In The Name of God

First Homework

Static Linear systems estimation

Student:

Mohamad Alikhani

Professor:

Dr. Mehdi Aliyari Shoorehdeli

Fall of 2023

Contents

Analytical Question

Analytical Question 1

Analytical Question 1.A

Unlike typical LS algorithm, the RLS algorithm uses Parameter regularization to regularize the growth of parameters.

LS algorithm is as below:

$$\underline{\hat{\theta}} = (\mathrm{X^T X})^{-1}\mathrm{X^T y}$$

The RLS algorithm's parameter estimation is:

$$\theta = (U^T U + \alpha I)^{-1} U^T y$$

α is the regularization parameter

And the S.S.E cost function is this:

$$E = y - \hat{y}$$

$$J_{\hat{\theta}} = \min\ EE^T$$

Type equation here.

Analytical Question 2

$$H = diag(80, 0, 0.1, 6)$$

This is the hessian matrix we have. First we analyze what this matrix mean:

With regard to Least Square:

$$\underline{\hat{\theta}} = \left(\underline{X}^{\mathrm{T}}\underline{X}\right)^{-1}\underline{X}^{\mathrm{T}}\underline{y}$$

The hessian matrix is defined as below:

$$\underline{H} = \frac{\partial^2 I(\underline{\theta})}{\partial \underline{\theta}^2} = \underline{X}^{\mathrm{T}}\underline{X}$$

Therefore the cost function J is defined as:

$$I(\underline{\theta}) = \frac{1}{2}\underline{\theta}^{\mathrm{T}}\underline{H\theta} + \underline{h}^{\mathrm{T}}\underline{\theta} + h_0$$

The condition for hessian matrix is defined as:

$$\chi = \frac{\lambda_{\max}}{\lambda_{\min}}$$

The covariance of the estimated parameters is:

$$\mathrm{cov}\{\underline{\hat{\theta}}\} = \mathrm{E}\left\{(\underline{\hat{\theta}} - \mathrm{E}\{\underline{\hat{\theta}}\})(\underline{\hat{\theta}} - \mathrm{E}\{\underline{\hat{\theta}}\})^{\mathrm{T}}\right\}$$

Pay attention that $\mathrm{E}\{\underline{\hat{\theta}}\} = \left(\underline{X}^{\mathrm{T}}\underline{X}\right)^{-1}\underline{X}^{\mathrm{T}}\mathrm{E}\{\underline{y}\}$, after simplification, we will have:

$$\mathrm{cov}\{\hat{\theta}\} = \sigma^2\left(\underline{X}^{\mathrm{T}}\underline{X}\right)^{-1} = \sigma^2\underline{HH}^{-1}$$

If the hessian matrix is singular, or near singular, the inverse of it will be a matrix with huge values that makes the covariance matrix go of the chart, which results in very large estimations for system parameters. One of the solutions to this problem is using a regularization term, which makes the condition number, small.

Analytical Question 2.A

Because the hessian matrix is singular (it is diagonal and has a zero in it), the answer is no. It is not a unique answer.

Analytical Question 2.B

In case of better accuracy, we must

Now in this question, we have:

$$H = diag(80, 0, 0.1, 6)$$

Then for each parameter, we have calculate the variance of parameters with respect to each other:

$$\text{var}\{\hat{\theta}_1\} = \frac{1}{80}\sigma^2$$

$$\text{var}\{\hat{\theta}_2\} = \frac{1}{0}\sigma^2$$

$$\text{var}\{\hat{\theta}_3\} = \frac{1}{0.1}\sigma^2$$

$$\text{var}\{\hat{\theta}_4\} = \frac{1}{6}\sigma^2$$

The more the denominator is bigger, the more the estimation gets.

With the above said, we can say in terms of accuracy, the order is θ_1, θ_4, θ_3 and θ_2. The reason for this is that if we have a big denominator the variance will be smaller, so the estimated values are near each other.

Analytical Question 2.C

$$e = y - \hat{y}$$

$$e = y - X(X^T X)^{-1} y$$

$$e = (X\theta + n) - X(X^T X)^{-1}$$

In the context of least squares error algorithm for parameter estimation, the parameter that is less sensitive to errors is the one for which the partial derivative of the error with respect to that parameter is small. In other words, if a small change in a particular parameter results in only a small change in the error, then that parameter is less sensitive to errors.

Mathematically, for a system of equations represented by $Ux = y$, where U is the matrix of coefficients, x is the vector of parameters to be estimated, and y is the

vector of outputs, the least squares solution minimizes the sum of squared differences between the observed and predicted values. The solution is given by:

$$\underline{\hat{\theta}} = (\mathrm{X}^{\mathrm{T}}\mathrm{X})^{-1}\mathrm{X}^{\mathrm{T}}y$$

Now, to assess the sensitivity of a parameter, you can look at the elements of the inverse matrix $\left(\underline{X}^{\mathrm{T}}\underline{X}\right)^{-1}$, which happens to be the inverse of the hessian matrix. If an element in the diagonal of this matrix is small, it indicates that the corresponding parameter is less sensitive to errors.

In summary, in the least squares error algorithm, the parameter that is less sensitive to errors can be identified by examining the diagonal elements of the inverse of the matrix $\left(\underline{X}^{\mathrm{T}}\underline{X}\right)$. If an element is small, the corresponding parameter is less sensitive to errors in the observations.

So first, we have to calculate the inverse of the hessian matrix, then based on the representative elements of the main diagonal, decide which parameter has less sensitivity to error:

Here is the the 4x4 matrix defined in matlab:

H =

80.0000	0	0	0
0	0	0	0
0	0	0.1000	0
0	0	0	6.0000

Calculate the inverse and we get this matrix:

H_inverse =

Inf	Inf	Inf	Inf
Inf	Inf	Inf	Inf
Inf	Inf	Inf	Inf
Inf	Inf	Inf	Inf

So all parameters of the matrix are sensitive to error.

Maybe this is the solution:

$$e = y - y$$

$$e = X\theta + n - (X\hat{\theta})$$

Now for assessing the sensitivity of error to variables, we must take the derivative of error with respect to parameters:

$$\frac{\partial e}{\partial \hat{\theta}} = -X$$

We know that we have $X^T X$ which is the hessian matrix so we use the Cholesky decomposition:

If we have a matrix multiplication in form of $A^T A = B$ we have B and we want to find A, you can solve for A by taking the square root of B's inverse. The solution is not unique, as there could be multiple matrices A that satisfy the equation, but one common approach is to use the Cholesky decomposition.

Here's how you can approach this:

1. **Cholesky Decomposition:**
   - Write $B = C^T C$ Where C is the Cholesky factor of B
   - If B is a positive definite matrix, Cholesky decomposition exists, and C is an upper triangular matrix.
2. **Reconstruct A:**
   - Once you have C, you can reconstruct A as $A = C^T$

Here is more detailed explanation:

$$AA^T = B \rightarrow (A^T A)^{\frac{1}{2}} = B^{\frac{1}{2}} = C^T$$

Please note:

The Cholesky decomposition requires B to be a positive definite matrix. **If $\boldsymbol{B}$ is not positive definite, the decomposition is not possible.**

Cholesky decomposition is not unique.

If we give matlab this matrix B:

$$B = \begin{matrix} 80 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & 6 \end{matrix}$$

The result is that, B is not a positive definite matrix, so there will not be a cholesky factor.

But what I think, is that there must be some sort of typo so we are going to assume the element 22 is not zero, but fairly small number.

If we give matlab this matrix B:

$$H = A = \begin{bmatrix} 80 & 0 & 0 & 0 \\ 0 & 0.0001 & 0 & 0 \\ 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & 6 \end{bmatrix}$$

The matrix A or so to say the matrix X in this case will be:

$$X = B = \begin{bmatrix} 8.9443 & 0 & 0 & 0 \\ 0 & 0.0100 & 0 & 0 \\ 0 & 0 & 0.3162 & 0 \\ 0 & 0 & 0 & 2.4495 \end{bmatrix}$$

So Sensitivity of Nosie with respect to estimated parameters is as bellow

$$SE(\theta) = -\begin{bmatrix} 8.9443 & 0 & 0 & 0 \\ 0 & 0.0100 & 0 & 0 \\ 0 & 0 & 0.3162 & 0 \\ 0 & 0 & 0 & 2.4495 \end{bmatrix}$$

The error is most sensitive to elements θ_1, θ_4, θ_3 and θ_2.

So error is least sensitive to θ_2.

Analytical Question 3

Just like a regular LS problem we proceed to solve the problem.

Think of the problem as this:

$$\theta_1 \cdot X_1 \ + \ \theta_2 \cdot X_2 \ \ + \ \theta_3 \cdot X_3 = y$$

and the constraint:

$$\theta_1 = \theta_2 + \theta_3$$

We can rewrite the model in terms of the constraints:

$$(\theta_2 + \theta_3) \cdot X_1 + \theta_2 \cdot X_2 + \theta_3 \cdot X_3 = y$$

Now, let's set up the LS estimation problem. Assuming you have a set of observations $(X_{1i}, X_{2i}, X_{3i}, y_i)$ for $i = 1,2, \dots, n,$ the LS objective function to minimize is:

Minimize $\sum_{i=1}^{n} \left((\theta_2 + \theta_3) \cdot X_{1i} + \theta_2 \cdot X_{2i} + \theta_3 \cdot X_{3i} - y_i \right)^2$

Now, let's denote the design matrix as X and the parameter vector as θ:

$$X = \begin{bmatrix} X_{1,1} & X_{2,1} & X_{3,1} \\ X_{2,2} & X_{2,2} & X_{3,2} \\ \vdots & \vdots & \vdots \\ X_{n,1} & X_{n,2} & X_{n,3} \end{bmatrix}$$

The model equation in matrix form is then:

$$X.\theta = y$$

The LS solution is given by:

$$\hat{\theta} = (X^T X)^{-1} X^T y$$

Now, you can use this formula to estimate the parameters $\hat{\theta}_2$ and $\hat{\theta}_3$. Once you have these parameters estimated, **you can check to see if the constraint holds.**

Keep in mind that the existence and uniqueness of the solution depend on the properties of the input matrix X. **If $X^T X$ is invertible the solution is Unique.**

Just like another ordinary LS estimation Problem we treat this section, don't worry this works fine☺

And for further validation we use the m-file in the respective folder in code section. Even with the presence of noise.

Analytical Question 4

To estimate $x(0)$ and $\left.\frac{dx}{dt}\right|_{t=0}$ for the given second-order linear homogeneous differential equation:

$$\frac{d^2x}{dt^2} - 3\frac{dx}{dt} + 2x = 0$$

we'll need initial conditions. The general solution to this differential equation is of the form:

$$x(t) = Ae^{r_1 t} + Be^{r_2 t}$$

where A and B are constants to be determined, and r_1 and r_2 are the roots of the characteristic equation:

$$r^2 - 3r + 2 = 0$$

Solving this quadratic equation, you'll find two roots r_1 and r_2. The general solution depends on the nature of these roots:

1 If the roots are real and distinct $(r_1 \neq r_2)$, the general solution is:

$$x(t) = Ae^{r_1 t} + Be^{r_2 t}$$

2 If the roots are real and equal $(r_1 = r_2)$, the general solution is:

$$x(t) = (A + Bt)e^{rt}$$

3 If the roots are complex $(r_1, r_2 = \alpha \pm \beta i)$, the general solution is:

$$x(t) = e^{\alpha t}(C_1 \cos(\beta t) + C_2 \sin(\beta t))$$

Once you have the general solution, you can use the initial conditions to determine the values of the constants. If $x(0)$ and $\left.\frac{dx}{dt}\right|_{t=0}$ are given, you can substitute these values into the general solution:

1 For $x(0)$:

$$x(0) = A + B$$

2 For $\left.\frac{dx}{dt}\right|_{t=0}$:

$\left.\frac{dx}{dt}\right|_{t=0} = r_1 A + r_2 B$ (if roots are real and distinct)

Or

$\left.\frac{dx}{dt}\right|_{t=0} = r(A + Bt)$ (if roots are real and equal)

or

$\left.\frac{dx}{dt}\right|_{t=0} = \alpha C_1 + \beta\alpha C_2$ (if roots are complex)

Solving these equations will give you the values of the constants, and you can then substitute them back into the general solution to obtain the specific solution for the given initial conditions.

This is simply an estimation problem that we can easily solve via LS or other variants.

Analytical Question 5

For this question we are using this paper[1]

Filtering is a signal processing operations with diverse objectives [9]. From mathematical point of view, filtering is a function approximation technique. Adaptive filter may be understood as self-modifying digital filter that adjust its coefficients in order to minimize a predefined error function.

Recursive Least Squares is an algorithm used for online parameter estimation in linear regression models. It updates the estimates of model parameters sequentially as new data points become available.

QR-decomposition-based Recursive Least Squares (RLS) is a variant of the Recursive Least Squares algorithm that utilizes QR decomposition to efficiently update the inverse of the covariance matrix during the estimation process.

QR decomposition decomposes a matrix into the product of an orthogonal matrix (Q) and an upper triangular matrix (R). The QR decomposition is utilized in Recursive Least Squares to efficiently update the inverse of the covariance matrix involved in the parameter estimation.

The recursiveness of recursive least squares corresponds to adaptive filtering application. RLS solve adaptive filtering problem in order to compute coefficient vector and associated errors recursively. RLS depend heavily on input signal vector. RLS has excellent performance when working in time varying environment than stationary environment. The acceptance of the RLS algorithm has been impeded by unacceptable numerical performance in limited precision environment.

Why are we going to use QRD-RLS?

The problem with the RLS estimation is, it's computational cost and it's numerical instability.

Due to numerical instability associated with recursive least squares, QR decomposition was proposed. In adaptive filtering, QR decomposition apply time recursive in order to accept input data and desired signal at time instant k. **Conventional QR decomposition transform data matrix to orthogonal and upper triangular matrix and also transform desired signal vector**. QR decomposition

[1] https://paspk.org/wp-content/uploads/proceedings/52,%20No.1/cc8024e7On%20the%20stability.pdf

decomposes the data matrix into orthogonal matrix $Q(k)$ and upper triangular matrix $R(k)$.

The transformation process via QR decomposition reduces the original data matrix to a reduced form. QR decomposition technique is numerically robust and stable because of the norm preserving property for the 2- norm. Unlike RLS algorithms and its variant which are highly computationally intensive and also sometimes show some properties of numerical instability.

Second Part: Implementation Questions

Implementation Question 1

The input-output equation of the system is as follows:

$$y = -1.5 - 0.8u + 0.01u^3 - 0.65u^5 + 2.25u^6 - 1.7u^8 + n$$

As the question asks we know that the arrangement of regressors for an estimation is as follows:

$$\hat{y} = [1 \quad u \quad u^2 \quad u^3 \quad u^4 \quad u^5 \quad u^6 \quad u^7 \quad u^8] \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \\ \theta_5 \\ \theta_6 \\ \theta_7 \\ \theta_8 \end{bmatrix}$$

We can find the code for data generation in the **DataGeneration** m-file. In the first question we use the following convention to generate the data

The syntax for random number generation is as follows:

```
u = normrnd(mu, sigma, m, n)
```

`mu` **:** mean of the mean of the distribution. mean of the distribution = 0.5
`sigma`**:** standard deviation of the distribution we take the standard deviation =1

`m`**:** number of rows = 1000

`n`**:** number of columns = 1

before proceeding any further we normalize the data, using min-max normalization to bring the data to range [0, 1].

```
u = (u - min(u)) / (max(u) - min(u));
```

We want 1000 data points randomly chosen from the range [0,1] and split it 75% for train and 25% for test. After generation the data we can put them in the equation below to get the y without noise:

$$y = -1.5 - 0.8u + 0.01u^3 - 0.65u^5 + 2.25u^6 - 1.7u^8$$

Now it's time to add noise to our data. What should we choose for the noise?

The values we going to use for the **standard deviations** of the noise must be proportional to the amplitude of the regressors. Since our data is in range [0,1] when our regressors are powered to, let's say 5, this number gets smaller and smaller. Our biggest power is 8 so we must acquire the histogram (probability distribution) of this regressor to wisely choose the **std** parameter of noise, for low noise, medium noise and high noise data.

