PROJECT DELIVERABLE 4

Team 4 (Sky Blue)

CSCD01

March 14, 2016

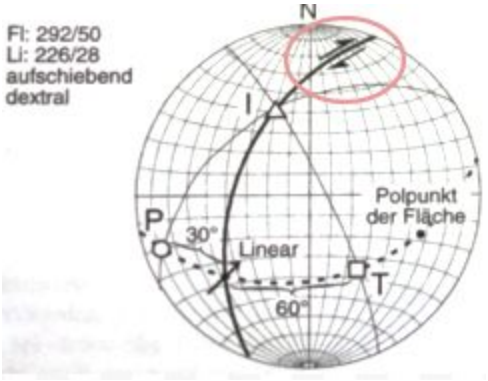
Team Members:
Chun Cho
Richard Luo
Yuxuan Hu
Geoffrey Hong
Kai Lin

Table of Contents

Feature 1 - Shear Arrows

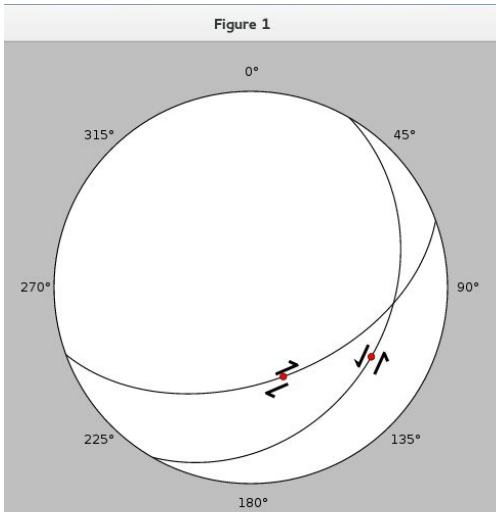
Title: Easier way to plot shear arrows (or half arrows) #5095
Link: https://github.com/matplotlib/matplotlib/issues/5095
Issue Description:
In order to do proper structural geology plots, users need shear arrows (half arrows) to indicate sinistral and dextral movements along a plane. Currently, there is **no way** to draw shear arrows in matplotlib.

Example of how the shear arrows should appear on a Figure (provided by the issue author):



