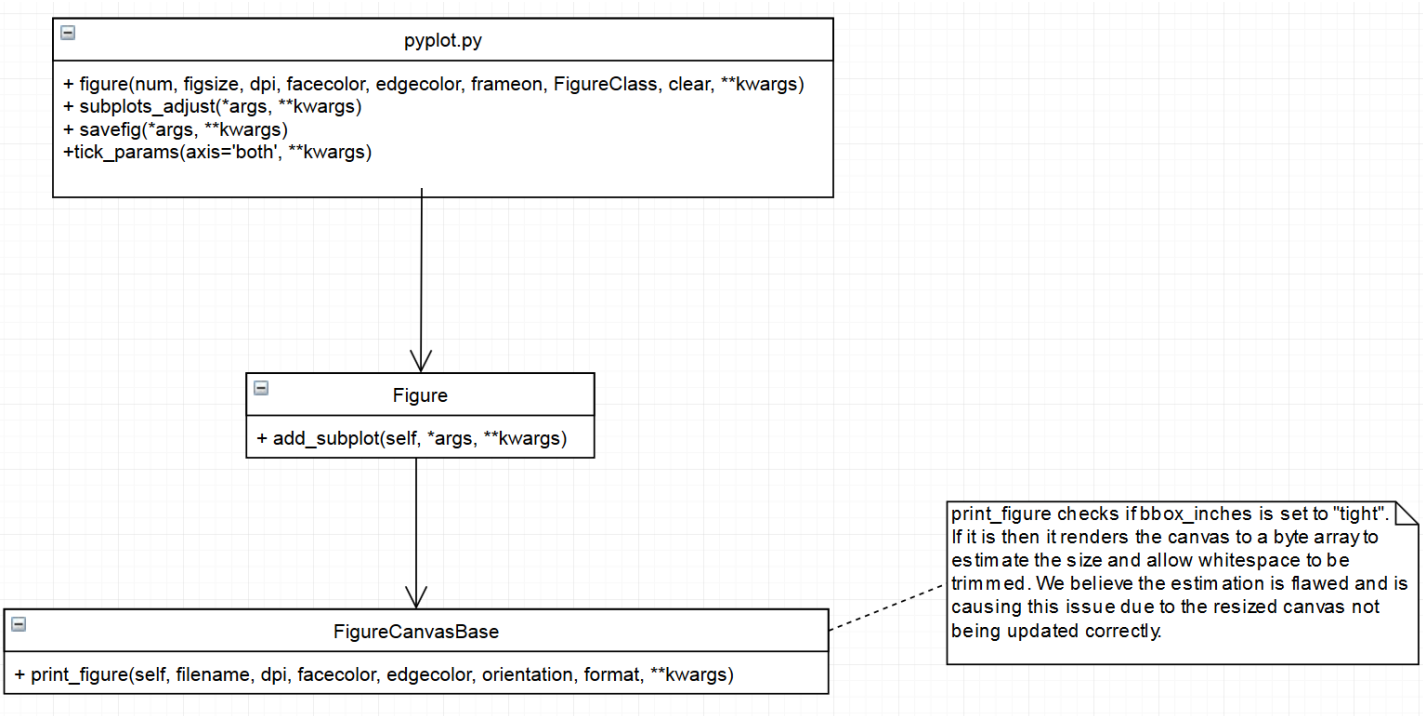
Note that the image has lots of whitespace around the subplots, as expected (gray border added to place emphasis on the boundaries of the image).

We demonstrate the behaviour of rendering with the *bbox_inches='tight'* parameter and show that the subplots are rendered, but with misaligned bottom edges:



Note that the image contains 4 boxes, in which the 3rd box from the left is misaligned with the other 3 boxes. The borders tightly bound the figure, as expected (gray border added to place emphasis on the boundaries of the image).

UML



Analysis

As mentioned in the UML diagram, our team believes the estimation of size when rendering the canvas to a byte array may be flawed, causing the bug to occur. However, we are not completely sure that this is the source of error, and so we may need to do additional research regarding

image rendering in order to fully understand and fix this bug. For this reason, there is a lot of story points going into this bug from the coding/research aspect.

