

**TWENTY SIXERS**  
**DELIVERABLE 4**

Stefan Mitic  
Kevin Bato  
Chia Anamekwe  
Zongda Wang  
Byron Leung



# TABLE OF CONTENTS

<b>User Guide</b>	<b>2</b>
<b>Code Design</b>	<b>6</b>
Widgets	6
GUI Backend	8
Changes in previous design	9
<b>Acceptance Tests</b>	<b>10</b>
Navigation toolbar	10
Opening the parameter editor	10
Modifying axis properties	10
Switching between the Axes and Curves tab	11
Selecting a curve	11
Modifying line properties	11
Modifying marker properties	12
<b>Running On Different Backends</b>	<b>13</b>
Gtk3 Backend	13
Tk Backend	15
Wx Backend	16
<b>Unit Tests</b>	<b>18</b>

# User Guide

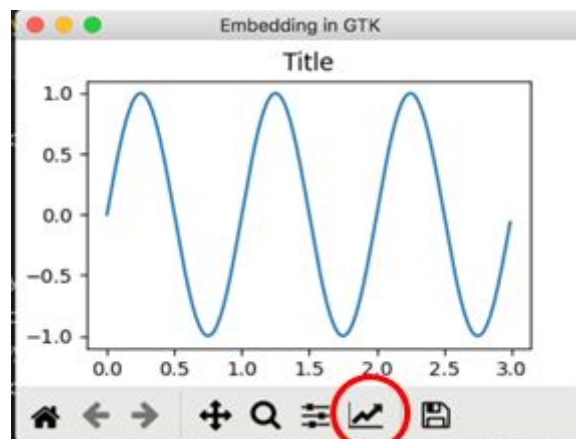
## *Introduction*

Welcome to the latest new feature of matplotlib! Below, you will find a comprehensive guide on all the capabilities of this new feature, and how it can be used.

The feature enables users to edit figure parameters, such as title, labels and line styles, directly through an added widget on the toolbar, providing a uniform user experience regardless of GUI backend (Qt, GTK, Tk or WX).

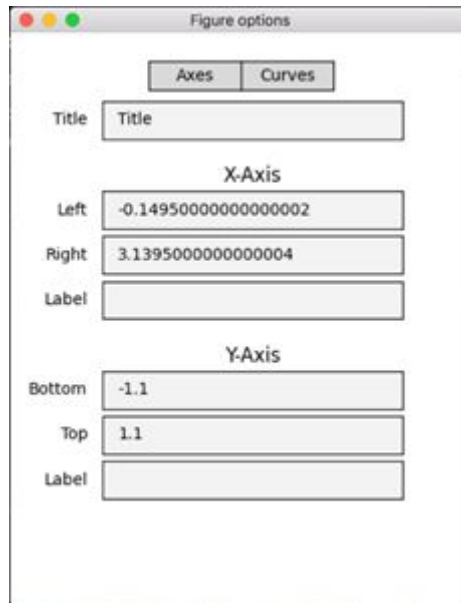
## *Figure Plot*

Below is an example plot. The toolbar is located at the bottom of the figure, with the added feature indicated by the red circle.



## *Using the edit parameters widget - Axes*

Clicking on the widget opens a dialogue box, shown below. The dialogue opens up under the 'Axes' tab, displaying the various parameters the user can edit relating to the axes of the plot.

