COMP 1012 Fall 2016 Assignment 4

Due Date: Friday, November 25, 2016, 11:59 PM

Material Covered

- numpy arrays

Notes:

- When programming, follow the posted programming standards to avoid losing marks. A program called `CheckStandardsV2.py` is provided to enable you to check that you are following standards.
- You will hand in two script files for this assignment. Name your script files as follows: `<LastName><FirstName>A4Q1.py` and `<LastName><FirstName>A4Q2.py`. For example, `LiJaneA3Q1.py` is a valid name for a student named Jane Li. If you wish to add a version number to your file, you may add it to the end of the file name. For example, `SmithRobA4Q1V2.py` is a valid name for Rob Smith's script file.
- Submit output in similarly-named output files: e.g., `<LastName><FirstName>A4Q1Output.txt`. To generate this file, open a new empty tab in Spyder and save it under the name above. **Make sure you choose *Text files* from the *Save as type:* list before you save the file!** Then copy and paste the output of your program from the console window to this file, and save it.
- You must complete the checklist for a ***Blanket Honesty Declaration*** in UM Learn to have your assignment counted. This one honesty declaration applies to all assignments in COMP 1012.
- To submit the assignment follow the instructions on the course website carefully (link: http://www.cs.umanitoba.ca/~comp1012/Documents/Handin_UMLearn.pdf). You will upload both script file and output via a dropbox on the course website. There will be a period of several days before the due date when you can submit your assignment. ***Do not be late!*** If you try to submit your assignment within 48 hours ***after*** the deadline, it will be accepted, but you will receive a penalty (of 25% per day late).

Group Work

NOTE: This assignment allows you to work with others as a group.

If you do the assignment all on your own and hand it in, you can get full marks if you achieve about 70% on the assignment. There will be no bonuses for doing better than that.

If you do the work with a friend, then each of you must hand in your own copy of the assignment. They do not have to be the same, but they can be, except for the inner author identification. If the solution you hand in is completely correct, you can get full marks, and similarly for your friend.

If you do the work with two others, then each of you must hand in your own copy of the assignment. Even if the solution you hand in is completely correct, you can get at maximum about 3.5 out of 4 marks. If you are part of a group of four, then each can get a maximum of 3 out of 4 marks. If you are part of a group of six, then each can get a maximum of about 2.5 out of 4 marks. If you are part of a group of nine, then each can get a maximum of 2 out of 4 marks. The more people you work with, the smaller is your possible mark. In general, the maximum mark is min $(4, 6/\sqrt{n})$ when n people work as a group.

If you work as part of a group containing James Dean, Gwen Worobec and Calvin Wong as well as yourself, you must insert lines like the following into your code's initial doc string to avoid a charge

of academic dishonesty. The names deanj12, worobg and wongc23 are the UMnetIDs of your associates.

```
@with deanj12
@with worobg
@with wongc23
```

They in turn should name you and one another in their submitted assignments.

If you do the assignment by yourself, put this line into your initial doc string:

```
@with nobody
```

Question 1—Least Squares Regression [10 marks]

Description

A common problem in statistics and machine learning is to estimate the relationship between data given only a sample of that data. In this example, we will have a collection of (x,y) pairs representing the relationship between an independent variable (x) and a dependent variable (y). You will be given a collection of such pairs (x_1,y_1), …, (x_n,y_n).

As an example, consider you have n gold nuggets. For each i, x_i is the weight of the nugget and y_i is the volume. You should expect that the x_i and y_i are linearly related (based on the density of gold), but that there is also some variation due to impurity and measurement error. What is the best estimate of the density from these n samples?

One way to determine this is to calculate the least squares regression for the data. Given the data x_i and y_i, we want to calculate the line y = ax + b that minimizes the square of the errors for each data point. That is, we want to minimize $(y_i - (ax_i + b))^2$, since $ax_i + b$ is the estimated value of y_i according to our line y = ax + b. Therefore, we want to minimize $\sum_{i=1}^{n}(y_i - (ax_i + b))^2$.

By taking derivatives of the sum $\sum_{i=1}^{n}(y_i - (ax_i + b))^2$ and solving, we can obtain formulas for a and b:

(NOTE, Nov 17: a and b were originally backwards in the formulas below. They are now reversed.)

$$a = \frac{\sum_{i=1}^{n}(y_i - \mu_y)(x_i - \mu_x)}{\sum_{i=1}^{n}(x_i - \mu_x)^2}, b = \mu_y - a\mu_x$$

In these formulas, μ_x is the average of the x values and similarly for μ_x. For this and other formulas in the assignment you may use the numpy.average() function, which takes an array and returns the average of all values.

Write a program that works with two numpy arrays containing the x and y values, and calculates the values of b and a. In particular, you should write the following functions:

- `generateData(n):` Generate random data according to the process described below. The function should return a 2-tuple containing x and y values, stored in numpy arrays. Both numpy arrays should have length n. This function cannot have any loops
- `calculateLeastSquares(x,y):` return a tuple (a,b) that represents the linear least squares regression line. The parameters x and y are numpy arrays. This function cannot contain any loops.

- `pearson(x,y,a,b):` Calculate the Pearson Correlation Coefficient using the process described below. This function cannot have any loops.
- `showMenu():` Display a menu to the user allowing them the choose how to generate and process the data. See below for more details.
- `showTermination():` show the three termination lines (author, date, "end of processing"). ***You will not get marks*** for this function, but it must be present.

For each function, use assertions to verify that numpy arrays have the correct type and that arrays have the same length when necessary.

Generating Data

For the `generateData` function, you will use two functions:

- `numpy.random.random(n)`: this generates an array of length n filled with floats that take on values between 0 and 1, uniformly. You may scale these results by multiplying the vector by a constant. If you omit the parameter n, the function returns a single value between 0 and 1.
- `numpy.random.randn(n):` generates an array of length n filled with floats that follow a normal distribution with mean 0 and variance 1.

In your `generateData` function you should first generate two random floats of A and B (these are the values that `calculateLeastSquares` will eventually estimate). The values of A and B should each be between -100 and +100. Then generate a vector of length n of values between 0 and 250 for x. Finally, generate the vector y from x using the values of A and B, as well as some random noise. In particular, the "ideal" value of y_i should be $Ax_i + B$, but the ideal value should have some noise added. The added noise should be values from a normal distribution with mean 0 and variance 100.

Once you have generated both the x and y arrays, return them as a tuple from the function `generateData`.

Calculating Perason's Correlation Coefficient

How do we know how good our estimated function is? We can calculate Pearson's Correlation Coefficient to determine how strong the relationship is between the actual y values and our estimates based on the values a and b.

To calculate Pearson's Correlation Coefficient, we need to first give calculate the estimate for each data point x_i. Recall that our estimate is given by the line ax+b, so our estimate for x_i is $ax_i + b$. Call this quantity e_i (for the i-th estimate).

With this computed, we use the following formula:

$$r = \frac{\sum_{i=1}^{n}(e_i - \mu_e)(y_i - \mu_y)}{\sqrt{\sum_{i=1}^{n}(e_i - \mu_e)^2}\sqrt{\sum_{i=1}^{n}(y_i - \mu_y)^2}}$$

Here, as above, μ_e is the average of the e_i values. The value of r can range between -1 and +1. Values close to zero reflect no relationship between the y values and the estimates. Values close to +1 indicate a strong positive linear correlation and values close to -1 indicate a strong negative linear correlation. For more examples of the relationships indicated by these values, see https://en.wikipedia.org/wiki/Pearson_product-moment_correlation_coefficient.

Once you have calculated this value, return the correlation coefficient.

Menu

Present the user with the following menu:

```
Select one of the following options:
1. Generate a new random dataset.
2. Calculate least squares for the dataset.
3. Calculate the Pearson Correlation Coefficient for the data set and the estimate.
4. Quit.
```

Note that there is an order to these operations: if the user selects 2 before 1 or 3 before 2, the program should report an error. Every time option 1 is selected, the user should be prompted for a new length and a new data set should be computed and stored. If a user input an option that is not 1-4, or a value that is not an integer, an error should be reported. The menu should be repeatedly displayed until the user selects 4.

Sample Output

```
LEAST SQUARES REGRESSION
Select one of the following options:
1. Generate a new random dataset.
2. Calculate least squares for the dataset.
3. Calculate the Pearson Correlation Coefficient for the data set and the estimate.
4. Quit.
Enter your selection: 2
Error: no data generated yet

Select one of the following options:
1. Generate a new random dataset.
2. Calculate least squares for the dataset.
3. Calculate the Pearson Correlation Coefficient for the data set and the estimate.
4. Quit.
Enter your selection: 1

Enter length of data: 200
Data Generated.

Select one of the following options:
1. Generate a new random dataset.
2. Calculate least squares for the dataset.
3. Calculate the Pearson Correlation Coefficient for the data set and the estimate.
4. Quit.
Enter your selection: 3
Error: data generated but least squares not completed

Select one of the following options:
1. Generate a new random dataset.
2. Calculate least squares for the dataset.
3. Calculate the Pearson Correlation Coefficient for the data set and the estimate.
```

```
4. Quit.
Enter your selection: 2
Least squares line: y = -110.61 x -25.62

Select one of the following options:
1. Generate a new random dataset.
2. Calculate least squares for the dataset.
3. Calculate the Pearson Correlation Coefficient for the data set and the estimate.
4. Quit.
Enter your selection: 3
Pearson correlation coefficient: 1.00

Select one of the following options:
1. Generate a new random dataset.
2. Calculate least squares for the dataset.
3. Calculate the Pearson Correlation Coefficient for the data set and the estimate.
4. Quit.
Enter your selection: 4
Programmed by the Instructors
Date: Fri Nov  4 10:06:33 2016
End of Processing
```

Hand-in

Submit your script file and your output showing the results for all two different randomly generated data sets: one of size 10 and one of size 300. Make at least one error in your user input.

Question 2—Curve Length [10 marks]

Description

In this question you will take a function of *x* and from it determine the length of the curve along the function. This question does not have any user input.

To calculate the length of a curve given by a function, the function will be approximated by short line segments, and the length of these short line segments will be summed to give an approximate value for the length of the function. The shorter the line segments, the better the approximation, so your solution will compute the length for several different numbers of line segments, and show all of the computed lengths.

Functions

The main difference between this code and other code that you have written is that we will pass **functions** as parameters. In particular, the function `showLength` calls a function `targetFunc` that is passed to it as a parameter. In your `main` function, the argument you pass as `targetFnc` will be one of `halfCircle`, `fractionalPowers` or `xSinx`, the three target functions. You will write each of these target functions to compute a different function. All functions must have a clear docstring explaining their usage.

The functions you will write are:

- Script: calls `main()`.
- `main()` does the following:

- prints the program heading;
- calls `showLength` three times, each time passing one of the functions `halfCircle`, `fractionalPowers` or `xSinx` as an argument for the parameter `targetFunc` (note: if you pass 2 and 50 to `showLength` you can produce the same output as the sample);
- calls `showTermination`.

❖ `showLength(targetFunc, minPoints, maxPoints):` This function does the required calculations to print out the properties of objects created using `targetFunc`. There is **one loop** in `showLength`, a loop that changes the number of points along the curve. It starts with `minPoints`, and each time through the loop it doubles the number of points until it is greater than or equal to `maxPoints`. Each time through the loop it prints a row of properties for the given target function and the number of points on it. It gets the *x* and *y* values of the points on the curve by calling `targetFunc` with the number of points. That function returns a tuple of two values: ***numpy arrays*** for `xs` and for `ys`. `showLength` then passes the arrays down to `findLength` to actually compute the length, which are returned as a single floating point number. Note that in addition to the table rows itself, `showLength` prints a heading and also the column headers.

❖ `findLength(xs, ys):` This function takes a sequence of *x* and *y* values along a curve and determines the perimeter of the curve. There should be **NO** loops in this function. If the points are $\{(x_j, y_j)\}_{j=0,n}$ then the length of the curve is approximated by P:

$$P = \sum_{j=1}^{n} \sqrt{(x_j - x_{j-1})^2 + (y_j - y_{j-1})^2}$$

❖ `showTermination():` This function prints the termination output. It is the same as in Assignment 2. You may use your own code from that assignment, or the code from the posted solution (which you must reference), but of course you must name yourself. ***You will not get marks*** for this function, but it must be present.

Your solution also requires **three target functions**. Write each of these three functions to compute the (x,y) coordinates of the corresponding function over a defined interval. The number of points to use in the interval will be specified by the numPoints parameter in each case. Each function must return a tuple of two numpy arrays: the x coordinates (always evenly spaced) and the y coordinates (different for each function). **None of these functions should have any loops.**

❖ `halfCircle(numPoints):` This is a target function represents a half circle going from x = -5 to x = 5. The one parameter, `numPoints`, is an integer greater than 1. It represents the number of points to be placed along the curve. This function returns a tuple of two values: a numpy array of `numPoints` *x* values evenly spaced from –5 to +5, a numpy array of `numPoints` *y* values obtained by evaluating the following expression for each *x* value: $y = \sqrt{25 - x^2}$.

❖ `fractionalPowers(numPoints):` This is a target function that is a sum of fractional powers of x. The one parameter, `numPoints`, is an integer greater than 1. It represents the number of points to be placed along the curve. This function returns a tuple of two values: a numpy array of `numPoints` *x* values evenly spaced from 0 to 100, a numpy array of `numPoints` *y* values obtained by evaluating the following expression for each *x* value: $y = \sqrt[4]{x} + \sqrt[3]{x} + \sqrt{x}$.

❖ `xSinx(numPoints):` The one parameter, `numPoints`, is an integer greater than 1. It represents the number of points to be placed along the curve. This function returns a tuple of two values: a

numpy array of `numPoints` *x* values evenly spaced from -20 to 20, a numpy array of `numPoints` *y* values obtained by evaluating the following expression for each *x* value: $y = x\sin(x)$.

Sample Output:

```
CURVE LENGTHS

Length of the curve of the function halfCircle
#POINTS  LENGTH
     2    10.0000
     4    14.8803
     8    15.4810
    16    15.6361
    32    15.6838

Length of the curve of the function fractionalPowers
#POINTS  LENGTH
     2   101.5725
     4   102.2079
     8   102.7544
    16   103.2254
    32   103.5786

Length of the curve of the function xSinx
#POINTS  LENGTH
     2    40.0000
     4    54.6274
     8    55.0262
    16   216.9450
    32   256.5710

Programmed by Instructors
Date: Fri Nov  4 09:19:28 2016
End of processing
```

Hand-in

Submit your script file and your output showing the results for all three target functions, for a minimum of 3 points and a maximum of 5000 points.