GEBZE TEKNİK ÜNİVERSİTESİ

DEPARTMENT OF ELECTRONIC ENGINEERING

MATH214 NUMERICAL METHODS

2020 – 2021 FALL TERM

Project 5

1801022001	<u>Preparation date</u>	<u>Upload date</u>
Ömer Sarımeşe	01.01.2021	03.01.2021

I. Problem Definition and Formulation

The main purpose of this project is determining possibly the best lines to represent given data. Signal level by distance is given in data file. These values can be assumed as xy-coordinates. In this report, lines will be created as linear, second degree, third degree polynomials. The formula for finding linear least square polynomial is

$$y_i = a_1 x_i + a_0 \tag{1}$$

$$a_0 = \frac{\sum_{i=1}^{m} x_i{}^2 . \sum_{i=1}^{m} y_i - \sum_{i=1}^{m} x_i . y_i . \sum_{i=1}^{m} x_i}{m(\sum_{i=1}^{m} x_i{}^2) - (\sum_{i=1}^{m} x_i)^2} \ , \ a_1 = \frac{\sum_{i=1}^{m} x_i y_i - \sum_{i=1}^{m} x_i . \sum_{i=1}^{m} y_i}{m(\sum_{i=1}^{m} x_i{}^2) - (\sum_{i=1}^{m} x_i)^2}$$

In this formula(1), a_0 , a_1 are coefficients. This formula gives linear polynomial. The second method is n-th-degree least square polynomial. Formula for that is

$$\sum_{k=0}^{n} a_k . \sum_{i=1}^{m} x_i^{j+k} = \sum_{i=1}^{m} y_i x_i^j \tag{2}$$

In formula(2), j value is going from zero to n. There will be n+1 number of normal equations and n+1 unknown coefficients a_j . For instance, there should be 3 equations with 3 unknown to find second degree polynomial. To find these coefficients, equations are solved using inverse matrix. Error value of these methods are same and can be shown as

$$E = \sum_{i=1}^{m} (y_i - p(x_i))^2 \ , \ p(x_i) = a_n x_i^n + a_{n-1} x_i^{n-1} + \cdots + a_1 x + a_0 \tag{3}$$

Formula(3) is using to find error of line. There formulas will be using in the following sections.

II. Matlab Code Explanation

Code gives plot of lines and error values as output. Other values like coefficients and sum of series are shown at workspace section in matlab. pr5data.dat file is imported and first column assigned to x array, second column assigned to y array with little changes. To determine linear line, sum of series are calculating with for loops, then constants are finding by formula(1). Same method is using for the n-th degree polynomial line fitting. But this time, another for loop is required to calculate first equation. Other equations are calculated by same for loop. The reason is matlab can not start for loop with initial value as zero. Also array indices must be positive integers. After finding coefficients, code convert matrix to inverse of it and multiply by right side. After the lines between 7-101, Lines are plotting and error values are calculating. Error values finding by formula(3). Code does not produces warnings and works perfectly.

III. Project Results and Discussions

Linear polynomial fit section have six data which are xi, yi, xi^2, xiyi, p(xi) and yi-p(xi). If we look at Table-1 which is located below, we can understand this method better. These method's error should be more than n-th degree polynomial methods. Second degree polynomial method has xi, yi, p(xi) and yi-p(xi) values represented by Table-2. Third degree polynomial method's table has the same column names as second degree polynomial. Third degree polynomial method perform slightly better than the second degree polynomial method. Also both of them are has significantly less error than linear polynomial method.

x_i	y_i	${x_i}^2$	$x_i y_i$	$p(x_i)$	$y_i - p(x_i)$
1.0000	30.49564	1.00000	30.49564	28.55612	1.93951
1.1000	27.59442	1.21000	30.35386	27.03117	0.56324
1.2000	25.53873	1.44000	30.64647	25.50622	0.03250
1.3000	22.99776	1.69000	29.89708	23.98127	-0.98351
1.4000	21.34025	1.96000	29.87634	22.45631	-1.11606
1.5000	20.60466	2.25000	30.90699	20.93136	-0.32670
1.6000	17.47580	2.56000	27.96128	19.40641	-1.93060
1.7000	17.34430	2.89000	29.48532	17.88146	-0.53715
1.8000	15.90431	3.24000	28.62775	16.35650	-0.45219
1.9000	15.55992	3.61000	29.56381	14.83155	0.72836
2.0000	15.38920	4.00000	30.77841	13.30660	2.08260

Table-1: Linear Polynomial Results

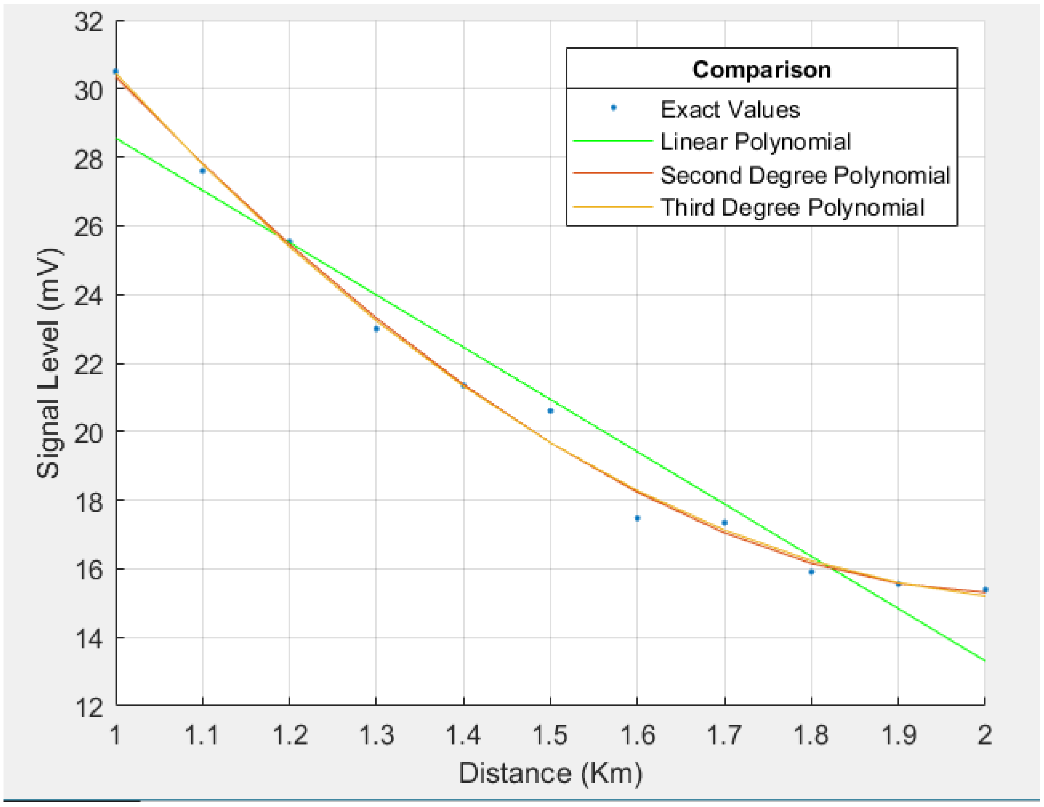
x_i	y_i	$p(x_i)$	$y_i - p(x_i)$
1.0000	30.49564	30.44969	0.045943
1.1000	27.59442	27.78860	-0.19418
1.2000	25.53873	25.37998	0.15874
1.3000	22.99776	23.22384	-0.22608
1.4000	21.34025	21.32017	0.02007
1.5000	20.60466	19.66898	0.93567
1.6000	17.47580	18.27027	-0.79446
1.7000	17.34430	17.12403	0.22027
1.8000	15.90431	16.23026	-0.32595
1.9000	15.55992	15.58898	-0.02906
2.0000	15.38920	15.20017	0.18903

Table-2: Second Degree Polynomial Results

x_i	y_i	$p(x_i)$	$y_i - p(x_i)$
1.0000	30.49564	30.34408	0.15155
1.1000	27.59442	27.80972	-0.21530
1.2000	25.53873	25.45743	0.08129
1.3000	22.99776	23.30481	-0.30705
1.4000	21.34025	21.36946	-0.02921
1.5000	20.60466	19.66898	0.93567
1.6000	17.47580	18.22098	-0.74517
1.7000	17.34430	17.04306	0.30124
1.8000	15.90431	16.15281	-0.24850
1.9000	15.55992	15.56786	-0.00793
2.0000	15.38920	15.30578	0.08341

Table-3: Third Degree Polynomial Results

Error values can be assume as $y_i - p(x_i)$ values at tables. But it makes comparison between different degree polynomial methods harder. So for that reason, Formula(3) has used. Error of linear polynomial method is founded 15.4876. Error of second degree polynomial method is founded 1.81455 . Last one which is Third degree polynomial method's error is 1.76138. However, The results are similar to table's implications of error. Higher degree polynomial fit suits better than less degree one. This can be seen at graph-1 which is located below.



Graph-1: Comparison between fitted lines

Second and Third degree polynomial fits are looks like overlap to each other. Although, Third degree polynomial line is fitted moderately better because of the order. In this case, all lines are quite ok.

If we know which degree of polynomial can be suit better to our data set, then these methods will be useful. I think smaller data sets can cause lots of problem and misleading curves at higher order. Also these lines should go between the points, not diverge to points. Interpolation will make these lines much better.

MATLAB CODE

```
clear all;
close all;
format long;
f=importdata('pr5data.dat');
x=f(:,1)/1000;
y=f(:,2)*1000;
%----------linear-----------
m=length(f);
a0=0;
a1=0;
yi=y(1);
xi=x(1);
xi2=x(1)^2;
xiyi=x(1)*y(1);
for j=2:1:m
    xi2=(x(j)^2)+xi2;
end
for j=2:1:m
    xiyi=(x(j)*y(j))+xiyi;
end
for j=2:1:m
    yi=y(j)+yi;
end
for j=2:1:m
    xi=x(j)+xi;
end
a0=((xi2*yi)-(xiyi*xi))/((m*xi2)-xi^2);
a1=((m*xiyi)-(xi*yi))/((m*xi2)-xi^2);
yl=zeros(1,m);
for i=1:1:m
    yl(i)=(a1*x(i))+a0;
end
%--------degree n=3------
n=3;
A=zeros(n+1,n+1);

for a=1:1:n+1                     % For the First Equation
    xi=x(1)^(a-1);                % because of matlab cannot start
loop
    for t=2:1:m                   % with initial value equals
zero.
        xi=xi+(x(t)^(a-1));
    end
    A(1,a)=xi;
end

for j=1:1:n                       % For the (2...n+1)th Equations
    count=1;
    for i=j:1:j+n
```

```
        xi=x(1)^i;
        for k=2:1:m
            xi=xi+(x(k)^(i));
        end
        A(j+1,count)=xi;
        count=count+1;
    end
end
B=zeros(n+1,1);
for c=1:1:n+1
    yixi=y(1)*(x(1)^(c-1));
    for t=2:1:m
        yixi=yixi+(y(t)*(x(t)^(c-1)));
    end
    B(c,1)=yixi;
end
inv_A=inv(A);
Coef=(inv_A)*B;
p=[Coef(4) Coef(3) Coef(2) Coef(1)];
%------------------------n=2---------------------------------
n=2;
A2=zeros(n+1,n+1);

for a=1:1:n+1                    % For the First Equation
    xi=x(1)^(a-1);               % because of matlab cannot start
loop
    for t=2:1:m                  % with initial value equals
zero.
        xi=xi+(x(t)^(a-1));
    end
    A2(1,a)=xi;
end

for j=1:1:n                      % For the (2...n+1)th Equations
    count=1;
    for i=j:1:j+n
        xi=x(1)^i;
        for k=2:1:m
            xi=xi+(x(k)^(i));
        end
        A2(j+1,count)=xi;
        count=count+1;
    end
end
B2=zeros(n+1,1);
for c=1:1:n+1
    yixi=y(1)*(x(1)^(c-1));
    for t=2:1:m
        yixi=yixi+(y(t)*(x(t)^(c-1)));
    end
    B2(c,1)=yixi;
```

```
end
inv_A2=inv(A2);
Coef2=(inv_A2)*B2;
p2=[Coef2(3) Coef2(2) Coef2(1)];
%------------plot-----------------
figure
grid on;
hold on;
plot(x,y,'.');
plot(x,yl,'g');
plot(x,polyval(p,x));
plot(x,polyval(p2,x));
xlabel('Distance (Km)');
ylabel('Signal Level (mV)');
a=legend('Exact Values','Linear Polynomial','Second Degree
Polynomial','Third Degree Polynomial');
title(a,'Comparison');
%------------Error------------
E_linear=0;
for i=1:1:m
    E_linear=E_linear+(y(i)-yl(i))^2;
end
display(E_linear);

E_poly2=0;
for i=1:1:m
    pxi=Coef2(1)+Coef2(2)*x(i)+Coef2(3)*x(i)^2;
    E_poly2=E_poly2+((y(i)-pxi)^2);
end
display(E_poly2);

E_poly3=0;
for i=1:1:m
    pxi=Coef(1)+Coef(2)*x(i)+Coef(3)*x(i)^2+Coef(4)*x(i)^3;
    E_poly3=E_poly3+((y(i)-pxi)^2);
end
display(E_poly3);
```