GEBZE TECHNICAL UNIVERSITY

ELECTRONIC ENGINEERING

MATH214 – NUMERICAL ANALYSIS

Alican Bayındır
200102002087

Project - 5

Preparation Date
05.01.2021
Upload Date
06.01.2021

Problem definition and formulas;

The received signal level at the terminals of a receiver antenna located between 1-2 km away from the transmitting antenna is measured with 100 m steps. The measured values are plotted in Figure 1 and given in the data file (pr5data.dat), where the first column is the distance in m and the second column is the measured signal level in V. Note that the received signal is distorted due to noise, environmental effects, and polarization mismatches. The formulas to be used in this Project can be seen as following;

- Linear Least Squares Approximation

$$y_i = a_1 x_i + a_0$$

Where yi is an dependent variable that is not exactly known and xi is and independent variable that is known. The most probable values of a_0 (intercept) and a_1 (slope) can be estimated from a set of n pairs of experimental data (x_1, y_1), (x_2, y_2)…, (xn, yn), where y-values are contaminated with a normally distributed - zero mean random error (e.g. noise, experimental uncertainty). This estimation is known as least-squares linear regression.

Least-squares linear regression is only a partial case of least-squares polynomial regression analysis. By implementing this analysis, it is easy to fit any polynomial of m degree To find the a_0 and a_1 the following equation can be used;

$$a_0 = \frac{\sum_{i=1}^{m} x_i^2 \sum_{i=1}^{m} y_i - \sum_{i=1}^{m} x_i y_i . \sum_{i=1}^{m} x_i}{m(\sum_{i=1}^{m} x_i^2) - (\sum_{i=1}^{m} x_i)^2}$$

$$a_1 = \frac{m\sum_{i=1}^{m} x_i y_i - \sum_{i=1}^{m} x_i . \sum_{i=1}^{m} y_i}{m(\sum_{i=1}^{m} x_i^2) - (\sum_{i=1}^{m} x_i)^2}$$

- Polynomial Least Squares Approximation

$$P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

$$a_0 \sum_{i=1}^{m} x_i^0 + a_1 \sum_{i=1}^{m} x_i^1 + a_2 \sum_{i=1}^{m} x_i^2 + \cdots + a_n \sum_{i=1}^{m} x_i^n = \sum_{i=1}^{m} y_i x_i^0$$

$$a_0 \sum_{i=1}^{m} x_i^1 + a_1 \sum_{i=1}^{m} x_i^2 + a_2 \sum_{i=1}^{m} x_i^3 + \cdots + a_n \sum_{i=1}^{m} x_i^{n+1} = \sum_{i=1}^{m} y_i x_i^1$$

$$\vdots$$

$$a_0 \sum_{i=1}^{m} x_i^n + a_1 \sum_{i=1}^{m} x_i^{n+1} + a_2 \sum_{i=1}^{m} x_i^{n+2} + \cdots + a_n \sum_{i=1}^{m} x_i^{2n} = \sum_{i=1}^{m} y_i x_i^n$$

Codes, Inputs and Discussion;

First, the constant values and interface of the script can be described in MATLAB platform as follows;

```
%% Alican Bayındır 200102002087
% MATH 214 - Project 5
% 05.01.2021
close all; clear all; clc;
format short;
load pr5data.dat;

kilometer_values = pr5data(:, 1)'./1000;
voltage_values = pr5data(:, 2)';

x = 0; x2 = 0; y = 0; xy = 0; elsize = 11;
```

Figure 1 Defining constants and loading .dat file into MATLAB Platform.

As a result of this part of the code we have loaded the data into the MATLAB platform and specified the constant values to be used in future operations in script.

```
for i = 1:elsize
    x = x + (kilometer_values(1,i));
    y = y + (voltage_values(1,i));
    xy = xy + (kilometer_values(1,i) * voltage_values(1,i));
    x2 = x2 + (kilometer_values(1,i) * kilometer_values(1,i));
end

a0 = ((x2 * y) - (xy * x)) / ((elsize * x2) - ((x) * (x)));
a1 = ((elsize * xy) - (x * y)) / ((elsize * x2) - ((x) * (x)));

yi = [];
for i = 1:elsize
    yi = [yi (a1 * kilometer_values(1,i) + a0)];
end
```

Figure 2 The code part above finds the significant values such as a0 and a1 for approximation methods. (Linear Least Square approximation.)

The code in Figure 2 calculates the Linear Least Squares and the codes seen in Figure 3 and 4 is almost the same method. The difference between them is only “n” which is the degree of the polynomial. The figure 3 and 4 calculates the linear least squares degree 2 and 3.

```
x3 = 0;
x4 = 0;
yx2 = 0;
for i = 1:elsize
    x3 = x3 + (kilometer_values(1,i) * kilometer_values(1,i) * kilometer_values(1,i));
    x4 = x4 + (kilometer_values(1,i) * kilometer_values(1,i) * kilometer_values(1,i) * kilometer_values(1,i));
    yx2 = yx2 + (voltage_values(1,i) * kilometer_values(1,i) * kilometer_values(1,i));
end

A = [elsize x x2; x x2 x3; x2 x3 x4];
S = [y; xy; yx2];
B = inv(A) * S;

Y = [];
for i = 1:elsize
    Y = [Y ((B(3,1) * kilometer_values(1,i) * kilometer_values(1,i)) + (B(2,1) * kilometer_values(1,i) + (B(1,1))))];
end
```

Figure 3 The code part above is to find second degree polynomial least squares approximation.

```
x5 = 0;
for i = 1:elsize
    x5 = x5 + (kilometer_values(1,i) * kilometer_values(1,i) * ...
    kilometer_values(1,i) * kilometer_values(1,i) * kilometer_values(1,i));
end

x6=0;
for i = 1:elsize
    x6 = x6 + (kilometer_values(1,i) * kilometer_values(1,i) * kilometer_values(1,i) ...
    * kilometer_values(1,i) * kilometer_values(1,i) * kilometer_values(1,i));
end

yx3 = 0;
for i = 1:elsize
    yx3 = yx3 + (voltage_values(1,i) * kilometer_values(1,i) * kilometer_values(1,i) * kilometer_values(1,i));
end
```

Figure 4 The code part above is to find the third degree of polynomial least squares approximation.

After running the codes above, the plot that is seen in Figure 5 will be seen. It can be understood from the plot that the results we found via code are really accurate and the calculation of the error rates can be seen in the figure below;

```
error_degree_linear = []; error_degree_two = []; error_degree_three = [];
for i = 1:elsize
    error_degree_linear = [error_degree_linear (voltage_values(1,i)-yi(1,i))];
    error_degree_two = [error_degree_two (voltage_values(1,i)-Y(1,i))];
    error_degree_three = [error_degree_three (voltage_values(1,i)-Y2(1,i))];
end
```

Figure 5 Error calculation.