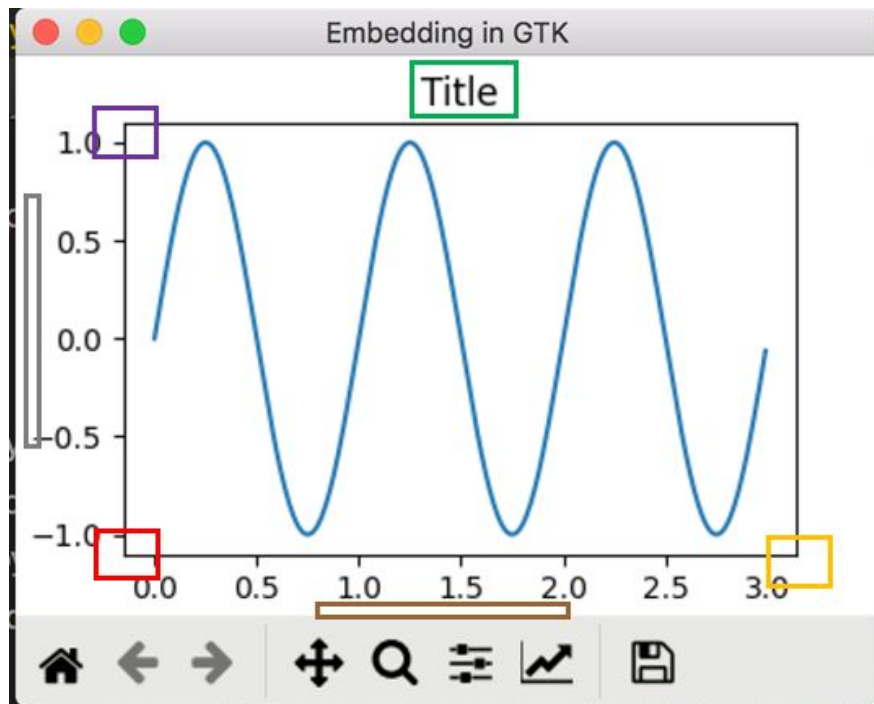


The image shows a 'Figure options' dialog box with two tabs: 'Axes' and 'Curves'. The 'Axes' tab is selected. It contains the following fields:

- Title:** A text box containing the word 'Title'.
- X-Axis:**
  - Left:** A text box containing the value '-0.14950000000000002'.
  - Right:** A text box containing the value '3.13950000000000004'.
  - Label:** An empty text box.
- Y-Axis:**
  - Bottom:** A text box containing the value '-1.1'.
  - Top:** A text box containing the value '1.1'.
  - Label:** An empty text box.

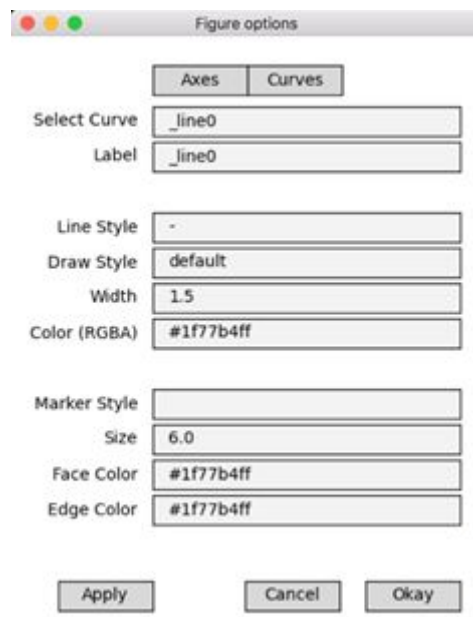
There are 7 configurable fields:

1. Title – updating the value in this text box changes the title of the plot, indicated in green on the image below.
2. Left – updating the value in this text box changes the starting value of the x axis, indicated in red on the image below.
3. Right – updating the value in this text box changes the ending value of the x axis, indicated in orange on the image below.
4. Label – updating the value in this text box changes the label of the x axis, indicated in brown on the image below.
5. Bottom – updating the value in this text box changes the starting value of the y axis, indicated in red on the image below.
6. Top – updating the value in this text box changes the ending value of the y axis, indicated in purple on the image below.
7. Label – updating the value in this text box changes the label of the x axis, indicated in grey on the image below.



### *Using the edit parameters widget - Curves*

Clicking on the 'Curves' tab opens a new page of the dialogue box, shown below. This displays the various parameters the user can edit relating to the line of the plot.





There are 10 configurable fields:

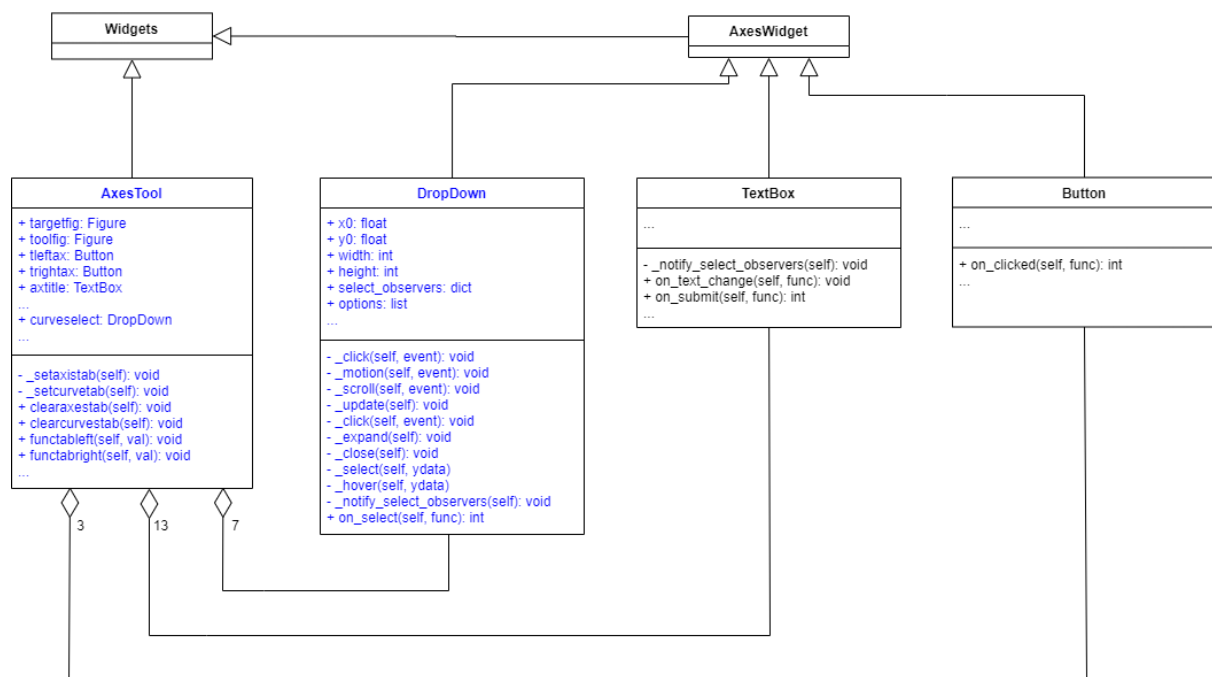
1. Select Curve – selecting a value from this dropdown changes the line in which the following parameters will be applied to (dropdown shows all lines currently on plot, allows values to be set independently for multiple lines).
2. Label – updating the value in this text box changes the label associated with the line selected in 'Select Curve'.
3. Line Style – selecting a value from this dropdown changes the style of the selected line (possible values are solid, dashed, dashdot (dash-dotted line), dotted or None).
4. Draw Style – selecting a value from this dropdown changes the way steps are drawn (possible values are steps, steps-mid or steps-post).
5. Width – updating the value in this text box changes the width of the selected line. Values are in points.
6. Color (RGBA) – changing the value in this text box changes the color of the selected line to the specified RGBA value.
7. Marker Style – selecting a value from this dropdown changes the style of markers (points on graph). Many possible values, including circle, triangle and star.
8. Size – updating the value in this text box changes the size of the markers. Values are in points.
9. Face Color – updating the value in this text box changes the face (base) color of the line markers to the specified hex value.
10. Edge Color – updating the value in this text box changes the edge (outline) color of the line markers to the specified RGBA value.

# Code Design

## Widgets

**File modified:** matplotlib.widgets

AxesTool class is a new class where axes and curve modifications are made on the toolbar while displaying a graph. Axes modification includes: setting title of the plot or label of x and y axis, modifying range limits of x and y axis as shown on User Guide. Lastly, curve modification includes: creating a label, changing the properties of a curve (line style, draw style, line width, colour, etc..) also shown on User Guide.



Based on other pre-existing widgets such as SubplotTool, Lasso and MultiCursor which inherit Widgets, AxesTool class is made to a similar design and structure of these widgets. The layout of AxesTool widget is similar to the SubplotTool widget by using the function, `add_subplot(...)` for a figure layout. This function adds GUI elements based on a grid layout using rows and columns which can position the GUI elements shown on the toolbar. The main GUI elements contain several Textbox, Button and Dropdown classes. The class reuses base widgets from matplotlib's GUI elements such as Textbox and Button, and also uses a newly constructed DropDown



class as indicated above on the UML. The blue on the UML indicates a new class on matplotlib.

The Dropdown class is a new GUI element added on the AxesTool widget. Like AxesTool widget, Dropdown class is similar in design to GUI elements such as TextBox, Button, Slider, etc. that all inherit from AxesWidget to remain consistent with the existing codebase and allow reusability for new or existing widgets. An interesting observer design pattern is used on GUI elements, they require an action for it to trigger an event. For example, a Button class is clicked (action) which triggers an event that occurs due to an `on_clicked` function. With this design pattern in mind, Dropdown class has many event triggering functions as shown on the UML.

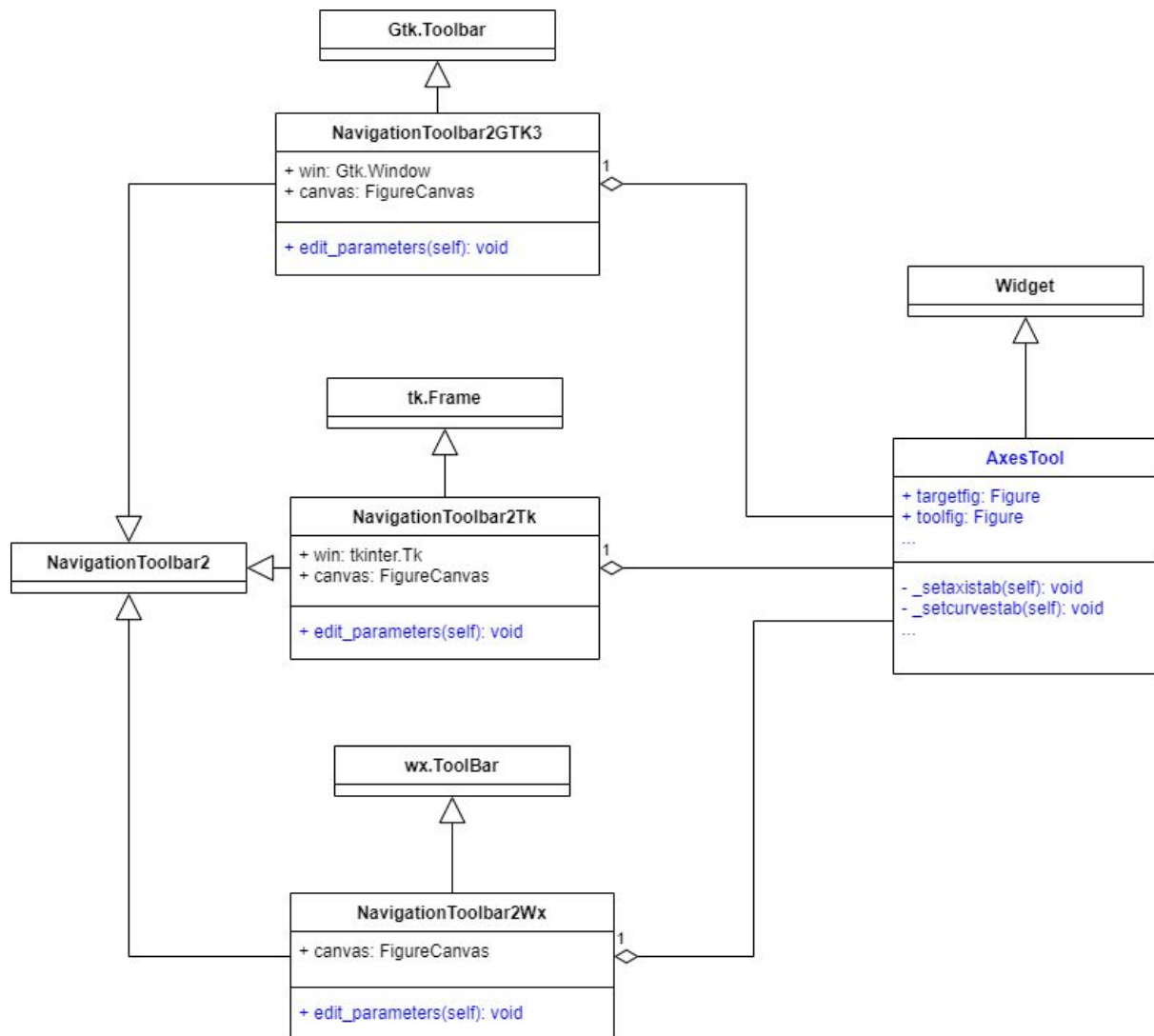


## GUI Backend

### Files modified:

- matplotlib.backend\_bases,
- matplotlib.backends.backend\_gtk3,
- matplotlib.backends.\_backend\_tk,
- Matplotlib.backends.\_backend\_wx

The feature allows users to edit the parameters of a GTK, Tk, or WX-embedded figure through the GUI. Note that Qt backend was not modified since it already supports edit parameters.





in `matplotlib.backend_bases`:

- By implementing the parameter editor for all the backends, we integrate it for the base `NavigationToolbar2` class which is inherited by the navigation bars for all the backends.

in `matplotlib.backends.backend_gtk3`:

- This file contains classes for many components that can be included in a GTK window including the `NavigationToolbar`, which currently contains buttons that allow the user to do actions such as pan, zoom, and save the figure. We will add our parameter editor tool to this toolbar, so we must create and add a button for it, which can be done by the `edit_parameters` function as shown in blue on the UML above. The button will call another function within the class that we need to implement which will display and configure the GUI element for our parameter editor.

in `matplotlib.backends._backend_tk`:

- Same as above but for Tk backend

in `matplotlib.backends.backend_wx`:

- Same as above but for Wx backend

## Changes in previous design

There was no difference in the implementation to the several backends, most of the planning was already thought out during the previous deliverable. However, the `AxesTool` widget needed some more planning to incorporate different GUI elements. Especially creating a new GUI element, dropdown menu, since it was not a part of the basic GUI element that matplotlib provided. Also, to research more on setting changes to a plot from the toolbar then drawing into the figure.

## Acceptance Tests

### Navigation toolbar

Tests:

1. Create and display a GTK-embedded matplotlib figure with a navigation toolbar.
2. Create and display a Tk-embedded matplotlib figure with a navigation toolbar.
3. Create and display a WX-embedded matplotlib figure with a navigation toolbar.

Expected outcome:

- A GUI window should pop up containing the matplotlib figure and navigation toolbar.

### Opening the parameter editor

Tests:

1. Click the parameter editor tool icon on the GTK navigation bar.
2. Click the parameter editor tool icon on the Tk navigation bar.
3. Click the parameter editor tool icon on the WX navigation bar.

Expected outcome:

- The parameter editor tool titled “Figure options” should pop up in a new window on the Axes tab where the user can edit axis properties.

### Modifying axis properties

Tests (for each backend):

1. Change the title of the figure by entering a new title in the title textbox and hitting enter.
2. Change the x or y-axis limits of an axes through the parameter editor by entering a new value in the limit value textbox and hitting enter.
3. Change the x or y-axis label of an axes through the parameter editor by using the label textbox.

Expected outcome:

- The axis properties on the figure should change accordingly.



## Switching between the Axes and Curves tab

Tests (for each backend):

1. Switch between the Axes and Curves tab using the two buttons near the top of the figure options window.

Expected outcome:

- The window should update accordingly to allow the user to edit either axis or curves properties.

## Selecting a curve

Tests (for each backend):

1. Switch to Curves tab using the button near the top of the figure window.
2. Click on the dropdown menu labelled “Select Curve” and select from the list. If there are more curves to select, use the scroll wheel of your mouse to find more options.

Expected outcome:

- The desired line should appear on the figure options with its line properties.

## Modifying line properties

Tests (for each backend):

1. Switch to Curves tab using the button near the top of the figure window.
2. Select a curve you wish to modify by selecting on the dropdown menu labelled “Select Curve” through the parameter editor.
3. Change the label of a line by typing into a text box named “Label” and hit enter through the parameter editor.
4. Change the style of a line (dashed, dotted, etc.) by selecting on the dropdown menu through the parameter editor.
5. Change the width of a line through the parameter editor by typing a positive numerical value and hit enter.
6. Change the color of a line through the parameter editor by typing a “#” at the beginning with a hexadecimal number of length 8 and hit enter.
7. Click the apply button to apply your changes.

Expected outcome:

- The properties of the line in the figure should change accordingly.

## Modifying marker properties

Tests (for each backend):

1. Switch to Curves tab using the button near the top of the figure window.
2. Select a curve you wish to modify by selecting on the dropdown menu labelled "Select Curve" through the parameter editor.
3. Change the style of a marker (point, star, etc.) by selecting on the dropdown menu through the parameter editor. There are more options if you use your scroll wheel on your mouse.
4. Change the size of a marker through the parameter editor by typing a positive numerical value and hit enter.
5. Change the face color of a marker through the parameter editor by typing a "#" at the beginning with a hexadecimal number of length 8 and hit enter.
6. Change the edge color of a marker through the parameter editor by typing a "#" at the beginning with a hexadecimal number of length 8 and hit enter.
7. Click the apply button to apply your changes.

Expected outcome:

- The marker in the figure should change accordingly..

## Running On Different Backends

### Gtk3 Backend

```
import gi
gi.require_version('Gtk', '3.0')
from gi.repository import Gtk

from matplotlib.backends.backend_gtk3 import (
    NavigationToolbar2GTK3 as NavigationToolbar)
from matplotlib.backends.backend_gtk3agg import (
    FigureCanvasGTK3Agg as FigureCanvas)
from matplotlib.figure import Figure
import numpy as np

win = Gtk.Window()
win.connect("delete-event", Gtk.main_quit)
win.set_default_size(400, 300)
win.set_title("Embedding in GTK")

f = Figure(figsize=(5, 4), dpi=100)
a = f.add_subplot(1, 1, 1)
t = np.arange(0.0, 3.0, 0.01)
s = np.sin(2*np.pi*t)
a.plot(t, s)

vbox = Gtk.VBox()
win.add(vbox)

canvas = FigureCanvas(f) # a Gtk.DrawingArea
vbox.pack_start(canvas, True, True, 0)

toolbar = NavigationToolbar(canvas, win)
vbox.pack_start(toolbar, False, False, 0)

win.show_all()
Gtk.main()
```

