Skill-05-Basic-Plotting-Student

September 12, 2024

0.1 # Skill Homework #5 - Basic Plotting

1 Plotting with `matplotlib`

The python module `matplotlib` is a very widely used package. It is based on the plotting in the commercial program `Matlab`. There are several way to import and use `matplotlib`. We will use what is called the *object-oriented* approach. You can see in the cell below how the module is imported in this approach.

The line `%matplotlib inline` tells jupyter to display the plots in this web browser page.

Run the cell below to read in the `numpy` and `matplotlib` modules and set up the inline plots.

To get you started I put in some code to create some data.

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
#
# Now make some data
np.random.seed(20150108) # seed the random number generator so eveyone has the same data
time = np.linspace(0, 2, 101)
volts = 2.5 * np.sin(2 * np.pi * time + 1.0) + np.random.normal(0,0.1,len(time))
vModel = 2.5 * np.sin(2 * np.pi * time + 1.0)
```

[ ]:

2 Quick Plots

Quick and dirty plots can be made with the one line of code:

```
plt.plot(time, volts)
```

You may gave noticed the cell prints out annoying stuff above the plot. To avoid this make the last line in the cell `pass`. This is a NOP (No OPeration) command in python. Go ahead and run this in the cell below.

[ ]:

2.1 Dressing up the Plot

You need to learn to embellish the plot to make it minimall acceptable. Every plot needs two things: a title, and axis labels with units. These are added with `plt`.

`ax.set_xlabel("myXLabel")`

sets `"myXLabel"` as the x label. Your label should have a one or two word description followed by the units in parentheses, like `Time (s)`. Similarly, the method for adding a y label is

`ax.set_ylabel(FILL THIS IN)`

To add a title, the method is

`ax.set_title(GIVE ME A TITLE)`

Give the plot the title `Sensor Output`.

Unfortunately, jupyter can't modify the plot in a previous cell. So, in the cell below, copy and paste the code, then add the code for adding the labels and title.

[ ]:

3 Multiple Lines on Plot

3.1 Symbols, Colors, and Line Types

One problem with the above plot is that since the data is not smooth, it should be plotted with dots or symbols. Data should almost always be plotted with dots or symbols. `matplotlib`, like `Matlab`, does this with a formatting parameterin the `plot` command. You can append and optional third parameter after the y array with formatting information about the plot line. This formatting is a very compact string with the line type, the symbol, and the color encoded in a three character string. Here are some examples: * `'-k'` - plots only a black line. (`k` means blac***k***.) * `'.r'` - plots only dots in the color red * `'--g'` - plots only a green dashed line * `'ob'` - plots only circles in blue * `'^:m'` - plots triangle connected by a dotted line in magenta * `help(plt.plot)` is always available with more options and details

Other colors are `y` and `c`. Other symbols are `v`, `*`, `+`, `x`, `'s'`, and `d`. Another line types is `'-.'`. Your plot should have a line like

`plt.plot(time, volts, '.b')`

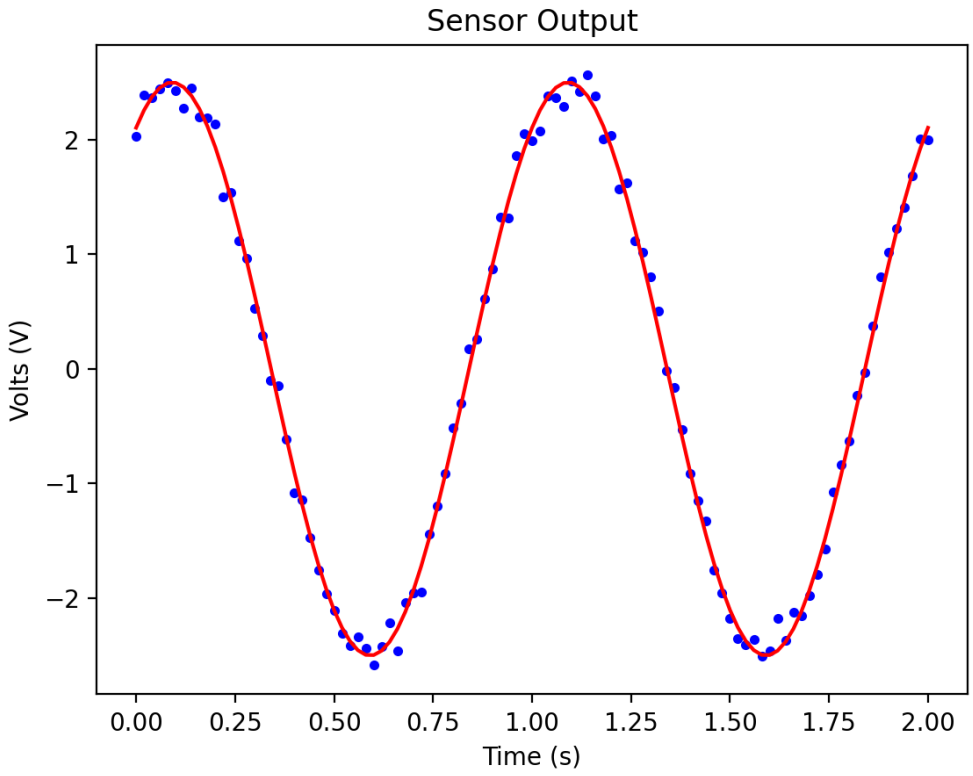
and another line to plot model as a solid red line.

3.2 Adding More Lines to a Plot

Adding another line (or set of symbols) to the plot is just a matter of adding another `plt.plot` call. Each sucessive call of `plot` will add another line to the axis.

Copy and paste your previous code in the cell below. Then modify it to plot the data points in blue dots without a line, and the model `vModel` as a red smooth line.

3.3 It Should Look Like This ...



If yours does not, keep trying until it does.

[ ]:

4 Legends and a Grid

The next two modifications are optional. Often it is useful to have background grid on your plots. This is done by calling `plt.grid()`.

Next, sometimes you are plotting multiple data sets or models on the graph, so you need a *legend* to help the reader decode what you plotted. There are two steps to doing this. 1. In each call to `plot`, add a parameter like `label="Data"`. Whatever string you assign to `label` will be used in making a legend. 2. Add the call

