COMP 1012 Fall 2016 Assignment 5

Due Date: Friday, December 9, 2016, 11:59 PM

Material Covered

- numpy 2D arrays
- matplotlib
- file processing

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>A5Q1.py`. For example, `LiJaneA5Q1.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, `SmithRobA5Q1V2.py` is a valid name for Rob Smith's script file.
- Submit output in similarly-named output files: e.g., `<LastName><FirstName>A5Q1Output.png`. To generate a text 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
```

Description

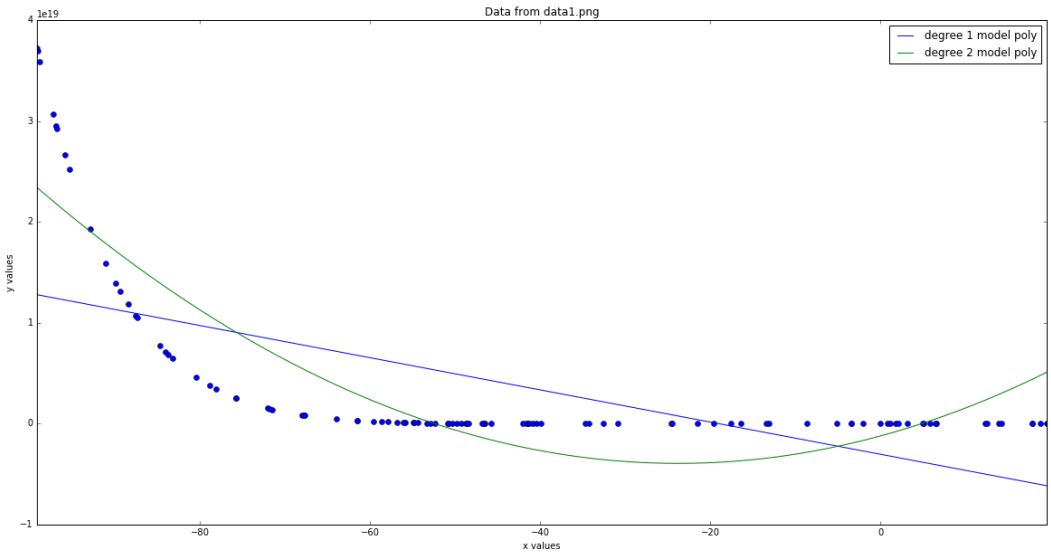
In this question, you will extend the work of Assignment 4, Question 1 to be able to approximate data with a higher order polynomial.

In particular, you will be working with a set of (x, y) pairs, as before, representing a relationship between an independent variable (x) and a dependent variable (y). You will be given a collection of such pairs $(x_1, y_1), \ldots, (x_n, y_n)$. However, instead of trying to calculate the line of best fit $f(x) = ax + b$, we will be trying to calculate a *model polynomial*

$$f(x) = \sum_{i=0}^{m} a_i \, x^i \tag{1}$$

Thus, we are trying to calculate coefficients $a_0, a_1, \ldots, a_m$ for a degree-m model polynomial, instead of just coefficients a and b, as in Assignment 4. For example, if $m = 2$, then the model polynomial we are trying to calculate is: $a_0 + a_1 x + a_2 x^2$.

For example, the figure below shows a data set (blue dots), and two model polynomials: a linear polynomial (blue line) and a quadratic (degree two) polynomial (green curve). As you can see, neither approximates the data set well.



The derivation of a formula for the coefficients of the model polynomial is done in the same way as for the linear case in Assignment 4 (by taking derivatives and solving), but the result is expressed as a formula in terms of matrices and vectors:

$$a = (V^T V)^{-1}(V^T)y \tag{2}$$

Here, there are three quantities and three operators:

- **The output of the formula** is a is the vector $(a_0, a_1, \ldots, a_m)$ of coefficients we are solving for, where m is the degree of the polynomial. This represents the polynomial in equation (1) that we are looking for.
- y is the vector of data $(y_0, y_1, \ldots, y_n)$, where n is the number of data points.
- V is a special 2D matrix of values called the Vandermonde matrix. It is built using the x_i, and is described below.
- V^T is the transpose of V, obtained by flipping V along the major diagonal.
- The power of -1 represents inverting a matrix. The matrix inverse of a matrix A (as in Lab 8) is the matrix B such that AB is the identity matrix.
- Multiplications in the formula are matrix/vector multiplications, obtained using the dot product in numpy.

Numpy Operations

To facilitate computing the vector a, three new numpy operations will be useful:

1. `numpy.linalg.inv(A)`: given a square matrix A, this function returns its inverse.
2. `numpy.transpose(A)`: given a 2D matrix A, this function returns its transpose.
3. `numpy.vander(xs,m,increasing=True)`: this function creates 2D Vandermonde matrices. This is described in detail below.

Vandermonde Matrix

The Vandermonde matrix is used to determine the coefficients a_i of the model polynomial, and is represented by V in formula (2). The Vandermonde matrix is defined using the degree of the model polynomial and the values x_i. (Note that the y_i are incorporated in the formula (2) directly.)

For values $(x_1, x_2, \ldots, x_n)$ and an integer value m, the Vandermonde matrix V is given by

$$V = \begin{pmatrix} 1 & x_1 & {x_1}^2 & {x_1}^3 & \cdots & {x_1}^m \\ 1 & x_2 & {x_2}^2 & {x_2}^3 & \cdots & {x_2}^m \\ 1 & x_3 & {x_3}^2 & {x_3}^3 & \cdots & {x_3}^m \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & {x_n}^2 & {x_n}^3 & \cdots & {x_n}^m \end{pmatrix}$$

Note that if V is this Vandermonde matrix, and $a = (a_0, a_1, \ldots, a_m)$, then Va is a vector where the i-th entry is $f(x_i)$, where f is the model polynomial in formula (1) and m is the degree of f. Thus, the vector a we are calculating is the best solution to $y = Va$.

To calculate the Vandermonde matrix, use the built-in numpy function `numpy.vander()`. The function takes three parameters:

1. The first parameter is a numpy array containing the values x_i.
2. The second parameter how many columns the matrix should have. Note that this value should be **one more than the degree** of the polynomial that you are trying to fit to the data.

3. For the third parameter, you should specify "increasing=True". If you don't you will get the mirror image of the Vandermonde matrix above.

Functions

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

- `loadData(fileName):` Load data from the file given by the string variable `fileName`. The file is assumed to be in the same directory as the script, and the contents of the file satisfy the format described in the "File Format" section below (that is, your program can crash if the file doesn't exist or isn't in the right format). The function should return a 2-tuple containing x and y values, stored in numpy arrays. This function may have a loop to read the file.
- `calculateModelPoly(xs,ys,deg):` return a numpy array that represents the coefficients model polynomial of degree `deg` for the data `xs` and `ys`. The parameters `xs` and `ys` are numpy arrays. This function cannot contain any loops.
- `evaluatePoly(poly,xs):` this function takes two parameters: a polynomial `poly`, given as a numpy array of coefficients, and `xs`, a sequence of x values that the polynomial should be evaluated at, which is a numpy array. The function should return another numpy array of the same length as `xs` that contains, in position i, the value of the polynomial `poly` evaluated at x_i (the i-th entry of `xs`). This function may have **one** loop. You can use Horner's method or standard polynomial evaluation.
- `plotModelPolys(xs,ys,start,end,filename):` this function takes five parameters: two numpy arrays `xs` and `ys` of the same length, two integers, `start` and `end` and a `filename` as a string. This function should compute the model polynomials of all degrees d between `start` and `end` (inclusive) and plot them in a matplotlib plot, as well as a scatter plot of the `xs` and `ys`. The function should save the plots as a png in the filename provided, according to the requirements given below in the section "Model Polynomial Plots". The function should return nothing. This function may have **one** loop over the different degrees from start to end.
- `showMenu():` Display a menu to the user allowing them the choose how to generate and process the data. See below for more details. This function may have loops to validate input.
- `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.

Model Polynomial Plots

The function plotModelPolys should ensure the following requirements are met for the plot:

1. All data points should be displayed.
2. The plotted range of the model polynomials should be from the minimum value of the xs to the maximum value of the xs.
3. The polynomials should be evaluated at 1000 points in the interval.
4. Each polynomial should be coloured with a different colour.
5. Ensure that the output figure has appropriate labels on the x and y axes, as well as a reasonable title.

Menu

Present the user with the following menu:

```
Select one of the following options:
1. Load a data dataset.
2. Display model polynomials for the dataset.
3. Quit.
```

As with Assignment 4, there is an order to the commands: if a user selects 2 without 1, then an error should be shown. Additionally, the menu should behave as follows:

- Every time option 1 is selected, the user should be prompted for a file name.
- Every time option 2 is selected, the user should be prompted for start and end values for the degree (as used in plotModelPolys), as well as an output filename. This should result in a figure being saved to the output file with all model polynomials from the start to end degrees (inclusive).
- If a user input an option that is not 1-3, or a value that is not an integer, an error should be reported. The menu should be repeatedly displayed until the user selects 3.

Sample Output

An example run of the program is given below. It uses

```
Select one of the following options:
1. Load a data dataset.
2. Display model polynomials for the dataset.
3. Quit.
Enter your selection: 1
Enter the filename: data1.csv
Data loaded

Select one of the following options:
1. Load a data dataset.
2. Display model polynomials for the dataset.
3. Quit.
Enter your selection: 2
Enter starting degree: 1
Enter ending degree: 2
Enter the output filename: data1.png

Select one of the following options:
1. Load a data dataset.
```

```
2. Display model polynomials for the dataset.
3. Quit.
Enter your selection: 3
Programmed by the Instructors
Date: Mon Nov 21 16:19:32 2016
End of Processing
```

Input File Format

The input files are guaranteed to have the following format. The first line of the file contains a single number N, which is the number of data points. The next N lines contain two integers per line, separated by a single comma. Each line represents a single data point of the form x,y.

MatplotLib Drawing Hints

Here are two drawing-specific hints for Matplotlib:

- mpl.figure(figsize=(20,10)) will change the size of the figure. You may need to adjust the size of your figure so that the legend does not overlap with the data.
- mpl.ioff() will ensure that the figure is not drawn in the window, when saving a file.

Hand-in

Submit your script file and your output (txt file) showing the results for all input files provided. **Use appropriate values for the degrees based on the data sets.** You should also submit all of the images generated by the script as png files. Make at least one error in your user input.