PHYS-4007/5007: Computational Physics

Python Tutorial
Creating Plots in Python

1 Introduction: Download the Sample Codes

Log into your Linux account and open the web browser. Open the course web page at:

http://faculty.etsu.edu/lutter/courses/phys4007/

Scroll down to the Useful Python Programs web page and click the link. Once on this Python web page, click on the "myplot1.py" link in the table under the Python Plotting Tutorials and Programs section heading. This will show the program in the web browser GUI.

Now, open a terminal window and change directories to your python subdirectory. Open a new file called "myplot1.py" (without the double quotes) with emacs at the Linux prompt:

```
emacs myplot1.py &
```

where the ampersand symbol (&) puts the emacs session in background so that you can still enter commands at the Linux prompt. Go back to the web browser GUI highlight all of the text in the GUI, and copy and paste this program into the emacs GUI. Finally, save this file and exit the emacs GUI.

Go back to the Python Plotting Tutorials and Programs web page and repeat the actions above for the "myplot2.py" file. Once you have saved this second file, exit that emacs GUI session and proceed to the next section below.

2 Examining These Two Python Codes

2.1 Python code: myplot1.py

Let's now examine each of these codes and highlight what they are doing. Note that both codes need to be run using python3. The first code, myplot1.py is shown below:

```
# This program will make plots in python, version 3+, using
# utilities from matplotlib and numpy.

# Always include these next two import commands.

import numpy as np
import matplotlib.pyplot as plt

# This shows how to define your own function.

def func(x):
    return np.sin(2*np.pi*x)

# ***** How to make a plot of data with error bars. ******

# Generate some fake data.

x1 = np.arange(10) + 2*np.random.randn(10)
y1 = np.arange(10) + 2*np.random.randn(10)

# Generate fake errors for the data.

x1err = 2*np.random.random(10)
y1err = 2*np.random.random(10)

# Make the plot.

plt.errorbar(x1, y1, xerr=x1err, yerr=y1err, fmt='bo')
plt.xlabel('X Data', labelpad=10)
plt.ylabel('Y Data', labelpad=10)
plt.title('Random Plot with Error Bars')
plt.show()

# ***** How to make a plot of multiple functions. *****

# Now make another plot with multiple plots on the same
# screen.

x2 = np.arange(1, 19, .4)
y2a = np.log10(x2)
y2b = 0.01 * x2**2
y2c = 0.9 * np.sin(x2)

plt.plot(x2, y2a, 'r-', label='y2a')
plt.plot(x2, y2b, 'b^', label='y2b')
```

```
plt.plot(x2, y2c, 'go', label='y2c')
plt.plot(x2, y2a+y2b+y2c, '+', label='y2a+y2b+y2c')
plt.legend(loc=2)
plt.show()

# ***** How to make more than one plot on a page. *****

# We'll make use of of the defined function 'func' from
# the beginning of this Python program.

x3a = np.arange(0.0, 4.0, 0.1)
x3b = np.arange(0.0, 4.0, 0.01)

y3a = func(x3a)
y3b = func(x3b)
y3an = y3a + 0.1*np.random.randn(len(x3a))

plt.figure()  # Initialize the figure space.

# Make 2 plots vertically + 1 horizontally.
# Start with the first (upper left).

plt.subplot(211)

plt.plot(x3a, y3a, 'bo', x3b, y3b, 'r:')

plt.subplot(212)  # Now the 2nd (lower left) plot.

plt.plot(x3a, y3an, 'bo', x3b, y3b, 'r:')

plt.show()
```

Remember that any text written after the pound-sign (#) symbol tells Python that the following text is a comment. The first two executable lines of the code imports the NumPy library and the pyplot utility from the Matplotlib library and relabels them as np and plt respectively. The NumPy library contains a variety of math functions and offers multidimensional array creation and manipulation functions. Matplotlib is a Python library that offers 2D and 3D plotting capabilities.

Next, this code shows the user how to define their own functions. In this case, the function func, loads an array stored in x and calculates the sine of 2π times the values stored in x. Note that the math sine function is located in the NumPy library.

Following this, the code creates two array variables using the NumPy arange function and random.randn function. Following this, the code makes two array variables containing random errors data using the NumPy random.random function:

- arange: Returns evenly spaced values within a given interval. Values are generated within the half-open interval [start, stop) (in other words, the interval including start but excluding stop). For integer arguments the function is equivalent to the Python built-in range function, but returns an ndarray rather than a list.
- random.randn: Returns a sample (or samples) from the "standard normal" distribution. If the argument is positive, int_like or int-convertible arguments are provided, randn generates an array of shape (d0, d1, ..., dn), filled with random floats sampled from a univariate"normal" (Gaussian) distribution of mean 0 and variance 1 (if any of the d_i are floats, they are first converted to integers by truncation). A single float randomly sampled from the distribution is returned if no argument is provided.
- random.random: The random module implements pseudo-random number generators for various distributions indicated by the attached function, in this case ".random". As such, 'random.random(N)' returns the next random floating point number in the range [N+0.0, N+1.0) for each N in the array of integers from 0 to N-1.

We next create the plot with the following Matplotlib PyPlot utilities:

- .errorbar(x, y, xerr=xerr, yerr=yerr, fmt=''): Plot an errorbar graph using the following data:
  - x: Scalar or an array of data contained in the independent variable.
  - y: Scalar or an array of data contained in the dependent variable. Note that the size of the array stored in y must be the same as that stored in x.
  - xerr: Scalar or an array of data containing the length of the error bars which are drawn horizontally at the +/-value relative to the data stored in x. Note that the size of the array stored in xerr must be the same as that stored in x. Default is None.
  - yerr: Scalar or an array of data containing the length of the error bars which are drawn vertically at the +/-value relative to the y data. Note that the size of the array stored in yerr must be the same as that stored in y. Default is None.
  - fmt='': Plot format string, optional, default: None. If fmt is none (case-insensitive), only the errorbars are plotted. The properties of the format string are identical to the defaults used for the plot() function (see below).
- Various plot labeling commands in Matplotlib PyPlot:

  - .text(x, y, s, fontsize=N, bbox=dict(facecolor='color')): Add text string s at an arbitrary location x & y using data coordinates. The fontsize keyword changes the size of the font. The bbox keyword will draw a box around the text, filling in the box with a color indicated in facecolor.
  - .xlabel(s): Add a label s (string) to the x-axis.
  - .ylabel(s): Add a label s (string) to the y-axis.
  - .title(s): Add a main title s (string) above the graph box.
  - .figtext(x, y, s): Similar to the text() using relative coordinates – x=0.0 (left side), 1.0 (right side); y=0.0 (bottom), y=1.0 (top).

- .show(): Create the plot on the computer screen.

The next part of this code will create a plot with multiple graphs drawn in the figure. We first make a set of independent variable data using the NumPy arange utility:

- .arange(start, stop, step): Return evenly spaced values within a given interval.
  - start: Start of interval (number, optional). The interval includes this value. The default start value is 0.
  - stop: End of interval (number). The interval does not include this value, except in some cases where step is not an integer and floating point round-off affects the length of out.
  - step: Spacing between values (number, optional. For any output 'out', this is the distance between two adjacent values, out[i+1] - out[i]. The default step size is 1. If step is specified, start must also be given.

We then make three dependent variable arrays, one using the NumPy common logarithm function, one squaring 'x', and one using the sine function from NumPy. Following this, we make 4 different curves on the plot, where the 4th curve is the sum of the three dependent variable arrays we just described. The Matplotlib PyPlot plot command has the following parameters that can be passed to it:

- .plot(x, y, string, label=' '): Plot lines and/or markers.
  - x: Array of numbers containing the independent data (optional).
  - y: Array of numbers containing the dependent data.
  - string: A string specifying the color, line style, or marker type of the curve. This string is usually a three-character field 'CMS', for color (Table 1), marker (Table 2), and line style (Table 2), respectively.

Table 1: Colors used in the plot() and errorbars() utilities.

character	**color**
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

Table 2: Markers and line styles used in the plot() and errorbars() utilities.

character	**description**
'-'	solid line style
'--'	dashed line style
'-.'	dash-dot line style
':'	dotted line style
'.'	point marker
','	pixel marker
'o'	circle marker
'v'	triangle_down marker
'∧'	triangle_up marker
'<'	triangle_left marker
'>'	triangle_right marker
'1'	tri_down marker
'2'	tri_up marker
'3'	tri_left marker
'4'	tri_right marker
's'	square marker
'p'	pentagon marker
'*'	star marker
'h'	hexagon1 marker
'H'	hexagon2 marker
'+'	plus marker
'x'	x marker
'D'	diamond marker
'd'	thin_diamond marker
'|'	vline marker
'_'	hline marker

– label='': Place text string in a separate legend box associating the text with the plot line color, marker, and style (optional).

