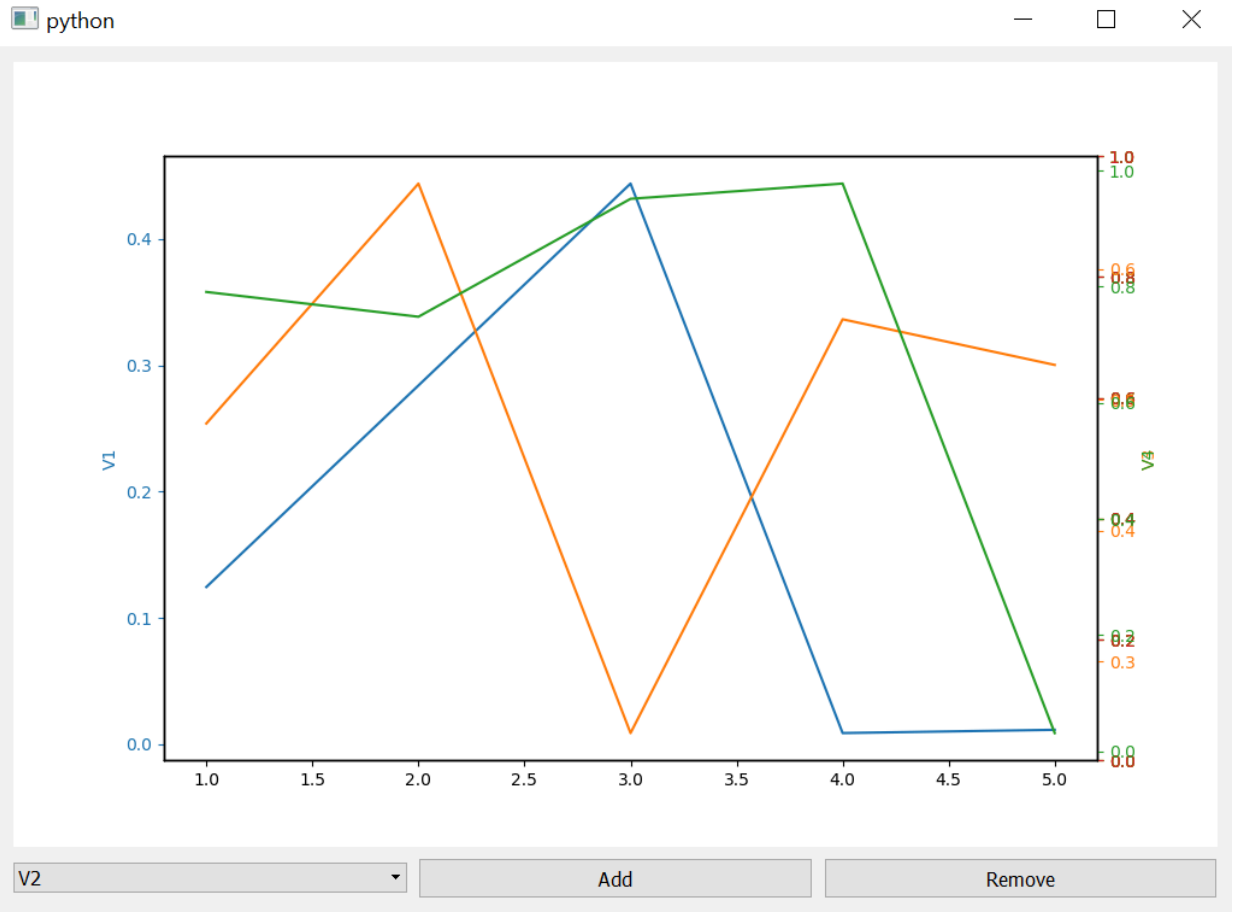
How to aesthetically show a generic number of axes in matplotlib?

Asked 6 years, 4 months ago Modified 6 years, 4 months ago Viewed 142 times

2

I want to do a simple GUI that allows the user to add or remove traces from a plot for any number of traces. It looks like this:



The problems I'm having:

- I don't know how to make the axes not to superpose with each other for a **generic** number of plots.
- When I plot more than one trace, and then delete all but one, there are two axes showing for some reason. **There should always be one axis per trace being shown.**

Is there a way to fix these issues? You can find my code below. The only function that should be changed is `update_canvas()`, I believe. To try it out, just modify the list `name_vars` in the `main` with the number of variables you want. The rest of the example code is self-contained.

```
import numpy as np
from matplotlib.backends.qt_compat import QtWidgets
from matplotlib.backends.backend_qt5agg import FigureCanvas
from matplotlib.figure import Figure

class ApplicationWindow(QtWidgets.QMainWindow):
    def __init__(self, parent=None):
        super(ApplicationWindow, self).__init__(parent)
        global name_vars
        self.x = np.array([1,2,3,4,5])
        self.y = np.random.random((5, len(name_vars)))
        self.num_vars = np.size(self.y,1)
        self.name_vars = name_vars

        self.tags_on = [0] * self.num_vars
        self.colors = ['#1F77B4','#FF7F0E','#2CA02C','#D62728','#9467BD',
                       '#8C564B','#E377C2','#F7F7F7','#BCBD22','#17BECF']

        self._main = QtWidgets.QWidget()
        self.setCentralWidget(self._main)

        canvas = FigureCanvas(Figure(figsize=(10, 10)))
        self.canvas_ax = canvas.figure.subplots()
        self.canvas_ax.set_xlabel("Time")
        self.canvas_ax_twin = []

        self.list_tags = QtWidgets.QComboBox(self)
        for name in self.name_vars:
            self.list_tags.addItem(name)
        button_add = QtWidgets.QPushButton('Add', self)
        button_remove = QtWidgets.QPushButton('Remove', self)
        button_add.clicked.connect(self.add_plot)
        button_remove.clicked.connect(self.remove_plot)

        layout = QtWidgets.QGridLayout(self._main)
        layout.addWidget(canvas, 0, 0)
        dropdown_layout = QtWidgets.QHBoxLayout()
        dropdown_layout.addWidget(self.list_tags)
        dropdown_layout.addWidget(button_add)
        dropdown_layout.addWidget(button_remove)
        layout.addLayout(dropdown_layout, 1, 0)
        self.show()

    def add_plot(self):
        selected_tag = self.list_tags.currentIndex()
        self.tags_on[selected_tag] = 1
        self.update_canvas()

    def remove_plot(self):
        selected_tag = self.list_tags.currentIndex()
        self.tags_on[selected_tag] = 0
        self.update_canvas()

    def update_canvas(self):
        # Delete all traces
        self.canvas_ax.clear()
        [i.clear() for i in self.canvas_ax_twin]
        self.canvas_ax_twin = []
        num_plots = 0
        for ii in range(self.num_vars):
            if self.tags_on[ii] == 1:
```

```
                # If it's not the first trace, create a twin axis
                if num_plots != 0:
                    self.canvas_ax_twin.append(self.canvas_ax.twinx())
                    self.canvas_ax_twin[-1].plot(self.x, self.y[:,ii],
self.colors[num_plots])
                    self.canvas_ax_twin[-1].set_ylabel(self.name_vars[ii])

self.canvas_ax_twin[-1].yaxis.label.set_color(self.colors[num_plots])
                    self.canvas_ax_twin[-1].tick_params(axis='y',
colors=self.colors[num_plots])
                    num_plots += 1
                # If it's the first trace, use the original axis
                else:
                    self.canvas_ax.plot(self.x, self.y[:,ii],
self.colors[num_plots])
                    self.canvas_ax.set_ylabel(self.name_vars[ii])

self.canvas_ax.yaxis.label.set_color(self.colors[num_plots])
                    self.canvas_ax.tick_params(axis='y',
colors=self.colors[num_plots])
                    num_plots += 1
        # Show the final plot
        self.canvas_ax.figure.canvas.draw()

if __name__ == '__main__':
    # Edit the number of elements in name_vars to try the code
    name_vars = ['V1','V2','V3','V4']
    app = QtWidgets.QApplication([])
    ex = ApplicationWindow()
    ex.show()
    app.exec_()
```

python user-interface matplotlib pyqt pyqt5

Share

Improve this question

Follow

edited Aug 2, 2018 at 16:45

asked Aug 2, 2018 at 15:20

Tendero

1,166 ● 3 ● 21 ● 40

(1) Did you look at this example from the matplotlib page? (2) Thats expected. Each first line with each axes has the same color. You can set the colors yourself if you want. (4) axes don't have a color, you would need to hold a reference to your line and do `line.get_color()`. (3) Seems to be the only real issue here. So can you limit this question to one question per question (as in "ask a question" - not many)? – ImportanceOfBeingErnest Aug 2, 2018 at 15:38

@ImportanceOfBeingErnest Thanks for your suggestions! I was able to fix two of the problems. Regarding (1), that example shows how to do it when you know beforehand the number of traces, but I'm not being able to figure out how to do it generically. Now that there are only two issues remaining, I think that one post for both of them is alright as they are pretty related (multiple axes using matplotlib). If not, let me know and I'll separate the questions. – Tendero Aug 2, 2018 at 16:42

1 Answer

Sorted by: Highest score (default)

2

I would suggest to separate the logic from the actual plotting. This makes it easier to follow through. This solves the second question about not removing all axes.

The question about not letting the axes superimpose may be solved by setting the position of additional twin axes to some distance from the axes, depending on how many axes you have.

```
ax.spines["right"].set_position(("axes", 1+(n-1)*0.1))
```

where `n` is the axes number starting from 0. The main axes (n=0) should be excluded, and the first axes will stay at position 1. Further axes are positionned in steps of 0.1.

Then it makes sense to also adjust the right margin of the main axes to give enough space for the extra spines.

```
import numpy as np
from matplotlib.backends.qt_compat import QtWidgets
from matplotlib.backends.backend_qt5agg import FigureCanvas
from matplotlib.figure import Figure

class ApplicationWindow(QtWidgets.QMainWindow):
    def __init__(self, parent=None, name_vars=[]):
        super(ApplicationWindow, self).__init__(parent)

        self.x = np.array([1,2,3,4,5])
        self.y = np.random.random((5, len(name_vars)))
        self.num_vars = np.size(self.y,1)
        self.name_vars = name_vars
        self.tags_on = [0] * self.num_vars

        self._main = QtWidgets.QWidget()
        self.setCentralWidget(self._main)

        self.figure = Figure(figsize=(10, 10))
        canvas = FigureCanvas(self.figure)
        self.left = self.figure.subplotpars.left
        self.right = self.figure.subplotpars.right
        self.canvas_ax = canvas.figure.subplots()
        self.canvas_ax.set_xlabel("Time")
        self.axes = [self.canvas_ax]

        self.list_tags = QtWidgets.QComboBox(self)
        for name in self.name_vars:
            self.list_tags.addItem(name)
        button_add = QtWidgets.QPushButton('Add', self)
        button_remove = QtWidgets.QPushButton('Remove', self)
        button_add.clicked.connect(self.add_plot)
        button_remove.clicked.connect(self.remove_plot)

        layout = QtWidgets.QGridLayout(self._main)
```

```
        layout.addWidget(canvas, 0, 0)
        dropdown_layout = QtWidgets.QHBoxLayout()
        dropdown_layout.addWidget(self.list_tags)
        dropdown_layout.addWidget(button_add)
        dropdown_layout.addWidget(button_remove)
        layout.addLayout(dropdown_layout, 1, 0)
        self.show()

    def add_plot(self):
        selected_tag = self.list_tags.currentIndex()
        self.tags_on[selected_tag] = 1
        self.update_canvas()

    def remove_plot(self):
        selected_tag = self.list_tags.currentIndex()
        self.tags_on[selected_tag] = 0
        self.update_canvas()

    def create_nth_axes(self, n, dataset):
        if n == 0:
            ax = self.canvas_ax
        else:
            ax = self.canvas_ax.twinx()
            ax.spines["right"].set_position(("axes", 1+(n-1)*0.1))
            for direction in ["left", "bottom", "top"]:
                ax.spines[direction].set_visible(False)
            # adjust subplotparams to make space for new axes spine
            new_right = (self.right-self.left)/(1+(n-1)*0.1)+self.left
            self.figure.subplots_adjust(right=new_right)
        color = next(self.canvas_ax._get_lines.prop_cycler)['color']
        ax.set_ylabel(self.name_vars[dataset], color=color)
        ax.plot(self.x, self.y[:,dataset], color=color)
        return ax

    def clear_canvas(self):
        # Clear main axes
        self.canvas_ax.clear()
        # clear and remove other axes
        for ax in self.axes[1:]:
            ax.clear()
            ax.remove()
        self.axes = [self.canvas_ax]
        self.figure.subplots_adjust(right=0.9)

    def update_canvas(self):
        self.clear_canvas()
        k = 0
        for i, tag in enumerate(self.tags_on):
            if tag:
                ax = self.create_nth_axes(k, i)
                if k > 0:
                    self.axes.append(ax)
                k += 1
        self.canvas_ax.figure.canvas.draw()

if __name__ == '__main__':
    # Edit the number of elements in name_vars to try the code
    name_vars = ['V1','V2','V3','V4']
    app = QtWidgets.QApplication([])
    ex = ApplicationWindow(name_vars=name_vars)
```

```
    ex.show()
    app.exec_()
```

Share Improve this answer Follow

answered Aug 2, 2018 at 17:28

ImportanceOfBeingErnest
338k 60 729 757