System Identification Project

2020-2021

Part 1

Students:

Nicolae Cristian – Mihai

Roșa Andreea

Socaci Marius – Cristian

Group 18/18

Contents

Introduction

Given a model shaped by the static function $f(x_1, x_2) = y$, the goal is to generate a polynomial approximator g of the function f using two provided data sets of inputs-outputs. The first data set is used to build the approximator, and the second one for the validation of the computed polynomial approximator. Both data sets are generated using the same function f.

Theoretical Approach

In this part, the key component in modelling such an approximator is the linear regression. For example:

$$\begin{gathered}
y_1 = \theta_1 \cdot x_1^0 + \theta_2 \cdot x_1^1 \\
y_2 = \theta_1 \cdot x_2^0 + \theta_2 \cdot x_2^1 \\
\vdots \\
y_n = \theta_1 \cdot x_n^0 + \theta_2 \cdot x_n^1
\end{gathered}$$

This can be rewritten in matrix form as:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} x_1^0 & x_1^1 \\ x_2^0 & x_2^1 \\ \vdots & \vdots \\ x_n^0 & x_n^1 \end{bmatrix} \cdot \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}$$

Or, in a more general case, where n stands for the number of data entries and m for the degree of the equation:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} x_1^0 & x_1^1 & \cdots & x_1^m \\ x_2^0 & x_2^1 & \cdots & x_2^m \\ \vdots & \vdots & \ddots & \vdots \\ x_n^0 & x_n^1 & \cdots & x_n^m \end{bmatrix} \cdot \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_{m+1} \end{bmatrix},$$

where x_i represents input data points and y_i represents output data points, $i = \overline{1{:}n}$. Y of size $\mathrm{n} \times 1$, matrix Φ of size $\mathrm{n} \times \mathrm{m}$ and the column vector θ is of size depending on the number of elements of each line in the regressor matrix Φ, $(\mathrm{m} + 1) \times 1$ in this case. The matrices can be rewritten as it follows:

$$Y = \Phi \cdot \theta$$

In matrix form Y stands for the regressed variable matrix on which the outcome is predicted, Φ represents the regressor matrix and θ the unknown parameters vector.

The polynomial approximator of a function f, which has the degree m, has the following form:

For $m = 1$:

$$g = [1, x_1, x_2] \cdot \theta = \theta_1 + \theta_2 \cdot x_1 + \theta_3 \cdot x_2$$

For $m = 2$:

$$g = [1, x_1, x_2, x_1^2, x_1\, x_2, x_2^2] \cdot \theta = \theta_1 + \theta_2 \cdot x_1 + \theta_3 \cdot x_2 + \theta_4 \cdot x_1^2 + \theta_5 \cdot x_1\, x_2 + \theta_6 \cdot x_2^2,$$

where g is the polynomial approximator, $[1, x_1, x_2, x_1^2, x_1\, x_2, x_2^2]$ is the regressor vector Φ and θ stands for the unknown parameters vector.

Implementation

The given data represents two sets of input vectors and one corresponding matrix output:

$$X_1 = [x_{11} \quad x_{12} \quad \cdots \quad x_{1n}]$$

$$X_2 = [x_{21} \quad x_{22} \quad \cdots \quad x_{2m}]$$

$$Y = \begin{bmatrix} y(x_{11}, x_{21}) & y(x_{11}, x_{22}) & \cdots & y(x_{11}, x_{2m}) \\ y(x_{12}, x_{21}) & y(x_{12}, x_{22}) & \cdots & y(x_{12}, x_{2m}) \\ \vdots & \vdots & \ddots & \vdots \\ y(x_{1n}, x_{21}) & y(x_{1n}, x_{22}) & \cdots & y(x_{1n}, x_{2m}) \end{bmatrix} \xrightarrow{yields} Y = \begin{bmatrix} y_{11} & y_{12} & \cdots & y_{1m} \\ y_{21} & y_{22} & \cdots & y_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ y_{n1} & y_{n2} & \cdots & y_{nm} \end{bmatrix}$$

It can be observed that each combination between the two inputs corresponds to a single output value. If x_1 has the size $1 \times n$ and x_2 has the size $1 \times m$, then y matrix has the size $n \times m$. Because the given data is in a bidimensional format, the algorithm will not work correctly, so it is necessary to reshape the data into a one-dimensional format, process that is named "flatten" into the attached solution. Hence, the matrices are of the following form:

$$X = \begin{bmatrix} x_{11} & x_{21} \\ x_{11} & x_{22} \\ \vdots & \vdots \\ x_{11} & x_{2m} \\ x_{12} & x_{21} \\ \vdots & \vdots \\ x_{12} & x_{2m} \\ \vdots & \vdots \\ x_{1n} & x_{2m} \end{bmatrix} \qquad Y = \begin{bmatrix} y_{11} \\ \vdots \\ y_{1m} \\ y_{21} \\ \vdots \\ y_{2m} \\ \vdots \\ y_{n1} \\ \vdots \\ y_{nm} \end{bmatrix}$$

Now that the matrices' sizes satisfy the desired conditions, the next step is to compute the regressor matrix, Φ. It is created recursively using the powers of x_1 and x_2 depending on the degree m.

Looking at the form of the regressor one can observe that any degree contains all the elements of the previous degrees:

$$\Phi_{\mathrm{m}} = [\Phi_{\mathrm{m}-1}, \Phi_{\mathrm{m}}^{*}],$$

where Φ_{m} is the regressor matrix of degree m and Φ_{m}^{*} represents the new elements of Φ_{m}, which are computed using the formula $x_1^{m-i} \cdot x_2^{i},\ i = \overline{0{:}\,m}$.

In order to obtain the approximator g it is essential to find the unknown parameters vector starting from the formula of the static function $f(x_1, x_2) = y$, where y represent the regressed variable and has the following form:

$$\mathrm{Y} = \Phi \cdot \theta$$

To find the unknown parameters vector θ, one would multiply the equation with Φ^{-1}. In fact, this is not possible because the regressor vector is not a square matrix, therefore we cannot compute its inverse. The getaway trick in this case is to multiply the equation with Φ^{T} which stands for the transpose of the matrix Φ.

$$\Phi^{T} \cdot \mathrm{Y} = \Phi^{T} \cdot \Phi \cdot \theta$$

Since the matrix product $\Phi^{T}\,\Phi$ yields a square matrix which supposedly is non-singular, it is legal to apply its inverse to the previous equation:

$$(\Phi^{T} \cdot \Phi)^{-1} \cdot \Phi^{T} \cdot \mathrm{Y} = (\Phi^{T} \cdot \Phi)^{-1} \cdot \Phi^{T} \cdot \Phi \cdot \theta$$

Thus, the matrix θ is obtained:

$$\theta = (\Phi^{T} \cdot \Phi)^{-1} \cdot \Phi^{T} \cdot \mathrm{Y}$$

A similar algorithm is performed when executing the following code in MATLAB, where \ operator stands for left matrix division:

$$\theta = \Phi \backslash \mathrm{Y}$$

The polynomial approximator of the given degree m is then calculated by multiplying the regressor vector and the now known parameters vector:

$$\mathrm{g} = \Phi_{\mathrm{m}} \cdot \theta$$

Tuning results

To ensure that the calculations are good, a check of the mean squared error on the identification data is performed:

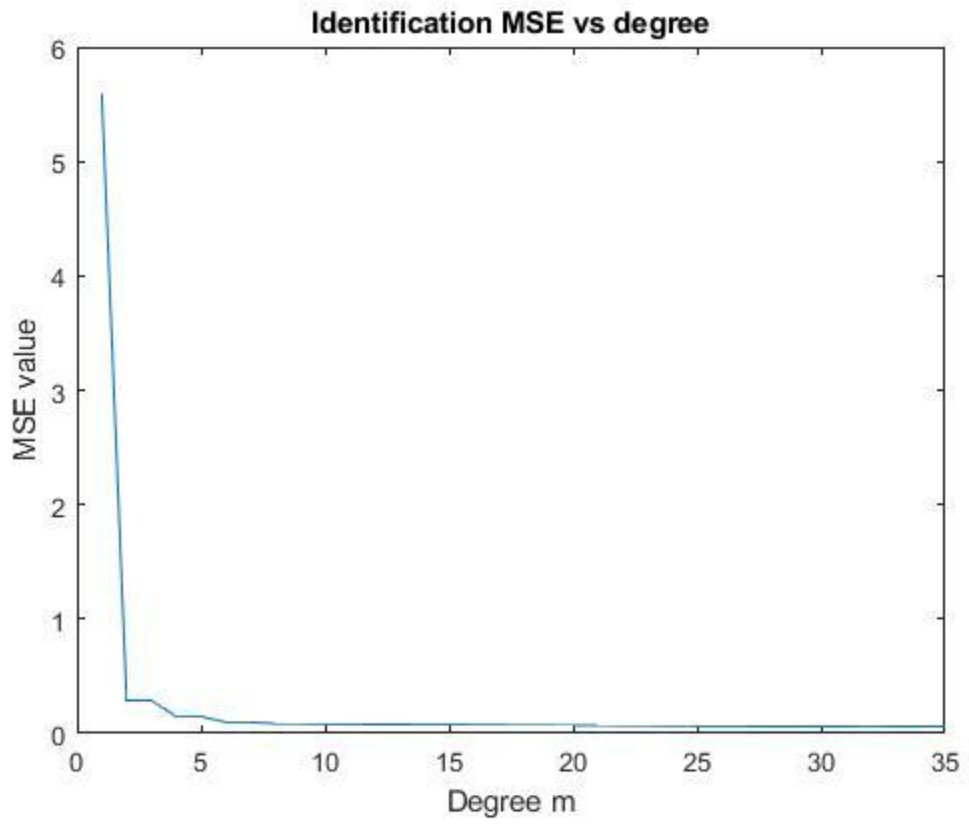

Figure 1 – MSE computed against the identification data

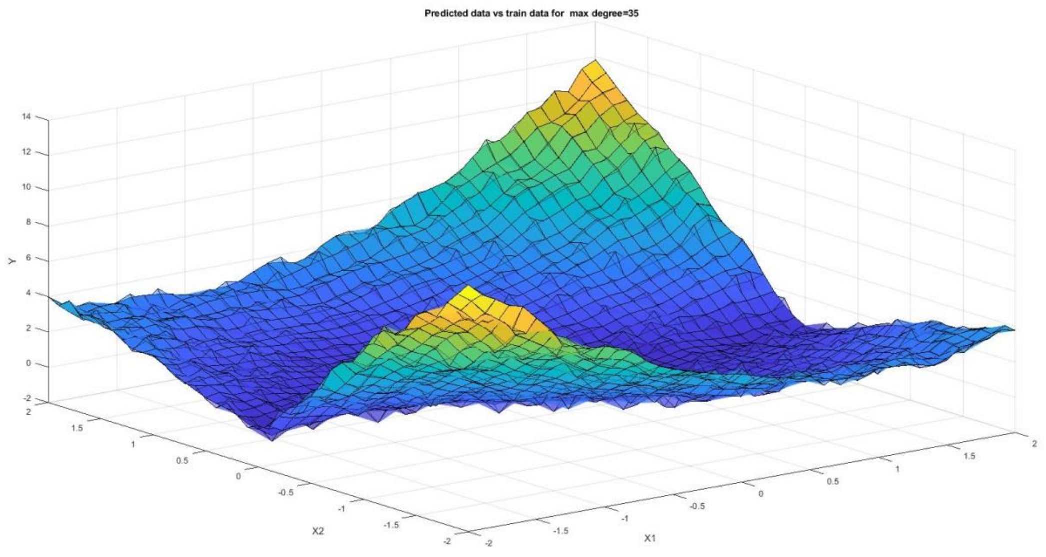

Figure 2 – Predicted surface vs train surface

Exactly as expected, MSE value approaches 0 as the degree m gets higher, which means the algorithm applied is correct (Figure 1). In Figure 2 is shown a representative plot for the fit on the training data, which is computed using the approximator with the highest degree in the used range of testing, $m \in (1, 35)$. In order to find the closest polynomial approximator to the static function, which is used to compute the real outputs, it is necessary to compute the mean squared error on the validation data against the predicted data for every degree. The best fit will be given by the degree for which the smallest value of MSE is returned. Figure 3 shows that for the degree $m = 10$ results the smallest value of the mean squared error, $MSE = 0.0875$; therefore, the best fit of the static function is given for the degree $m = 10$ of the polynomial approximator g. It is pointless to try higher degrees than $m = 35$, which was used in the code, since that would yield an overfit of the static function, because the real outputs are corrupted by noise. Therefore, trying to make the approximator too close to the corrupted data leads to higher MSE errors.