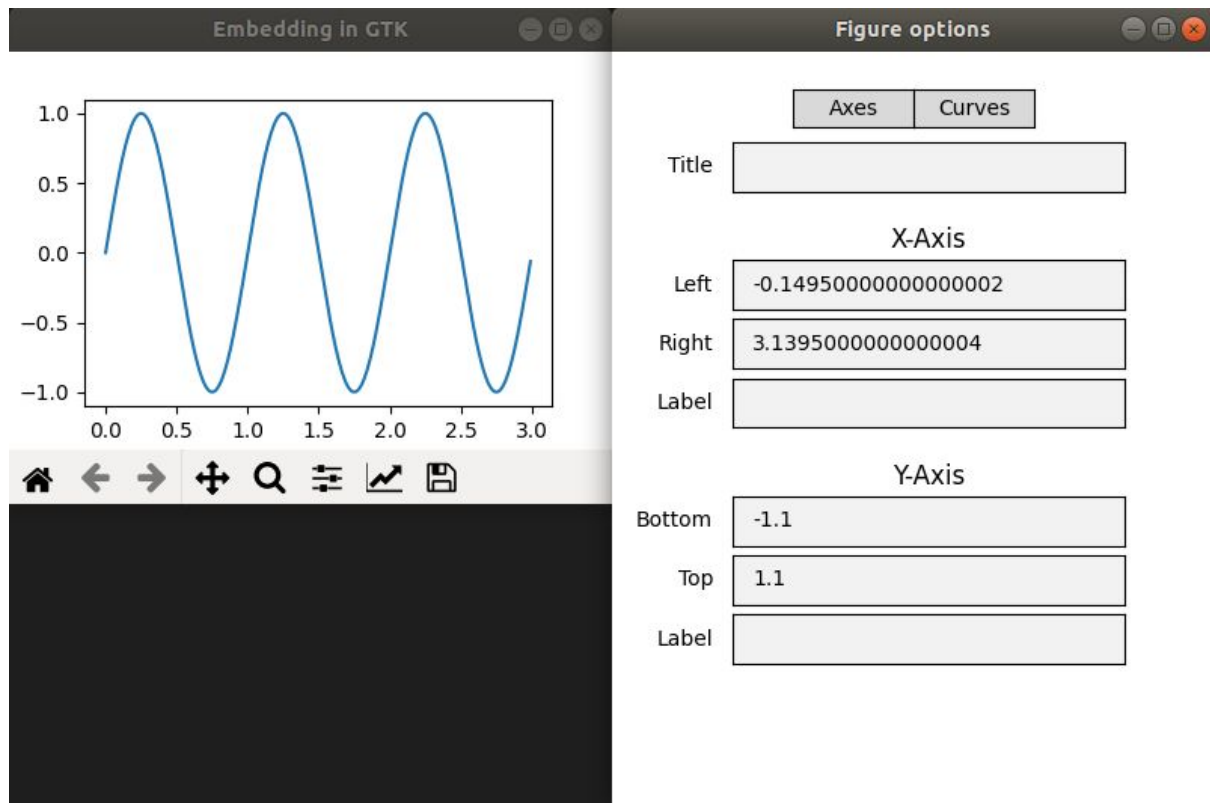


Figure options

Axes
Curves

Select Curve

Label

Line Style

Draw Style

Width

Color (RGBA)

Marker Style

Size

Face Color

Edge Color

Apply
Cancel
Okay

## Tk Backend

```
import tkinter
from matplotlib.backends.backend_tkagg import (
    FigureCanvasTkAgg, NavigationToolbar2Tk)
from matplotlib.backends_bases import key_press_handler
from matplotlib.figure import Figure
import numpy as np

root = tkinter.Tk()
root.wm_title("Embedding in Tk")

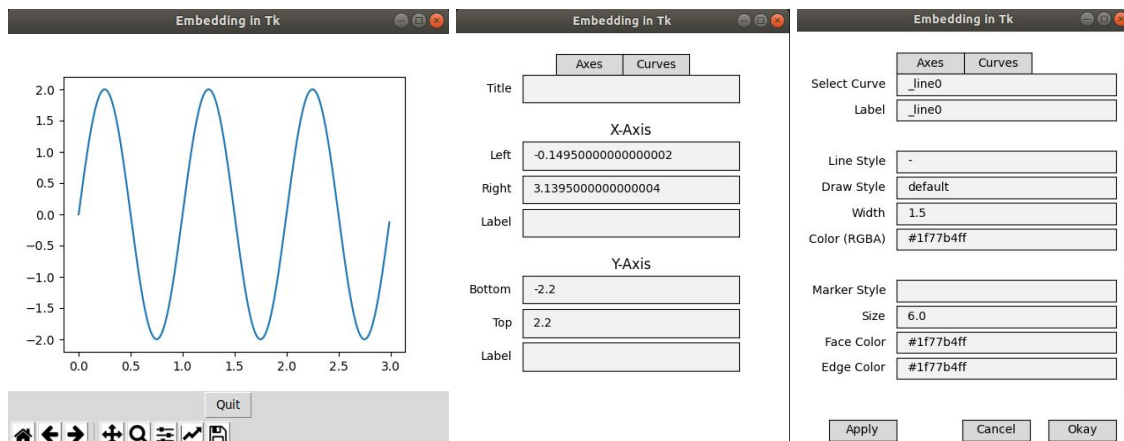
fig = Figure(figsize=(5, 4), dpi=100)
t = np.arange(0, 3, .01)
fig.add_subplot(111).plot(t, 2 * np.sin(2 * np.pi * t))

canvas = FigureCanvasTkAgg(fig, master=root) # A tk.DrawingArea.
canvas.draw()

toolbar = NavigationToolbar2Tk(canvas, root)
toolbar.update()

button = tkinter.Button(master=root, text="Quit", command=root.quit)
button.pack(side=tkinter.BOTTOM)
canvas.get_tk_widget().pack(side=tkinter.TOP, fill=tkinter.BOTH, expand=1)

tkinter.mainloop()
```







## Wx Backend

```
from matplotlib.backends.backend_wxagg import (
    FigureCanvasWxAgg as FigureCanvas,
    NavigationToolbar2WxAgg as NavigationToolbar,
)
from matplotlib.figure import Figure
import numpy as np
import wx

class CanvasFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, -1,
                          'CanvasFrame', size=(550, 350))

        self.figure = Figure(figsize=(5, 4), dpi=100)
        self.axes = self.figure.add_subplot(111)
        t = np.arange(0.0, 3.0, 0.01)
        s = np.sin(2 * np.pi * t)

        self.axes.plot(t, s)
        self.canvas = FigureCanvas(self, -1, self.figure)
        self.sizer = wx.BoxSizer(wx.VERTICAL)
        self.sizer.Add(self.canvas, 1, wx.TOP | wx.LEFT | wx.EXPAND)
        self.toolbar = NavigationToolbar(self.canvas)
        self.toolbar.Realize()
        self.sizer.Add(self.toolbar, 0, wx.LEFT | wx.EXPAND)
        self.toolbar.update()
        self.SetSizer(self.sizer)
        self.Fit()

class App(wx.App):
    def OnInit(self):
        'Create the main window and insert the custom frame'
        frame = CanvasFrame()
        frame.Show(True)

        return True

app = App(0)
app.MainLoop()
```

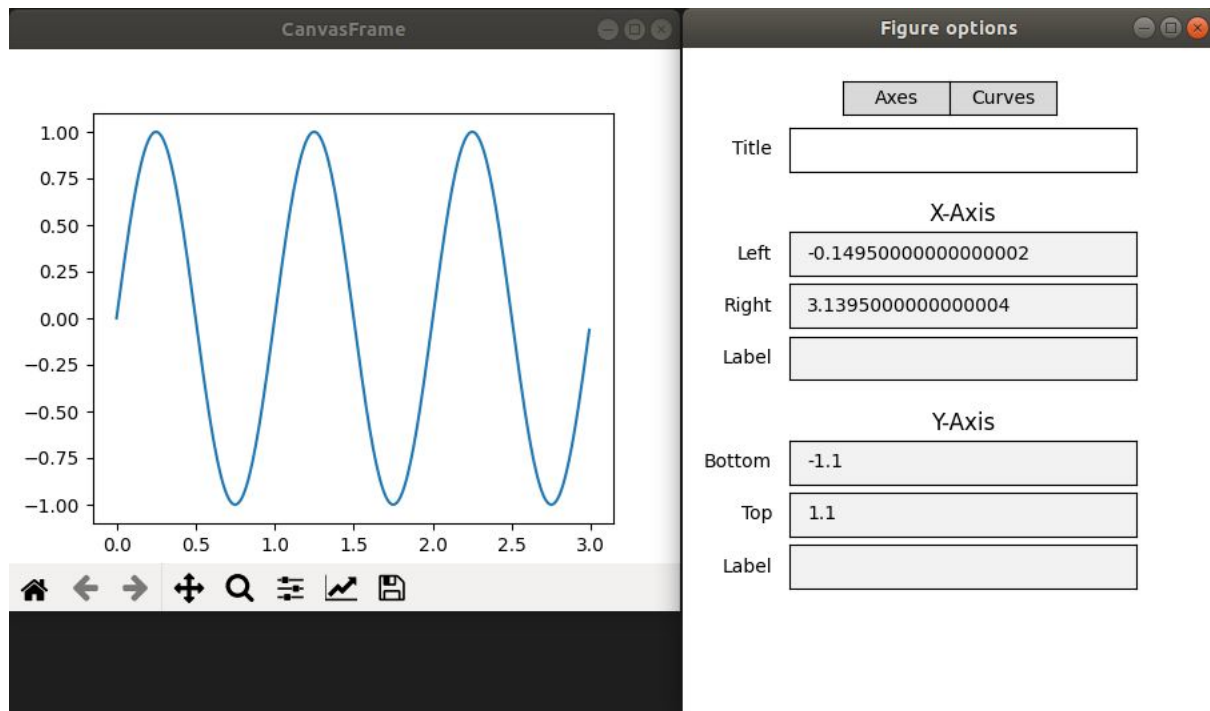


Figure options

Axes
Curves

Select Curve

Label

Line Style

Draw Style

Width

Color (RGBA)

Marker Style

Size

Face Color

Edge Color

Apply
Cancel
Okay



## Unit Tests

matplotlib.tests.test\_widgets:

- test\_dropdown\_select
  - Since the dropdown widget must be expanded before an option can be selected, this tests both expanding and selecting an option in the dropdown..
- test\_dropdown\_init\_index\_invalid
  - Tests an invalid initial option index for the dropdown widget.
- test\_dropdown\_max\_height\_invalid
  - Tests a negative maximum height for the dropdown.

We did not have unit tests for our AxesTool widget because it only works in conjunction with the functions in each GUI backend that creates it, while the Dropdown widget can be added to and used within any figure.