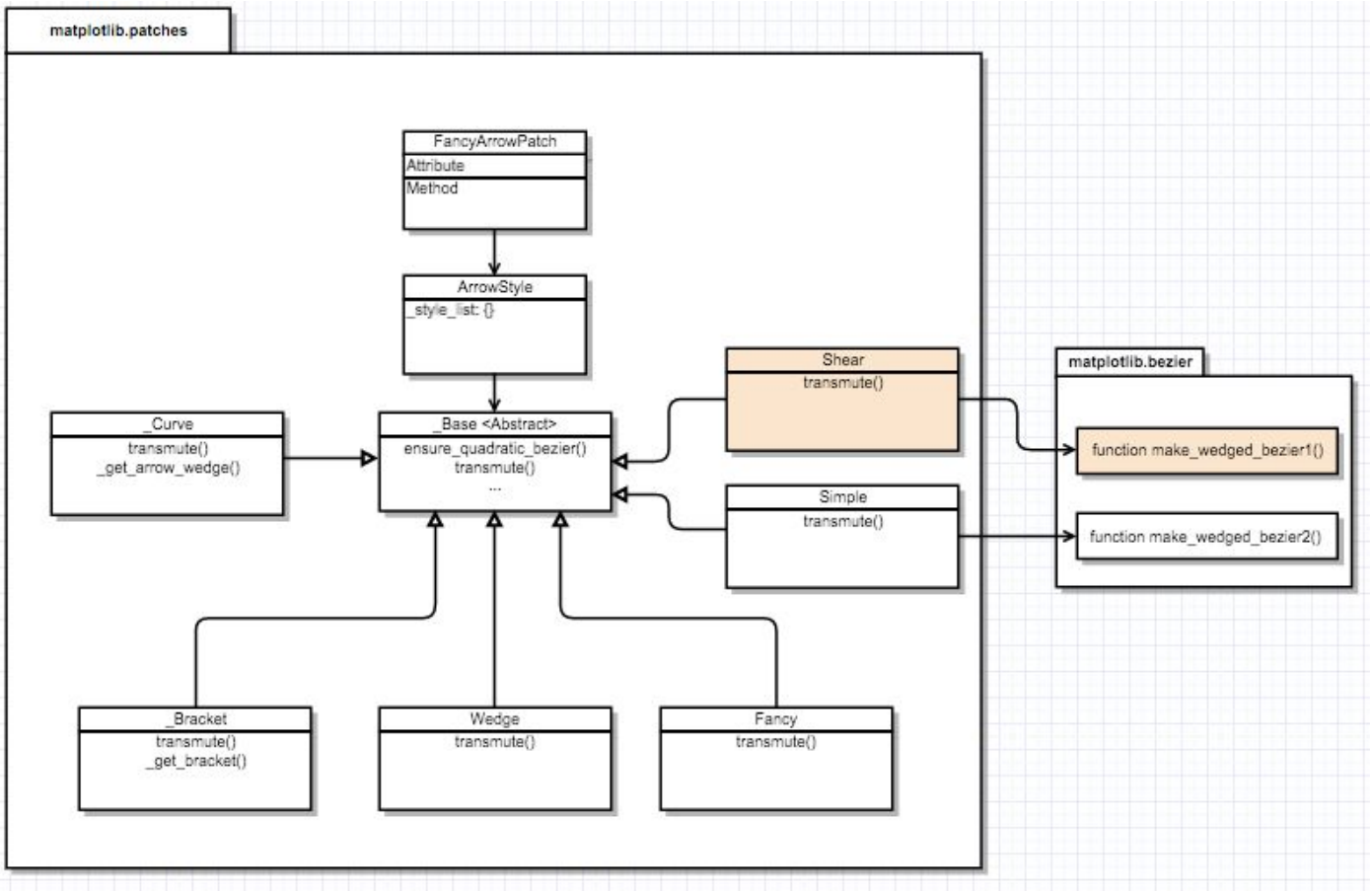
Users of matplotlib currently use a tedious work around, which involves drawing “1”s to represent shear arrows on the figure. This work around requires many lines of code, which is inefficient and messy. The issue author asked for the shear arrows to be officially implemented as a feature.

Example of the work around on matplotlib:



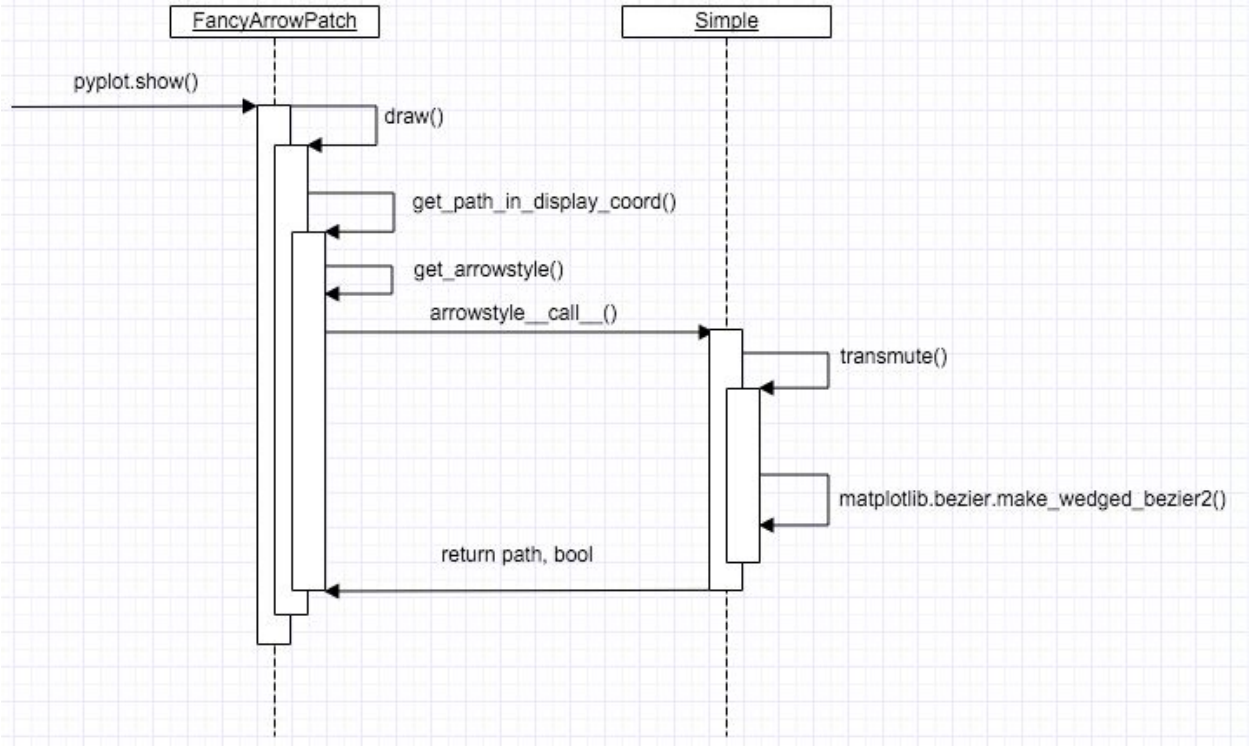
Reason for selecting this feature:
This issue was chosen over the other issues because this is a feature that would be helpful to many users and adding the code required to implement this feature would have no effect on pre-existing features. There is low risk for this feature as there will be no modify to existing code, only extension.

UML Diagram:
Produce designs for the selected features, describing your plans for the organization of new code, as well as all interactions between new code and existing code. UML diagrams would be very helpful here

