Study on Methodologies for Estimating Unknown Functions

Author: Tudor-Cristian Sîngerean

1. Project description

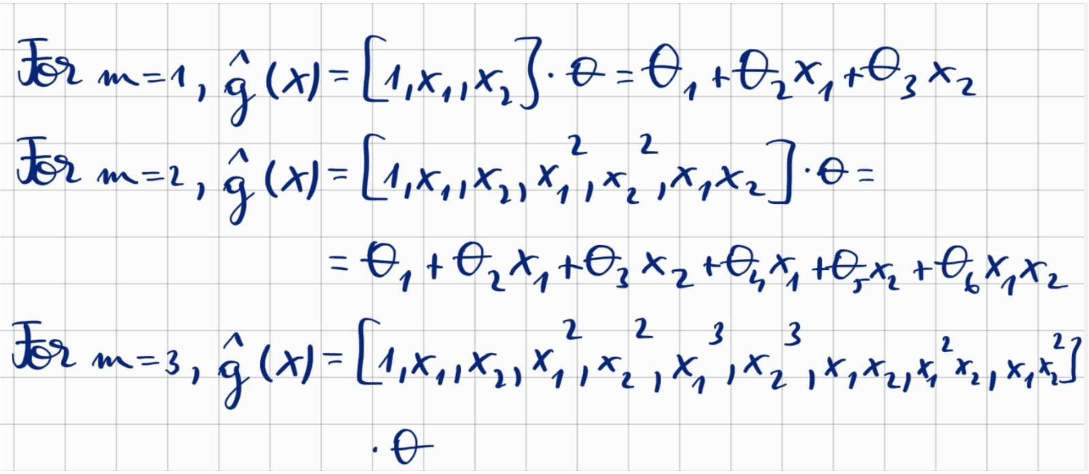
The objective of this project is to perform polynomial regression on a two-variable dataset, aiming to model the relationship between the input variables and the output variable.

The code utilizes MATLAB to implement polynomial regression, exploring different polynomial degrees and evaluate the model's performance on identification and validation datasets.

Imagine we have a dataset containing information about the temperature (X1) and humidity (X2) in a particular region and their corresponding effect on crop yield (Y). The goal of this project could be to develop a predictive model using polynomial regression to understand the non-linear relationship between temperature, humidity, and crop yield.

2. Implementation

For developing a model for a specific function, using a polynomial approximator is the optimal approach. The model is based on a specific function called ‘g_line’ which will be used in order to create **ĝ**.



For $m=1$, $\hat{g}(x)=[1,x_1,x_2]\cdot\theta=\theta_1+\theta_2x_1+\theta_3x_2$

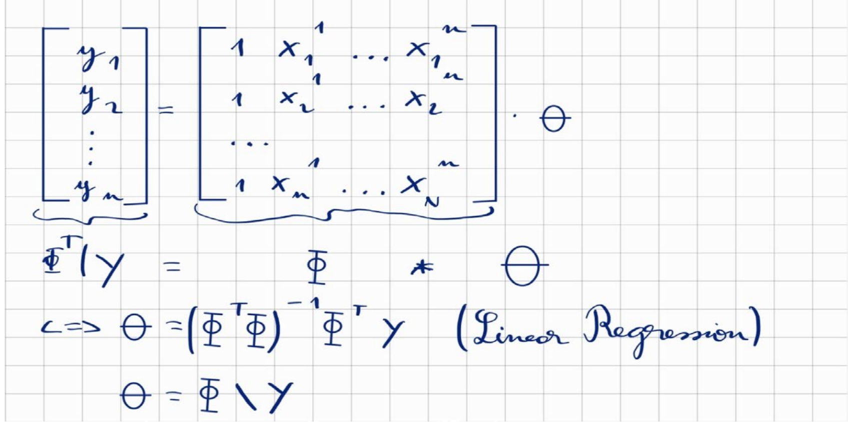
For $m=2$, $\hat{g}(x)=[1,x_1,x_2,x_1^2,x_2^2,x_1x_2]\cdot\theta=$

$=\theta_1+\theta_2x_1+\theta_3x_2+\theta_4x_1+\theta_5x_2+\theta_6x_1x_2$

For $m=3$, $\hat{g}(x)=[1,x_1,x_2,x_1^2,x_2^2,x_1^3,x_2^3,x_1x_2,x_1^2x_2,x_1x_2^2]$

$\cdot\theta$

The computation of the model's coefficients (θ) relies on an essential vector, which is obtained by dividing the ϕ and Y matrices θ = ϕ / Y



$$\underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}}_{} = \underbrace{\begin{bmatrix} 1 & x_1^1 & \dots & x_1^n \\ 1 & x_2^1 & \dots & x_2^n \\ \dots & & & \\ 1 & x_n^1 & \dots & x_N^n \end{bmatrix}}_{} \cdot \Theta$$

$$\Phi^T \, | \, Y = \Phi * \Theta$$

$$\Leftrightarrow \Theta = (\Phi^T \Phi)^{-1} \Phi^T Y \quad \text{(Linear Regression)}$$

$$\Theta = \Phi \setminus Y$$

By increasing or decreasing the degree m, the dimensions of the matrix ϕ changes.

Upon achieving the coefficients θ , the generation of the predictive model for both the identification and validation datasets results through the matrix multiplication of ϕ and θ.

3. Tuning parameters

The value of m varies and one can observe that by taking certain values of m , the validation data differs significantly.

By increasing the m , the model will be more fitted to the identification data and will cause the ‘overfitting’ phenomenon.

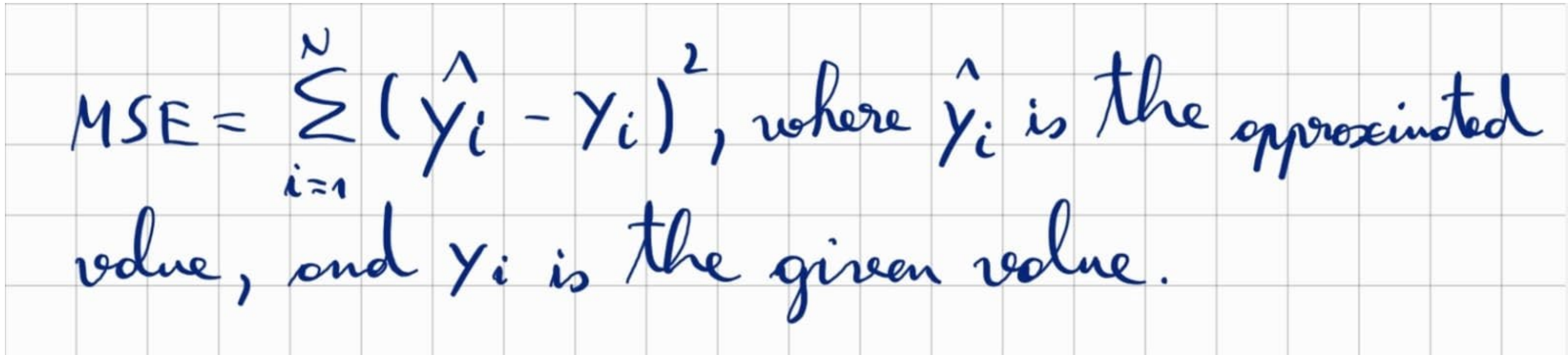
By decreasing the m, the model will be less fitted to the identification data and will cause the ‘underfitting’ phenomenon.

It can be observed that, by increasing m drastically, not only that the model will be extremely overfitted but the performance diminishes also.

4. Plots

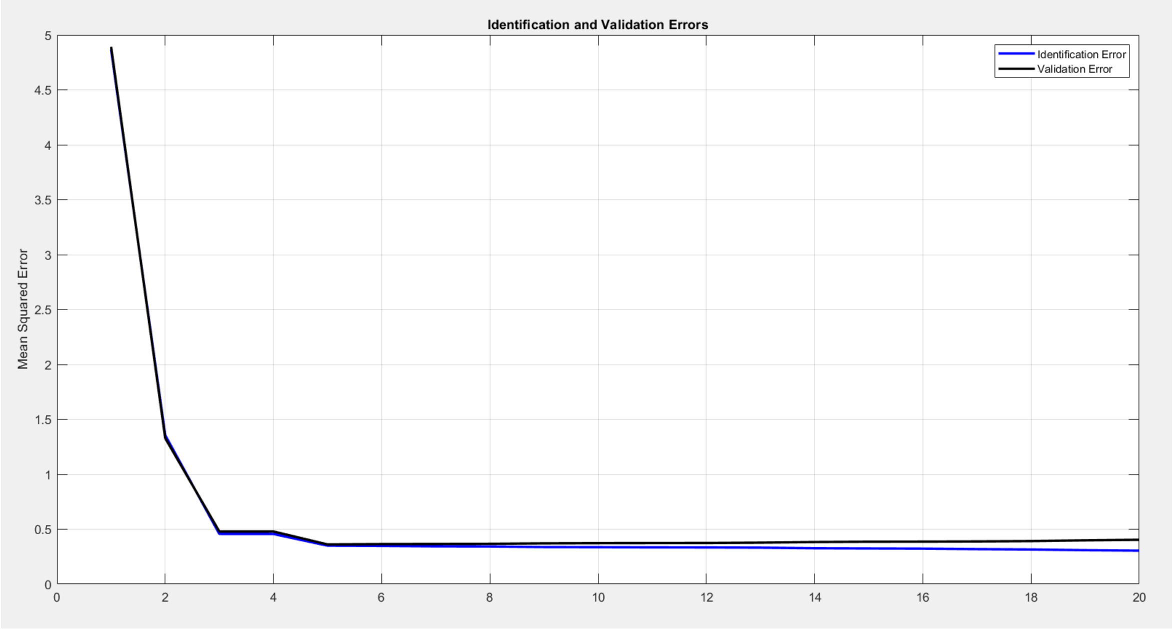
To provide the precision of our model on the validation data, we calculate the Mean Squared Error (MSE) for each value of m. This way we can measure how close our model's predictions are to the actual values from the validation set.

By calculating the MSE for different values of m with the below formula, the understanding of how well our model is doing under certain conditions is easier.



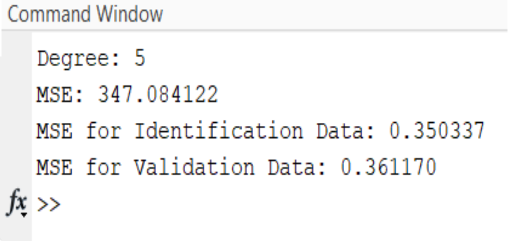
$MSE = \sum_{i=1}^{N} (\hat{y}_i - y_i)^2$, where $\hat{y}_i$ is the approximated value, and y_i is the given value.

The most straightforward way to visualize the accuracy of the model concerning the m value is by creating a plot using Matlab.

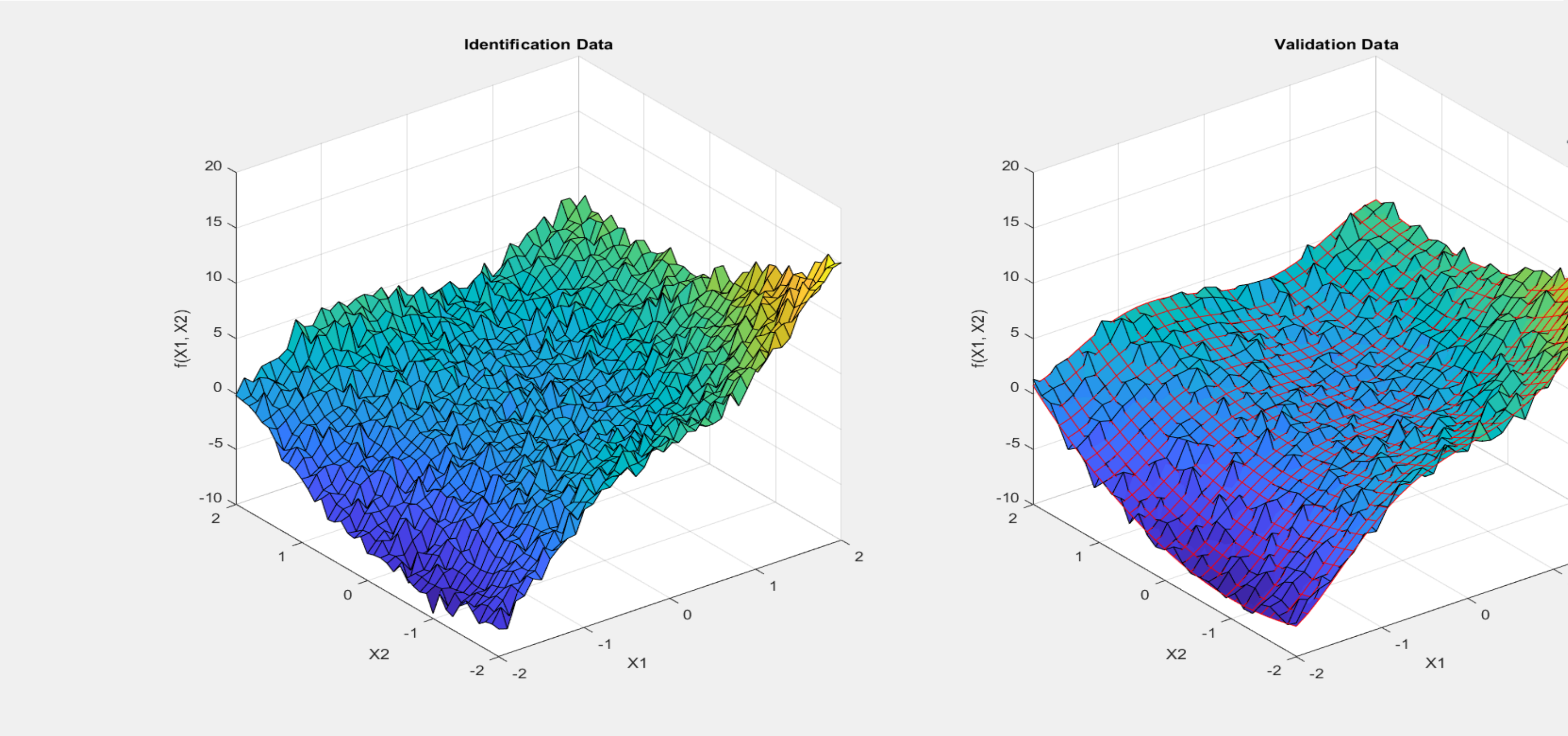


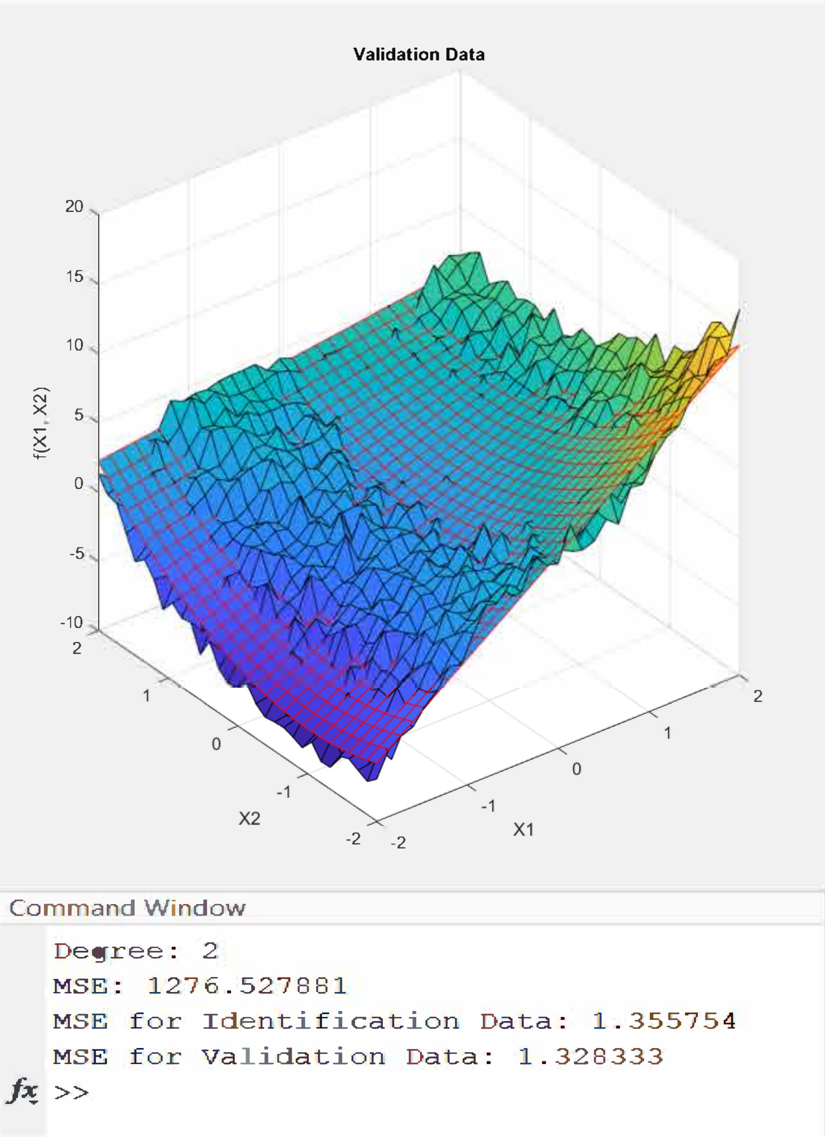
It can be observed that the identification error slightly differs from the validation error.

In our provided dataset, the optimal model for the validation data corresponds to $m = 5$. Additionally, it's noticeable that as m increases, the Mean Squared Error (MSE) for the training data decreases, indicating a better fit. However, for the validation data, the MSE tends to increase for larger m values, suggesting the occurrence of overfitting.

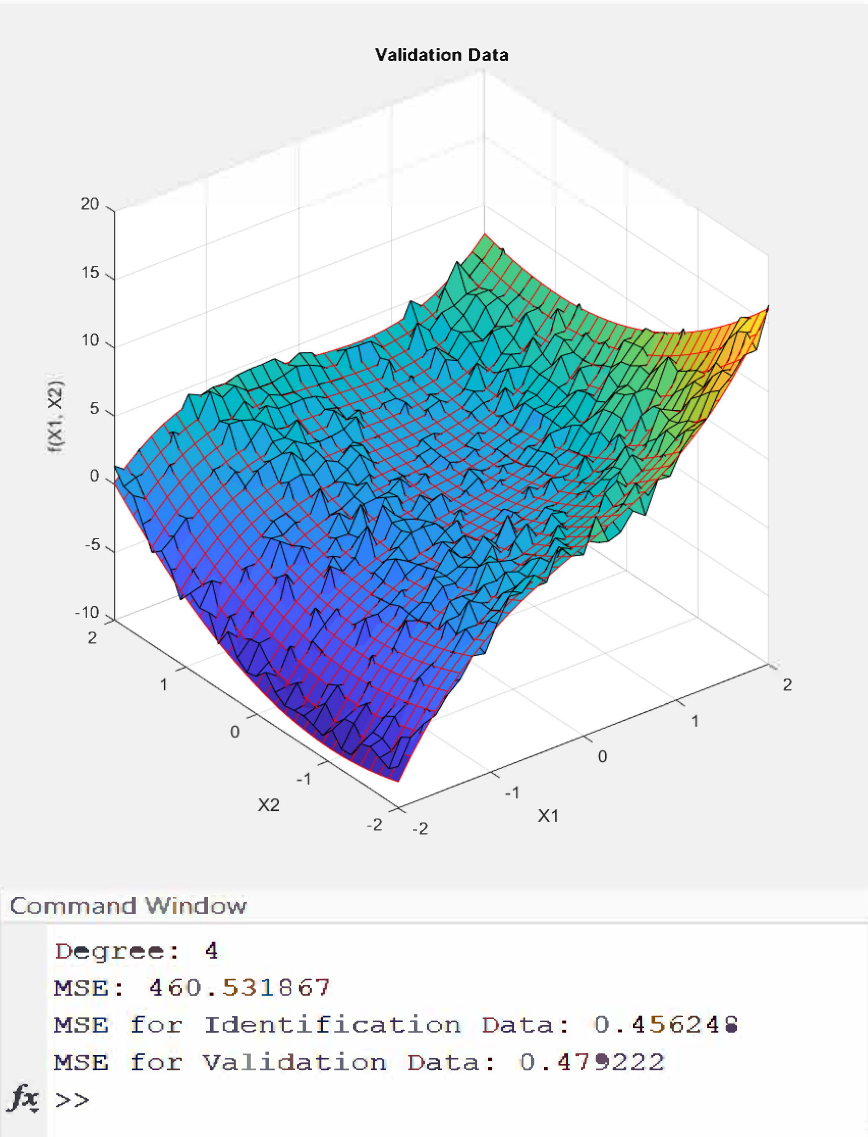


```
Degree: 5
MSE: 347.084122
MSE for Identification Data: 0.350337
MSE for Validation Data: 0.361170
fx >>
```

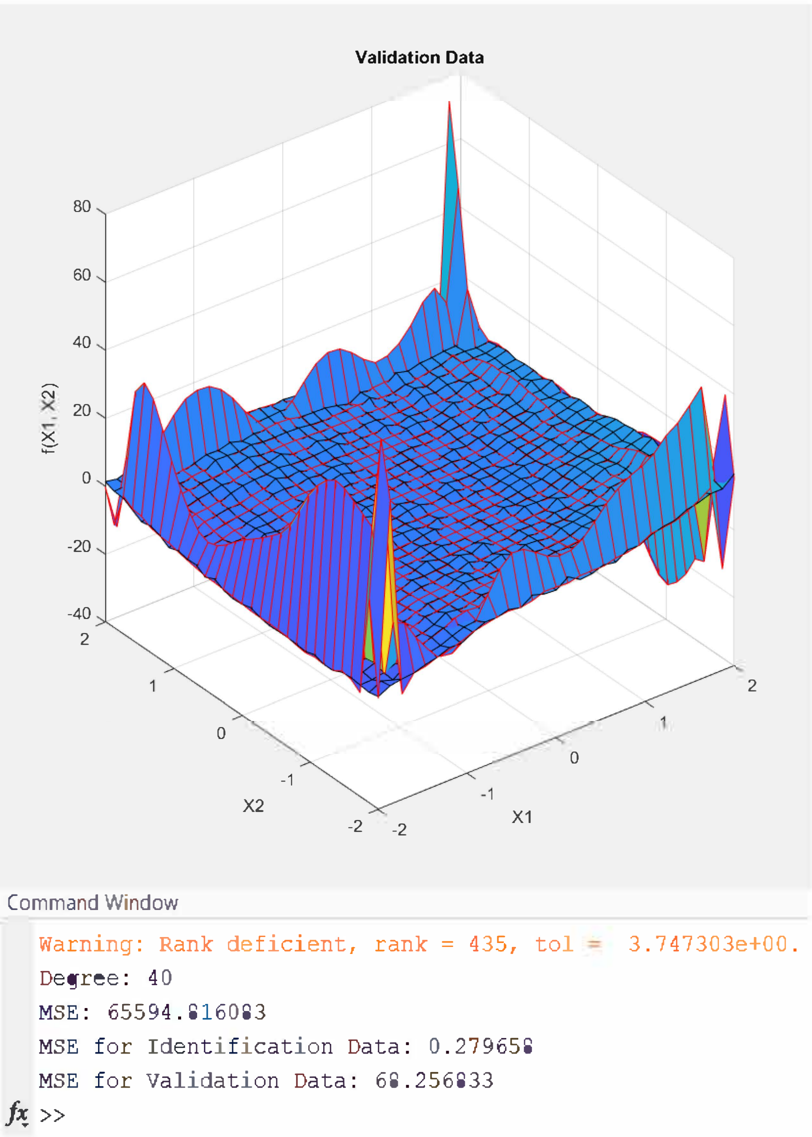




```
Degree: 2
MSE: 1276.527881
MSE for Identification Data: 1.355754
MSE for Validation Data: 1.328333
fx >>
```



```
Degree: 4
MSE: 460.531867
MSE for Identification Data: 0.456248
MSE for Validation Data: 0.479222
fx >>
```



```
Warning: Rank deficient, rank = 435, tol =  3.747303e+00.
Degree: 40
MSE: 65594.816083
MSE for Identification Data: 0.279658
MSE for Validation Data: 68.256833
fx >>
```

5. Conclusions

- Depending on the given m , our predicted model can suffer several changes due to its malleable nature. Both overfitted and underfitted cases are based on our m .

- Our $\hat{\mathbf{g}}$ matrix is computed by two input datasets that are approximated by a polynomial function in order to result a predictive model on a given situation.

- Real life problems such as predicting the spread of a virus or other infectious diseases, evaluating trends and forecasts.

6. **Appendix**

Reading data and storing dimensions in variables

```
clear all;
clc;
% Load the identification and validation data
load('proj_fit_27.mat'); % Loading the project data


% Generate the design matrix for identification data
n1_id = size(id.X{1}, 2); % Number of grid points
n2_id = size(id.X{2}, 2);
n1_val = size(val.X{1}, 2); % Number of grid points
n2_val = size(val.X{2}, 2);
power=0;
newel=1;
% Initialize variables to store MSE for identification and validation
mse_id_array = [];
mse_val_array = [];
```

The ‘**g_line**’ function

```
function gline = g_line(gline, vect, power, m, newel, indice)


newel = newel * vect(indice);% We create the new element which we are gonna put in gline
power = power + 1; % We increase the power of the polynom
if power == m % If our grade is equal with the predefined power m
gline = [gline ,newel]; % We put in gline the newel variable
else
for i = indice:length(vect)
% We take the variable 'indice' to not create duplicates
% If the power is not the one we want,we call it recursive
% In order to create the correct newel
gline = g_line(gline, vect, power, m, newel, i);
end
end
end
```

The implementation of the idea of polynomial approximation for identification and validation data, for any degree m of the polynomial, using the ‘**g_line**’ function.

```
for m = 1:10 % We take m as the power for polynomial
ghat = []; % Create the matrix for the combination of inputs
for i = 1:n1_id
for j = 1:n2_id
x1 = id.X{1}(i); % Identification input matrix rows
x2 = id.X{2}(j);
gline = []; % Here we stock the data for a single line of the matrix
vect = [1 x1 x2]; % Here are the elements that go to create gline
for indice = 1:length(vect)
gline = g_line(gline, vect, power, m, newel, indice);
end
ghat = [ghat; gline]; % Here is the creation of the matrix
end
end
y_id = id.Y(:);
theta = ghat \ y_id;
yhat_id = ghat * theta;
MSE = 0;
for i = 1:length(y_id)
MSE = MSE + ((yhat_id(i) - y_id(i))^2);
end
mse_id = (1/length(y_id) * MSE);
mse_id_array(m) = mse_id;
% From here validation begins
ghat = [];
for i = 1:n1_val
for j = 1:n2_val
x1 = val.X{1}(i); % Identification input matrix rows
x2 = val.X{2}(j);
gline = []; % Here we stock the data for a single line of the matrix
vect = [1 x1 x2]; % Here are the elements that go to create gline
for indice = 1:length(vect)
gline = g_line(gline, vect, power, m, newel, indice);
end
ghat = [ghat; gline]; % Here is the creation of the matrix
end
end


y_val = val.Y(:);
yhat_val = ghat * theta;
MSE = 0;
for i = 1:length(y_val)
MSE = MSE + ((yhat_val(i) - y_val(i))^2);
end
mse_val = (1/length(y_val) * MSE);
mse_val_array(m) = mse_val;
end
```

