Deliverable 2: Matplotlib Bug Fixes

Benjamin's Grooper

CSCD01

Anya Tafliovich

David Gao, Simon Ngo, Andrei Grumazescu, Raunak Mathur, Tony Ng

Winter 2019

Table of Contents

Issue 8844

Description of Bug:

Under certain circumstances when using pcolourmesh() with Flat shading a graph is produced, but given the same parameters but asked to use gouraud shading a ValueError is returned.

Underlying Cause:

The bug occurs when using gouraud shading but not flat shading because flat shading and gouraud actually use the parameters X and Y in different ways, which is not made clear. As a consequence of this, X and Y parameters which work with flat shading do not always work with gouraud shading. Specifically when X and Y are defined as ranges from some Integer +/- 0.5 to a different Integer -/+ 0.5, and the range taken in by the c parameter are one smaller than the stated size of X and Y. Normally this is ok as pcolormesh allows X and Y to be one greater than the declared size of X and Y, but X and Y start and end at a non-integer in conjunction with the use of gouraud shading this functionality no longer works, and a ValueError is returned insead.

The Fix:

Solving this bug came with the requirement that we would approach the solution in a different way than the methods proposed by issue 9594, which was a pending verification work in progress fix to the underlying cause, and involved changing the inconsistency between the way flat and gouraud shading handles parameters X and Y. We decided that TA meeting that even if the underlying cause cannot be fixed the same way the following things are agreed upon and should be worked towards:

1. The same parameters given to pcolormesh on successful flat shading should not result in a crash when using gouraud shading. This is because the parameters of pcolormesh should not be different or depend on the implementation of gouraud shading. If anything it should be handled inside the body of pcolormesh, or change the gouraud implementation.

2. Using pcolormesh’s functionality to slightly reshape the graph or some other process that slightly modifies the parameters to a functional set would be a prefered alternative to crashing, providing the user is given a warning that their parameters have been modified.

The changed code starts on line 5981 of _axes.py inside of matplotlib\axes_axes.py. The new code catches a ValueError if gouraud shading is being performed by pcolormesh and then attempts to create a flat shading graph with the same parameters. If an error is returned by this then we know it is not an instance of the bug, and the original pcolormesh with gouraud shading method is called to return a precise error. However if the flat shading graph can be made without errors then the parameters should be slightly adjusted to allow a gouraud shading graph to be returned. This is done by making a new X and new Y which have the original range, but instead of a size declared which is one larger than the range and 1 larger than the given c, it is exactly equal to the size of the range. A side effect of this is that it cuts off the last row and column of c, but the pcolormesh no longer returns a value error, and the graph should look relatively the same as it would have otherwise unless it is working with a very small set of parameters.

Potential Side Effects:

Besides the side effect of cutting off the last row and column of c, there are no side effects to this modification. This is because the modified graph only is created when gouraud shading would fail and flat shading would succeed before. Instead of improperly getting an error, the user now receives a graph, and only in that circumstance. Under no other circumstance would the user receive a graph with modified parameters, nor receive a graph when it should be an error instead.

Issue 5456

Description of the bug:

Whenever a graph has been labeled with axes that are very large in length and has tight_layout applied to it, matplotlib denies the user the ability to use tight_layout. Instead, the tight_layout line is ignored and a warning message is displayed in the console.

Underlying Cause:

The purpose of tight_layout is to adjust subplot parameters so they would all fit into the graph nicely. Since some axes can be very big, matplotlib denies the user the ability to use tight_layout since the axes cannot fit into the graph under any circumstances. As discussed in the TA meeting, there could be a solution where tight_layout could still be used if matplotlib manipulated the length of the axes.