```
plt.legend()
```

after your last call to `plot`. This turns on the legend.

In the cell below, add a grid and legend to your plot.

[ ]:

5 Setting Axis Limits

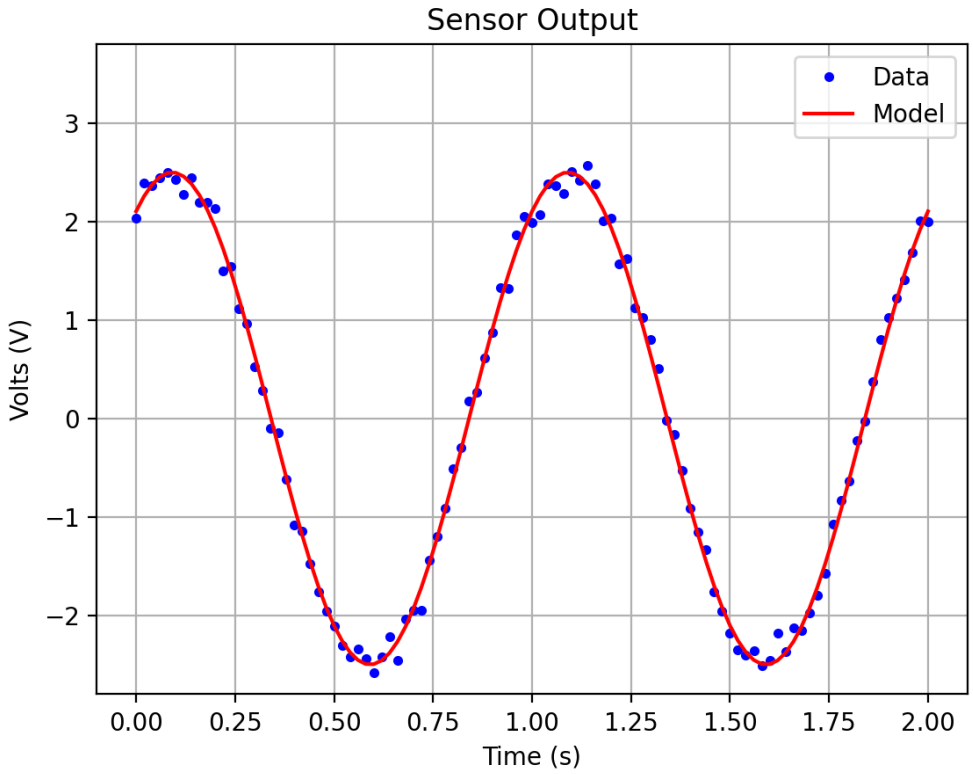
Usually `matplotlib` chooses the axis limits well, but sometimes the default limits need to be adjusted. The methods

```
plt.xlim(min,max)
plt.ylim(min,max)
```

set the limits to your choice. For example setting the y limits from -2.8 to 3.8 leaves better room for the legend box.

Copy and paste your previous code and add both x and y limits to the plot. Also expanding the x limits a little bit, like 0.1 could show better where the data begins and ends.

6 Your Best Shot



It should look like the figure above.

[ ]:

Additional Topics

7 Logarithmic Axes

Sometimes your data or models are shown better with a logarithmic axis for one or both axes. The methods

```
plt.yscale('log')
plt.xscale('log')
```

change the scaling on the respective axes.

Use the following code in the cell below to generate more data