This bug occurs in a very specific condition (on subplots with no labels, ticks, grid, and spacing on a tightly bound canvas), however, altering the behaviour of rendering a canvas will affect all types of rendering. Hence, we will need to do a thorough testing as well to ensure that rendering other types of figures are not affected, on all operating systems. Hence, there is also a lot of story points going into the testing aspect of this bug.

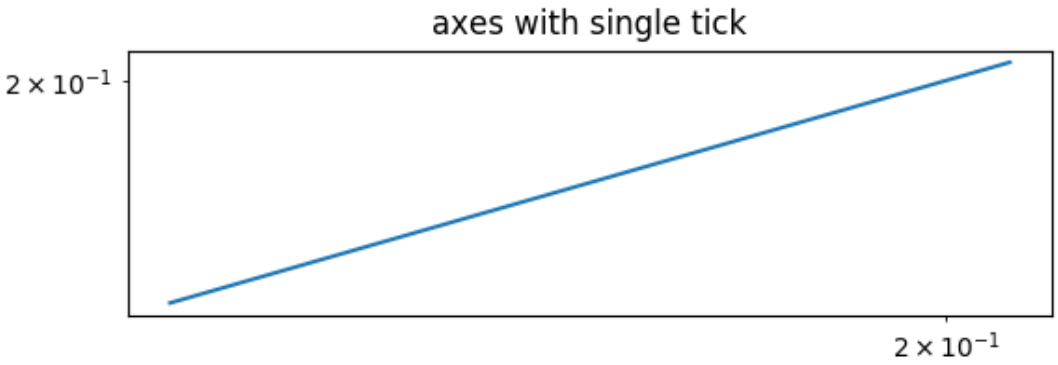
We're assigning this issue 8 points because the issue is relatively isolated and requires less testing. 2 points are allocated towards researching the image renderer, 4 points are allocated to implementing improved resizing/trimming logic for "tight" graphs, and 2 points are allocated to testing to ensure desired behaviour and lack of regressions.

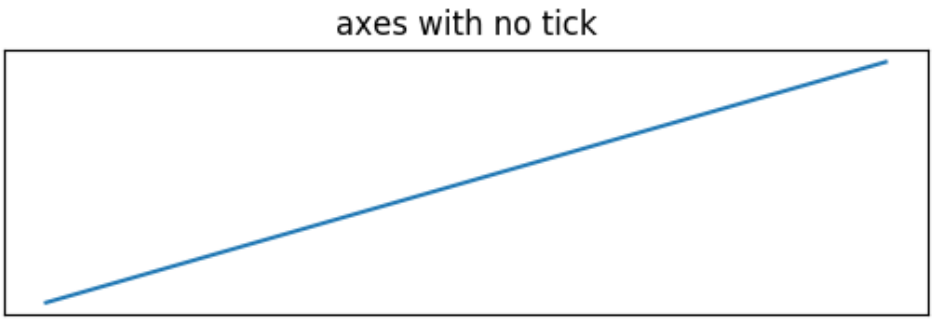
Issue #8768

Issue #8768 on Github || Occurs in Python 2 and 3

When using *pyplot.loglog()* to make a plot with log scaling on both the *x* and *y* axis, plotting two points within a small range of values will default to an insufficient amount of ticks on the axes.

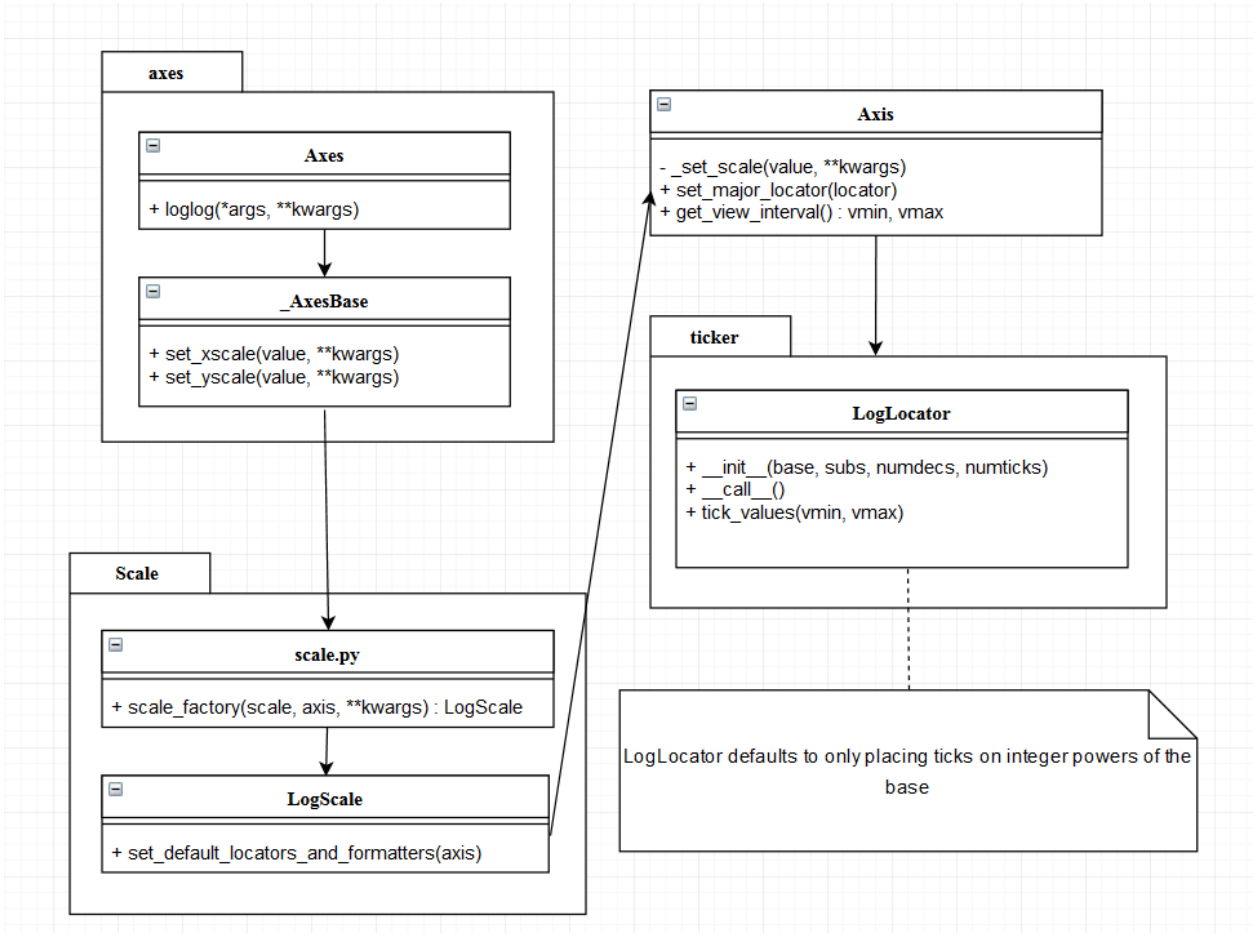
For example, plotting (0.11, 0.21), (0.11, 0.21) using pyplot.loglog will only display one tick per axis, and (0.11, 0.12), (0.11, 0.12) will display none. *LogLocator* defaults to only placing ticks on integer powers of the base. For instance, (0.11, 0.21) would only display a tick at 2 *10^-1 = 0.2. This most likely wasn't considered in the design due to particular interactions between logarithms and small numbers. The oversight, however, is that graphs with less than two ticks per axis are almost meaningless. Below are two examples of graphs with small ranges that result in 1 tick for values in the 0.1 range and no ticks for values that are significantly smaller.





For now, the expected behaviour can obviously be produced by manually providing the optimal axis ticks the user wishes to observe in their figure.

UML



Analysis

The time to fully fix this bug is dependent on the appropriated solution. One such solution would be to switch to using *MaxNLocator* when the range drops below a certain threshold. Validation to ensure this does not negatively affect any other areas of code would take some time since this would be implemented outside of *LogLocator*. Generally, another solution would involve modifying *LogLocator* appropriately, which would minimize the area of affected code, but require more effort in understanding how the number of tickers are calculated.

We're assigning this issue 4 points because the fix is straightforward. 2 points are being allocated toward developing the heuristic change in tick formatting for small ranges, and 2 points are dedicated to ensuring no regressions have occured.