Tutorial to Python, Numpy, and Matplotlib

Background

We are going to be using the Python 3 data analysis ecosystem for many of the projects and homework in this class. When you get good at Python, it becomes an indispensable tool. Python itself is a high-level, interpreted general-purpose programming language. It becomes useful for data analysis through add-on libraries such as NumPy (Numerical Python), matplotlib (a plotting and visualization library), and many others. Anaconda is a cross-platform (Windows, Mac, and Linux) distribution and software environment manager that includes almost all of the major data analysis tools that you will need. Python and its data analysis libraries are free and open source but can often provide the same capabilities as the commercial product MATLAB.

There are many different ways to run python code. In this class, we are going to focus on using the Jupyter Lab / Jupyter Notebook environment. This is what you are using right now. It provides a friendly, interactive interface to python while giving you the ability to add-in formatted notes and comments. This is done through markdown cells; you can get a basic syntax guide here: https://www.markdownguide.org/basic-syntax/. You can even write equations directly in the markdown cells with $\LaTeX$. See here for documentation on $\LaTeX$ in Jupyter Notebooks.

Jupyter Notebooks are ideal for prototyping of your own ideas and code, sharing ideas with others, and submitting homework. As you need to do more complcated analysese, you may eventually move to actual python scripts and programming.

Learning Objectives

The learning objectives of this tutorial are:

1. Become familiar with the fundamentals of Jupyter Notebook and the Python Language.
2. Learn the fundamentals of NumPy and Matplotlib
3. Place these skills in the context of the plotting a parabola

Imports

In [ ]:

```
# In this cell, we import the libraries and modules we need for our analysis.
# It is good practice to keep all of the imports for a piece of code at the top
# of the file. Note that we can import packages "as" a particular variable. This
# allows us to use a shortcut/abbreviation when refering to that package later
# in the script.

import numpy as np
import matplotlib.pyplot as plt

# The following line tells matplotlib to plot directly in the Jupyter Notebook,
```

```
# opposed to creating a separate window.
%matplotlib inline
```

In [ ]:
```
plt.rcParams['font.size']       = 18
plt.rcParams['axes.grid']       = True
plt.rcParams['axes.xmargin']    = 0
plt.rcParams['axes.ymargin']    = 0
plt.rcParams['grid.linestyle'] = ':'
plt.rcParams['figure.figsize'] = (8,6)
```

Plotting with NumPy and Matplotlib

To introduce NumPy and Matplotlib, we are going to work with something we will all be familiar with: parabolas!

Plotting a parabola

To start, we will start with plotting a parabola. While it is not possible for the computer to actually store any type of continuous signal, we can make a plot of what looks like a continuous parabola by making the sampling interval very small. We will compare two different time steps, the first of 1, and the second of .001.

In [ ]:
```
# Create the list of x-values.
# x is a vector/array with step size of .001

x    = np.arange(-50,50,.001)
```

In [ ]:
```
# Print out the x-values. Here are the x-values for -50 to 50 in steps of .001
x
```

In [ ]:
```
# By the command 'shape' one can determine how many elements are in the array.
# If it is multi-dimensional it will also give the number in each dimension.

x.shape
```

In [ ]:
```
# Let's plot a parabola. We can define the function in the form y(x) = a*x^2 + b

def y(x, a=3, b=2, c=-2):
    return a*x**2 + b*x + c

# Please note, I am using a general parabola function of the form y(x) =a*x^2 +
```

The actual plotting is done below. Note how the axes and tick marks are clearly labeled. You always want your plots to be able to stand on their own. This is not only important for communicating your results to others, but also for yourself. It is easy to build up a collection of many plots in a project and not remember what was actually being plotted because the labels were not done well.

In [ ]:
```
# Now we want to plot the figure.

fig = plt.figure()
ax  = fig.add_subplot(111)

#These lines will add in the legend
lbl = 'Parabola $y(x)$'
ax.plot(x,y(x),label=lbl)

ax.set_xlim(-3,3)
ax.set_ylim(-5,10)
ax.set_title('$y(x) = ax^2 + bx + c$')
ax.set_xlabel('$x$ ')
ax.set_ylabel('$y(x)$')

ax.legend(loc='upper right',fontsize='x-small')

plt.show()
```

Once you finish, complete these tasks:

1. Change axis limits
2. Move the legend to the bottom right.
3. Plot the same parabola but with step sizes of .1 (in x-value)
4. Overlay the smallSteps parabola with the parabola with step sizes of .1
5. Plot a different function. y(x) = x^3+2 or another one of your favorite functions.
6. Answer the following in a markdown cell using complete sentences: Why does the parabola look so bad when the x step size is 1?

1. Change the x limits to -10 to 10 and the y limits to -10 to 10

In [ ]:

1. Move the legend to the lower right

In [ ]:

1. Plot the same parabola, but the x-values are in steps of 1.

In [ ]:

1. Overlay the 1.0 step size parabola on top of the 0.01 step size parabola. Use the legend to indicate which is which.

In [ ]:

1. Plot a different function. y(x) = x^3+2 or another one of your favorite functions. Make sure

you change the plot title to match your function!

In [ ]:

1. Answer the following in a markdown cell using complete sentences: Why does the parabola look so bad when the x step size is 1?

In [ ]:

Additional Resources

- Matplotlib Gallery
- A Quick Guide to LaTeX
- Remember that when you don't know how to do something, Google and StackOverflow are your friends!

Helpful hints for submitting work (when required)

1. When you think you are finished, **do a clean run of the notebook** by going to "Kernel" → "Restart Kernel and Run All Cells".
2. **Double-check the output for any errors!** Doing the clean run may reveal things that you thought were working are actually broken.
3. **Save the notebook.**
4. **Upload the final `*.ipynb` file to the Brightspace dropbox.** This file is stored locally on your computer. In Jupyter Lab, you can find the file path by right-clicking on the notebook tab and selecting "Show File in Browser". Alternatively, you can download a copy of the notebook in your browser by going to "File" → "Download".

In [ ]: