Numerical Methods

Lab 7 - Least Square Method - Polyfit

MAR 2022

Submitted by

Aabhusan Aryal

076BCT001

Submitted to

Department of Electronics and Computer Engineering,

Pulchowk Campus, IOE

POLYNOMIAL CURVE FITTING USING LEAST SQUARE METHOD

THEORY:

Consider k sets of data $(x_1, y_1), (x_2, y_2) \ldots (x_k, y_k)$. Then we can determine a best fit polynomial equation of degree n,

$$y = c_0 + c_1 x + c_2 x^2 + c_3 x^3 + \ldots + c_n x^n \quad \text{--- (a)}$$

using the least square method of curve fitting.

Using least square method, we obtain the equations,

$$\Sigma y = n c_0 + c_1 \Sigma x + c_2 \Sigma x^2 + \ldots + c_n \Sigma x^n \quad \text{--- (1)}$$

$$\Sigma xy = c_0 \Sigma x + c_1 \Sigma x^2 + c_2 \Sigma x^3 + \ldots + c_n \Sigma x^{n+1} \quad \text{--- (2)}$$

$$\vdots$$

$$\Sigma x^n y = c_0 \Sigma x^n + c_1 \Sigma x^{n+1} + c_2 \Sigma x^{n+2} + \ldots + c_n \Sigma x^{2n} \quad \text{--- (n)}$$

Solving these equations,

we can obtain $c_0, c_1, c_2, \ldots c_n$ and hence determine the best fit polynomial equation (a).

Now, writing eq^n (1), (2), ... (n) in matrix form, we get,

$$\begin{bmatrix} n & \Sigma x & \Sigma x^2 & \ldots & \Sigma x^n \\ \Sigma x & \Sigma x^2 & \Sigma x^3 & \ldots & \Sigma x^{n+1} \\ \Sigma x^2 & \Sigma x^3 & \Sigma x^4 & \ldots & \Sigma x^{n+2} \\ \vdots & & & & \\ \Sigma x^n & \Sigma x^{n+1} & \Sigma x^{n+2} & \ldots & \Sigma x^{2n} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} \Sigma y \\ \Sigma xy \\ \Sigma x^2 y \\ \vdots \\ \Sigma x^n y \end{bmatrix}$$

The augmented matrix can be written as:

$$\begin{bmatrix} \Sigma x^0 & \Sigma x & \Sigma x^2 & \ldots & \Sigma x^n \\ \Sigma x & \Sigma x^2 & \Sigma x^3 & \ldots & \Sigma x^{n+1} \\ \Sigma x^2 & \Sigma x^3 & \Sigma x^4 & \ldots & \Sigma x^{n+2} \\ \vdots & & & & \\ \Sigma x^n & \Sigma x^{n+1} & \Sigma x^{n+2} & \ldots & \Sigma x^{2n} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} \Sigma x^0 y \\ \Sigma x^1 y \\ \Sigma x^2 y \\ \vdots \\ \Sigma x^n y \end{bmatrix}$$

~~Now, the~~

$$\Rightarrow \left[\begin{array}{ccccc:c} \Sigma x^0 & \Sigma x & \Sigma x^2 & \ldots & \Sigma x^n & \Sigma x^0 y \\ \Sigma x & \Sigma x^2 & \Sigma x^3 & \ldots & \Sigma x^{n+1} & \Sigma x^1 y \\ \Sigma x^2 & \Sigma x^3 & \Sigma x^4 & \ldots & \Sigma x^{n+2} & \Sigma x^2 y \\ \vdots & & & & & \\ \Sigma x^n & \Sigma x^{n+1} & \Sigma x^{n+2} & \ldots & \Sigma x^{2n} & \Sigma x^n y \end{array}\right]$$

Now, using this augmented matrix, the solution can be easily found out using methods like Gauss Jordan Method

Algorithm:

1. Input no. of data (n), degree of polynomial (m) and data pairs (x_k, y_k).

2. Construction of augmented matrix (A) of order $(m+1)\times(m+2)$

```
For i = 0 to m+1
    A[i, m+2] = 0
    For j = 0 to m+1
        A[i,j] = 0
        For k = 0 to n
            sum = 0
            For p = 0 to n
                sum = sum + x_p ^ (i+j)
            Next p
            A[i,j] = A[i,j] + sum
        Next k
    Next j
    For k = 0 to n
        Sum = 0
        For p = 0 to n
        sum = sum + (x_p ^ i) * y_p
        Next p
        A[i, m+2] = A[i, m+2] + sum
    Next k
Next i
```

