Matplotlib

Deliverable 4: Design And Plan of Implement Significant MatplotLib Bug

By: Team Legend (Team 4)

Matplotlib Feature 6569: Axis Swapping

Feature URL
https://github.com/matplotlib/matplotlib/issues/6569

Detailed Feature Description

The feature for issue #6569 is to develop an axis swapping API that allows users to swap the axes (ie. using the y-axis as the x-axis).

In the current version of matplotlib, in order to permute the axes, the user would need to pass in an “orientation” parameter to be either “vertical” or “horizontal” (Some functions have set orientation = ‘horizontal’ as default and other have set it default to ‘vertical’). The “orientation” parameter are present in many functions such as `hist(...)` and `eventplot(...)` in class Axes where each function has implemented both cases when orientation is horizontal and vertical.

The development of an axis swapping API would allow users to swap the axes, thus they are no longer required to insert the desired orientation in parameters of various functions.

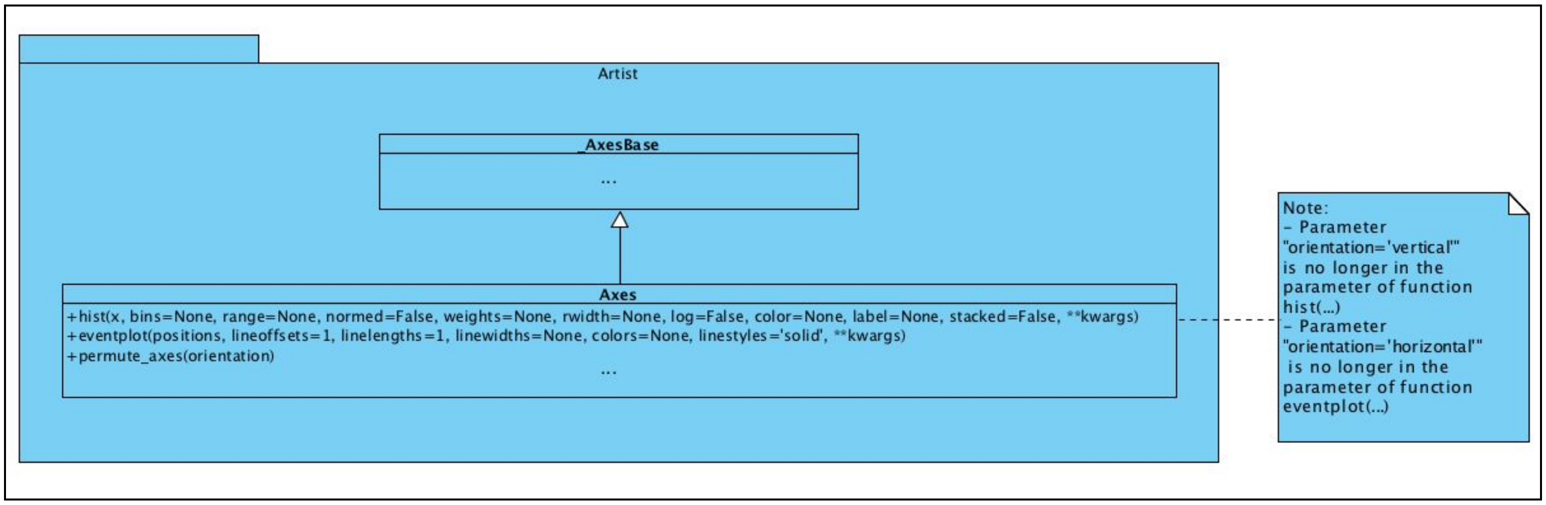
Section of Modified Code Base

The part of the code base that will be modified comes from the `_axes.py` file. In this file we will create a new function called `permute_axes(orientation)` in the Axes class that will take the x-axis and use it as the y-axis and take the y-axis and use it as the x-axis. By doing this, the orientation parameter in various functions (ex. `hist(...)` and `eventplot(...)`), which are used to swap the axes, now are no longer needed. Thus the parameters and content of these functions need to be updated.