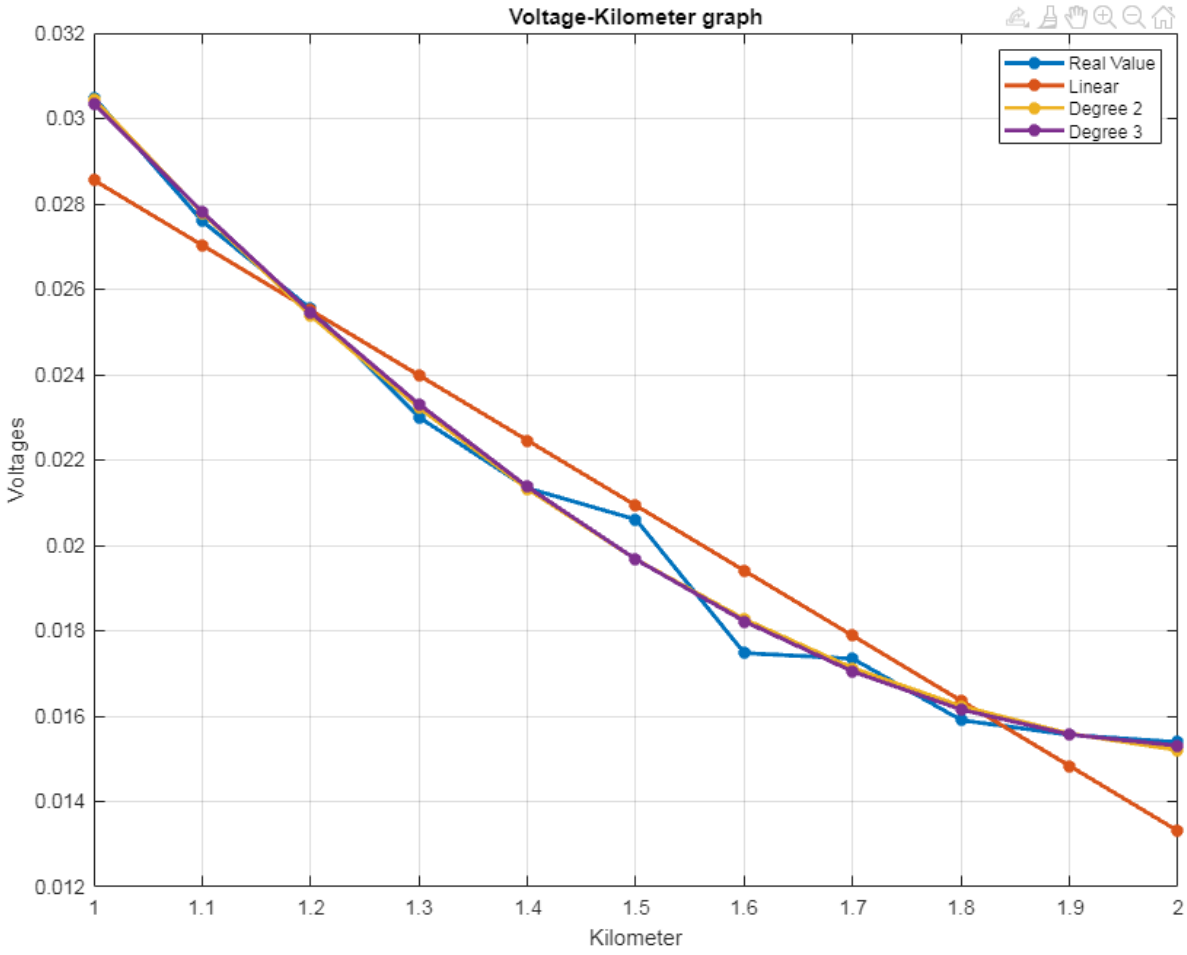

Figure 6 The output of the code.

Coefficients of the polynomials;

Cofficient's name	Linear	Degree 2	Degree 3
a_0	4.3805×10^{-2}	7.0946×10^{-2}	6.1828×10^{-2}
a_1	-1.5249×10^{-5}	-5.3120×10^{-5}	-3.3840×10^{-5}
a_2		1.2623×10^{-8}	-5.7805×10^{-10}
a_3			2.9337×10^{-12}

Error rates of each approximations;

Iterations	Linear	2nd degree	3rd degree
1	0.0019	0.0459	0.1516
2	0.0006	-0.1942	-0.2153
3	0	0.1587	0.0813
4	-0.001	-0.2261	-0.3071
5	-0.0011	0.0201	-0.0292
6	-0.0003	0.9357	0.9357
7	-0.0019	-0.7945	-0.7452
8	-0.0005	0.2203	0.3012
9	-0.0005	-0.326	-0.2485
10	0.0007	-0.0291	-0.0079
11	0.0021	0.189	0.0834

In this Project, the results we found were really accurate. Discrete Least Square approximation used to approximate to the data that are given in pr5data.dat file. As seen in plot (Figure 6). The approximation methods were really succesful. Moreover, the most succesfull Discrete Least Square approximation method was Linear Square Method. The reason of this is that the coefficients decrease while degrees getting higher and the total distinction between the higher degree and smaller degree is getting closer.