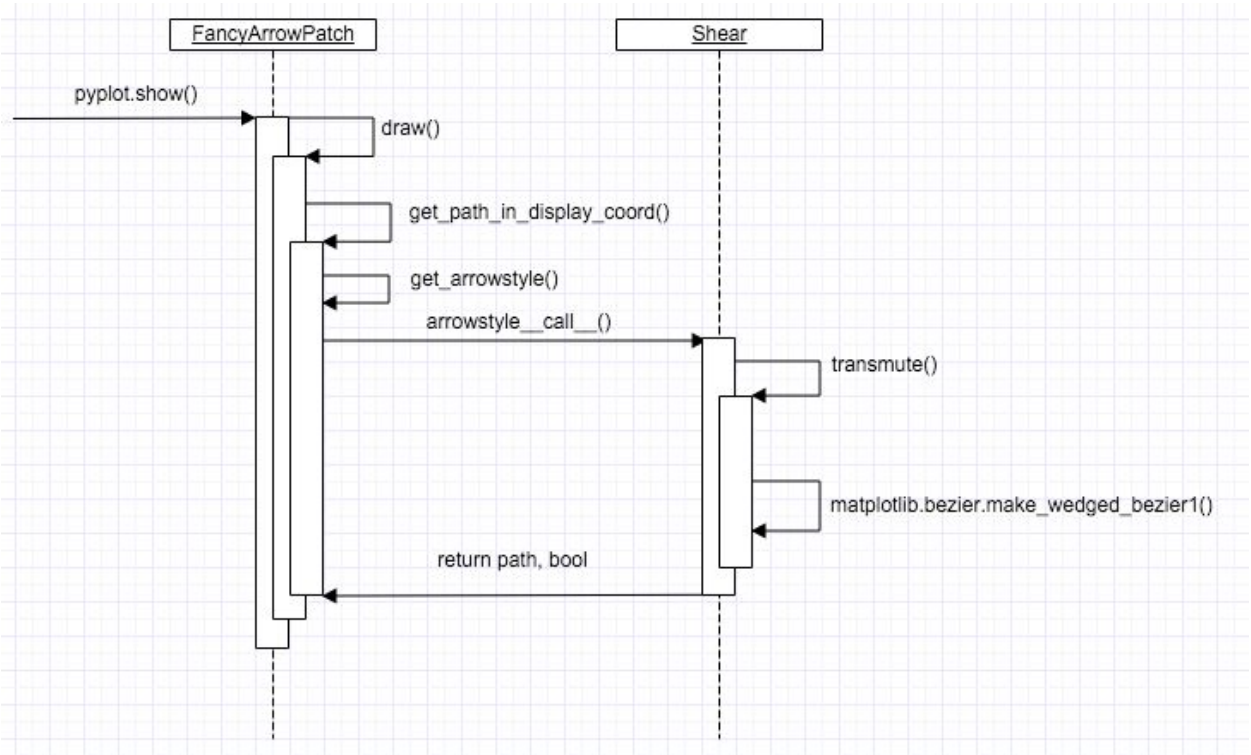


Shear and make_wedged_bazier1() are newly created class and function. The creation of Shear is similar to other arrows (Wedge, fancy, Simple, etc) such that it inherits _Base class which is used by ArrowStyle. bazier.py contains multiple function declarations, including make_wedged_bazier1(), which will be used to help draw the shear arrow.

Sequence Diagram of existing code used to create a simple arrow:



Sequence Diagram of new feature's code to create a shear arrow:



Implementation Plan:
Parts of existing code base which will be modified:

- matplotlib/lib/matplotlib/patches.py
- matplotlib/lib/matplotlib/bezier.py

From analysis, the code used to draw arrows in matplotlib is found in patches.py. In this file, there is a container class called Arrowstyle, which contains multiple different arrow style classes by using a dictionary. These classes define specifications to draw different arrows (ex. Simple arrow, Fancy arrow). They all extend the _Base class.

The Arrowstyle instances require the use of an Bezier's methods ("make_wedge_bezier2()" for Simple arrow instance for example) in order to draw the arrows.

From this analysis, it appears there currently does not exist the functionalities to create a shear arrow. To create a shear arrow, there is need for the creation of a new class which will define the specifications required to draw the arrow. This class will be called Shear. Shear will be extending _Base, as it contains similar functionalities and properties, and will be contained in the ArrowStyle class.

After creating the Shear class, there are two ways to define the specifications required to draw the Shear arrow.

Method 1:
Modify the existing make_wedge_bezier2() function to work with the new shear arrows. This modification will require the user to pass in an extra parameter to give specifications on the desired arrow (i.e. Simple arrow has 3 lines, Shear has 2 lines).

Method 2:
Create a new function named make_wedge_bezier1() to be used by shear arrow class. This modification will be similar to make_wedge_bezier2() but changed to provide a shear arrow rather than a normal arrow.

Chosen Method: Method 2
Reasoning:
Method 2 is chosen because it does not change any existing code that uses make_wedge_bezier2(). "Simple" arrow class for example, require the usage of make_wedge_bezier2() to draw the arrow. By modifying this function, modifications will

need to be made with the classes using this function, which goes against the SOLID principle (open for extension, closed for modification).

Acceptance Test:
Generate working examples of different types of figures with previously working normal arrows and newly created shear arrows. Generate images (.png) with such figures. For each test case, verify produced images to ensure they are as expected (if normal or shear, then it should appear on the image).