- `hist(...)` is located in line 5913 in _axes.py
- `eventplot(...)` is located in line 1090 in _axes.py

Implementation Plan

For methods such as `hist(...)` and `eventplot(...)`, the orientation parameter would not be needed. Thus the section of code that reflect the choice of this parameter can be removed. They can be just set to vertical or horizontal as default. When the user wants to change the orientation, they can then call our axis swapping API `permute_axes(orientation)` with their desired orientation. The UML diagram below demonstrates the design.


Artist
_AxesBase
...
Axes
+hist(x, bins=None, range=None, normed=False, weights=None, rwidth=None, log=False, color=None, label=None, stacked=False, **kwargs)
+eventplot(positions, lineoffsets=1, linelengths=1, linewidths=None, colors=None, linestyles='solid', **kwargs)
+permute_axes(orientation)
...
Note:
- Parameter "orientation='vertical'" is no longer in the parameter of function hist(...)
- Parameter "orientation='horizontal'" is no longer in the parameter of function eventplot(...)

Matplotlib Feature 8283: Custom Ordering of Legend

Feature URL

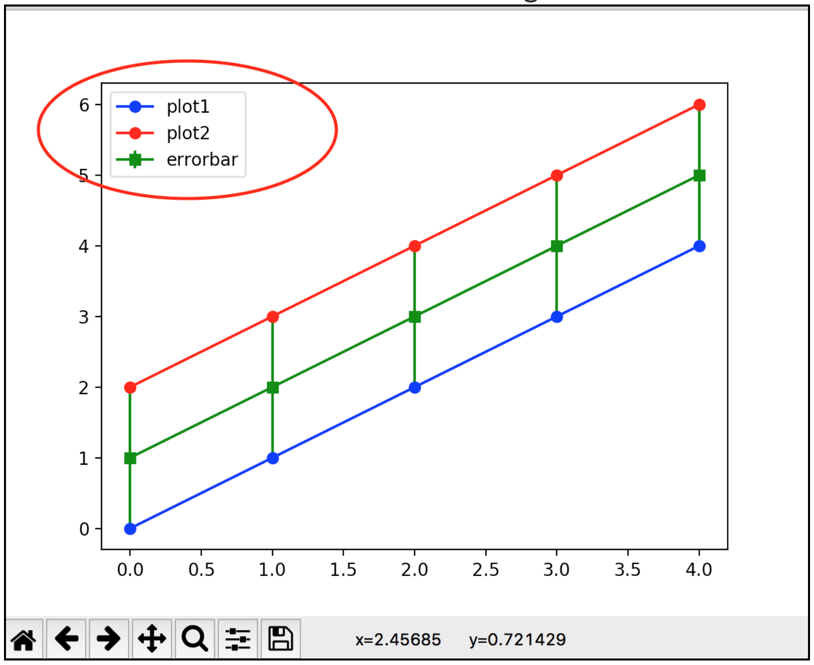
https://github.com/matplotlib/matplotlib/issues/8283

Detailed Feature Description

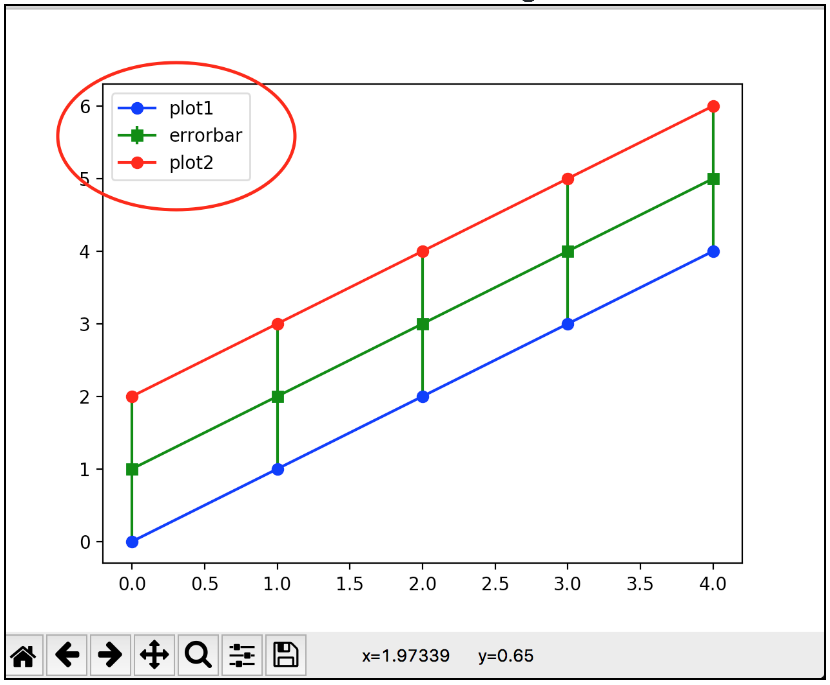
The feature requests to have an ability to customize ordering of labels of the legend, when using multiple artists on the same axis. Currently the default order in the legend is based on the object type (lines first, patches, collections and containers last) followed by when it was created. Often users are want to have the legend in a particular order. Currently there are several tedious ways to achieve this behaviour. The first is to use the axes method `get_legend_handles_labels()` which returns handles and labels for the legend. Then handles can be use to reorder the labels. This can be time consuming because one has to manually figure out the order of the current layout and repack the custom order. Below is shown a code snippet explaining this recommendation.

```
import matplotlib.pyplot as plt
import numpy as np
fig,ax = plt.subplots(1)

ax.plot(np.arange(5),np.arange(5),'bo-',label='plot1')
ax.errorbar(np.arange(5),np.arange(1,6),yerr=1,marker='s',color='g',label='errorbar')
ax.plot(np.arange(5),np.arange(2,7),'ro-',label='plot2')

handles,labels = ax.get_legend_handles_labels()
handles = [handles[0], handles[2], handles[1]]
labels = [labels[0], labels[2], labels[1]]

ax.legend(handles,labels,loc=2)
plt.show()
```

Before Reordering



After Reordering



Second **workaround** is to store the return value of `pyplot.plot(...)` and pass two tuple to legend method one being the stored values and other being the corresponding labels. Example below will have the same result as the images shown above.

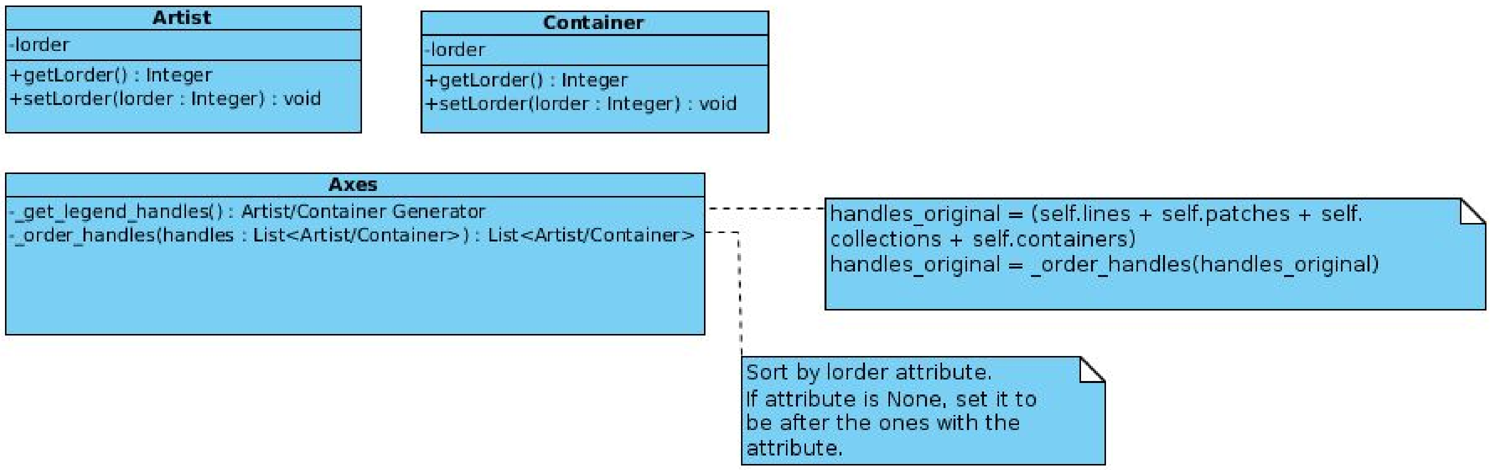
```
import matplotlib.pyplot as plt
import numpy as np
fig,ax = plt.subplots(1)

art1 = ax.plot(np.arange(5),np.arange(5),'bo-',label='plot1')[0]
art2= ax.errorbar(np.arange(5),np.arange(1,6),yerr=1,marker='s',color='g',label='eb')
art3 = ax.plot(np.arange(5),np.arange(2,7),'ro-',label='plot2')[0]

plt.legend((art1, art2, art3), ("label1", "label2", "label3"))
plt.show()
```

