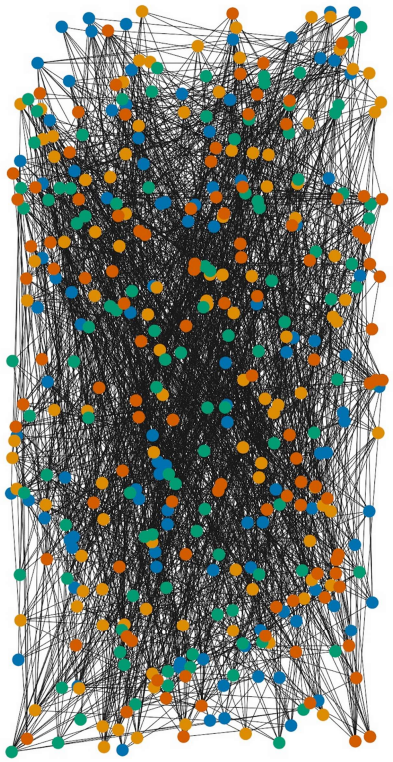
23.08.2023

Neural Networks

Understanding The Least Squares Algorithm-Applications in MATLAB

Version 05



Prof. Dr. Mehmet Önder Efe

Contents

1. Introduction

The Least Squares (LS) algorithm was proposed by Widrow and Hopf [1] in 1960 and proposes a solution **based on matrix algebra**, [1-3]. The algorithm is **related to neural networks** closely.

In this report, the development of models from the data obtained for N **observations** will be considered. The Least Squares (LS) method will be used to determine the parameters of the model. This method, which is frequently encountered in the literature, **produces the best parameters** that the model can have after the model structure is selected. Here, issues such as **pre-processing of data, extraction of data showing exceptional characteristics will not be considered**, and all available data will be assumed to be used. The variables to use in the report are listed below.

Table 1: Used Variables and Their Meanings

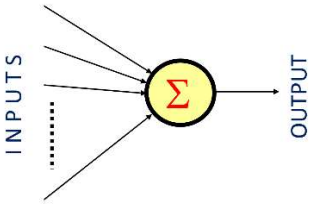
t	Independent variable (time)
N	Number of measurement points
m	Number of model parameters
x	m×1 vector, the parameter vector of the model
A	A matrix of dimensions N×m
b	Measurements of dimension N×1
J	Cost (loss) function
f	A scalar function of two vectors
E	Error vector
y	Output variable of the process. Process is the data generating system.
$\hat{y}$	Output of the model. Model is the systen that fits the process output.

According to the above list, it is also **necessary to know the structure of the model** with m parameters. For example, the process given below is the kind of process in which we can apply the LS algorithm.

$$\underbrace{y = a_1x_1 + a_2x_2 + \cdots + a_mx_m}_{\text{Truth}} \quad \text{or} \quad \underbrace{y = a_1x_1 + a_2x_2 + \cdots + a_mx_m + \text{noise}}_{\text{Noisy measurements}} \tag{1.1}$$

Since the a_i's multiply x_i's and they are summed, **the model above is linear and LS algorithm is applicable**. We can rewrite this as

$$y = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \end{bmatrix}^T \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} = \underbrace{\begin{bmatrix} a_1 & a_2 & \cdots & a_m \end{bmatrix}}_{\text{Coefficients}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}}_{\text{Unknowns}} \tag{1.2}$$



If we can separate the terms like we see above, **LS algorithm can solve the unknowns using the pseudoinverse**. We will consider this in detail yet LS algorithm is applicable when the model is not linear. A good model parameter set can be obtained by **using neural networks iteratively**. In LS algorithm, we will **get the solution in one step**.

LS algorithm is used successfully in many areas of engineering, e.g. signal and image processing, digital filters, prediction models, and many machine learning algorithms.

1.1. The Data

The data set we will use here looks like the following. Here $x_1, x_2, \cdots, x_m$ parameters (unknowns) multiply a_i columns and scale those columns.

Table 2: Strucure of the Dataset

a_1	a_2	...	a_m
a_{11}	a_{12}	...	a_{1m}
a_{21}	a_{22}	...	a_{2m}
a_{31}	a_{32}	...	a_{3m}
$\vdots$	$\vdots$	$\ddots$	$\vdots$
a_{N1}	a_{N2}	...	a_{Nm}

b (y_i values)
b_1
b_2
b_3
$\vdots$
b_N

Let's define A and b as follows:

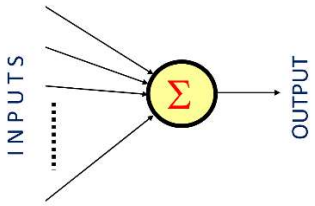
$$A := \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & \cdots & a_{Nm} \end{bmatrix}, \quad b := \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{bmatrix} \tag{1.3}$$

Every row of the dataset **generates an equation, this equation is the claim of data from the model**. The equality of the first row is obtained as follows:

$$\begin{bmatrix} a_{11} \\ a_{12} \\ \vdots \\ a_{1m} \end{bmatrix}^T \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} = \underbrace{\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \end{bmatrix}}_{\text{Coefficients}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}}_{\text{Unknowns}} = a_{11}x_1 + a_{12}x_2 + \cdots + a_{1m}x_m = b_1 \tag{1.4}$$

The rightmost term above is b_1, if we write down the equations; we will obtain the following matrix equation.

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & \cdots & a_{Nm} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{bmatrix} \tag{1.5}$$



Shortly, we will write this equation as in (1.6).

$$Ax = b \tag{1.6}$$

In above, $A \in \Re^{N \times m}, x \in \Re^{m \times 1}, b \in \Re^{N \times 1}$ and since A is (generally) not square, **we cannot take the inverse of it**. In a typical modeling application, we have many rows of data and few parameters, that is, we have **typically $N > m$**. In such a case, A is a tall rectangular matrix, which is not square.

Now we wrote down the equations, now we can derive the solution formula and interpret it.

2. Derivation of the Algorithm

In order to solve the equation set given by $Ax = b$, let's **multiply both sides of $Ax = b$ by A^T from the left**[1]. This would yield $A^T Ax = A^T b$. Here $A^T A$ is a square matrix. Now we can solve this equation set for the unknown x. Let's left-**multiply both sides by the inverse of $A^T A$**. This would produce the solution in (2.1).

$$x = (A^T A)^{-1} A^T b \tag{2.1}$$

In this solution, $A^+ := (A^T A)^{-1} A^T$ is the pseudoinverse of A. This solution gives the best parameter vector for the given dataset yet when we use this x, instead of b, we would get

$$Ax = A(A^T A)^{-1} A^T b \tag{2.2}$$

Therefore, the model produces the following error:

$$\begin{aligned} E &:= b - A\underbrace{(A^T A)^{-1} A^T b}_{x} \\ &= b - Ax \end{aligned} \tag{2.3}$$

The solution x will produce the smallest $E^T E$ among all other alternatives. We can study this in two different ways.

2.1. Derivation of LS Algorithm Using the Gradients

Let the matrix A has full column rank. Let x and y be vectors of appropriate dimensions, and let F be a matrix. We have the following scalar functions of x, y and F, and we have the gradients given as below.

$$f(x,y) = x^T Fy \qquad \nabla_x f(x,y) = Fy \tag{2.4}$$

$$f(x,y) = y^T Fx \qquad \nabla_x f(x,y) = F^T y \tag{2.5}$$

$$f(x,y) = x^T Fx \quad \nabla_x f(x,y) = \left(F + F^T\right) x \tag{2.6}$$

[1] Why? If we multiply by another matrix, say B, the solution will not yield the smallest possible $E^T E$.

Now write the cost function as given in (2.7).

$$\begin{aligned} J &= E^T E \\ &= (b - Ax)^T (b - Ax) \end{aligned} \tag{2.7}$$

We can expand the terms as given below.

$$\begin{aligned} J &= \left(b^T - \left(Ax\right)^T\right)(b - Ax) \\ &= \left(b^T - x^T A^T\right)(b - Ax) \\ &= b^T b - b^T Ax - x^T A^T b + x^T A^T Ax \end{aligned} \tag{2.8}$$

We can now utilize (2.4)-(2.6) and calculate the gradients of each term above, i.e. we have

$$\begin{aligned} &\nabla_x J = \nabla_x \left(b^T b - b^T Ax - x^T A^T b + x^T A^T Ax\right) \\ &= \nabla_x \left(b^T b\right) - \nabla_x \left(b^T Ax\right) - \nabla_x \left(x^T A^T b\right) + \nabla_x \left(x^T A^T Ax\right) \end{aligned} \tag{2.9}$$

The first term above does not depend on x, so we have (2.10).

$$\nabla_x \left(b^T b\right) = 0 \tag{2.10}$$

For the second term, we use (2.5), which says $f(x, y) = y^T Fx,\ \nabla_x f(x, y) = F^T y$

$$\nabla_x \left(b^T Ax\right) = A^T b \tag{2.11}$$

For the third term, we use (2.4), which says $f(x, y) = x^T Fy,\ \ \nabla_x f(x, y) = Fy$

$$\nabla_x \left(x^T A^T b\right) = A^T b \tag{2.12}$$

For the last term, we use (2.6), which says $f(x, y) = x^T Fx,\ \nabla_x f(x, y) = \left(F + F^T\right)x$

$$\nabla_x \left(x^T A^T Ax\right) = \left(A^T A + (A^T A)^T\right)x = 2A^T Ax \tag{2.13}$$

Concatenating the terms and equating the gradient to zero would yield the solution x as follows:

$$\begin{aligned} &\nabla_x J = -A^T b - A^T b + 2A^T Ax \\ &= -2A^T b + 2A^T Ax \\ &= 0 \qquad \Rightarrow x = (A^T A)^{-1} A^T b \end{aligned} \tag{2.14}$$

2.2. Derivation of the Algorithm with Vectoral Interpretation

Let's use the columns of matrix A, i.e. a_i for the i-th column.

$$A = \begin{bmatrix} a_1 & a_2 & \cdots & a_m \end{bmatrix} \tag{2.15}$$

We can write the transpose of matrix A as given below.

$$A^T = \begin{bmatrix} a_1^T \\ a_2^T \\ \vdots \\ a_m^T \end{bmatrix} \tag{2.16}$$

Now focus on Figure 1. The plane in Figure 1 is the column space of the matrix A. The vector b in the figure is the b vector in our equation set $Ax=b$, the **vector b_1 is the implementable part of b**, and the **vector b_2 is the unimplementable part of vector b**.

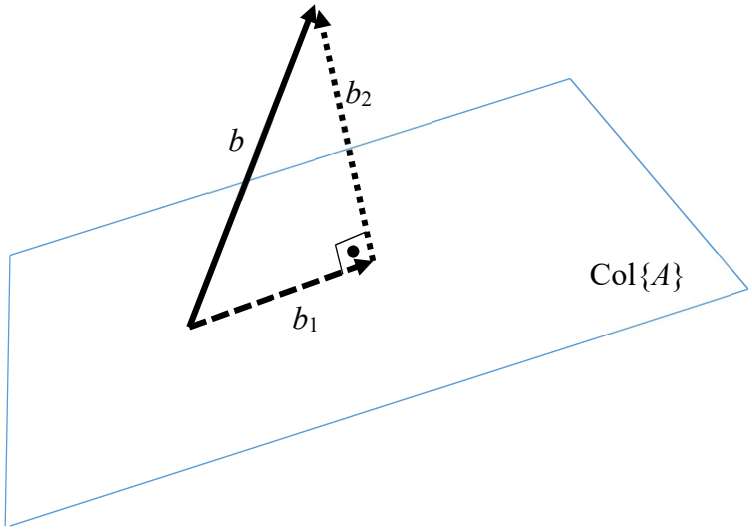


Figure 1: Column space of matrix A, vector b, implementable vector b_1, unimplementable vector b_2.

At this point, we must explain $\text{Col}\{A\}$. Scaling every column in $A = \begin{bmatrix} a_1 & a_2 & \cdots & a_m \end{bmatrix}$ and adding up creates a vector. This is as seen below; the entries of x are the scalors.

$$Ax = \begin{bmatrix} a_1 & a_2 & \cdots & a_m \end{bmatrix} x = \underbrace{a_1x_1 + a_2x_2 + \cdots + a_mx_m}_{a_i\text{'s are columns}} \tag{2.17}$$

The result Ax is a vector and it lies in the plane shown in Figure 1. In other words, for every x_1, x_2,…, x_m constants, the **set of vectors obtained via Ax lies in $\text{Col}\{A\}$**, which is the plane in Figure 1. Once again, the space named **$\text{Col}\{A\}$ is formed by linearly combining the columns of the matrix A**.

According to Figure 1, we see that the vector b is not in the column space of A and we cannot obtain b by linearly combining the columns of matrix A. **However, the best we can do is b_1, which is the projection of b onto the column space of A.** The vectors b_1 and b_2 are related to b as given below:

$$b = b_1 + b_2 \tag{2.18}$$

b_2 is perpendicular to $\mathrm{Col}\{A\}$. More explicitly, b_2 is perpendicular to all columns of A. We can write the following.

$$A^T b_2 = \begin{bmatrix} a_1^T \\ a_2^T \\ \vdots \\ a_m^T \end{bmatrix} b_2 = \begin{bmatrix} a_1^T b_2 \\ a_2^T b_2 \\ \vdots \\ a_m^T b_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = 0 \tag{2.19}$$

This leads to

$$\begin{aligned} x &= (A^T A)^{-1} A^T b \\ &= (A^T A)^{-1} A^T (b_1 + b_2) \\ &= \underbrace{(A^T A)^{-1} \underbrace{A^T b_2}_{0}}_{0} + (A^T A)^{-1} A^T b_1 \\ &= (A^T A)^{-1} A^T b_1 \end{aligned} \tag{2.20}$$

The final equation above tells us that **the best we can do is to obtain b_1 and the solution x realizes b_1, therefore the solution x is optimal in this setting.**

3. Numerical Examples

In this section, we will consider the results on a number of exemplar cases. We will consider

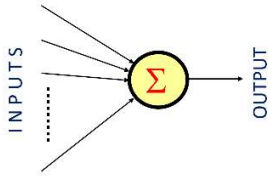
- **The number of data**
- **Model complexity**
- **Noise**

in our examples. The programming will be done in Matlab and we will give sample code in the document.

3.1 Example-1: y=t+1, yhat=x_1t+x_2, x_1=1, x_2=1

The true process that generates the true data is given as below.

$$y = t + 1 \tag{3.1-27}$$



Let's **assume that we know the structure above but we do not know the coefficients**, i.e. we choose the following model:

$$\hat{y} = x_1 t + x_2 \tag{3.2}$$

Clearly, if our approach works, we must find x_1=1 and x_2=1. We chose the model in (3.2) for the process in (3.1), why? This is an important question that we will discuss in detail[2]. Below is a Matlab code that generates the following figures.

```
clear all
close all
clc

FigureCounter = 1;
Earray=[];
Nvec = [2 5 10 20 50 100 200 500 1000 10000];
for N=Nvec;

    NoiseLevel = 0.1;

    t=linspace(0,1,N)';
    n = NoiseLevel*(2*rand(N,1)-1);
    y = t+1;

    figure(FigureCounter)
    set(gcf,'Position',[203.4000 199 808 290.4000])

    subplot(1,3,3)
    plot(t,y+n,'or')
    A=[t ones(size(t))];
    p=inv(A'*A)*A'*(y+n)
    hold on
    ym = p(1)*t+p(2); % yhat
    subplot(1,3,3)
    plot(t,y,'-b',t,ym,'-g','LineWidth',2)
    xlabel('t')
    ylabel('y')
    E=(1/N)*((y-ym)'*(y-ym))^0.5;
    Earray = [Earray;E];
    grid
    title({['N=' num2str(N) '    (1/N)\Sigma_{i=1}^Ne_i^2= ' num2str(E)],[' x_1=' num2str(p(1))
' x_2=' num2str(p(2))]},'FontWeight','normal')
    legend('Data Points','True Line','Model Response','Location','northwest')
    ax = axis;

    subplot(1,3,1)
    plot(t,y,'-b','LineWidth',2)
    xlabel('t')
    ylabel('y')
    title('True Line','FontWeight','normal')
    grid on
    legend('True Line','Location','northwest')
    axis(ax)

    subplot(1,3,2)
    plot(t,y+n,'or')
    hold on
    plot(t,y,'-b','LineWidth',2)
    xlabel('t')
    title(['|Noise Level| \leq ' num2str(NoiseLevel)],'FontWeight','normal')
    legend('Noisy Data','True Line','Location','northwest')
    grid on
    axis(ax)

    FigureCounter = FigureCounter + 1;
end
```

[2] For example, we chose $\hat{y} = x_1 t + x_2$ but why didn't we not chose $\hat{y} = x_1 t + x_2 + x_1 \cos(t)$? We will return to this issue.

The additive noise level in all figures produced by the above code is considered as 0.1. This means that the process allows only noisy measurements given as $y = t + 1 + \text{noise}$ and the noise signal has a uniform distribution between -1 and +1.

In Figure 2-Figure 11, **we increase the number of observations (N, the number of data)** and we consider the numbers in **Nvec** in the above Matlab code. In Figure 2-Figure 11, **on the left we see the true line**, **we see the true line and observed noisy data in the middle**, and on the right, the noise data points, true line and the model response (green) are shown together.

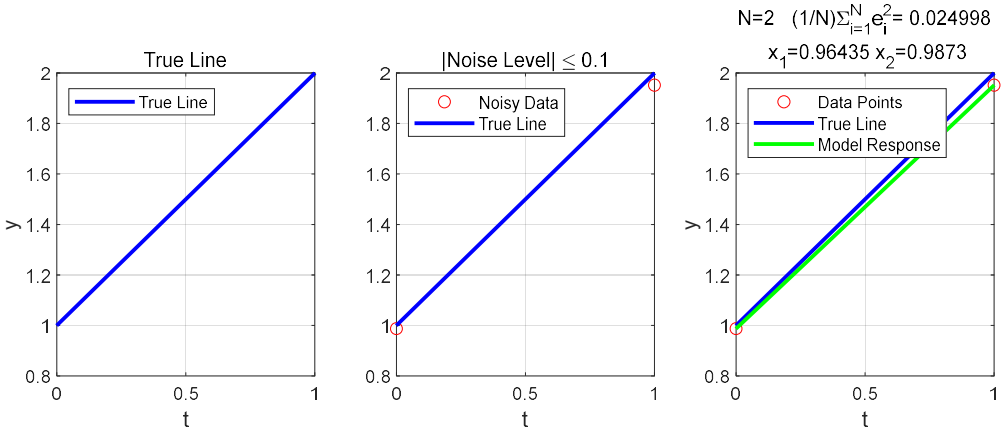


Figure 2: Least squares modeling of a first order process with 2 data points

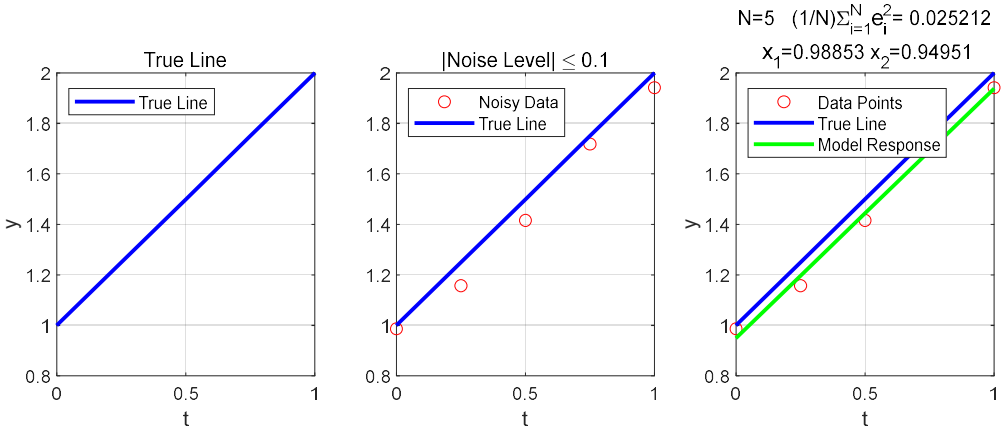


Figure 3: Least squares modeling of a first order process with 5 data points

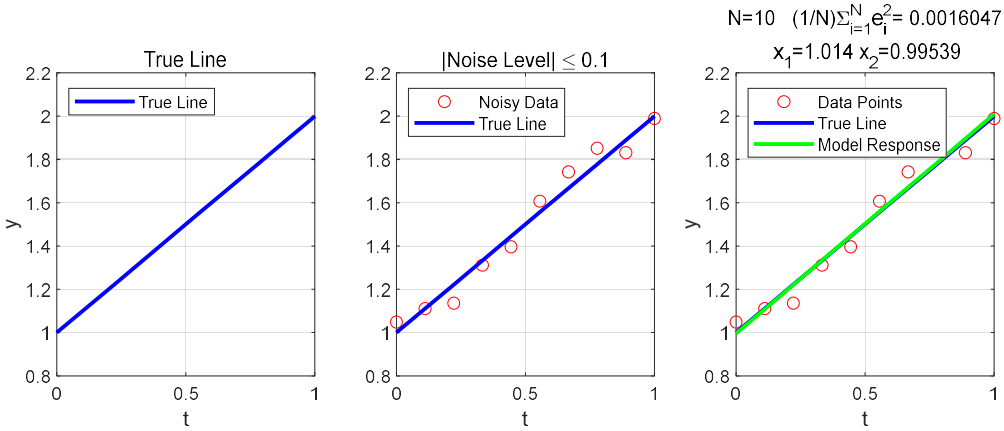


Figure 4: Least squares modeling of a first order process with 10 data points

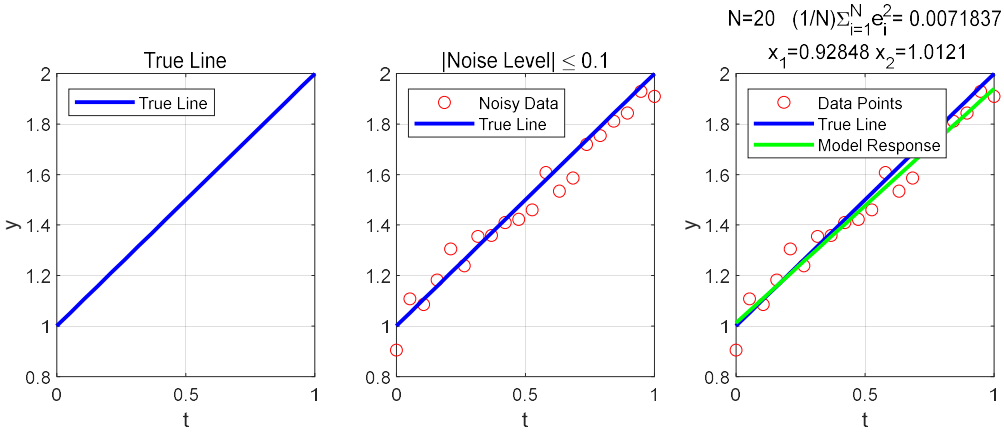


Figure 5: Least squares modeling of a first order process with 20 data points

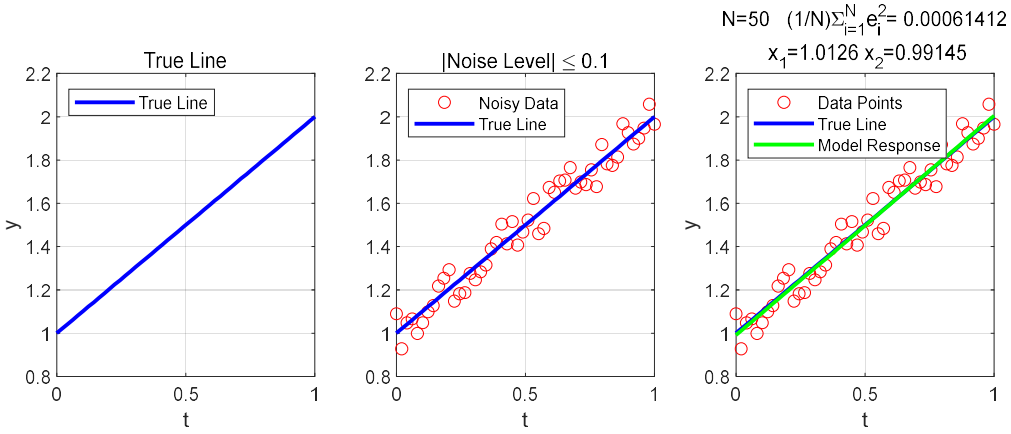


Figure 6: Least squares modeling of a first order process with 50 data points

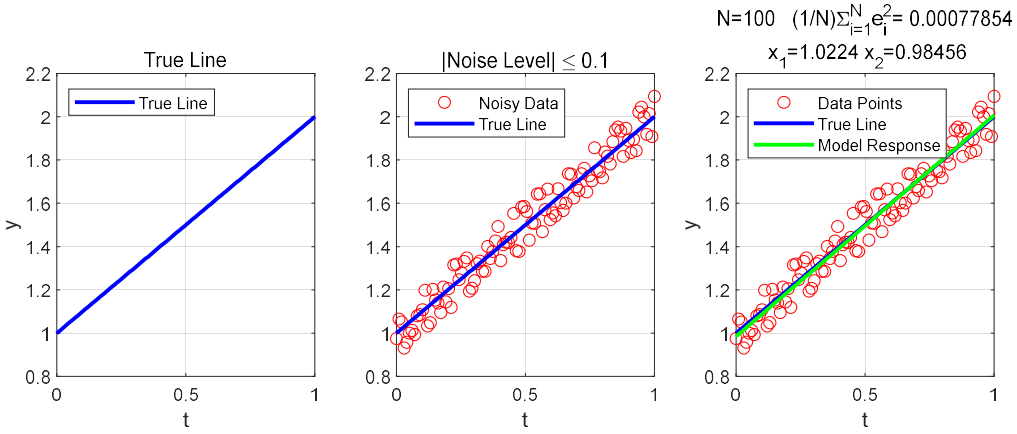


Figure 7: Least squares modeling of a first order process with 100 data points

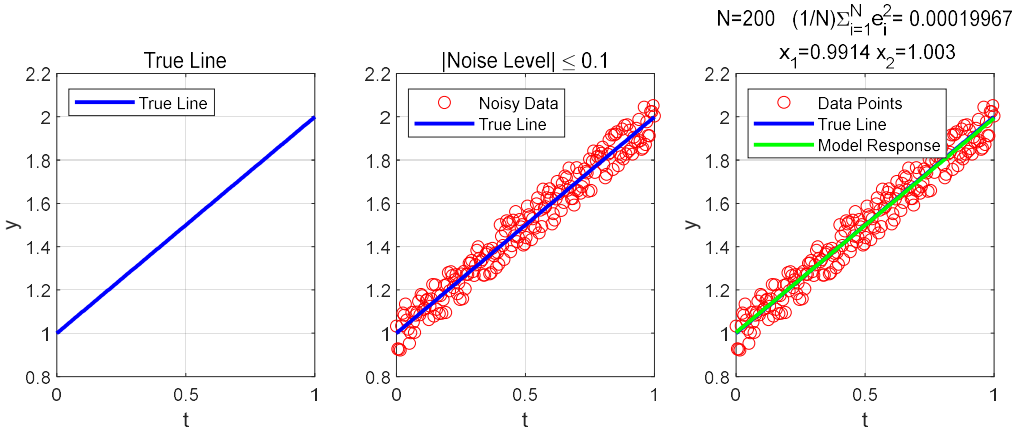


Figure 8: Least squares modeling of a first order process with 200 data points

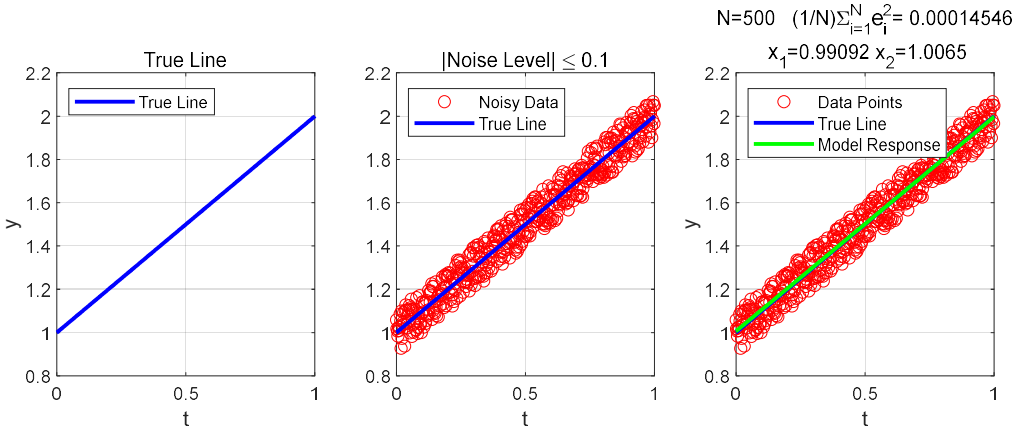


Figure 9: Least squares modeling of a first order process with 500 data points

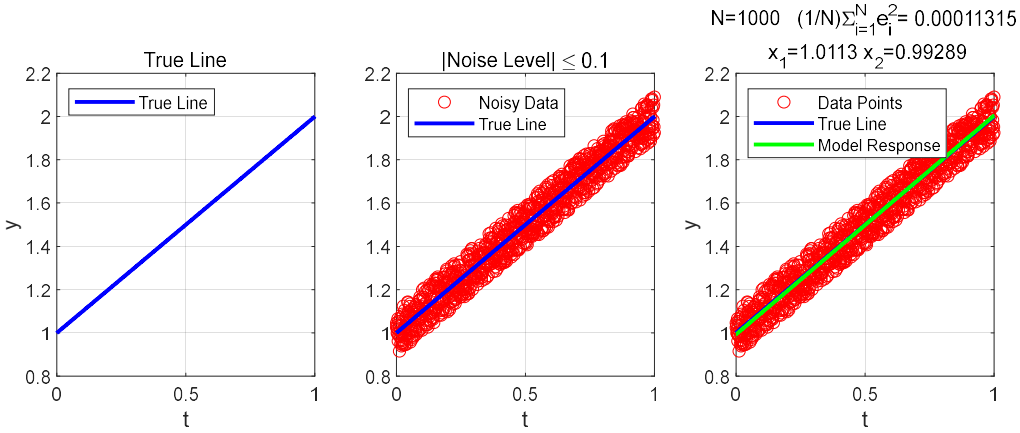


Figure 10: Least squares modeling of a first order process with 1000 data points

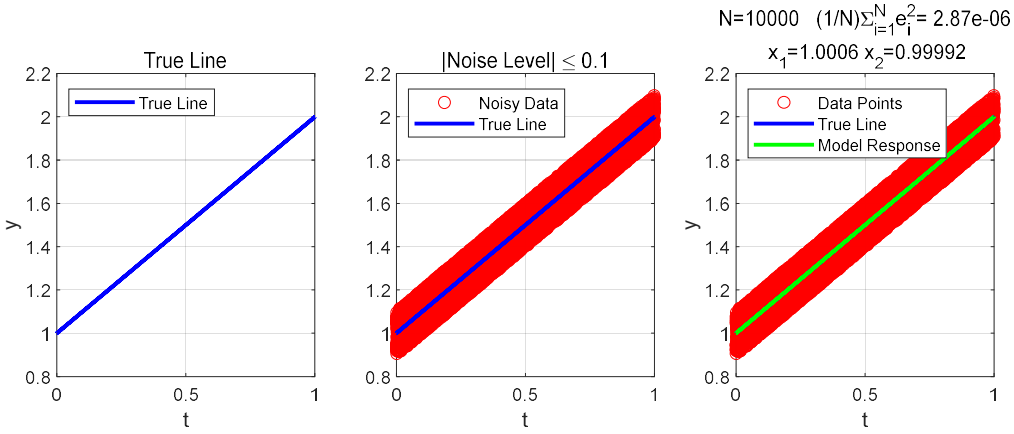


Figure 11: Least squares modeling of a first order process with 10000 data points

According to the shown results, as the number of obervations (N) increases, the performance gets better. This is seen well in Figure 12, which is based on the cost function below.

$$\text{Mean Squared Error} = \frac{1}{N} E^T E = \frac{1}{N} (b - Ax)^T (b - Ax) \tag{3.3}$$

We will now see how the mean squared error above changes when we increase N. We will do many experiments and monitor the mean value for each N.

```
clear all
close all
clc

NumberExperiment = 20;
Nvec = [2 5 10 20 50 100 200 500 1000 2000 5000 10000 20000 50000];
Earray=zeros(length(Nvec),NumberExperiment);

for experiment = 1:NumberExperiment
    vec = [];
    for N=Nvec;
        NoiseLevel = 0.1;
        t=linspace(0,1,N)';
        n = NoiseLevel*(2*rand(N,1)-1);
        y = t+1;
        A=[t ones(size(t))];
        p=inv(A'*A)*A'*(y+n);
        ym = p(1)*t+p(2); % yhat
        E=(1/N)*((y-ym)'*(y-ym))^0.5;
        vec = [vec;E];
    end
    Earray = [Earray vec];
end
figure
subplot(1,2,1)
loglog(Nvec,Earray,'ok','LineWidth',1)
hold on
loglog(Nvec,Earray,'LineWidth',1)
xlabel('N')
ylabel('Mean Squared Error')
grid
axis tight

subplot(1,2,2)
loglog(Nvec,mean(Earray'),'ok','LineWidth',2)
hold on
loglog(Nvec,mean(Earray'),'-b','LineWidth',2)
xlabel('N')
ylabel('Averaged Mean Squared Error')
grid
axis tight
```

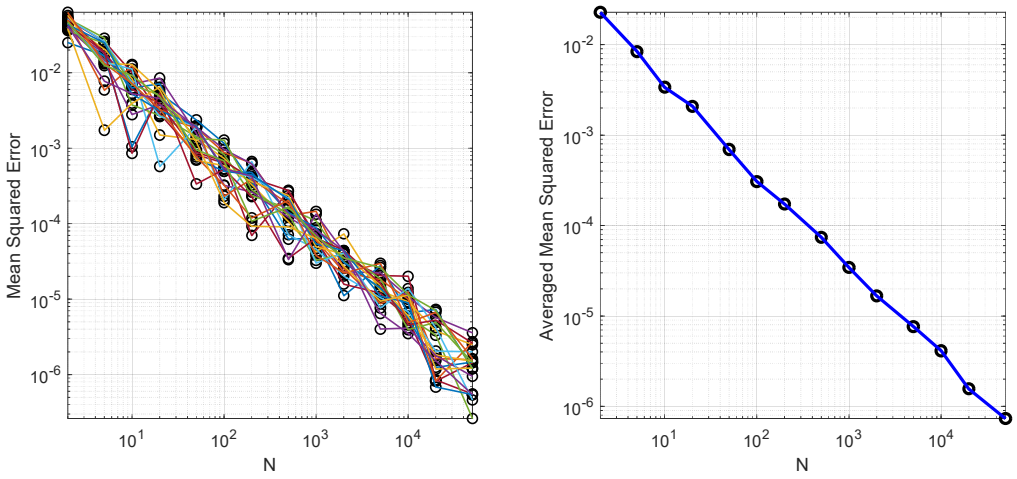


Figure 12: Change of squared error values on the left and their mean is on the right. As N gets larger, predictions made by the model become better. Data is valuable!

In the second set of experiments, we will fix N, and analyze the effects of noise on the performance. The Matlab code is given below.

```
clear all
close all
clc

FigureCounter = 1;
Earray=[];
N=100;
NL = [0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1];
for NoiseLevel=NL;

    t=linspace(0,1,N)';
    n = NoiseLevel*(2*rand(N,1)-1);
    y = t+1;

    figure(FigureCounter)
    set(gcf,'Position',[203.4000 199 808 290.4000])

    subplot(1,3,3)
    plot(t,y+n,'or')
    A=[t ones(size(t))];
    p=inv(A'*A)*A'*(y+n)
    hold on
    ym = p(1)*t+p(2); % yhat
    subplot(1,3,3)
    plot(t,y,'-b',t,ym,'-g','LineWidth',2)
    xlabel('t')
    ylabel('y')
    E=(1/N)*((y-ym)'*(y-ym))^0.5;
    Earray = [Earray;E];
    grid
    title({['N=' num2str(N) '    (1/N)\Sigma_{i=1}^Ne_i^2= ' num2str(E)],[' x_1=' num2str(p(1))
' x_2=' num2str(p(2))]},'FontWeight','normal')
    legend('Data points','True Line','Predicted Line','Location','northwest')
    ax = axis;

    subplot(1,3,1)
    plot(t,y,'-b','LineWidth',2)
    xlabel('t')
    ylabel('y')
    title('True Line','FontWeight','normal')
    grid on
    legend('True Line','Location','northwest')
    axis(ax)
```

```
    subplot(1,3,2)
    plot(t,y+n,'or')
    hold on
    plot(t,y,'-b','LineWidth',2)
    xlabel('t')
    ylabel('y')
    title(['|Noise Level| \leq ' num2str(NoiseLevel)],'FontWeight','normal')
    legend('Noisy Data','True Line','Location','northwest')
    grid on
    axis(ax)

    FigureCounter = FigureCounter + 1;
end

figure
plot(NL,Earray,'ok','LineWidth',2)
hold on
plot(NL,Earray,'-b','LineWidth',2)
xlabel('Noise Level')
ylabel('Mean Squared Error')
grid
axis tight
title(['max(E)=' num2str(max(Earray)) '   min(E)='
num2str(min(Earray))],'FontWeight','normal')
```

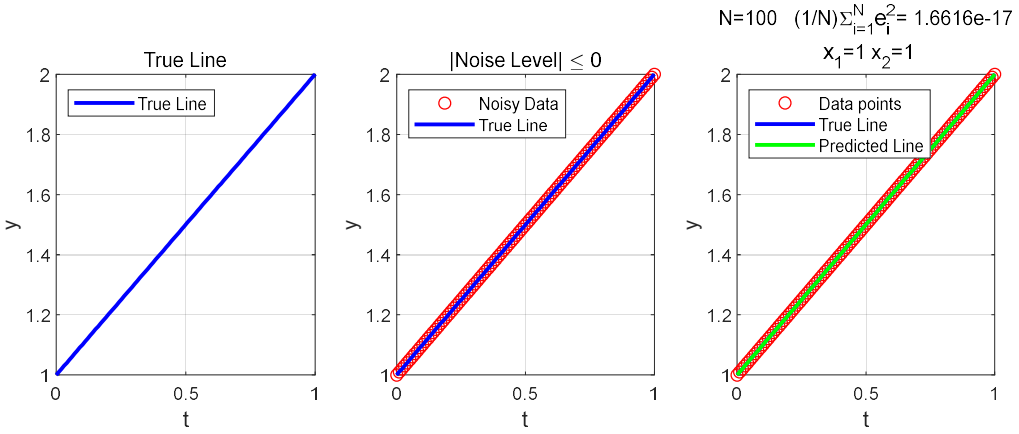


Figure 13: Results with N=100 data points and Noise Level = 0

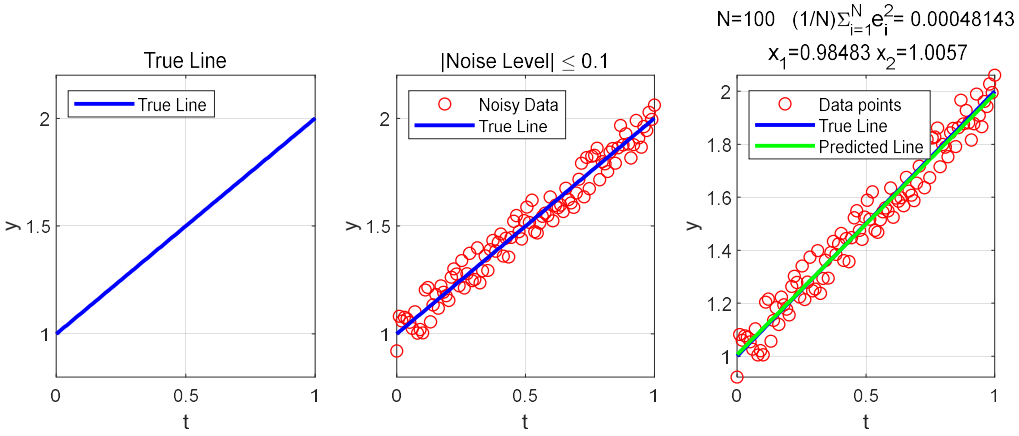


Figure 14: Results with N=100 data points and Noise Level = 0.1

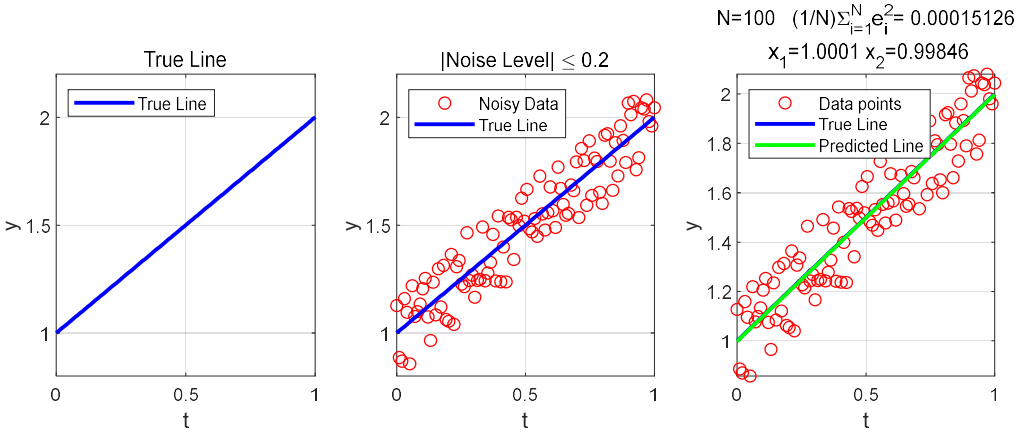


Figure 15: Results with N=100 data points and Noise Level = 0.2

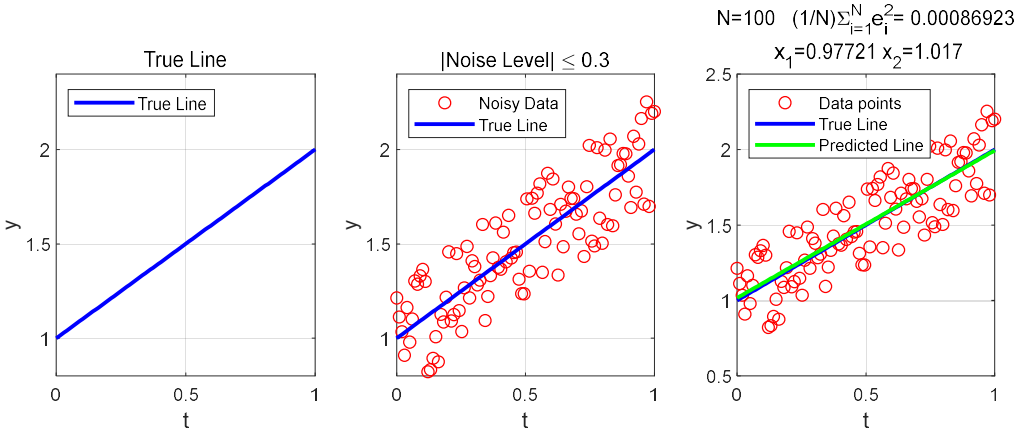


Figure 16: Results with N=100 data points and Noise Level = 0.3

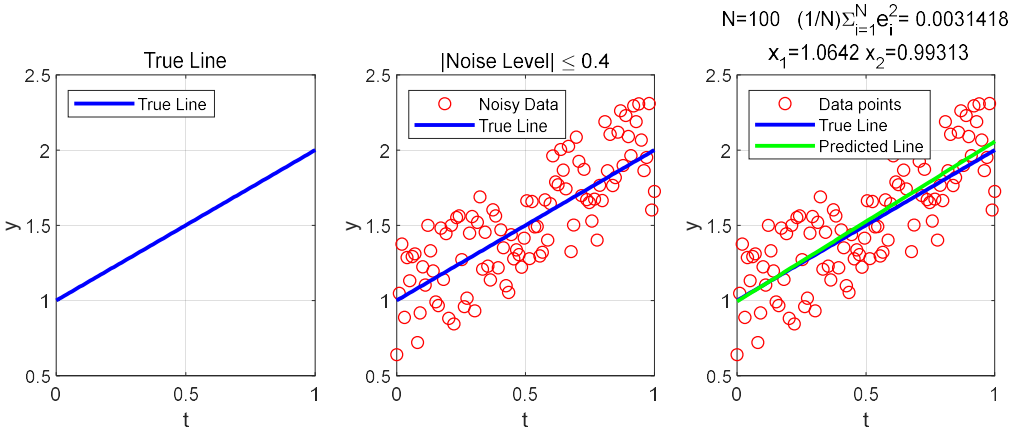


Figure 17: Results with N=100 data points and Noise Level = 0.4

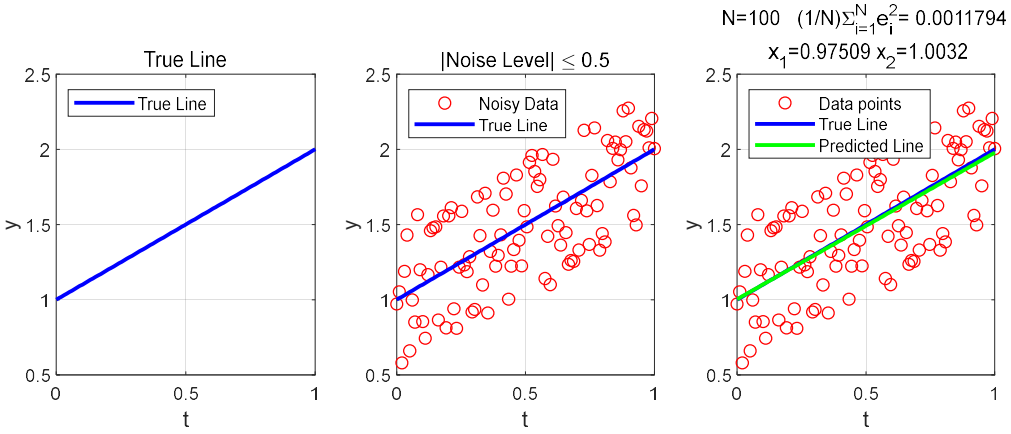


Figure 18: Results with N=100 data points and Noise Level = 0.5

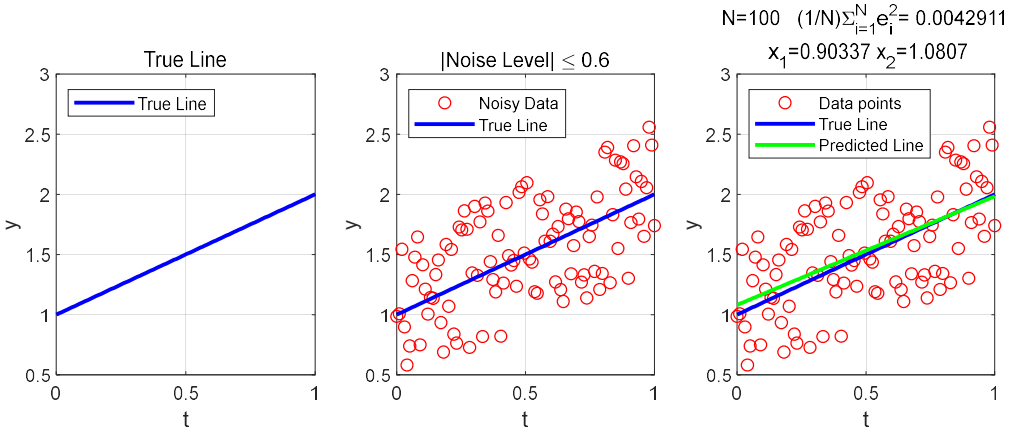


Figure 19: Results with N=100 data points and Noise Level = 0.6

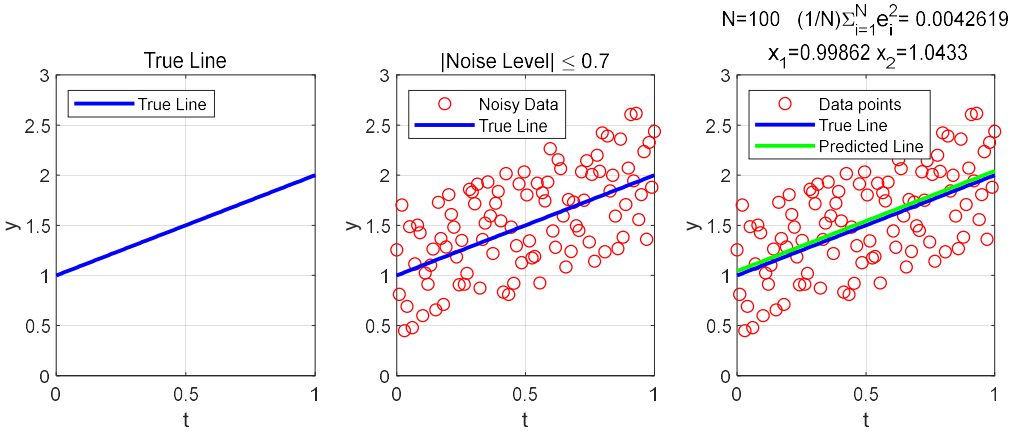


Figure 20: Results with N=100 data points and Noise Level = 0.7

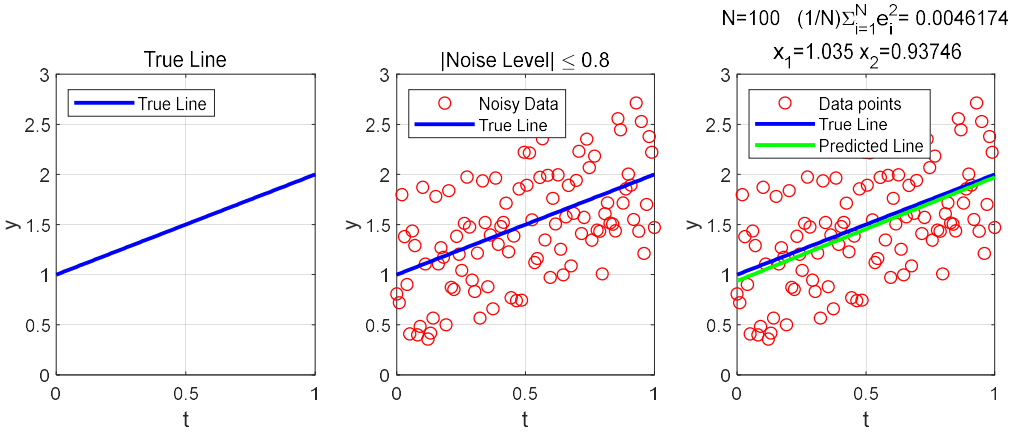


Figure 21: Results with N=100 data points and Noise Level = 0.7

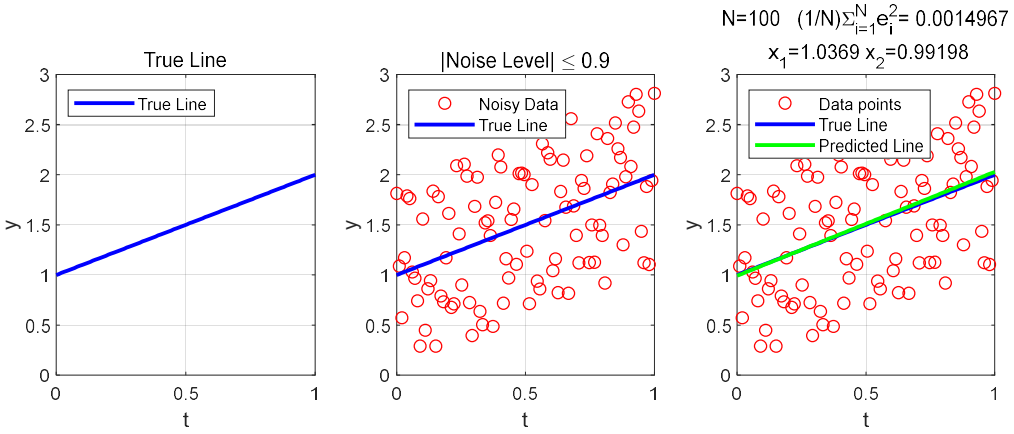


Figure 22: Results with N=100 data points and Noise Level = 0.9

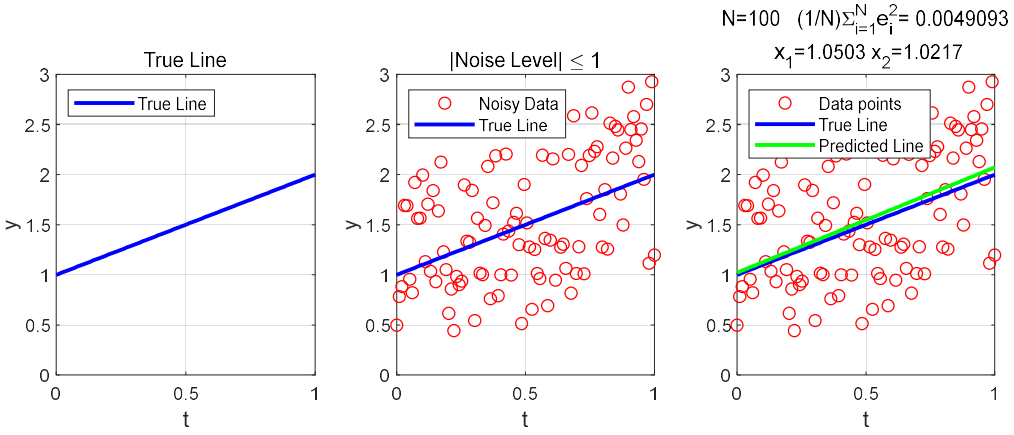


Figure 23: Results with N=100 data points and Noise Level = 1

```
clear all
close all
clc

Experiments = 100;
N=100;
NL = [0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1];
Earray=zeros(Experiments,length(NL));

rng(12345)

for experiment = 1:Experiments
    nl = 1;
    for NoiseLevel=NL;

        t=linspace(0,1,N)';
        n = NoiseLevel*(2*rand(N,1)-1);
        y = t+1;

        A=[t ones(size(t))];
        p=inv(A'*A)*A'*(y+n);
        ym = p(1)*t+p(2); % yhat
        E=(1/N)*((y-ym)'*(y-ym))^0.5;
        Earray(experiment,nl) = E;
        nl = nl + 1;
    end
end

figure
set(gcf,'Position',[181.8000 173 883.2000 367.2000])

subplot(1,2,1)
plot(NL,Earray','o')
xlabel('Noise Level')
ylabel('Mean Squared Error')
title('Each Experiment Shown')
hold on
plot(NL,min(Earray),'-k',NL,max(Earray),'-k')

subplot(1,2,2)
plot(NL,mean(Earray),'LineWidth',2)
xlabel('Noise Level')
ylabel('Mean Squared Error')
title('Experiments Averaged')
grid
```

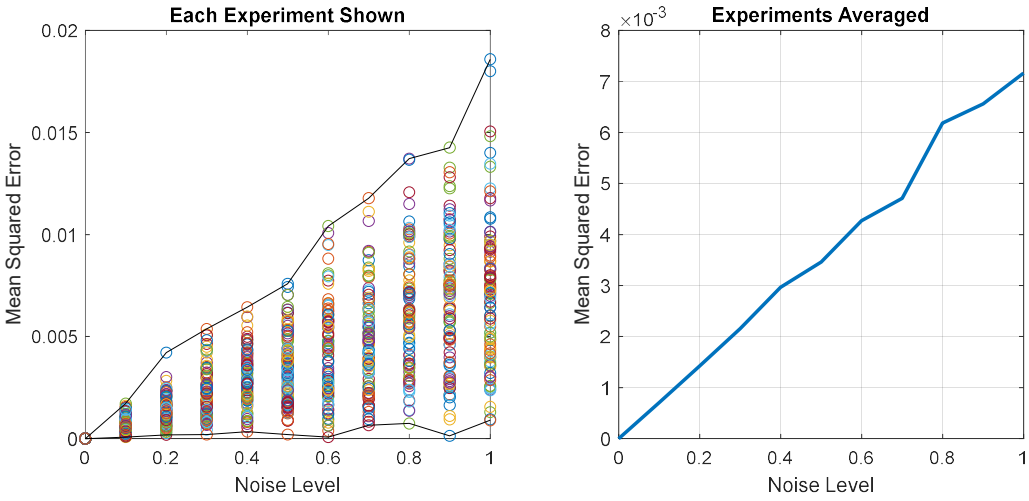


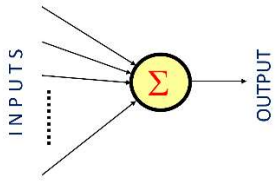
Figure 24: Change of the mean squared error w.r.t noise for N =100 observations.

According to the results seen above, repeating the experiment many times for each noise level creates the result seen on the left. **When the noise level is low, the predictions are good**. On the right, we illustrate the aveage of all circles seen on the left. **The average performance gets worse as the noise level gets larger**. **The only remedy is to increase the number of observations (N)!**

3.2 Example -2: $y=3t_1-4t_2$, $\hat{y}=x_1t_1+x_2t_2$, $x_1=3$, $x_2=-4$

In the second example, we will study the following process.

$$y = 3t_1 - 4t_2 \tag{3.4}$$



and the LS algorithm will consider the following model for fitting:

$$\hat{y} = x_1t_1 + x_2t_2 \tag{3.5}$$

The Matlab code for this is given below.

```
clear all
close all
clc

NoiseLevel = 0.1;
Earray = [];
Nvec = [5 20 100 200 500 1000];

figure(1)
set(gcf,'Position',[193.4000 177 775.2000 420])

for i=1:6
    N = Nvec(i);

    t1 = 2*rand(N,1)-1;
    t2 = 2*rand(N,1)-1;

    [T1,T2] = meshgrid(min(t1):(max(t1)-min(t1))/20:max(t1) , min(t2):(max(t2)-
min(t2))/20:max(t2));

    y = 3*t1 - 4*t2;
    yn = y + NoiseLevel*(2*rand(N,1)-1);
    Y = 3*T1 - 4*T2;

    A = [t1 t2];
    x = inv(A'*A)*A'*yn
    yhat = A*x;
    YHAT = x(1)*T1+x(2)*T2;

    subplot(2,3,i)
    ph1=surf(T1(1,:)',T2(:,1),Y);
    set(ph1,'facealpha',0.3)
    set(ph1,'edgealpha',0.3)
    hold on

    ph2=surf(T1(1,:)',T2(:,1),YHAT,'edgecolor','black','facecolor','yellow','LineWidth',1);
    set(ph2,'facealpha',0.3)
    set(ph2,'edgealpha',0.3)
    hold on

    xlabel('t_1')
    ylabel('t_2')
    zlabel('y')
```

```
    grid on
    axis tight

    hold on
    plot3(t1,t2,yn,'or','LineWidth',1)

    E = sum((y-yhat).^2)/N;
    Earray = [Earray;E];
    title(['N=' num2str(N) ' x_1=' num2str(x(1)) ' x_2=' num2str(x(2))],'FontWeight','normal')
end

figure
loglog(Nvec,Earray,'ok','LineWidth',2)
hold on
loglog(Nvec,Earray,'-b','LineWidth',2)
xlabel('N')
ylabel('Mean Squared Error')
grid
axis tight
title(['max(E)=' num2str(max(Earray)) '   min(E)='
num2str(min(Earray))],'FontWeight','normal')
```

In Figure 25, we show the results for different N values. **As the number of observations (N) increases, the performance gets better** as illustrated by Figure 26.

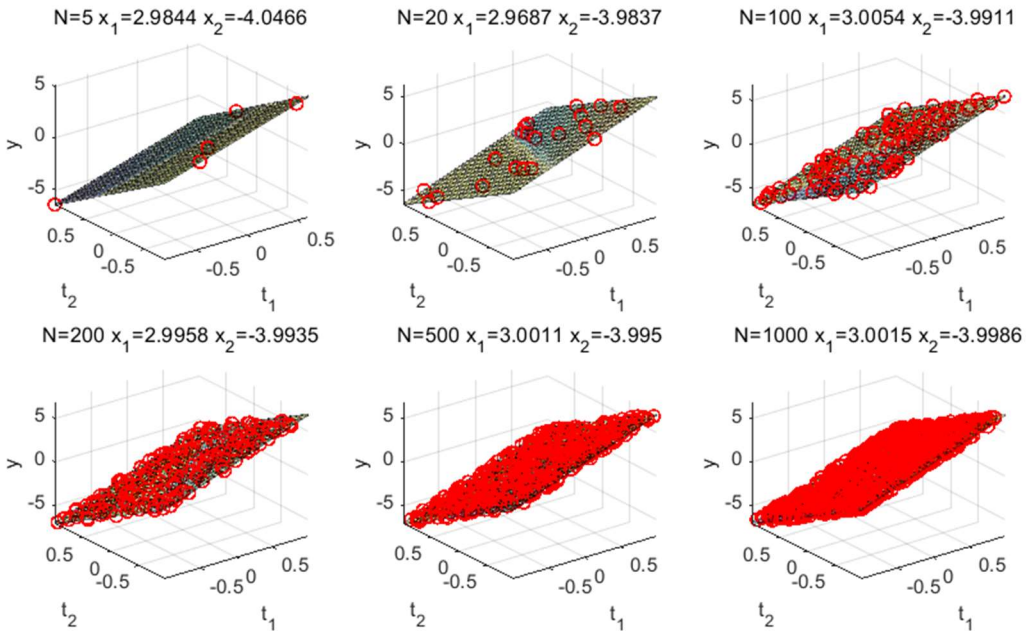


Figure 25: Finding the parameters of a plane equation that has two prameters

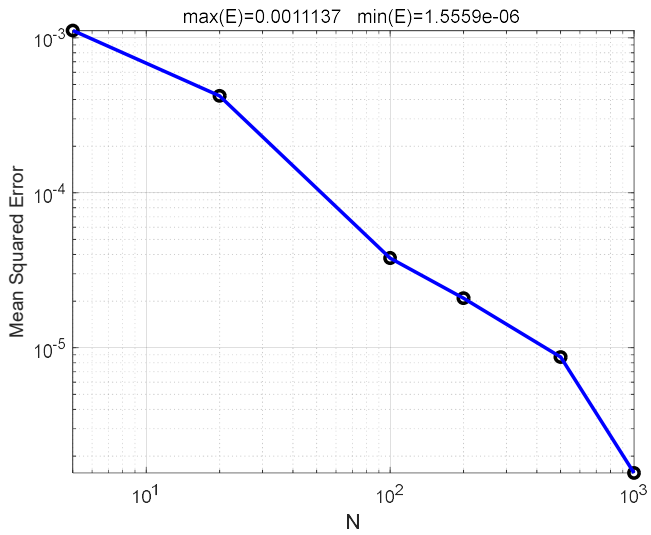


Figure 26: Change of the mean squared error with N. This is a sample case; the average of many experiments will display such a decreasing trend.

In our previous discussions, we wondered the structure of the model. In a realistic scenario, we have only the data and **the user needs to try different model structures to get a good fit. In Figure 27, we study this issue.**

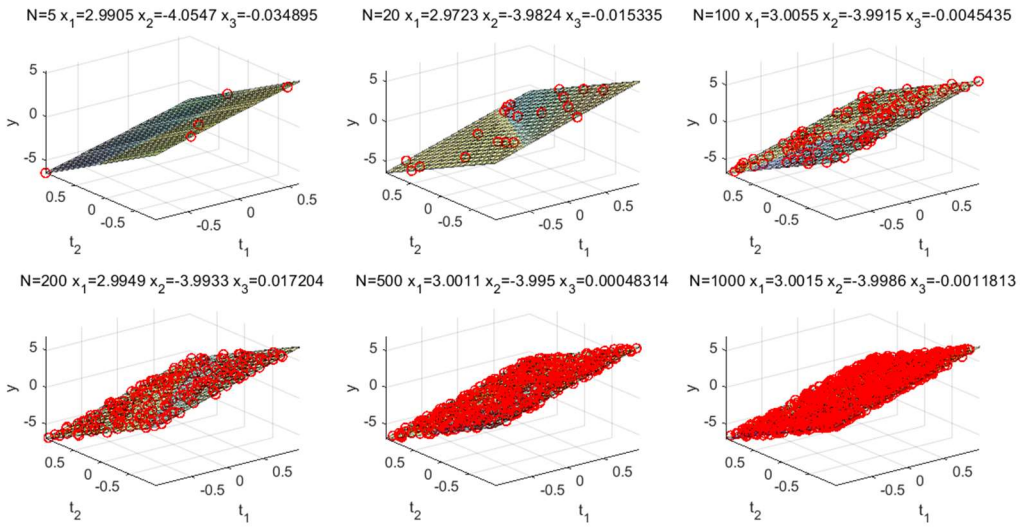


Figure 27: For a plane, we have 2 parameters (x_1 and x_2) but our model is: $y = x_1 t_1 + x_2 t_2 + x_3 t_1 t_2$.

As seen from Figure 27, the **coefficient x_3 is nearly zero in all cases**. The reason for this is very clear, x_3 is not relevant to the process that generates the data. Here model is $y = x_1 t_1 + x_2 t_2 + x_3 t_1 t_2$ and the process is $y = 3t_1 - 4t_2$.

The important conclusion to be drawn from this is the following: **We do not know what kind of process the data was produced by, but it is possible to choose an appropriate function type by visualizing the data**, and the best possible parameters of this function can be obtained with the LS algorithm.

According to the results seen in Figure 28, we understand that the MSE decreases as the number of data (N) increases.

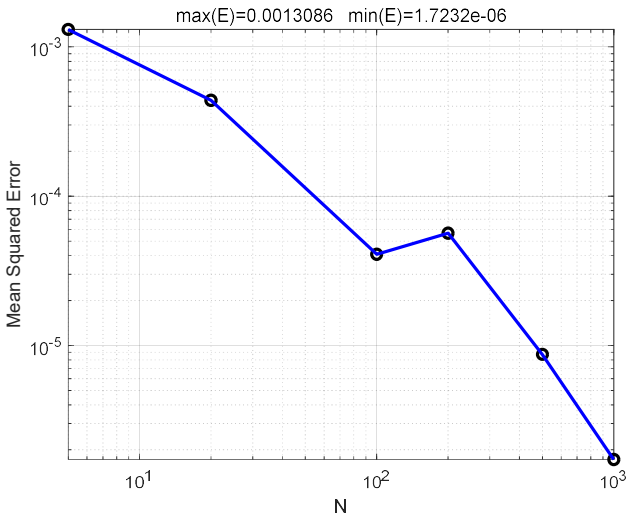


Figure 28: Change of the MSE for the model $y = x_1 t_1 + x_2 t_2 + x_3 t_1 t_2$ according to N.

3.3 Example -3: $y=t_1+2t_2-3t_3+4t_4-5t_5-6t_6$, $\hat{y}=\sum_{i=1}^{6} x_i t_i$, $x_1=1$, $x_2=2$, $x_3=-3$, $x_4=4$, $x_5=-5$, $x_6=-6$

In this example, we will consider a process having six parameters. If the number of unknows are known a priori, then we choose a model having 6 parameters, otherwise, we need to try different models to get the best fit.

The Matlab code for this example is as follows:

```
clear all
close all
clc

rng(12345);

Nvec= floor(logspace(1,5,20));
NoiseLevel = linspace(0.01,10,20);
Earray = [];

Experiment = 20;
for experiment = 1:Experiment

    Matrix = zeros(20,20);
    for i = 1:length(Nvec)
        N = Nvec(i);

        x1 = 2*rand(N,1)-1;
        x2 = 2*rand(N,1)-1;
        x3 = 2*rand(N,1)-1;
        x4 = 2*rand(N,1)-1;
        x5 = 2*rand(N,1)-1;
        x6 = 2*rand(N,1)-1;

        for j=1:length(NoiseLevel)
            NL = NoiseLevel(j);

            y = x1 + 2*x2 -3*x3 + 4*x4 -5*x5 -6*x6 + NL*(2*rand(N,1)-1);
            A = [x1 x2 x3 x4 x5 x6];

            params = inv(A'*A)*A'*y;
            yhat = A*params;

            E = sum((y-yhat).^2)/N;
```

```
            Earray(i,j) = E;
        end
    end
    Matrix = Matrix + Earray;
end
Matrix = Matrix/Experiment;

figure(1)
surf(NoiseLevel,log10(Nvec),log10(Matrix))
grid on
xlabel('Noise Level')
ylabel('log(N)')
zlabel('MSE')
axis tight
box on

figure(2)
loglog(NoiseLevel,Matrix')
```

In the bold line of the code above, we have the last term that is a random number varying in between +1 and -1 and multiplied by the variable NL (Noise Level). We will study the effect of NL on the performance.

When executed, the surface shown in Figure 29 is obtained. **We see that as the noise level increases, the performance gets worse. Increasing the number of data, we obtain smoother curves.**

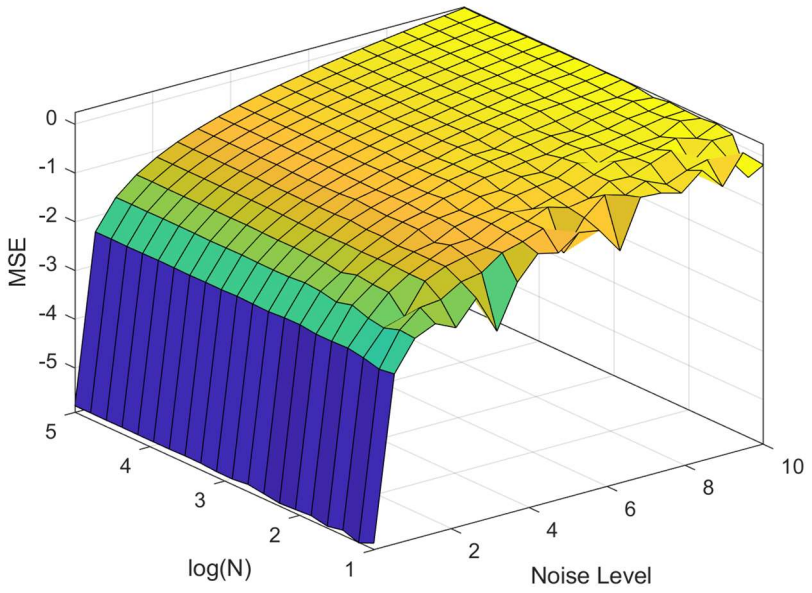


Figure 29: For the model $\hat{y} = \sum_{i=1}^{6} x_i t_i$ change of the mean squared error with respect to noise level and log(N)

3.4 Example -4: Effect of the Number of Data

Let's assume that we have m unknowns and N observations for a modeling problem. In this section, we will study the relation between N and m (if any). **What would happen if $N=m$, or $N>m$ or $N<m$?**

Let's assume that the **process that generates the data is a linear** process. In this particular problem, we will not add observation noise to see that the differences are not because of noise but because of another reason. The Matlab code is given below:

```
clear all
close all
clc

figno = 1;

% Below are the number of model parameters
for m = [25 40 100 200]

    % Below are the true values of the parameters
    TrueParameters = 10*rand(m,1)-5;

    figure(figno)
    plotno=1;

    for N = [m+20 m+10 m+5 m+1 m m-1 m-5 m-10 m-20]

        A = 2*rand(N,m)-1;
        y = A*TrueParameters;

        x = inv(A'*A)*A'*y;
        yhat = A*x;

        E = sum((y-yhat).^2)/N;

        subplot(3,3,plotno)
        plot([1:N]',A*TrueParameters,'-or',[1:N]',A*x,'-xb')
        legend('True Data','Prediction')
        title([num2str(plotno) ') #Parameter=' num2str(m) ' #Data=' num2str(N) ' E='
num2str(E)])
        axis tight
        box on
        plotno = plotno + 1;
    end
    figno = figno + 1;

end
```

Our model is as given below and the results are shown in Figure 30.

$$\hat{y} = \sum_{i=1}^{m} x_i t_i, \quad m = 25 \tag{3.6}$$

In this study, we selected the values N = [m+20 m+10 m+5 m+1 m m-1 m-5 m-10 m-20] one by one and drawn the results. According to this, the results for N = [m+20 m+10 m+5 m+1 m] are very good, and the mean squared error (E) is almost zero. We conclude that $N \geq m$ generates good results.

When $N<m$, i.e. N = [m-1 m-5 m-10 m-20], we see that the results get poorer as the difference m-N gets larger. In this case ($N<m$) the number of unknowns is larger than the number of obsevations (N), therefore LS algorithm finds one solution among infinitely many alternatives and the found solution is not satisfactory.

In Figure 30, the performance is good up to the 5th subplot, and after that, the performance gets worse.

In Figure 31, we repeat the same experiment for m=40 case, in Figure 32 m=100 case is studied and in Figure 33, m=200 case is considered. In all cases, the LS modeling performance is good till m=N, if m>N the model is poor.

In the shown figures, (mu) μ is a parameter, which we will consider next.

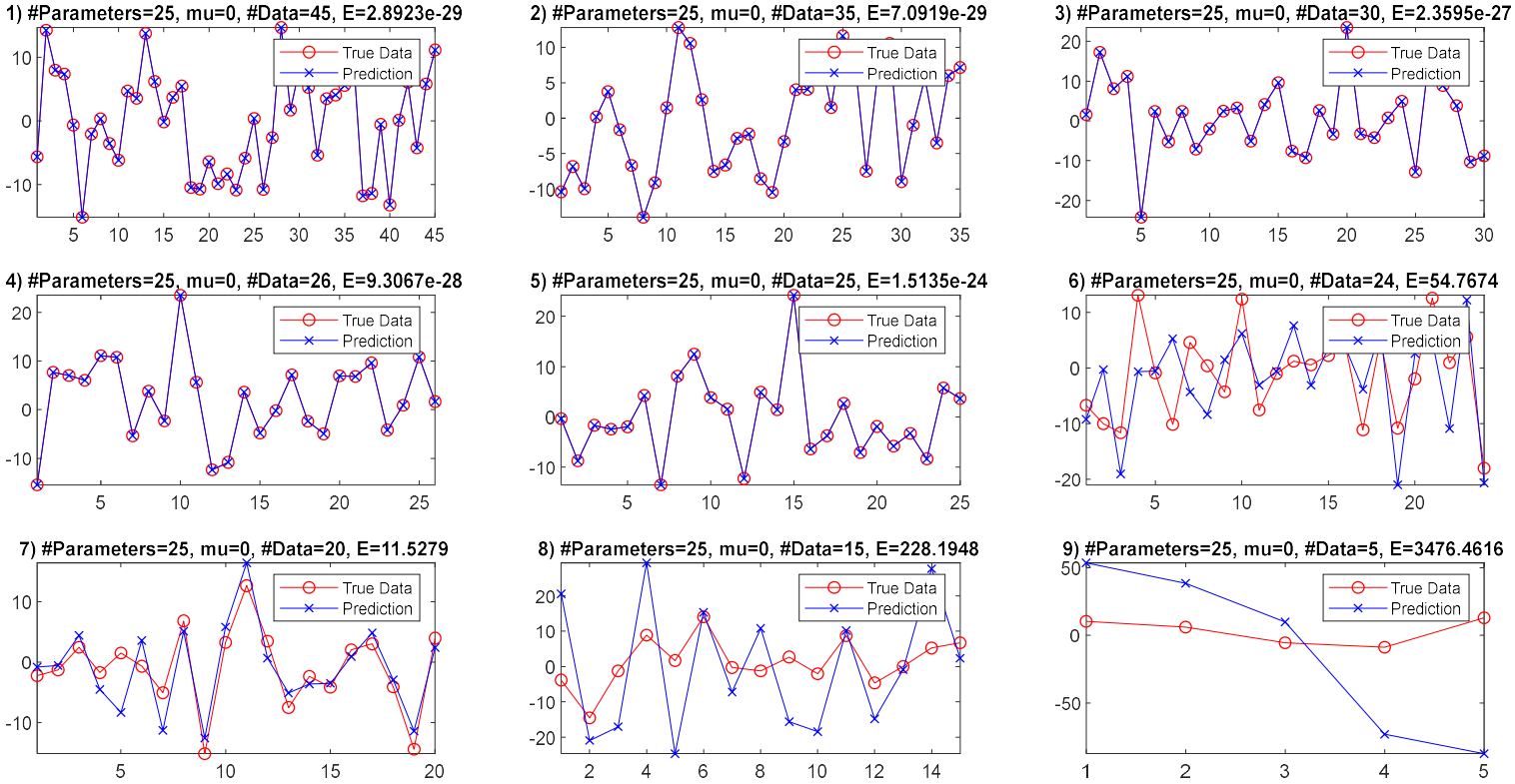


Figure 30: m=25, μ=0 and $\hat{y} = \sum_{i=1}^{m} x_i t_i$, we have N = [m+20 m+10 m+5 m+1 m m-1 m-5 m-10 m-20] observations

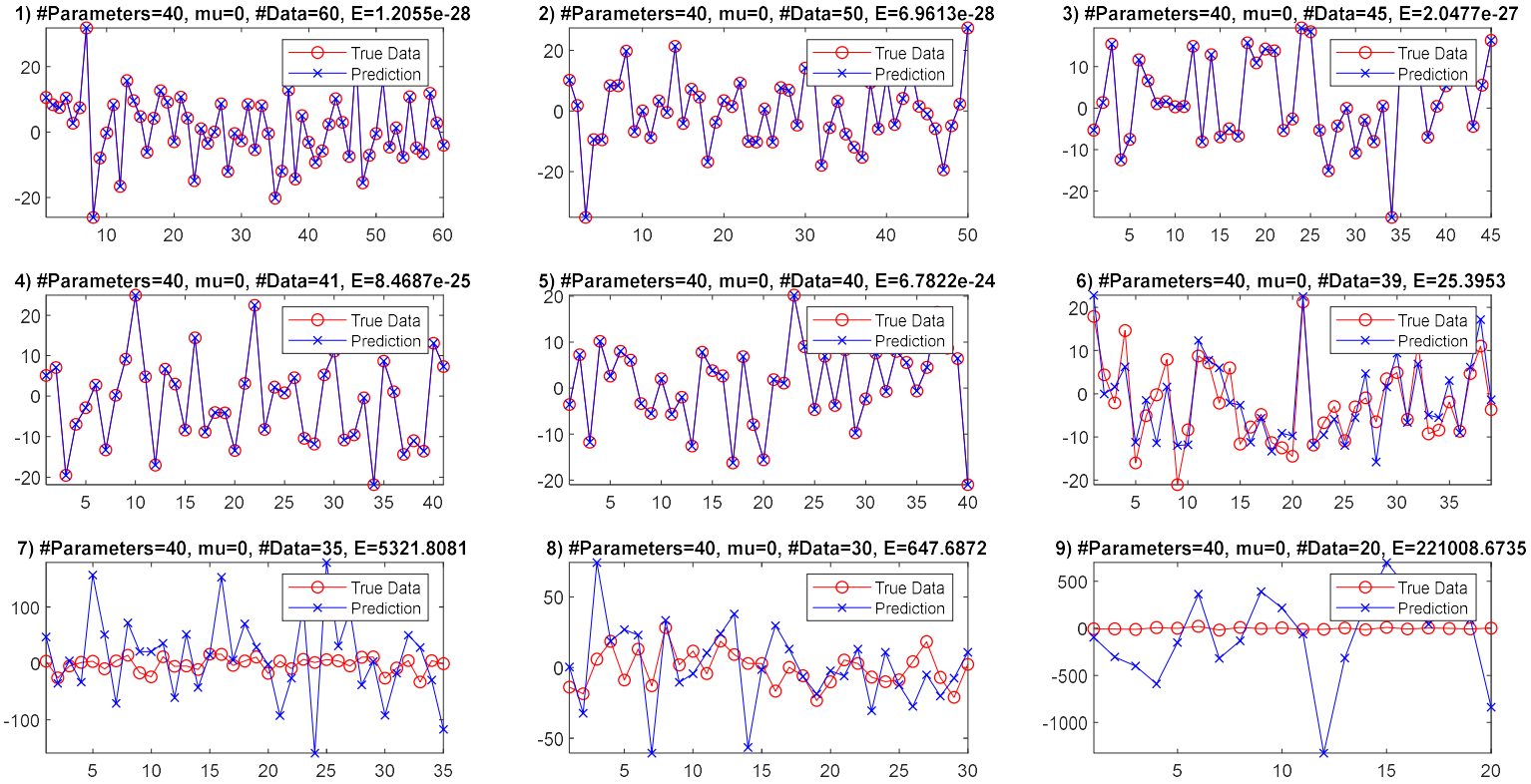


Figure 31: m=40, μ=0 and $\hat{y} = \sum_{i=1}^{m} x_i t_i$, we have N = [m+20 m+10 m+5 m+1 m m-1 m-5 m-10 m-20] observations

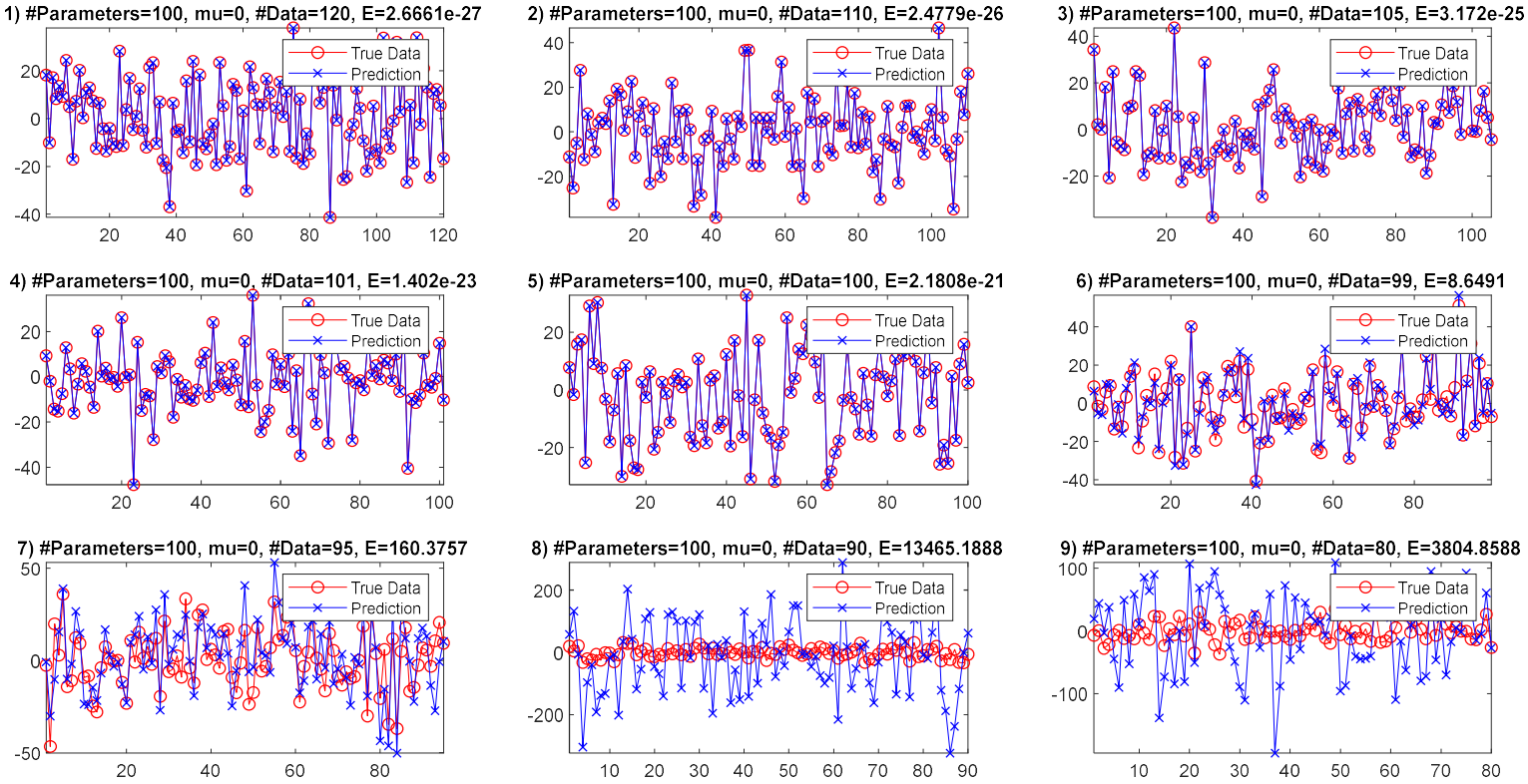


Figure 32: m=100, μ=0 and $\hat{y} = \sum_{i=1}^{m} x_i t_i$, we have N = [m+20 m+10 m+5 m+1 m m-1 m-5 m-10 m-20] observations

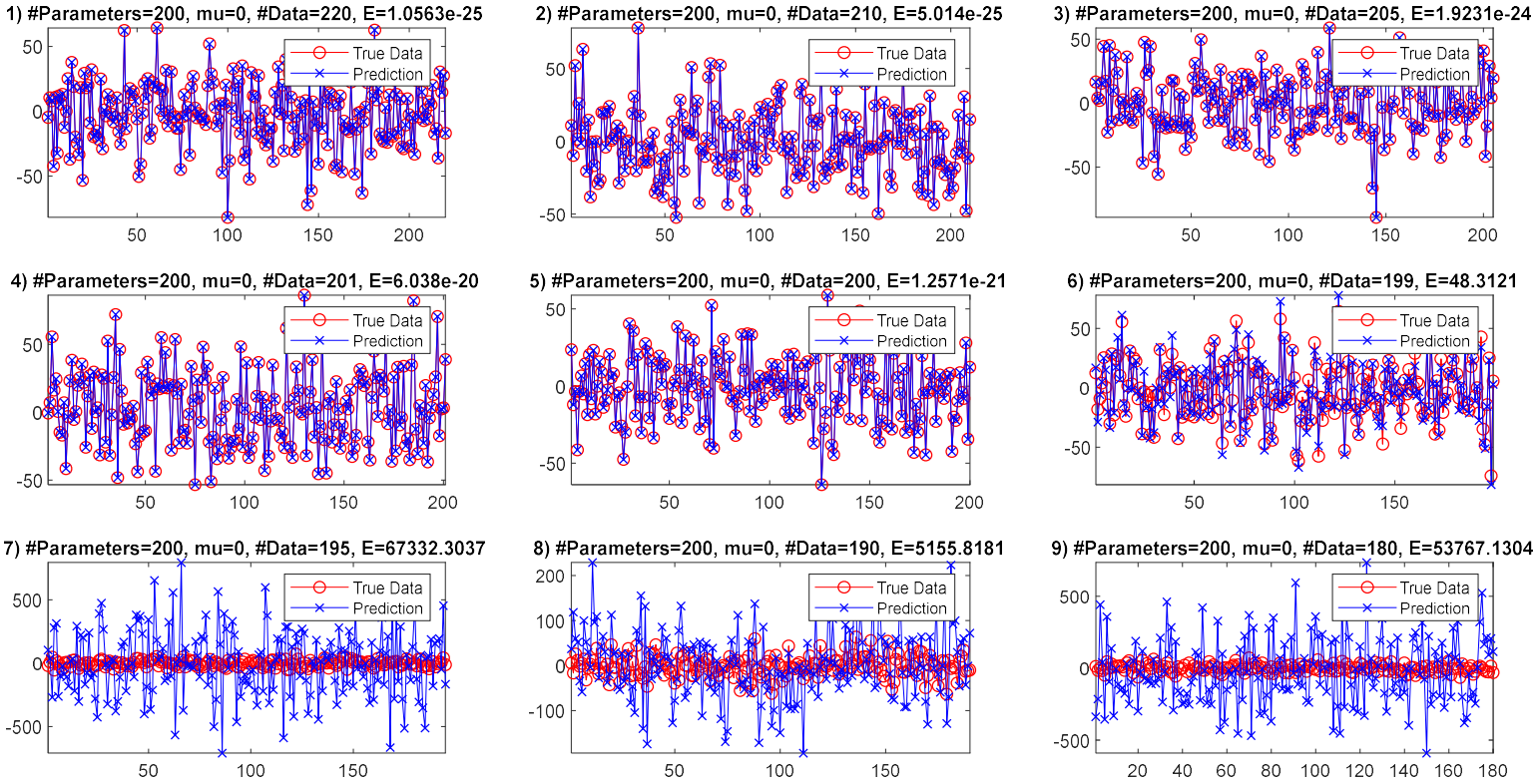
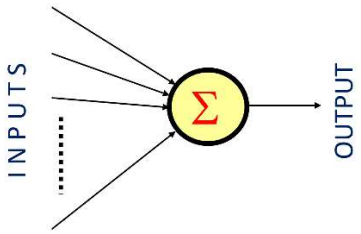


Figure 33: m=200, μ=0 and $\hat{y} = \sum_{i=1}^{m} x_i t_i$, we have N = [m+20 m+10 m+5 m+1 m m-1 m-5 m-10 m-20] observations

Now we will **modify the algorithm** slightly. Considering $\mu > 0$, we will compute the parameters using the equation $x = (\mu I + A^T A)^{-1} A^T b$. This law changes the cost function in (2.7) with that given below.

$$J = E^T E + \frac{\mu}{2} x^T x$$
$$= (b - Ax)^T (b - Ax) + \frac{\mu}{2} x^T x \qquad (3.7)$$



Such a cost function penalizes large x values and seeks a solution that has small norm. For this case, we will repeat our results.

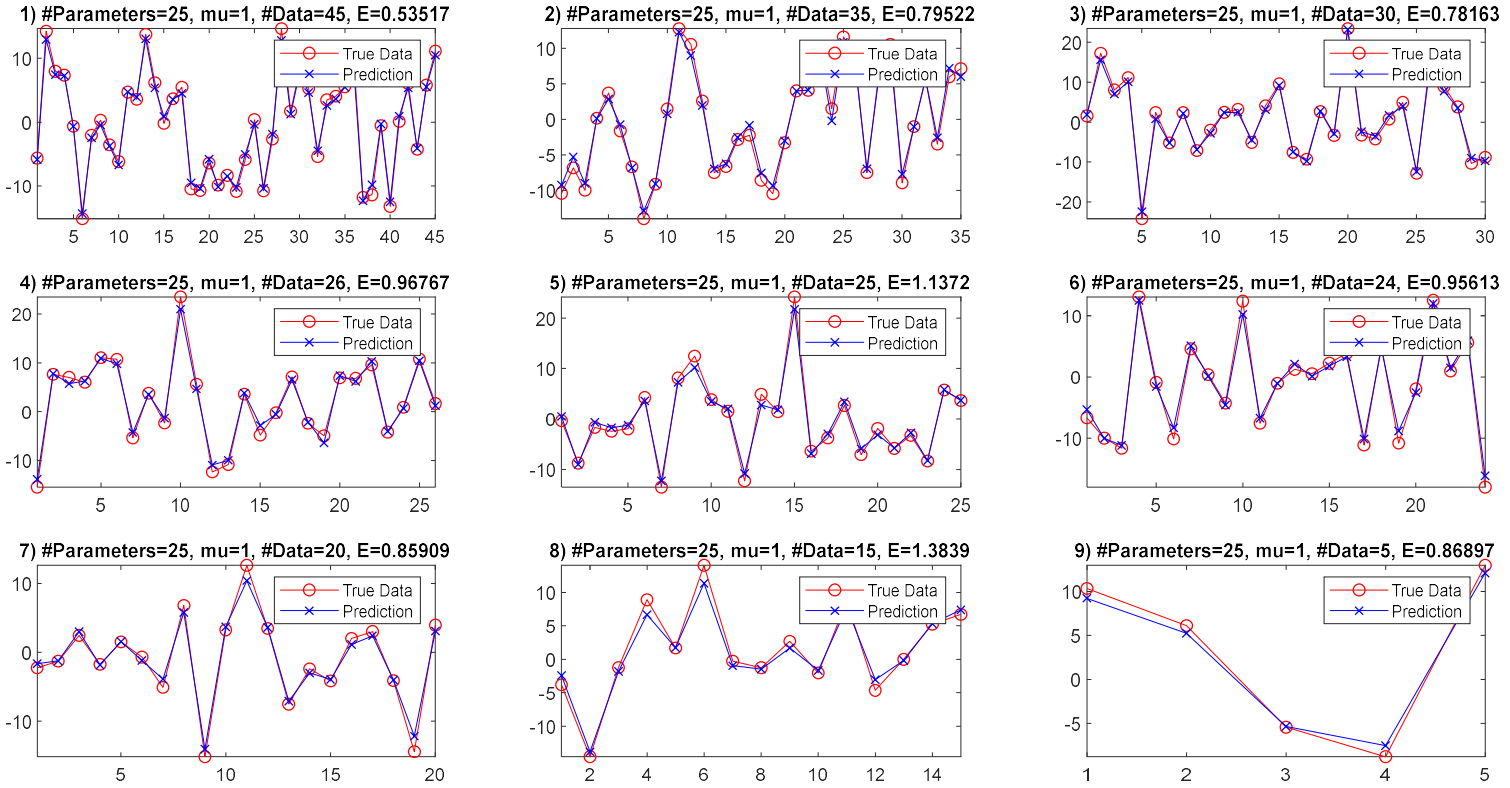


Figure 34: m=25, μ=1 and $\hat{y} = \sum_{i=1}^{m} x_i t_i$, we have N = [m+20 m+10 m+5 m+1 m m-1 m-5 m-10 m-20] observations

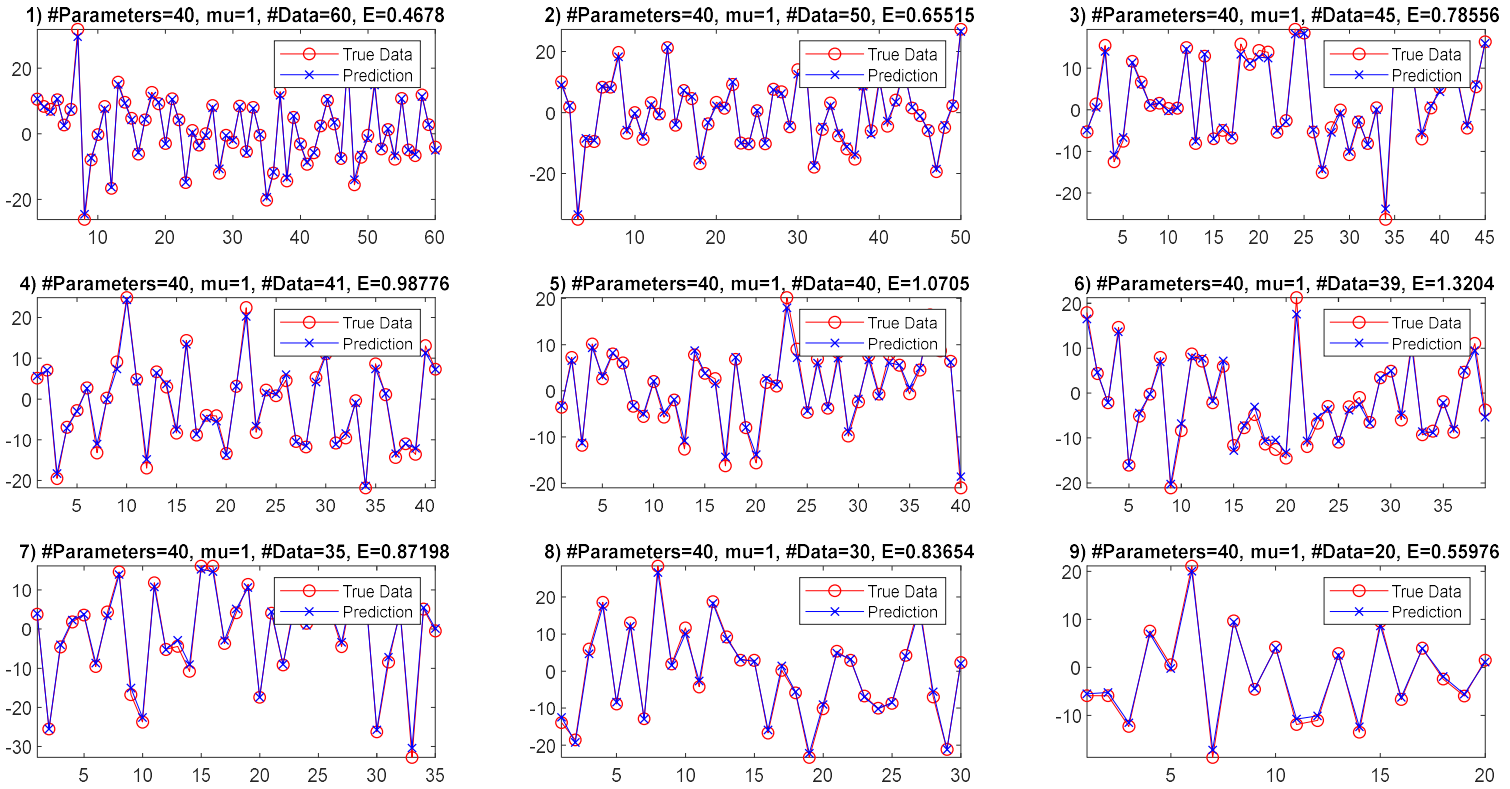


Figure 35: m=40, μ=1 and $\hat{y} = \sum_{i=1}^{m} x_i t_i$, we have N = [m+20 m+10 m+5 m+1 m m-1 m-5 m-10 m-20] observations

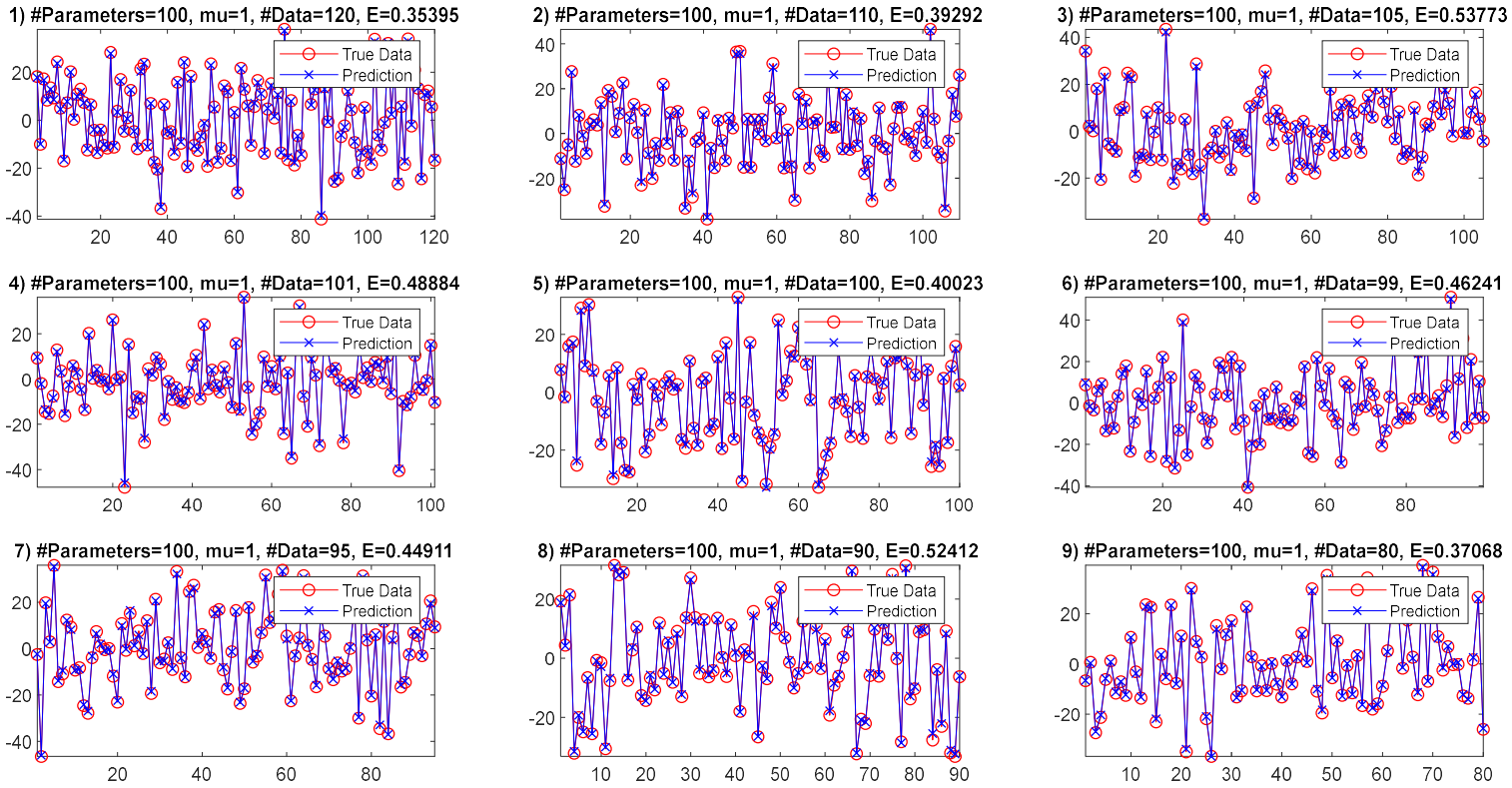


Figure 36: m=200, μ=1 and $\hat{y} = \sum_{i=1}^{m} x_i t_i$, we have N = [m+20 m+10 m+5 m+1 m m-1 m-5 m-10 m-20] observations

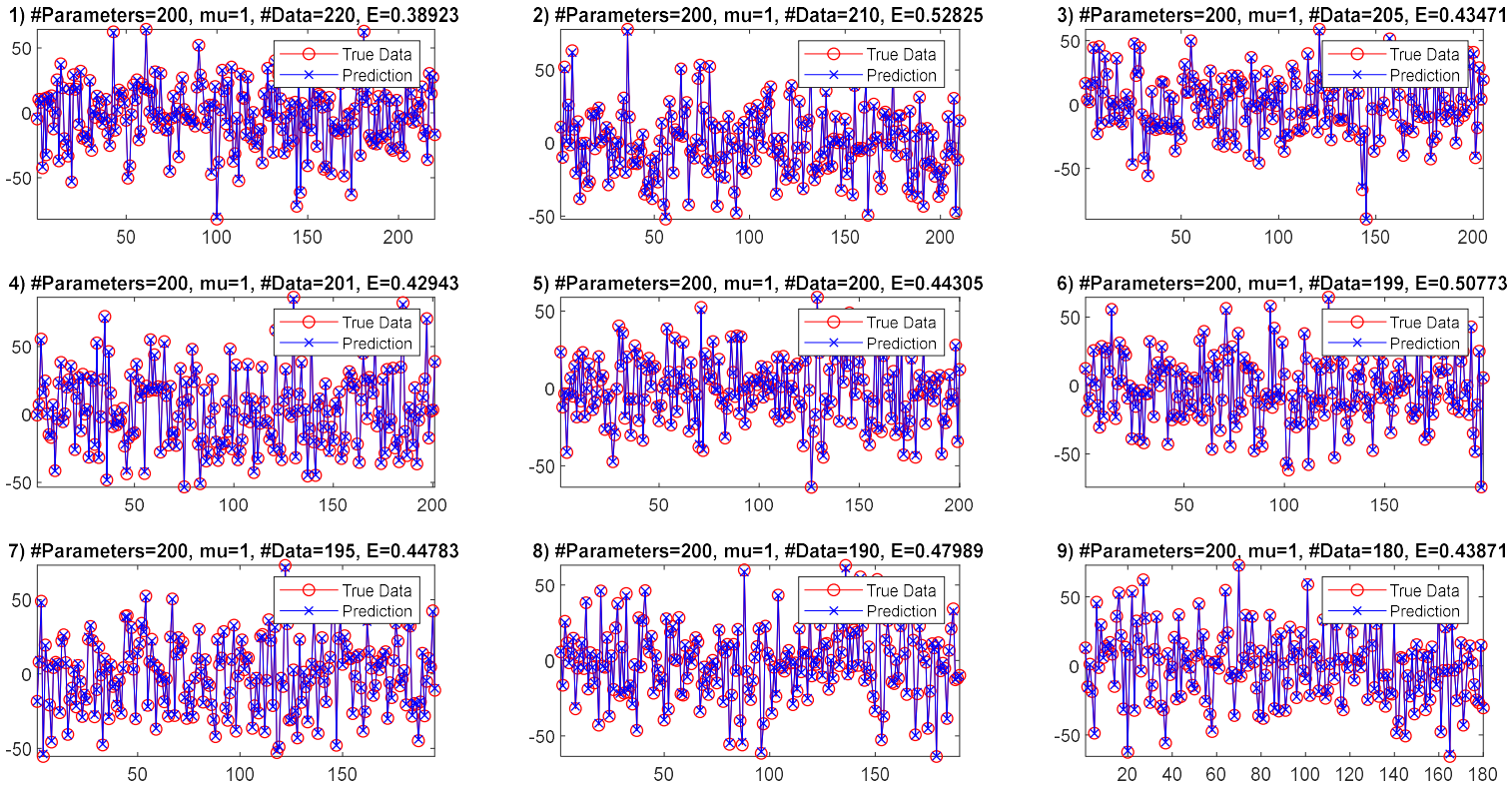


Figure 37: m=200, μ=1 and $\hat{y} = \sum_{i=1}^{m} x_i t_i$, we have N = [m+20 m+10 m+5 m+1 m m-1 m-5 m-10 m-20] observations

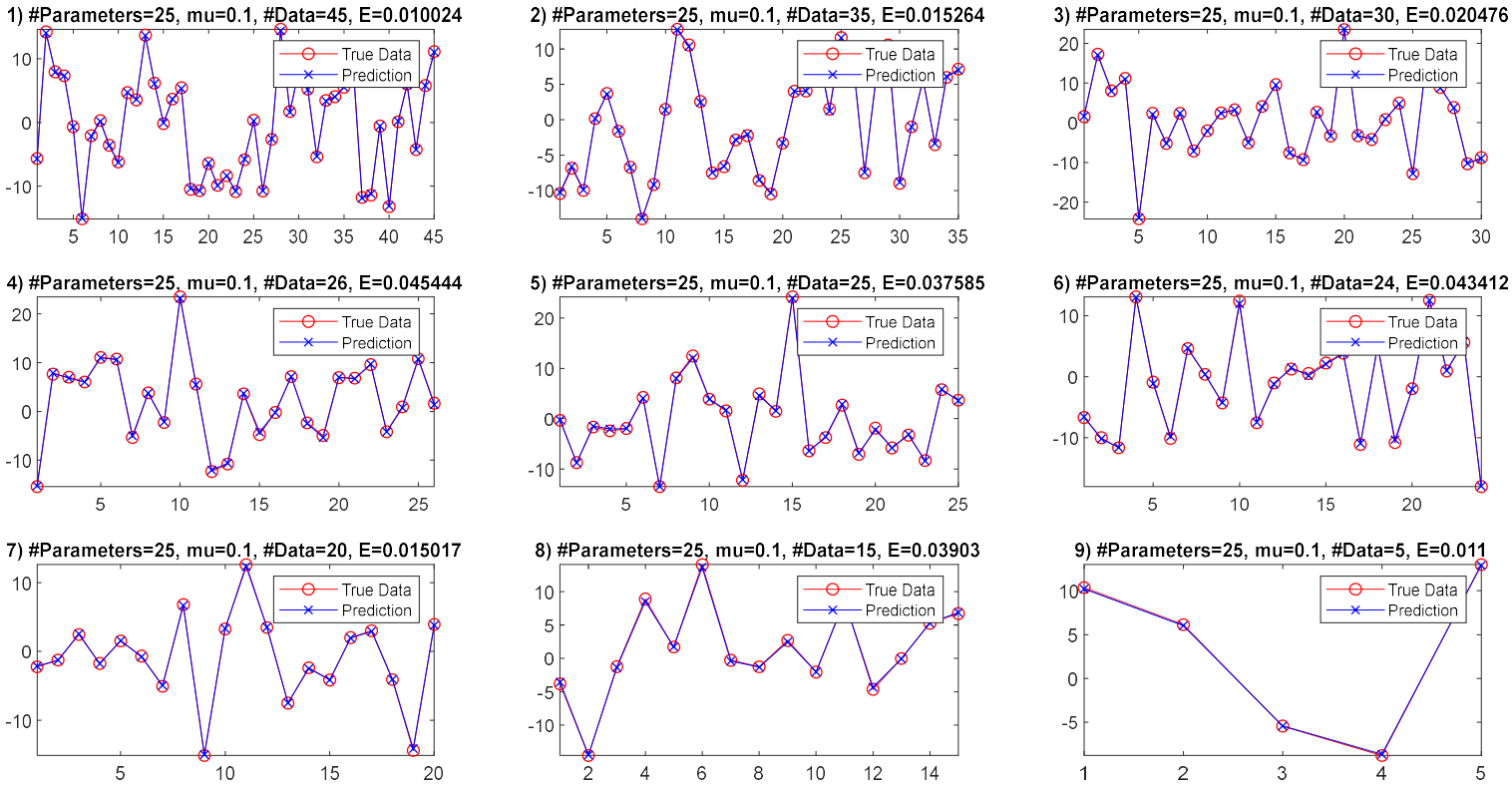


Figure 38: m=25, μ=0.1 and $\hat{y} = \sum_{i=1}^{m} x_i t_i$, we have N = [m+20 m+10 m+5 m+1 m m-1 m-5 m-10 m-20] observations

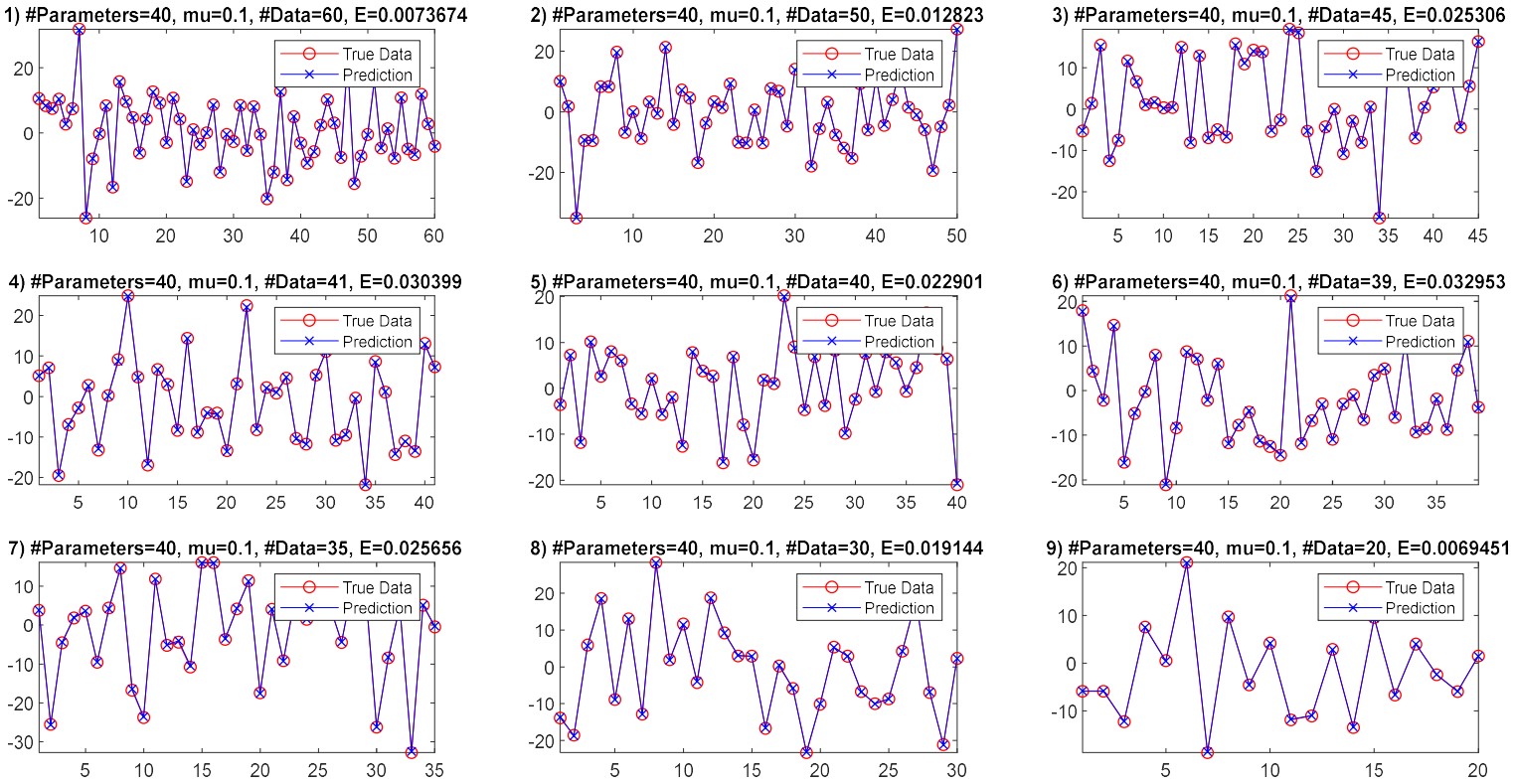


Figure 39: m=40, μ=0.1 and $\hat{y} = \sum_{i=1}^{m} x_i t_i$, we have N = [m+20 m+10 m+5 m+1 m m-1 m-5 m-10 m-20] observations

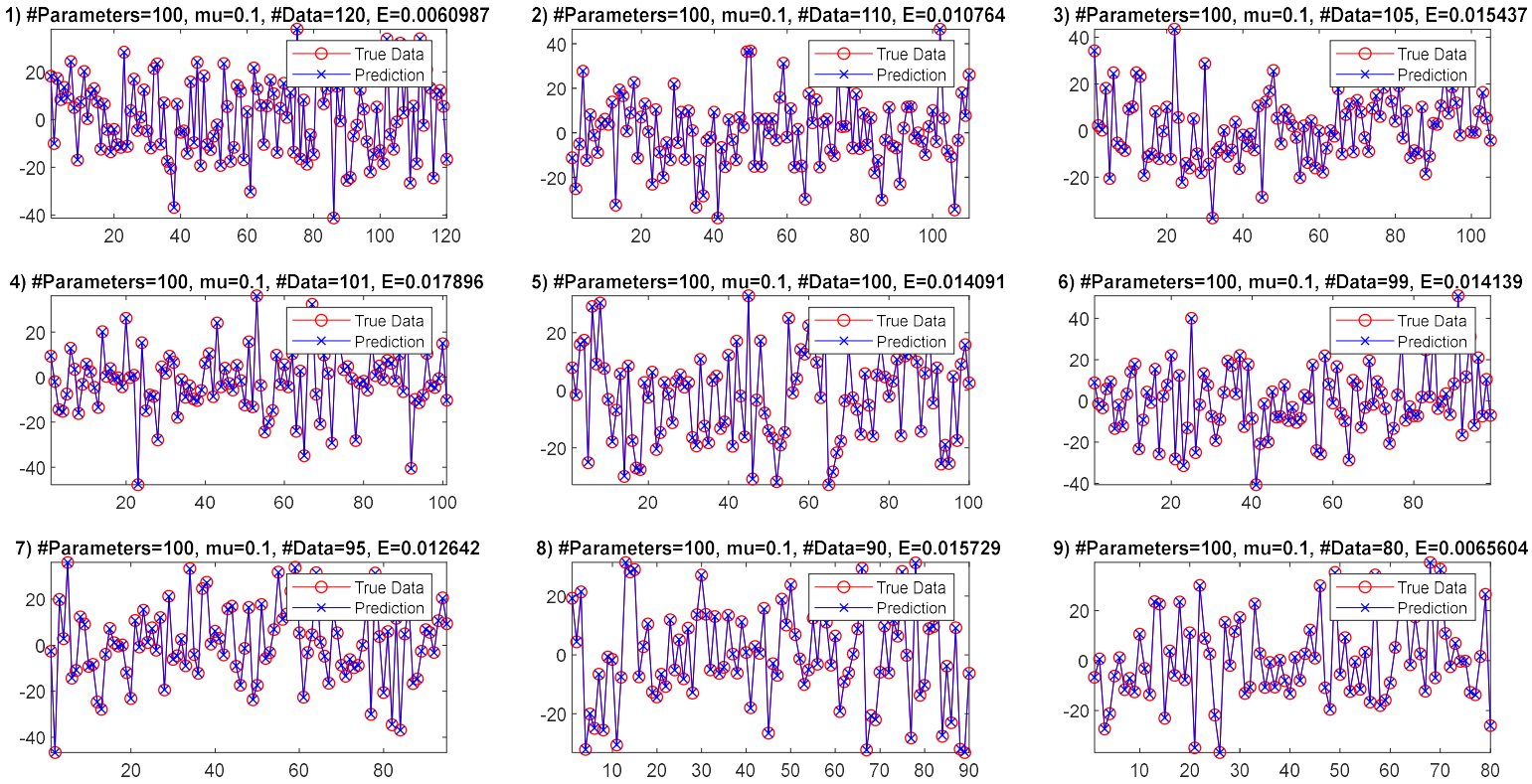


Figure 40: m=100, μ=0.1 and $\hat{y} = \sum_{i=1}^{m} x_i t_i$, we have N = [m+20 m+10 m+5 m+1 m m-1 m-5 m-10 m-20] observations

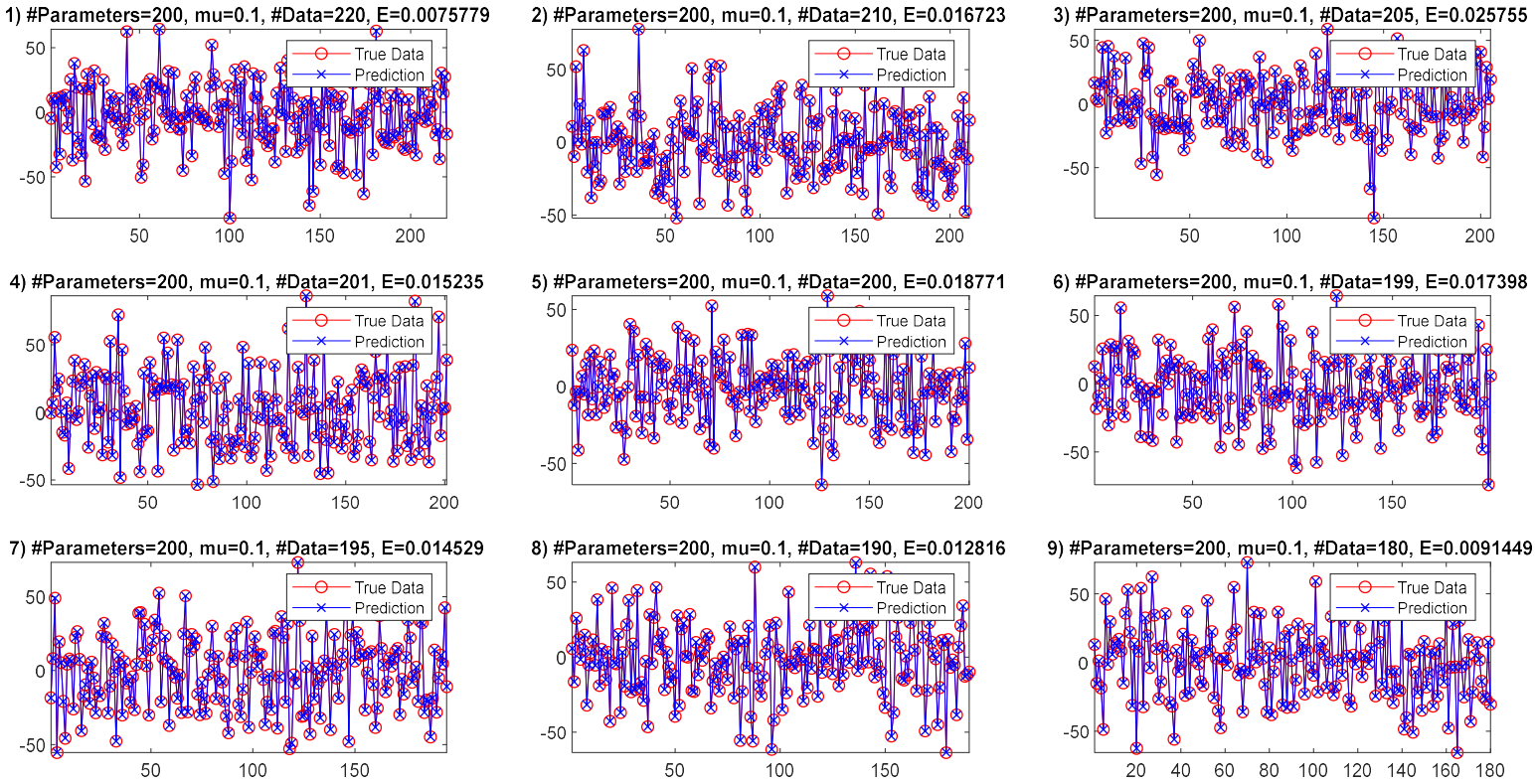


Figure 41: m=200, μ=0.1 and $\hat{y} = \sum_{i=1}^{m} x_i t_i$, we have N = [m+20 m+10 m+5 m+1 m m-1 m-5 m-10 m-20] observations

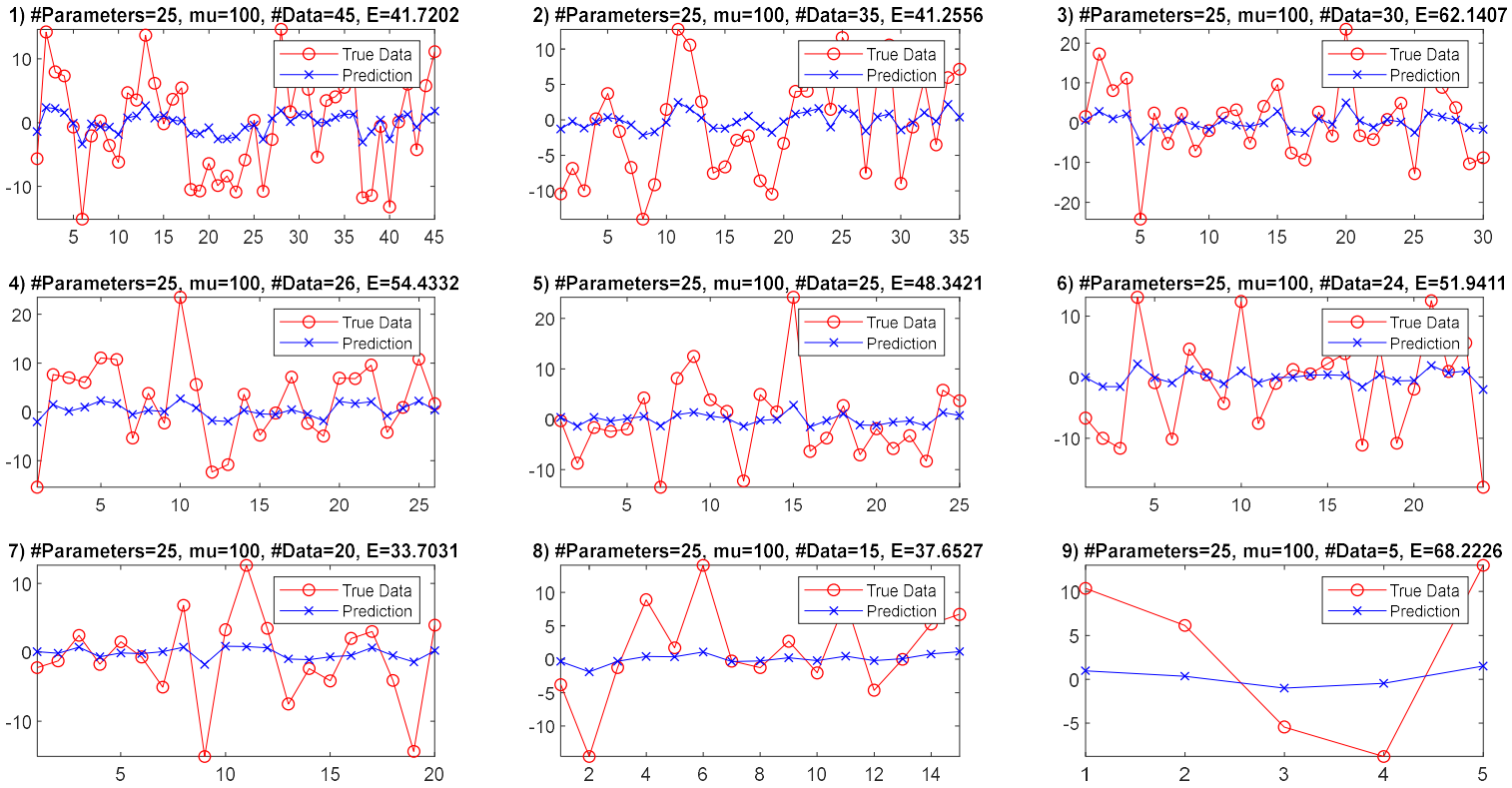


Figure 42: m=25, μ=100 and $\hat{y} = \sum_{i=1}^{m} x_i t_i$, we have N = [m+20 m+10 m+5 m+1 m m-1 m-5 m-10 m-20] observations

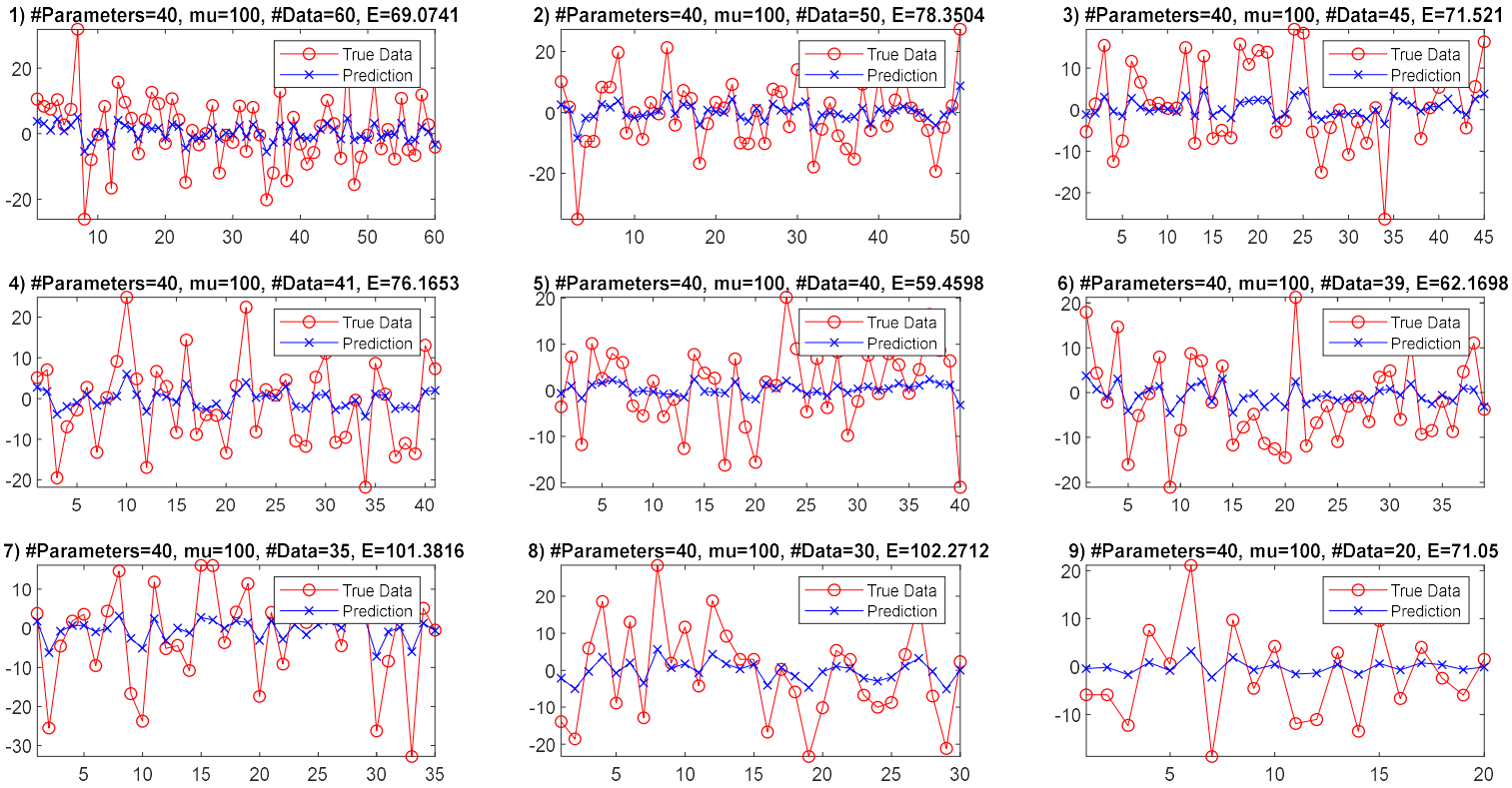


Figure 43: m=40, μ=100 and $\hat{y} = \sum_{i=1}^{m} x_i t_i$, we have N = [m+20 m+10 m+5 m+1 m m-1 m-5 m-10 m-20] observations

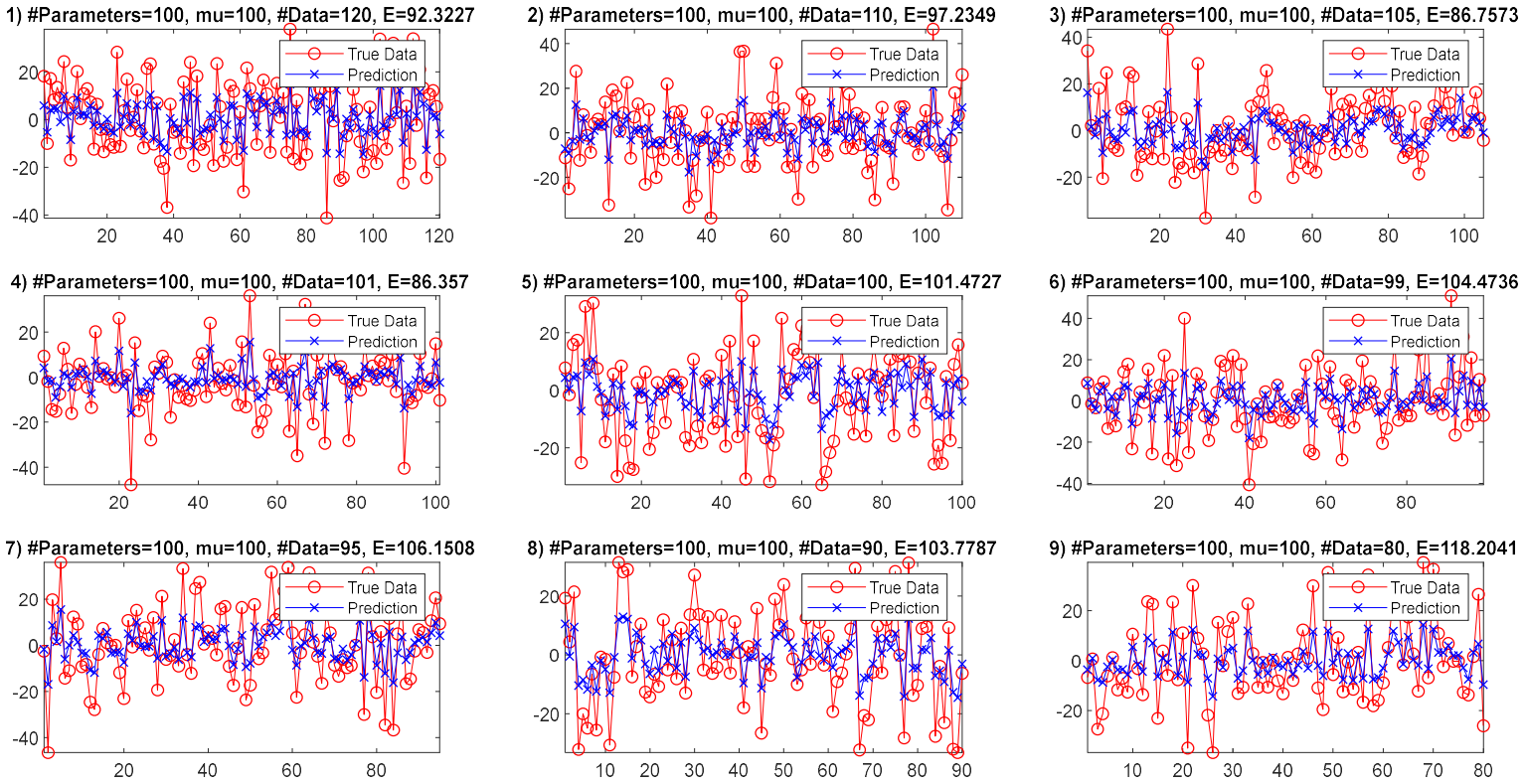


Figure 44: m=100, μ=100 and $\hat{y} = \sum_{i=1}^{m} x_i t_i$, we have N = [m+20 m+10 m+5 m+1 m m-1 m-5 m-10 m-20] observations

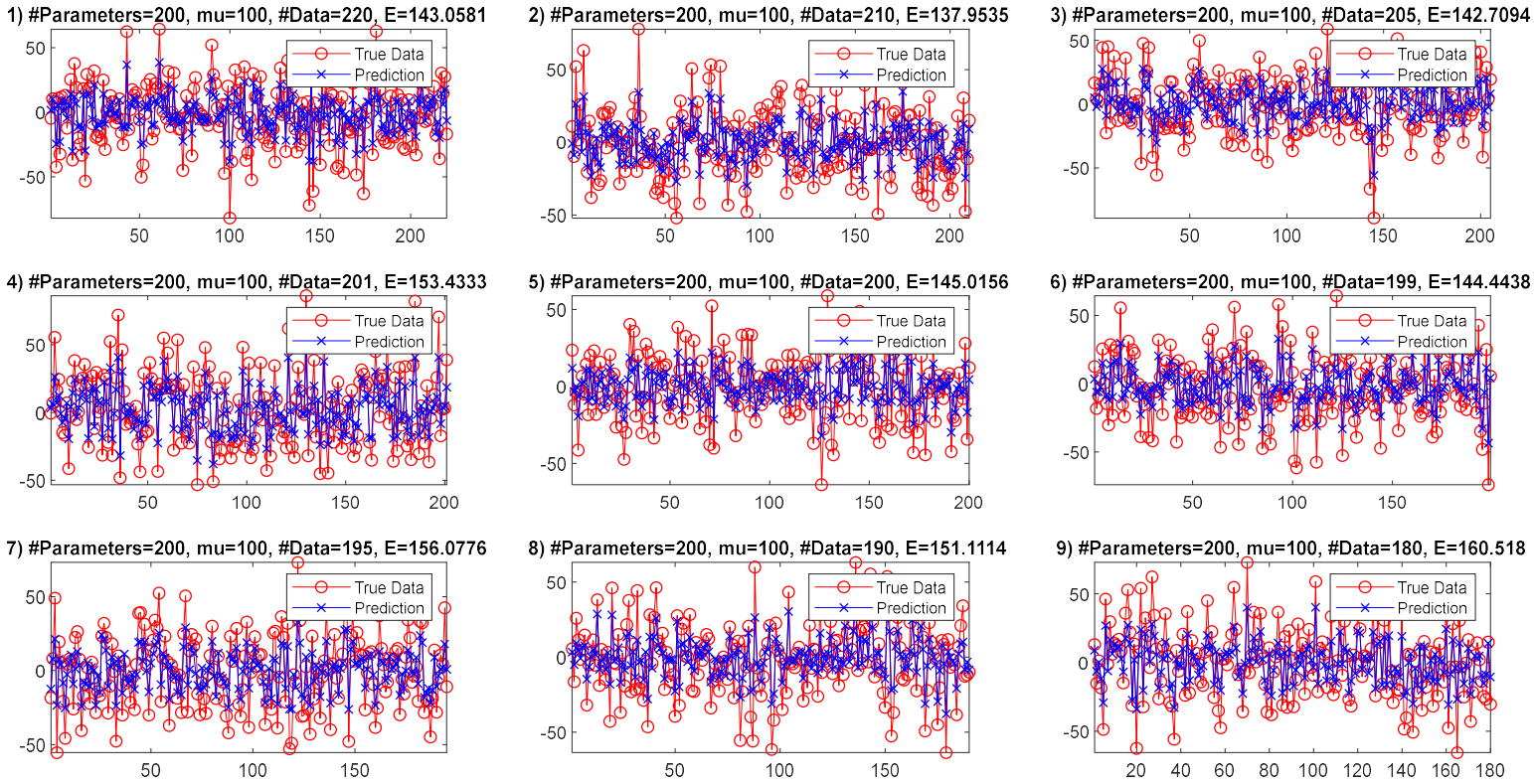


Figure 45: m=200, μ=100 and $\hat{y} = \sum_{i=1}^{m} x_i t_i$, we have N = [m+20 m+10 m+5 m+1 m m-1 m-5 m-10 m-20] observations

What would happen if the true parameter vector has a coefficient, which is far away from the others? Let us study this with the following code. We chose m=5 and N=m-1=4 lines of data are available.

```
clear
close all
clc

% Below are the true values of the parameters
TrueParameters = [2*rand(4,1)-1;100];

% Number of model parameters
m= length(TrueParameters);

% Regularization parameter
mu = 0.001;

N = m-1

A = 2*rand(N,m)-1;
y = A*TrueParameters;

x = inv(mu*eye(size(A'*A))+A'*A)*A'*y;
yhat = A*x;

E = sum((y-yhat).^2)/N;

plot([1:N]',A*TrueParameters,'-or',[1:N]',A*x,'-xb')
legend('True Data','Prediction')
title(['#Parameter=' num2str(m) ' #Data=' num2str(N) ' E=' num2str(E)])

[x TrueParameters]
```

mu=0.001 ans =		mu=0.1 ans =		mu=1 ans =	
3.5252	0.2680	3.3170	0.1673	-0.3465	-0.8494
-2.4731	-0.9560	20.8509	0.0015	5.5074	-0.7577
-1.3285	0.3377	26.8913	0.0592	-16.4907	-0.9052
3.1061	-0.1476	9.2395	-0.5362	16.1692	0.3472
99.3894	100.0000	77.2162	100.0000	51.4184	100.0000

The results above indicate that the 5th parameter is different in magnitude. If the true parameters do not resemble each other, the results are not similar although y is similar to yhat. What could be a remedy to this?

$$J = (b - Ax)^T (b - Ax) + \frac{1}{2} x^T \underbrace{\text{diag}(\mu, \mu, \cdots, \mu, 0.01\mu)}_{Q} x \qquad (3.xxx)$$

$$x = \left(Q^T Q + A^T A\right)^{-1} A^T b \qquad (3.xxx)$$

We will update the code as follows:

```
clear
close all
clc

% Below are the true values of the parameters
TrueParameters = [2*rand(4,1)-1;100];

% Number of model parameters
m= length(TrueParameters);

% Regularization parameter
mu = 0.001;

% Form matrix Q
Q = blkdiag(diag(mu*ones(1,m-1)),0.01*mu);

N = m-1;

A = 2*rand(N,m)-1;
y = A*TrueParameters;

x = inv(Q'*Q+A'*A)*A'*y;
yhat = A*x;

E = sum((y-yhat).^2)/N;

plot([1:N]',A*TrueParameters,'-or',[1:N]',A*x,'-xb')
legend('True Data','Prediction')
title(['#Parameter=' num2str(m) ' #Data=' num2str(N) ' E=' num2str(E)])
```

[x TrueParameters]

mu=0.001 ans =		mu=0.1 ans =		mu=1 ans =	
-0.5411	-0.6298	-0.4623	-0.7397	0.1605	0.6870
0.1014	0.1116	0.4675	0.5765	-0.0234	-0.1487
-0.6263	-0.5792	-0.2646	0.5195	0.0405	0.5342
0.3049	0.2412	0.6873	0.9153	-0.2504	-0.5813
100.0114	100.0000	100.5776	100.0000	99.9745	100.0000

3.5. Kernelized Ridge Regression

Given the cost function $J = E^T E + \frac{\mu}{2} x^T x = (b - Ax)^T (b - Ax) + \frac{\mu}{2} x^T x$, $x = (\mu I_{m\times m} + A^T A)^{-1} A^T b$ solves the problem. For this, we need to invert an $m \times m$ matrix if m >> N, the complexity is $O(m^3)$. The representer theorem tells us that x should be in the span of data, i.e. $x = \sum_i \alpha_i a_i^T$, where a_i is the i-th column of A. Using Woodbury matrix identity we obtain $x = A^T (\mu I_{N\times N} + AA^T)^{-1} b$. The $N \times N$ matrix AA^T is called Gram matrix and costs $O(N^2 m)$ to compute. This shows us that depending on the number of data and number of unknownsi one can follow a numerically efficient form of the algorithm.

Finally, in this section, when $m=N$, i.e. the number of parameters is equal to the number of observations will be considered. Let's consider a simple example.

$$\hat{y} = x_0 + x_1 t \tag{3.8}$$

Consider we have two observations, i.e.

$$\left.\begin{aligned}\hat{y}_1 &= x_0 + x_1 t_1 \\ \hat{y}_2 &= x_0 + x_1 t_2\end{aligned}\right\} \Rightarrow \begin{bmatrix}\hat{y}_1 \\ \hat{y}_2\end{bmatrix} = \underbrace{\begin{bmatrix}t_1 & 1 \\ t_2 & 1\end{bmatrix}}_{A} \begin{bmatrix}x_1 \\ x_0\end{bmatrix} \tag{3.9}$$

Here A is a 2×2 matrix. In order to find the parameter vector $\begin{bmatrix}x_1 \\ x_0\end{bmatrix}$ we can perform the following operation.

$$\begin{bmatrix}x_1 \\ x_0\end{bmatrix} == \underbrace{\begin{bmatrix}t_1 & 1 \\ t_2 & 1\end{bmatrix}^{-1}}_{A^{-1}} \begin{bmatrix}\hat{y}_1 \\ \hat{y}_2\end{bmatrix} \tag{3.10}$$

Now our question is the following:

$$\left(A^T A\right)^{-1} A^T \overset{?}{=} A^{-1} \tag{3.11}$$

Obviously if we multiply the both sides of this equality by A from the right, we see that the equation holds true. It is of course possible to perform the operations as shown below:

$$A^{-1} = \frac{1}{t_1 - t_2}\begin{bmatrix}1 & -1 \\ -t_2 & t_1\end{bmatrix} \tag{3.12}$$

$$A^T A = \begin{bmatrix}t_1 & t_2 \\ 1 & 1\end{bmatrix}\begin{bmatrix}t_1 & 1 \\ t_2 & 1\end{bmatrix} = \begin{bmatrix}t_1^2 + t_2^2 & t_1 + t_2 \\ t_1 + t_2 & 2\end{bmatrix} \tag{3.13}$$

$$\left(A^T A\right)^{-1} = \begin{bmatrix}t_1^2 + t_2^2 & t_1 + t_2 \\ t_1 + t_2 & 2\end{bmatrix}^{-1} = \frac{1}{(t_1 - t_2)^2}\begin{bmatrix}2 & -(t_1 + t_2) \\ -(t_1 + t_2) & t_1^2 + t_2^2\end{bmatrix} \tag{3.14}$$

$$\left(A^T A\right)^{-1} A^T = \frac{1}{(t_1 - t_2)^2}\begin{bmatrix}2 & -(t_1 + t_2) \\ -(t_1 + t_2) & t_1^2 + t_2^2\end{bmatrix}\begin{bmatrix}t_1 & t_2 \\ 1 & 1\end{bmatrix} \tag{3.15}$$

$$= \frac{1}{(t_1 - t_2)^2}\begin{bmatrix}t_1 - t_2 & t_2 - t_1 \\ t_2(t_2 - t_1) & t_1(t_1 - t_2)\end{bmatrix} \tag{3.16}$$

$$= \frac{1}{t_1 - t_2}\begin{bmatrix}1 & -1 \\ -t_2 & t_1\end{bmatrix} = A^{-1} \tag{3.17}$$

3.6 Example -5: Modeling of a Time Series Whose Paramaters are Unknown

In the fifth example of this report, we will consider the data shown in Figure 46, where the horizontal axis is time and the vertical axis is the output variable. **We do not have any prior information about the process generating this data and we will choose several alternatives**.

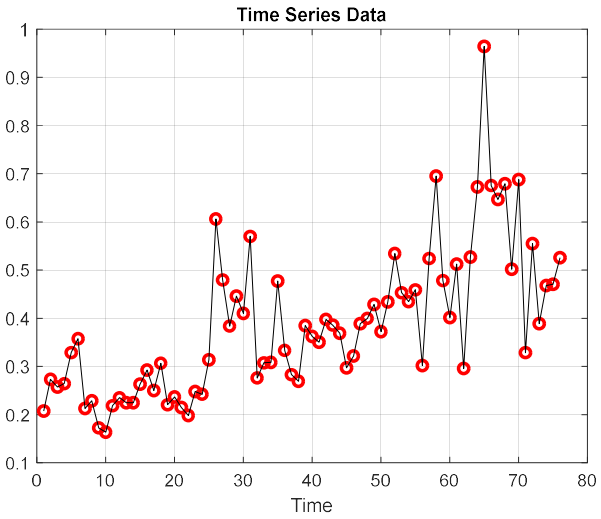


Figure 46: Appearance of raw data

We will choose polynomial models to fit the data seen in Figure 46, i.e., we choose

$$\hat{y} = \sum_{i=0}^{m} x_i t^i = x_0 + x_1 t + x_2 t^2 + \cdots + x_m t^m \tag{3.18}$$

This **model generates a line when m=1, a parabola when m=2 an a cubic curve when m=3 ans so on**. We can choose the model order m and obtain the model results. The Matlab code for this example is given below.

In Figure 47, the order of the polynomial will start from m=1 and increased till m=12. Looking closely, we see that the model catches the localities as the model order increases.

In Figure 47, the last subplot if far from being a representative of the process. Therefore, we cannot deduce that the model gets better as m increases. In obtaining the model, very large and very small numbers become effective on the precision of the matrix operations. Here the values are in between zero and one, and we take their 12^{th} power, which produces very small numbers and this deteriorates the numericak accuracy as we take a matrix inverse during the determination of model parameters.

What happens if none of these models is appropriate? It is of course possible choose more complicated models; another popular alternative is the use of artificial neural networks and benefit from their layered structure

```
clear all
close all
clc

y = [4565.03
6010.59                         10548.49                        11755.96
5660.73                         8442.69                         9967.28
5814.19                         9813.15                         9557.54
7232.74                         9024.62                         10099.29
7870.31                         12540.00                        6641.86
4686.09                         6077.11                         11528.72
5032.73                         6765.97                         15289.86
3795.78                         6782.71                         10521.42
3597.80                         10497.52                        8833.55
4805.98                         7333.86                         11269.22
5173.26                         6225.12                         6509.10
4940.37                         5927.25                         11590.60
4943.27                         8463.25                         14796.97
5792.62                         7968.48                         21214.20
6433.21                         7704.99                         14860.97
5504.29                         8750.48                         14223.63
6749.25                         8486.84                         14943.50
4850.15                         8116.99                         11042.08
5211.54                         6540.24                         15133.43
4729.84                         7079.27                         7230.17
4359.88                         8551.61                         12208.96
5452.36                         8795.46                         8557.84
5330.17                         9430.46                         10289.97
6904.68                         8186.23                         10354.70
13336.80                        9546.85                         11570.22]/2.2e4;

N = length(y);

t = linspace(0,1,N)';
A = [ones(N,1)];
Earray = [];

figure(1)
for order = 1:12
    subplot(3,4,order)
    A = [t.^order A];
    x = inv(A'*A)*A'*y;
    yhat = A*x;

    E = sum((y-yhat).^2)/N;
    Earray = [Earray;E];
    plot(t,y,'-o')
    hold on
    plot(t,yhat,'-','LineWidth',2)
    title([num2str(order) '. Degree Polynomial, E=' num2str(E)],'FontWeight','normal')
    xlabel('t','FontWeight','normal')
end

xlabel('Time')
ylabel('y and ym')
grid
```

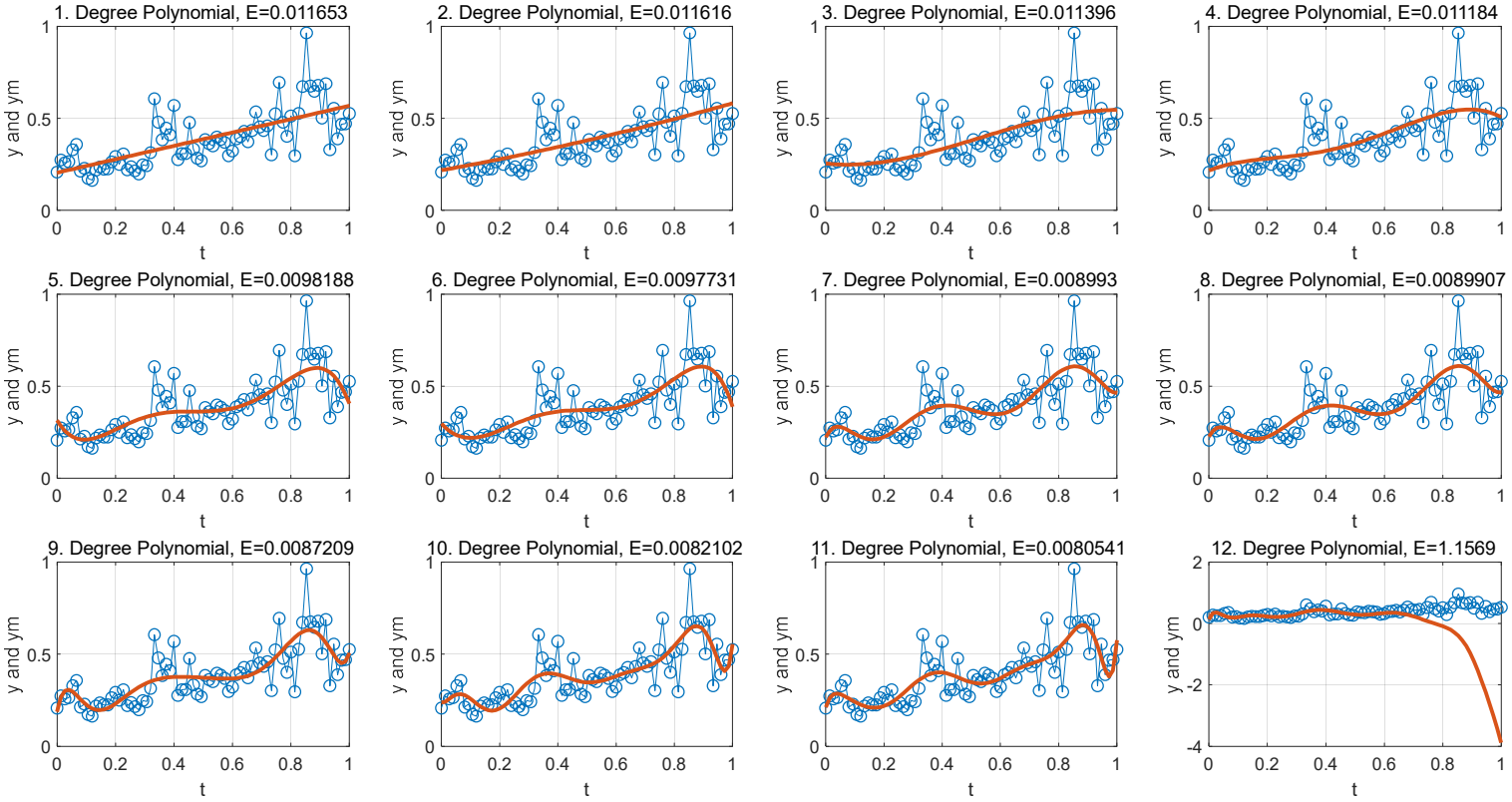


Figure 47: Data and the model as m increases.

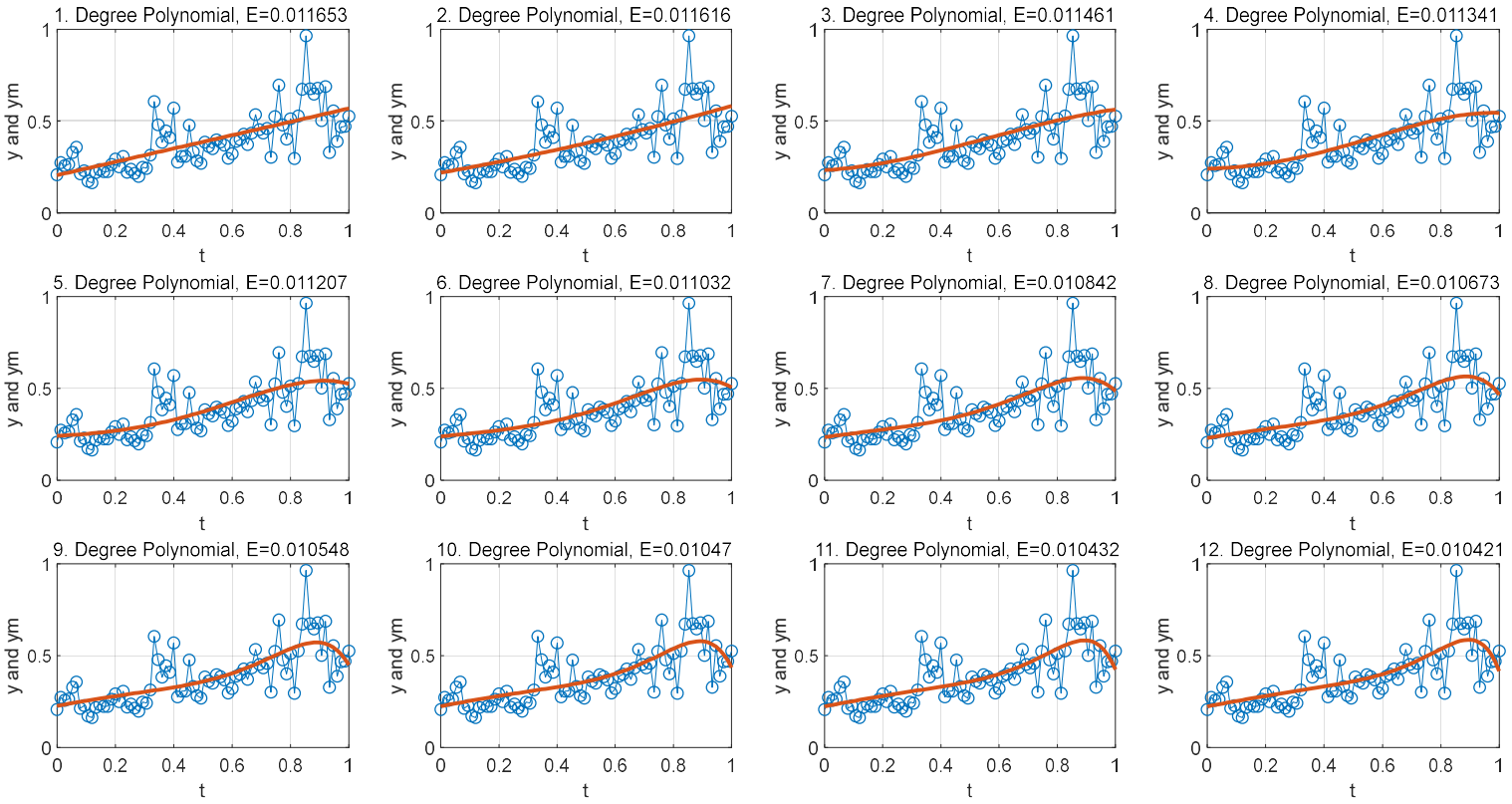


Figure 47a: Data and the model as m increases. We chose μ=0.01 and **`x=inv(mu*eye(size(A'*A))+A'*A)*A'*y`**

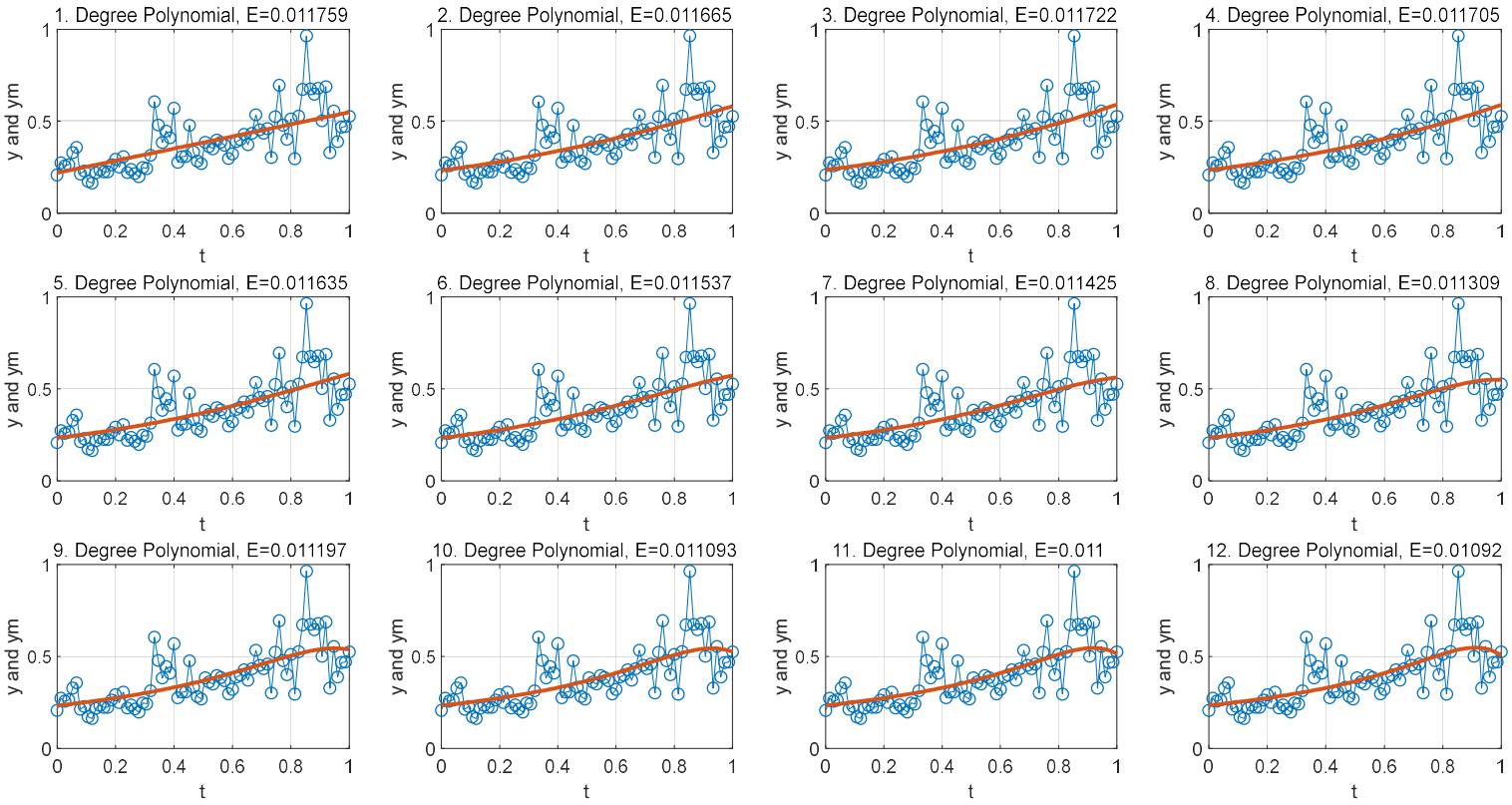


Figure 47b: Data and the model as m increases. We chose μ=1 and **`x=inv(mu*eye(size(A'*A))+A'*A)*A'*y`**

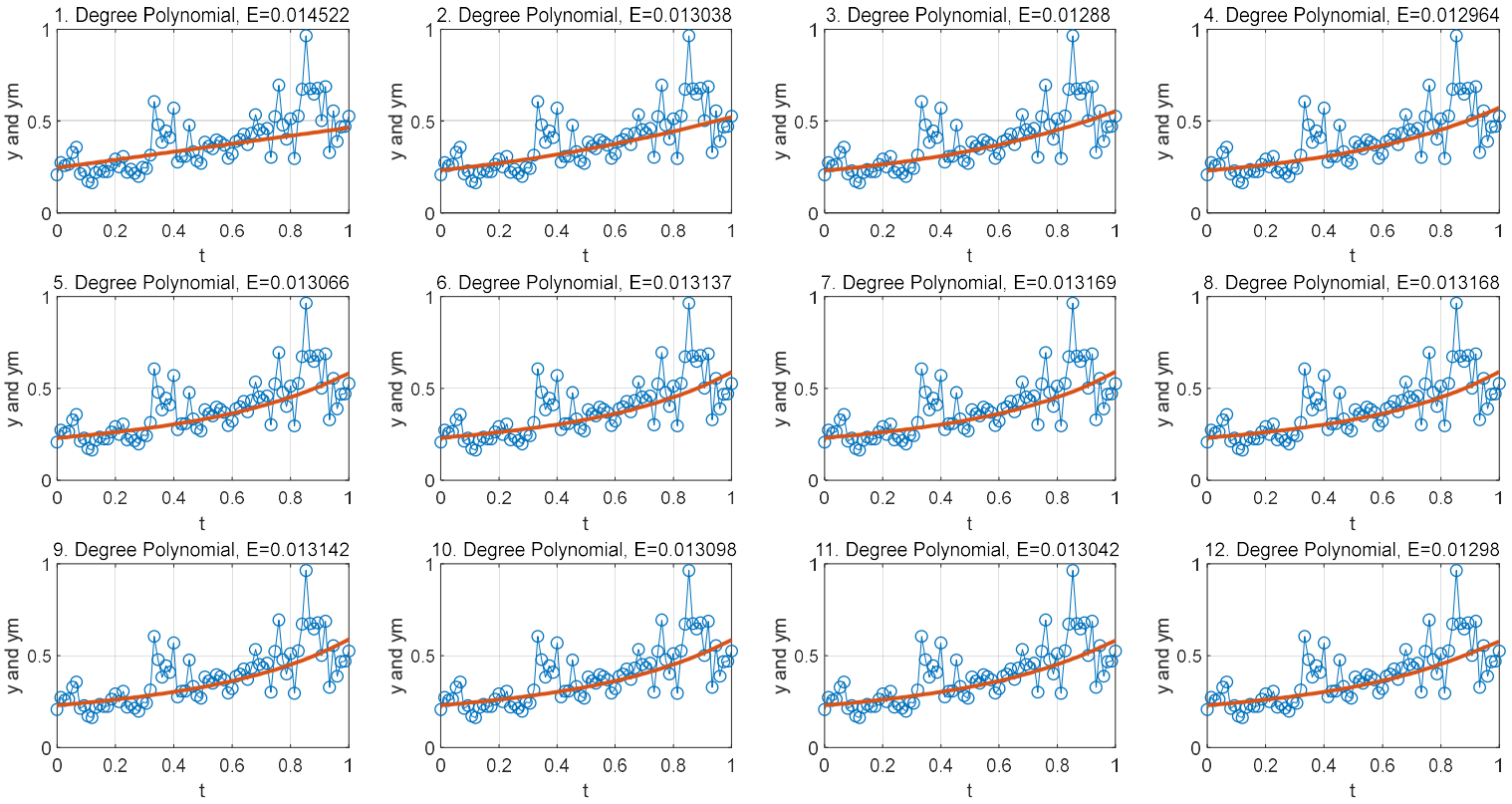


Figure 47c: Data and the model as m increases. We chose μ=10 and **`x=inv(mu*eye(size(A'*A))+A'*A)*A'*y`**

3.7 Example -6: Global Positioning Using Sensory Information

In Figure 48, **we have three sensors and each sensor tells the distance to the user standing at the square sign**. The distance to the first sensor is denoted by r_1, that to the second sensor is denoted by r_2 and that to the third sensor is r_3. These are our measurements. Let's **draw three circles using these three distances.** We are looking for the coordinates of the intersection point of these three sensors. Let's denote the user's (square sign) coordinate by $X_0 := \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$. The centers of the circles can now be given as $X_1 := \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}, X_2 := \begin{bmatrix} x_2 \\ y_2 \end{bmatrix}, X_3 := \begin{bmatrix} x_3 \\ y_3 \end{bmatrix}$. Let $X := \begin{bmatrix} x \\ y \end{bmatrix}$, the equation describing the first circle is $(x-x_1)^2 + (y-y_1)^2 = r_1^2$ or alternatively we can write $(X-X_1)^T(X-X_1) = r_1^2$ for the first circle. Based on this, **we can write the circle equations as given below**.

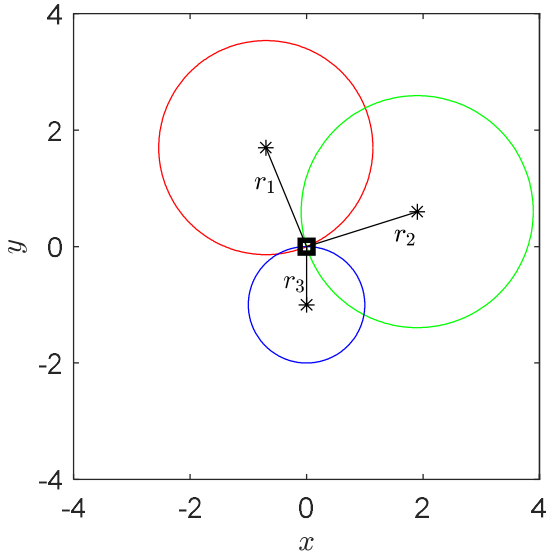


Figure 48: Finding the coordinates of the receiver using the measuements from three sensors.

$$\begin{aligned} (X-X_1)^T(X-X_1) &= r_1^2 \\ (X-X_2)^T(X-X_2) &= r_2^2 \\ (X-X_3)^T(X-X_3) &= r_3^2 \end{aligned} \tag{3.19}$$

The **intersection point of these three circles is** $X = X_0 := \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$. **Subtracting the first equation from the second and the third**, we obtain the following.

$$\begin{aligned} (X-X_2)^T(X-X_2) - (X-X_1)^T(X-X_1) &= r_2^2 - r_1^2 \\ (X-X_3)^T(X-X_3) - (X-X_1)^T(X-X_1) &= r_3^2 - r_1^2 \end{aligned} \tag{3.20}$$

Rearranging the equations will yield

$$(X^T - X_2^T)(X - X_2) - (X^T - X_1^T)(X - X_1) = r_2^2 - r_1^2$$
$$(X^T - X_3^T)(X - X_3) - (X - X_1^T)(X - X_1) = r_3^2 - r_1^2 \tag{3.21}$$

Now we will expand these equations and rearrange once again to obtain (3.22).

$$\underbrace{2\begin{bmatrix}(X_2 - X_1)^T \\ (X_3 - X_1)^T\end{bmatrix}}_{A} X = \underbrace{\begin{bmatrix}r_1^2 - r_2^2 + X_2^T X_2 - X_1^T X_1 \\ r_1^2 - r_3^2 + X_3^T X_3 - X_1^T X_1\end{bmatrix}}_{b} \tag{3.22}$$

For the above set of equations, we write the solution as $X = A^{-1}b = \begin{bmatrix}x_0 \\ y_0\end{bmatrix}$. If there are many circles (many sensors) say N, we have the following set of equations.

$$\underbrace{2\begin{bmatrix}(X_2 - X_1)^T \\ (X_3 - X_1)^T \\ \vdots \\ (X_N - X_1)^T\end{bmatrix}}_{A} X = \underbrace{\begin{bmatrix}r_1^2 - r_2^2 + X_2^T X_2 - X_1^T X_1 \\ r_1^2 - r_3^2 + X_3^T X_3 - X_1^T X_1 \\ \vdots \\ r_1^2 - r_N^2 + X_N^T X_N - X_1^T X_1\end{bmatrix}}_{b} \tag{3.23}$$

The solution is given as $X = \left(A^T A\right)^{-1} A^T b = \begin{bmatrix}x_0 \\ y_0\end{bmatrix}$. In cases where some sensors provide noisy observations, the solution above will be the best solution in the least squares sense.

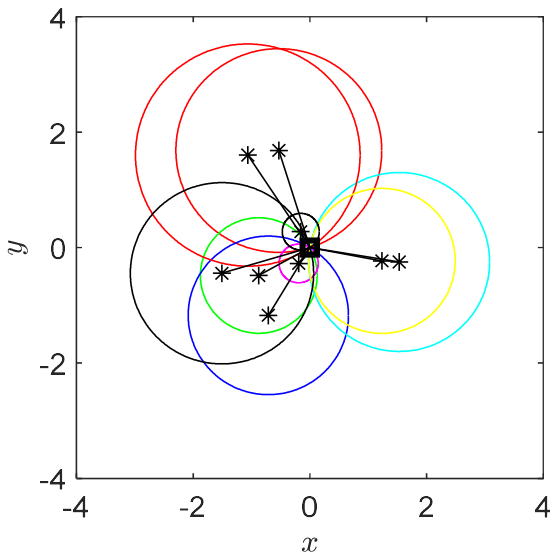


Figure 49: An exemplar situation with nine sensors.

3.8 Example -7: Finding Fourier Series Expansion Coefficients

Fourier series expansion gives the sinusoidal components and their associated coefficients of a given periodic function. Assume the period is given by T and the fundamental frequency is defined as $w_0 = \frac{2\pi}{T}$. The Fourier series expansion is given by the folowing expression, where *t* denotes time or independent variable.

$$y = a_0 + \sum_{k=1}^{M} a_k \sin(kw_0 t) + b_k \cos(kw_0 t) \tag{3.24}$$

We consider the time vector as given below.

$$t = \begin{bmatrix} 0 \\ \Delta t \\ 2\Delta t \\ \vdots \\ T/\Delta t \end{bmatrix},\ t = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ \vdots \\ t_N \end{bmatrix} \tag{3.25}$$

Our goal is to find out the coefficients a_i's and b_j's using the LS algorithm. Can we cast this problem into a matrix equation? The following equation gives the answer.

$$\begin{bmatrix} y_0 \\ y_{\Delta t} \\ y_{2\Delta t} \\ \vdots \\ y_{T/\Delta t} \end{bmatrix} = \begin{bmatrix} 1 & \sin(1w_0 0) & \cdots & \sin(Mw_0 0) & \cos(1w_0 0) & \cos(2w_0 0) & \cdots & \cos(Mw_0 0) \\ 1 & \sin(1w_0\Delta t) & \cdots & \sin(Mw_0\Delta t) & \cos(1w_0\Delta t) & \cos(2w_0\Delta t) & \cdots & \cos(Mw_0\Delta t) \\ 1 & \sin(1w_0 2\Delta t) & \cdots & \sin(Mw_0 2\Delta t) & \cos(1w_0 2\Delta t) & \cos(2w_0 2\Delta t) & \cdots & \cos(Mw_0 2\Delta t) \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \sin(1w_0 T) & \cdots & \sin(Mw_0 T) & \cos(1w_0 T) & \cos(2w_0 T) & \cdots & \cos(Mw_0 T) \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_M \\ b_1 \\ b_2 \\ \vdots \\ b_M \end{bmatrix} \tag{3.26}$$

In general, for a number of time values, say t_1, t_2, …, t_N, we would have the following matrix equality.

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{N-1} \end{bmatrix} = \begin{bmatrix} 1 & \sin(1w_0 t_1) & \cdots & \sin(Mw_0 t_1) & \cos(1w_0 t_1) & \cos(2w_0 t_1) & \cdots & \cos(Mw_0 t_1) \\ 1 & \sin(1w_0 t_2) & \cdots & \sin(Mw_0 t_2) & \cos(1w_0 t_2) & \cos(2w_0 t_2) & \cdots & \cos(Mw_0 t_2) \\ 1 & \sin(1w_0 t_3) & \cdots & \sin(Mw_0 t_3) & \cos(1w_0 t_3) & \cos(2w_0 t_3) & \cdots & \cos(Mw_0 t_3) \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \sin(1w_0 t_N) & \cdots & \sin(Mw_0 t_N) & \cos(1w_0 t_N) & \cos(2w_0 t_N) & \cdots & \cos(Mw_0 t_N) \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_M \\ b_1 \\ b_2 \\ \vdots \\ b_M \end{bmatrix} \tag{3.27}$$

The above matrix equality can be solved using the least squares algorithm. The following code simply describes the process.

```
clear
close all
clc

% Principal time interval
t = [-1:0.001:1]';

% Period
T = max(t)-min(t);

% Periodic function over the principal interval
y = sign(sin(pi*t))+0*randn(size(t));

% Number of modes
M = 40;

% A matrix
A = ones(size(t));
for k=1:M
    A = [A sin(2*pi*k*t/T) cos(2*pi*k*t/T)];
end

% Parameter vector
p = inv(A'*A)*A'*y;

figure(1)
subplot(2,1,1)
plot(t,y,'-r',t,A*p,'.b')
legend('True','Predicted','location','southeast')
grid
xlabel('t')
ylabel('y')

subplot(2,1,2)
stem([0:length(p)-1]',p)
grid
xlabel('Parameter Index (a_0,a_1,...a_M,b_1,b_2,...,b_M)')
ylabel('Magnitude')
axis([0 length(p)-1 -0.25 1.5])
```

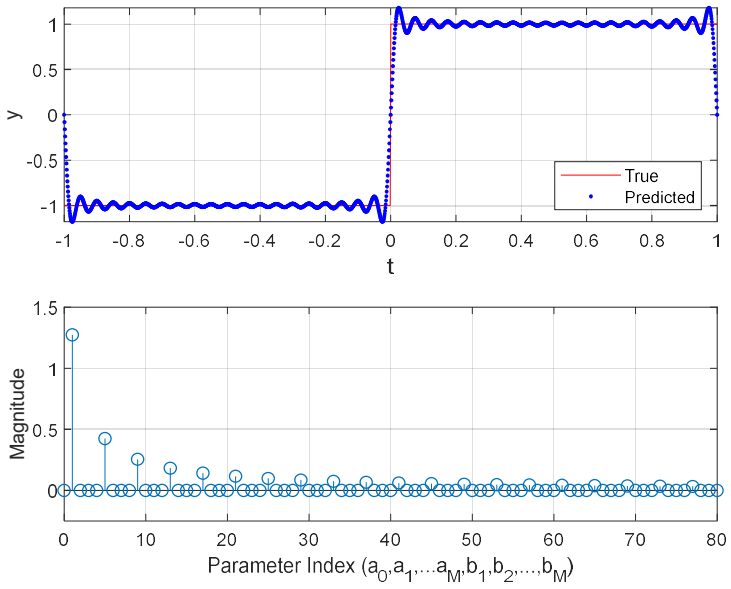


Figure 50: Extraction of the Fourier series expansion coefficients (No noise)

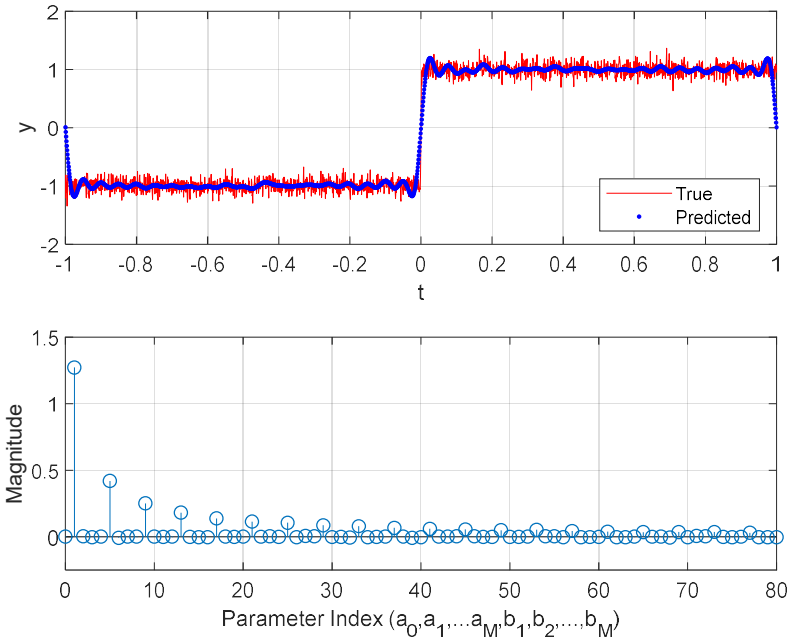


Figure 51: Extraction of the Fourier series expansion coefficients with noise coefficient is 0.1

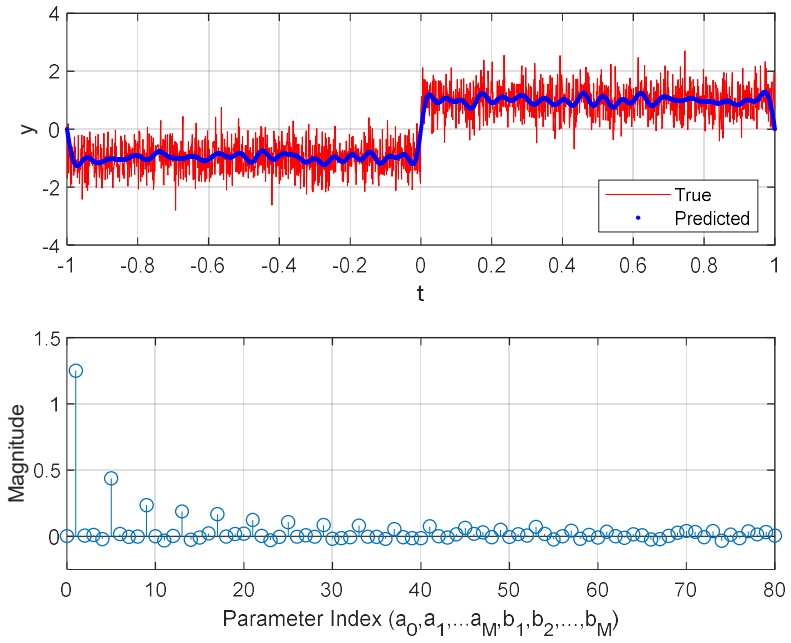


Figure 52: Extraction of the Fourier series expansion coefficients with noise coefficient is 0.5

What is our benefit in using LS algorithm for such a purpose? We know Fourier analysis provides exact values of the coefficients given the closed form of the function over the principal period. **LS algorithm develops the same result using the numerical observations, which could be possibly noisy**.

4. Application Specific Issues

4.1 R^2 Measure

In a modeling study, the **first step must be to visualize the available data**. This lets us **understand the distribution of data, outliers, mean, dimensionality** etc. Below, we will study an exemplar case.

In the top left subplot Figure 53, a nosiy data set and a fitted line is shown. In the top right, the data has a parabolic nature but the model is a linear first order one (there is no noise). In the bottom left, the model and data are linear but there is an outlier. In the bottom right, an outlier changes the whole model behavior. In all four cases, we used least squares algorithm to get the best fitting line.

In the titles of each subplot, we give the R^2 value, which describes the quality of the model. **Large values indicate good models.**

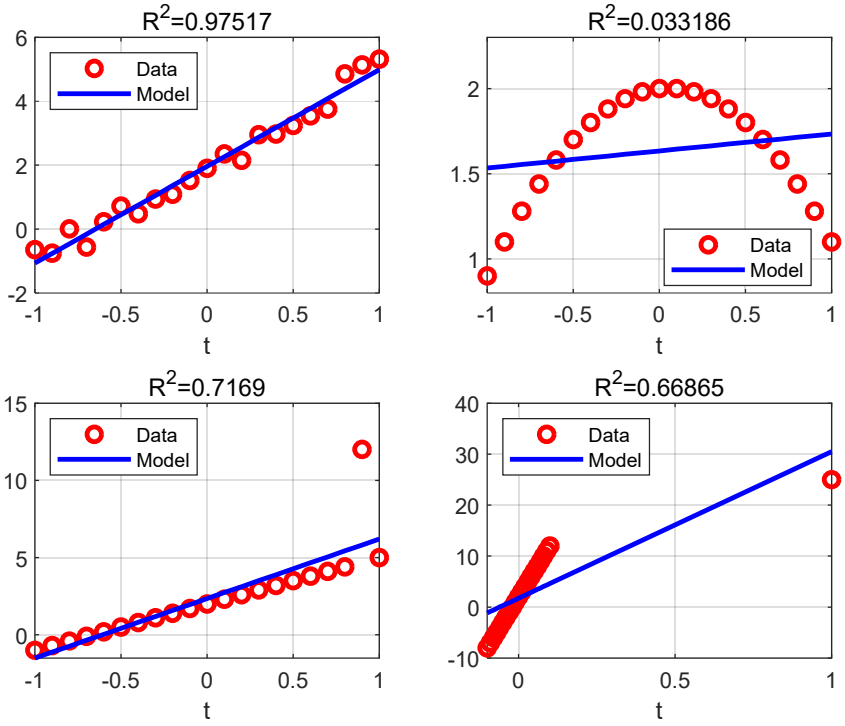


Figure 53: Value of R^2 for four different cases and the line predicted by LS algorithm.

We have two fundamental questions about Figure 53: **1) Are these models good? 2) Can these models be made better?** As an answer to these questions, a measure called R^2 is exploited. Calculation of R^2 is given below and the measurement of the values are shown in Figure 54.

$$SStot = \text{Sum Square of Total} = \sum_{i=1}^{N} (y_i - \bar{y})^2 \tag{4.1}$$

$$SSreg = \text{Sum Square of Regression} = \sum_{i=1}^{N} (\hat{y}_i - \bar{y})^2 \tag{4.2}$$

$$SSres = \text{Sum Square of Residuals} = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \tag{4.3}$$

$$R^2 = 1 - \frac{SSres}{SStot} = \frac{SSreg}{SStot} \tag{4.4}$$

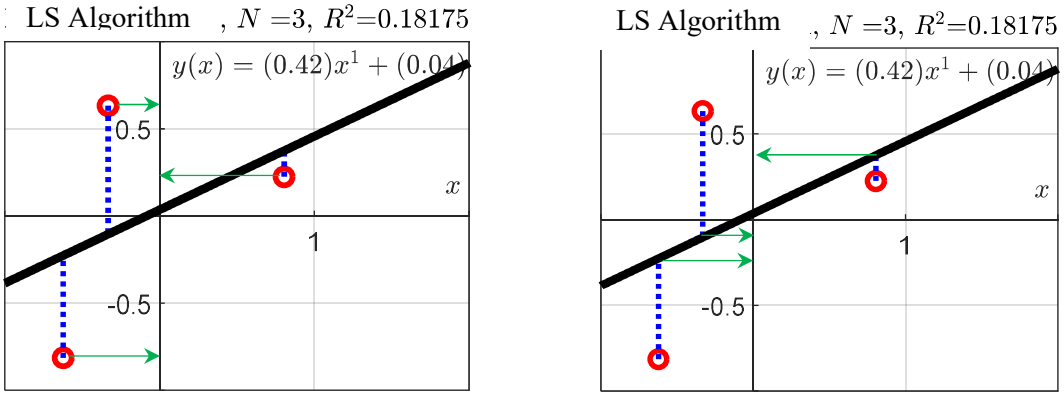


Figure 54: Left: $\bar{y}$ values, Right: $\hat{y}_i$ values

The closer the value of R^2 to 1 the better the model performance. Now, we will reconsider the R^2 values shown in Figure 53. It seems that the best modeling is accomplished in top left subplot.

For the model in top right, we have to change the model structure, i.e. the order of our model. For the model in the bottom subplots, we must exclude the outlier, and then we need to repeat the modeling study for a better model.

The Matlab code that generates Figure 55 is given below. There are two different ways to compute R^2 value, first is to compute the above quantities, the other is to use the `corrcoef` command to find the correlation coefficients. With the suggested modifications, the results are obtained as given in Figure 55.

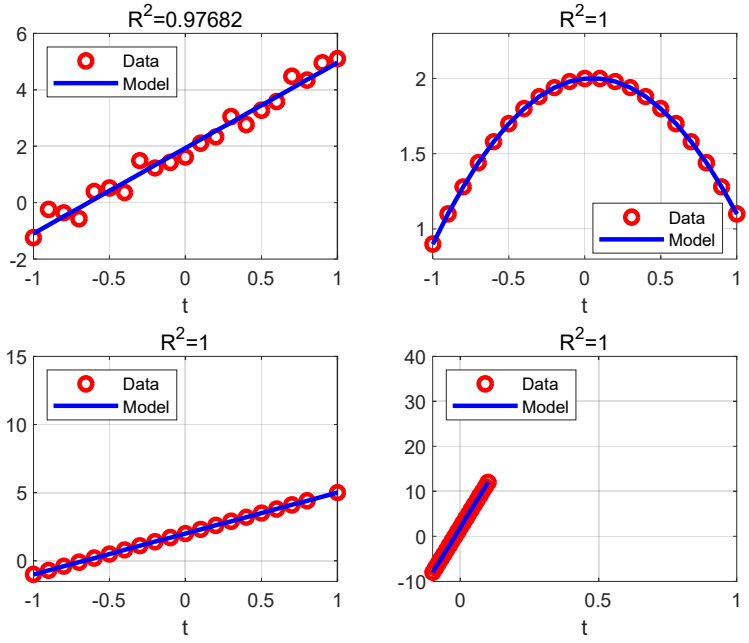


Figure 55: The results with the new modeling effort and R^2 values

We see that the R^2 values rise up to 1 for cases where there is no noise and this means that we arrived at the true process parameters.

```
clear all
close all
clc

figure(1)
t = [-1:0.1:1]';
A = [t.^1 ones(size(t))];

subplot(2,2,1)
y1 = 3*t + 2 +  0.5*(2*rand(size(t))-1);
x1 = inv(A'*A)*A'*y1;
yhat1 = A*x1;
plot(t,y1,'or',t,yhat1,'-b','LineWidth',2)
xlabel('t')
grid on
y_ortalama = mean(y1);
SStot = sum((y1 - y_ortalama).^2);
SSreg = sum((yhat1 - y_ortalama).^2);
SSres = sum((y1 - yhat1).^2);
R21 = 1 - SSres/SStot;
R1 = corrcoef(t,y1);
Rsquare1 = R1(1,2).^2;
legend('Data','Model','FontSize',8,'Location','northwest')
title(['R^2=' num2str(Rsquare1)],'FontWeight','normal')

subplot(2,2,2)
y2 = -t.^2+0.1*t+2;
x2 = inv(A'*A)*A'*y2;
yhat2 = A*x2;
plot(t,y2,'or',t,yhat2,'-b','LineWidth',2)
xlabel('t')
grid on
y_ortalama = mean(y2);
SStot = sum((y2 - y_ortalama).^2);
SSreg = sum((yhat2 - y_ortalama).^2);
SSres = sum((y2 - yhat2).^2);
R22 = 1 - SSres/SStot;
R2 = corrcoef(t,y2);
Rsquare2 = R2(1,2).^2;
legend('Data','Model','FontSize',8,'Location','northwest')
title(['R^2=' num2str(Rsquare2)],'FontWeight','normal')
axis([-1 1 0.8 2.3])

subplot(2,2,3)
y3 = 3*t + 2;
y3(length(y3)-1) = 12;
x3 = inv(A'*A)*A'*y3;
yhat3 = A*x3;
plot(t,y3,'or',t,yhat3,'-b','LineWidth',2)
xlabel('t')
grid on
y_ortalama = mean(y3);
SStot = sum((y3 - y_ortalama).^2);
SSreg = sum((yhat3 - y_ortalama).^2);
SSres = sum((y3 - yhat3).^2);
R23 = 1 - SSres/SStot;
R3 = corrcoef(t,y3);
Rsquare3 = R3(1,2).^2;
legend('Data','Model','FontSize',8,'Location','northwest')
title(['R^2=' num2str(Rsquare3)],'FontWeight','normal')

subplot(2,2,4)
t = [[-0.1:0.01:0.1]';1];
A = [t.^1 ones(size(t))];
y4 = 100*t(1:length(t)-1) + 2;
y4= [y4;25];
x4 = inv(A'*A)*A'*y4;
yhat4 = A*x4;
plot(t,y4,'or',t,yhat4,'-b','LineWidth',2)
xlabel('t')
grid on
y_ortalama = mean(y4);
SStot = sum((y4 - y_ortalama).^2);
```

```
SSreg = sum((yhat4 - y_ortalama).^2);
SSres = sum((y4 - yhat4).^2);
R24 = 1 - SSres/SStot;
R4 = corrcoef(t,y4);
Rsquare4 = R4(1,2).^2;
legend('Data','Model','FontSize',8,'Location','northwest')
title(['R^2=' num2str(Rsquare4)],'FontWeight','normal')

[Rsquare1 Rsquare2 Rsquare3 Rsquare4]
```

4.2 Adjusted R^2 Measure

In this section, we will consider the adjusted R^2 measure. Let's consider the case, where we have N observations and m parameters. For this case, least squares algorithm predicts the R^2 value as given in (4.4), i.e. we have $R^2 = 1 - \frac{SSres}{SStot} = \frac{SSreg}{SStot}$. If there are many paramaters (m) we expect a better model performance. **Adjusted R^2 is defined as below and it favors models with less parameters.**

$$R^2 = 1 - \left(\frac{N-1}{N-m}\right)\left(\frac{SSres}{SStot}\right) \tag{4.5}$$

This makes it fair to compare models exploiting few parameters and many parameters, [6].

4.3 Matlab Options (Equivalent Commands)

In this section, we present different Matlab commands that help us in performing least squares modeling.

```
clear all
% Number of observations
N=100;

% Noise level
NoiseLevel = 0.1;

% Horizontal axis variable
t=linspace(0,1,N)';

% Noise signal
n = NoiseLevel*(2*rand(N,1)-1);

% Process model and noisy measurements
y = t+1 + n;

% LS model order
d = 1;

% A matrix has the form [t^d t^(d-1)... t 1]
A=[t.^d ones(size(t))];

% First method
x1=inv(A'*A)*A'*y;
yhat1 = A*x1;

% Second method
x2 = polyfit(t, y, d)';
yhat2 = polyval(x2, t);

% Third method
x3 = regress(y,A);
yhat3 = A*x3;

% Result for all three methods
[x1 x2 x3]
```

4.4 Time Varying Model Parameters and Forgetting Factor

In the classical least squares modeling, we assumed that we have N observations. Here, we will study the least squares modeling of the processes, where **the process parameters are time varying**. For this, let's consider the constant coefficient difference equation given below. **Constant coefficients here are x_1 and x_2. In (4.6), p_k is the process output, u_k is the process input.**

$$p_{k+1} = x_1 p_k + x_2 u_k = \begin{bmatrix} p_k & u_k \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{4.6}$$

The case with time varying coefficients can be described as, i.e. we have x_{1k} and x_{2k}.

$$p_{k+1} = x_{1k} p_k + x_{2k} u_k = \begin{bmatrix} p_k & u_k \end{bmatrix} \begin{bmatrix} x_{1k} \\ x_{2k} \end{bmatrix} \tag{4.7}$$

If we write the equation for each time instant (k), we have the following set of equations:

$$\begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ \vdots \\ p_{N-1} \\ p_N \end{bmatrix} = \underbrace{\begin{bmatrix} x_{10} p_0 + x_{20} u_0 \\ x_{11} p_1 + x_{21} u_1 \\ x_{12} p_2 + x_{22} u_2 \\ \vdots \\ x_{1,N-2} p_{N-2} + x_{2,N-2} u_{N-2} \\ x_{1,N-1} p_{N-1} + x_{2,N-1} u_{N-1} \end{bmatrix}}_{b} = \underbrace{\begin{bmatrix} p_0 & u_0 \\ p_1 & u_1 \\ p_2 & u_2 \\ \vdots & \vdots \\ p_{N-2} & u_{N-2} \\ p_{N-1} & u_{N-1} \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} \hat{x}_1 \\ \hat{x}_2 \end{bmatrix}}_{\hat{x}} \tag{4.8}$$

As time progresses (as we go from top to bottom equations above) x_1 and x_2 are changing. LS algorithm will find the best $\hat{x}$ based on the available N observations. **Nevertheless, the most "true" values will be extracted from the equations in the bottom.** Having this in mind, we have to weight the equations and give larger weights to the latest observations. This would produce predictions that are more accurate.

We can write the weighted equation set as follows:

$$WA\hat{x} = Wb \tag{4.9}$$

Here **W is a weight matrix**. Moltipliying both sides from the left by $(WA)^T$ yields:

$$(WA)^T (WA)\hat{x} = (WA)^T Wb \tag{4.10}$$

Then we can obtain the solution given as:

$$\begin{aligned} \hat{x} &= \left((WA)^T (WA)\right)^{-1} (WA)^T Wb \\ &= \left(A^T W^T WA\right)^{-1} A^T W^T Wb \\ &= \left(A^T QA\right)^{-1} A^T Qb, \quad Q := W^T W \end{aligned} \tag{4.11}$$

Let's **choose the weight matrix W** as below.

$$W=\begin{bmatrix} \lambda^{N-1} & 0 & 0 & 0 & 0 \\ 0 & \ddots & 0 & 0 & 0 \\ 0 & 0 & \lambda^{2} & 0 & 0 \\ 0 & 0 & 0 & \lambda & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad Q=\begin{bmatrix} \lambda^{2(N-1)} & 0 & 0 & 0 & 0 \\ 0 & \ddots & 0 & 0 & 0 \\ 0 & 0 & \lambda^{4} & 0 & 0 \\ 0 & 0 & 0 & \lambda^{2} & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.12}$$

Here λ satisfies $0<\lambda<1$ and it is close to 1, **called forgetting factor**. The Matlab code for the implementation is given below:

```
clear all
close all
clc

N = 3000;
u = 2*rand(N,1)-1;
PredictionWindow = 300;
Weight = zeros(PredictionWindow,PredictionWindow);
ForgetFactor = 0.95;
for i=1:PredictionWindow
    Weight(i,i) = ForgetFactor^(PredictionWindow-i);
end
Q = Weight'*Weight;

Response = zeros(N,1);
Response(1)=0;
x = [1 -1]';
X = [];
for k=2:N-1
    Response(k) = x(1)*Response(k-1) + x(2)*u(k-1) + 0.01*randn;
    if k>PredictionWindow
        x(1) = 0.9*sign(sin(2*pi*k/800));
        x(2) = -(cos(2*pi*k/1000));
        A = [Response(k-PredictionWindow:k-1) u(k-PredictionWindow:k-1)];

        xhat = inv(A'*A)*A'*Response(k-PredictionWindow+1:k);
        xhatweighted = inv(A'*Q*A)*A'*Q*Response(k-PredictionWindow+1:k);
        X = [X;[x' xhat' xhatweighted']];
    end
end

figure(1)
[row col] = size(X);
t = [1:row]';

subplot(2,1,1)
plot(t,X(:,1),'--r',t,X(:,3),'-b',t,X(:,5),'-g','LineWidth',2)
xlabel('Time')
ylabel('x_1')
grid
legend('True Value','Classical','Weighted')

subplot(2,1,2)
plot(t,X(:,2),'--r',t,X(:,4),'-b',t,X(:,6),'-g','LineWidth',2)
xlabel('Time')
ylabel('x_2')
grid
legend('True Value','Classical','Weighted')
```

In the above code, we study the following time variance in the parameters:

$$
\begin{aligned}
p_{k+1} &= x_{1k}\, p_k + x_{2k} u_k \\
x_{1k} &= 0.9\,\mathrm{sgn}\left(\sin\left(\frac{2\pi k}{800}\right)\right) \\
x_{2k} &= -\cos\left(\frac{2\pi k}{1000}\right)
\end{aligned}
\tag{4.13}
$$

Here we choose N=300, and the forgetting factor is λ=0.95. The obtained results are illustrated in Figure 56.

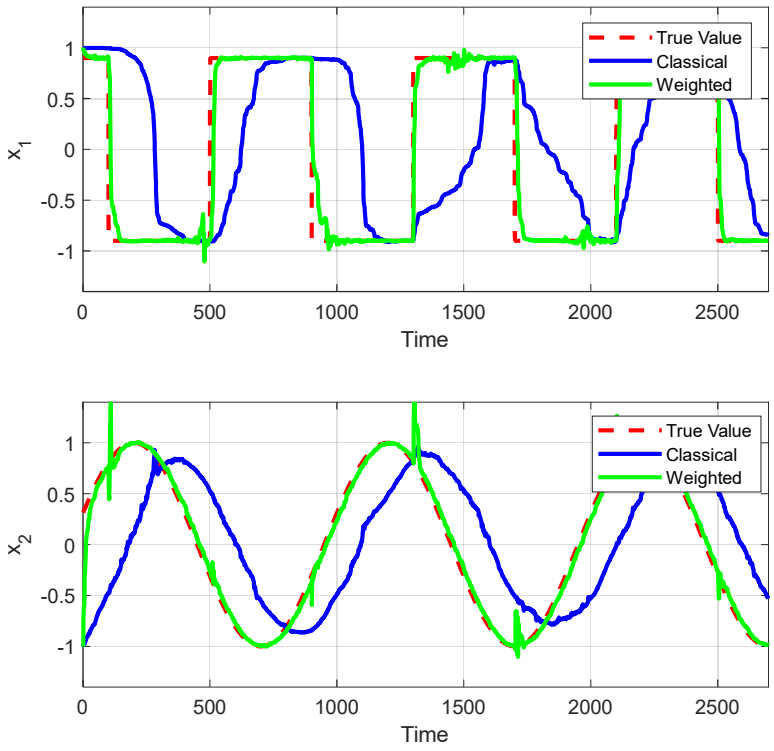


Figure 56a: A Comparison of Classical LS Algorithm and Weighted LS Algorithm

According to the results seen in Figure 56, we see the **true values as red dashed curves** defined by (4.13). **Classical LS algorithm (blue cuves)** finds a constant parameter set but that set does not follow the true values closely. The main reason is that all measurements are equally weighted and old ones have negative impact on the result.

The solution with the forgetting facor generates the green curves. With the chosen N and λ values, we see that the results are closer to the true ones than those with classical scheme.

We determined the values of N and λ via trial and error. Since the very recent values have more impact on the result, **when there is a sudden change in the true values, then we observe convergent fluctuations in the predictions, which are absent in the classical scheme.**

In summary, **if the instantaneous importance of data is changing, we can reflect this to the solution** and we can **consider the recent data more dominantly** in our solution than old data. This leads us to the LS algorithm with forgetting factor.

Instead of exponential decay, let's choose a linear array on the diagonal, which is given in the below equation, where x is a small number.

$$Q = \begin{bmatrix} x & 0 & 0 & 0 & 0 \\ 0 & \ddots & 0 & 0 & 0 \\ 0 & 0 & 1-2x & 0 & 0 \\ 0 & 0 & 0 & 1-x & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}^P \tag{4.x}$$

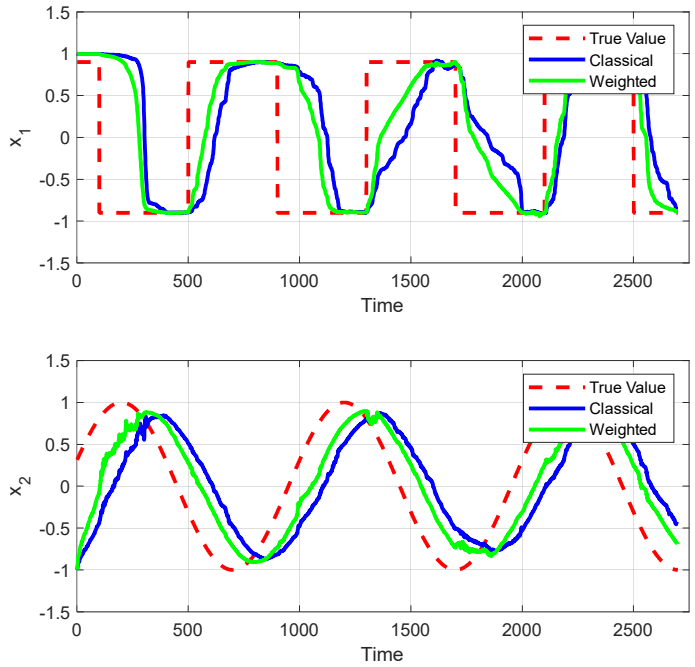


Figure 56b: A Comparison of Classical LS Algorithm and Weighted LS Algorithm when `Q = diag([1:PredictionWindow]/PredictionWindow).^P and P=1`

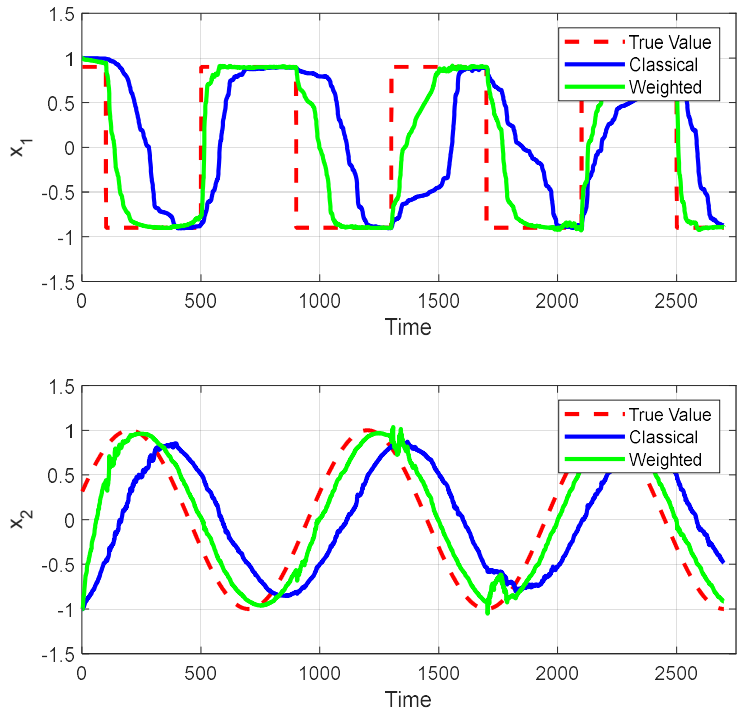


Figure 56c: A Comparison of Classical LS Algorithm and Weighted LS Algorithm when `Q = diag([1:PredictionWindow]/PredictionWindow).^P and P=4`

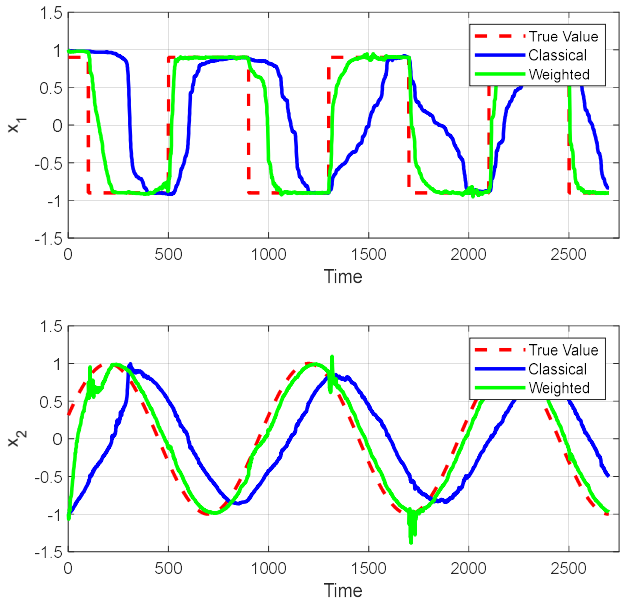


Figure 56d: A Comparison of Classical LS Algorithm and Weighted LS Algorithm when `Q = diag([1:PredictionWindow]/PredictionWindow).^P and P=8`

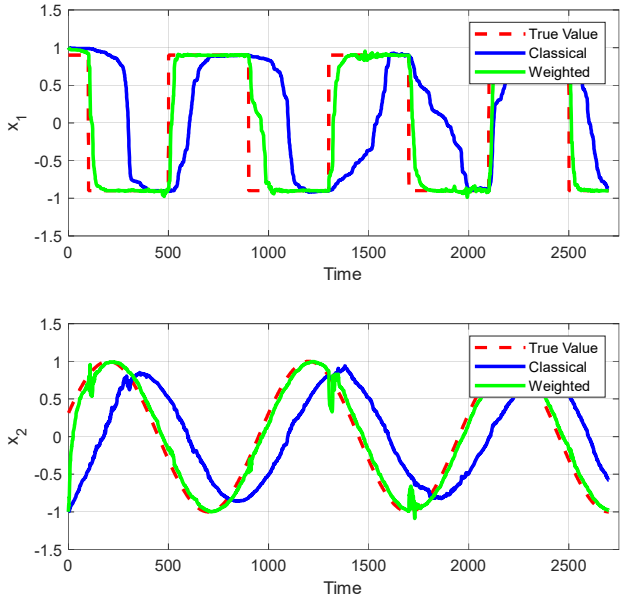


Figure 56e: A Comparison of Classical LS Algorithm and Weighted LS Algorithm when `Q = diag([1:PredictionWindow]/PredictionWindow).^P and P=13`

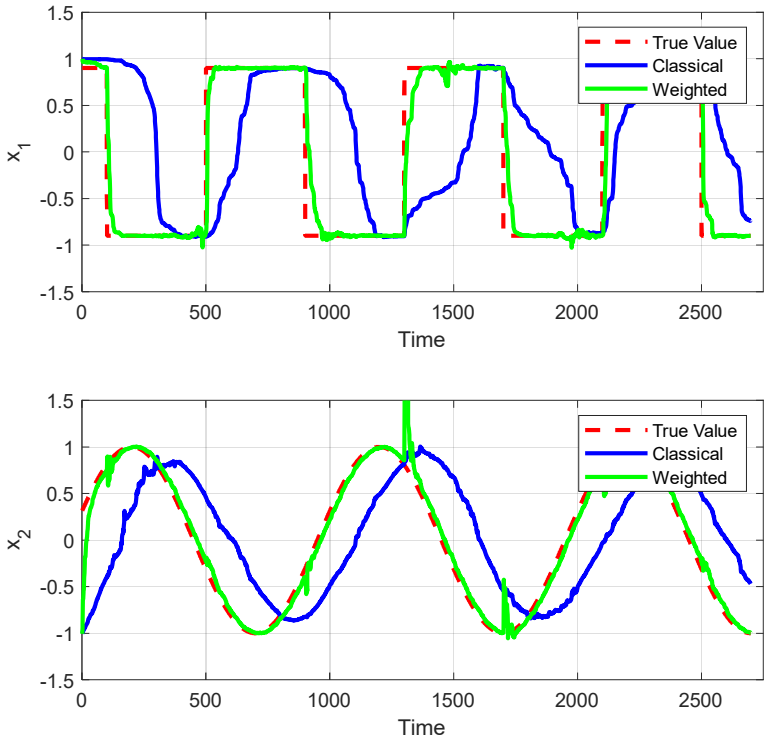
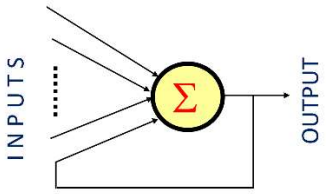


Figure 56f: A Comparison of Classical LS Algorithm and Weighted LS Algorithm when `Q = diag([1:PredictionWindow]/PredictionWindow).^P and P=20`

4.5 Determining AR, MA and ARMA Model Coefficients

An *Autoregressive Moving Average* (ARMA) model is given as in (4.14). In this model, if x_i=0, we obtain moving average (MA) model, if y_i =0, we obtain autoregressive (AR) model. We will study autoregressive movin average (ARMA) process model.

$$p_{k+1} = x_0 p_k + x_1 p_{k-1} + \cdots + x_n p_{k-n} + y_0 u_k + y_1 u_{k-1} + \cdots + y_m u_{k-q} \tag{4.14}$$



We can rewrite this equation using simple vectors as in (4.15).

$$p_{k+1} = X^T P_k + Y^T U_k \tag{4.15}$$

where

$$X = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}, P_k = \begin{bmatrix} p_k \\ p_{k-1} \\ \vdots \\ p_{k-n} \end{bmatrix}, Y = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_q \end{bmatrix}, U_k = \begin{bmatrix} u_k \\ u_{k-1} \\ \vdots \\ u_{k-q} \end{bmatrix} \tag{4.16}$$

According to these definitions, the ARMA process can be written as in (4.17). This representation says that the **ARMA model is linear in its parameters**. With adequate number of observations (i.e. $N>n+q+2$) we can find out unknown x_i and y_i values, in other words, we can identify the process.

$$\begin{aligned} p_{k+1} &= X^T P_k + Y^T U_k \\ &= \begin{bmatrix} X^T & Y^T \end{bmatrix} \begin{bmatrix} P_k \\ U_k \end{bmatrix} \\ &= \begin{bmatrix} P_k^T & U_k^T \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix} \end{aligned} \tag{4.17}$$

Let's write the measurements as given below, this will let us obtain the equation set $Ax=b$ again.

$$\underbrace{\begin{bmatrix} p_{k+1} \\ p_{k+2} \\ p_{k+3} \\ \vdots \\ p_{k+N} \end{bmatrix}}_{b} = \underbrace{\begin{bmatrix} P_k^T & U_k^T \\ P_{k+1}^T & U_{k+1}^T \\ P_{k+2}^T & U_{k+2}^T \\ \vdots & \vdots \\ P_{k+N-1}^T & U_{k+N-1}^T \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} X \\ Y \end{bmatrix}}_{x} \tag{4.18}$$

LS solution of the above equation set can be obtained as below.

$$\begin{bmatrix} X \\ Y \end{bmatrix} = \left(\begin{bmatrix} P_k^T & U_k^T \\ P_{k+1}^T & U_{k+1}^T \\ P_{k+2}^T & U_{k+2}^T \\ \vdots & \vdots \\ P_{k+N-1}^T & U_{k+N-1}^T \end{bmatrix}^T \begin{bmatrix} P_k^T & U_k^T \\ P_{k+1}^T & U_{k+1}^T \\ P_{k+2}^T & U_{k+2}^T \\ \vdots & \vdots \\ P_{k+N-1}^T & U_{k+N-1}^T \end{bmatrix} \right)^{-1} \begin{bmatrix} P_k^T & U_k^T \\ P_{k+1}^T & U_{k+1}^T \\ P_{k+2}^T & U_{k+2}^T \\ \vdots & \vdots \\ P_{k+N-1}^T & U_{k+N-1}^T \end{bmatrix}^T \begin{bmatrix} p_{k+1} \\ p_{k+2} \\ p_{k+3} \\ \vdots \\ p_{k+N} \end{bmatrix} \tag{4.19}$$

In the following, we give an exemplar identification problem. The Matlab code extracting the coefficients from the observations is given next. The code is as below and the results are seen in Figure 57.

$$p_{k+1} = x_0 p_k + x_1 p_{k-1} + \cdots + x_n p_{k-n} + y_0 u_k + y_1 u_{k-1} + \cdots + y_m u_{k-q} \tag{4.20}$$
$$p_{k+1} = 0.1 p_k + 0.2 p_{k-1} - 0.3 p_{k-2} + 0.6 u_k + 0.9 u_{k-1}$$

```
clear all
close all
clc

% Parameters
x0 = 0.1;
x1 = 0.2;
x2 = -0.3;

x = [x0 x1 x2];
n = length(x)-1;
m = 1;

y0 = 0.6;
y1 = 0.9;

y = [y0 y1];
q = length(y)-1;

p = zeros(n+1,1);
u = zeros(m+1,1);

A = [];
b = [];
NoiseLevel = 0.1;

% Number of observarions
N=150;

% Data collection from the process
for k = n+1:N
    u(k) = randn;
    noise(k) = NoiseLevel*randn;
    p(k+1) = x0*p(k) + x1*p(k-1) + x2*p(k-2) +...
        y0*u(k) + y1*u(k-1) + noise(k);
    Pk = [p(k) p(k-1) p(k-2)];
    Uk = [u(k) u(k-1)];
    A = [A;[Pk Uk]];
    b = [b;p(k+1)];
end

% LS prediction
par = inv(A'*A)*A'*b;
```

```
% Resolving the parameters
x0hat = par(1);
x1hat = par(2);
x2hat = par(3);
y0hat = par(4);
y1hat = par(5);

% Testing the model
phat = zeros(n+1,1);
for k = n+1:N
    phat(k+1) = x0hat*phat(k) + x1hat*phat(k-1) + x2hat*phat(k-2) +...
        y0hat*u(k) + y1hat*u(k-1);
end

% Plotting the results
figure(1)
subplot(2,1,1)
time = [0:length(p)-1];
stairs(time,p,'LineWidth',1)
hold on
stairs(time,phat,'--','LineWidth',1)
xlabel('Time')
legend('p','phat')
title(['x=[' num2str([x0 x1 x2]) '], y=[' num2str([y0 y1]) ']'])
ylabel('p,phat')

subplot(2,1,2)
stairs(time,p-phat,'LineWidth',1)
xlabel('Time')
title(['xhat=[' num2str([x0hat x1hat x2hat]) '], yhat=[' num2str([y0hat y1hat])
']'])
ylabel('p-phat')
```

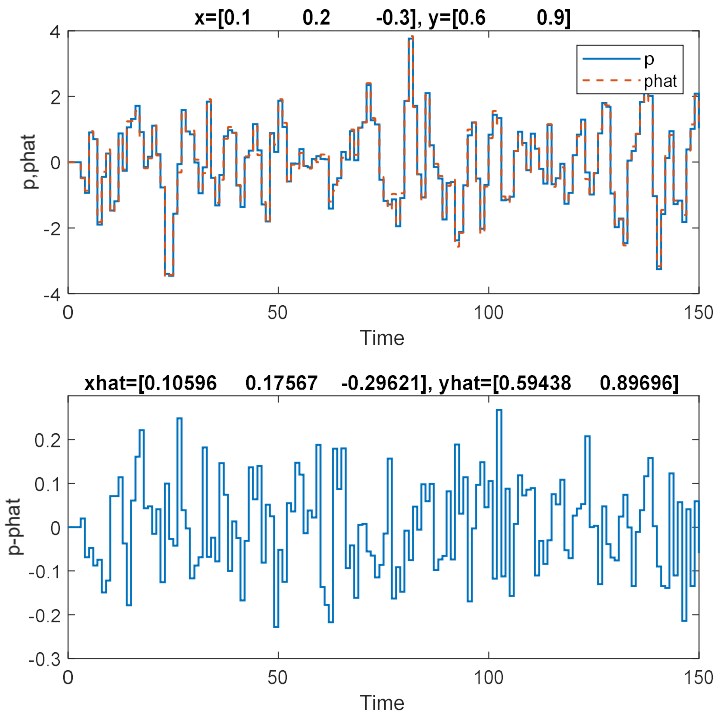


Figure 57: Determining the ARMA process coefficients using least squares algorithm

The results seen in Figure 57 indicate that the process parameters can be found under the presence of noise. This simple example shows us that a set of **model parameters can be extracted from the input/output observations** if the model out put is linear in the unknown parameters.

4.6 Irrelevant Input Variables

Under this title, two different cases can be of interest. These are:

1) Some inputs are linear combinations of others,
2) Some inputs are irrelevant to the process model.

In this section, we cill study these two cases with examples and we will use Matlab to determine the situation.

In the following two sections, we will change our notation to comply with Matlab. The model structure is $\hat{y} = \beta_0 + \beta_1 x_1 + \cdots + \beta_{m-1} x_{m-1}$, the unknown model parameters are denoted by β_i, and the variables x_i are the input variables (predictor) of the model. In Matlab environment, the variable β_0 is called *intercept*.

4.6.1 One Input is a Linear Combination of the Others

Let the process generating the data is as given in (69). The input variables to the process are $x_1, x_2, \ldots, x_m$ and there are m patrameters, these are $\beta_1^*, \beta_2^*, \cdots, \beta_m^*$. The true values of these parameters are assumed to be unknown.

$$y = \sum_{i=1}^{m} \beta_i^* x_i \tag{4.21}$$

Let's devise a model having $m+1$ parameters. **We will set $(m+1)^{th}$ input as a linear combination of the first m variables**, i.e. we have

$$\hat{y} = \sum_{i=1}^{m} \hat{\beta}_i x_i + \hat{\beta}_{m+1}(\alpha_1 x_1 + \alpha_2 x_2 + \cdots + \alpha_m x_m) \tag{4.22}$$

In this case, the equation set $Ax=b$ will have the A matrix and b vector as given below.

$$\underbrace{\begin{bmatrix} x_{11} & x_{21} & \cdots & x_{m+1,1} \\ x_{12} & x_{22} & \cdots & x_{m+1,2} \\ \vdots & \vdots & \ddots & \vdots \\ x_{1N} & x_{2N} & \cdots & x_{m+1,N} \end{bmatrix}}_{A}_{N\times(m+1)} \underbrace{\begin{bmatrix} \hat{\beta}_1 \\ \hat{\beta}_2 \\ \vdots \\ \hat{\beta}_{m+1} \end{bmatrix}}_{\hat{\beta}}_{(m+1)\times 1} = \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}}_{b}_{N\times 1} \tag{4.23}$$

Denoting the columns of matrix A as a_i would yield the following.

$$\begin{bmatrix} a_1 & a_2 & \cdots & a_m & a_{m+1} \end{bmatrix} \hat{\beta} = b \tag{4.24}$$

Now we will **express the last column as a linear combination of the first m columns** as given below;

$$\underbrace{\begin{bmatrix} a_1 & a_2 & \cdots & a_m & \alpha_1 a_1 + \alpha_1 a_2 + \cdots + \alpha_m a_m \end{bmatrix}}_{A} \hat{\beta} = b \tag{4.25}$$

The column rank of the A matrix seen above can at most be equal to m, i.e. **last column did not contribute a new information to the modeling problem**. Rearranging (4.22) as in (4.26) proves this, there is no new information in the $(m+1)^{th}$ column of A.

$$\hat{y} = \sum_{i=1}^{m} \hat{\beta}_i x_i + \hat{\beta}_{m+1}(\alpha_1 x_1 + \alpha_2 x_2 + \cdots + \alpha_m x_m) = \sum_{i=1}^{m} \left(\hat{\beta}_i + \hat{\beta}_{m+1}\alpha_i\right) x_i \tag{4.26}$$

Now our question turns into the following: **Is there a way to determine the linearly combined inputs that do not contribute to the modeling problem?** We will see how Matlab handles these cases in the following examples.

The **mdl** object obtained using the **fitlm command** containg quite a lot of information about the LS modeling. The abbreviation is due to "fit linear model".

```
mdl = fitlm(A,b)
```

Check the following Matlab code.

```
clear all
clc

% Independent variable (horizontal axis)
x=[0:0.01:1]';

% Exact model of the process that generates the data
y=3*x+4;

% Matrix A (fitlm command does not require constant input, i.e. the intercept)
A=x;

% Process response
b=y;

% Modeling command
mdl=fitlm(A,y)
```

Below is the output of the code:

```
Linear regression model:
    y ~ 1 + x1

Estimated Coefficients:
                   Estimate        SE           tStat       pValue
                   ________    __________    __________    ______

    (Intercept)       4        1.3388e-08    2.9877e+08      0
    x1                3        2.3131e-08     1.297e+08      0

Number of observations: 101, Error degrees of freedom: 99
Root Mean Squared Error: 6.78e-08
R-squared: 1,  Adjusted R-Squared: 1
F-statistic vs. constant model: 1.68e+16, p-value = 0
```

In the above box, the line representing the linear regression is indicated by **y ~ 1 + x1**. The meaning of this is that for a representation $\hat{y} = \hat{\beta}_0 \times \underbrace{1}_{\text{Intercept}} + \hat{\beta}_1 \times \underbrace{x}_{x1}$, the intercept denoted by $\hat{\beta}_0$ is equal to 4, and the coefficient dnoted by $\hat{\beta}_1$ is equal to 3. These are seen below the *Estimate* column above. The SE column is for Standard Error and to understand this one can refer to the following Matlab code[3].

[3] Code taken from: https://www.mathworks.com/matlabcentral/answers/373940-how-does-matlab-calculate-standard-error-in-fitlm

```
%%Sample Data Definition
X = [1 2 3 4 5 6 7 8 9 10];
Y = [1 1.5 2.1 4 5 6.3 7.3 7.8 9 9.9];
%%Calculation of Standard Error Without Intercept
%  Number of observations
N = length(X);
%  Calculates slope
Slope = sum(X.*Y)/sum(X.^2);
%  Fitted response values based on the slope
yfit=X*Slope;
%  r is the residuals, which is the observed minus fitted values
r = Y - yfit;
%  SSE is the sum of squared errors
SSE = sum(r.^2);
%  Mean Squared Error
MSE=SSE/(N-1);
%  Standard error of the slope
Code_NoIntercept_SE=sqrt(MSE/sum(X.^2))
%  Calls 'fitlm'
model1 = fitlm(X,Y,'Intercept',false);
%  Queries the standard error of MATLAB's model
MATLAB_NoIntercept_SE=model1.Coefficients.SE


%%Clear Variables then Redefine Data
clear all
X = [1 2 3 4 5 6 7 8 9 10];
Y = [1 1.5 2.1 4 5 6.3 7.3 7.8 9 9.9];
%%Calculation of Standard Error With Intercept
%  Number of observations
N = length(X);
%  Calculates mean of X
XBar=mean(X);
%  Calculates mean of Y
YBar=mean(Y);
Sxx=sum((X-XBar).^2);
Sxy=sum((X-XBar).*(Y-YBar));
%  Calculates Slope
Slope = Sxy/Sxx;
%  Calculates Intercept
Intercept= YBar-Slope*XBar;
%  Fitted response values based on the slope
yfit=Intercept + X*Slope;
%  r is the residuals, which is the observed minus fitted values
r = Y - yfit;
%  SSE is the sum of squared errors
SSE = sum(r.^2);
%  Mean Squared Error
MSE=SSE/(N-2);
%  Standard Error of the regression coefficients
Code_Intercept_SE=[
    %  Standard Error of the intercept coefficient
    sqrt(MSE*sum(X.^2)/(N*Sxx));
    %  Standard Error of the slope coefficient
    sqrt(MSE/Sxx)]
%  Calls 'fitlm'
model2 = fitlm(X,Y,'Intercept',true);
%  Queries the standard error of MATLAB's model
MATLAB_Intercept_SE=model2.Coefficients.SE
```

Obviously, for a good modeling, we need to see small values under the SE column. **Let's see the experiments with noise now.**

```
x=[0:0.01:1]';
y=3*x+4+0.01*randn(size(x));
A=x;
b=y;
mdl=fitlm(A,y)


Linear regression model:
    y ~ 1 + x1

Estimated Coefficients:
                   Estimate       SE         tStat       pValue
                   ________    _________    ______    ____________

    (Intercept)     4.0045     0.0022697    1764.3    1.8817e-224
    x1              2.9935     0.0039214    763.38    1.9594e-188


Number of observations: 101, Error degrees of freedom: 99
Root Mean Squared Error: 0.0115
R-squared: 1,  Adjusted R-Squared: 1
F-statistic vs. constant model: 5.83e+05, p-value = 1.96e-188
```

```
x=[0:0.01:1]';
y=3*x+4+0.1*randn(size(x));
A=x;
b=y;
mdl=fitlm(A,y)

Linear regression model:
    y ~ 1 + x1

Estimated Coefficients:
                   Estimate       SE        tStat       pValue
                   ________    ________    ______    ____________

    (Intercept)     3.9839     0.019848    200.72    4.7644e-131
    x1              3.0144     0.034292    87.905     9.0926e-96


Number of observations: 101, Error degrees of freedom: 99
Root Mean Squared Error: 0.1
R-squared: 0.987,  Adjusted R-Squared: 0.987
F-statistic vs. constant model: 7.73e+03, p-value = 9.09e-96
```

```
x=[0:0.01:1]';
y=3*x+4+randn(size(x));
A=x;
b=y;
mdl=fitlm(A,y)

Linear regression model:
    y ~ 1 + x1

Estimated Coefficients:
                   Estimate      SE       tStat       pValue
                   ________    _______    ______    __________

    (Intercept)     3.9877     0.20054    19.884    2.3869e-36
    x1              2.8129     0.34648    8.1185    1.3369e-12


Number of observations: 101, Error degrees of freedom: 99
Root Mean Squared Error: 1.02
R-squared: 0.4,  Adjusted R-Squared: 0.394
F-statistic vs. constant model: 65.9, p-value = 1.34e-12
```

Now we will remove the noise component and change the model structure to $\hat{y}(x) = \hat{\beta}_0 + \hat{\beta}_1 x + \hat{\beta}_2 \left(\alpha_0 + \alpha_1 x\right)$. Matlab code and the results are seen below.

```
x=[0:0.01:1]';
y=3*x+4;
A=[x 5*x-9];
b=y;
mdl=fitlm(A,y)
yhat = predict(mdl,A);
plot(x,y,'or',x,yhat,'xb')


Warning: Regression design matrix is rank deficient to within machine precision.

mdl =


Linear regression model:
    y ~ 1 + x1 + x2

Estimated Coefficients:
                   Estimate        SE           tStat       pValue
                   ________    __________    ___________    ______

    (Intercept)           0             0           NaN      NaN
    x1               5.2222    2.4212e-08    2.1568e+08        0
    x2              -0.44444    2.1037e-09    -2.1126e+08       0


Number of observations: 101, Error degrees of freedom: 99
Root Mean Squared Error: 9.58e-08
R-squared: 1,  Adjusted R-Squared: 1
F-statistic vs. constant model: 8.41e+15, p-value = 0
```

As expected, **we obtained an error message and the warning about rank is displayed**. In spite of this, **`fitlm` command considered all terms supplied (no variable elimination occured)** and we obtained the above results. The reader must pay attention to the **`predict` command, which is used to get the model response.**

Now we will repet the same experiment with **stepwiselm** command. The reader must **pay attention to the elimination of the third term** and correct prediction of the coefficients.

```
x=[0:0.01:1]';
y=3*x+4;
A=[x 5*x-9];
b=y;
mdl=stepwiselm(A,y)
yhat = predict(mdl,A);
plot(x,y,'or',x,yhat,'xb')


1. Adding x1, FStat = 1.682084565218227e+16, pValue = 0

mdl =


Linear regression model:
    y ~ 1 + x1

Estimated Coefficients:
                   Estimate        SE           tStat       pValue
                   ________    __________    __________    ______

    (Intercept)       4        1.3388e-08    2.9877e+08      0
    x1                3        2.3131e-08     1.297e+08      0


Number of observations: 101, Error degrees of freedom: 99
Root Mean Squared Error: 6.78e-08
R-squared: 1,  Adjusted R-Squared: 1
F-statistic vs. constant model: 1.68e+16, p-value = 0
```

Now we will **increase the noise level** and see the results of **`stepwiselm`** command.

```
x=[0:0.01:1]';
y=3*x+4+0.01*randn(size(x));
A=[x 5*x-9];
b=y;
mdl=stepwiselm(A,y)
yhat = predict(mdl,A);
plot(x,y,'or',x,yhat,'xb')


1. Adding x1, FStat = 882807.5886, pValue = 2.313189819e-197

mdl =


Linear regression model:
    y ~ 1 + x1

Estimated Coefficients:
                   Estimate        SE         tStat       pValue
                   ________    _________    _______    ____________

    (Intercept)           4     0.0018476       2165    2.9887e-233
    x1               2.9992     0.0031921    939.58    2.3132e-197


Number of observations: 101, Error degrees of freedom: 99
Root Mean Squared Error: 0.00935
R-squared: 1,  Adjusted R-Squared: 1
F-statistic vs. constant model: 8.83e+05, p-value = 2.31e-197
```

```
x=[0:0.01:1]';
y=3*x+4+0.1*randn(size(x));
A=[x 5*x-9];
b=y;
mdl=stepwiselm(A,y)
yhat = predict(mdl,A);
plot(x,y,'or',x,yhat,'xb')


1. Adding x1, FStat = 7702.1429, pValue = 1.0662363e-95

mdl =


Linear regression model:
    y ~ 1 + x1

Estimated Coefficients:
                   Estimate        SE        tStat       pValue
                   ________    _________    _______    ____________

    (Intercept)      4.0036     0.019607    204.19    8.7583e-132
    x1                2.973     0.033876    87.762     1.0662e-95


Number of observations: 101, Error degrees of freedom: 99
Root Mean Squared Error: 0.0993
R-squared: 0.987,  Adjusted R-Squared: 0.987
F-statistic vs. constant model: 7.7e+03, p-value = 1.07e-95
```

```
x=[0:0.01:1]';
y=3*x+4+randn(size(x));
A=[x 5*x-9];
b=y;
mdl=stepwiselm(A,y)
yhat = predict(mdl,A);
plot(x,y,'or',x,yhat,'xb')


1. Adding x1, FStat = 67.2954, pValue = 8.78464e-13

mdl =


Linear regression model:
    y ~ 1 + x1

Estimated Coefficients:
                   Estimate      SE       tStat       pValue
                   ________    _______    ______    __________

    (Intercept)     4.0769     0.20845    19.558    8.8331e-36
    x1              2.9544     0.36014    8.2034    8.7846e-13


Number of observations: 101, Error degrees of freedom: 99
Root Mean Squared Error: 1.06
R-squared: 0.405,  Adjusted R-Squared: 0.399
F-statistic vs. constant model: 67.3, p-value = 8.78e-13
```

In another repetition of the above experiment, x2 variable is chosen, this is shown below.

```
1. Adding x2, FStat = 71.6396, pValue = 2.40837e-13

mdl =


Linear regression model:
    y ~ 1 + x2

Estimated Coefficients:
                   Estimate       SE       tStat       pValue
                   ________    ________    ______    __________

    (Intercept)     9.0551      0.43028    21.045    2.5202e-38
    x2              0.54671    0.064592     8.464    2.4084e-13


Number of observations: 101, Error degrees of freedom: 99
Root Mean Squared Error: 0.946
R-squared: 0.42,  Adjusted R-Squared: 0.414
F-statistic vs. constant model: 71.6, p-value = 2.41e-13
```

Now we will ad many variables and analyze the variable selection approach of **stepwiselm**. We have `A=[x 5*x-9 8*x+2 pi*x-0.4 x/pi-4]`. **Except the first term, the rest of the content of A are unnecessary**.The algorithm might choose any one of these terms but we need to see the elimination of irrelevant variables.

```
x=[0:0.01:1]';
y=3*x+4+randn(size(x));
A=[x 5*x-9 8*x+2 pi*x-0.4 x/pi-4];
b=y;
mdl=stepwiselm(A,y)
yhat = predict(mdl,A);
plot(x,y,'or',x,yhat,'xb')


1. Adding x1, FStat = 42.6862, pValue = 2.80092e-09

mdl =


Linear regression model:
    y ~ 1 + x1

Estimated Coefficients:
                   Estimate      SE       tStat       pValue
                   ________    _______    ______    __________

    (Intercept)      4.4669     0.19963    22.375    1.6562e-40
    x1               2.2535     0.34491    6.5335    2.8009e-09


Number of observations: 101, Error degrees of freedom: 99
Root Mean Squared Error: 1.01
R-squared: 0.301,  Adjusted R-Squared: 0.294
F-statistic vs. constant model: 42.7, p-value = 2.8e-09
```

As seen above, only x1 variable is chosen and we see that **stepwiselm** is able to eliminate the linearly combined inputs (predictor).

4.6.2 Some Inputs are Irrelevant

In this section, we will study the cases where there are irrelevant variables. **We will first consider `fitlm` command then `stwpwiselm` command** to generate the model and we will consider different noise levels.

```
x=[0:0.01:1]';
y=3*x+4;
A=[x x.^2];% x1=x and x2=x.^2
b=y;
mdl=fitlm(A,y)
yhat = predict(mdl,A);
plot(x,y,'or',x,yhat,'xb')


mdl =


Linear regression model:
    y ~ 1 + x1 + x2

Estimated Coefficients:
                    Estimate          SE           tStat       pValue
                   ___________    __________    ___________    ______

    (Intercept)              4    1.9938e-08     2.0062e+08       0
    x1                       3    9.2149e-08     3.2556e+07       0
    x2             -9.7831e-16    8.9168e-08    -1.0971e-08       1


Number of observations: 101, Error degrees of freedom: 98
Root Mean Squared Error: 6.81e-08
R-squared: 1,  Adjusted R-Squared: 1
F-statistic vs. constant model: 8.33e+15, p-value = 0
```

```
x=[0:0.01:1]';
y=3*x+4+0.01*randn(size(x));
A=[x x.^2];
b=y;
mdl=fitlm(A,y)
yhat = predict(mdl,A);
plot(x,y,'or',x,yhat,'xb')


mdl =


Linear regression model:
    y ~ 1 + x1 + x2

Estimated Coefficients:
                   Estimate        SE          tStat        pValue
                   _________    _________    ________    ___________

    (Intercept)        3.996    0.0027964        1429    1.9156e-213
    x1                3.0144     0.012924      233.24    2.4726e-136
    x2              -0.011876     0.012506    -0.94962        0.34464


Number of observations: 101, Error degrees of freedom: 98
Root Mean Squared Error: 0.00955
R-squared: 1,  Adjusted R-Squared: 1
F-statistic vs. constant model: 4.24e+05, p-value = 1.2e-193
```

```
x=[0:0.01:1]';
y=3*x+4+0.1*randn(size(x));
A=[x x.^2];
b=y;
mdl=fitlm(A,y)
yhat = predict(mdl,A);
plot(x,y,'or',x,yhat,'xb')


mdl =


Linear regression model:
    y ~ 1 + x1 + x2

Estimated Coefficients:
                   Estimate        SE         tStat        pValue
                   _________    ________    ________    ___________

    (Intercept)       4.0082    0.027433      146.11    1.7419e-116
    x1                2.9818     0.12679      23.518     4.3747e-42
    x2              -0.014226     0.12269    -0.11595        0.90793


Number of observations: 101, Error degrees of freedom: 98
Root Mean Squared Error: 0.0937
R-squared: 0.989,  Adjusted R-Squared: 0.989
F-statistic vs. constant model: 4.3e+03, p-value = 3.33e-96
```

```
x=[0:0.01:1]';
y=3*x+4+randn(size(x));
A=[x x.^2];
b=y;
mdl=fitlm(A,y)
yhat = predict(mdl,A);
plot(x,y,'or',x,yhat,'xb')


mdl =


Linear regression model:
    y ~ 1 + x1 + x2

Estimated Coefficients:
                   Estimate      SE         tStat       pValue
                   ________    _______    _______    __________

    (Intercept)     4.2008     0.27344     15.363    7.6558e-28
    x1              2.4028      1.2637     1.9014      0.060193
    x2              0.28205     1.2229    0.23065       0.81807


Number of observations: 101, Error degrees of freedom: 98
Root Mean Squared Error: 0.934
R-squared: 0.42,  Adjusted R-Squared: 0.408
F-statistic vs. constant model: 35.5, p-value = 2.56e-12
```

As seen abovei **`fitlm` command considers the third term in the model** and generates the optimal solution under the presence of this term. As the noise level increases, the coefficient of this term gets larger; **however, this one is an irrelevant term**. Let's consider the following case.

```
x=[0:0.01:1]';
y=3*x+4;
A=[x x.^2];
b=y;
mdl=stepwiselm(A,y)
yhat = predict(mdl,A);
plot(x,y,'or',x,yhat,'xb')

1. Adding x1, FStat = 1.682084565218227e+16, pValue = 0

mdl =


Linear regression model:
    y ~ 1 + x1

Estimated Coefficients:
                   Estimate        SE          tStat       pValue
                   ________    __________    __________    ______

    (Intercept)       4        1.3388e-08    2.9877e+08      0
    x1                3        2.3131e-08     1.297e+08      0


Number of observations: 101, Error degrees of freedom: 99
Root Mean Squared Error: 6.78e-08
R-squared: 1,  Adjusted R-Squared: 1
F-statistic vs. constant model: 1.68e+16, p-value = 0
```

```
x=[0:0.01:1]';
y=3*x+4+0.01*randn(size(x));
A=[x x.^2];
b=y;
mdl=stepwiselm(A,y)
yhat = predict(mdl,A);
plot(x,y,'or',x,yhat,'xb')


1. Adding x1, FStat = 701137.3943, pValue = 2.073517901e-192

mdl =


Linear regression model:
    y ~ 1 + x1

Estimated Coefficients:
                   Estimate       SE         tStat       pValue
                   ________    _________    ______    ___________

    (Intercept)     3.9991      0.002074    1928.2     2.853e-228
    x1              3.0004     0.0035833    837.34    2.0735e-192


Number of observations: 101, Error degrees of freedom: 99
Root Mean Squared Error: 0.0105
R-squared: 1,  Adjusted R-Squared: 1
F-statistic vs. constant model: 7.01e+05, p-value = 2.07e-192
```

```
x=[0:0.01:1]';
y=3*x+4+0.1*randn(size(x));
A=[x x.^2];
b=y;
mdl=stepwiselm(A,y)
yhat = predict(mdl,A);
plot(x,y,'or',x,yhat,'xb')


1. Adding x1, FStat = 7343.7862, pValue = 1.0937252e-94

mdl =


Linear regression model:
    y ~ 1 + x1

Estimated Coefficients:
                   Estimate       SE       tStat       pValue
                   ________    ________    ______    __________

    (Intercept)     3.9753     0.020523     193.7    1.6014e-129
    x1              3.0386     0.035458    85.696     1.0937e-94


Number of observations: 101, Error degrees of freedom: 99
Root Mean Squared Error: 0.104
R-squared: 0.987,  Adjusted R-Squared: 0.987
F-statistic vs. constant model: 7.34e+03, p-value = 1.09e-94
```

```
x=[0:0.01:1]';
y=3*x+4+randn(size(x));
A=[x x.^2];
b=y;
mdl=stepwiselm(A,y)
yhat = predict(mdl,A);
plot(x,y,'or',x,yhat,'xb')


1. Adding x1, FStat = 98.8215, pValue = 1.47207e-16

mdl =


Linear regression model:
    y ~ 1 + x1

Estimated Coefficients:
                   Estimate      SE       tStat       pValue
                   ________    _______    ______    __________

    (Intercept)        3.8     0.19596    19.391    1.7359e-35
    x1              3.3657     0.33857    9.9409    1.4721e-16


Number of observations: 101, Error degrees of freedom: 99
Root Mean Squared Error: 0.992
R-squared: 0.5,  Adjusted R-Squared: 0.494
F-statistic vs. constant model: 98.8, p-value = 1.47e-16
```

4.6.3 Both Irrelevant Inputs and Linearly Combined Inputs are Available

Lastly, in this section, we will **consider the cases where both linearly combined inputs and irrelevant inputs are available**. We will consider the process $y(x)=4+3x$ with no noise and we will select the following model.

$$\hat{y}(x)=\hat{\beta}_0+\hat{\beta}_1 x+\underbrace{\hat{\beta}_2 x^2+\hat{\beta}_3 x^3+\hat{\beta}_4\sin(2\pi x)}_{\text{Irrelevant}}+\underbrace{\hat{\beta}_5\left(\frac{x}{\pi}-7\right)+\hat{\beta}_6(-x-13)+\hat{\beta}_7(7x+0.2)}_{\text{Linearly dependent}} \tag{4.27}$$

We are expecting to see our approach eliminates the linearly combined inputs and irrelevant inputs. The Matlab code and its results are given below.

```
x=[0:0.01:1]';
y=3*x+4+0.01*randn(size(x));
A=[x x.^2 x.^3 sin(2*pi*x) x/pi-7 -x-13 7*x+0.2];
b=y;
mdl=stepwiselm(A,y)
yhat = predict(mdl,A);
plot(x,y,'or',x,yhat,'xb')


1. Adding x1, FStat = 849011.8776, pValue = 1.596815386e-196

mdl =


Linear regression model:
    y ~ 1 + x1

Estimated Coefficients:
                   Estimate        SE         tStat       pValue
                   ________    _________    _______    ____________

    (Intercept)      3.9998     0.0018837    2123.4     2.0429e-232
    x1               2.9988     0.0032545    921.42     1.5968e-196


Number of observations: 101, Error degrees of freedom: 99
Root Mean Squared Error: 0.00954
R-squared: 1,  Adjusted R-Squared: 1
F-statistic vs. constant model: 8.49e+05, p-value = 1.6e-196
```

According to the results seen above, coefficients of *intercept* and x1 variables are determined correctly. **An expert seeing this picture can propose a simpler and better model**.

It is of course possible to obtain **different results after different runnings of the code**. For example, following two cases generate different results although the settings are kept the same.

```
1. Adding x5, FStat = 1052696.677, pValue = 3.8102662682e-201

mdl =


Linear regression model:
    y ~ 1 + x5

Estimated Coefficients:
                   Estimate       SE        tStat       pValue
                   ________    _________    ______    ___________

    (Intercept)      69.99       0.06286    1113.4    1.1644e-204
    x5               9.4271     0.0091881      1026    3.8103e-201


Number of observations: 101, Error degrees of freedom: 99
Root Mean Squared Error: 0.00857
R-squared: 1,  Adjusted R-Squared: 1
F-statistic vs. constant model: 1.05e+06, p-value = 3.81e-201
```

```
1. Adding x1, FStat = 824543.7002, pValue = 6.789463074e-196
2. Adding x2, FStat = 5.2812, pValue = 0.023682

mdl =


Linear regression model:
    y ~ 1 + x1 + x2

Estimated Coefficients:
                   Estimate       SE        tStat       pValue
                   ________    _________    ______    ___________

    (Intercept)      4.0046     0.0027737    1443.8    6.9729e-214
    x1               2.9712      0.012819    231.77    4.5796e-136
    x2              0.028507     0.012405    2.2981       0.023682


Number of observations: 101, Error degrees of freedom: 98
Root Mean Squared Error: 0.00948
R-squared: 1,  Adjusted R-Squared: 1
F-statistic vs. constant model: 4.3e+05, p-value = 5.92e-194
```

This short discussion shows how to use the commands **`fitlm`** and **`stepwiselm`** and their benefits. For more information about the command **`stepwiselm`**, the reader is referred to [7].

5. Conclusions

Least squares method is a very popular method, which is used very frequently in **fitting a linear model to a given set of data**. After the visualization of data, content of it, dimensionality and outliers can be detected and a suitable model can be proposed. In order to exploit LS method in modeling, **the model output must be linear in its parameters.**

In this report, the derivation of LS method has been studied with a number of examples, obtaining the models, effect of noise and the number of observations have been discusses. The model performance has been evaluated in each case.

The readers looking for more details are referred to [4] or the lecture series in [5].

6. References

1. Widrow, B., & Hoff, M. E. (1960). Adaptive switching circuits. Paper presented at the WESCON Convention Record Part IV.
2. Widrow, B. (1959). Adaptive sampled-data systems—a statistical theory of adaptation. Paper presented at the IRE Wescon Convention Record.
3. Widrow, B. (1960). Adaptive sampled-data systems. Paper presented at the Proceedings of the First International Congress of the International Federation of Automatic Control.
4. Strang, G. (2016). Introduction to linear algebra. 5th Ed. Wellesley-Cambridge Press.
5. https://www.youtube.com/watch?v=ZK3O402wf1c&list=PL49CF3715CB9EF31D&index=1 (10.10.2020).
6. https://www.mathworks.com/help/stats/coefficient-of-determination-r-squared.html (23.10.2020)
7. Matlab Statistics and Machine Learning Toolbox User's Guide, R2019a.

7. Thanks

I would like to thank the students of CMP684 Neural Networks, VBM655 Statistical Data Analysis and VBM622 Flexible Computing who contributed to the maturation of this document.