```
volts2 = 3 * np.exp(-time/0.25)
```

Then plot it using a logarithmic y-axis.

[ ]:

7.1 Log-Log Plot

Now to demonstrate the log-log plot, create the following data:

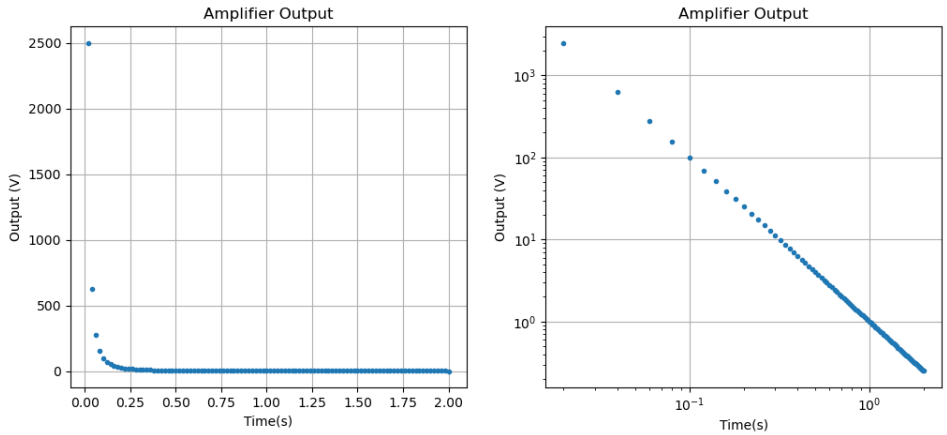
```
time3 = time[1:]
volts3 = time3 ** -2
```

We want to plot this data in plot linear and log-log axis scaling. The example below does some fancy figure handling. 1. I set the `figsize` manually to be extra wide by using

```
fig = plt.figure(figsize=(12,5))
```

2. I add a subplot axis using `121`. This means there will be one *row* amd two *columns* of plots, and this plot will be the first.
3. Use the line `fig.subplot(121)`
4. Do all of the plotting and labeling for figure 1
5.
6. The next set of axes are the second one, so it will be in the second columns. Note the first two numbers have to match.
7. Before starting the second plot, enter the line, `plt.subplot(122)`
8. Do all of the plotting and labeling for the second plot.

8 Plot



Keep trying until it looks like the above plots.

[ ]:

---

9 Saving Figures

Saving plots as graphics is a simple call to the method

`fig.savefig(fileName, dpi=dotsPerInch)`

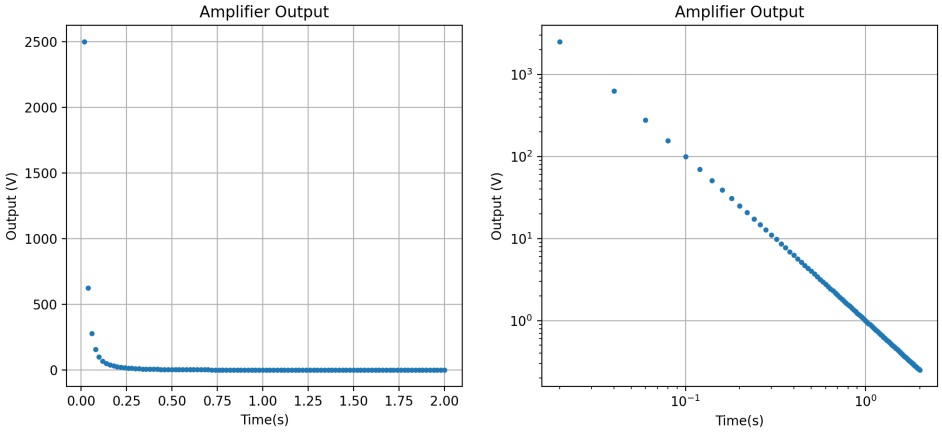
where the argument `dpi=dotsPerInch` is optional. The type of file produced is determined by the extension of the file name. Here are the file types I suggest: 1. `.png` with `dpi=200` - this makes a nice large file you can include in any document. The quality comes out better than if you use a `.jpg` file. 2. `.svg` with not `dpi` - this is a *vector* image file. Everything is describes by coordinates and sizes. This file scale to differnt sizes very well. This is a good choice if you are making a poster with large figures. 3. `.pdf` with `dpi=200` - sometimes this format works better with LaTeX documents.

I usually use `.png` format files. In the cell below save the previous pair of plots in this format.

BTW, jupyter remembers the previous plot, so you can just run the `savefig` code.

[ ]:
```
fig.savefig("test.png", dpi=200)
```

** Double Click Here to edit this hidden cell ** Here is your figure in this web page. Edit the name to match your file name. Then execute this cell by typing `<shift><enter>`.



---

10 More Plots

11 Histograms

Histograms are a powerful way of graphing data that has randomness around some value. It shows how the data are randomly distributed. The x-axis are the data values, and the y-axis is a bar graph of how many times each value occurs.

The code below simulates measuring a lenth of 5 cm ten thousand times with an uncertainty of 0.1 cm.

```
np.random.seed(0)
meas1 = np.random.normal(5.0, 0.1, 10000)
```

The basic histogram method is (after you have made a figure and an axis) is `plt.hist(data)`, where `data` are the numbers you want to make a histogram of.

Create a figure and axes, and plot this histogram.

[ ]:

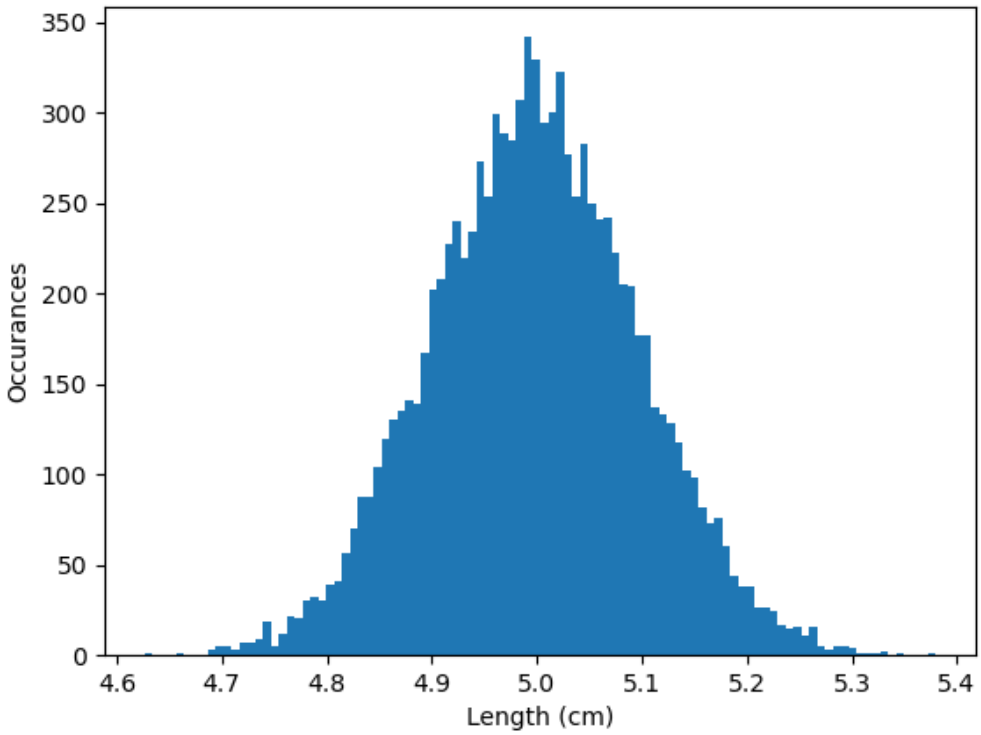
You might note that this distribution looks like the normal (or Gaussian) distribrution you have seen before.

12 Number of Bins

The most common modification to the default hostogram plot is to change the bins on the x-axis. One way is to add more bins by adding a second argument rifht after your data that is an integer setting the number of bins.

Replot the above data with 100 bins. When you do, you should see more small tails in the histogram and it will look more like the classical normal distribution.

13 Make It Look Like This ...



[ ]:

Finally, there is an optional argument `log=True` that make the y-axis logarithmic. In the cell below, plot the above histogram with the log scale turned on.

[ ]:

Finally you can provide a list of bin boundaries as an array. The following example simulates the rolling of a pair of dice. You know there only 11 possible values from "snake eyes" or 2 to "boxcars" or 12. The default `hist` doesn't get his right without helping it out. This examples feeds it the histogram boundries.

Finally, this example demonstrates the `normed` keyword that normalized the numbers so, like a probability density, the sum of the histogram heights adds up to 1.0.

[ ]:

14 Other Miscellaneous Topics

Here are some other topic you can explore on your own. I go to a web browser and google `matplotlib whatever` to find things out or refresh my memory.

14.1 More Plot Options

- `lw` - line width
- `ms` - marker size
- `mfc` - marker face color
- `mec` - marker edge color
- `mew` - marker edge width

14.2 More Colors

- names - 'red', etc. But also X11 colors like 'DodgeBlue'
- hex - '#ff8022' in rrggbb format
- RGB tuples - like `color=(0,0,1)`, rgb values range from 0 to 1
- RGBA tuple - like `color=(0,1,.5,.3)` where A is transparency

14.3 Adding Text Labels to Plots

The basic call is `plt.text(x,y,string)`

[ ]: