Deliverable 4: Implementing Feature

Benjamin's Grooper

CSCD01

Anya Tafliovich

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

Winter 2019

Table of Contents

Feature Description

https://github.com/matplotlib/matplotlib/issues/13575

Using the façade design pattern, we created a modified version of the requested feature described at issue #13575. This feature allows the user to create a customized 3d plot by only using one function with a few optional arguments. For instance, the user would have the ability of plotting a 3d line using a string representation of it's equation and highlighting/plotting specific points on that line with projections from each axis, where these projections can have different colours, be dotted/solid, and where each of these projection are different depending on the specified point. The user can also use the functionality of plotting individual points and projections without giving a 3d line, or only plot a 3d line, which will have its own properties.

Implementation

Line/Point Plotting Facade:

The approach would be to create a new function in pyplot.py called something along the lines of "projection_plot". We decided to create this function in pyplot.py as pyplot.py is already being used as a Facade for a lot of matplotlibs features and already imports some of the classes we need for our implementation. Our function would take in as its parameters: points to plot, options for customizing projection lines, axes limits, and the optional parameters for a line equation to generate a line as well as options for customizing that line. Using methods from the classes Axes3D, Axes, and Figure, we will create the desired plot with the given parameters so that the user can simply call this function when they want to be able to clearly view points with projection lines. First we will generate the axes with arrow heads to model a typical 3D plot with X, Y, and Z labels. This will be done with the quiver method found in Axes3D. Then we will iterate over the collection of points and also iterate over the collection of the custom options for the projection lines to be used for these points to individually plot each point and projection line starting from axes. In the case that the user wishes to also plot a line, we will parse the line equation that is passed as a string to plot on our axes using the same method as before to customize the properties of this line.

We decided to split parsing the line equation, the main function (projection_plot) and plotting the projection lines into 3 different functions. We chose to do it this way because it is easier to develop code and the code is cleaner. If there

are any bugs in the future, the bug can be easily isolated and if there are future features that are built on top of these methods, they can be reused instead of having code being duplicated. We implemented the main function so that every parameter except the axes limits are optional. This is to promote ease of access, flexibility and to maintain the facade design pattern.

The main difference between our current design and our design in deliverable 3 is that we further separated our implementation into another function for plotting projection lines if the user wants these. We also decided to make all of the arguments passed to our main projection_plot function optional except for axes_limits to make it more user friendly and allow users to create their a plot which shows only what they wish to see.

Our implementation does not modify pre existing code other than importing Axes3D into pyplot.py.

UML Diagrams

Updated UML displaying interaction between pyplot and classes within the subsystem for plotting points/lines on 3D axes along with options. All parameters except the axes_limits are optional.