Using the Matplotlib Library in Python 3

Matplotlib is a Python 2D plotting library that produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, error charts, scatterplots, etc., with just a few lines of code. Matplotlib is the premier Python 2D plotting package used by scientists and engineers all over the world.

Matplotlib's developer John Hunter decided to base its look and feel on MATLAB's plotting routines. Therefore, if you know a little MATLAB, many aspects of Matplotlib will look familiar.

This brief introduction is to get you started so you can read and use the online documentation.

1. **First import the proper modules.**

Include near the top of your Python program the following import statements.

```
import numpy as np
import matplotlib.pyplot as plt
```

Notice the use of the "`as`" clause that allows you to supply `plt` as a shorten form of `matplotlib`. That way you can write, for example, `plt.plot(x,y)` instead of `matplotlib.plot(x, y)`.

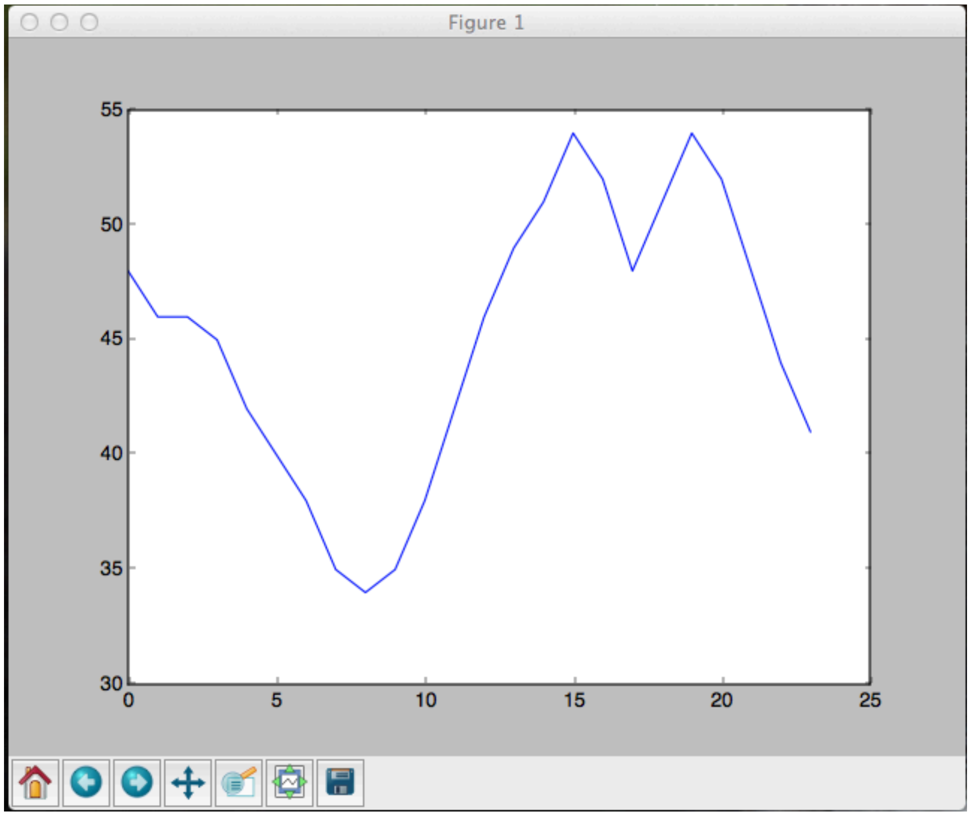
It is considered good style to supply the module name with each method call to alert the reader to the module the method belongs. Using "`from module import *`" style of import that we have used in class is frowned upon by professional programmers. And we expect you to use the above import forms.

NumPy is a high quality numerical package for scientific computing with Python. NumPy's website is at http://www.numpy.org/ Matplotlib uses NumPy extensively and when you install Matplotlib on your labtop, NumPy is installed along with it.

