1 Experiment No. 9

2 Experiment Title

Nonlinear polynomial curve fitting method using MATLAB.

3 Objective

The objectives of this lab are:

- To practice a nonlinear polynomial best curve fitting.
- To create a MATLAB program to find the variables according to the degree of the solving equation.

4 Theory

Curve fitting, or regression, is the process of constructing a curve or mathematical function that best fits a series of data points, possibly subject to constraints. It is a fundamental technique used in data analysis and modeling.
Curve fitting can be broadly classified into two categories:

1. Linear curve fitting
2. Nonlinear curve fitting

Nonlinear curve fitting can further be classified as:

- Exponential curve fitting
- Polynomial curve fitting

In this experiment, we focus on polynomial curve fitting. Let us illustrate this method by fitting a given set of data to a quadratic polynomial. Let the quadratic curve be represented by:

$$y = a_2x^2 + a_1x + a_0 \tag{9.1}$$

For each data point where $x = x_i$, the left-hand side of Equation (9.1) becomes:

$$\bar{y}_i = a_2x_i^2 + a_1x_i + a_0 \tag{9.2}$$

The sum of the squares of the deviations (errors) between the actual and estimated values is given by:

$$S = \sum(y_i - \bar{y}_i)^2 = \sum(y_i - a_2x_i^2 - a_1x_i - a_0)^2 \tag{9.3}$$

To find the best-fitting curve, we minimize S with respect to the coefficients a_0, a_1, and a_2. This is done by taking the partial derivatives of S with respect to each coefficient and setting them equal to zero:

$$na_0 + a_1\sum x_i + a_2\sum x_i^2 = \sum y_i \tag{9.4}$$

$$a_0\sum x_i + a_1\sum x_i^2 + a_2\sum x_i^3 = \sum x_iy_i \tag{9.5}$$

$$a_0 \sum x_i^2 + a_1 \sum x_i^3 + a_2 \sum x_i^4 = \sum x_i^2 y_i \tag{9.6}$$

These form a system of three linear equations with three unknowns: a_0, a_1, and a_2. Solving this system gives the coefficients of the best-fit quadratic polynomial.
This basic method can be generalized to fit an n^{th} order polynomial. In that case, there will be $n + 1$ simultaneous equations containing $n + 1$ unknown constants corresponding to the polynomial coefficients.

5 Algorithm

Input: Data points (x, y), order of the polynomial n, and number of data points m
Output: Coefficients of the best-fitting polynomial and visualization of the fitting curve

1. Read the order of the polynomial: n

   Set the number of coefficients: $c_n \leftarrow n + 1$

2. Read the x-values of the data points: x

   Read the y-values of the data points: y

3. Read the number of data points: m

4. **Constructing the Right-Hand Side of the Linear System (vector b):**

   (a) For $i = 1$ to c_n:

      i. Initialize sum $\leftarrow 0$

      ii. For $j = 1$ to m:

         A. sum $\leftarrow$ sum $+ x[j]^{(i-1)} \cdot y[j]$

      iii. End For

      iv. $b[i] \leftarrow$ sum

   (b) End For

5. **Constructing the Left-Hand Side of the Linear System (matrix C):**

   (a) For $i = 1$ to c_n:

      i. For $k = 1$ to c_n:

         A. Initialize sum $\leftarrow 0$

         B. For $j = 1$ to m:

            - sum $\leftarrow$ sum $+ x[j]^{(i+k-2)}$

         C. End For

         D. $C[i, k] \leftarrow$ sum

      ii. End For

   (b) End For

6. Compute the coefficient vector:

$$a \leftarrow C^{-1} \cdot b^T$$

7. **Determining Polynomial Coefficients:**

(a) For $i = 1$ to c_n:

$$a_n[i] \leftarrow a[c_n - i + 1]$$

(b) End For

8. Display the polynomial coefficients a_n

9. **Plotting:**

   (a) Generate $x_1 \leftarrow$ linspace$(0, \max(x), 100)$

   (b) Compute $y_1 \leftarrow$ evaluate polynomial a_n at x_1

   (c) Plot the data points (x, y) as red circles

   (d) Plot the fitting curve (x_1, y_1)

   (e) Plot the original data points (x, y)

   (f) Generate $x_m \leftarrow$ evenly spaced points from 0 to 2π

   (g) Compute $y_m \leftarrow \sin(x_m)$

   (h) Plot the sine curve (x_m, y_m)

10. **END**

5.1 MATLAB Implementation of Polynomial Curve Fitting

5.1.1 MATLAB Code:

```
n = input('Enter the order of the polynomial (n): ');
%n=10;
cn = n + 1;
%x = input('Enter the x values  ');
%y = input('Enter the y values  ');
x = [0 0.5 1 1.5 2 2.5 3 3.5 4 4.5 5 5.5 6];
y = [0 0.4794 0.8414 0.9974 0.9092 0.5984 0.1411 -0.3504 -0.7568 -0.9775
    -0.95892 -0.7055 0.2794];
m = length(x);
b = zeros(cn, 1);

for i = 1:cn
sum = 0;
for j = 1:m
sum = sum + (x(j)^(i - 1)) * y(j);
end
b(i) = sum;
end

c = zeros(cn, cn);
for i = 1:cn
for k = 1:cn
sum = 0;
for j = 1:m
sum = sum + x(j)^((i + k) - 2);
end
c(i, k) = sum;
end
end
a = inv(c) * b

an = zeros(1, cn);
for i = 1:cn
an(i) = a(cn - i + 1);
end

disp('Polynomial coefficients (highest degree first):');
disp(an);
x1 = linspace(0, max(x), 150);
y1 = polyval(an, x1);
plot(x, y, 'ro', 'MarkerFaceColor', 'r');
hold on;
plot(x1, y1, 'b', 'LineWidth', 2);
legend('Data points', 'Fitted Polynomial');
title('Polynomial Curve Fitting');
xlabel('x');
ylabel('y');
grid on;
```

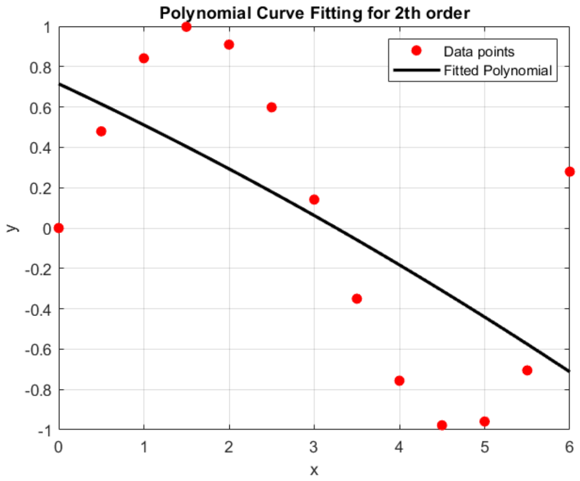
Listing 1: MATLAB code for Polynomial Curve Fitting

5.2 Output

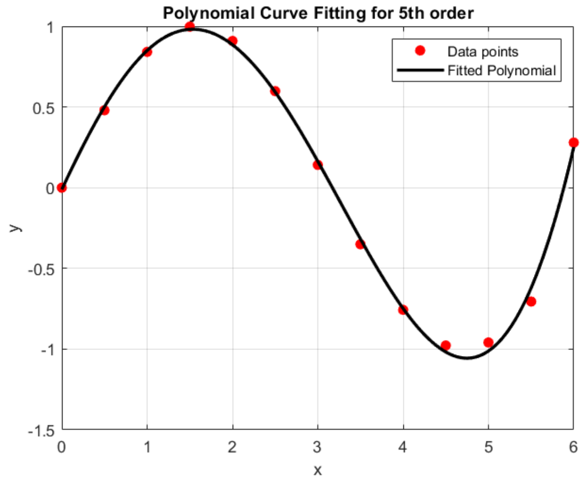
```
Enter the order of the polynomial (n): 2
Enter the order of the polynomial (n): 5
Enter the order of the polynomial (n): 10
Enter the order of the polynomial (n): 11
>>
```

Listing 2: Command Window

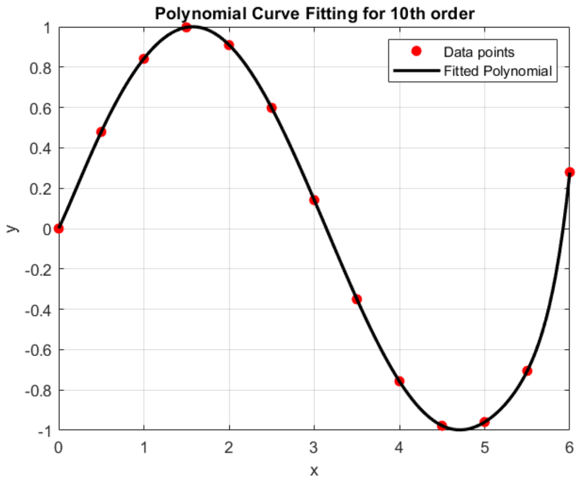
5.2.1 Plot Diagram



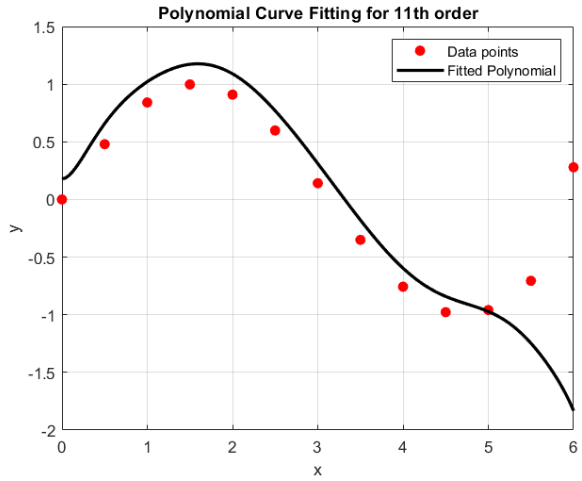
(a) Curve fitting for 2^{nd} order



(b) Curve fitting for 5^{th} order



(c) Curve fitting for 10^{th} order



(d) Curve fitting for 11^{th} order

Figure 1: Plot diagram for different order of polynomial curve fitting.

6 Discussion

In this experiment, polynomial curve fitting was applied using various polynomial orders to approximate a set of data points. Polynomial curve fitting was applied using 2nd, 5th, 10th, and 11th order polynomials. The second-order fit resembled a straight line and failed to capture data variation, indicating under-fitting. The fifth-order polynomial provided a better fit by following the data more closely.
The tenth-order polynomial gave the best result, accurately fitting the data with minimal deviation. However, the eleventh-order fit showed severe oscillations and poor accuracy due to **over-fitting**, where the curve attempted to pass through all points, causing instability. It was concluded that increasing the polynomial order improves fitting up to a point, after which over-fitting degrades performance.