CPSC 552 – Data Minining - Assignment #6

Curve Fitting via Squared Error Optimization

In this assignment, you will program the curve fitting approximation via an nth degree polynomial. The problem will be set up as an optimization problem to solve for the coefficients of the polynomial such that the squared error between the given data and the fitted data is minimized.

Problem #1: Develop the mathematical equations and program the curve fitting of a given data by a third order polynomial using the least squared error optimization.

Solution:

If the data to be fitted is a third order polynomial, then the general equation for the curve is.

$$y = \beta_0 x \quad + \beta_1 x^1 + \beta_2 x^2 + \beta_3 x^3$$

where $\beta_0, \beta_1, \beta_2, \beta_3$ are the coefficients to be determined that optimally fit the given data.

For a given point x_i, the corresponding y_i is given as:

$$y_i = \beta_0 + \beta_1 x_i^1 + \beta_2 x_i^2 + \beta_3 x_i^3$$

The squared error between the actual and the polynomial estimate for *n* given data points will then be:

$$E = \sum_{i=1}^{n} (\beta_0 + \beta_1 x_i^1 + \beta_2 x_i^2 + \beta_3 x_i^3 - y_i)^2$$

To optimally solve for the β coefficients, we can take the partial derivative of the above equation and set it to 0. Thus, we will get four equations as:

$$\frac{\partial E}{\partial \beta_0} = 2(\beta_0 + \beta_1 x_i^1 + \beta_2 x_i^2 + \beta_3 x_i^3 - y_i)\, 1 = 0$$

$$\frac{\partial E}{\partial \beta_1} = 2(\beta_0 + \beta_1 x_i^1 + \beta_2 x_i^2 + \beta_3 x_i^3 - y_i)\, x_i^1 = 0$$

$$\frac{\partial E}{\partial \beta_2} = 2(\beta_0 + \beta_1 x_i^1 + \beta_2 x_i^2 + \beta_3 x_i^3 - y_i)\, x_i^2 = 0$$

$$\frac{\partial E}{\partial \beta_3} = 2(\beta_0 + \beta_1 x_i^1 + \beta_2 x_i^2 + \beta_3 x_i^3 - y_i)\, x_i^3 = 0$$

Which can be rewritten as:

$$\sum_{i=1}^{n} (\beta_0 + \beta_1 x_i^1 + \beta_2 x_i^2 + \beta_3 x_i^3) = \sum_{i=1}^{n} y_i$$

$$\sum_{i=1}^{n} (\beta_0 x_i^1 + \beta_1 x_i^2 + \beta_2 x_i^3 + \beta_3 x_i^4) = \sum_{i=1}^{n} y_i\, x_i^1$$

$$\sum_{i=1}^{n} (\beta_0 x_i^2 + \beta_1 x_i^3 + \beta_2 x_i^4 + \beta_3 x_i^5) = \sum_{i=1}^{n} y_i\, x_i^2$$

$$\sum_{i=1}^{n} (\beta_0 x_i^3 + \beta_1 x_i^4 + \beta_2 x_i^5 + \beta_3 x_i^6) = \sum_{i=1}^{n} y_i\, x_i^3$$

In matrix form, above four equations can be written as:

$$\sum \begin{bmatrix} 1 & x_i^1 & x_i^2 & x_i^3 \\ x_i^1 & x_i^2 & x_i^3 & x_i^4 \\ x_i^2 & x_i^3 & x_i^4 & x_i^5 \\ x_i^3 & x_i^4 & x_i^5 & x_i^6 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix} = \sum \begin{bmatrix} y_i \\ y_i x_i^1 \\ y_i x_i^2 \\ y_i x_i^3 \end{bmatrix}$$

Given *n* data points, each entry in the matrix on the left hand side and the vector on the right hand side is a summation over *n* points.
The matrix equation then becomes:

$$\boldsymbol{A\beta} = \boldsymbol{b}$$

Where $\boldsymbol{A} = \sum \begin{bmatrix} 1 & x_i^1 & x_i^2 & x_i^3 \\ x_i^1 & x_i^2 & x_i^3 & x_i^4 \\ x_i^2 & x_i^3 & x_i^4 & x_i^5 \\ x_i^3 & x_i^4 & x_i^5 & x_i^6 \end{bmatrix}$, $\boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix}$ and $\boldsymbol{b} = \sum \begin{bmatrix} y_i \\ y_i x_i^1 \\ y_i x_i^2 \\ y_i x_i^3 \end{bmatrix}$

We can solve for the unknown coefficients $\boldsymbol{\beta}$ as:

$$\boldsymbol{\beta} = \boldsymbol{A^{-1}b}$$

Since $\boldsymbol{A}$ is a square matrix, a normal inverse can be computed (instead of the pseudo inverse).

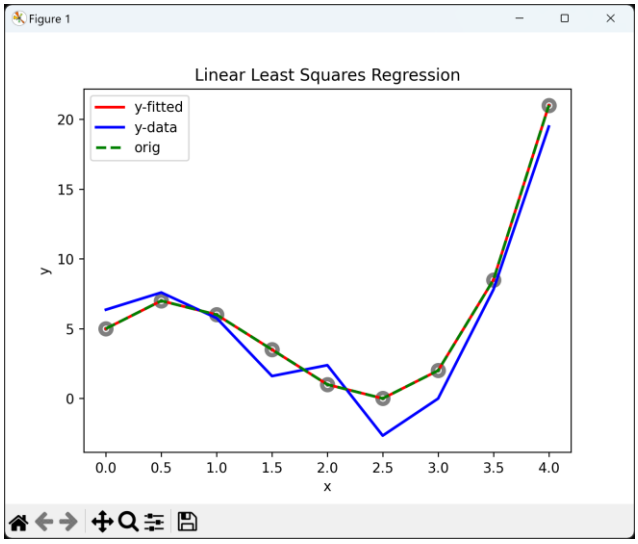
We will program the above approach.