Determining the value of m (degree) for minimum Mean Squared Error (MSE) and plotting MSE for identification and validation datasets:

```
% Find the minimum MSE and corresponding m
[min_mse_val, min_m] = min(mse_val_array);


% Plotting the identification error and validation error
plot(mse_id_array, 'b', 'LineWidth', 2); grid; shg;
hold on
plot(mse_val_array, 'k', 'LineWidth', 2);


% Adding legend
legend('Identification Error', 'Validation Error');


% Adding titles and labels if needed
title('Identification and Validation Errors');
ylabel('Mean Squared Error');


% Display the results
fprintf('Minimum MSE for Validation Data: %f (m = %d)\n',
min_mse_val, min_m);


% Display the MSE for both identification and validation data
fprintf('MSE for Identification Data: %f\n', mse_id);
fprintf('MSE for Validation Data: %f\n', mse_val);
```

The implementation of the polynomial approximation function for the minimum degree *m*.

```
% Initialize variables to store MSE for identification and
validation
mse_id_array = [];
mse_val_array = [];
m=5;
ghat = []; % Create the matrix for the combination of inputs
for i = 1:n1_id
for j = 1:n2_id
x1 = id.X{1}(i); % Identification input matrix rows
x2 = id.X{2}(j);
gline = []; % Here we stock the data for a single line of the matrix
vect = [1 x1 x2]; % Here are the elements that go to create gline
for indice = 1:length(vect)
gline = g_line(gline, vect, power, m, newel, indice);
end
ghat = [ghat; gline]; % Here is the creation of the matrix
end
end
y_id = id.Y(:);
theta = ghat \ y_id;
yhat_id = ghat * theta;
MSE = 0;
for i = 1:length(y_id)
MSE = MSE + ((yhat_id(i) - y_id(i))^2);
end
mse_id = (1/length(y_id) * MSE);

% From here validation begins
ghat = [];
for i = 1:n1_val
for j = 1:n2_val
x1 = val.X{1}(i); % Identification input matrix rows
x2 = val.X{2}(j);
gline = []; % Here we stock the data for a single line of the matrix
vect = [1 x1 x2]; % Here are the elements that go to create gline
for indice = 1:length(vect)
gline = g_line(gline, vect, power, m, newel, indice);
end
ghat = [ghat; gline]; % Here is the creation of the matrix
end
end


y_val = val.Y(:);
yhat_val = ghat * theta;
MSE = 0;
for i = 1:length(y_val)
MSE = MSE + ((yhat_val(i) - y_val(i))^2);
end
mse_val = (1/length(y_val) * MSE);
```

Defining a grid of points and plotting the validation data together with the 3D representation of the polynomial approximator for validation data.

```
% Define a grid of points for 3D visualization
[x1_grid, x2_grid] = meshgrid(id.X{1}, id.X{2});
x1_flat = x1_grid(:);
x2_flat = x2_grid(:);


% Plot the identification data and the polynomial approximator in 3D
figure;
subplot(121);
surf(x1_grid, x2_grid, id.Y);
title('Identification Data');
xlabel('X1');
ylabel('X2');
zlabel('f(X1, X2)');


% Define a grid of points for 3D visualization on validation data
[x1_val_grid, x2_val_grid] = meshgrid(val.X{1}, val.X{2});
x1_val_flat = x1_val_grid(:);
x2_val_flat = x2_val_grid(:);


% Plot the validation data and the polynomial approximator in 3D for validation data
subplot(122);
surf(x1_val_grid, x2_val_grid, reshape(yhat_val, size(x1_val_grid)), 'EdgeColor','r');
hold on;
surf(x1_val_grid, x2_val_grid, reshape(y_val, size(x1_val_grid)), 'EdgeColor','k');
title('Validation Data');
xlabel('X1');
ylabel('X2');
zlabel('f(X1, X2)');


fprintf('Degree: %d\n', m);
fprintf('MSE: %f\n', MSE);
fprintf('MSE for Identification Data: %f\n', mse_id);
fprintf('MSE for Validation Data: %f\n', mse_val);
```