Purpose

The goal of this lab is to introduce you to some of the plotting and graphing functions available in Python, and to give you some practice using them. We also want to get you started thinking about how long it takes to do things in Python, and how this time depends on how you implement your functions.

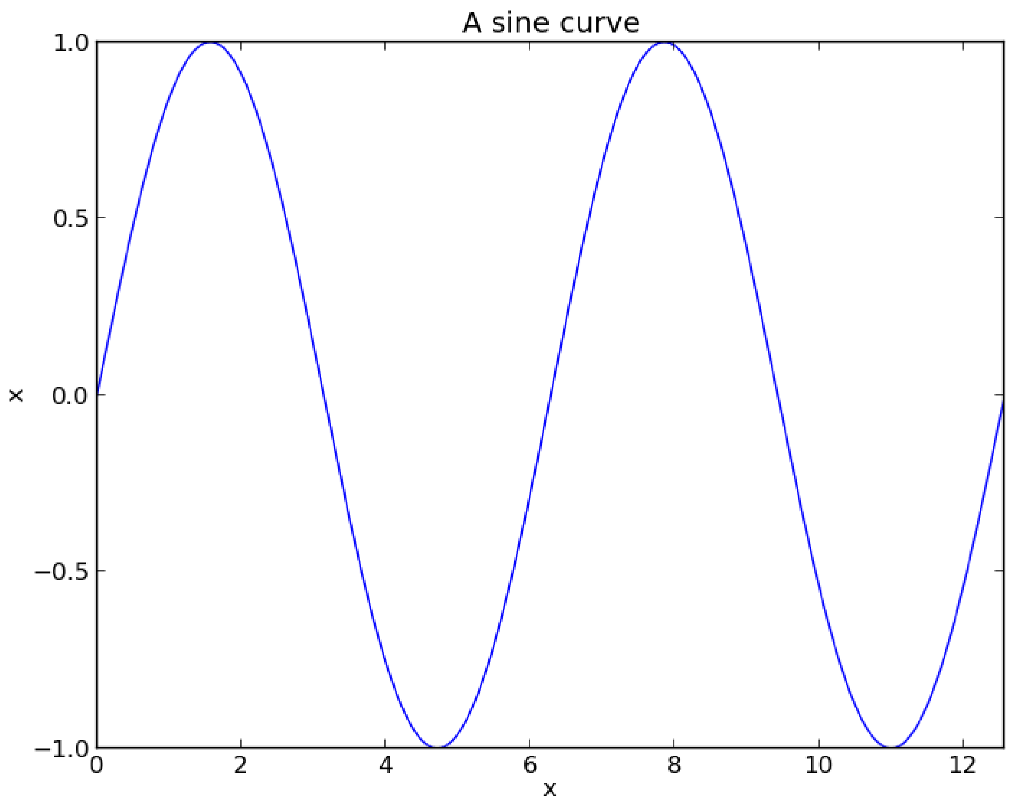
Preparation

Make sure you're familiar with the material from the previous labs.

Assignment

1. There are a number of plotting and graphing libraries that are available in Python. We're going to focus on `matplotlib` (Links to an external site.), one of the most popular ones. Start off by working through the `pyplot` tutorial (Links to an external site.). We're only going to scratch the surface of `pyplot` in this lab, but it's probably got most of the plotting and graphing functionality that you're likely to want (for most applications, at least).

2. Just to make sure that you know how to use `pyplot`, 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 a axis labels. Your output should look like this (complete with labels):



3. 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 10000), and store what it returns in a list. Finally, plot the values in this list as a histogram. The distribution should be normal, but the y values will not be from 0-1. Add a title and axes labels.

4. **Graduate Students:** Use your code from the previous part of the assignment to generate histograms for different sample sizes. Generate histograms for 10, 100, 1000, 10000, 100000, and 1000000 samples, and show them on the same figure (use `subplot`)

5. Look at the code in msd.py. This simulates a spring-mass-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 = SpringMassDamper(1.0, 2.0, 3.0)
   ```

   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 the example file. 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 (because I'm sure you remember how a spring-mass system works, right?).

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

6. Generate a plot of how long it takes to sort a 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. For both of these, make sure you count only the time taken to do the sorting, not the time taken to generate the lists.

Grading

To get points for this lab, you should show your code to the class TA. If you need some extra time to complete the lab, then you can show your solutions to the TA at the start of next week's first lab with no penalty. We assume that most if not all of the grading will take place at the beginning of next week's lab.

There are 14 points available for 499 students, and 17 points available for graduate students in this lab assignment.

1. Show the TA code that generates a sine curve graph like the one above. 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. Show a TA the 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. **Graduate Students:** Show the TA the plot with six histograms on it. 1 point for six histograms. 2 points for the right shape for each of them. [3 points total.]

4. Show the TA a plot of a simulated spring-mass-damper system. 1 point for a successful plot. 1 point for it looking right. 1 point for axis labels and a title. [3 points total]

5. Show the TA your system generating the plot of times for `sort` and `sum`. 1 point for getting the right plot for each. 1 point for telling the TA what the difference between the plots is. [3 points total]

Thoughts

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. When showing the multiple histograms, vary the number of samples from the function you wrote, not the number of samples used to calculate the sum. Leave the original function the same, and only vary the number of times you call it and store the result.

3. To plot the displacement 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.

4. 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.

5. 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.

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.