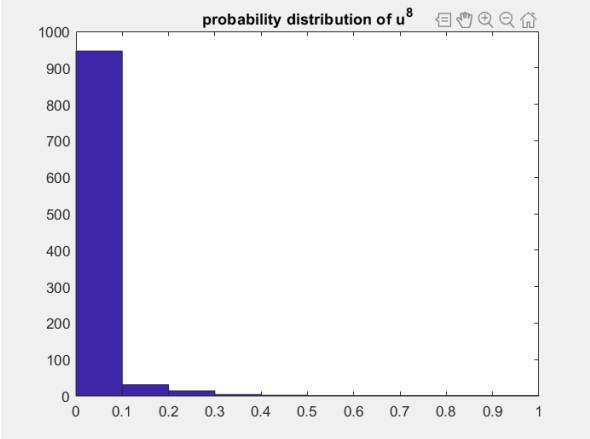


Fig 1) Probability distribution of vector u to the power 8

According to Fig 1, we can see that most of the data points are smaller than 0.1 so we must choose the noise std parameter in a way that be proportional to our data points.

As it's obvious the majority of the data points for u^8 are in the range [0,0.1] so we can deduce that the proper std parameter for the noise is 0.1. Therefore std = 0.1 and variance = 0.01.

The high noise variance would be 0.01

Medium noise variance would be 0.0064

Low noise variance would be 0.0001

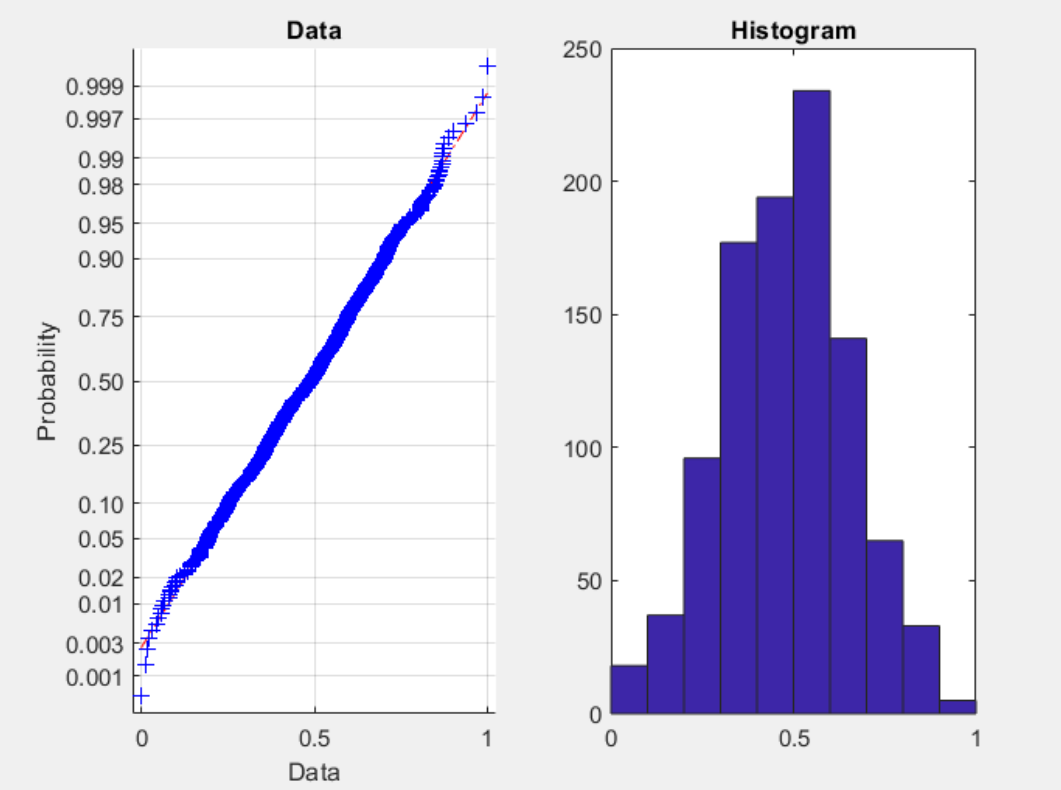


Fig 2) Histogram and normplot of the data u

Using normplot and hist command to see if the data follows normal distribution. We see the data points follow the normal distribution fairly.

The regressor vector we will be using in this estimation is as this:

$$\overline{x^q} = \begin{bmatrix} 1 & u_q & u_q^2 & u_q^3 & u_q^4 & u_q^5 & u_q^6 & u_q^7 & u_q^8 \end{bmatrix}$$

It means that the qth observation is the result of the vector above. And since we must break the data into 10 batches of 100 data points. So the input that we want to carry out the estimation with, is as below:

$$X = \begin{bmatrix} 1 & u_1 & u_1^2 & u_1^3 & u_1^4 & u_1^5 & u_1^6 & u_1^7 & u_1^8 \\ 1 & u_2 & u_2^2 & u_2^3 & u_2^4 & u_2^5 & u_2^6 & u_2^7 & u_2^8 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & u_{100} & u_{100}^2 & u_{100}^3 & u_{100}^4 & u_{100}^5 & u_{100}^6 & u_{100}^7 & u_{100}^8 \end{bmatrix}$$

In the class we used vector U. But here we just stick to Nelles's convention and we consider the input as vector X.

Implementation Question 1.1

Question 1.1.A

The parameter estimation of the Least Square is derived through the following equation:

$$\hat{\theta} = (X^T X)^{-1} X^T Y$$

In this part we use the 10 batches of 100 data points each, then use 75% and 25% respectively for train and test.

We use LS algorithm for all 10 batches and derive the estimated parameters for each batch. We have two approaches.

First approach: Now that we have 10 set of estimated parameters we take the average of these 10 parameter then calculate the error. Second approach is we take each local batch and calculate the error separately. We follow both approaches.

Fist approach: Using the average of estimated values:

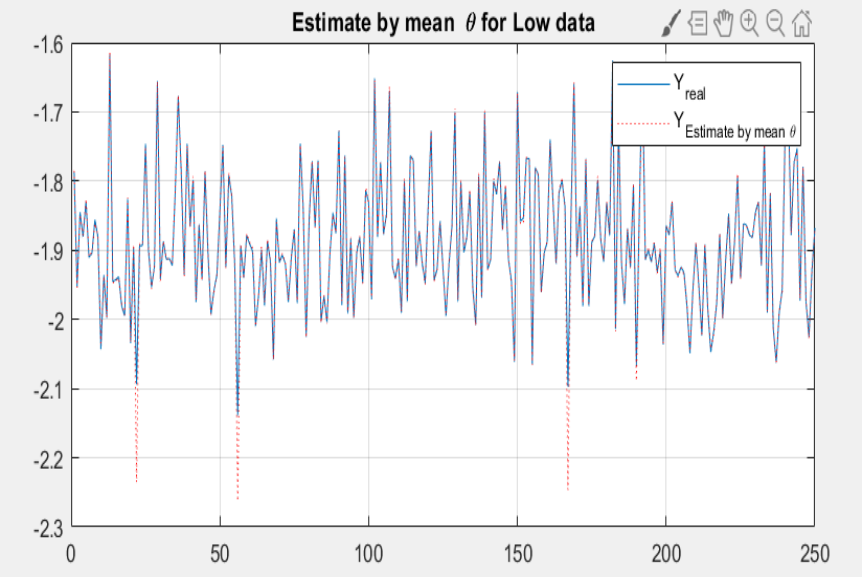


Fig 3) The plot of output data and the actual data using the average estimated parameters for Low noise

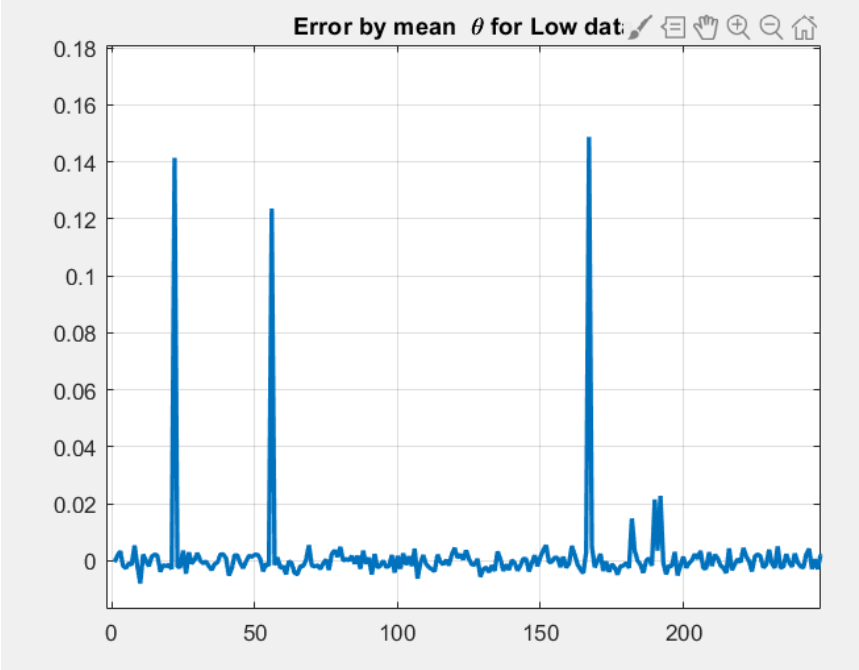


Fig 4) Error plot for the average estimated parameters and Low noise

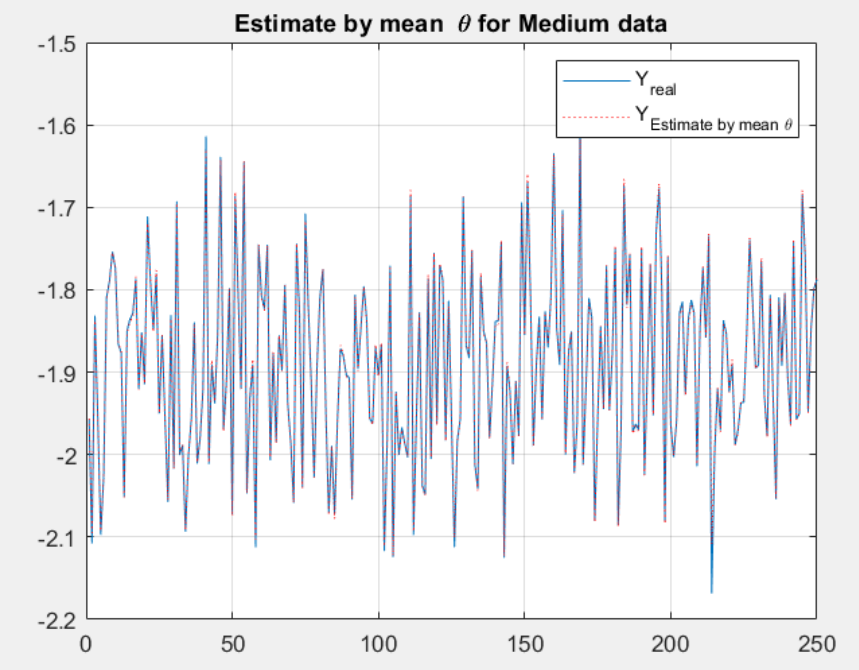


Fig 5) The plot of output data and the actual data using the average estimated parameters for Medium noise

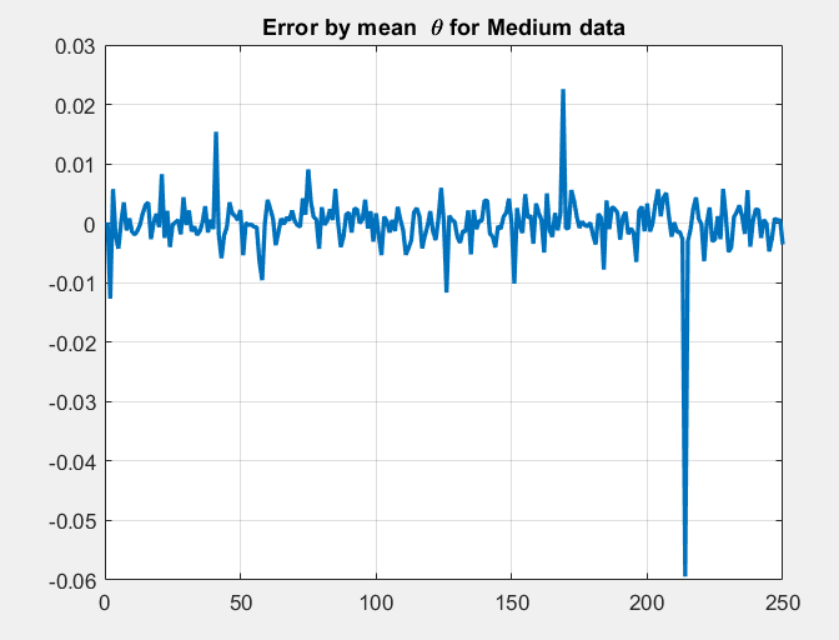


Fig 6) Error plot for the average estimated parameters and Medium noise

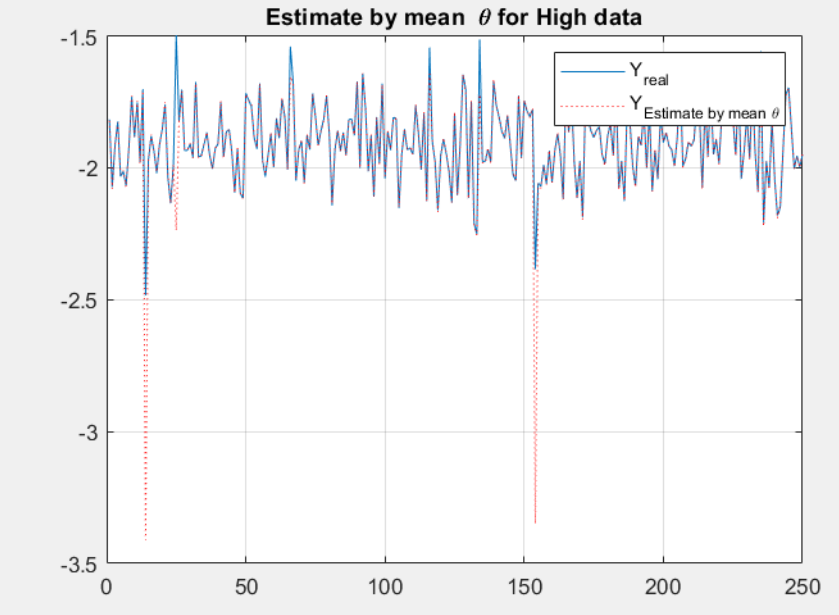


Fig 7) The plot of output data and the actual data using the average estimated parameters for High noise

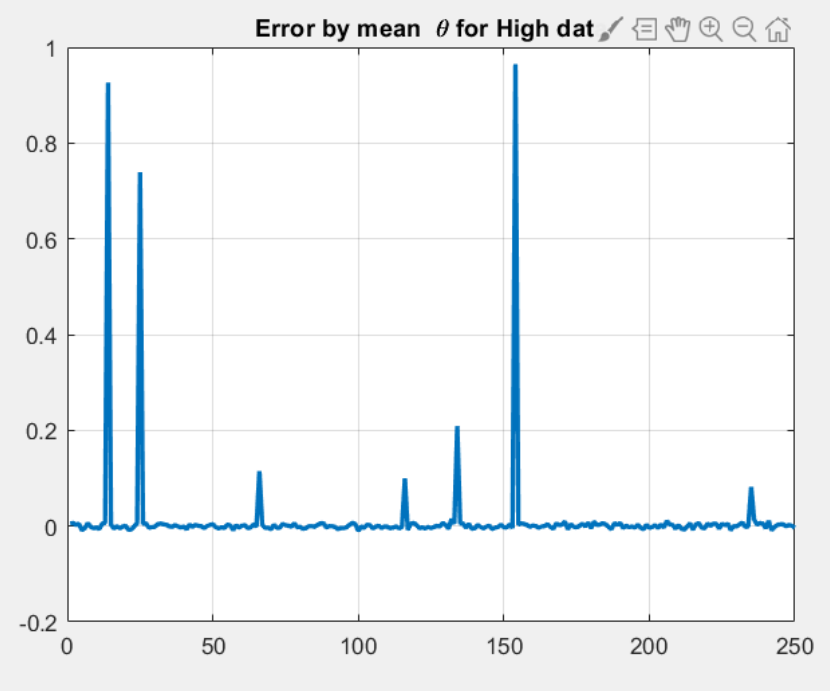


Fig 8) Error plot for the average estimated parameters and High noise

It is obvious from the above figures that the estimated parameters can track the actual signal pretty well. And it yields to acceptable errors.

theta =

1.0e+03 *

-0.004957838294645
0.041622194969551
-0.257522050313723
0.899273400482761
-1.924349336846286
2.588117362236736
-2.132719169389619
0.982016877433692
-0.192988401048342

Error_local theta Low=1.213594e+00
Error_mean theta Low=6.024648e-02

theta =

1.0e+03 *

```
  0.001120339504442
 -0.057943658197652
  0.426450342401572
 -1.705406654988364
  4.092832039768585
 -6.028227870221706
  5.322563276119308
 -2.576757516726401
  0.523513291504754

Error_local theta Medium=5.385833e-03
Error_mean theta Medium=6.571766e-03


theta =

  1.0e+03 *

 -0.003410877112712
  0.022411138983187
 -0.190700376527976
  0.926853287935961
 -2.689750562224508
  4.774743406888127
 -5.077498327961654
  2.970148182032851
 -0.735423281185962

Error_local theta High=2.521139e+00
Error_mean theta High=2.414458e+00
```

It's obvious that the variance of noise has a very big impact on the estimation process and the accumulative error. As we increase the variance the error increases and we get a worse estimation. Our Goal in LS estimation is to make the error look like the noise.

Let's look at the local estimates from each batch:

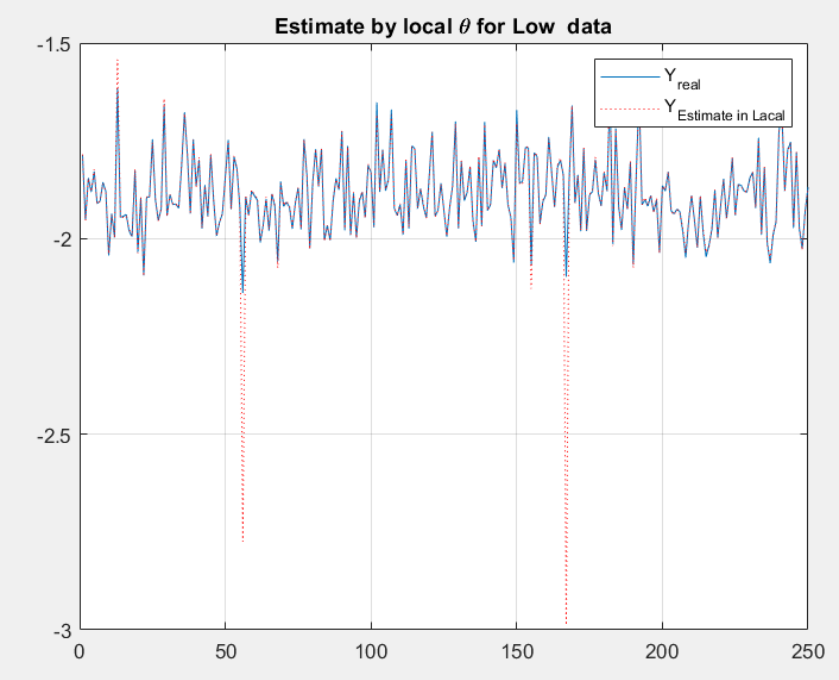


Fig 9) The plot of output data and the actual data using the local estimated parameters for Low noise

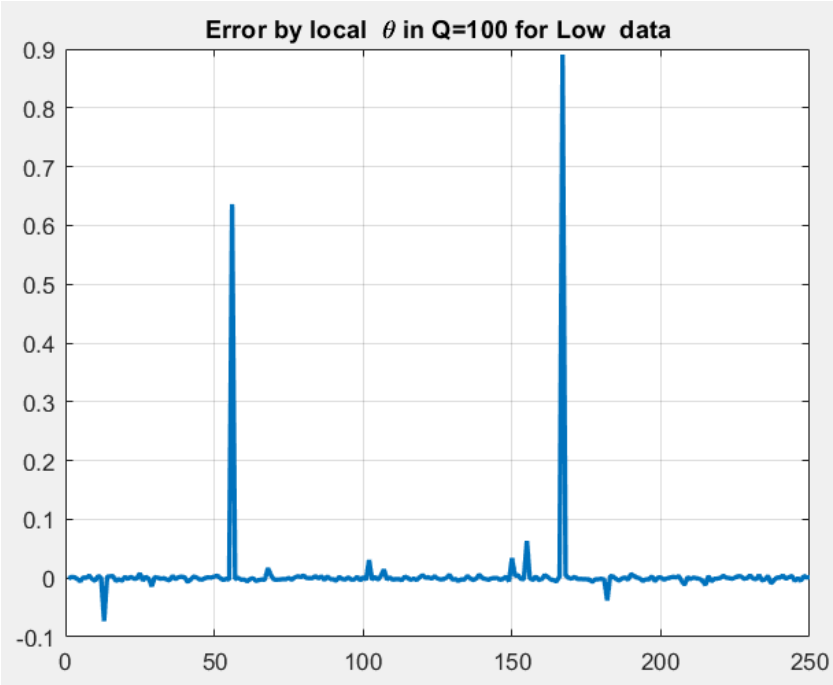


Fig 10) Error plot for the average estimated parameters and Low noise

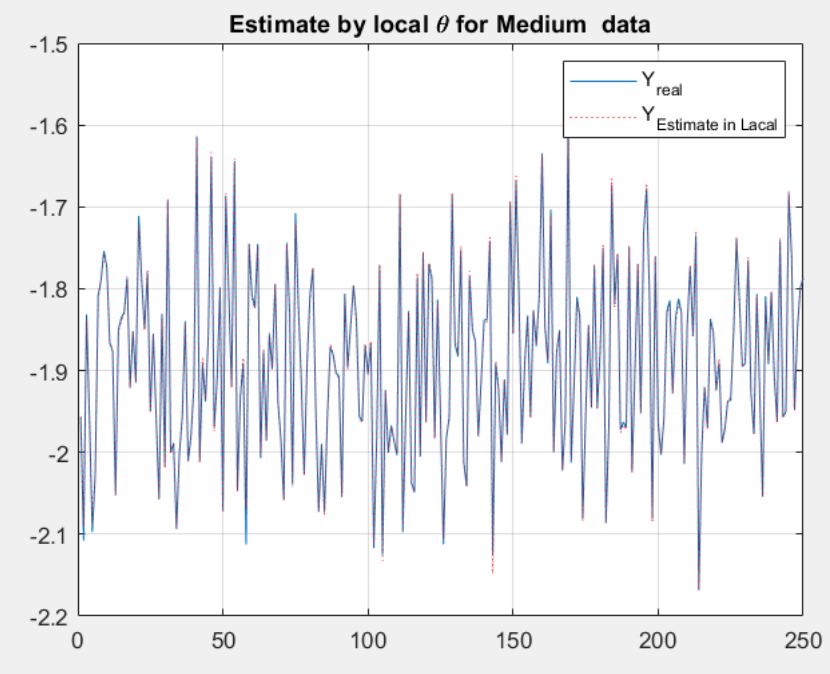


Fig 11) The plot of output data and the actual data using the local estimated parameters for Medium noise

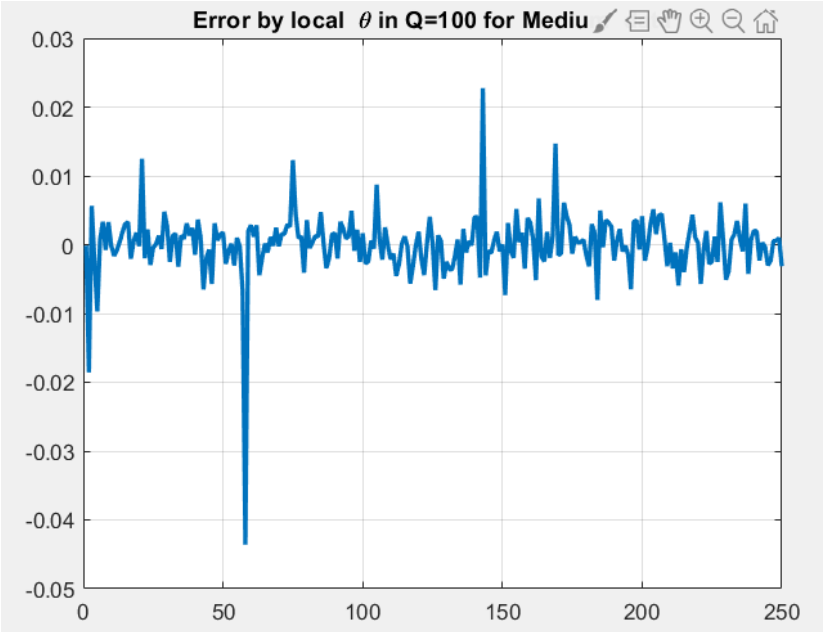


Fig 12) Error plot for the average estimated parameters and Medium noise

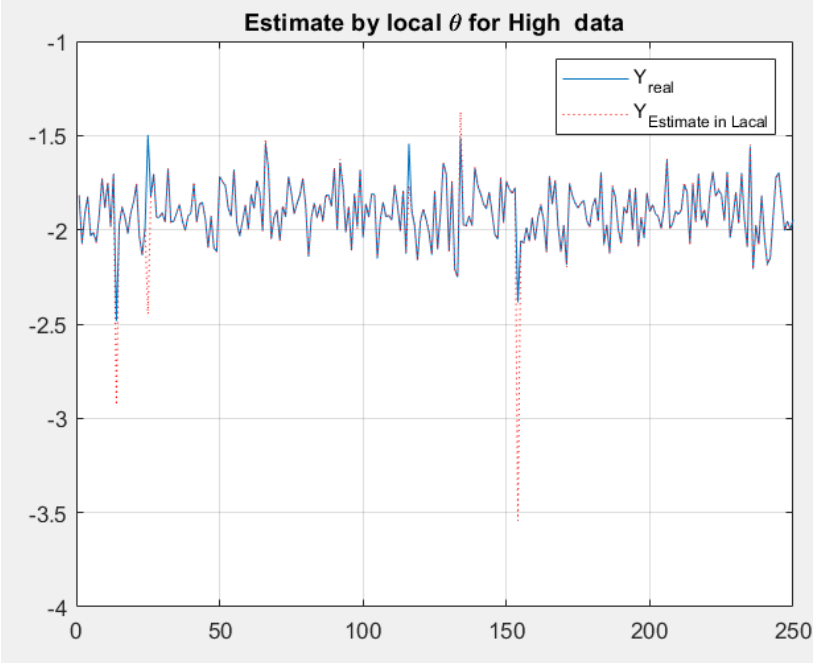


Fig 13) The plot of output data and the actual data using the local estimated parameters for High noise

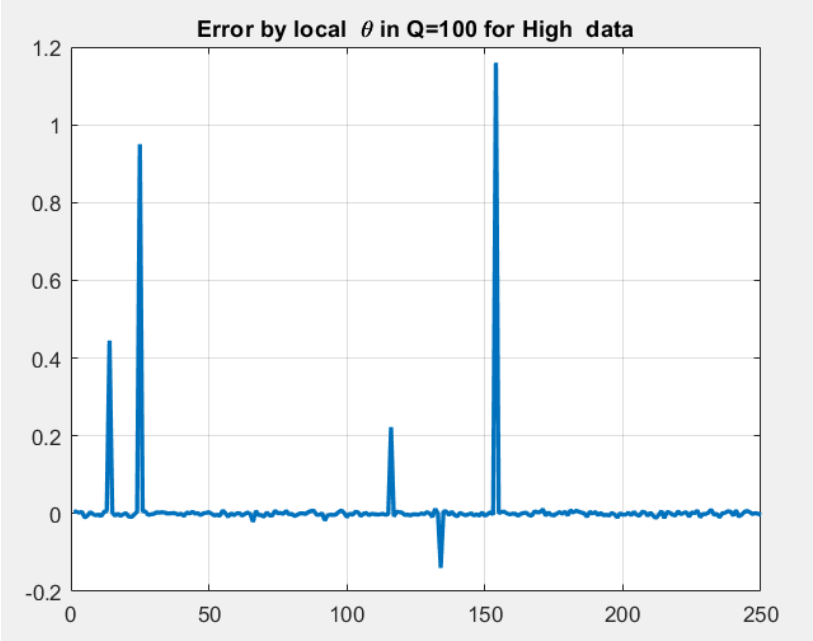


Fig 14) Error plot for the average estimated parameters and High noise

The Second approach yields to better results, but we tend to use the first approach (average parameters) so we will have a unit estimate.

Question 1.1.B

An error bar shows the degree of confidence in the estimation. Actually this graph is a bound that makes sure the estimated output ($\hat{y}$) is in it. In addition to that shows the degree of confidence we have for our estimate in different sections of the estimation. This bound is defined by $y \pm \sigma_y$. In sections where this bound is wide the estimate has some error and we have a lower degree of confidence in our estimate. In places where the bound is narrow we have higher degrees of confidence in our estimate. For error bar we need the standard deviation of the output (σ_y). But since we don't have this value we use it's estimated value ($\hat{\sigma}_y$). We have two choices for that one is the standard deviation of the output and the other is the use of $\hat{y} \pm \sqrt{diag(cov(\hat{y}))}$. In this section we try to solve the problem using the error bars:

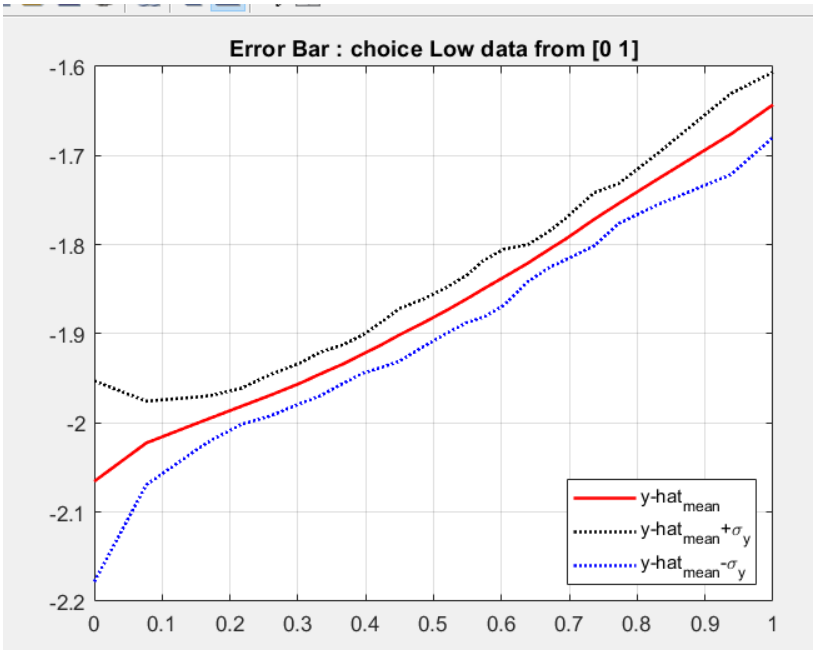


Fig 15) Error bar plot for the average estimated parameters and Low noise variance

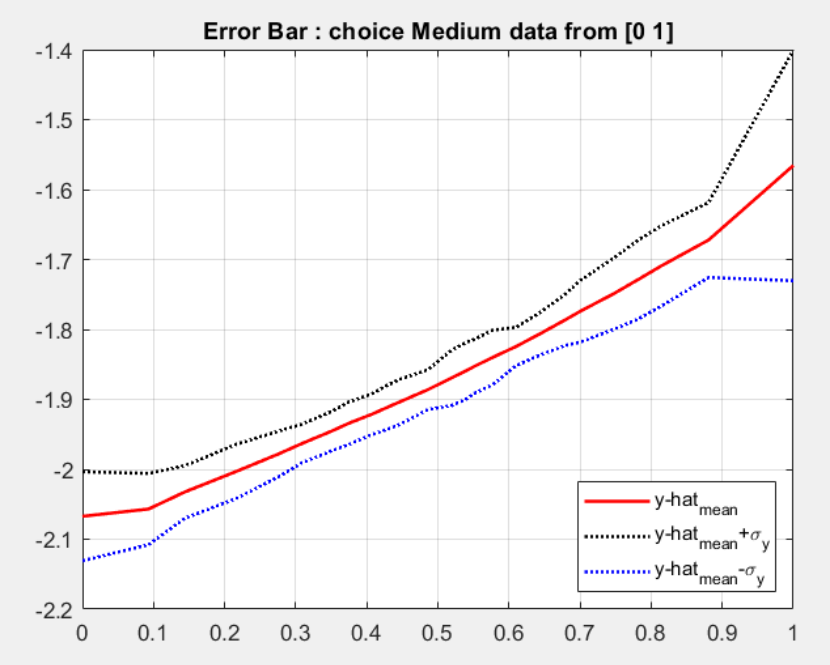


Fig 16) Fig 15) Error bar plot for the average estimated parameters and Medium noise variance

