Homework (due 04/26/2022 11.30am)

Use your code (a solution) for a problem below and make the following plots using matplotlib:

a) For each θ: 0rad, pi/6 rad, pi/2 rad plot f' versus v/c with uncertainties on f' (3 plots in total)

b) For ach v: 0.0001c, 0.001c, 0.01c, 0.1c, 0.3c, 0.6c, 0.9c plot f' versus θ with uncertainties on f' (7 plots in total)

Use: https://colab.research.google.com/#scrollTo=UdRyKR44dcNI

or https://www.tutorialspoint.com/execute_python3_online.php or your computer with anaconda3

Email your code to Joanna.Kiryluk@stonybrook.edu

In python write a function f_prime which returns values of f' and uncertainty on f'. In the main code call f_prime for given values of f,v,θ and print (rounded) results: f' +/- sigma_f'

The relativistic Doppler effect causes a shift in the frequency *f* of light originating from a source that is moving in relation to the observer, so that the wave is observed to have frequency *f'*:

$$f' = f \frac{1 - \frac{v}{c} cos\theta}{\sqrt{1 - \frac{v^2}{c^2}}}$$

where ν is the velocity of the source in the observer's rest frame, θ is the angle between the velocity vector and the observer-source direction measured in the reference frame of the source, and *c* is the speed of light. [c = 3*10^8 m/s]

Assuming ν is measured with uncertainty σ_ν = 0.1 *ν and θ is measured with with uncertainty σ_θ = 0.05 * θ to evaluate uncertainty on f' using uncertainty propagation.

f = 100 Hz

v= 0.0001c, 0.001c, 0.01c, 0.1c, 0.3c, 0.6c, 0.9c

θ = 0rad, pi/6 rad, pi/2 rad

BACKUP / HELPFUL

https://matplotlib.org/stable/tutorials/index.html

https://matplotlib.org/stable/tutorials/introductory/pyplot.html#sphx-glr-tutorials-introductory-pyplot-py

Pyplot tutorial

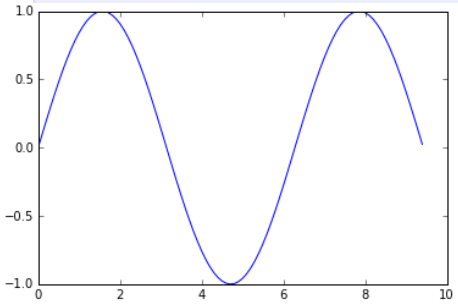
`matplotlib.pyplot` is a collection of command style functions that make matplotlib work like MATLAB. Each `pyplot` function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc. In `matplotlib.pyplot` various states are preserved across function calls, so that it keeps track of things like the current figure and plotting area, and the plotting functions are directed to the current axes (please note that "axes" here and in most places in the documentation refers to the *axes* part of a figure and not the strict mathematical term for more than one axis).

Matplotlib.pyplot (very brief here)

Matplotlib - a plotting library.
`matplotlib.pyplot` - module, which provides a plotting system
The most important function in matplotlib is plot

Example:

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

# Compute the x and y coordinates for points on a sine curve
x = np.arange(0, 3 * np.pi, 0.1)
y = np.sin(x)

# Plot the points using matplotlib
plt.plot(x, y)
plt.show()  # You must call plt.show() to make graphics appear.
```

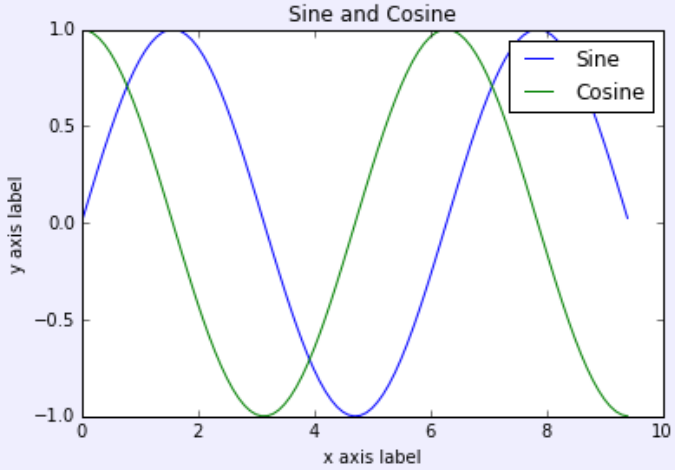


Exercise:

a) re-do the above example for step=1, 0.5, 0.2 and 0.01 values
b) re-do the above example by using linspace function instead of arange

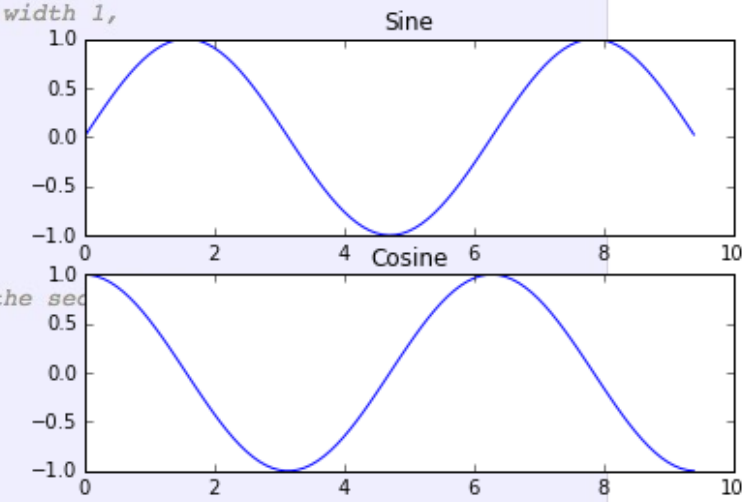
Example:

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

# Compute the x and y coordinates for points on sine and cosine curves
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)

# Plot the points using matplotlib
plt.plot(x, y_sin)
plt.plot(x, y_cos)
plt.xlabel('x axis label')
plt.ylabel('y axis label')
plt.title('Sine and Cosine')
plt.legend(['Sine', 'Cosine'])
plt.show()
```



Example:

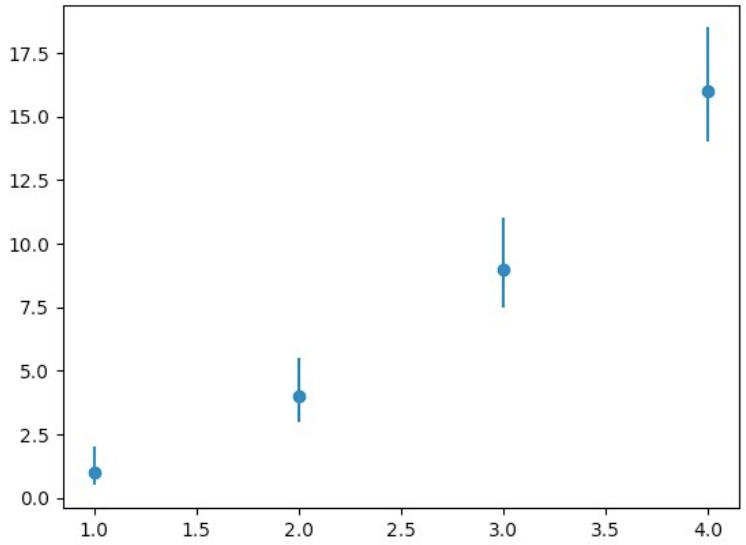
```
import numpy as np
import matplotlib.pyplot as plt

# Compute the x and y coordinates for points on sine and cosine curves
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)

# Set up a subplot grid that has height 2 and width 1,
# and set the first such subplot as active.
plt.subplot(2, 1, 1)

# Make the first plot
plt.plot(x, y_sin)
plt.title('Sine')

# Set the second subplot as active, and make the sec
plt.subplot(2, 1, 2)
plt.plot(x, y_cos)
plt.title('Cosine')

# Show the figure.
plt.show()
```




```
fig_file='fig1.png'
plt.savefig(fig_file)
#plt.clf()
plt.close()
```

Plotting data points with errors

```
import matplotlib.pyplot as plt
x= [1,2,3,4]
y=[1,4,9,16]
ey=[0.5,1.,1.5,2.]
le=[0.5,1.,1.5,2.]
ue=[1.0,1.5,2.0,2.5]

#plt.errorbar(x,y,ey,fmt='o')
plt.errorbar(x,y,yerr=[le,ue],fmt='o')

plt.show()
```



https://pythonhealthcare.org/2018/04/13/51-matplotlib-adding-error-bars-to-charts/

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

%matplotlib inline

# example data
x = np.arange(0.1, 10, 1)
y = x ** 2

# calculate example errors (could also be from list or NumPy a
rray)
lower_y_error = y * 0.2
upper_y_error = y * 0.3
y_error = [lower_y_error, upper_y_error]
lower_x_error = x * 0.05
upper_x_error = x * 0.05
x_error = [lower_x_error, upper_x_error]

# To use only x or y errors simple omit the relevant argument

plt.errorbar(x, y, yerr = y_error, xerr = x_error, fmt='-o')

plt.show()
```