Name: Evan Leglar	**Lab Time** Tuesday 12:00pm

Names of people you worked with:
•

Websites you used:
• Matplotlib • StackExchange

Approximately how many hours did it take you to complete this assignment (to nearest whole number)?	8

The Rules: Everything you do for this lab should be your own work. Don't look up the answers on the web, or copy them from any other source. You can look up general information about Python on the web, but no copying code you find there. Read the code, close the browser, then write your own code.

By writing or typing your name below you affirm that all of the work contained herein is your own, and was not copied or copied and altered.

ELeglar

Note: Failure to sign this page will result in a 50 percent penalty. Failure to list people you worked with may result in no grade for this lab. Failure to fill out hours approximation will result in a 10-percent penalty.

Turn in Python code to Gradescope or your assignment will not be graded

BEFORE YOU BEGIN

1. Make a new project for this lab
   a. Create a lab4.py file for your code
   b. Download msd.py from Canvas and put in your project
2. Get matplotlib if you don't have it already
   a. If you're using a Mac, you might have to put these lines at the top of your `lab4.py` file to make things work properly, depending on how you installed Python.

      ```
      import matplotlib as mpl
      mpl.use('TkAgg')
      ```

   b. You might have to install some extra Python packages for this lab (matplotlib and its dependencies). If you don't know how to do this, ask the TA in the lab session for help.
3. Work through the PyPlot tutorial
4. Google "python time"

Learning Objectives:

You should be able to do the following:

- Plot and graph data using matplot lib
- Analyze your code for time complexity – O(n), etc

Thoughts (come back and read these after you've read the problems):

1. The function to return the sum of 10 samples should use the function `random` from the `random` package. `from random import random` will do what you want. Call the function 10 times, add up the values that it returns, and return this sum from the function. There are a bunch of ways to do this with Python; try to find one that's compact and elegant.

2. To plot the displacement of the mass, you're going to have to extract the displacements from the list of states returned by the simulation. A list comprehension might be helpful here.

3. For the last question, try to write as little code as possible. Use loops and functions as much as you can. Hint: there's a function in the math library called `pow` that calculates exponents. `from math import pow` will let you get to it. It returns a floating point number. If you want an integer, you can convince Python to convert with the `int` function.

4. Your times can be approximate, in the sense that you can have other stuff running on the computer and the garbage collector working. The goal of this part of the lab is to show you how the times scale as the number of items in your list grows, not to get exact timing information. Having said that, only time the actual sorting or summing of the code. Don't include the time it takes to generate the code.

Grading Checkpoints

All your code that generates graphs should be in `lab4.py`. Put the actual code for the other parts in their own file(s), and `import` the important stuff into `lab4.py`. When you run `lab4.py`, it should generate all the plots.

1. Code that generates a sine curve graph. 1 point for a plot. 1 point for the right x axis values. 1 point for labels on both axes and a title. 1 point for having the graph frame fit snugly around the sine curve. [4 points total]

2. Code that generates a histogram (part 3 of this lab). 2 points for a valid histogram. 1 point for one what looks like a normal distribution. 1 point for axis labels and a title. [4 points total]

3. Code that generates a plot of a simulated spring-mass-damper system. 1 point for a successful plot that looks correct, 1 point for axis labels and a title. [2 points total]

4. A plot of times for `sort` and `sum`. 1 point for getting the right plots. 1 point for writing down what the difference between the plots is, in the title. 1 point for answering the run-time question [3 points total]

Remember to hand in the the python files and this file (as a PDF) to Gradescope.