3. Solution of augmented matrix using Gauss Jordan Method (Say order of augmented matrix is $n\times(n+1)$)

```
For j = 1 to n
    For i = 1 to n
        If i ≠ j, then
            c = A[i,j] / A[j,j]
            For k = 1 to n+1
                A[i,k] = A[i,k] - c * A[j,k]
            Next k
        End if
    Next i
Next j
Let c0, c1, ... cn be the required solution
For i = 0 to n
    Ci = A[i,n] / A[i,i]
Next i
```

4. End

SOURCE CODE:

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <math.h>

#define MAXSIZE 20

typedef struct {
    float x;
    float y;
}Data;

float matrix[MAXSIZE][MAXSIZE];
Data data[MAXSIZE];

float sumx(uint32_t cap, uint32_t order){
    float sum=0;
    for(uint32_t i=0; i<cap; i++){
        sum += powf(data[i].x, order);
    }
    return sum;
}

float sumxy(uint32_t cap, uint32_t order){
    float sum=0;
    for(uint32_t i=0; i<cap; i++){
        sum += powf(data[i].x, order) * data[i].y;
    }
    return sum;

}
```

```
void enterData(uint32_t cap, uint32_t order) {
    for (uint32_t i=0; i<cap; i++) {
        printf("Enter x-%d, y-%d: ", i, i);
        scanf("%f%f", &data[i].x, &data[i].y);
    }

    for (uint32_t i=0; i<order+1; i++) {
        matrix[i][order+1] = 0;
        for (uint32_t j=0; j<order; j++) {
            matrix[i][j] = 0;
            for (uint32_t k=0; k<cap; k++) {
                matrix[i][j] += sumx(cap, i+j);
            }
        }
        for (uint32_t k=0; k<cap; k++) {
            matrix[i][order+1] += sumxy(cap, i);
        }
    }
}

void eliminate (uint32_t i, uint32_t j, uint32_t order) {
    float c = matrix[i][j] / matrix[j][j];
    for (uint32_t k=j; k<=order; k++) {
        matrix[i][k] -= c * matrix[j][k];
    }
}

void gaussJordan (uint32_t order) {
    for (uint32_t j=0; j<order; j++) {
        for (uint32_t i=0; i<order; i++) {
            if (i!=j) {
                eliminate (i, j, order);
            }
        }
    }
    float value[order];
```

```
    printf("y =");
    for (uint32_t i=0; i<order; i++){
        value[i]= matrix[i][order] / matrix[i][i];
        printf("%.4f * x^%d", value[i], i);
        if(i!= order -1){
            printf(" + ");
        }
    }
    printf("\n");
}

int main() {
    uint32_t order, cap;
    printf(" Enter the no of unknowns and order of equation: ");
    scanf("%d %d", &cap, &order);
    ~~gauss Jor~~
    enterData(cap, order);
    gaussJordan(order +1);
    return 0;
}
```

TEST CASES:

Case ①: Polynomial eqⁿ of order 2

```
Enter the no. of unknowns and order of equation: 5 2
Enter x_0, y_0 : 0 1
Enter x_1, y_1 : 1 3
Enter x_2, y_2 : 2 7
Enter x_3, y_3 : 3 13
Enter x_4, y_4   4 21
y = 1.0000 * x^0 + 1.0000 * x^1 + 1.0000 * x^2
```

Case ②: Polynomial equation of order 3

Enter the no. of unknowns and order of equation: 5 3
Enter x-0, y-0 : 2 1
Enter x-1, y-1 : 4 5
Enter x-2, y-2 : 7 9
Enter x-3, y-3 : 4 8
Enter x-4, y-4 : 5 6

y = -27.1695*x^0 +22.5524*x^1 + -4.9339*x^2+0.3500*x^3

Case ③: Polynomial equation of order 4

Enter the no. of unknowns and order of equation: 5 4
Enter x-0, y-0 : 15 14
Enter x-1, y-1 : 25 48
Enter x-2, y-2 : 75 499
Enter x-3, y-3 : 85 1456
Enter x-4, y-4 : 255 25589

y = -708.9996*x^0+89.4067*x^1+-3.2642*x^2 + 0.0378*x^3 +-0.0001*x^4

DISCUSSION:

We were assigned to write a program that finds the best fit polynomial curve to a given dataset using least square method. For that purpose, a program is written in C through Visual Studio Code and then compiled using GCC(GNU C Compiler). Different test data were given and the output was noted, which was as expected, which showed that the program worked.

CONCLUSION:

In this way, polynomial curve fitting was implemented using least square method.