2. A simple Python program using Matplotlib libraries

```
''' Python 3 program to draw a simple line graph using matplotlib - plot1.py'''

import numpy as np
import matplotlib.pyplot as plt

def simpleLinePlot():

    y = [48, 46, 46, 45, 42, 40, 38, 35, 34, 35, 38, 42, \
          46, 49, 51, 54, 52, 48, 51, 54, 52, 48, 44, 41]

    x = range(len(y))

    plt.plot(x, y)

    plt.show()

#run the function
simpleLinePlot()
```

Running the above program produces the following graph.



The `plot()` method requires two lists of equal length, in this case **x** and **y**. The `plot()` method plots a line drawing the best it can by using its defaults values. The x-axis from 0 to 25 and the y-axis from 30 to 55 are automatically determined by the data. Probably not what you want. Typically, you specify the ranges on the axes, add labels, a title, and other features before you make the graph visible by calling `plt.show()`.

2. Plot Design

The goal of a plot is to express a message, i.e., tell a story. The above plot is poor in that it conveys little or no message! In fact, it has several misleading elements. People expect the y-axis of graphs to start at zero. Why does the y-axis start at 30? Also, a line graph implies to the viewer that this is a continuous function when in fact it is a collection of temperature readings at each hour. The data should be plotted as points. Since we want plots that convey a message, we have work to do to improve the presentation.

3. Plotting points, specifying axes, adding title, labeling axes

The second program attempts to fix some of the above-mentioned short comings. It plots points, starts the y-axis at zero, adds labels on the x and y axes, and adds a meaningful title. Notice it uses an optional third format string argument in the `plot()` method. The letters and symbols of the format string are from MATLAB, and you concatenate a color string with a line style string. The default format string is 'b-', which is a solid blue line. The string 'r*' in our program means to plot points that look like a star and use red for the color. See http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.plot for the `plot()` command's details on how to use string characters that control the line style or marker type.

```
''' Python 3 program to draw a simple plot of temperatures using matplotlib - plot2.py'''

import numpy as np
import matplotlib.pyplot as plt

def simpleLinePlot():

    y = [48, 46, 46, 45, 42, 40, 38, 35, 34, 35, 38, 42, \
          46, 49, 51, 54, 52, 48, 51, 54, 52, 48, 44, 41]

    x = range(len(y))

    plt.figure(2)

    plt.xlim([0, 24])
    plt.ylim([0, 60])
    plt.xlabel('Air Temperature for each hour of day')
    plt.ylabel('Degrees in Fahrenheit')
    plt.title('Air Temperature on April 4, 2016')

    plt.plot(x, y, 'r*')

    plt.show()

#run the function
simpleLinePlot()
```

Our second version of the program produces the following plot.