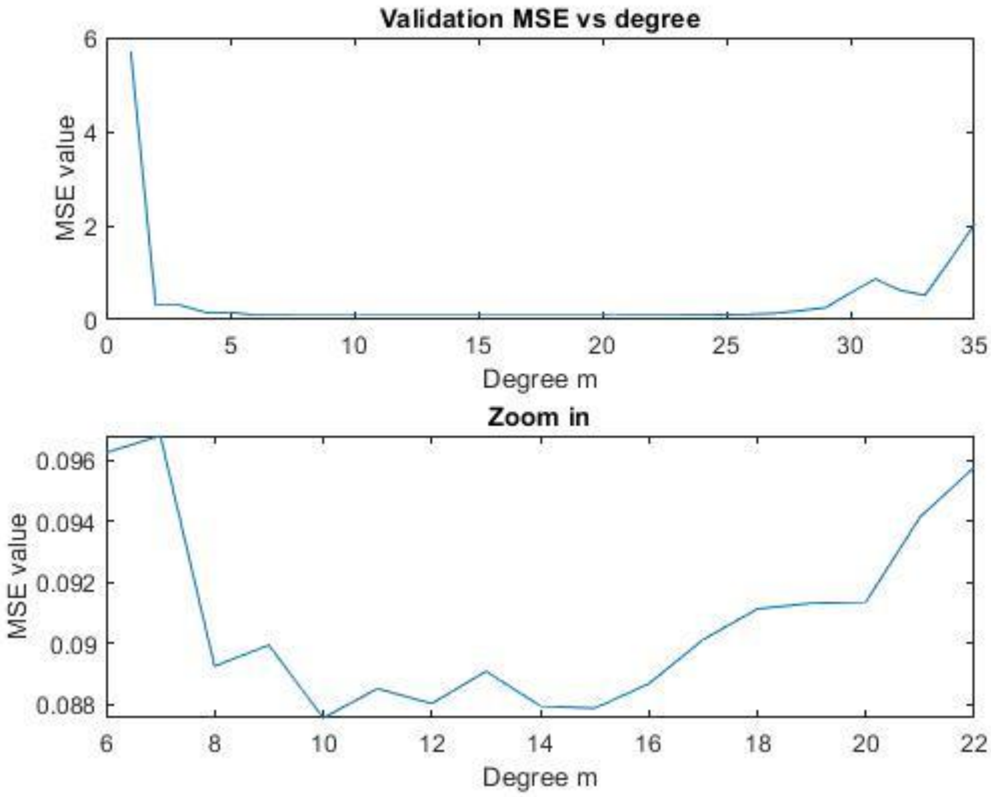

Figure 3 – MSE computed against the validation data

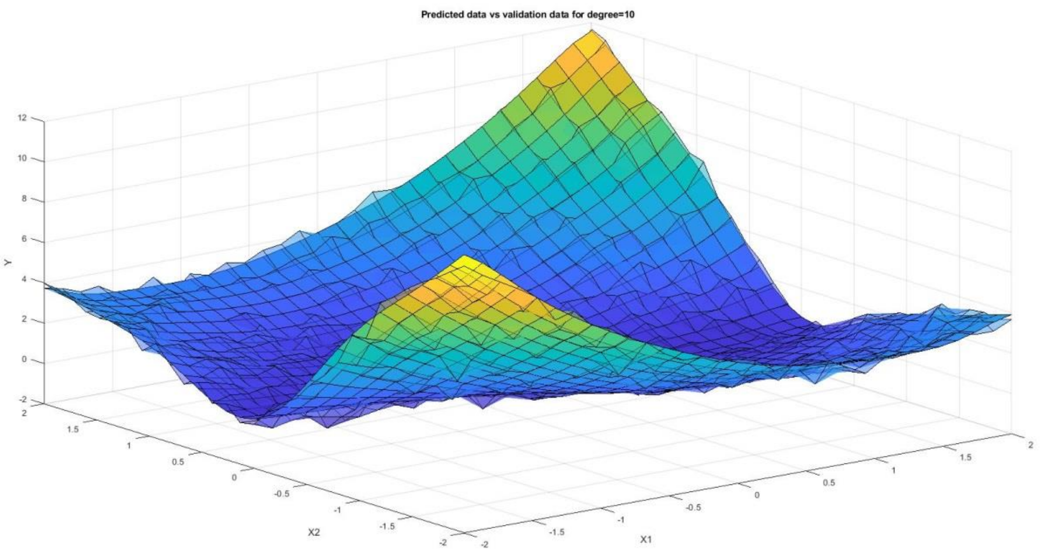

Fig. 4 – Predicted data versus validation data

Of course, after finding the degree for which MSE value is minimum, one would like to see how good the fit is compared to the real data points. In Figure 4 is represented the surface constructed using the points resulted from the polynomial approximator of best degree found, $m = 10$. It is visible that when comparing the approximated points to the validation data points the surface is not as precise that it is when comparing to the train data points, as shown in Figure 3.

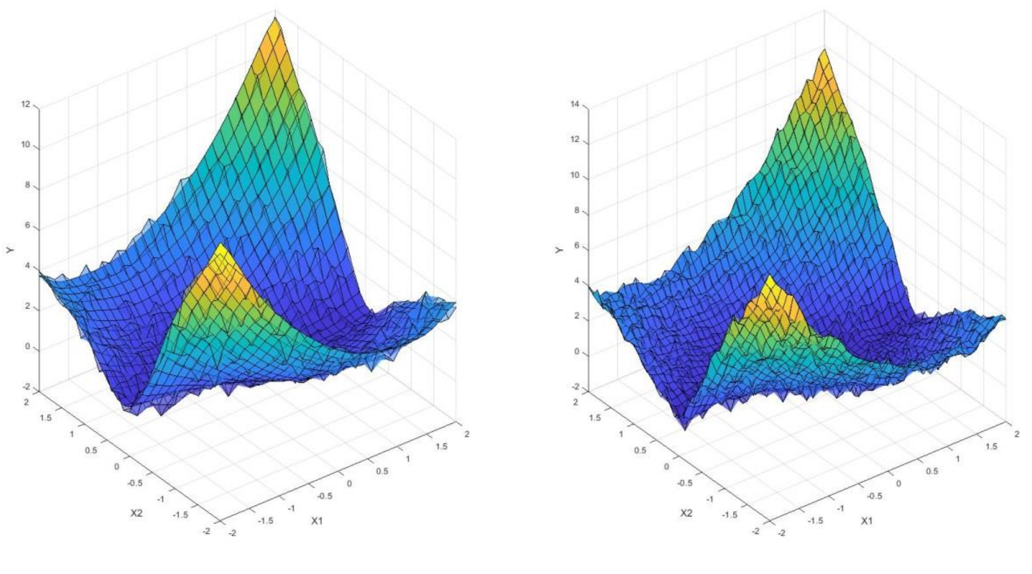

Fig. 5 – The two plots put together

Figure 5 makes the eye comparison easier between the two plots. The left-hand side stands for the validation data versus the predicted data, and the right-hand side stands for the train data vs the predicted data. It is perfectly normal that the predicted data is closer to the identification data, mainly because that was the data set which is used to obtain the polynomial approximator which computes the predicted data. Another reason for this effect to happen would be that the real data is corrupted by noise.

Conclusion

Linear regression is used in order to obtain an approximation as accurate as possible of the static function depending on the growth of polynomial degree. However, it is recommended to avoid the excessive increase of degree, due to overfitting which appears as a result of the fact that the output of the system is corrupted by noise.

The most efficient method toward obtaining the best approximation of the static function is computing the mean squared error for each polynomial degree from 1 to 100 and verifying which one is the lowest considering that this one provides the best fit on the validation data.

Appendx-MATLAB code

Main program

```
load('proj_fit_18.mat');

% Flatten the data
Xflat = flatten(id.X{1},id.X{2});
Yflat = reshape(id.Y, length(id.Y)^2, 1);
Xflatval = flatten(val.X{1}, val.X{2});
Yflatval = reshape(val.Y, length(val.Y)^2, 1);

maxdegree = 35;
M = 1:maxdegree;
mseid = zeros(1, maxdegree);
mseval = zeros(1, maxdegree);
for m = M
    % Identification
    [yhatid, mseid(m), theta] = trainandpredict(Xflat,Yflat,m);

    % Validation
    [yhatflat, mseval(m)] = predict(Xflatval,m,theta,Yflatval);
end
```

```
% MSE Plots
plot(1:maxdegree,    mseid),title("Identification    MSE    vs    degree");
xlabel("Degree m"); ylabel("MSE value");
figure
subplot(211),    plot(1:maxdegree,    mseval),title("Validation    MSE    vs
degree");xlabel("Degree m"); ylabel("MSE value");
subplot(212), plot(6:22, mseval(6:22)), title("Zoom in"); xlabel("Degree m");
ylabel("MSE value");

% Find the best mse and compute the yhat for it
msestar = min(mseval);
mstar = M(mseval==msestar);
thetastar = train(Xflat, Yflat, mstar);
[yhatstar, ] = predict(Xflatval,mstar,thetastar,Yflatval);
yhatstar = reshape(yhatstar, length(val.Y), length(val.Y));

thetastarid=train(Xflat, Yflat, 10);
[yhatstarid, ] = predict(Xflat,10,thetastarid,Yflat);
yhatstarid = reshape(yhatstarid, length(id.Y), length(id.Y));


% Plot the best yhat
figure
subplot(1, 2,1);
surf(val.X{1},val.X{2}, yhatstar);
hold on
surf(val.X{1}, val.X{2}, val.Y, 'FaceAlpha',0.5)
xlabel("X1"); ylabel("X2"); zlabel("Y")

subplot(1, 2,2);
surf(id.X{1},id.X{2}, yhatstarid);
hold on
surf(id.X{1}, id.X{2}, id.Y, 'FaceAlpha',0.5)
xlabel("X1"); ylabel("X2"); zlabel("Y")

figure
surf(val.X{1},val.X{2}, yhatstar), title("Predicted data vs validation data
for degree="+mstar)
hold on
surf(val.X{1}, val.X{2}, val.Y, 'FaceAlpha',0.5)
xlabel("X1"); ylabel("X2"); zlabel("Y")

figure
surf(id.X{1},id.X{2}, yhatstarid), title("Predicted data vs train data for
max degree="+35)
hold on
surf(id.X{1}, id.X{2}, id.Y, 'FaceAlpha',0.5)
xlabel("X1"); ylabel("X2"); zlabel("Y")
```

Flatten function

```
 function [xflat] = flatten(x1, x2)
%  Creates  a  matrix  that  contains  all  the  combinations  between  x1  and  x2
%values

x1repeat = repmat(x1,length(x2),1);
x1repeat = reshape(x1repeat,length(x1)*length(x2),1);
xflat = [x1repeat, repmat(x2', length(x1),1)];
end
```

Deg function

```
function [out] = deg(x1, x2, m)
% Returns the unique elements of a certain degree regressor;
% For example: a regressor of degree 0 contains [1],
% a regressor of degree 1 contains [1, x1, x2],
% a regressor of degree 2 contains [1, x1, x2, x1^2, x1x2, x2^2], and so
% on. We can see that the unique elements of a certain degree, m, are the
% basis elements of a 2D polynomial of degree m which can be calculated
% using the formula: x1^(m-i)*x2^i, i=0:m

out=zeros(length(x1),m+1);
for i = 0:m
    out(:,i+1) = x1.^(m-i) .* x2.^(i);
end

end
```

Regressor function

```
function [regr] = regressor(x1,x2,m)
% Recursive function to find the regressor
% uses the fact that the regressor of a certain degree contains all the
% elements of the lower degree regressor and some unique elements for that
% degree which are calculated and explained in the deg function

if m == 0
    regr = deg(x1,x2,0);
else
    regr = [regressor(x1,x2,m-1) deg(x1,x2,m)];
end
en
```

Train function

```
function [theta] = train(x,y, m)
% Finds the coefficients theta for a model of degree m

    regr = regressor(x(:,1),x(:,2),m); % Find the regressor
    theta = regr\y; % Use linear regression to find the coefficients
end
```

Traindandpredict function

```
function [yhat,mse, theta] = trainandpredict(x, y, m)
% Used for efficency when training and also calculating a prediction on the
% same dataset because the regressor is not calculated twice

    regr = regressor(x(:,1),x(:,2),m); % Find the regressor
    theta = regr\y; % Use linear regression to find the coefficients
    yhat = regr*theta; % Predict the output of the identification data using
the computed coefficients
    mse = immse(yhat, y); % Compute the mse
end
```

Predict function

```
function [yhat,mse] = predict(x,m,theta,y)
% Computes a predicted output, yhat, using the coefficients, theta, of a
% model of degree m calculated by the train function and also finds the mse

regr = regressor(x(:,1),x(:,2),m); % Compute the regressor
yhat = regr*theta; % Predict the output of the identification data using the
computed coefficients
mse = immse(yhat, y); % Compute the mse

end
```