System Identification Project
Part 1: Fitting an unknown function

Data index:
22/10

Students: Matei-Alexandru Blaj

Joelle-Mirela Danciu

Mihai-Antonio Dîcă

Group 30333

Guiding professors: Zoltan Nagy, Lucian Buşoniu

Contents

1. Introduction

We are given two data sets (one for identification and one for validation), representing the input-output values of an unknown function with two input variables (x_1, x_2) and one output (y). Our goal is to find a model for the unknown function in order to predict its behavior using a polynomial approximator with variable degree, and optimize it using the least-squares method. MATLAB software was used to complete the assignment.

2. Linear Regression

2.1. Objective

The objective of linear regression is, as its name suggests, to find a model for an unknown system by going to a state previous to the input and output data, which would obviously be a function describing those data sets.

From a plot perspective, linear regression is a tool that helps us find a line or a curve that best fits the output data points (affected by noise).

2.2. Polynomial Approximators

Because a polynomial can take any shape by increasing its degree and adjusting its parameters, we can fit our function with a polynomial using this property. It is known that a higher degree of a polynomial yields a larger number of inflection points, thus allowing for a better fit for our identification data. However, by increasing the degree of the polynomial approximator, we run the risk of fitting other sets poorly (more on that later).

In our case, we have a system with two inputs (x_1, x_2) and one output (y). This means that our polynomial approximator will have the form:

$$y = \boldsymbol{\theta_1} + \sum_{\boldsymbol{i=1}}^{\boldsymbol{m}} \boldsymbol{\theta_{1+i}} \cdot \boldsymbol{x_1}^{\boldsymbol{i}} + \sum_{\boldsymbol{i=1}}^{\boldsymbol{m}} \boldsymbol{\theta_{m+1+i}} \cdot \boldsymbol{x_2}^{\boldsymbol{i}} + \sum_{\boldsymbol{i=1}}^{\boldsymbol{m-1}} \sum_{\boldsymbol{j=1}}^{\boldsymbol{m-i}} \boldsymbol{\theta_{2m+1+\frac{(2m-i)\cdot(i-1)}{2}+j}} \cdot \boldsymbol{x_1}^{\boldsymbol{i}} \cdot \boldsymbol{x_2}^{\boldsymbol{j}} \qquad (2.1)$$

where:

$$m - the\ degree\ of\ the\ polynomial$$

$$\theta_1, \theta_2, \theta_3, \dots, \theta_{\frac{(m+1)(m+2)}{2}} - the\ unknown\ parameters$$

We devised formula (2.1) in order to obtain a sum of all the possible products between the exponentiated x_1 and x_2, while keeping in mind that the sum of their exponents must be lower or equal to m in order to preserve the chosen degree of the polynomial. The index for the last θ in (2.1) was given by iterating over each element of a triangular matrix (the lines are the powers of x_1 and the columns are the powers of x_2).

Because our data sets are discrete, we can write all the tuples ($x_1(k), x_2(k), y(k)$) in matrix form:

$$\begin{bmatrix} \boldsymbol{y(1)} \\ \boldsymbol{y(2)} \\ \vdots \\ \vdots \\ \boldsymbol{y(k)} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\varphi_1}(\boldsymbol{x_1(1), x_2(1)}) & \boldsymbol{\varphi_2}(\boldsymbol{x_1(1), x_2(1)}) \dots & \boldsymbol{\varphi_{\frac{(m+1)(m+2)}{2}}}(\boldsymbol{x_1(1), x_2(1)}) \\ \boldsymbol{\varphi_1}(\boldsymbol{x_1(2), x_2(2)}) & \boldsymbol{\varphi_2}(\boldsymbol{x_1(2), x_2(2)}) \dots & \boldsymbol{\varphi_{\frac{(m+1)(m+2)}{2}}}(\boldsymbol{x_1(2), x_2(2)}) \\ & \vdots & \\ \boldsymbol{\varphi_1}(\boldsymbol{x_1(k), x_2(k)}) & \boldsymbol{\varphi_2}(\boldsymbol{x_1(k), x_2(k)}) \dots & \boldsymbol{\varphi_{\frac{(m+1)(m+2)}{2}}}(\boldsymbol{x_1(k), x_2(k)}) \end{bmatrix} . \begin{bmatrix} \boldsymbol{\theta_1} \\ \boldsymbol{\theta_2} \\ \vdots \\ \vdots \\ \boldsymbol{\theta_{\frac{(m+1)(m+2)}{2}}} \end{bmatrix} \quad (2.2)$$

where:

$$\varphi_i - \text{ all the possible products between the exponentiated } x_1 \text{ and } x_2 \; from \; (2.1)$$

$$\begin{bmatrix} \varphi_1(x_1(1), x_2(1)) & \varphi_2(x_1(1), x_2(1)) \dots & \varphi_{\frac{(m+1)(m+2)}{2}}(x_1(1), x_2(1)) \\ \varphi_1(x_1(2), x_2(2)) & \varphi_2(x_1(2), x_2(2)) \dots & \varphi_{\frac{(m+1)(m+2)}{2}}(x_1(2), x_2(2)) \\ & \vdots & \\ \varphi_1(x_1(k), x_2(k)) & \varphi_2(x_1(k), x_2(k)) \dots & \varphi_{\frac{(m+1)(m+2)}{2}}(x_1(k), x_2(k)) \end{bmatrix} = \phi - the \; regressor$$

2.3. Flattening the data

The problem that appears when we try to bring our data to the form described at (2.2) is that x_1 and x_2 are given as individual vectors and y is given as a matrix, so we can't form the tuples described above necessary for the matrix form.

What we have to do in order to fix this is to "flatten" our data by transforming y into a column vector (line by line) and x_1, x_2 into a matrix with two columns that contains each term of the cartesian product between the discrete values of x_1 and x_2 at each line.

After a model for our system is found, its outputs will also inherit this flat form, so they have to be brought back to matrix form in order to have any real meaning. This can be done by following the procedure for flattening in reverse order.

2.4. The unknown parameter vector

The unknown parameter vector, along with the degree of the polynomial (m) to which it corresponds is enough to mathematically characterize the model of our system. Thus, finding it becomes our main goal.

The problem of linear regression boils down to solving the system of linear equations described at (2.2) for the θ vector. We accomplished this by using MATLAB's integrated function for matrix left division:

$$\boldsymbol{\theta = \phi \setminus Y} \quad (2.3)$$

In order to obtain a set of outputs for any other input data, the regressor ϕ has to be computed for the new inputs and then simply multiplied with the parameter column vector θ that was found using (2.3).

3. Validation

3.1. Mean-squared error

To validate the output of our computed model and reach a good degree *m* for our polynomial, we need a precise, numerical indicator of how good our approximated data fits the real data. This is why we use the mean-squared error, as it computes the average difference between the estimated and the actual data, while always being positive due to the squared nature (an estimated data point may be greater or lower than the actual value of the real data and thus we can't simply subtract them). A smaller MSE value is translated to a better fit for our function.

$$\boldsymbol{MSE} = \frac{\mathbf{1}}{\boldsymbol{N}}\sum_{\boldsymbol{i=1}}^{\boldsymbol{N}}(\boldsymbol{y_i} - \boldsymbol{\hat{y}_i})^{\mathbf{2}} \tag{3.1}$$

where:

N - *the number of values in* y / $\hat{y}$

y – *the real output data vector*

$\hat{y}$ – *the estimated data vector*

3.2. Optimization and the risk of overfitting

Because the input-output data set that we use in order to find a model for our function is affected by noise, increasing the degree of the approximator polynomial after a certain point will give bad results for any other input data. In layman's terms, the approximator will find meaning in meaningless data. This creates the problem of optimization, the purpose of which is to find the largest value of *m* without overfitting the identification data set.

To optimize our model, we need a validation data set (different from the identification one). From this data, we take the inputs, plug them into the regressor and then multiply that by the parameter vector computed for the identification data, which will give us the approximated values for the output ($\hat{y}$). Finally, the MSE is computed against the real values for the output (y).

The process described above is repeated for an interval of integer values for *m* starting from 1, with the purpose of finding the minimum value of the MSE (least-squares method).

Figure 3.1 shows how poorly the model found for m=40 fits the validation data set (MSE=210.381) compared to the identification data set (MSE=0.64927).

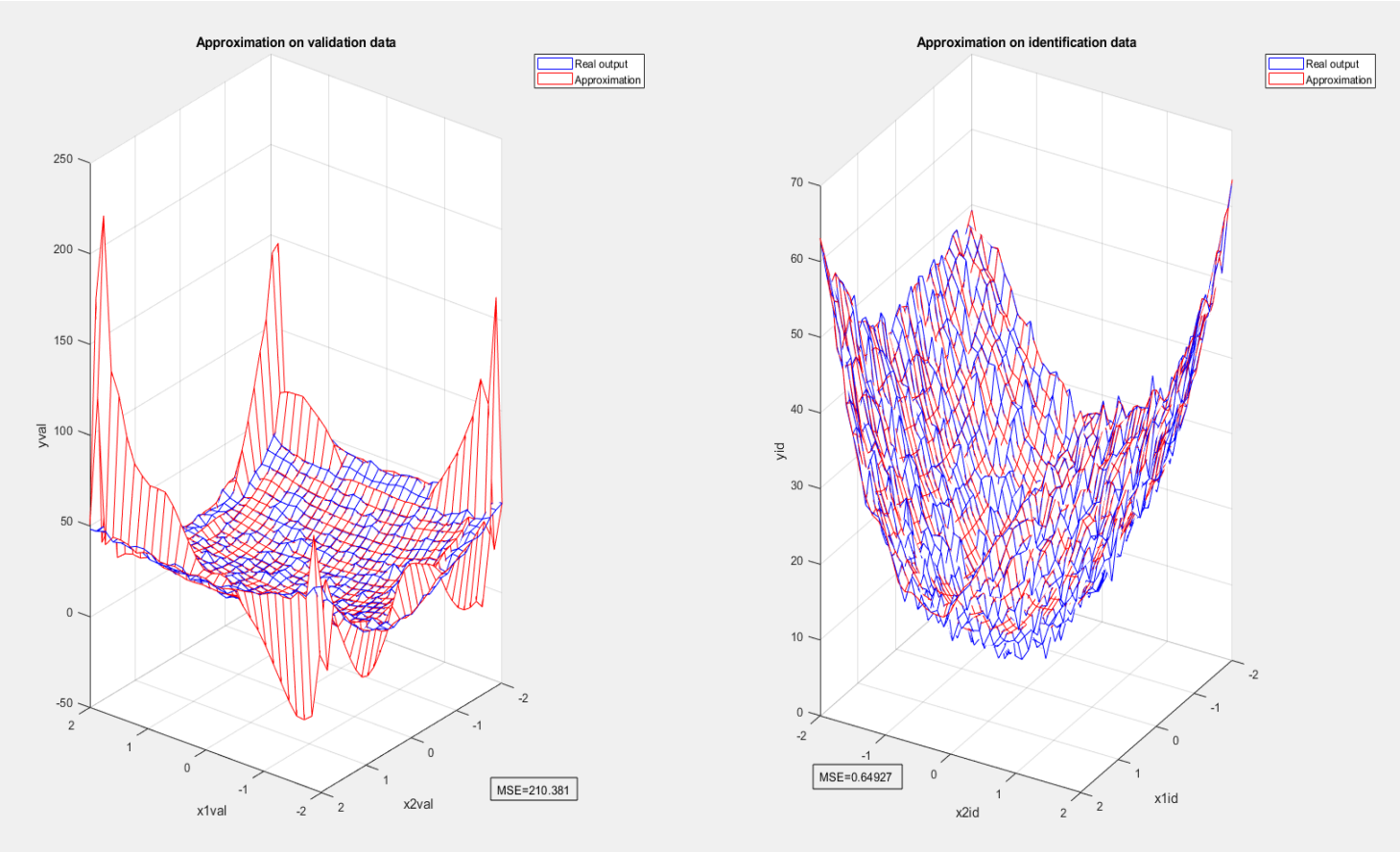

Fig. 3.1: example of overfitting for m=40

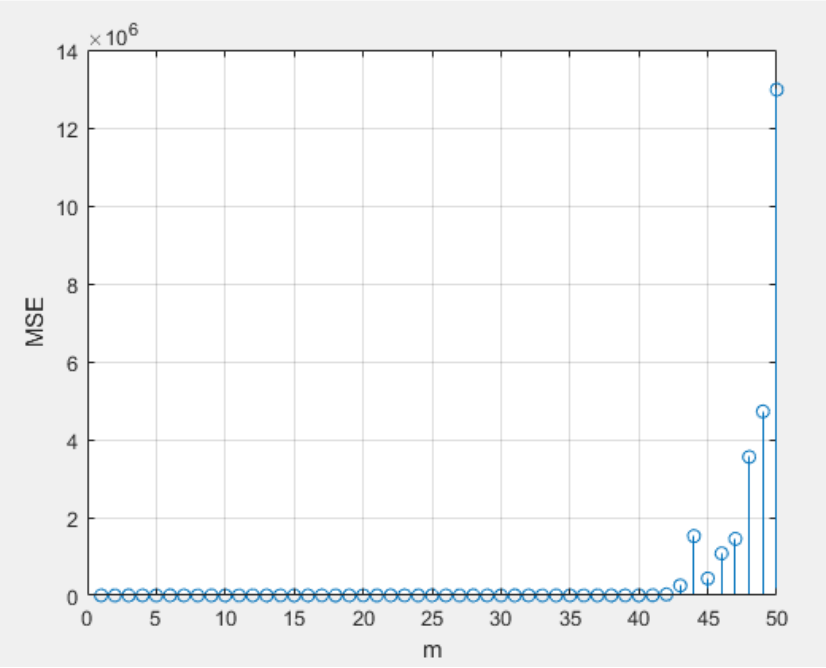

Fig. 3.2: MSE for different values of m

As it can be observed in fig. 3.3, the smallest MSE that we reached on the validation output vs. approximation was obtained for m=8. We also verified that this value is indeed the smallest using a 'for' statement in MATLAB.

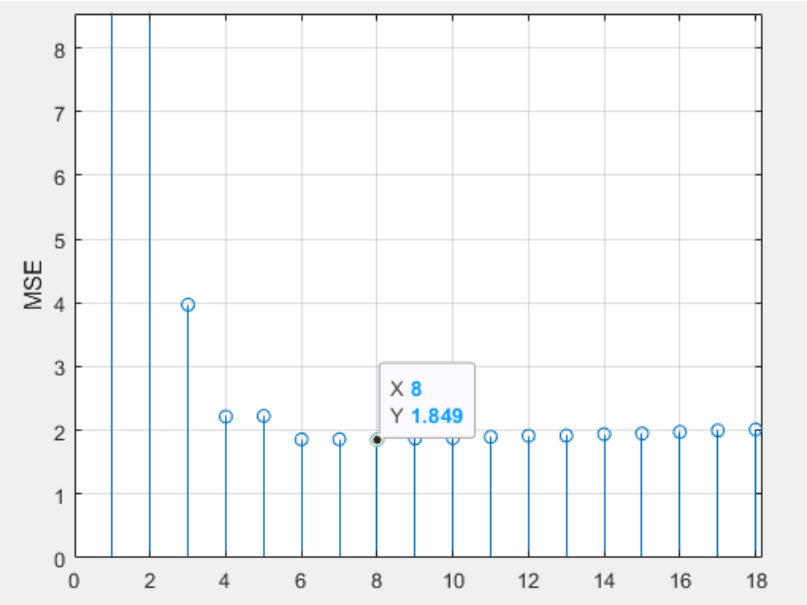

Fig. 3.3: Smallest MSE (zoomed in from fig. 3.2)

For m=8, the approximated output looks like:

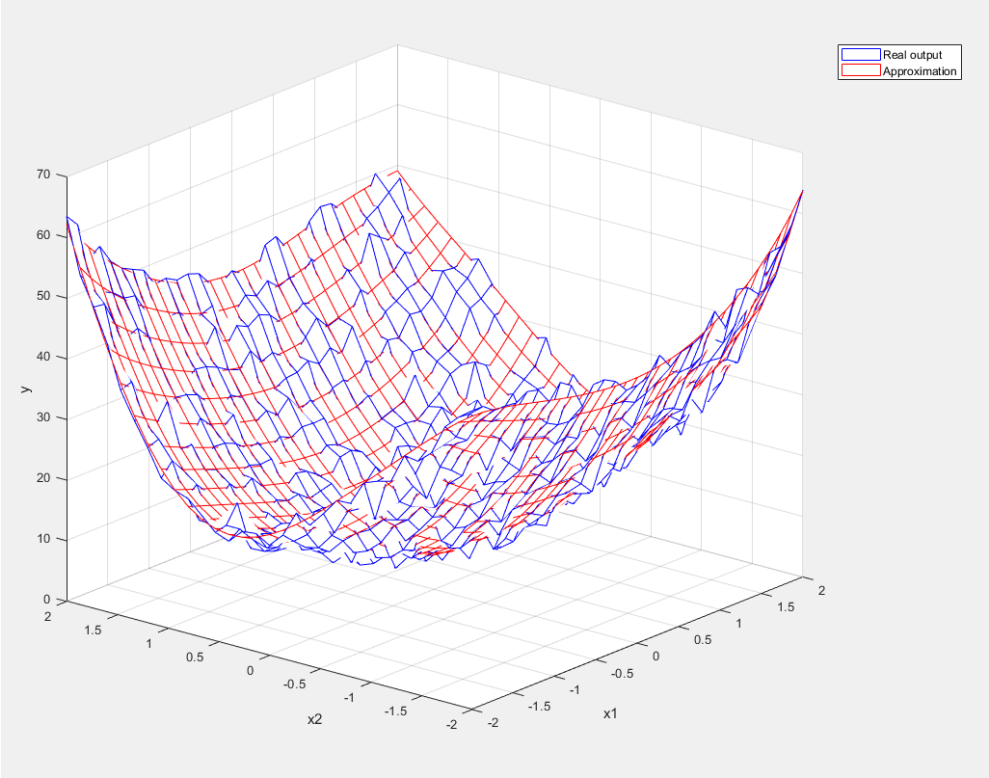

Fig. 3.4: Best approximation (m=8)

4. Conclusion

Linear regression is a useful tool that helps us fit unknown functions, but when used with polynomial approximators, it yields rather rough results compared to the real output of the system (fig 3.4. shows how details of the real output, the peaks and the valleys, are not fitted even though it is the best model from a least-squares perspective).

5. Appendices

A.1. MATLAB code

```
load proj_fit_22.mat %loading the given data sets
% identification data:
x1id=id.X{1};
x2id=id.X{2};
yid=id.Y;
surf(x1id,x2id,yid)
% validation data:
x1val=val.X{1};
x2val=val.X{2};
yval=val.Y;

%% the flattening:
ind=0;
for i=1:length(x1id)
    for j=1:length(x2id)
        ind=ind+1;
        xidflat(ind,1)=x1id(i);
        xidflat(ind,2)=x2id(j);
    end
end

ind=0;
for i=1:length(x1val)
    for j=1:length(x2val)
        ind=ind+1;
        xvalflat(ind,1)=x1val(i);
        xvalflat(ind,2)=x2val(j);
    end
end

sizeyid=size(yid);
ind=0;
for i=1:sizeyid(1)
    for j=1:sizeyid(2)
        ind=ind+1;
        yidflat(ind)=yid(i,j);
    end
end

sizeyval=size(yval);
ind=0;
for i=1:sizeyval(1)
    for j=1:sizeyval(2)
        ind=ind+1;
        yvalflat(ind)=yval(i,j);
    end
end
```

```
%% Validation
for m=1:50
    X=regressor(m,xidflat);
    theta = X\yidflat.';
    Xval=regressor(m,xvalflat);
    yhatflat = Xval*theta;
    MSE = 0;
    for i = 1:length(yvalflat)
        MSE = MSE + (yvalflat(i)-yhatflat(i))^2;
    end
    mse(m) = MSE/length(yvalflat);
end

best_m=1;
for i=1:length(mse)
    if mse(i)<mse(best_m)
        best_m=i;
    end
end

%% MSE plot
stem(mse(1:50))
grid
ylabel('MSE')
xlabel('m')

%% the unflattening of the best solution:
m=best_m;
X=regressor(m,xidflat);
theta = X\yidflat.';
Xval=regressor(m,xvalflat);
yhatflat = Xval*theta;

sizeyval=size(yval);
ind=0;
for i=1:sizeyval(1)
    for j=1:sizeyval(2)
        ind=ind+1;
        yhat(i,j)=yhatflat(ind);
    end
end

mesh(x1val,x2val,yval,'EdgeColor','blue'); hold on
mesh(x1val,x2val,yhat,'EdgeColor','red'); hold on
legend('Real output','Approximation')
xlabel('x1')
ylabel('x2')
zlabel('y')

%% m=40 overfit
m=40;
X=regressor(m,xidflat);
theta = X\yidflat.';
Xval=regressor(m,xvalflat);
yhatflat = Xval*theta;

sizeyval=size(yval);
ind=0;
for i=1:sizeyval(1)
    for j=1:sizeyval(2)
        ind=ind+1;
        yhat(i,j)=yhatflat(ind);
    end
end
subplot(121)
mesh(x1val,x2val,yval,'EdgeColor','blue'); hold on
mesh(x1val,x2val,yhat,'EdgeColor','red'); hold on
```

```
legend('Real output','Approximation')
xlabel('x1val')
ylabel('x2val')
zlabel('yval')
title('Approximation on validation data')
%
X=regressor(m,xidflat);
theta = X\yidflat.';
yflat=X*theta;
sizey=size(yid);
ind=0;
for i=1:sizey(1)
    for j=1:sizey(2)
        ind=ind+1;
        y(i,j)=yflat(ind);
    end
end

MSE = 0;
for i = 1:length(yid)
    MSE = MSE + (yid(i)-y(i))^2;
end
MSE=MSE/length(yid);
subplot(122)
mesh(x1id,x2id,yid,'EdgeColor','blue'); hold on
mesh(x1id,x2id,y,'EdgeColor','red'); hold on
legend('Real output','Approximation')
xlabel('x1id')
ylabel('x2id')
zlabel('yid')
title('Approximation on identification data')

%%
%function for building the regressor matrix:
function r = regressor(m,x)
    size_x=size(x);
    length_x=size_x(1);

    for i=1:length_x
        r(i,1)=1;
        for j=1:m
             r(i,2*j)=x(i,1)^j;
             r(i,2*j+1)=x(i,2)^j;
        end
        ind=2*m;
        for j=1:m-1
            for k=1:m-j
                ind=ind+1;
                r(i,ind)=(x(i,1)^j)*(x(i,2)^k);
            end
        end
     end
end
```