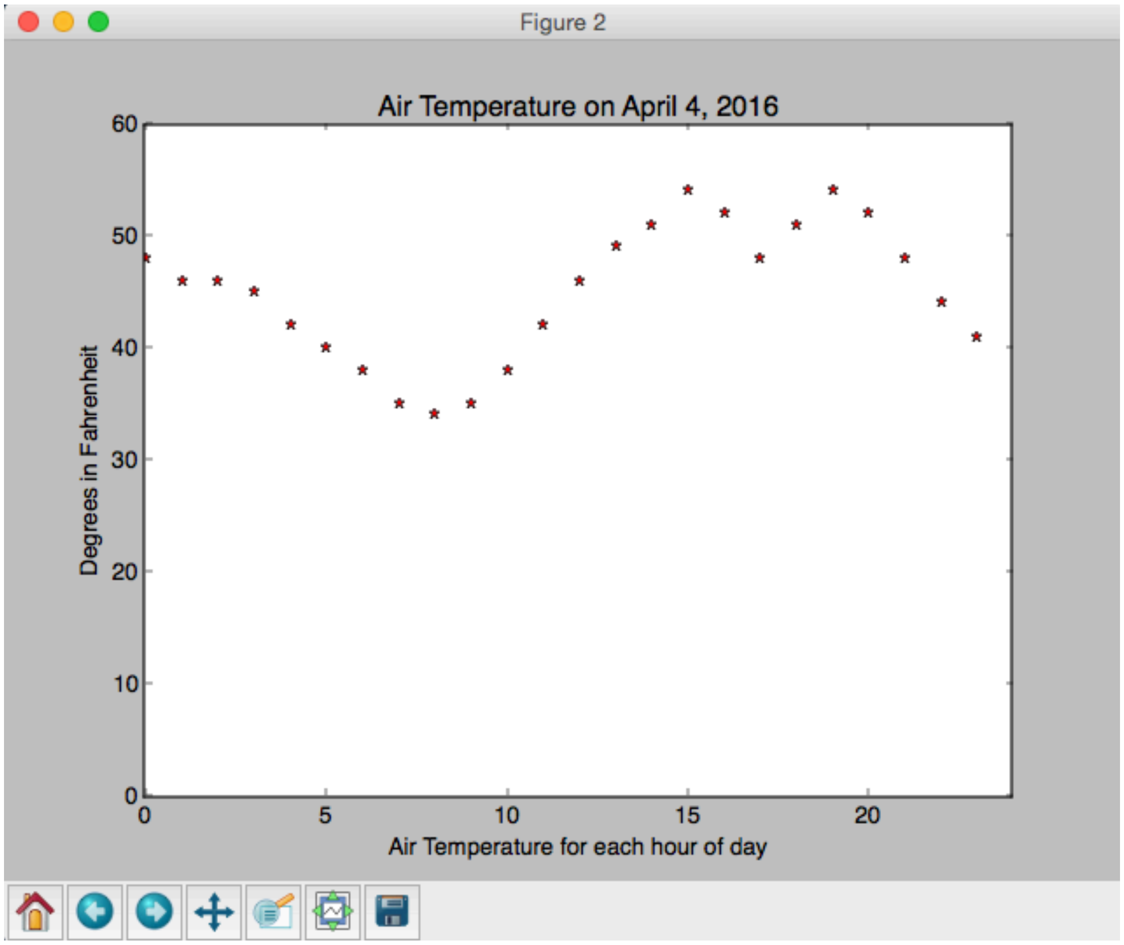


Figure 2
Air Temperature on April 4, 2016
Degrees in Fahrenheit
0
10
20
30
40
50
60
0
5
10
15
20
Air Temperature for each hour of day

The second plot is a big improvement but it's still confusing on what the x-axis means. What hours?

