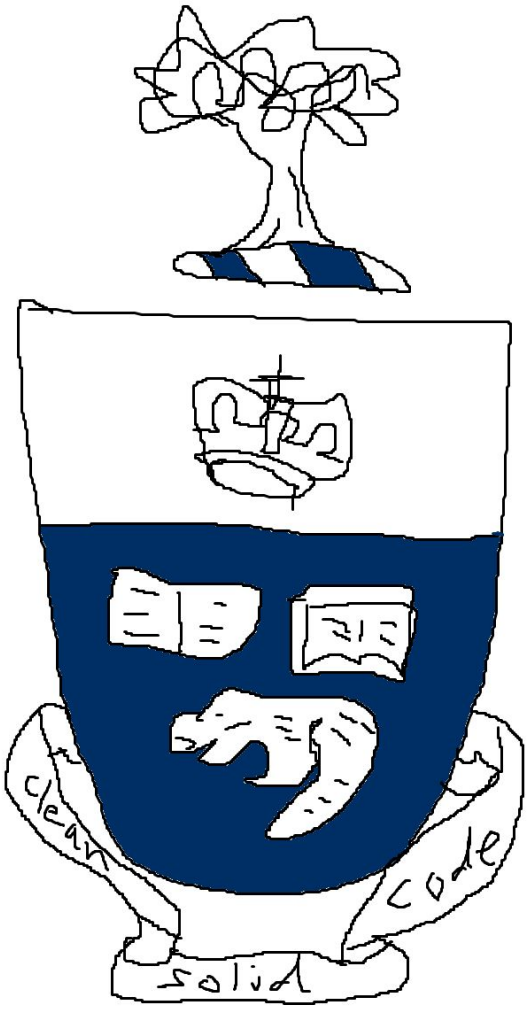


Deliverable 2p2: Features and Bug

Deliverable #2p1: Raising Issues
March 4th 2019
Team Members: Minqi Wang, Shuang Wu,
James Nicol, Xinrui Tong, Zixing Gong

Table of Contents

Feature 1 - Matplotlib Keymap Focus

Issue #: 13484
URL: https://github.com/matplotlib/matplotlib/issues/13484
Type of Issue: Feature Request
Short Explanation:
When using the example on embedding matplotlib in tk in the link below (Code Snippet section), if the user presses the tab key, the focus is shifted to the next widget, disabling keyboard interactivity with the mpl plot until tab is pressed enough times to shift the focus back onto the plot. Note that clicking on the plot will not restore the keyboard interactivity, which is what the feature request is for: to change mpl to set the focus to the clicked widget when using mpl with tkinter (and other applicable 3rd party GUIs)
Solution:
Added a function _tk_backend.py would call after left mouse click to reset focus back to plot canvas to allow for keymap to resume expected function. This feature/bug fix is related to the way tkinter handles 'tab' key input. Since no customize 'tab' input key is defined it uses the default 'tab' function and keymaps weren't functioning as intended.
Solution Code Snippet and Explanation:

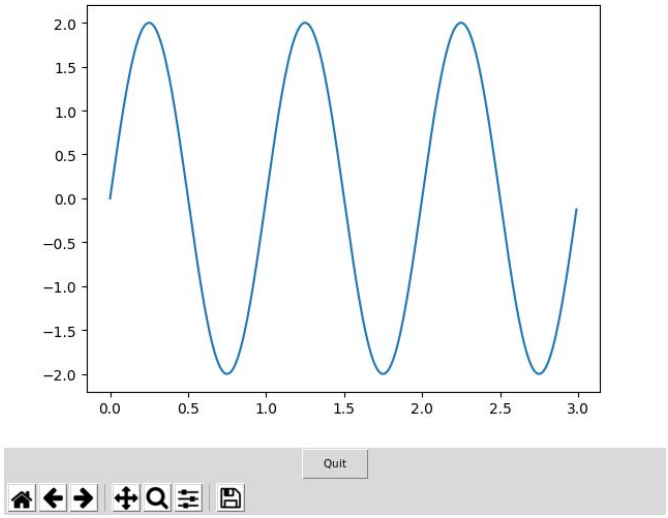
```
def __init__(self, figure, master=None, resize_callback=None):
    super(FigureCanvasTk, self).__init__(figure)
    self._idle = True
    self._idle_callback = None
    t1, t2, w, h = self.figure.bbox.bounds
    w, h = int(w), int(h)
    self._tkcanvas = tk.Canvas(
        master=master, background="white",
        width=w, height=h, borderwidth=0, highlightthickness=0)
    self._tkphoto = tk.PhotoImage(
        master=self._tkcanvas, width=w, height=h)
    self._tkcanvas.create_image(w//2, h//2, image=self._tkphoto)
    self._resize_callback = resize_callback
    self._tkcanvas.bind("<Configure>", self.resize)
    self._tkcanvas.bind("<Key>", self.key_press)
    self._tkcanvas.bind("<Motion>", self.motion_notify_event)
    self._tkcanvas.bind("<Enter>", self.enter_notify_event)
    self._tkcanvas.bind("<Leave>", self.leave_notify_event)
    self._tkcanvas.bind("<KeyRelease>", self.key_release)
    self._tkcanvas.bind("<Button-1>", self.button_press_focus)
    for name in "<Button-2>", "<Button-3>":
        self._tkcanvas.bind(name, self.button_press_event)
    for name in "<Double-Button-1>", "<Double-Button-2>", "<Double-Button-3>":
        self._tkcanvas.bind(name, self.button_dblclick_event)
    for name in "<ButtonRelease-1>", "<ButtonRelease-2>", "<ButtonRelease-3>":
        self._tkcanvas.bind(name, self.button_release_event)
                                                          192,5         18%
```

Refactored and added a bind for left mouse click detection within plot canvas based on other binds

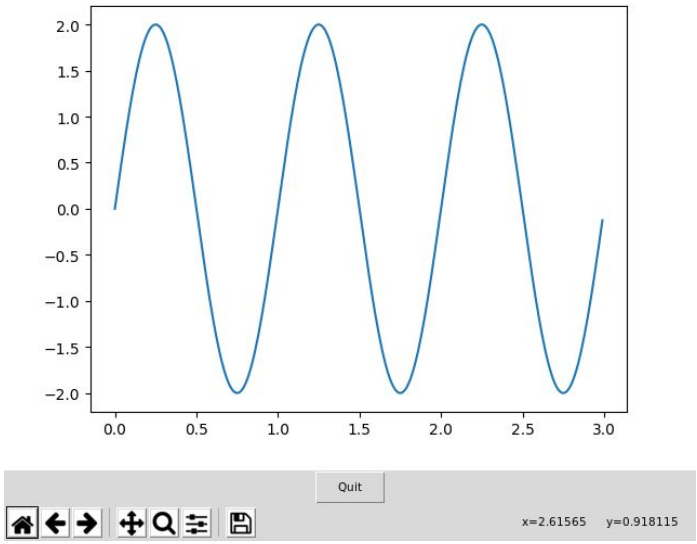
```
def button_press_focus(self, event, dblclick=False):
    self.get_tk_widget().focus_force()
    self.button_press_event(event, dblclick)
```

Added a function that is called by left mouse click (single click) to set focus back to plot and to ensure keymap is again functional.

The above changes follow the previous structure of _tk_backend.py without doing anything too novel. The button_press_focus function utilizes calling button_press_event to reuse working implementation for left mouse click before refactor.
Old Code Without Fix:
https://matplotlib.org/gallery/user_interfaces/embedding_in_tk_sgskip.html



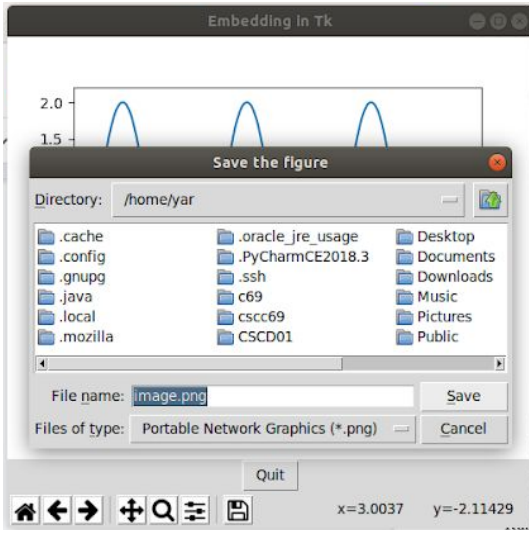
Focus is on the plot

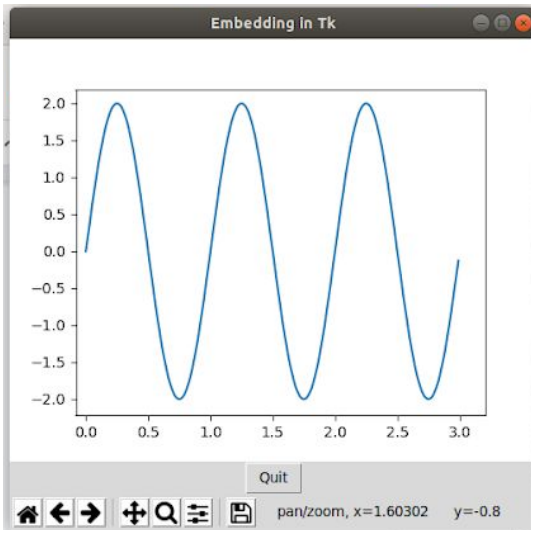


Focus is on the home button and keymap (e.g. button ‘s’) unfunctional.

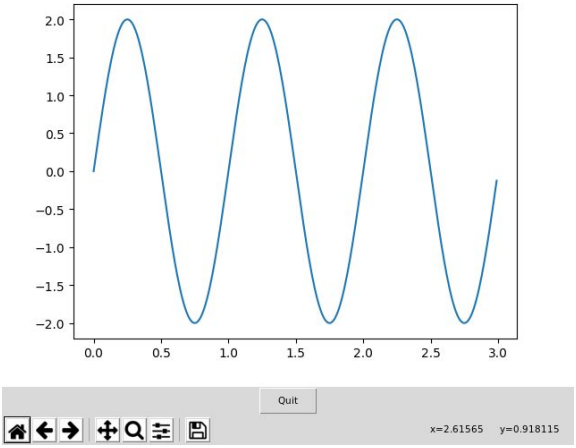
Testing:

https://matplotlib.org/gallery/user_interfaces/embedding_in_tk_sgskip.html

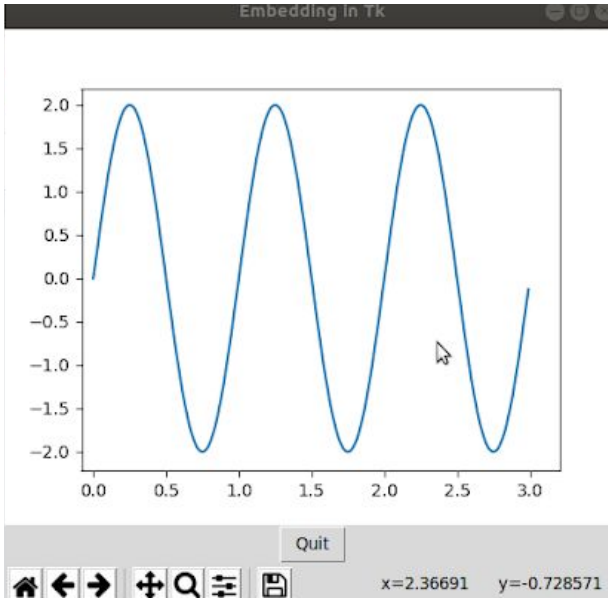




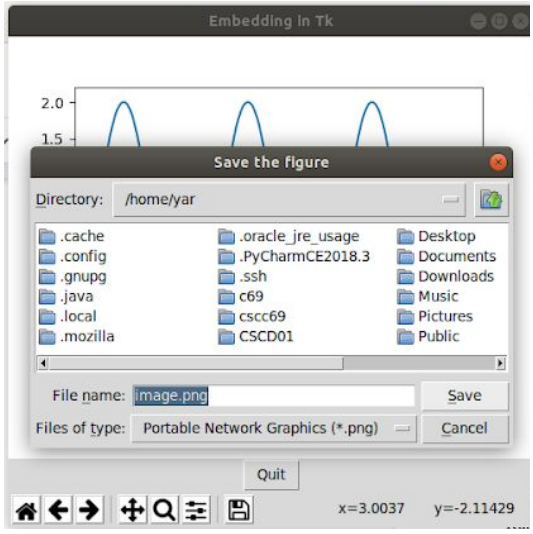
Running the code above should bring up sample plot with working keymaps. Use button ‘s’ on keyboard and ‘p’ to test keymaps are working. (Note the pan/zoom after pressing ‘p’)



After pressing the ‘tab’ key the focus is now on Home the home button. As expected keymaps currently do not work here.



After clicking left mouse click the focus should resume back to the plot and keymaps should work.



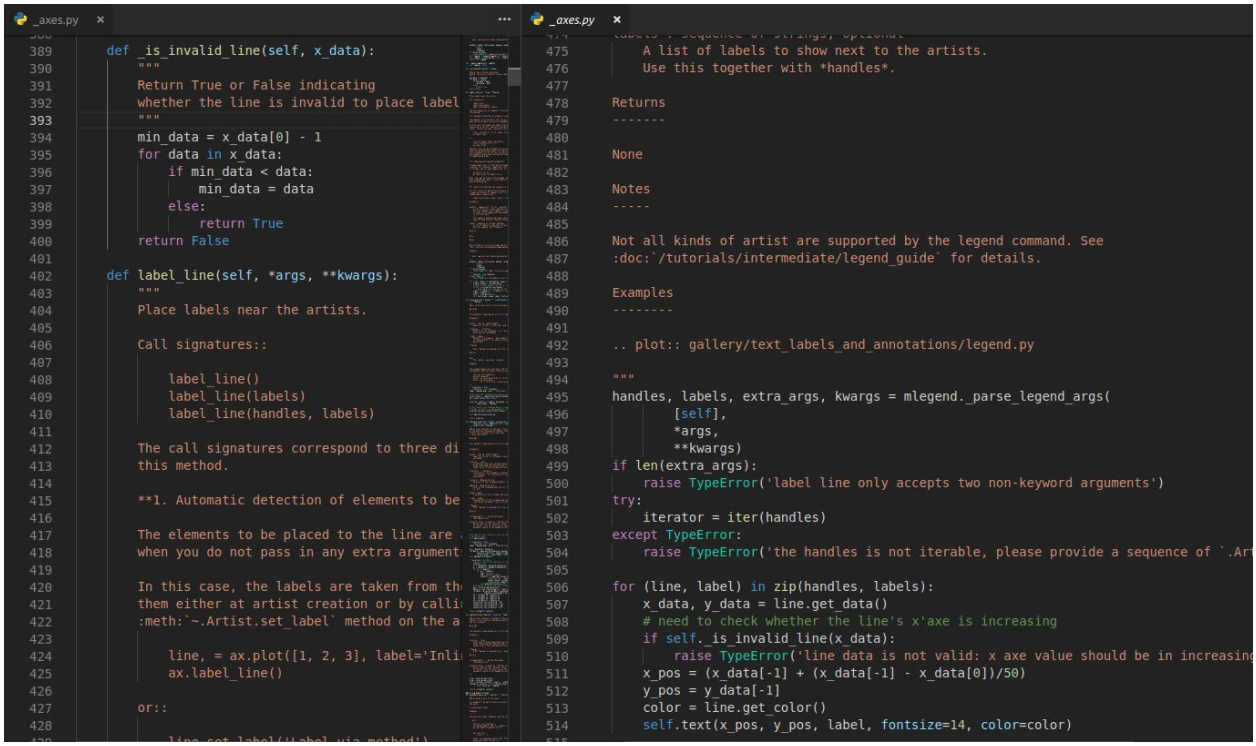
Pressing button ‘s’ works once again because keymaps are re-enabled once focus is resumed on canvas.

Feature 2 - Matplotlib label_line Feature Request

Issue Name: Option to place legend labels near to the data
Issue #: 12939
URL: https://github.com/matplotlib/matplotlib/issues/12939
Type of Issue: Feature Requested / Wishlist
Short Explanation:
The current way to add legend to label near to the line is complicated, OP would like to have a simpler API to accomplish this.
Idea of Solution:
This is actually not a bug but a feature requested from the OP. According to the comments below this opening issue, contributor jklymak had suggested implementing a new method instead of using .legend() API. Member of Matplotlib, QuLogic, provided a link to an example as a potential basis for the newly method: https://matplotlib.org/gallery/showcase/bachelors_degrees_by_gender.html. Therefore, we may use this as our starting point to implementing this feature.
Feature Concept:
Giving a new API, called label_line(), automatically put all the text labels to the end of the corresponding line, with same color to the line.
Solution Code Snippet and Explanation:
Brief Explanation: the solution is to find the lines end's x and y position as well as the color of the lines, and place the text labels using these information.

```
2673  def label_line(*args, **kwargs):
2674      return gca().label_line(*args, **kwargs)
```

User can call the new method API label_line (under pyplot.py), and label_line will do its job accordingly



```
_axes.py
389     def _is_invalid_line(self, x_data):
390         """
391         Return True or False indicating
392         whether the line is invalid to place label
393         """
394         min_data = x_data[0] - 1
395         for data in x_data:
396             if min_data < data:
397                 min_data = data
398             else:
399                 return True
400         return False
401
402     def label_line(self, *args, **kwargs):
403         """
404         Place labels near the artists.
405
406         Call signatures::
407
408             label_line()
409             label_line(labels)
410             label_line(handles, labels)
411
412         The call signatures correspond to three di
413         this method.
414
415         **1. Automatic detection of elements to be
416
417         The elements to be placed to the line are
418         when you do not pass in any extra argument
419
420         In this case, the labels are taken from th
421         them either at artist creation or by calli
422         :meth:`~.Artist.set_label` method on the a
423
424             line, = ax.plot([1, 2, 3], label='Inli
425             ax.label_line()
426
427         or::
428
```

```
_axes.py
475         A list of labels to show next to the artists.
476         Use this together with *handles*.
477
478     Returns
479     -------
480
481     None
482
483     Notes
484     -----
485
486     Not all kinds of artist are supported by the legend command. See
487     :doc:`/tutorials/intermediate/legend_guide` for details.
488
489     Examples
490     --------
491
492     .. plot:: gallery/text_labels_and_annotations/legend.py
493
494     """
495     handles, labels, extra_args, kwargs = mlegend._parse_legend_args(
496             [self],
497             *args,
498             **kwargs)
499     if len(extra_args):
500         raise TypeError('label line only accepts two non-keyword arguments')
501     try:
502         iterator = iter(handles)
503     except TypeError:
504         raise TypeError('the handles is not iterable, please provide a sequence of `.Ar
505
506     for (line, label) in zip(handles, labels):
507         x_data, y_data = line.get_data()
508         # need to check whether the line's x'axe is increasing
509         if self._is_invalid_line(x_data):
510             raise TypeError('line data is not valid: x axe value should be in increasing
511         x_pos = (x_data[-1] + (x_data[-1] - x_data[0])/50)
512         y_pos = y_data[-1]
513         color = line.get_color()
514         self.text(x_pos, y_pos, label, fontsize=14, color=color)
```

