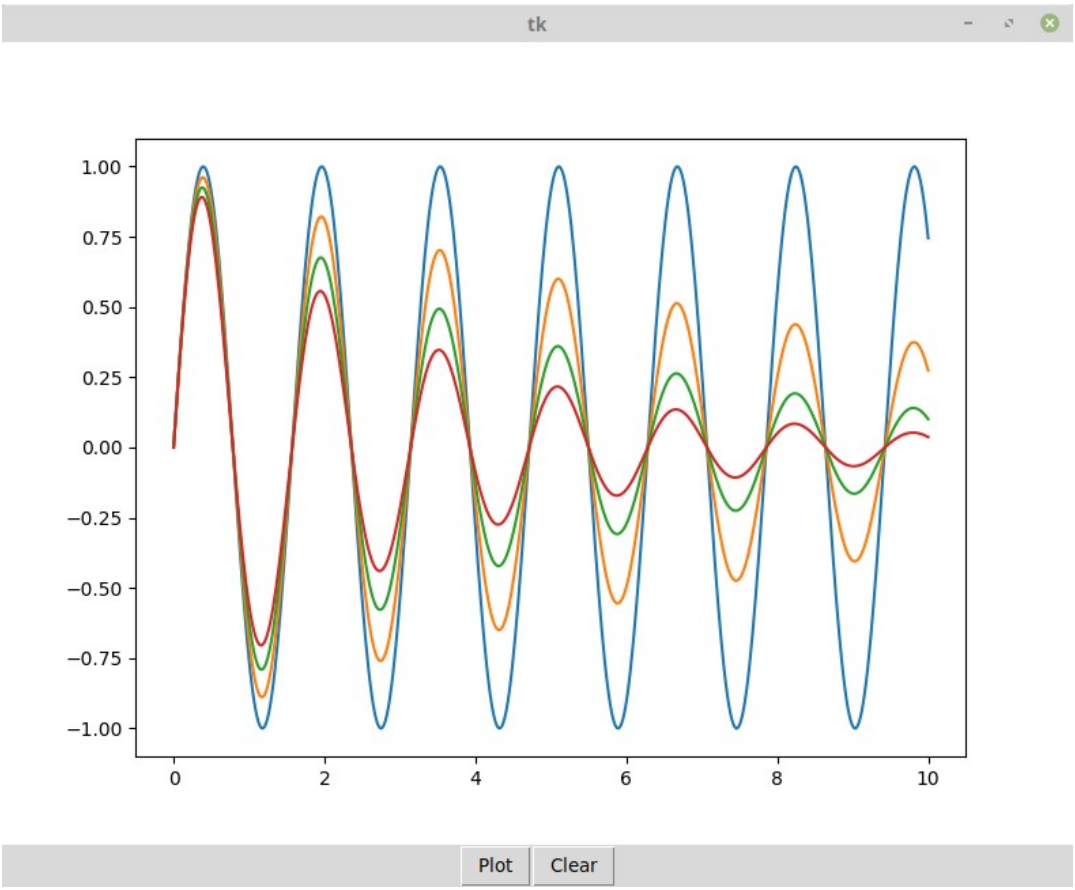
Das Einbetten von Matplotlib in Tkinter

jean-claude.feltes@education.lu



Dieser Artikel beschreibt ein Diagrammfenster, das in Tkinter eingebettet werden kann.

Imports

Matplotlib kann in verschiedenem Kontext eingesetzt werden: interaktiv, in einem GUI (mit Tkinter, Wx, GTK, Qt usw.), in einer Webanwendung oder zum Abspeichern von Diagrammen in einem Bildformat.
In all diesen Fällen ist das Backend zuständig für das eigentliche Zeichnen.

Zum Einbetten des Diagramms in Tkinter muss also zuerst das TkAgg-Backend festgelegt werden:

```
import matplotlib
matplotlib.use('TkAgg')
```

Weiter Infos zu Backends:
https://matplotlib.org/faq/usage_faq.html#what-is-a-backend

Als nächstes wird ein FigureCanvas-Objekt importiert:

```
    from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
```

Dieses ist die eigentliche Zeichenfläche.