Next we need to specify where to place the various legend labels that we passed to the .plot command, plt.legend(loc=2), where the loc keyword has for different locations: 1 (upper-right), 2 (upper-left), 3 (lower-left), and 4 (lower-right). We then display the plot with the plt.show() command.

Our third and final plot created with this code shows the user how to make two separate plots stacked on top of each other on the same page. Here we create two separate dependent variable arrays, x3a, x3b, we then create three separate dependent variable arrays, y3a, y3b, and y3an, using the defined function that we created near the top of the file.

When making two separate plots on the same page, we first need to let Python know that we plan to do this by issuing the plt.figure() command. Following this, we need to let Python know how we plan to arrange the individual plots with the plt.subplot(VHI) command, here V indicates how many plots there will be in the vertical direction, H indicates how many in the horizontal direction, and I is the marker for which plot we will make first. In our program here, we have plt.subplot(211), indicating that we are making 2 plots stacked in the vertical direction, one horizontally, and that we will start with the first top plot.

We once again use the plot command from the Matplotlib Pyplot library, however this time, we do something a little different by plotting two separate plots using the same plot command: plt.plot(x3a, y3a, 'bo', x3b, y3b, 'r:'). Here we plot y3a as a function of x3a using 'blue' circles, then we overplot y3b as a function of x3b using a dotted line. We then issue the command, plt.subplot(212) to let Python know that we will now make the second lower plot on this page. Once again, we make two plots using the same plot command. Finally, we tell Python to display this figure.

2.2 Python code: myplot2.py

This code makes simple plots, but shows how to manipulate main titles, x– and y–axis labels using LaTeX commands, and shows the user how to include and manipulate in-plot text. It also shows the user how to change the width of lines and change the size of the fonts used in the text.

```
# This program will make plots in python, version 3+, using
# utilities from matplotlib and numpy.  This program investigates
# the use of the 'rc' module of matplotlib.pyplot to change
# line widths and font sizes.  It also does examples to make
# use of LaTeX math symbols in text.
```

```
# Always include these next two import commands.

import numpy as np
import matplotlib.pyplot as plt

# ***** Make a plot with axis labels & in picture text. *****

x1 = np.arange(1, 19, .4)
y1a = np.log10(x1)
y1b = 0.01 * x1**2
y1c = 0.9 * np.sin(x1)

plt.plot(x1, y1a, 'r-')
plt.xlabel('X Data', labelpad=10)
plt.ylabel('Log(X)', labelpad=10)
plt.title('The Common Log Function')
plt.text(2.0, 1.2, 'This is an in-picture text.')

plt.show()

# This plot reproduces the plot above, but rotates the in
# picture text by 30 degrees.

plt.plot(x1, y1a, 'r-')
plt.xlabel('X Data', labelpad=10)
plt.ylabel('Log(X)', labelpad=10)
plt.title('The Common Log Function')
plt.text(2.0, 1.2, 'This is an in-picture text.', rotation=30)

plt.show()

# Now reproduce the first plot, but make the lines thicker
# (both the plot and axes) and font bigger of all of the
# text.

plt.rc('lines', linewidth=2)
plt.rc('font', size=16)
plt.rc('axes', linewidth=2)

plt.plot(x1, y1a, 'r-')
plt.xlabel('X Data', labelpad=10)
plt.ylabel('Log(X)', labelpad=10)
plt.title('The Common Log Function')
plt.text(2.0, 1.2, 'This is an in-picture text.')
```

```
plt.show()

# We will now include some LaTeX math commands in the
# various text captions.

plt.plot(x1, y1c, 'bo-')
plt.xlabel(r'Angle $\theta$')
plt.ylabel(r'sin $\theta$')
plt.title('The Sine Function')

plt.show()
```

This code will make a grand total of 4 plots. The first plot shows the common logarithm function. Note the labelpad keyword in the label commands [*i.e.*, plt.xlabel('X Data', labelpad=10) and plt.ylabel('Log(X)', labelpad=10)] — this keyword adds space between the axis label and the axis. This is needed sometimes if the axis label is too close to the numeric 'tick-mark' labels fo the axis. This plot also shows the user how to place text within the plot using the command plt.text(x, y, 'string'), where x and y are the position (in data coordinates) of the start of the string (*i.e.*, the 3rd entry in this command). Note that one could also use the .figtext command to place in-plot text using relative coordinates (se Page 5 for details).

The next figure shows how to write 'in-plot' text at an angle to the horizontal orientation using the rotation keyword in the .text command. Here the angle is in degrees, rotated in the counterclockwise direction (positive values) or clockwise direction (negative numbers).

In the third plot, we introduce ourselves to the rc utility in the Matplotlib Pyplot library. Note that with this utility, we can carry out a variety of changes to line thicknesses and text appearances. Note that we could also have made these changes with keywords in the variety of plotting and labeling commands, but we can make these changes universally within a code for all plots created in a given code. These commands make the following changes:

- plt.rc('lines', linewidth=2): Double the thickness of all lines drawn in a plot.
- plt.rc('font', size=16): This follows the font sizes use in LaTeX. The default size is 12 point, here we will use 16 point to increase the size of all of the text (*i.e.*, titles, axis labels, tick labels, and in-plot text).
- plt.rc('axes', linewidth=2): Double the thickness of the axes lines.

The 4th and final plot made from this code makes a plot of the sine function and uses LaTeX math symbols in the axes labels. This is done by including the letter 'r' before the string to

be printed and enclosing the LaTeX command between the dollar sign ($) symbols, similar to way we include equations inside paragraphs in a LaTeX file.

3 Running These Two Python Codes

For the two codes that you have saved on your Linux account, make sure that you are in the subdirectory where you saved these files and enter the following from the Linux prompt:

```
python3 myplot1.py
```

You will see a GUI pop up containing your plot. At the bottom of this GUI, you will see 7 control buttons that will do a variety of different options to this figure. Here we will just use the last button which will allow us to make a 'hardcopy' version of this figure. Click this button now. You will now see a second GUI pop up with the title "Save the figure". Follow this sequence to make an encapsulated postscript file of this figure:

- From the window containing all of the subdirectories in your login directory, scroll over to the directory where you want to save this figure file (typically this will be the directory where your Python code is located and double clikck that directory.
- Note that the Potable Network Graphics (*.png) format is the default for saved picture files. Go to the Filies of type: pulldown menu bar and select the second entry, Encapsulated Postscript (*.eps).
- Note however that the filename is still listed as a '.png' file. Change this name to figure_1.eps and click the Save button.
- Following this, quit this plot GUI by clicking the red-x button on the top-left of this GUI.

Once that GUI disappears, a new plot GUI will appear with the second plot. Repeat as we did above making sure that your new filename is unique from the already created file. If you don't come up with a unique name, a GUI will pop up asking if you wish to overwrite the previous file. Repeat once again for the third plot.

Now run the second code with

```
python3 myplot2.py
```

Again, making sure that you have unique filenames for all of the '.eps' files you have created.

Once you have finished running these two codes, you can view these files on the computer screen by using the Linux Ghostview command:

```
gv figure_1.eps
```

From the Ghostview GUI, you can carry out a variety of options pertaining to the figure in the GUI. For now, just quit the GUI by depressing the 'File' button at the top left of the GUI and scroll down to 'Quit' and release the mouse button (note, keep depressing the right mouse button until you have selected the 'Quit' option).

Feel free to investigate all of the figures that you have saved to your subdirectory.

4 Make Your Own Code From Scratch

Using the information contained in these two sample codes, make your own code that will do the following:

- Create a plot of the function

$$y = \sin(x) * \exp^{(-x/100)} ,$$

  where x is an array containing 10,000 elements from 0 (zero) to 999.

- Include axes labels and a main plot title.
- Include an in-plot text string containing the equation above written using LaTeX math symbols.

Finally, when the Python plot GUI appears, save this figure as a regular postscript file (*i.e.*, filename ending in '.ps'). When done, print this postscript file out using the Linux 'lpr' command. I'll give instructions for doing this on the board in class.