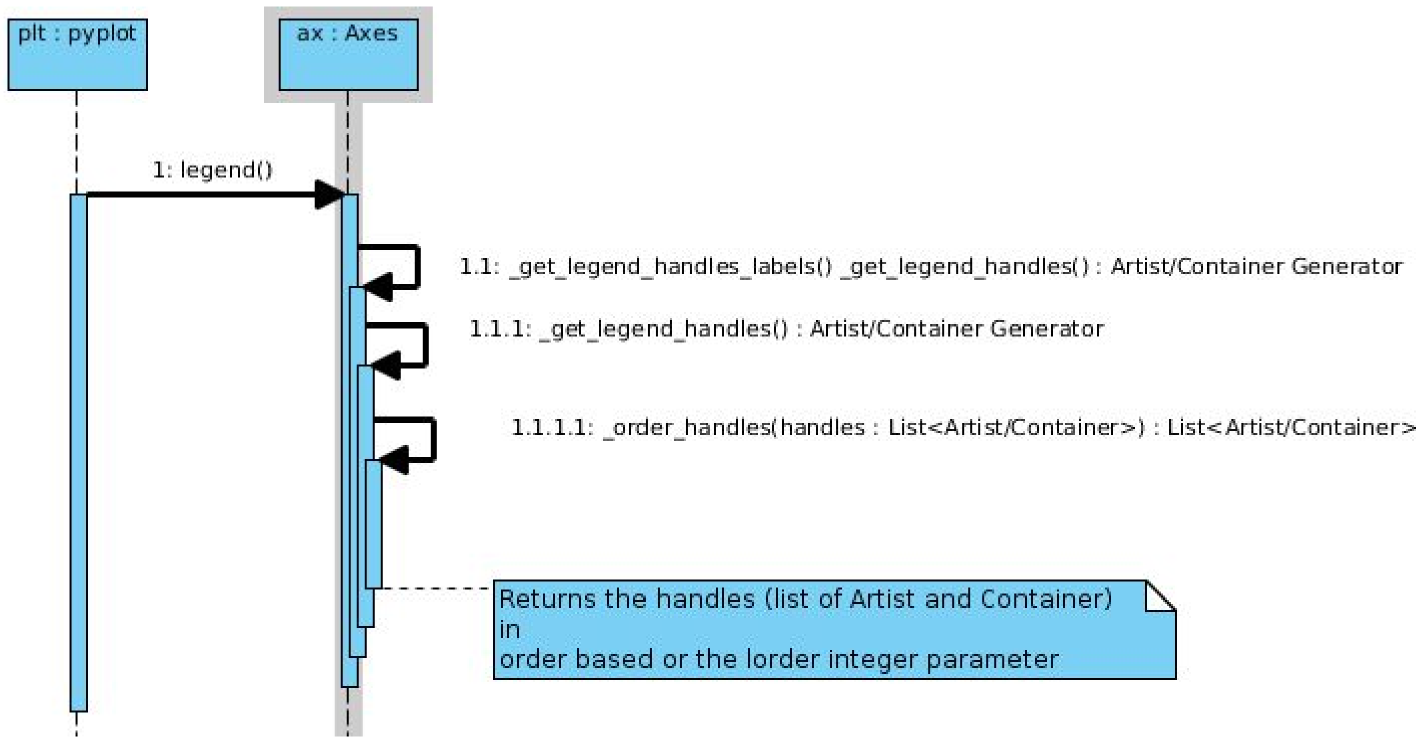
It would be more convenient to have an additional parameter called "`lorder`" that is used to assign priority for the labels shown in the legend. This will make matplotlib easier to use the for user.