Problem 1 Plot a sine wave

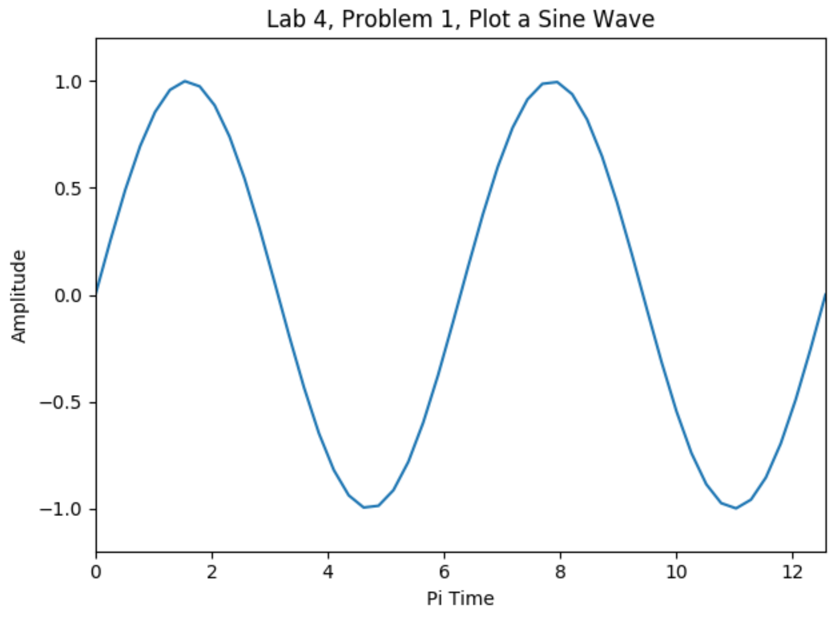
Use `pyplot` to plot a sine wave from 0 to 4 pi. Make sure you get the x axis values right, and that you have a title and axis labels.

Comments for grader/additional information (if any)

Plot code

```
#!/usr/bin/env python3
from math import pi as pi
import numpy as np
import matplotlib.pyplot as plt

"""Sine wave plot"""
x = np.linspace(0, 4*pi, 50)
y = np.sin(x)

plt.plot(x, y)
plt.xlabel("Pi Time")
plt.ylabel("Amplitude")
plt.title("Lab 4, Problem 1, Plot a Sine Wave")
plt.axis([0, 4*pi, -1.2, 1.2])
```

Plot output



Problem 2 histogram

Write a function that returns the sum of ten samples from a uniform distribution from 0 to 1. Run this function some number of times (say 10,000), and put these values in a list. Finally, plot the values in this list as a histogram. The distribution of points should look (approximately) like a normal distribution. Add a title and label your axes.

Comments for grader/additional information (if any)

Plot code requires importing numpy, the samples code, and the matplotlib. Plot code selection only shows code required for the histogram from the lab4.py file.

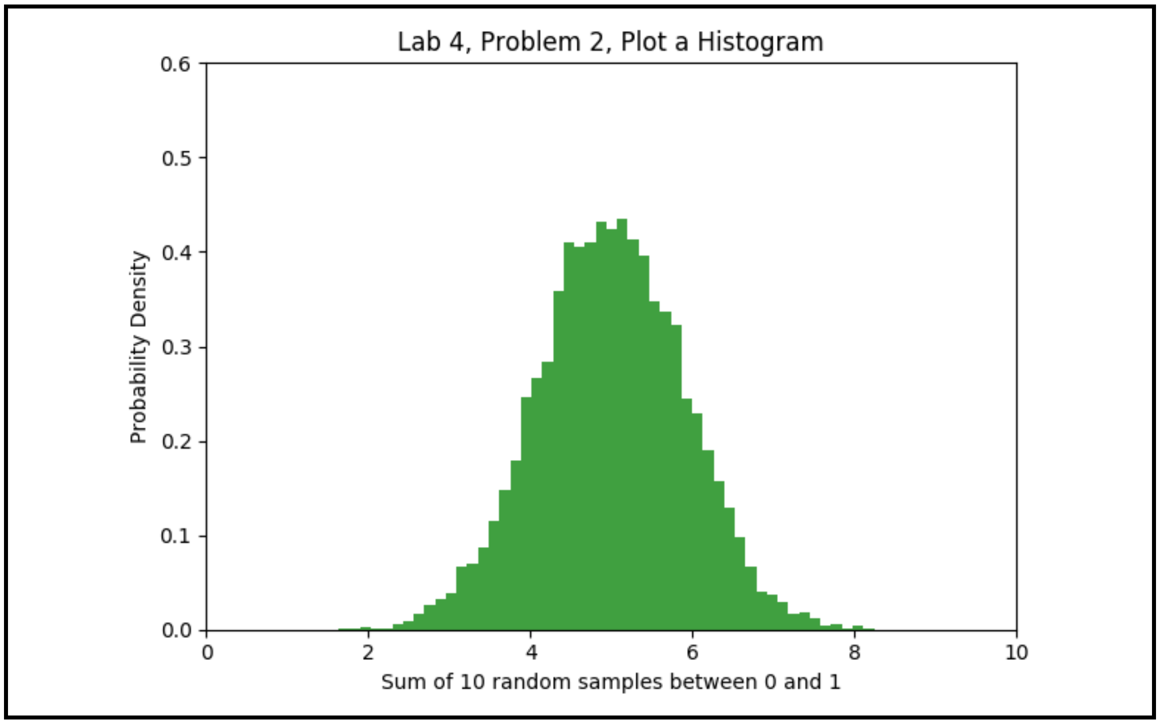
Samples code

```
#!/bin/usr/env python3
import numpy as np
from random import random


def uniform_dist(n):  # average, standard dist, number of runs
    dist_sum = sum(np.random.uniform(0, 1, n))
    return dist_sum


print(uniform_dist(10))
```

Plot code

```
dist_list = []

for i in range(10000):
    dist_list.append(uniform_dist(10))
plt.hist(dist_list, 50, density=1, facecolor='g', alpha=0.75)
plt.xlabel("Sum of 10 Random Samples Between 0 and 1")
plt.ylabel("Probability Density")
plt.title("Lab 4, Problem 2, Plot a Histogram")
plt.axis([0, 10, 0, 0.6])
plt.show()
```

Plot output



Problem 3 Plot spring damper system

1. Look at the code in msd.py. This simulates a mass-spring-damper system, using the ODE integrator that is part of the `scipy` scientific python package. Don't worry if you can't completely understand what's going on right now, since the code involves a class and some fancy lamdba function stuff; we'll talk about it in class. For now, all you need to know is that you initialize the simulation with a line like:

```
smd = MassSpringDamper(m=1.0, k=25, c=5)
```

where the three parameters are the mass, the spring constant, and the damping factor. Once you've done this, run the simulation with the `simulate` function, as shown in msd.py. This will return the states that the system went through, and the corresponding timesteps (in `state` and `t` variables, respectively). Take a look at these: state is a list, where each element is itself a list of length 2, representing the position and velocity of the mass at a given time. It doesn't matter what mass, spring constant, or damping factor you use, just as long as the plot makes sense.

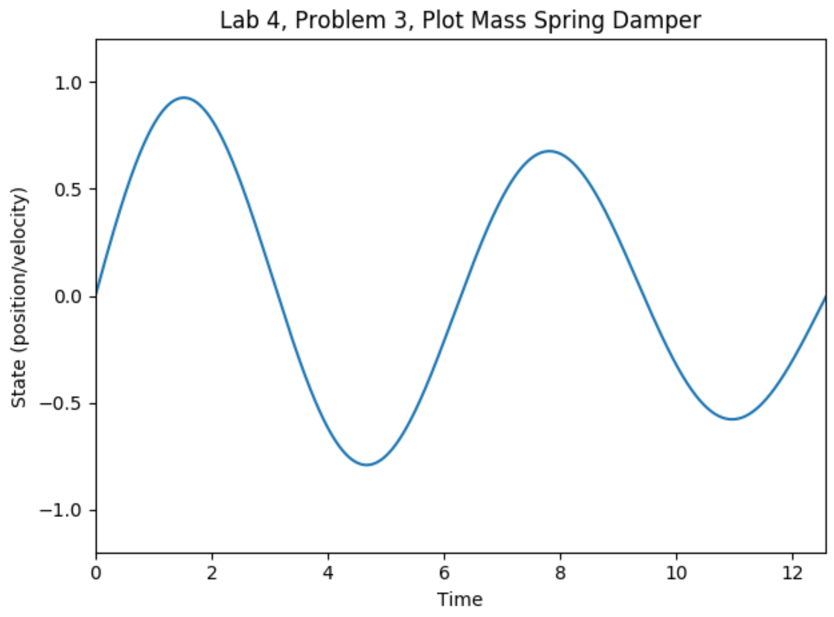
Plot the displacement of the mass over time, using `pyplot`.

Comments for grader/additional information (if any)

Code

```
"""Problem 3: Plotting Mass Spring Damper System"""
smd = MassSpringDamper(m=10.0, k=10.0, c=1.0)
state, t = smd.simulate(0.0, 1.0)

plt.plot(t, state, label='state')
plt.xlabel("Time")
plt.ylabel("State (position/velocity)")
plt.title("Lab 4, Problem 3, Plot Mass Spring Damper")
plt.axis([0, 4*pi, -1.2, 1.2])
plt.show()
```

Plot



Problem 4 Sort times

1. Generate a plot of how long it takes to sort lists of varying lengths with the built-in list `sorted` function. The numbers in the lists should be randomly generated. Show results for lists of length 1, 10, 100, 1,000, 10,000, 100,000, and 1,000,000.

   Also plot the time taken to sum the elements of these arrays, using the `sum` function, on the same graph. For both of these, make sure you count only the time taken to do the sorting, not the time taken to generate the lists.

   Comment briefly on the difference between these lines, in the title of the graph.

   HINT: You may need to change the scale of an axis to properly view the data.

Comments for grader/additional information (if any)

Code

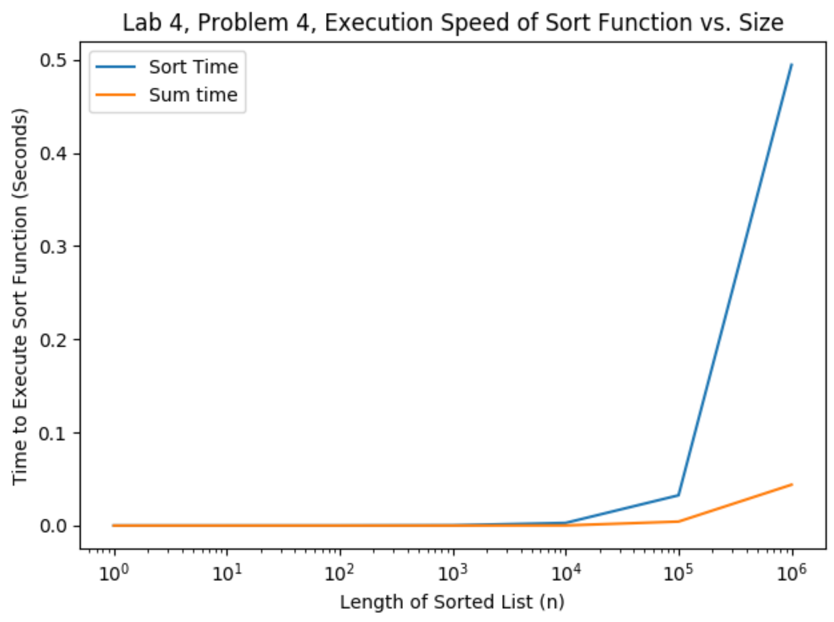
```
sortlength = [10**x for x in range(7)]
timelist_sort = []
timelist_sum = []

for i in sortlength:
    randomlist1 = random.sample(range(0, i*2), i)
    start = timer()
    sortedlist1 = sorted(randomlist1)
    end = timer()
    time2exec = end-start
    timelist_sort.append(time2exec)

for i in sortlength:
    randomlist2 = random.sample(range(0, i * 2), i)
    start = timer()
    sortedlist = sum(randomlist2)
    end = timer()
    time2exec = end - start
    timelist_sum.append(time2exec)

plt.plot(sortlength, timelist_sort, label="Sort Time")
plt.plot(sortlength, timelist_sum, label="Sum time")
plt.ticklabel_format(style='sci', axis='x', scilimits=(0,0))
plt.xlabel("Length of Sorted List (n)")
plt.ylabel("Time to Execute Sort Function (Seconds)")
plt.title("Lab 4, Problem 4, Execution Speed of Sort Function vs.
Size")
plt.legend()
plt.show()
```

Running time

What is the running time of sorted in terms of the number of elements? What about sum? Give your answer as O(n), O(n^2), etc. I.e., what is the shape of the timing curve(s)?

The running time of the sum() function appears to be linear and in some form of the O(n) time complexity until n becomes larger. The running time of the sorted() function also appears to be linear at larger scale but almost at a time complexity of k*O(n) at the higher end of n.

Plot


Lab 4, Problem 4, Execution Speed of Sort Function vs. Size
Sort Time
Sum time
Time to Execute Sort Function (Seconds)
0.5
0.4
0.3
0.2
0.1
0.0
10^0
10^1
10^2
10^3
10^4
10^5
10^6
Length of Sorted List (n)