Problem – 1

Question -1: Complete the functions in polyfit.py, which accepts as input a dataset to be fit and polynomial degrees to be tried, and outputs a list of fitted models. The specifications for the main, feature matrix, and least squares functions are contained as comments in the skeleton code. The key steps are parsing the input data, creating the feature matrix, and solving the least squares equations.

Answer-1:

```
import numpy as np
import math as m
import matplotlib.pyplot as plt

#Return fitted model parameters to the dataset at datapath for each choice in degrees.
#Input: datapath as a string specifying a .txt file, degrees as a list of positive integers.
#Output: paramFits, a list with the same length as degrees, where paramFits[i] is the list of
#coefficients when fitting a polynomial of n = degrees[i].
def main(datapath, degrees):
    paramFits = []
    #fill in
    x = []
    y = []
    y_pred = []
    #read the input file, assuming it has two columns, where each row is of the form [x y] as
    #in problem1.txt.
    data = np.loadtxt(datapath)
    for i in range(len(data)):
        x.append(float(data[i][0]))
        y.append(float(data[i][1]))
    #iterate through each n in degrees, calling the feature_matrix and least_squares functions to solve
    #for the model parameters in each case. Append the result to paramFits each time.
    for i in range (len(degrees)):
        newB = (least_squares(feature_matrix(x,degrees[i]),y))
        paramFits.append(newB)
        #For plotting
        X = np.array(feature_matrix(x,degrees[i]))
        y_pred_value_list = np.dot(X,np.array(newB))
        y_pred_value_list.sort()
        y_pred.append(y_pred_value_list)
    x.sort()
    y.sort()
    plt.scatter(x,y,label='Data Set',color="black")
    plt.plot(x,y_pred[0],label ="Degree 1')
    plt.plot(x,y_pred[1],label ="Degree 2')
    plt.plot(x,y_pred[2],label ="Degree 3')
    plt.plot(x,y_pred[3],label ="Degree 4')
    plt.plot(x,y_pred[4],label ="Degree 5')

    plt.xlabel('degree values')
    plt.ylabel('paramFits (coefficients) unit')
    plt.title('Estimated paramFits values')
    plt.legend()
    plt.show()


    #Question 4
    netval = 0
    currenthighestdegree = 5
    for i in paramFits[4]:
        netval = netval + i* 2**currenthighestdegree
        currenthighestdegree = currenthighestdegree - 1
    print("The estimated y_pred value at x = 2 is " + str(netval) + "\n" )
    return paramFits


#Return the feature matrix for fitting a polynomial of degree n based on the explanatory variable
#samples in x.
#Input: x as a list of the independent variable samples, and n as an integer.
#Output: X, a list of features for each sample, where X[i][j] corresponds to the jth coefficient
#for the ith sample. Viewed as a matrix, X should have dimension #samples by n+1.
def feature_matrix(x, n):

    #fill in
    #There are several ways to write this function. The most efficient would be a nested list comprehension
    #which for each sample in x calculates x^n, x^(n-1), ..., x^0.
    X = []
    for i in range (len(x)):
        temp = []
        for k in range (n , -1 , -1):
            temp.append(x[i] ** k)
        X.append(temp)
    return X


#Return the least squares solution based on the feature matrix X and corresponding target variable samples in y.
#Input: X as a list of features for each sample, and y as a list of target variable samples.
#Output: B, a list of the fitted model parameters based on the least squares solution.
def least_squares(X, y):
    X = np.array(X)
    y = np.array(y)
    #fill in
    B = []
    #Use the matrix algebra functions in numpy to solve the least squares equations. This can be done in just one line.
    Btemp = np.dot(np.linalg.inv(np.dot(np.transpose(X),X)),np.transpose(X))
    for i in range (len(Btemp)):
       B.append(np.dot(Btemp[i],y))
    return B

if __name__ == '__main__':
    datapath = 'poly.txt'
    degrees = [1,2,3,4,5]

    paramFits = main(datapath, degrees)
    print(paramFits)
```

Result obtained after running the program:

```
$ python3 polyfit.py
[[7.001583333198903, 9.303864260428961, -239.3340329835962], [0.0059879637943775
, 0.7552180459273857, 0.2345598538372527, 1.1763636032243174, -175.8802882617245
3]]
```

Question-2: Use your completed polyfit.py to find fitted polynomial coefficients for n = 1, 2, 3, 4, 5 on the poly.txt dataset. Write out the resulting estimated functions y^n(x) for each n.