All codes that can be seen in APPENDIX part are written by me.

APPENDIX

```
%% Alican Bayındır 200102002087
% MATH 214 - Project 5
% 05.01.2021
close all; clear all; clc;
format short;
load pr5data.dat;


kilometer_values = pr5data(:, 1)'./1000;
voltage_values = pr5data(:, 2)';


x = 0; x2 = 0; y = 0; xy = 0; elsize = 11;


for i = 1:elsize
    x = x + (kilometer_values(1,i));
    y = y + (voltage_values(1,i));
    xy = xy + (kilometer_values(1,i) * voltage_values(1,i));
    x2 = x2 + (kilometer_values(1,i) * kilometer_values(1,i));
end


a0 = ((x2 * y) - (xy * x)) / ((elsize * x2) - ((x) * (x)));
a1 = ((elsize * xy) - (x * y)) / ((elsize * x2) - ((x) * (x)));


yi = [];
for i = 1:elsize
    yi = [yi (a1 * kilometer_values(1,i) + a0)];
end


x3 = 0;
x4 = 0;
yx2 = 0;
for i = 1:elsize
    x3 = x3 + (kilometer_values(1,i) * kilometer_values(1,i) * kilometer_values(1,i));
    x4 = x4 + (kilometer_values(1,i) * kilometer_values(1,i) * kilometer_values(1,i) *
kilometer_values(1,i));
    yx2 = yx2 + (voltage_values(1,i) * kilometer_values(1,i) * kilometer_values(1,i));
end


A = [elsize x x2; x x2 x3; x2 x3 x4];
S = [y; xy; yx2];
B = inv(A) * S;


Y = [];
for i = 1:elsize
    Y = [Y ((B(3,1) * kilometer_values(1,i) * kilometer_values(1,i)) + (B(2,1) *
kilometer_values(1,i) + (B(1,1))))];
end


plot(kilometer_values, voltage_values, '-*', 'LineWidth', 2);
hold on;
```

```
plot(kilometer_values, yi, '-*', 'LineWidth', 2);
hold on;


plot(kilometer_values, Y,'-*', 'LineWidth', 2);
xlabel('Kilometer'); ylabel('Voltages'); grid on;
title('Voltage-Kilometer graph');
hold on;


x5 = 0;
for i = 1:elsize
    x5 = x5 + (kilometer_values(1,i) * kilometer_values(1,i) * ...
    kilometer_values(1,i) * kilometer_values(1,i) * kilometer_values(1,i));
end


x6=0;
for i = 1:elsize
    x6 = x6 + (kilometer_values(1,i) * kilometer_values(1,i) * kilometer_values(1,i) ...
    * kilometer_values(1,i) * kilometer_values(1,i) * kilometer_values(1,i));
end


yx3 = 0;
for i = 1:elsize
    yx3 = yx3 + (voltage_values(1,i) * kilometer_values(1,i) * kilometer_values(1,i) *
kilometer_values(1,i));
end


A2 = [elsize x x2 x3; x x2 x3 x4; x2 x3 x4 x5; x3 x4 x5 x6];
S2 = [y; xy; yx2; yx3];
B2 = inv(A2) * S2;


Y2  = [];
for i = 1:elsize
    Y2 = [Y2 ((B2(4,1) * kilometer_values(1,i) * kilometer_values(1,i) * ...
    kilometer_values(1,i)) + (B2(3,1) * kilometer_values(1,i) * ...
    kilometer_values(1,i)) + (B2(2,1) * kilometer_values(1,i)+(B2(1,1))))];
end
plot(kilometer_values, Y2,'-*', 'LineWidth', 2);
legend('Real Value', 'Linear', 'Degree 2', 'Degree 3');


error_degree_linear = []; error_degree_two = []; error_degree_three = [];
for i = 1:elsize
    error_degree_linear = [error_degree_linear (voltage_values(1,i)-yi(1,i))];
    error_degree_two = [error_degree_two (voltage_values(1,i)-Y(1,i))];
    error_degree_three = [error_degree_three (voltage_values(1,i)-Y2(1,i))];
end
```