In this solution code snippet, the label_line() method in Class Axes perform the main operation to place the text labels near to the corresponding lines.

Note: the method _is_invalid_line() is to check whether the line has an increasing order on the value of x-data.

Test Cases for this implementation:

- Test1:

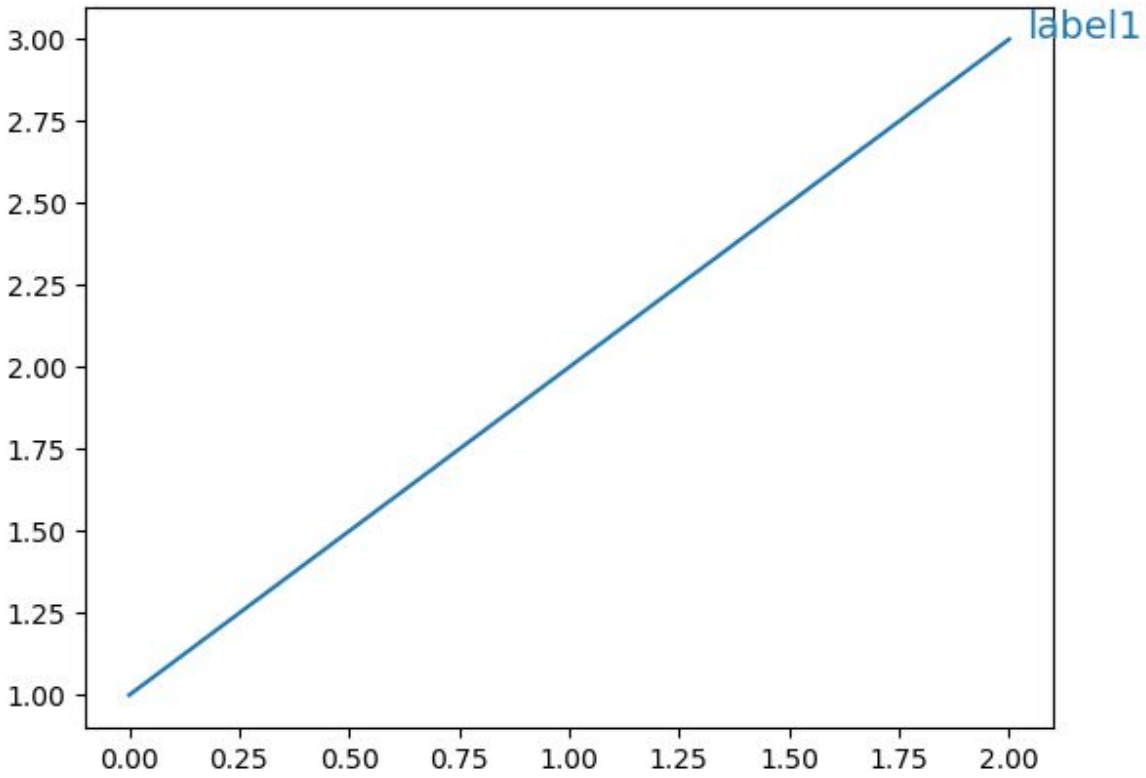
  Description:

  User can let label_line method automatically handle the label for you as long as the label is set in the line.

  Code Snippet:

```
23  import matplotlib.pyplot as plt
24
25  line1, = plt.plot([1,2,3], label="label1", color="#1f77b4")
26
27  plt.label_line()
28  plt.show()
```

  Result:



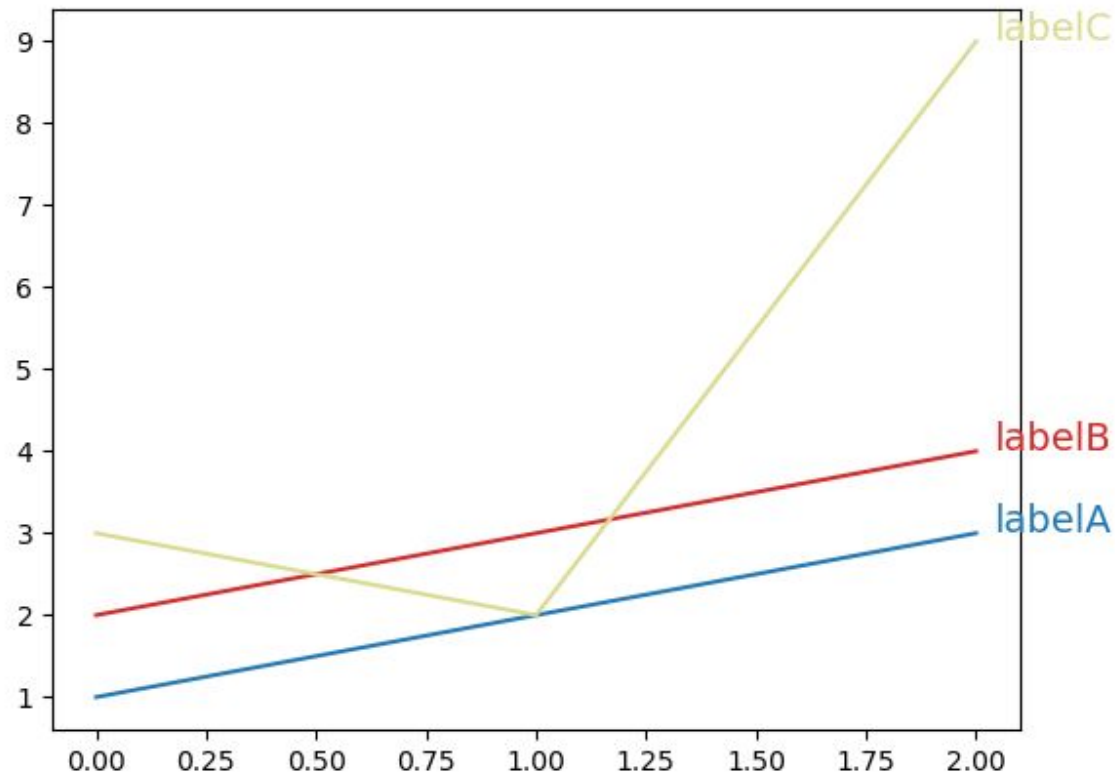
- Test2:
  Description:
  User can implicitly assign labels to lines, this could easily mismatch with the lines, so this approach is not recommended
  Code Snippet:

```
import matplotlib.pyplot as plt

line1, = plt.plot([1,2,3], color="#1f77b4")
line2, = plt.plot([2,3,4], color="#d62728")
line3, = plt.plot([3,2,9], color="#dbdb8d")

plt.label_line(("labelA", "labelB", "labelC"))
plt.show()
```

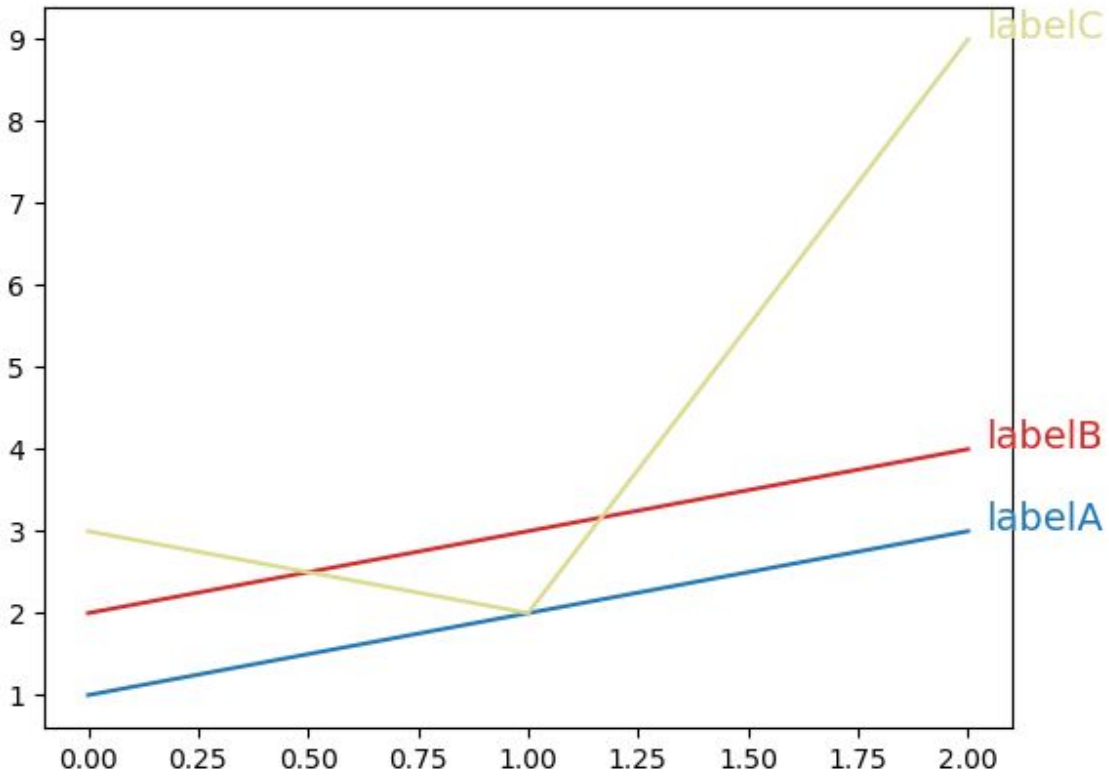
  Result:



- Test3:
  Description:
  User can explicitly assign labels to lines, the label will be placed to the line2D object in the order you specified.
  Code Snippet:

```
12  import matplotlib.pyplot as plt
13
14  line1, = plt.plot([1,2,3], color="#1f77b4")
15  line2, = plt.plot([2,3,4], color="#d62728")
16  line3, = plt.plot([3,2,9], color="#dbdb8d")
17
18  plt.label_line((line1, line2, line3), ("labelA", "labelB", "labelC"))
19  plt.show()
```

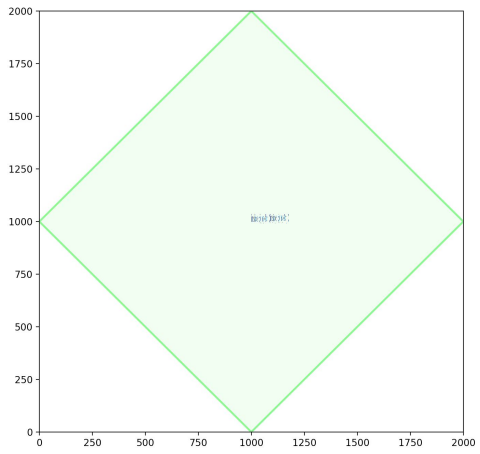
  Result:

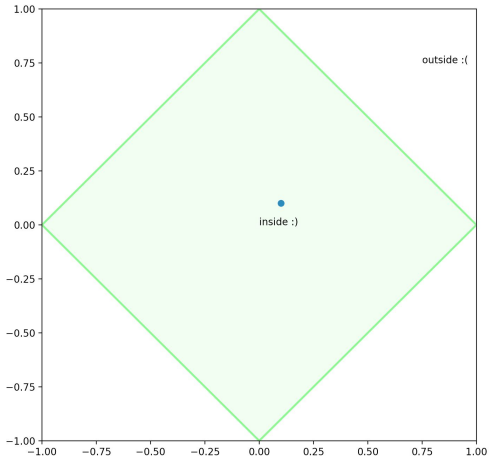

labelC
labelB
labelA
9
8
7
6
5
4
3
2
1
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

Bug Fix 1 - Matplotlib Text object clip_path attribute

Issue Name: text is not clipped by clip_path
Issue #: 8270
URL: https://github.com/matplotlib/matplotlib/issues/8270
Type of Issue: text
Short Explanation:
The matplotlib.text.Text class should have the capability of clipping the text to some specified path by taking the named arguments "clip_on=True/False" and "clip_path=somepath" in its constructor. The issue is that in some cases when "clip_on=True" and "clip_path=somepath" are set, the text outside the path still present. In the example below specifically, when adding 2 matplotlib.text.Text objects to ax (Axes), the firstly added one's clip path does not work.
Idea of Solution:
It seems that the possible cause of this problem is that the matplotlib.text.Text object's clip path gets dropped or reset at some point, so the solution of this problem is probably going through the related functions to find where the text's clip path gets dropped and then come up with a proper way to fix it. Starting with looking at the " _AxesBase.add_artist" and "Artist.set_clip_path" methods is suggested to resolve this bug.
(Partial) Solution:
The bug is not completely fixed, so only a partial solution is implemented. The first cause of this bug is the text's clip_path attribute gets unconditionally updated by " _AxesBase.add_artist" when being added. This is solved in the partial solution by modifying the code to only update clip_path if it hasn't been set. The second cause of this bug is the low level AGG C++ API "RendererAgg::draw_text_image()" doesn't handle clip path properly for text. The partial solution handles clip path for text by rendering it in a similar way as how images are rendered, which unfortunately couldn't solve another problem that text has grayscale pixel data (1 int per pixel) and images have RGBA data (3 int tuple per pixel) and there's no easy way to make the conversion in the new code. Ideally, solving the pixel data conversion issue will completely resolve this bug.