Cases:

- Line graphs with normal and shear arrows
- Geology graphs (globe figures) with normal and shear arrows
- Other types of associated graphs that can work with normal and shear arrows

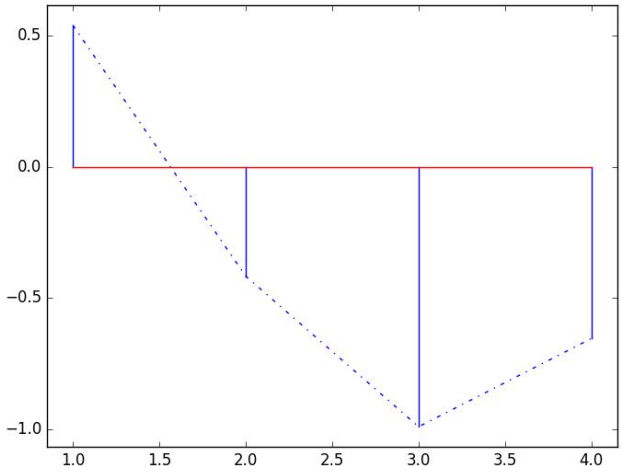
Feature 2 - Horizontal stem plot

Title: Horizontal stem plot #5856
Link: https://github.com/matplotlib/matplotlib/issues/5856

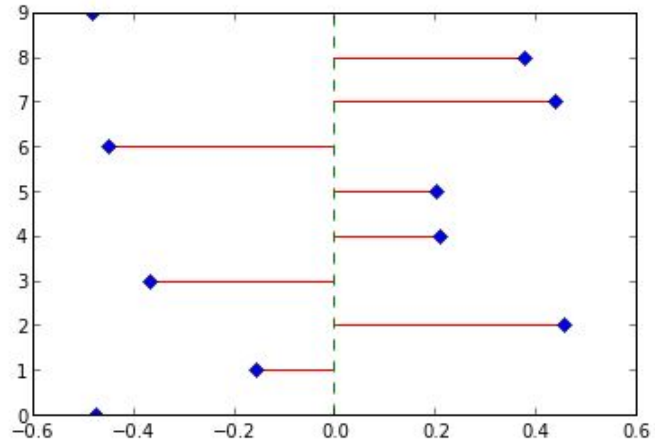
Issue Description:
The issue revolves around how matplotlib only allows the drawing of vertical stem bar with horizontal base line as shown below. However, there are no equivalent of horizontal stem bars at the moment. The requester indicated that he wanted a way to implement a horizontal version of stem bars using a keyword like:

orientation = {“vertical” | “horizontal”}

Example of vertical stem with horizontal baseline (Available in matplotlib):

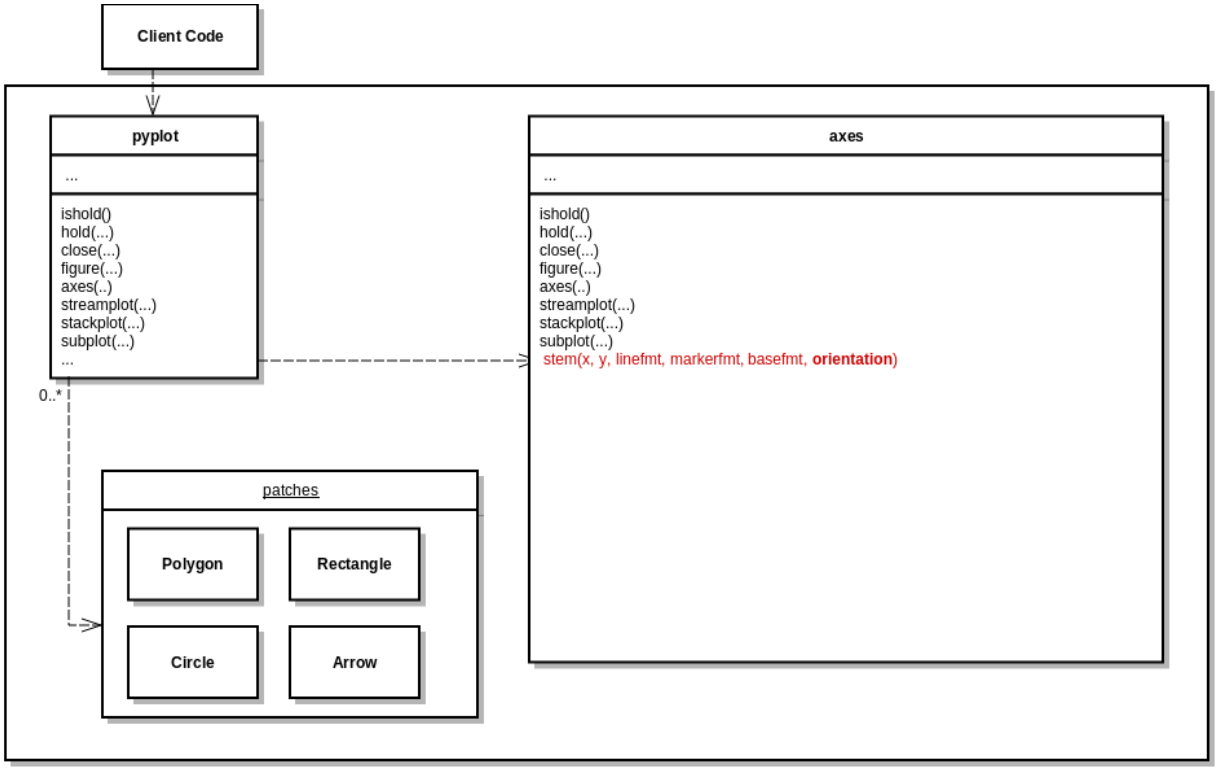


Example of horizontal stem with vertical baseline (To be implemented):



Reason for selecting this feature:
This issue was chosen over the other features because it is reasonable and is a feature that would benefit matplotlib and its user base. Unlike other features where it could be something entirely new, a vertical version of stem bar should come with the horizontal version as well. There are also helpful references that could be used to assist in the implementation of the feature.

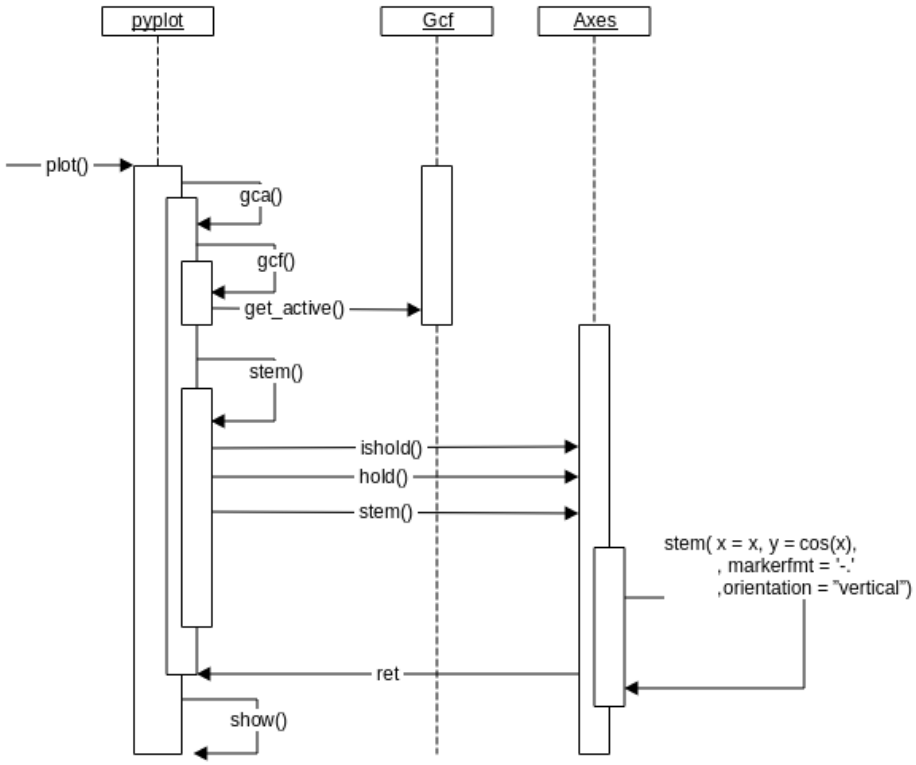
UML Diagram:



stem(x, y, linefmt='b-', markerfmt='bo', basefmt='r-', orientation = “vertical”)
stem(x, y, linefmt='b-', markerfmt='bo', basefmt='r-', orientation = “horizontal”)
stem(x, y, linefmt='b-', markerfmt='bo', basefmt='r-')
The red highlight indicate the change to are going to add to the *arg parameter.

