D01LAR BILLS/Team 30

CSCD01 WINTER 2020

Deliverable 2

Submitted By :
Joseph Sokolon
Wesley Ma
Raya Farhadi
Edgar Sarkisian

Submitted To:
Dr. Anya Tafliovich

Contents

1 Bugs

1.1 “Size” ignored if placed before fontproperties

The kwarg ‘size’ doesn’t get set when placed before the kwarg ‘fontproperties’ in functions that relate to functions that display characters on the figure. This behaviour is unintuitive, as we expect more specific properties to take precedent over less specific ones, regardless of order. In this case, specifically setting the font size should overwrite attributes set by setting a FontProperties class, which assigns a set of default values.

e.g.

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

data = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4]
plt.hist(data, bins=10, facecolor="blue", edgecolor="black", alpha=0.5)
plt.xlabel("value", fontproperties='DejaVu Sans',size=20  ) # this will work
plt.ylabel("counts",size=20, fontproperties='DejaVu Sans')
# this fails to set the font size to 20.
plt.show()
```

Having fixed this issue, the outputted plot should have the same sized fonts for its axis labels

This most likely is an ordering issue within the update function of the class Text. The update function reads the given params and applies them. Should only need to modify the Text class.

* We are selecting this bug, because it’s clearly scoped to a specific class, and is straight forward to fix. A single test case is enough as well.

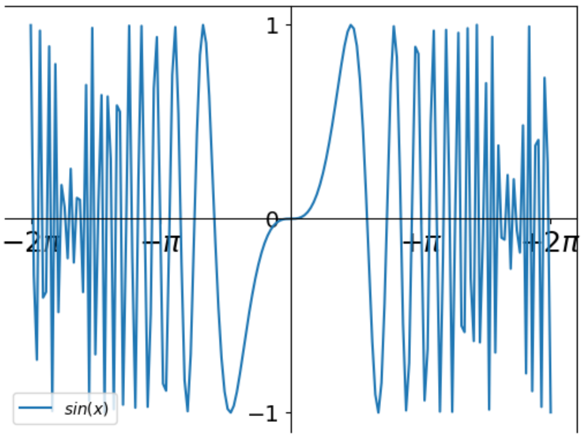
Our group estimated this issue would take 2 hours to fix.

1.2 Tick labels rendering under curve

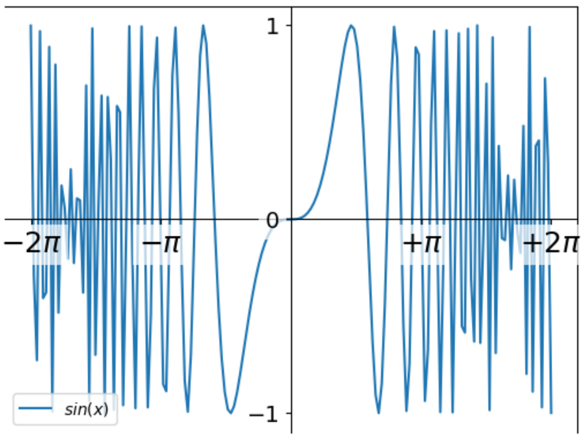
In a previous version of mpl when a graph is made with the axes going through the middle, the tick labels would show on top of the function curve. Now, they're being rendered behind the curve, so if the line thickness is large they would be substantially covered. Additionally, if the tick labels have bounding boxes (background rectangles of a different color than the curve to increase readability) they are also rendered beneath the curve so they are useless.

```
ax.spines['bottom'].set_position(('data',0))
plt.xticks( [-6.28, -3.14, 3.14, 6.28])
for xtick in ax.get_xticklabels():
        xtick.set_bbox(dict(facecolor='white', edgecolor='None', alpha=0.7 ))
```

creates



instead of

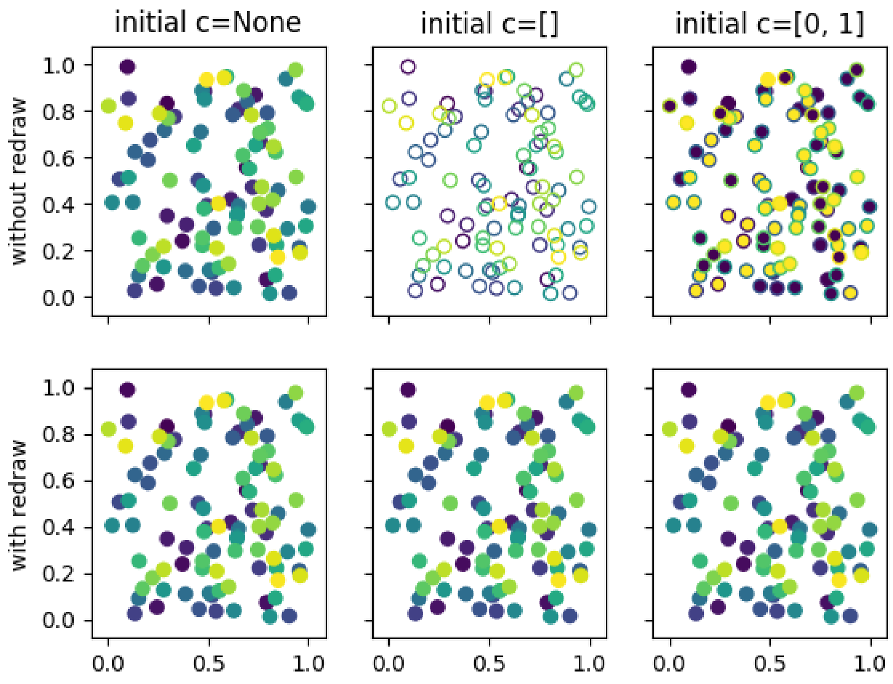


The zorder of the artists depends on the z-order of the axes' children, since in Axes' draw() it sorts all its children by z-order and calls their draw() functions one by one. According to a zorder demo file, the Line2D artist has a zorder of 2 while the text artist has a z-order of 3, but since text isn't a child of Axes, (its a child of Tick which is a child of Axis which is a child of Axes), its z-order isn't considered in Axes' paint function and so the line2d is drawn first.

* We believe this is a good issue to tackle because it seems to be an isolated problem - its highly unlikely that other artists have composites that need to be drawn at different z-levels so this would be a small change in the Axes' draw() function. We estimate it will take 5 hours to implement and write thorough tests for this feature.

1.3 Odd behavior when using 'collections.set_color'

In scatter plots, if the c argument is not None, the fill color is not passed to the scatter plot. Upon redraw, the fill color is passed correctly to the scatter plot.



The expected behaviour is that without redrawing, if a c argument is passed to pyplot.scatter(), the fill color is still propagated correctly to scatter plot. In the init function of scatter, a method called _parse_scatter_color_args is called. Our hypothesis is that in this function, there is a logic error which is causing colors not to be set correctly when an array is passed. The variable c_is_mapped is set to true, which causes the colors variable to be set to None.

Our group estimated this issue would take 8 hours to fix.

1.4 Axes with sharex can have divergent axes after setting tick markers

When we have multiple axes', with one sharing (depending) on the x or y axis of another, the depending axes won't be updated if we update the source axes' limits. Interestingly the other properties that need to be propagated (tick and scale) work as intended. e.g.

```
fig = plt.figure()
ax1 = fig.add_axes([0.1,0.5,0.8, 0.4])
ax2 = fig.add_axes([0.1,0.1,0.8, 0.4], sharex=ax1)
...
ax1.set_xticks(range(-10, 20, 1)) # update source
```

```
print (ax1.get_xlim())
print (ax2.get_xlim())