Implementation Plan

To implement this feature, a new parameter "`lorder`"needs to be added to Artist and Container object. This parameter will determine the order of the objects on the plot if legend() is called without any parameters..

Next, the method _get_legend_handles in Axes needs to be modified such that it sorts the handlers (which is a list of Artist and Containers) based on the "`lorder`" parameter.


Artist
-lorder
+getLorder() : Integer
+setLorder(lorder : Integer) : void
Container
-lorder
+getLorder() : Integer
+setLorder(lorder : Integer) : void
Axes
-_get_legend_handles() : Artist/Container Generator
-_order_handles(handles : List<Artist/Container>) : List<Artist/Container>
handles_original = (self.lines + self.patches + self.
collections + self.containers)
handles_original = _order_handles(handles_original)
Sort by lorder attribute.
If attribute is None, set it to
be after the ones with the
attribute.


plt : pyplot
ax : Axes
1: legend()
1.1: _get_legend_handles_labels() _get_legend_handles() : Artist/Container Generator
1.1.1: _get_legend_handles() : Artist/Container Generator
1.1.1.1: _order_handles(handles : List<Artist/Container>) : List<Artist/Container>
Returns the handles (list of Artist and Container) in order based or the lorder integer parameter

In terms of how it will be ordered given `lorder`:

1. If two handlers have the same `lorder`, arbitrary choose one ahead of the other.
2. If handlers does not have lorder set, place them after the ones that do have the lorder parameter set. The handlers that lack loder will follow the predetermined order by type and time of creation as before the implementation of this feature.

Reasons to Implement

We decided to implement issue #8283 because it is a frequently demanded requirement from the users. Since Matplotlib is heavily used for scientific publication, having an easy ability to customize the legend is a must. Furthermore, compared to issue #6569, many people have shown interest in the feature in the Github issue comments thread. Thus, it will be a good contribution to the MPL community. Lastly, implementing it will improve our understanding working with large and open source system because it requires good understanding of the existing code.

Acceptance and Unit Tests

The acceptance tests are located at Deliverable 4/Issue8283_acceptance_tests.txt

There is already an extensive unit test suite of for _axes.py in the file test_axes.py. Therefore we have simply added additional tests cases to the end of test_axes.py file starting on line 4947, and added the baseline image legend_lorder.png to the tests/baseline_images/test_axes. This way we can both perform regression testing using the existing test suite and test the new functionality of our added keyword argument `lorder` in one place, as well as being able to have the tests automatically run when test.py is run. Adding the `lorder` parameter to Artist should not effect/change Artist behaviour in any way except when Axes.legend() (wrapped by pyplot.legend()) and the associated helper functions/methods are called, therefore we can check that the addition of the `lorder` parameter to Artist does not cause unwanted behaviour by running test_artist.py which essentially regression tests Artist .