The function **stem(...)** will be modified. An additional parameter type will be added, which will indicate whether the stem plot should have vertical or horizontal orientation.

Sequence Diagram:



An example of calling a vertical stem() by inserting the parameter “vertical”

Example Code:

```
#!/usr/bin/env python
from pylab import *

x = [1,2,3,4]
stem(x, cos(x), markerfmt = '-.', orientation = "vertical")

show()
```

Implementation Plan:

Parts of existing code base which will be modified:

- matplotlib/lib/matplotlib/axes/_axes.py

```
def stem(self, *args, **kwargs)
```

matplotlib/axes/_axes.py Line:2361 function stem

The current function does not support any horizontal stem. It only works for vertical line starting from x to y.

The plan for the implementation is to add a keyword to the function, such that the function still defaults to vertical stem. However, if the user wishes to use horizontal orientation for their stem plot, it must be specified.

def stem(self, *args, **kwargs):

Method 1: Augmented the function

```
def stem(self, *args, **kwargs, orientation= “vertical”)
```

By default, stem(...) uses vertical stem. If an user wants to make it a horizontal stem, they must pass in “horizontal” for the orientation argument.

Method 2: Passing a new parameter to parameter args

After this change, the function call would be:

```
stem(self, *args, **kwargs)
```

By specific the type in the last parameter of args and check which type of stem the user wants to use.

Chosen Method: Method 2

Reasoning:

The reasoning behind this choice is because it does not change any pre-existing API. The call to the function is stem(self, *args, **kwargs), which does not add any new parameters to the function.

Acceptance Test:

For user acceptance tests, regression tests will be run in order to ensure that the introduction of the feature does not break pre-existing functionality (Figure 2). Additionally, there will be a test to see if the horizontal stem with vertical baseline functions as expected (Figure 1). Please see below.

