Lab 0 - Computational Python - Problems

January 10, 2018

1 Lab 0 - Python Intro and IPython Tutorial

```
In [ ]: %matplotlib inline
        import numpy as np
        import matplotlib.pyplot as plt
```

1.1 1 Warm-up Exercises

Try the following commands on your IPython terminal and see what output they produce.

```
In [ ]: a = 1 + 5
        b = 2
        c = a + b
        print(a / b)
        print(a // b)
        print(a - b)
        print(a * b)
        print(a**b)

In [ ]: a = np.array([[3, 1],
                      [1, 3]])
        b = np.array([[3],
                      [5]])
        print(a * b)
        print(np.dot(a, b))
        print(np.dot(b.T, a))
        c = a**(-1)
        print(c * a)

In [ ]: t = np.arange(10)
        g = np.sin(t)
        h = np.cos(t)
        plt.figure()
        plt.plot(t, g, 'k', t, h, 'r');

        t = np.arange(0, 9.1, 0.1)
        g = np.sin(t)
        h = np.cos(t)
        plt.figure()
        plt.plot(t, g, 'ok', t, h, '+r');
```

```
In [ ]: t = np.linspace(0, 10, 20)
        print(t)
        t = np.logspace(0.001, 10, 9)
        print(t)
        t = np.logspace(-3, 1, 9)
        print(t)
        y = np.exp(-t)

        plt.figure()
        plt.plot(t, y, 'ok')
        plt.figure()
        plt.semilogy(t, y, 'ok')
```

1.2 2 Integration Function

Here is a more complicated function that computes the integral $y(x)$ with interval dx:

$$c = \int y(x)dx \sim \sum_{i=1}^{N} y_i dx_i.$$

It can deal with both cases of even and uneven sampling.

```
In [ ]: def integral(y, dx):
            # function c = integral(y, dx)
            # To numerically calculate integral of vector y with interval dx:
            # c = integral[ y(x) dx]
            # ------ This is a demonstration program ------
            n = len(y) # Get the length of vector y
            nx = len(dx) if np.iterable(dx) else 1
            c = 0 # initialize c because we are going to use it
            # dx is a scalar <=> x is equally spaced
            if nx == 1: # '==', equal to, as a condition
                for k in range(1, n):
                    c = c + (y[k] + y[k-1]) * dx / 2
            # x is not equally spaced, then length of dx has to be n-1
            elif nx == n-1:
                for k in range(1, n):
                    c = c + (y[k] + y[k-1]) * dx[k-1] / 2
            # If nx is not 1 or n-1, display an error messege and terminate program
            else:
                print('Lengths of y and dx do not match!')
            return c
```

Save this program as `integral.py`. Now we can call it to compute $\int_0^{\pi} \sin(t)dt$ with an evenly sampled time series (`even.py`).

```
In [ ]: # number of samples
        nt = 100
        # generate time vector
        t = np.linspace(0, np.pi, nt)
        # compute sample interval (evenly sampled, only one number)
        dt = t[1] - t[0]
        y = np.sin(t)
        plt.plot(t, y, 'r+')
        c = integral(y, dt)
        print(c)
```

1.2.1 Part 1

First plot $y(t)$. Is the output c value what you are expecting for $\int_0^{\pi} \sin(t)dt$? How can you improve the accuracy of your computation?

1.2.2 Part 2

For an unevenly spaced time series that depicts $\sin(4\pi t^2)$ (so-called chirp function), compute $\int_0^1 \sin(4\pi t^2)dt$ (saved as `uneven.py`).

```
In [ ]: nt = 20
        # sampling between [0,0.5]
        t1 = np.linspace(0, 0.5, nt)
        # double sampling between [0.5,1]
        t2 = np.linspace(0.5, 1, 2*nt)
        # concatenate time vector
        t = np.concatenate((t1[:-1], t2))
        # compute y values (f=2t)
        y = np.sin(2 * np.pi * 2 * t**2)
        plt.plot(t, y)
        # compute sampling interval vector
        dt = t[1:] - t[:-1]
        c = integral(y, dt)
        print(c)
```

Show your plot of $y(t)$ (for $nt = 50$). Try different nt values and see how the integral results change. Write a `for` loop around the statements above to try a series of `nt` values (e.g, 20, 50, 100, 500, 1000) and generate a plot of $c(nt)$. What value does c converge to after using larger and larger nt? (Please attach your modified Python code.)

1.3 3 Accuracy of Sampling

Let us sample the function $g(t) = \cos(2\pi f t)$ at sampling interval $dt = 1$, for frequency values of $f = 0, 0.25, 0.5, 0.75, 1.0$ hertz.

In each case, plot on the screen the points of the resulting time series (as isolated red crosses) to see how well it approximates $g(t)$ (plotted as a blue-dotted line, try a very small dt fine sampling). (Submit only plots for frequencies of 0.25 and 0.75 Hertz, use xlabel, ylabel, title commands to annotate each plot). For each frequency that you investigated, do you think the sampling time series is a fair representation of the original time series $g(t)$? What is the apparent frequency for the sampling time series? (Figure out after how many points (N) the series repeats itself, then the apparent frequency = 1/(N*dt). You can do this either mathematically or by inspection. A flat time series has apparent frequency = 0.) Can you guess with a sampling interval of $dt = 1$, what is the maximum frequency f of $g(t)$ such that it can be fairly represented by the discrete time series? (Please attach your Python code.)