University of Toronto – Scarborough
CSCD01 – Engineering Large Software Systems

Deliverable 05 – Report

Prepared by ***GoonSquad*** Team

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

Date: April 06, 2018

Table of Contents

Feature #1: Contour Label Extensions

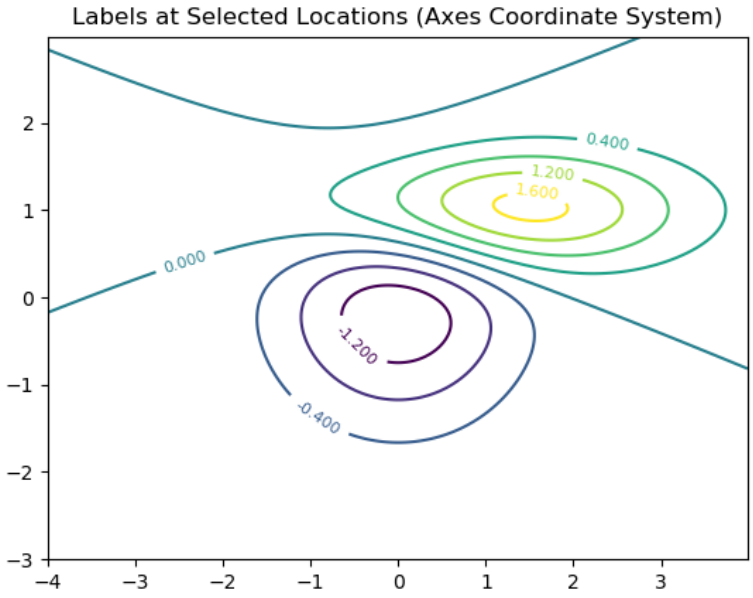
Feature Request: Extended manual positioning of contour labels #9109

Estimated Hours:

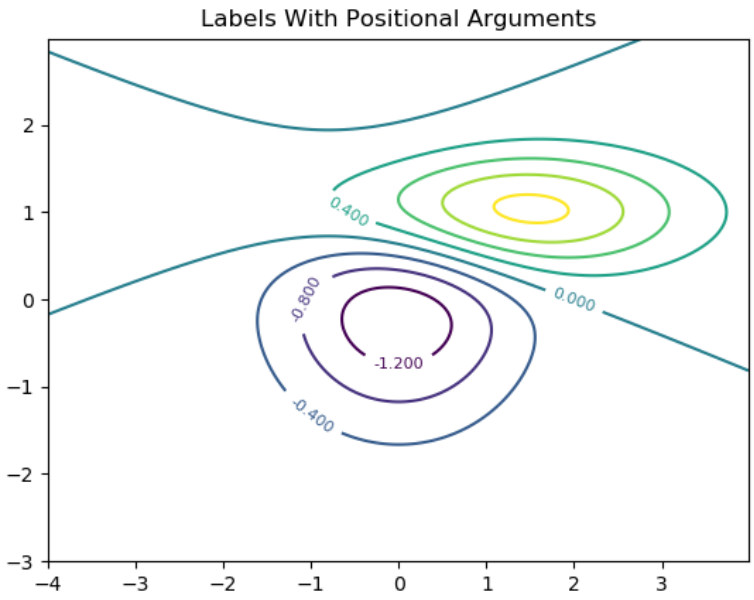
- Explore and create a solution (15 h)
- Implement solution (28 h)
- Testing/validation (8 h)
- Code Review (4 h)
- Documentation (3 h)
- *Total: 58 h*

Description:

This feature is actually a two in one bundle. The first feature consists of allowing a ***transform*** to be used with the ***clabel()*** method. In particular, a kwarg of ***transform*** parameter is added so that a user can pass the coordinate system they wish to use when entering manual locations for the contour label. Currently, only the data coordinate system is used, therefore locations are based off the data provided. But by passing in a ***transform*** such as ***ax.transAxes,*** the user can enter locations based off the axes coordinate system. For example, the figure below had their contour labels located using the axes coordinate system. The manual locations used were: [(0.4, 0.3), (0.47, 0.45), (0.2, 0.55), (0.7, 0.7), (0.72, 0.72), (0.8, 0.8)]. So as you can see, the pair of coordinates (0.4, 0.3) correspond to 40% of the x axis and 30% of the y axis. This is where the contour label of -0.400 is placed. Keep in mind that contour labels are placed to the nearest contour, which means that the locations provided are not necessarily in the exact coordinates given.