Ein Diagrammfenster als Klasse

In ihrer einfachsten Form hat diese Klasse die Methoden plotxy zum Plotten und clearplot zum Löschen der gezeichneten Kurven:

```
class Plotwindow():
    def __init__(self, masterframe, size):

        (w,h)=size
        inchsize=(w/25.4, h/25.4)
        self.figure = Figure(inchsize)
        self.axes = self.figure.add_subplot(111)

        # create canvas as matplotlib drawing area
        self.canvas = FigureCanvasTkAgg(self.figure, master=masterframe)
        self.canvas.get_tk_widget().pack()

    def plotxy(self, x, y):
        self.axes.plot(x,y)
        self.canvas.draw()

    def clearplot(self):
        self.axes.cla()
        self.canvas.draw()
```

Die __init__ - Funktion

Beim Initialisieren wird eine Frame als masterframe und die Größe in Millimetern als Tupel (width, height) übergeben. Die Größe wird in Inch umgerechnet, da Matplotlib diese Einheit erwartet.

Als erstes wird ein **Figure**-Objekt erzeugt. Dies ist die Zeichenfläche für ein oder mehrere Diagramme:

```
    self.figure = Figure(inchsize)
```

Dann wird ein **Axes**-Objekt erzeugt. Dies ist nicht das Koordinatensystem, wie man vielleicht geneigt ist zu denken, sondern die Zeichenfläche des Plots, d.h. die Fläche, die von den eventuellen Kurvenlinien beansprucht werden kann.

Eine Figure kann mehrere Axes-Objekte enthalten, wenn es mehrere Plots gibt.
In unserem Fall wollen wir nur ein Plot, was mit dem Aufruf

```
    self.axes = self.figure.add_subplot(111)
```

geschieht.
Die Kodierung für den Parameter ist dabei <nrows><ncols><figindex> mit ncol=Anzahl der Spalten, nrows = Anzahl der Zeilen, figindex= Nummer des Diagramms (beginnend mit 1).

Die **Axis**-Objekte, also die Koordinatenlinien mit Skala, werden beim Anlegen des Axes-Objektes automatisch mit erzeugt, mit Defaultwerten für die Skalierung.

Als Nächstes wird ein **Canvas**-Objekt erzeugt und in die Masterframe gepackt. Der Canvas ist die eigentliche Zeichenfläche auf die das Backend zugreift.

```
self.canvas = FigureCanvasTkAgg(self.figure, master=masterframe)
self.canvas.get_tk_widget().pack()
```

Die plotxy-Funktion

ist einfach aufgebaut:

```
def plotxy(self, x, y):
    self.axes.plot(x,y)
    self.canvas.draw()
```

Es werden die zu plottenden Punkte als Numpy-Vektoren x und y übergeben und geplottet. Damit die Änderung im Diagramm sichtbar wird, muß die canvas.draw-Funktion aufgerufen werden.

Wenn die plotxy-Funktion mehrmals aufgerufen wird, werden mehrere Kurven in das Diagramm gezeichnet.

Die clearplot-Funktion

Man muß wissen, dass es unterschiedliche Clear-Funktionen gibt:

https://stackoverflow.com/questions/8213522/when-to-use-cla-clf-or-close-for-clearing-a-plot-in-matplotlib

In unserem Fall soll ein Clear des Axes-Objekts, also des eigentlichen Plots geschehen:

```
def clearplot(self):
    self.axes.cla()
    self.canvas.draw()
```

Auch hier muß mit der draw-Funktion das Bild aufgefrischt werden.

Ein einfaches Testprogramm

Ein Generator für Testdaten

Um die Sache einfacher und praktischer zu machen, wird zunächst ein Generator für Testdaten definiert:

```
class Generatetestdata():
    def __init__(self):
        self.index=0        # index of function call
        self.xmin=0.0
```

```
        self.xmax=10.0
        self.nbvalues=500
    def getxy(self):
        n=self.index
        x=np.linspace(self.xmin, self.xmax, self.nbvalues)
        y=np.sin(4*x)*np.exp(-n*x/10)
        self.index+=1
        #self.xmax+=5.0
        return x,y
```

Dieses Objekt liefert beim Aufruf der getxy-Funktion zwei Vektoren x und y zurück.
Die y-Daten entsprechen einer gedämpften Sinusfunktion, wobei der Dämpfungsfaktor bei jedem Aufruf erhöht wird. Dies ist interessant wenn man überprüfen will ob mehrere Kurven korrekt gezeichnet werden.