Create a Python application called LeastSquaresRegression. Type the following code in the LeastSquaresRegression.py.

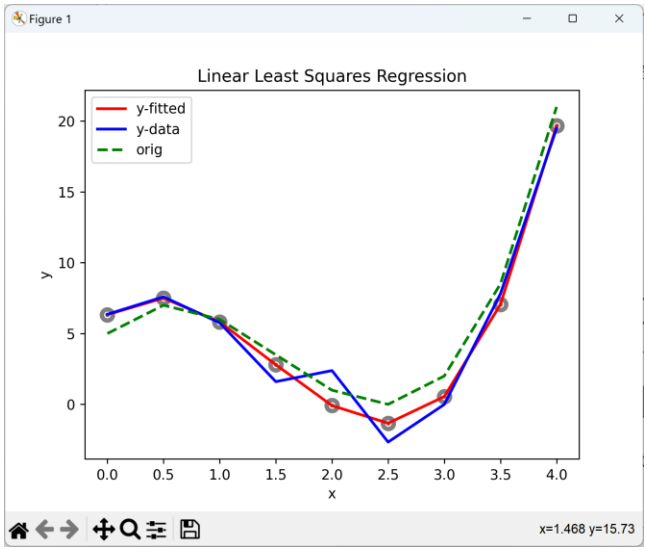
```
import sys
from tkinter.tix import Tree
import numpy as np
import matplotlib.pyplot as plt

def create_polynomial_data(x, poly_coeffs, add_noise):
    ynoise = np.random.normal(0, 2, len(x))
    #print(ynoise)
    y = poly_coeffs[0]*x**3 + poly_coeffs[1]*x**2 + poly_coeffs[2]*x +
poly_coeffs[3]
    if add_noise == True:
        noisy_y = y + ynoise
    else:
        noisy_y = y
    return y, noisy_y

def plot_data(x, y, y2, yorig):
    area = 10
    colors =['black']
    plt.scatter(x, y, s=area, c=colors, alpha=0.5, linewidths=8)
    plt.title('Pseudo Inverse - Linear Least Squares Regression')
    plt.xlabel('x')
    plt.ylabel('y')
    #plot the fitted line
```

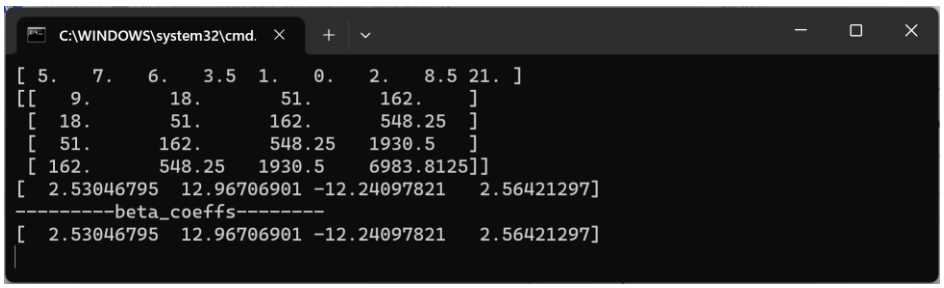
```
    line,=plt.plot(x, y, '-', linewidth=2, label="y-fitted")
    line.set_color('red')
    line,=plt.plot(x, y2, '-', linewidth=2, label="y-data")
    line.set_color('blue')
    line,=plt.plot(x, yorig, '--', linewidth=2, label="orig")
    line.set_color('green')
    plt.legend(loc="upper left")
    plt.show()


def solve_by_Linear_Least_Squares(x,y, poly_order):
    A = np.zeros((poly_order+1,poly_order+1))
    b = np.zeros((poly_order+1))
    for i in range(0,poly_order+1):
        for j in range(0,poly_order+1):
            sum = 0
            for k in range(0,len(x)): # sum the data
                sum = sum + (x[k]**j)*(x[k]**i)
            A[i,j] = sum
    print(A)
    for i in range(0,poly_order+1):
        sumy = 0
        for k in range(0,len(x)): # sum the data
            sumy = sumy + y[k] * (x[k]**i)
        b[i] = sumy
    # compute inverse of A
    ainv = np.linalg.inv(A)

    beta_coeffs = np.dot(ainv,b) # multiply inverse of A with b
    print(beta_coeffs)
    return beta_coeffs


def main():
    x = [0,0.5,1,1.5,2,2.5,3,3.5,4]
    x = np.asarray(x, float)
    poly_coeffs = [2, -9, 8, 5] # 2x^3 - 9x^2 + 8x + 5
    y, noisy_y = create_polynomial_data(x,poly_coeffs, True)
   # True means add noise, we are generating test data with noise
   # that has the starting polynomial of  2x^3 - 9x^2 + 8x + 5
    print(y)
    plot_data(x, y, noisy_y, y)

    beta_coeffs = solve_by_Linear_Least_Squares(x,noisy_y, 3)
    print('---------beta_coeffs--------')
    print(beta_coeffs)

    # np.flip reverses the coefficient array
    yfitted, noisy_y2 = create_polynomial_data(x,np.flip(beta_coeffs), False) #
false means do not add noise
    plot_data(x, yfitted, noisy_y, y) # noisy_y was the data we fitted


if __name__ == "__main__":
    sys.exit(int(main() or 0))
```

If you run the program, it will show the starting noisy data (shown in blue curve) as:



Once you close the above graph, the program will do the optimization of the coefficients and plot the estimated curve (red curve) for the noisy blue data.





Problem #2: Develop the mathematical equations and program the curve fitting of a given data by a fourth order polynomial using the least squared error optimization.