The Fix:

The idea behind the solution was to first check how long an axis is allowed to be and shrink the axis text until matplotlib is satisfied. By analyzing the function, auto_adjust_subplotpars() in the /matplotlib/tight_layout.py file we noticed that the margins for the left, right, top, and bottom of the graph were being calculated through a lengthy process involving the axes. Eventually, the margins for the x (left and right) and y (top and bottom) axes were checked to see if they exceeded a value of 1. Using this method, we can re-use the margin calculating code to continually recheck an axis with one less character at the end.
This led to the creation of 3 new functions in /matplotlib/tight_layout.py. The first two that worth mentioning are check_y_margin() and check_x_margin(). These functions re-use code from auto_adjust_subplotpars() to get the margins for the y and x axes after they have been changed. These two functions are helpful in the third function, proper_fit_text(). This function

gets the string values from the x and y axes and stores them. Then, it keeps re-writing the axes with the same text from the previous axes, but cuts off a character from the end. Afterwards, the margins are re-calculated and then this process repeats if necessary. This process repeats until the margins meet a satisfactory value. In this case it is <0.15. It is worth noting that auto_adjust_subplotpars() checks the sum of the axes margins to be < 1 and that commenters on the issue page on GitHub claim each individual margin should be <0.5, I have personally noticed that 0.15 provides the best results onto the graph, which is why it is used. Now, auto_adjust_subplotpars() has replaced the section of the code that ignored tight_layout and returned nothing with the proper_fit_text() function.

Potential Side Effects:

The process of calculating margins is lengthy. For margins that have lengths in the hundreds, graph creation may take a bit of time. This solution also removes the warning for the incorrect use of tight_layout. The is a chance that many may use tight_layout when it is not appropriate to do so. The axes were appended with ellipses to let the viewer know that their axes have been shortened. Since no other files except for /matplotlib/tight_layout.py were edited, it will not affect many other files in matplotlib.

Issue 11759

Description of Bug:

When the user tries to plot a 3D graph that displays arrows, the arrow heads colours are displayed with an inconsistent gradient and are not always matching the colour of their bodies. The body of the arrow is working and being plotted correctly by increasingly getting darker. This bug is found in the mpl_toolkits folder instead of matplotlib folder.

Underlying Cause:

Within the quiver method inside class Axes3D in mpl_toolkits/mplot3d/axes3d.py each arrow heads is actually created as two lines that are at a different angle to the body of the arrow. The angle at which the arrow heads are being set is done by the method calc_arrow within Axes3D. Originally the main body lines and the lines used to generate the arrow heads are being passed together to create a single Line3DCollection object.

```
lines = [*shafts, *heads]
.
.
.
linec = art3d.Line3DCollection(lines, *args[argi:], **kwargs)
```

This object generates the lines which will be used to plot on the axes. It includes generating the colours of the lines if that is included as a parameter when the object is created. Since originally the lines and both sets of lines for the arrow heads were being passed in to create the single Line3DCollection object, there were inconsistencies with generating the proper gradient for the final arrows. The lines representing the arrow body were gradually getting darker but the arrows were inconsistent in their gradient.

The Fix:

After understanding the implementation of each arrow head actually being two lines set at different angles based on the line representing the body of the arrow and seeing that the lines for the body of the arrow were already being plotted with the correct gradient, the fix was to create separate Line3DCollection for each collection of lines. Instead of one single Line3DCollection object to be plotted containing the body lines and then the arrow head lines, we now split it into a Line3DCollection object for the body lines, a Line3DCollection object for one side of the arrow heads, and a Line3DCollection object for the other side of the arrow heads. This essentially allows us to treat the arrow heads in the exact same way as the lines representing the body of the arrow without having to change any existing code. The reason the arrow heads must be separated depending on which side of the arrow body they are on is so that each final arrow has the same colour for all three lines. If the arrow heads are grouped together, the shade of the left and right lines representing each arrow head will be different. These new Line3DCollection objects are then added to the plot by calling self.add_collection method. The code for this fix can be found from line 2649-2668 within quiver method inside class Axes3D in mpl_toolkits/mplot3d/axes3d.py. To use this fix you must update the mpl_toolkits/mplot3d/axes3d.py with the new file.

Potential Side Effects:

This solution does not change any existing code other than slightly modifying lines variable shown above which initially contained the arrows body lines and the arrow head lines. Since the quiver method within class Axes3D in mpl_toolkits/mplot3d/axes3d.py is only being called when the user wishes to create arrows, the additional code used to fix this bug should not cause problems to anything else within matplotlib

Tests

Unit and acceptance tests can be found inside the matplotlib\testing\Deliverable_2_Test folder.