The partial solution is available on branch "issue2-partial-fix". Two changed files are:
https://github.com/CSCD01/team04-project/blob/issue2-partial-fix/matplotlib/src/_backend_agg.h
https://github.com/CSCD01/team04-project/blob/issue2-partial-fix/matplotlib/lib/matplotlib/axes/_base.py





The above screenshots shows the result of the partial fix, compared to the result of old code. As we can see, the text “outside :(” doesn’t show up anymore because its clip_path gets rendered. However, the text “inside :)” becomes some random pixels because the lack of conversion from grayscale data to RGBA data.

Solution Code Snippet and Explanation:

```
1793    def add_artist(self, a):
1794        """
1795        Add an `~.Artist` to the axes, and return the artist.
1796
1797        Use `add_artist` only for artists for which there is no dedicated
1798        "add" method; and if necessary, use a method such as `update_datalim`
1799        to manually update the dataLim if the artist is to be included in
1800        autoscaling.
1801
1802        If no ``transform`` has been specified when creating the artist (e.g.
1803        ``artist.get_transform() == None``) then the transform is set to
1804        ``ax.transData``.
1805        """
1806        a.axes = self
1807        self.artists.append(a)
1808        a._remove_method = self.artists.remove
1809        self._set_artist_props(a)
1810
1811        if (a.get_clip_path() is None):
1812            # Only set the artist's clip path if it hasn't been set
1813            a.set_clip_path(self.patch)
1814        self.stale = True
1815        return a
1816
```

The above screenshot shows how the condition of updating the text’s clip_path gets implemented in matplotlib/lib/matplotlib/axes/_base.py

```
793     typedef agg::span_allocator<agg::rgba8> color_span_alloc_type;
794     typedef agg::image_accessor_clip<pixfmt> image_accessor_type;
795     typedef agg::span_interpolator_linear<> interpolator_type;
796     typedef agg::span_image_filter_rgba_nn<image_accessor_type, interpolator_type>
797     image_span_gen_type;
798     typedef agg::span_converter<image_span_gen_type, span_conv_alpha> span_conv;
799     typedef agg::pixfmt_amask_adaptor<pixfmt, alpha_mask_type> pixfmt_amask_type;
800     typedef agg::renderer_base<pixfmt_amask_type> amask_ren_type;
801     typedef agg::renderer_scanline_aa<amask_ren_type, color_span_alloc_type, span_conv>
802     renderer_type_alpha;
803
804     theRasterizer.reset_clipping();
805     rendererBase.reset_clipping(true);
806     bool has_clippath = render_clippath(gc.clippath.path, gc.clippath.trans);
807     if (angle != 0.0 || has_clippath) {
808         agg::rendering_buffer srcbuf(
809                 image.data(), (unsigned)image.dim(1),
810                 (unsigned)image.dim(0), (unsigned)image.dim(1));
811         pixfmt pixf(srcbuf);
812         set_clipbox(gc.cliprect, theRasterizer);
813         double alpha = gc.alpha;
814         // EFegwg
815         agg::trans_affine mtx;
816         agg::path_storage rect;
817
818         mtx *= agg::trans_affine_translation(0, -image.dim(0));
819         mtx *= agg::trans_affine_rotation(-angle * agg::pi / 180.0);
820         mtx *= agg::trans_affine_translation(x, y);
821
822         rect.move_to(0, 0);
823         rect.line_to(image.dim(1), 0);
824         rect.line_to(image.dim(1), image.dim(0));
825         rect.line_to(0, image.dim(0));
826         rect.line_to(0, 0);
827
828         agg::conv_transform<agg::path_storage> rect2(rect, mtx);
829
830         agg::trans_affine inv_mtx(mtx);
831         inv_mtx.invert();
832
833         color_span_alloc_type sa;
834         image_accessor_type ia(pixf, agg::rgba8(0, 0, 0, 0));
835         interpolator_type interpolator(inv_mtx);
836         image_span_gen_type image_span_generator(ia, interpolator);
837         span_conv_alpha conv_alpha(alpha);
838         span_conv spans(image_span_generator, conv_alpha);
839
840         pixfmt_amask_type pfa(pixFmt, alphaMask);
841         amask_ren_type r(pfa);
842         renderer_type_alpha ri(r, sa, spans);
843
844         theRasterizer.add_path(rect2);
845         agg::render_scanlines(theRasterizer, scanlineAlphaMask, ri);
```

The above screenshot shows how text gets rendered (in a similar way as how images are rendered) in matplotlib/src/_backend_agg.h -- function RendererAgg::draw_text_image().

For detailed information, please refer to the comments in function RendererAgg::draw_text_image() -- https://github.com/CSCD01/team04-project/blob/issue2-partial-fix/matplotlib/src/_backend_agg.h