# ouput
(-10.0, 19.0)
(-2.0, 16.0)

# expected
(-10.0, 19.0)
(-10.0, 19.0)
```

In the above example, ax2 depends on ax1. If their x_lims were to be be printed right away, the outcome is as expected (same value). However, if the ax1's tick range is is updated, and we output again, ax2 does not have the updated result.
We believe the bug/fix resides in the _AxesBase class in axes/_base.py. Specifically, the set_xticks() function does NOT emit the updated values, unlike the other propagated properties. The potential fix is to calculate a new limit based off the updated tick range, and then call set_x/ylimit(), which will properly emit.

Our group estimated this issue would take 6 hours to fix, also accounting for any other possible side effects that need to be propagated.

1.5 Memory leak with log scale in pcolorfast, pcolormesh, imshow

The following functions pcolorfast, pcolormesh, imshow cause memory leaks when the scale of the Y Axis is set to logarithmic (set_yscale('log'). On a 4GB machine all memory was claimed in just over a minute. Because of this we consider this to be a critical issue that should be fixed for the next release. The function pcolorfast is an experimental improvement on pcolor for the Agg backend. Depending on the complexity of the input grid it uses different internal plotting algorithms. We noticed however that in the documentation for the function it is noted that there is a lack of support for logarithmic scaling of the axes. We think even before a fix, throwing an error when the scaling is switched to log would be an improvement. pcolormesh and imshow are other alternative and faster implementations to pcolor aswell, however they also suffer from the same memory leak. After further investigation, our team was not able to identify the exact location in the code of the memory leak. In the issue thread, a user hypothesized that the memory leak would be caused by a transform not being properly cleaned up.

Our group estimated this issue would take 15 hours to fix.

2 Fixes

2.1 Bug 1

This bug is located within the update() function of the Text class (in Text.py), which calls the super update() from class Artist. Artist's update() sequentially goes through all the given params and applies them, whether by assigning the attribute directly, or by calling another setter function for the type of param it's currently on (as is the case with fontproperties).
Before the fix, if it saw fontproperties first, it end up setting some default values about the font, such as size, and then later set the size once again overwriting the first time, as expected. However, if the actual size argument is seen first, then it would get overwritten by the default ones.

Our fix was to simply to pop and apply the fontproperties directly in Text's update() *before* the other more specific font attributes are handled in Artist.py's update(). Along with the fix, a basic test case was also added to confirm that two instances of Text had the same font-size regardless of size/fontproperties order.

2.2 Bug 2

This bug is located within the draw function of _AxesBase. The problem is that in this function the artists are sorted by zorder, however TickLabels are children of Axes, so whenever the Axes is drawn the TickLabels are drawn aswell, effectively causing them to inherit the zorder of the parent Axes when they have their own zorder value. This is confusing to users as changing the zorder of the Tick or TickLabel will have no effect on its rendering order.

Our attempted fix was to extract the TickLabels from the Axes and rendering them in the Axes draw method, thus respecting its z-order w.r.t Axes' other children, instead of having Axis's draw method render the tick-labels. We also stopped Axes from drawing TickLabels so they wouldn't be drawn twice. Our solution worked and fixed the original issue, however we inadvertently broke hundreds of other regression cases. We continued to resolve issues however we ultimately were unsuccessful at restoring regression. We further discuss the issues we encountered in our technical commentary below.

3 Technical commentary

Write a brief technical commentary on how your changes affect the design and/or the code of the project. List ALL relevant source code files that were added, modified, or removed as part of your implementation work.

3.1 Bug 1

Although our fix addresses the bug described in the issue, what if the same pattern (a property setting multiple other properties at once) occurs in another existing class, or will appear in a future class? Developers would have to add extra checks in the update() of all such classes. If this is the case, it may be worth considering modifying the individual setters, such as set_fontpropterties() to ensure that they are not overwriting already modified properties, rather than handling this issue by modifying the execution order.

3.2 Bug 2

As mentioned in section 1.2, we believed the fix was to extract the drawing of the tick labels from Axis's draw() function to Axes' draw function where it would be draw respecting its z-order. This, however, proved to be harder than expected. Getting the tick-labels from the axes required some more logic than we had initially estimated:

- we needed to check if the graph is 3D, in which case we had to call `.zaxes.get_ticklabels` and add them to the list of artists to be drawn
- we needed to check if the graph has multiple axes, like so, and get their tick-labels
- `axes.get_ticklabels()` also sometimes returned extra tick-labels that were to the left or to the right of the bounds of the figure being drawn, so we needed to catch and remove those

The function that originally took care of drawing the Ticks was doing much more logic than we had originally thought, and unfortunately we could not implement it all in the time we had on this deliverable.

Additionally, we realized that extracting all this logic into the Axes' draw() method would not lead to a good overall design that didn't break the class's cohesion. Most of this logic should be done by the Axis class as it does currently, but then the z-order would not be respected. We will continue working on this and take time to think of a better design.