The second feature is about only providing one xy-coordinate and a positional coordinate. A positional coordinate is a one-element list to determine the location of the nth intersection of a contour given the xy-coordinate. So, below is an example using positional coordinates. The locations given were [(-1, [0]), ([2], 0), (0, [2]), (2, [0]), ([0], 1)]. The contour label of -1.200 corresponds to the coordinate pair of (0, [2]). What it says is to place a contour label at x-value 0 and a y value where a 3rd contour intersection occurs. Note that the positions are 0-based. In addition, a user can start from the last intersection by providing a negative number, where -1 would be the last intersection.



Implementation:

Updated Forked Repo

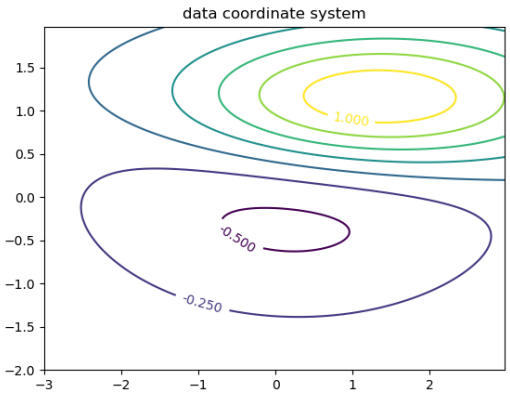
The code was implemented by tinkering and working exclusively with ***contour.py.*** However, other areas of MPL were looked at to further understand their functionalities. First of all, to add the ***transform*** kwarg,for ***clabel(),*** we added code to look for that kwarg when the function is called. Once we're able to do that, then we have the ***transform*** and thus the coordinate system the user wishes to utilize. From there, we let the already implemented method for transforming the given inputs to the corresponding coordinate system.

For the positional coordinates, it was a bit tougher to figure out since we needed to work with how MPL handles the intersections and how they place the label to the nearest contour. This took a lot of time to determine and was the bulk of our workload. In the end, we created a function called ***get_contour_intersects()*** to determine when a contour intersects for a given xy-coordinate. The decision to have a positional coordinate to be a one-element list was to differentiate between a xy-coordinate and a positional coordinate. Thus, we are happy with the implementation of this feature.

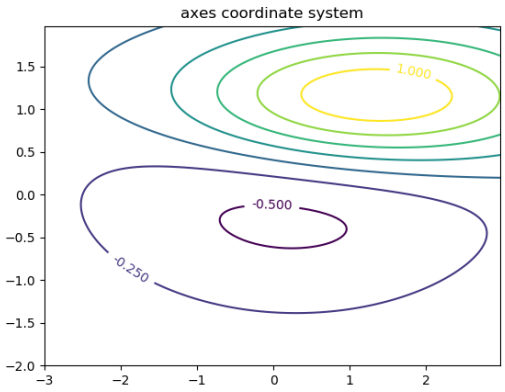
Testing:

To test that the feature works, image comparison tests were used, as suggested by matplotlib when testing for changes to the graph figure. The associated tests are in ***contour_label_extensions/test/***. The file ***test_contour_label_extensions.py*** contains the specific test cases for the ***transform*** kwarg and as well as for the positional coordinates. The result images are found in the folder ***contour_label_extensions/test_contour/***. In addition, ***test_contour.py*** is the formal way to test the contour functionality of MPL, thus it includes the existing tests for contours and the newly added test cases in ***test_contour_label_extensions.py.***

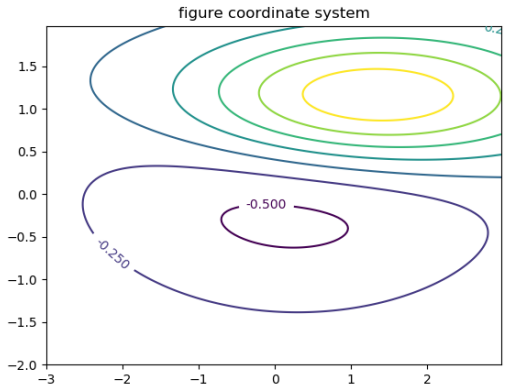
Image comparison tests were used since contour labels must appear in the appropriate place on the figure. Moreover, we can test that the coordinate system is being changed by having a data coordinate system and axes coordinate system have the same contour labels in the same position but different locations are given. Below are the images used for the tests.



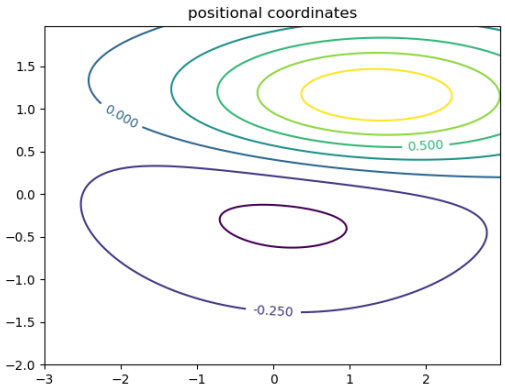
test_clabel_transform_data.png



test_clabel_transform_axes.png



test_clabel_transform_figure.png



test_clabel_positional_coordinate.png

Integration:

The code we provided for the feature will integrate well with the current capabilities of the contour labeling since it is an add-on. Therefore, the previous functionality will work as expected. In addition, a new method was added to ***contour.py*** so no other code was affected. All in all, the overall code does not impact the current MPL build and a user with code built with an older MPL version will still be able to run their code without any problems.

Documentation:

Please refer to /***contour_label_extensions/doc/contour_label_extensions_demo.html*** for a Matplotlib style documentation of the feature.

Feature #2: Axes Default Title Location

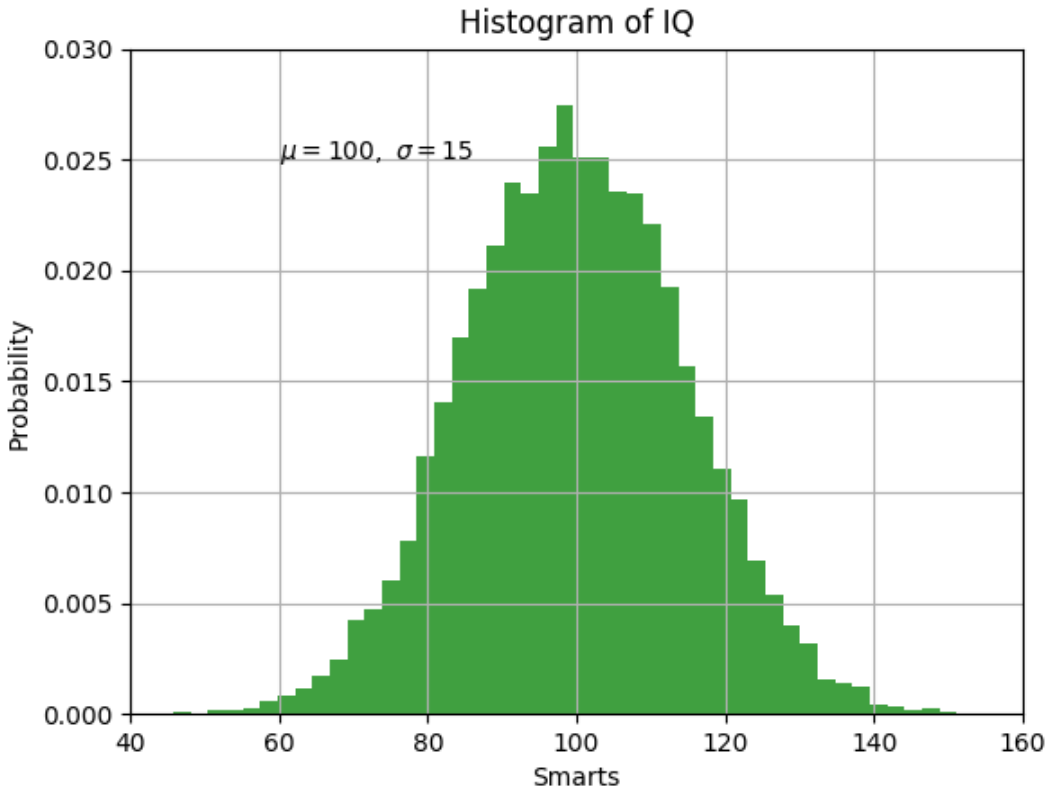
Feature Request: Can I change default title location with matplotlibrc? #9682

Estimated Hours:

- Explore and create a solution (5 h)
- Implement solution (5 h)
- Testing/validation (6 h)
- Code Review (3 h)
- Documentation (3 h)
- *Total: 22 h*

Description:

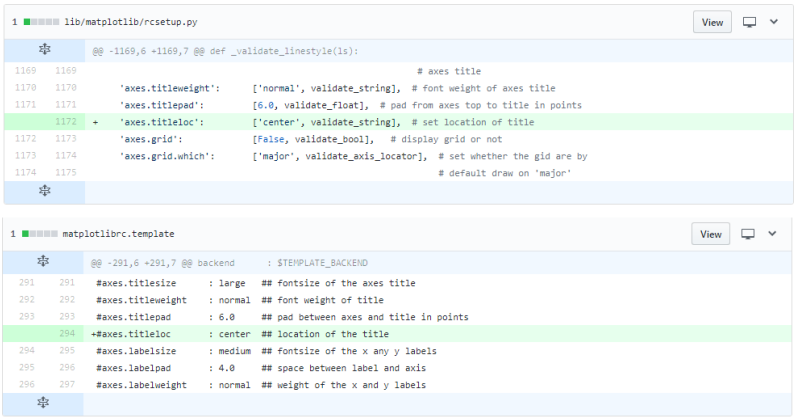
Currently, the default location of the title is “center” and there is no way to change this default location of the title on matplotlib. Usually the default behaviors can be specified by changing the matplotlib rc Params, however there is no such rc Params variable for the default location of the title. A request was made for this feature, as a user was building his/her own style and wanted to make the default title location left aligned.



Solution:

Updated Forked Repo

In the original feature request thread, the necessary work to be done was outlined. A new rc Param needed to be added, and the relevant logic needed to be added for the file responsible for creating the title. Following this outline, a few files were altered to introduce the new feature. First, ***lib/matplotlib/rcsetup.py*** and ***matplotlibrc.template*** was changed to include the new variable rcParams[‘axes.titleloc’]. This variable defaults to ‘center’, but it can be changed to ‘left’ or ‘right’ depending on the user’s preferences.



lib/matplotlib/rcsetup.py

```
@@ -1169,6 +1169,7 @@ def _validate_linestyle(ls):
                                                    # axes title
     'axes.titleweight':      ['normal', validate_string],  # font weight of axes title
     'axes.titlepad':         [6.0, validate_float],  # pad from axes top to title in points
+    'axes.titleloc':         ['center', validate_string],  # set location of title
     'axes.grid':             [False, validate_bool],   # display grid or not
     'axes.grid.which':       ['major', validate_axis_locator],  # set whether the gid are by
                                                                 # default draw on 'major'
```

matplotlibrc.template

```
@@ -291,6 +291,7 @@ backend      : $TEMPLATE_BACKEND
 #axes.titlesize      : large   ## fontsize of the axes title
 #axes.titleweight    : normal  ## font weight of title
 #axes.titlepad       : 6.0     ## pad between axes and title in points
+#axes.titleloc       : center  ## location of the title
 #axes.labelsize      : medium  ## fontsize of the x any y labels
 #axes.labelpad       : 4.0     ## space between label and axis
 #axes.labelweight    : normal  ## weight of the x and y labels
```

After the new variable rcParams[‘axes.titleloc’] was added, the corresponding logic was added to the file ***lib/matplotlib/axes/_axes.py*** to determine where the title of the figure/axes would be located. Originally, the variable ‘loc’ (which determines the location of the title) defaulted to ‘center’, and the user had the option of specifying where the title would be located at the time of creation. This option is still available with the new feature, so the user has the ability to not only specify the title location at time of creation, but also the default title location. If the user does not specify the location in the function call, it uses the value of rcParams[‘axes.titleloc’].

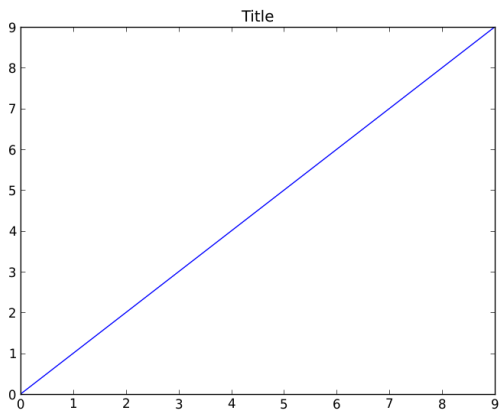
lib/matplotlib/axes/_axes.py

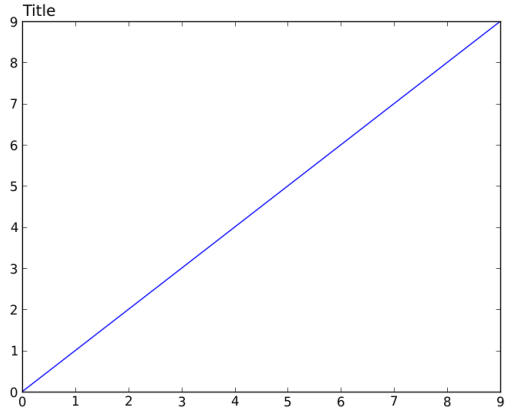
```
@@ -126,7 +126,7 @@ def get_title(self, loc="center"):
             raise ValueError("'%s' is not a valid location" % loc)
         return title.get_text()

-    def set_title(self, label, fontdict=None, loc="center", pad=None,
+    def set_title(self, label, fontdict=None, loc=None, pad=None,
                   **kwargs):
         """
         Set a title for the axes.
@@ -168,6 +168,10 @@ def set_title(self, label, fontdict=None, loc="center", pad=None,
             :class:`~matplotlib.text.Text` for a list of valid text
             properties.
         """
+        # if loc is not specified
+        # use rcParams value of default loc
+        if loc is None:
+            loc = rcParams['axes.titleloc']
         try:
             title = {'left': self._left_title,
                      'center': self.title,
```

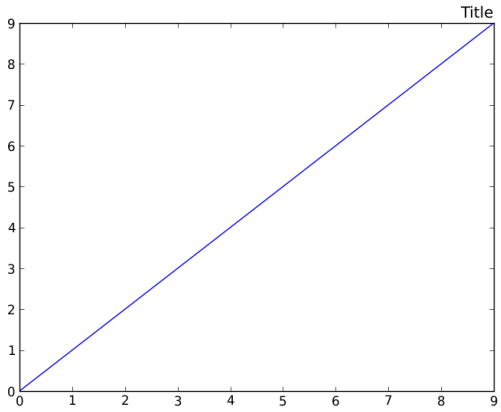
Testing:

To test and validate the feature, unit testing and integration testing were performed. The test cases/files are in ***GoonSquad/D5/default_title_loc/tests/.***

Since there are 3 possible default title locations, 3 cases were sufficient to cover all user-cases. Matplotlib's image comparison testing method was used to see if the resulting images had the correct default title placements with each rcParams['axes.titleloc'] being 'center', 'left', or 'right'.







Once the unit testing was successful, overall integration testing was performed as well as PEP8 and documentation testing.

Integration:

The confidence is high, as the implementation followed the outline given by a well-known contributor of Matplotlib, and all the correct procedures were followed for adding rc Params. The logic added was simple and got the job done. The test cases covered all possible cases, and the results of integration testing indicated that nothing was impacted or broken with the addition of this feature.

Documentation:

Please refer to ***/default_title_loc/doc/default-title-loc-doc.png*** for a Matplotlib style documentation of the feature.

Feature #3: Relative Linewidth and Markersize

Feature Request: Feature request: relative linewidth and markersize #10173

Estimated Hours:

- Explore and create a solution (8 h)
- Implement solution (7 h)
- Testing/validation (3 h)
- Code Review (3 h)
- Documentation (5 h)
- *Total: 26 h*

Description:

This feature is the option to choose from a set of strings (like "x-small", "medium", "large", etc.) when determining line widths and marker sizes. Additionally, new rcParams (lines.base.linewidth and lines.base.markersize) that affect the calculation of these relative size values should be created so that the values represented are configurable by the user.

Users should be able to set these new relative size options both by passing parameters into plotting functions and by using the existing lines.linewidth/lines.markersize properties of rcParams. Currently, the only type of parameter accepted as line widths and marker sizes are floats. The implementation of this feature would allow users to pick customizable relative sizes in additional to floats, as is the case for the existing fontsize property.

Solution:

Updated Forked Repo

In preparation for our solution, we read through the responses on the feature request, and made a quick plan for what we wanted to accomplish: adding the ability to specify the linewidth and/or markersize for their plots using quantitative string values, relative to a base value in rcParams, by converting it to a float when it's needed.

Following that, we looked for what files would be affected; along with what methods are relevant. We found that most of the work would be done in ***lines.py***, which is responsible for creating a line, specifying its linewidth, and also markersize. It contains the methods ***set_linewidth*** and ***set_markersize***, which we see as a good place to convert a string argument given as the width or size to a float. We also noticed through a search that ***patches.py*** and ***collections.py*** contain similar methods named ***set_linewidth***, and planned on adding support for this feature to those methods.

We began our solution by adding a couple new rcParam values to ***rcsetup.py*** to represent the base value, and also adding support to the existing linewidth and markersize rcParams by creating type verification functions for them.

After that, we discussed what string values we should support, and went with a format similar to the names we found ***font_manager.py*** in effort to be consistent with the existing code base.

For linewidth: ’xx-thin', 'x-thin', 'thin', 'medium', 'thick', 'x-thick', 'xx-thick'.

For markersize: 'xx-small', 'x-small', 'small', ‘medium’, 'large', 'x-large', 'xx-large'.

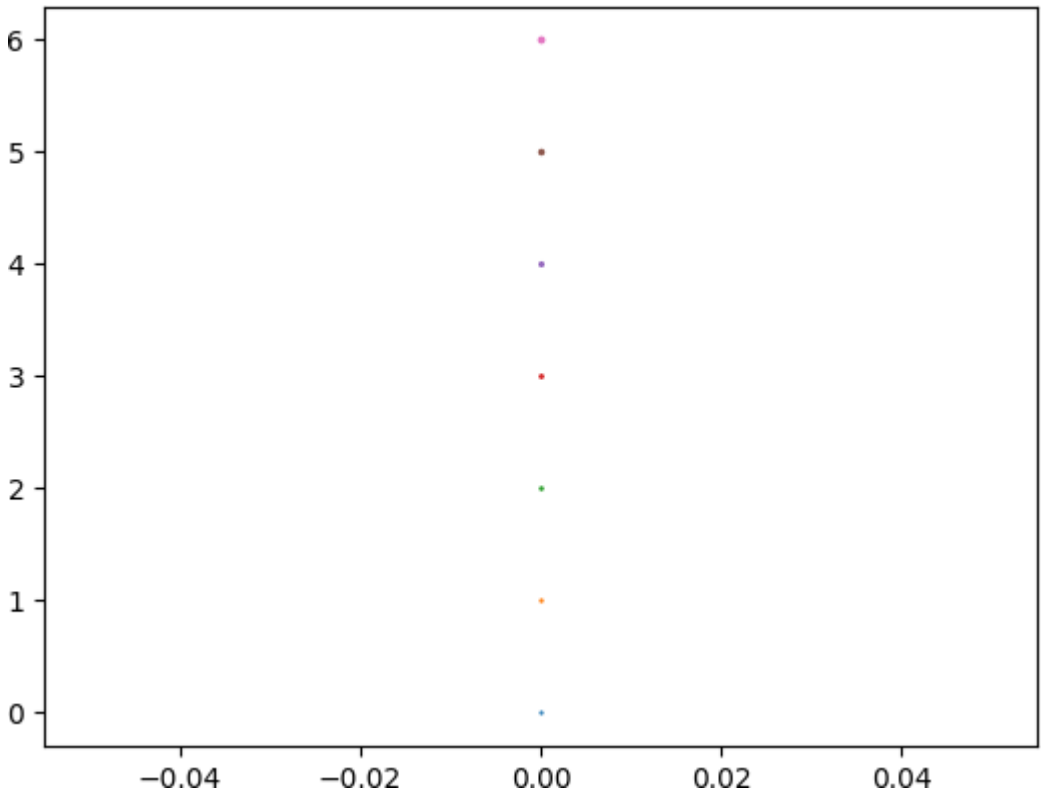
So, in ***lines.py,*** we added 2 dictionaries with those values as the key, and a multiplier as the value which would be multiplied with the base value rcParam we created. Following that, we created: ***_relative_value_to_points,*** which if the value is a string, it uses that to get the multiplier from the dictionary and multiplies it over the base rcParam value; returning a float. Otherwise, it just returns the value it was given. This method gets called by one of 2 other methods we created: ***linewidth_to_points, markersize_to_points***, which takes the linewidth/markersize, and returns its result. We created these methods so we wouldn’t have to repeat code that would otherwise have to specify which type of value we’re dealing with (linewidth or markersize) which are treated a little differently.

For most setter methods, we simply used ***linewidth_to_points*** or ***markersize_to_points,*** on the corresponding value to convert it into a float. The given value is only affected if it was originally a string. The method that needed another implementation was ***set_linewidth*** in ***collections.py,*** which can be given a sequence of linewidths. So we used list comprehensions to call ***linewidth_to_points*** on each item in the sequence, to convert them into floats.

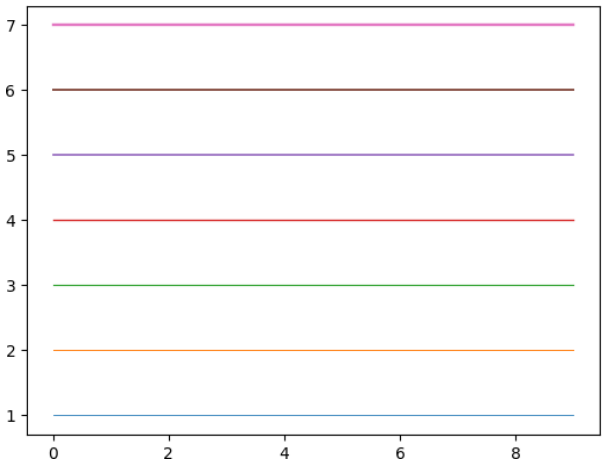
Testing:

Image comparison was used to test aspects of this new feature. Specifically, we tested that all the relative sizes for the markers as well as relative widths for the lines appeared as expected. We also tested our integration of relative widths and LineCollection by creating a LineCollection and passing in relative line widths as parameters and comparing the resulting figure to a baseline image.

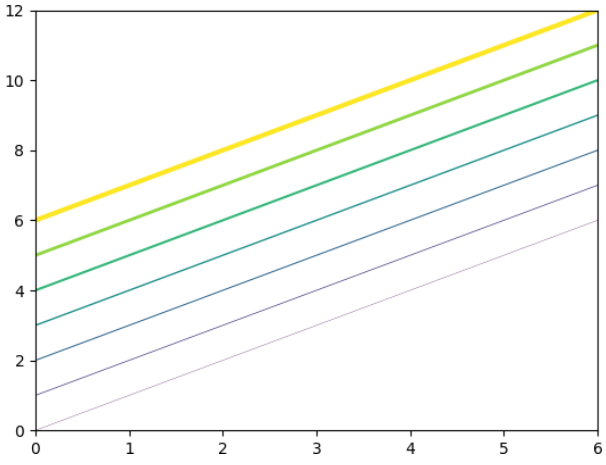
The new tests are located in the fork and also in the group repository under ***GoonSquad/D5/relative_linewidth_markersize_values/test/***. The tests were added to matplotlib's existing ***tests/baseline_images/test_lines.py*** because that's where their line and marker tests are.



marker_sizes.png



line_widths.py



line_collection_widths.py

Integration:

These changes fit well into the existing code base. Since the involved classes (**Line2D, Collection...**) call ***set_linewidth*** and/or ***set_markersize,*** rather than explicitly setting the value, there is no chance for a string value to cause an issue because it's converted into a useable float before anything else in the code base could have the chance to use it in an invalid state. In addition, they follow an existing naming convention (similar to font sizes in ***font_manager.py***).

Documentation:

Please refer to
relative_linewidth_markersize_values/doc/relative_linewidth_markersize_values_demo.html for a Matplotlib style documentation of the feature.

Feature 4: Scatter plot Contour

Feature Request: *Own feature*

Estimated Hours:

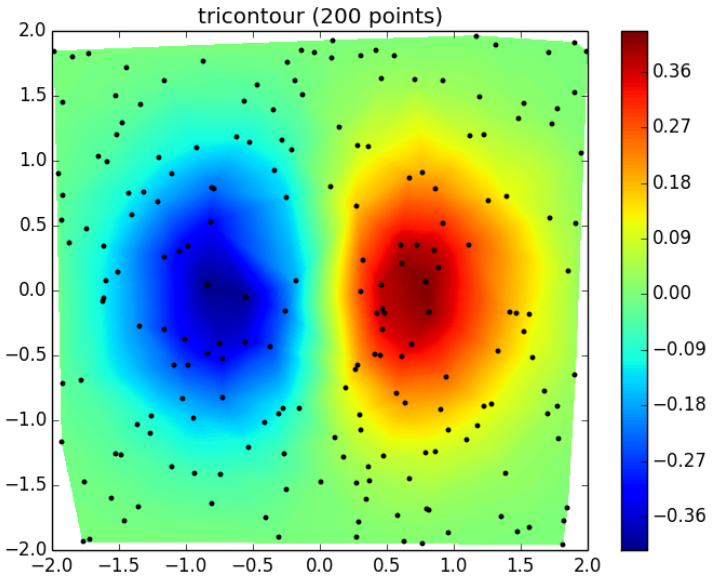
- Explore and create a solution (11 h)
- Implement solution (13.5 h)
- Testing/validation (4 h)
- Code Review (3 h)
- Documentation (5 h)

Description:

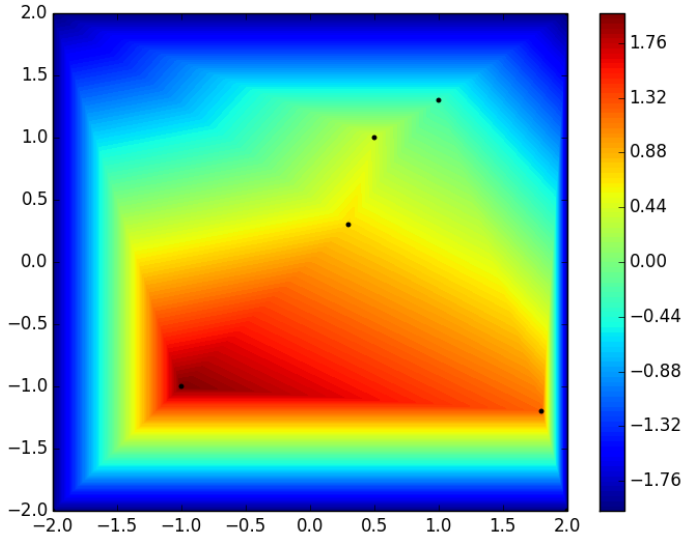
An easy to use, out of the box solution to create a color contour using a scatter plot without the user having to interpolate the data into the proper grid format

Solution:

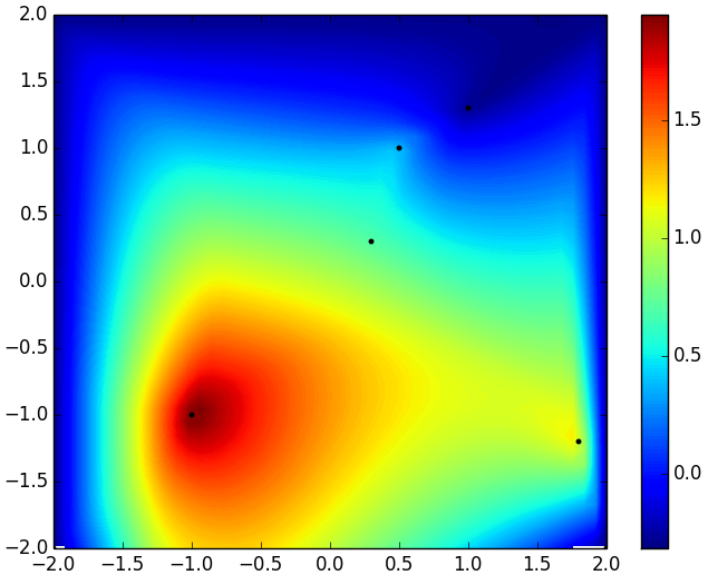
Our first idea when approaching this task was to use a linear interpolation of points to obtain a grid. However, while researching, we came across a method of interpolation using triangulation and tricontour that accomplished something similar to our desired results. Using this method, we came up with the following graph:



This solution seemed good, however, we realized that the triangulation method used created an undesirable sharpness to the contour when data density was low, as seen in the following graph:

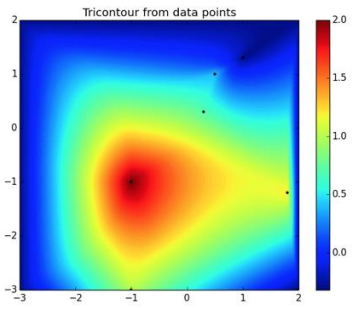


We then went back to a linear interpolation solution using contour and filling in the edge coordinates with default values to create a smooth contour:

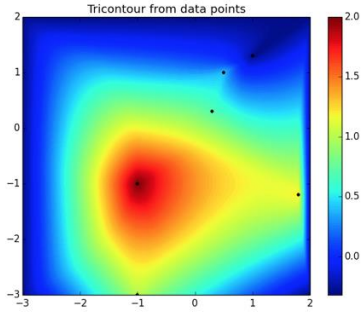


Testing:

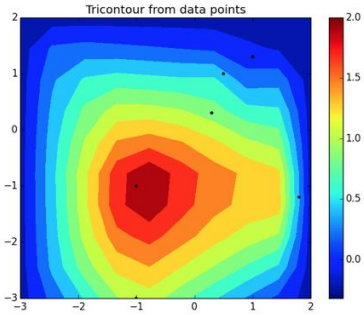
To test our feature, we insured that all the functionality was working as it is supposed to. We first tested the main functionalities by using sample data without setting any optional values. We then test each and every optional parameter to ensure either the default value was getting triggered or the value provided by the user. Below are the results of our test cases.



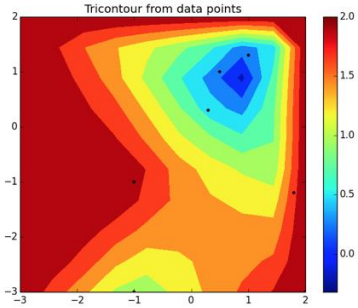
test_all_defaults.png



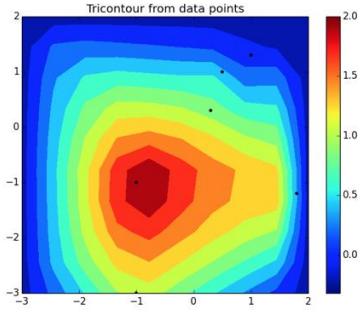
test_circular.png



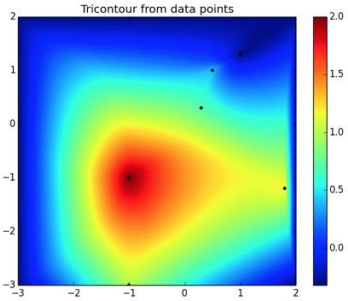
test_colour_density.png



test_edge_max.png



test_inputted_edge.png



test_triangular.png

Task Boards

Below are links to the task boards that we've used to help us keep track of our hours spent implementing the features and the details of the implementation of each feature.

D5 Overview

Contour Label Extensions

Axes Default Title Location

Relative Linewidth and Markersize

Time Cards (tracks all the hours we've spent on this deliverable)

These task boards were features that we started on but decided to implement another feature along the way.

Legend Title Font Size Param