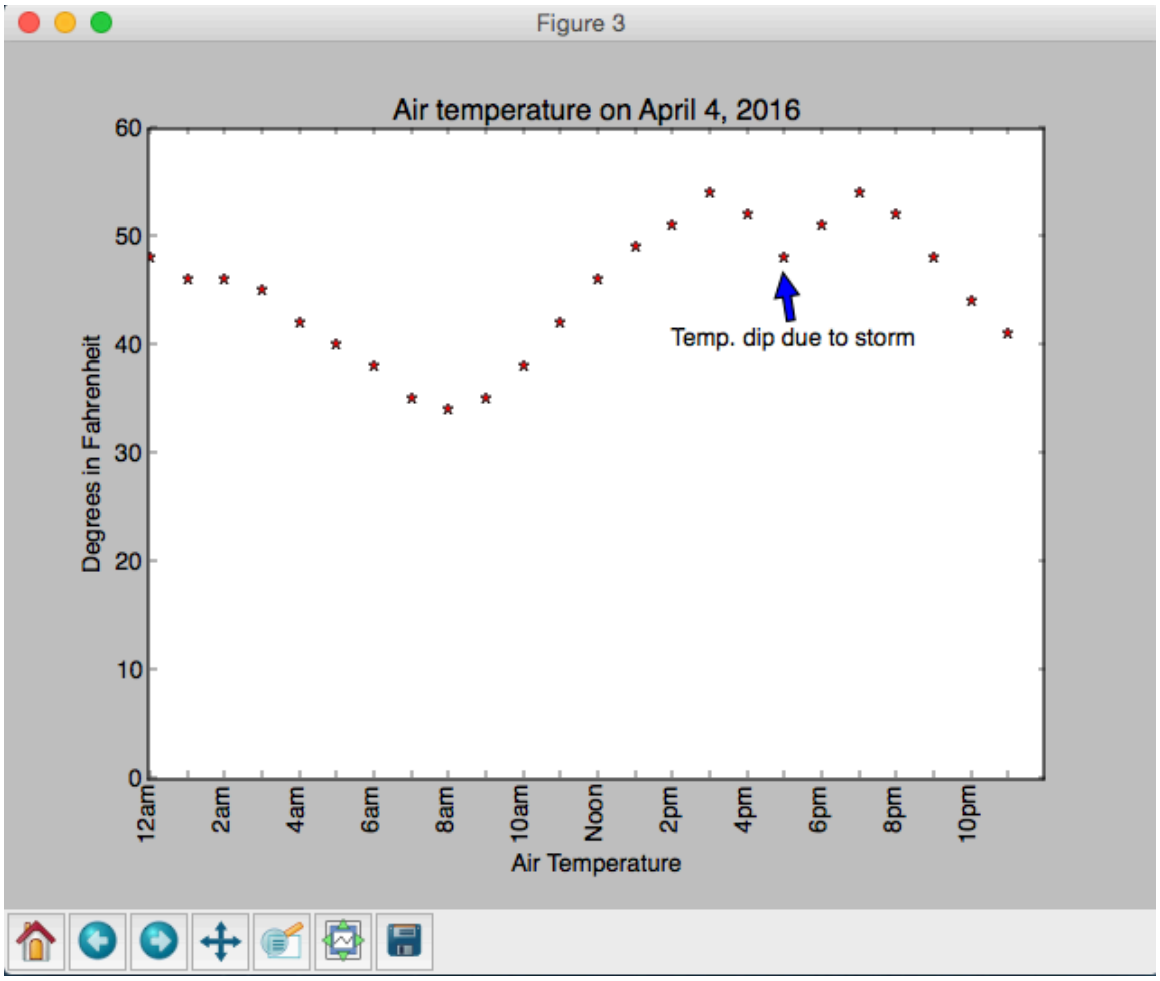
Where is midnight and noon? Why the dip around 17? The story is incomplete!

4. Adding labels on tick marks and text with arrow on the plot

The third version of our program tries to fill in the gaps in the story.

```python
''' Python 3 program to draw a simple plot of temperatures using matplotlib - plot3.py '''

import numpy as np
import matplotlib.pyplot as plt

def simpleLinePlot():

    y = [48, 46, 46, 45, 42, 40, 38, 35, 34, 35, 38, 42, \
         46, 49, 51, 54, 52, 48, 51, 54, 52, 48, 44, 41]

    x = range(len(y))

    plt.figure(3)

    # labels for tick mark on x-axis
    tickLabels = ['12am', ' ', '2am', ' ', '4am', ' ', '6am',\
    ' ', '8am', ' ', '10am', ' ', 'Noon', ' ', '2pm',\
    ' ', '4pm', ' ', '6pm', ' ', '8pm', ' ', '10pm',' ']

    # You can specify a rotation for the tick labels in degrees or with keywords.
    plt.xticks(x, tickLabels, rotation='vertical')
    # Pad margins so that markers don't get clipped by the axes
    plt.margins(0.2)
    # Tweak spacing to prevent clipping of tick-labels
    plt.subplots_adjust(bottom=0.15)

    # xy is position of arrow point; xytext is start of text; in problem coordinates.
    plt.annotate('Temp. dip due to storm', xy=(17, 47), xytext=(14, 40),
            arrowprops=dict(facecolor='blue', shrink=0.05))

    plt.xlim([0, 24])
    plt.ylim([0, 60])
    plt.xlabel('Air Temperature')
    plt.ylabel('Degrees in Fahrenheit')
    plt.title('Air temperature on April 4, 2016')

    plt.plot(x, y, 'r*')

    plt.show()

#run the function
simpleLinePlot()
```

The third program produces the following plot.