Will man auch noch die Breite der Daten in x-Richtung verändern um zu prüfen ob die Skalierung korrekt funktioniert, kann man die Zeile self.xmax+=5.0 auskommentieren, so dass xmax bei jedem Aufruf erhöht wird.

Das Tkinter-Programm

```
def plotdata():
    x,y=datgen.getxy()
    pw.plotxy(x,y)

def clear():
    pw.clearplot()

if __name__ == "__main__":

    datgen = Generatetestdata()

    root = tk.Tk()

    mf= tk.Frame()
    pw=Plotwindow(mf,(200,150))
    mf.pack()


    bf=tk.Frame()
    b1=tk.Button(bf,text="Plot", command=plotdata)
    b1.pack(side=tk.LEFT)
    b2=tk.Button(bf,text="Clear", command=clear)
    b2.pack(side=tk.LEFT)
    bf.pack()

    root.mainloop()
```

Es werden 2 Funktionen zum Plotten und Löschen der Kurven definiert.
Diese sind mit den Buttons „Plot“ und „Clear“ verbunden.

Die Plot-Funktion erzeugt bei jedem Aktivieren neue Daten mit dem Testgenerator.

So ergibt sich ein Bild wie oben.

Anhang: Auszüge aus der Matplotlib-Dokumentation

Die Clear-Funktionen

https://stackoverflow.com/questions/8213522/when-to-use-cla-clf-or-close-for-clearing-a-plot-in-matplotlib

`plt.cla()` clears an axes, i.e. the currently active axes in the current figure. It leaves the other axes untouched.

`plt.clf()` clears the entire current figure with all its axes, but leaves the window opened, such that it may be reused for other plots.

`plt.close()` closes a window, which will be the current window, if not specified otherwise.

Additionally, the `Figure` class provides methods for clearing figures. I'll assume in the following that `fig` is an instance of a `Figure`:

`fig.clf()` clears the entire figure. This call is equivalent to `plt.clf()` only if `fig` is the current figure.

`fig.clear()` is a synonym for `fig.clf()`

Note that even `del fig` will not close the associated figure window. As far as I know the only way to close a figure window is using `plt.close(fig)` as described above.

Definitionen zum Diagramm

`Figure`

The **whole** figure. The figure keeps track of all the child `Axes`, a smattering of 'special' artists (titles, figure legends, etc), and the **canvas**. (Don't worry too much about the canvas, it is crucial as it is the object that actually does the drawing to get you your plot, but as the user it is more-or-less invisible to you). A figure can have any number of `Axes`, but to be useful should have at least one.

The easiest way to create a new figure is with pyplot:

```
fig = plt.figure()  # an empty figure with no axes
fig, ax_lst = plt.subplots(2, 2)  # a figure with a 2x2 grid of Axes
```

`Axes`

This is what you think of as 'a plot', it is the region of the image with the data space (marked as the inner blue box). A given figure can contain many Axes, but a given `Axes` object can only be in one `Figure`. The Axes contains two (or three in the case of 3D) `Axis` objects (be aware of the difference between **Axes** and **Axis**) which take care of the data limits (the data limits can also be controlled via set via the `set_xlim()` and `set_ylim()` `Axes` methods). Each `Axes` has a title

(set via `set_title()`), an x-label (set via `set_xlabel()`), and a y-label set via `set_ylabel()`).

The `Axes` class and it's member functions are the primary entry point to working with the OO interface.

`Axis`

These are the number-line-like objects (circled in green). They take care of setting the graph limits and generating the ticks (the marks on the axis) and ticklabels (strings labeling the ticks). The location of the ticks is determined by a `Locator` object and the ticklabel strings are formatted by a `Formatter`. The combination of the correct `Locator` and `Formatter` gives very fine control over the tick locations and labels.

`Artist`

Basically everything you can see on the figure is an artist (even the `Figure`, `Axes`, and `Axis` objects). This includes `Text` objects, `Line2D` objects, `collection` objects, `Patch` objects ... (you get the idea). When the figure is rendered, all of the artists are drawn to the **canvas**. Most Artists are tied to an Axes; such an Artist cannot be shared by multiple Axes, or moved from one to another.

Einbetten in Tkinter

https://pythonprogramming.net/how-to-embed-matplotlib-graph-tkinter-gui/