Figure 1: Horizontal stem with vertical baseline

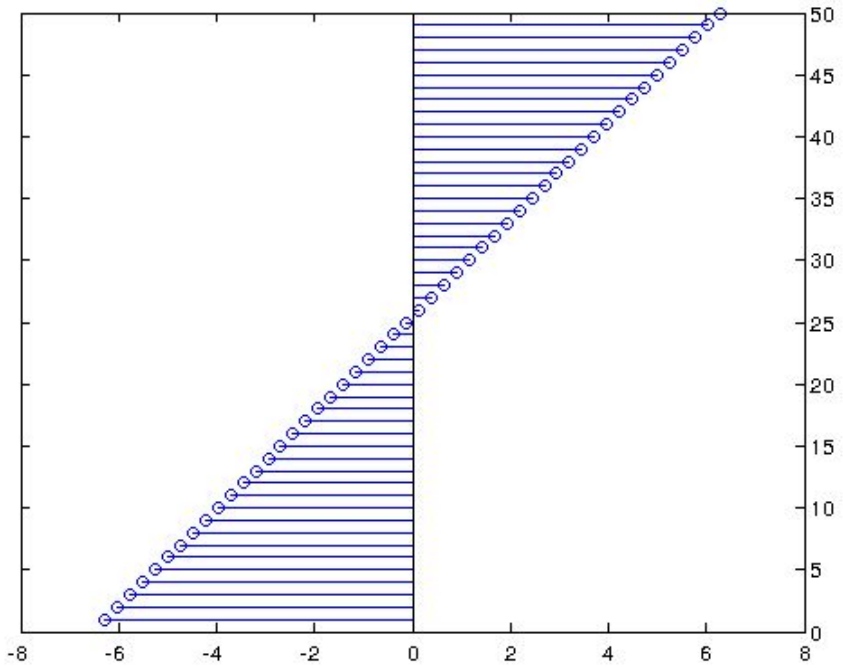
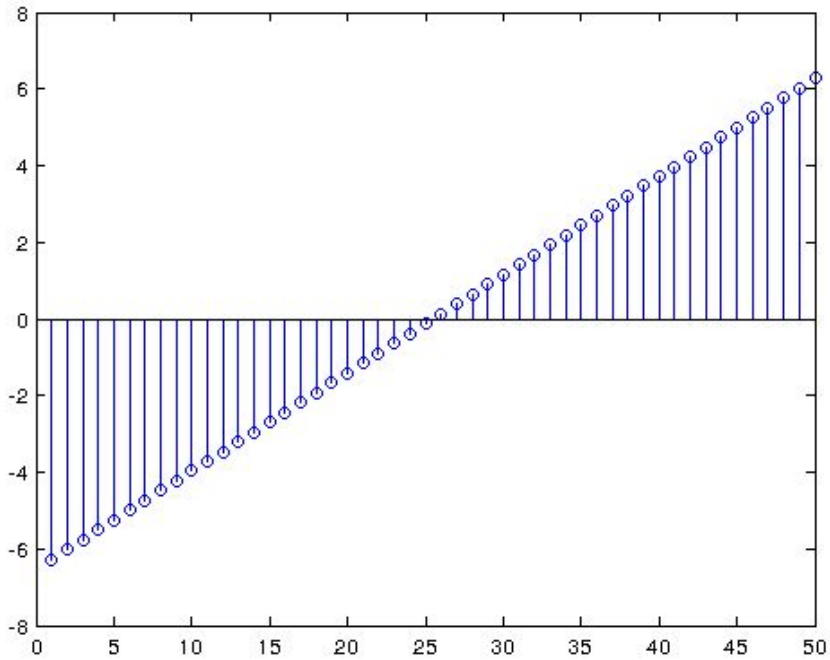


Figure 2: Vertical Stem with Horizontal baseline



Bonus Feature - Accepting Figure Arg.

Title: Accepting figure argument in subplot2grid #6105
Link: https://github.com/matplotlib/matplotlib/issues/6105
Description:
It is reasonable to accept figure augment in the subplot2grid. By default, fig=gcf().

Reason for selecting this feature:
Subplot2grid function started with fig=gcf(), and call the fig.add_subplot.
This augment allow user to put in different fig for different circumstances.

UML & Sequence diagram

- N/A because it's a bonus feature.
- However, there will be acceptance test for this feature to make sure it worked.

Implementation Plan:
Change the *def subplot2grid(shape, loc, rowspan=1, colspan=1, **kwargs):*

1. Add the parameter fig inside the **kwargs
2. Change the API call so that it takes another parameter of type fig

So the an example call would be :
*subplot2grid(shape, loc, rowspan=1, colspan=1, **fig=None**):*
And remove the line fig=gcf() in the function of subplot2grid(..)
After that, inside the function add:
If fig == None:
  fig = gcf();
This will produce the desired output.

Acceptance Test:
Run the regression test on subplot2grid, make sure it pass all the testcases for subplot2grid.
Try different kinds of fig to phrase in the function.

Decision of Which Feature to Implement

After much discussion, the team has decided to implement the second feature: “Horizontal Stem Plots”. The reasoning behind this choice is because the implementation of this feature is straightforward, and the feature would be useful for many users. Additionally, pre-existing code can be used as reference when implementing this feature, namely the Vertical Stem Plots. Also, we have decided Feature 1 - Shear arrows is potentially more difficult and risky in comparison to Feature 2, because it may require changes to the backend.

Due to the perceived simplicity of the implementation of this feature, the team has decided that in the event that the feature is implemented early, another feature will be queued for implementation. The next feature in line for implementation is the bonus feature and then Feature 1: Shear arrows.