Answer-2:

For n = 1, we get the following result:

```
agarw184@ecegrid-thin1 ~/ECE20875/PA07
$ python3 polyfit.py
[[52.158053801747236, -189.86610574097708]]
```

For n = 2, we get the following result:

```
agarw184@ecegrid-thin1 ~/ECE20875/PA07
$ python3 polyfit.py
[[7.001583333198903, 9.303864260428961, -239.33403298359621]]
```

For n = 3, we get the following result:

```
agarw184@ecegrid-thin1 ~/ECE20875/PA07
$ python3 polyfit.py
[[0.8201380988526097, 0.2617671231585569, -0.010327670472147665, -175.2771320089171]]
```

For n = 4, we get the following result:

```
agarw184@ecegrid-thin1 ~/ECE20875/PA07
$ python3 polyfit.py
[[0.0059879637943775, 0.7552180459273857, 0.2345598538372527, 1.1763636032243174, -175.88028826172453]]
```

For n = 5, we get the following result:

```
agarw184@ecegrid-thin1 ~/ECE20875/PA07
$ python3 polyfit.py
[[0.0008531198983968841, -0.004698036593526814, 0.7528113266208547, 0.5260849950255904, 0.9659162450376468, -176.8373688921271]]

agarw184@ecegrid-thin1 ~/ECE20875/PA07
```

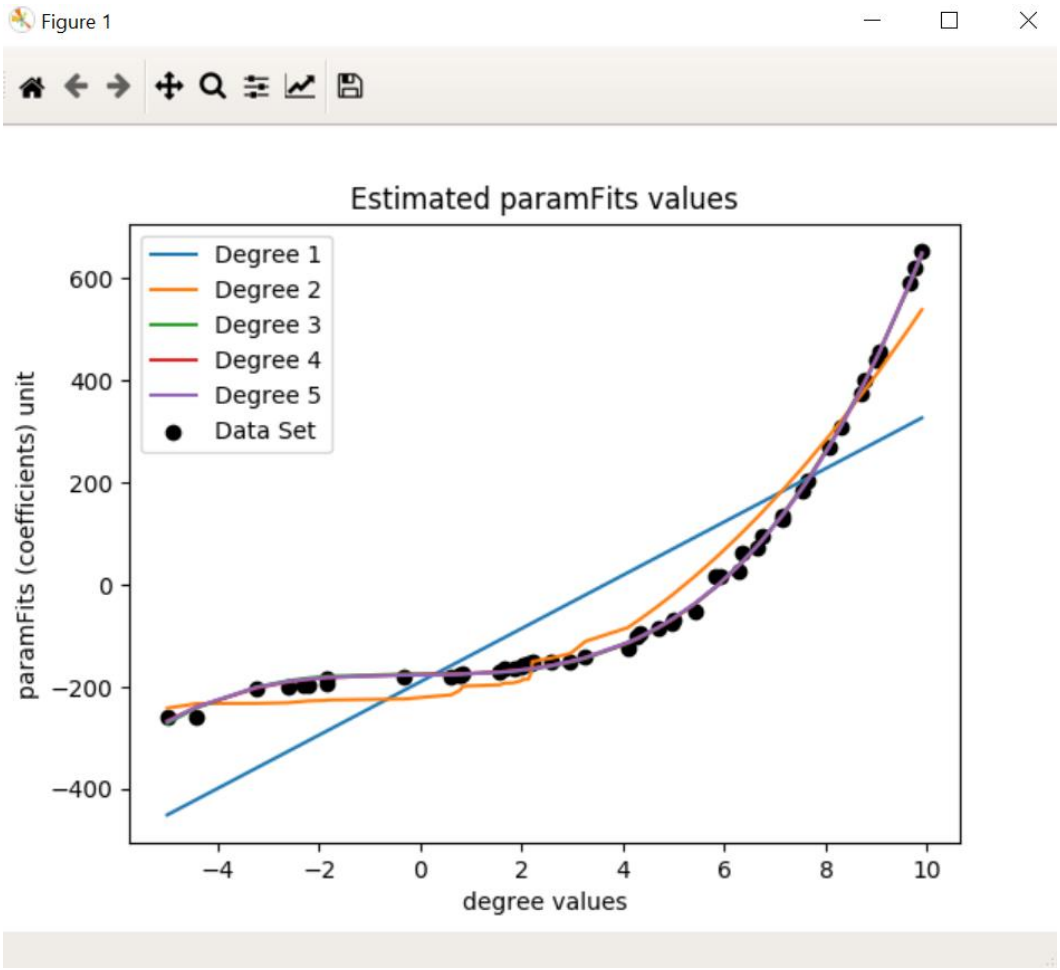
In totality, for n = [1,2,3,4,5], we get the following result:

```
$ python3 polyfit.py
[[52.158053801747236, -189.86610574097708], [7.001583333198903, 9.303864260428961, -239.3340329835962], [0.8201380988526097, 0.2617671231585569, -0.010327670472147665, -175.2771320089171
], [0.0059879637943775, 0.7552180459273857, 0.2345598538372527, 1.1763636032243174, -175.88028826172453], [0.0008531198983968841, -0.004698036593526814, 0.7528113266208547, 0.5260849950
255904, 0.9659162450376468, -176.837368892127]]
```

Question – 3 : Use the scatter and plot functions in the matplotlib.pyplot module to visualize the dataset and these fitted models on a single graph (i.e., for each x, plot y, ˆy1(x), ..., ˆy5(x)). Be sure to vary colors and include a legend so that each curve can be distinguished. What degree polynomial does the relationship seem to follow? Explain.

Answer-3:

Plot obtained:



Equations obtained for various degree values are as follows :

For n = 1:
$y_1(x) = 952.1580538017472(x) - 189.86610574097068$

For n = 2:
$y_2(x) = 7.0015833331989175(x^2) + 9.303864260428915(x) - 239.33403298359644$

For n = 3:
$y_3(x) = 0.820138098852604(x^3) + 0.26176712315855943(x^2) - 0.010327670472049966(x) - 175.27713200891736$

For n = 4:
$y_4(x) = 0.005987963794389907(x^4) + 0.7552180459272604(x^3) + 0.2345598538373384(x^2) + 1.176363603226044(x) - 175.88028826172624$

For n = 5:
$y_5(x) = 0.0008531198983950695(x^5) - 0.00469803659350717 7(x^4) + 0.7528113266207637(x^3) + 0.5260849950250703(x^2) + 0.9659162450397061(x) - 176.8373688921246$

Observation:
It is observed that the polynomial seems to follow the 5th degree order.
We decided that because it is observed that the dataset curve completely overlaps the obtained fit curve corresponding to degree being 5.

Question 4: If we measured a new datapoint x = 2, what would be the predicted value ˆy of y?

Answer 4:

Output obtained:

```
The estimated y_pred value at x = 2 is -166.82657455773023
```

Code Snippet Used:

```
#Question 4
netval = 0
currenthighestdegree = 5
for i in paramFits[4]:
    netval = netval + i* 2**currenthighestdegree
    currenthighestdegree = currenthighestdegree - 1
print("The estimated y_pred value at x = 2 is " + str(netval) + "\n" )
return paramFits
```