In Figure 3, we have added labels on the x-axis tick marks to specify the hour for the reading. When I added one for every hour, the bottom looked crowded so I added twenty-four tick labels with every other one just a space. We added an explanation for the dip in temperature with a blue arrow pointing to the event. Notice that we simplified the x-axis label as the longer text is no longer needed.

Now we have a graphic that tells a story. On April 4, 2016, the midnight air temperature gradually drops from 48 degrees to about 34 degrees by 8 am when the sun gradually warms the air back up to 54 degrees. In the late afternoon, a storm cools the air down four or five degrees but after the storm the temperature rebounds back to where it was before the storm. After 8 pm the air gradually cools down to 41 by midnight.

5. Plot multiple functions on same plot; use NumPy library; add a legend; add a grid

In this new example, we show how to plot more than one function on a graph and how to handle continuous functions such as sine(x).

When in the continuous arena, it's a good idea to utilize NumPy routines as much as possible. NumPy is a high quality numerical package developed by some of the world's best numerical analysts.

One routine that is particularly useful is `np.linspace()`. It means "linear space" and returns *num* evenly spaced samples, calculated over the interval [*start*, *stop*]. Default value for *num* is 50.

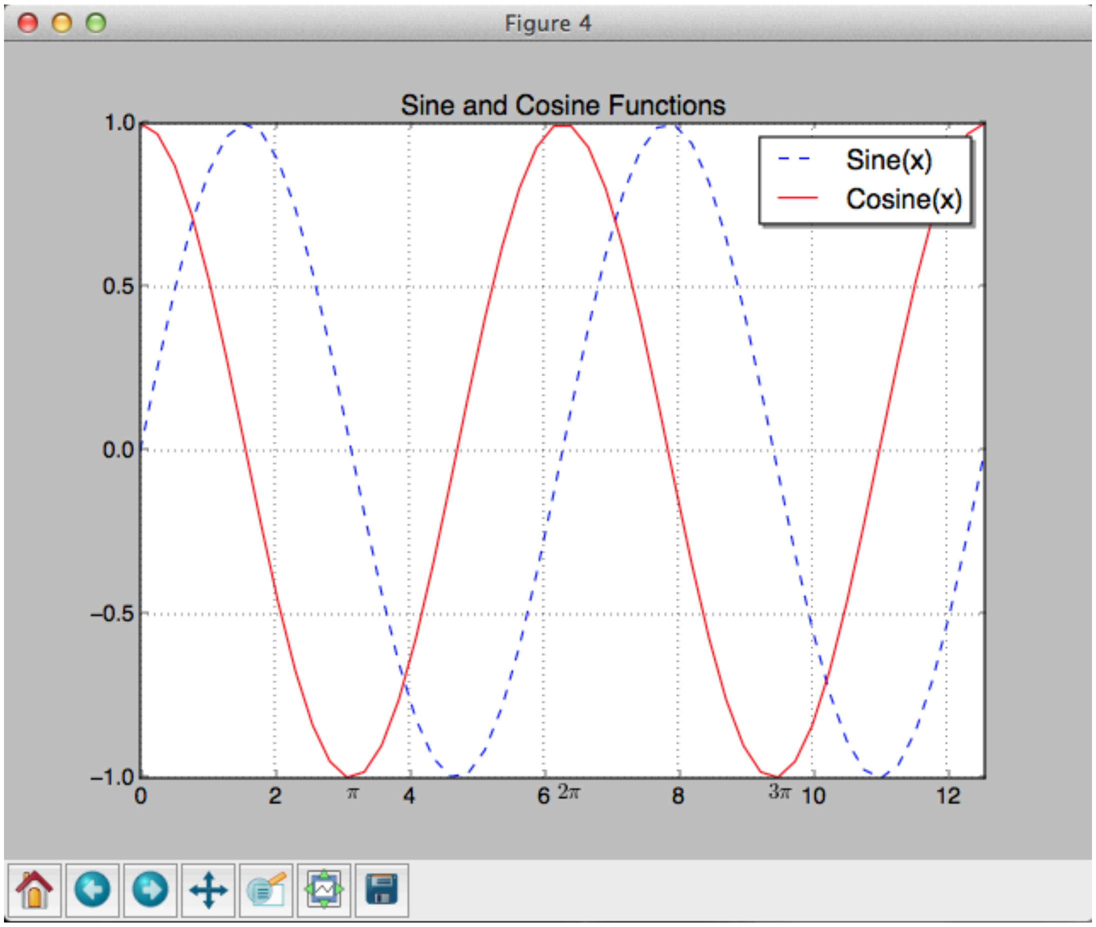
```
x = np.linspace(start, stop, num=50)
```

```
''' Python 3 program using matplotlib to draw plot of multiple functions,
    in this case, sine and cosine functions  - plot4.py '''

import numpy as np
import matplotlib.pyplot as plt

def multipleLinePlot():

    plt.figure(4)

    x = np.linspace(0, 4*np.pi)

    y1 = np.sin(x)
    y2 = np.cos(x)

   # Add a legend
    plt.plot(x, y1, '--', label='Sine(x)')   # dashed blue line
    plt.plot(x, y2, 'r-', label='Cosine(x)') # solid red line
    plt.legend(loc = 'upper right', shadow = True)

    plt.xlim([0, 4*np.pi])
    plt.title('Sine and Cosine Functions')

    # text just below the x-axis for pi, 2pi, 3pi
    plt.text(np.pi-.1, -1.07, r'$\pi$')
    plt.text(2*np.pi-.1, -1.07, r'$2\pi$')
    plt.text(3*np.pi-.1, -1.07, r'$3\pi$')

    # Add a grid
    plt.grid()

    plt.show()

#run the function
multipleLinePlot()
```

The line `x = np.linspace(0, 4*np.pi)` above generates a list structure (called a NumPy array) of 50 evenly spaced float values from 0 to 4 pi. Notice that we used the very accurate `np.pi` for the value of pi. The line `y1 = np.sin(x)` produces a list of 50 values for sin(x) from 0 to 4 pi that we use in the current plot. A second assignment for y2 produces 50 values of cos(x) that is drawn on the same current plot. Notice the legend in the right hand corner and the grid.

To demonstrate how to place text anywhere on the graphic, I used three calls to the `plt.text()` method to place the Greek letter for pi, 2pi, and 3pi just below the x-axis. The x and y values for the start of the text are in problem coordinates, e.g., the x value for first one is a shade before pi and the y value is a shade less than -1. The `r'$\pi$'` is a way to access the symbol for pi in the Latex library, a popular mathematical-oriented text processing language.

Running the above program generates Figure 4.



6. Read Pyplot Tutorial and look at Examples and Gallery

Read thru the **Pyplot Tutorial** at http://matplotlib.org/users/pyplot_tutorial.html skipping the two sections "Controlling line properties" and "Working with multiple figures and axes." Start reading again at section "Working with text" and read to the end of web page.

Browse through the **Gallery** at http://matplotlib.org/gallery.html and click on any thumbnail image to see the full image and the Python code. You can copy and paste any of the code and run it in IDLE. Or you can click on Source Code in any example and Matplotlib will automatically download the code, open IDLE if necessary and you are ready to run the code. Pretty neat! Try it. On some computers you might have to double click on the downloaded file icon.

A good way to develop your own program is to find the closest image to what you want, copy and paste the code or click on Source Code then add different features from different examples.

Browse **Examples** at http://matplotlib.org/examples/index.html and again click on an item to see image and code. Again you can click on Source Code to download the source code to IDLE.

Browse the **List of plotting commands** at http://matplotlib.org/api/pyplot_summary.html for details on each plot command.

7. Resources

Matplotlib's home page is http://matplotlib.org/

For a sampling of what Matplotlib can do, see the **Screenshots** at http://matplotlib.org/users/screenshots.html, **Gallery**, and **Examples** web pages listed above in Section 6.

Trying to learn how to do a particular kind of plot? Check out the **Gallery**, **Examples**, or the **List of plotting commands** listed above in Section 6.