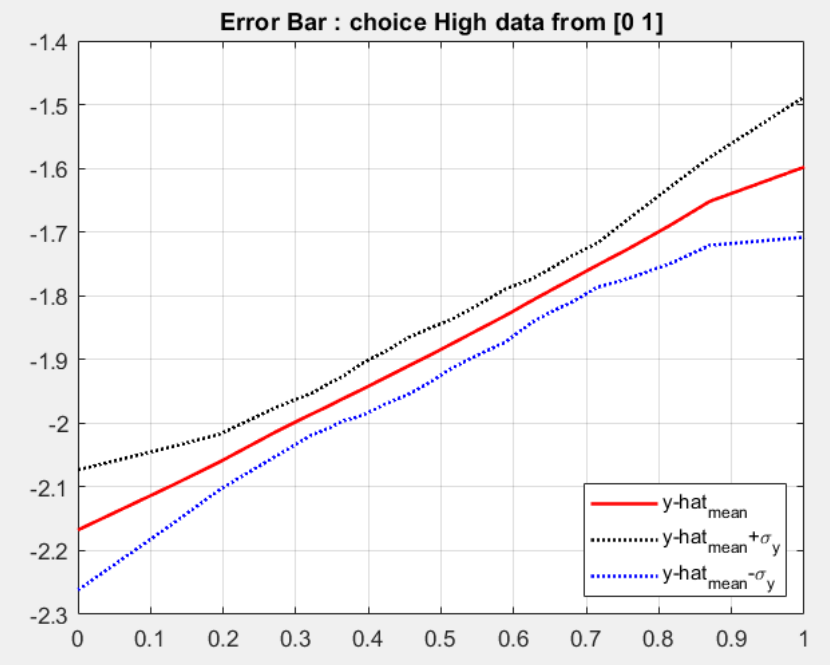


Fig 17) Fig 15) Error bar plot for the average estimated parameters and High noise variance

The more we are to the middle of the graph, the more confident our estimate gets. Towards the end and the start of the estimate we witness the error bar gets wider, this is because the lack of data.

Question 1.1.C

For this purpose we resolve the first part, two times. First with 2000 data points, and second with 600 data points, and observe what the outcome is.

First scenario: Estimation with more data points:

```
theta =


   1.0e+03 *

  -0.003079513859527
   0.018357835642063
  -0.140053749238562
   0.600333202990130
  -1.528952376440327
   2.340083454012819
  -2.053412294817104
   0.904456829825761
  -0.136308660522514

Error_local theta Low=1.065579e+01
Error_mean theta Low=1.011393e+01


theta =


   1.0e+03 *

  -0.002744400852214
   0.013670675123588
  -0.129497667773923
   0.677778019862779
  -2.048308077521618
   3.685524002536223
  -3.872950255809629
   2.182213055337740
  -0.506458752481669

Error_local theta Medium=1.533853e-02
Error_mean theta Medium=4.117325e-01
```

theta =

1.0e+03 *

-0.002838200410891
0.013186245887240
-0.116494814111368
0.591761382232860
-1.760735795700553
3.151319814640828
-3.324936843723163
1.899909372472049
-0.452761970290758

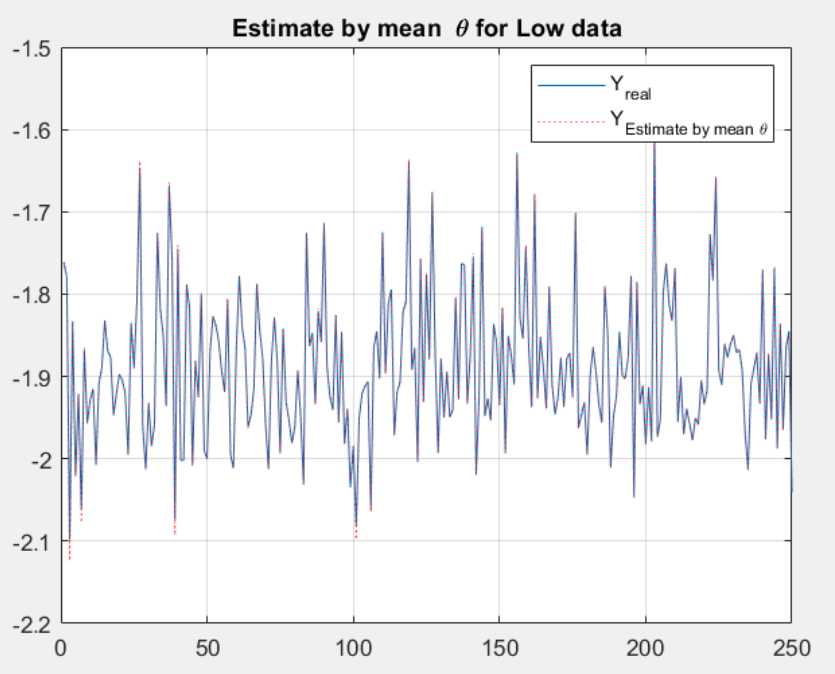
Error_local theta High=6.450151e-01
Error_mean theta High=7.215433e-02

From the observation of error we can see the table below to draw a conclusion

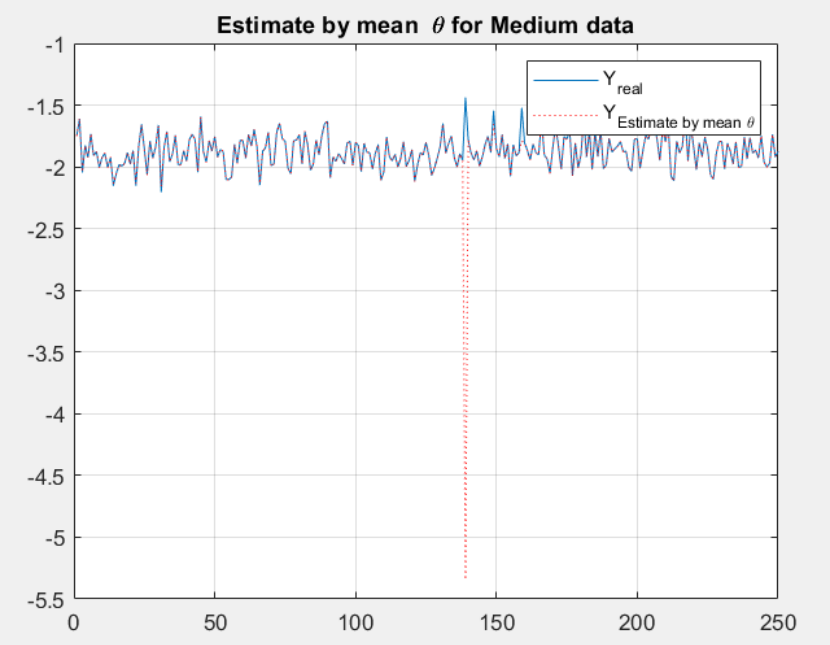
1000 data points	2000 data points	5000
Error_local theta Low=1.213594e+00 Error_mean theta Low=6.024648e-02	Error_local theta Low=1.065579e+01 Error_mean theta Low=1.011393e+01	Error_local theta Low=2.451076e-01 Error_mean theta Low=1.513441e-01
Error_local theta Medium=5.385833e-03 Error_mean theta Medium=6.571766e-03	Error_local theta Medium=1.533853e-02 Error_mean theta Medium=4.117325e-01	Error_local theta Medium=1.105975e+00 Error_mean theta Medium=7.586666e+00
Error_local theta High=2.521139e+00 Error_mean theta High=2.414458e+00	Error_local theta High=6.450151e-01 Error_mean theta High=7.215433e-02	Error_local theta High=2.996762e+00 Error_mean theta High=1.692329e-01

In the table above we see that increasing the number of data points does not show consistent behavior, in some cases it reduced the error and in some cases it decreases the error.

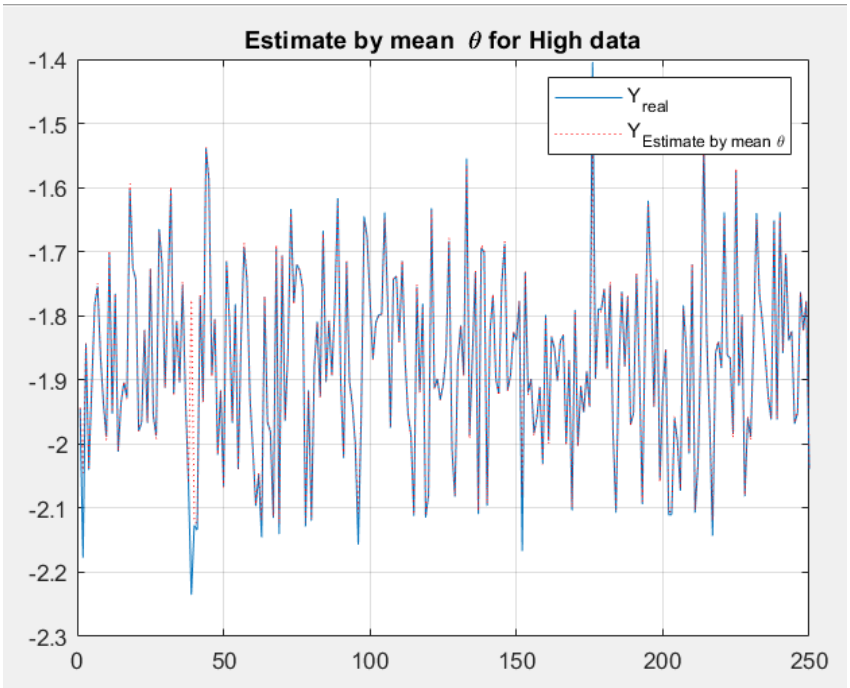
What is surprising is that in the from the perspective of error, cases with low noise density became worse as we can see from above and the first part errors are bigger than the two other densities(Medium and High)



The plot of output data and the actual data using the mean estimated parameters for low noise (More data)



The plot of output data and the actual data using the mean estimated parameters for Medium noise (more data)



The plot of output data and the actual data using the mean estimated parameters for high noise (more data)

Second Scenario: Now we decrease the number of observations to 600 and run the algorithm to see the results. Bear in mind that since we have 600 data points, we need to decrease the number of partitions from 10 to 6 as well.

theta =

1.0e+02 *

-0.021335163360169
0.012090361851441
0.060677507416987
-1.084664945561931
4.871046324747729
-8.967525559609834
5.536421035055731
2.361603739192816
-2.934030414591819

Error_local theta Low=9.655442e-01
Error_mean theta Low=8.329777e-01

theta =

1.0e+03 *

-0.002228751423853
0.000475512389707
0.030771112576235
-0.361366166563181
1.812552253432884
-4.734520966975715
6.742554291614477
-4.966144017142600
1.482084711115986

Error_local theta Medium=2.043894e-03
Error_mean theta Medium=1.877649e-03

theta =

1.0e+03 *

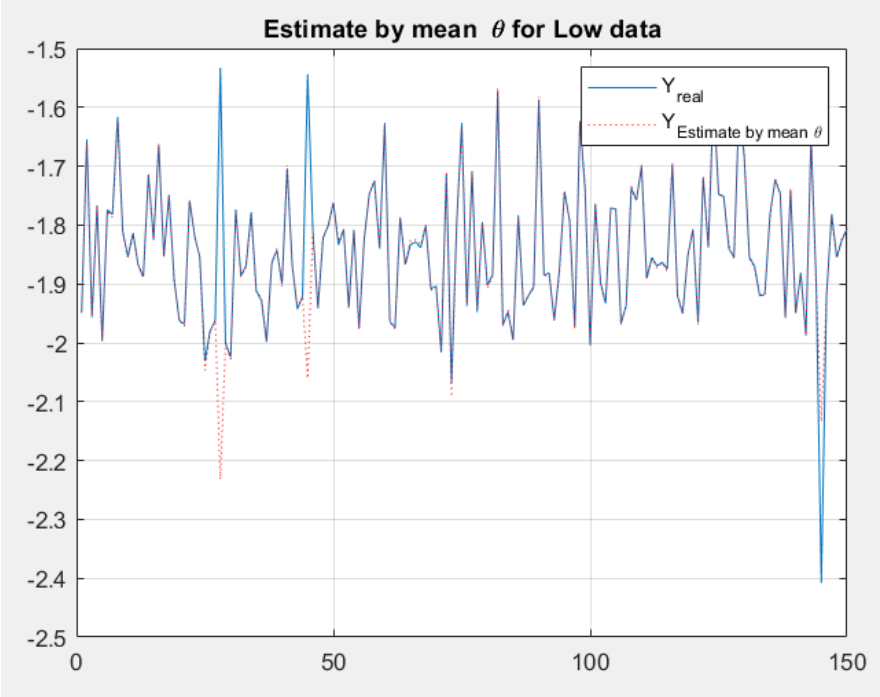
-0.002534758333481
0.008782001463899
-0.070162810368477
0.310293985682775
-0.819517369815353
1.448891763576333
-1.774798121493813
1.358171622318301
-0.468808265955259

Error_local theta High=2.079608e-01
Error_mean theta High=6.193502e-02

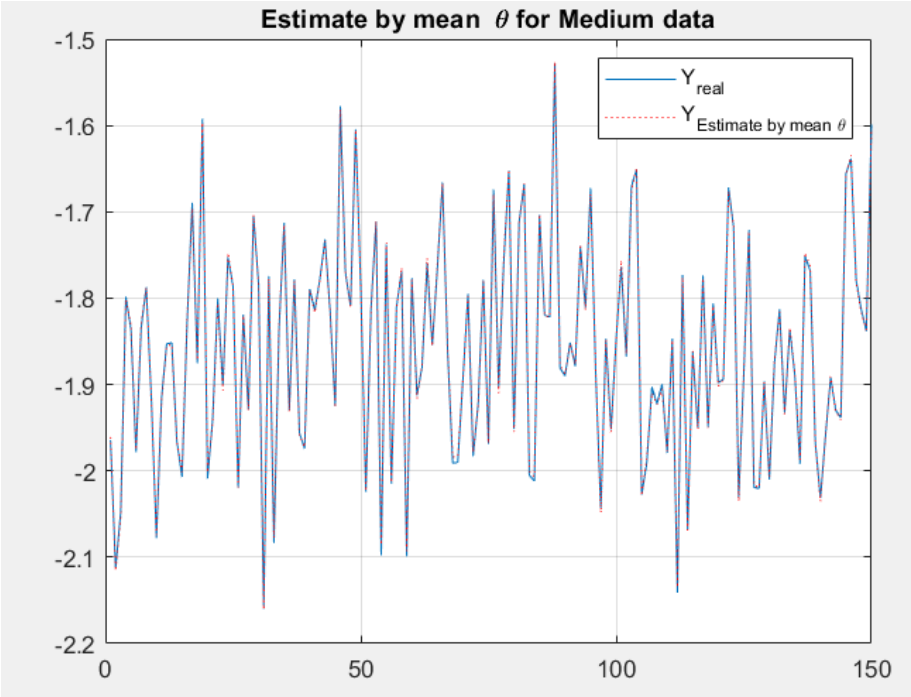
Table 1) the comparison between smaller observations and

1000 data points	600 data points
Error_local theta Low=1.213594e+00 Error_mean theta Low=6.024648e-02	Error_local theta Low=9.655442e-01 Error_mean theta Low=8.329777e-01
Error_local theta Medium=5.385833e-03 Error_mean theta Medium=6.571766e-03	Error_local theta Medium=2.043894e-03 Error_mean theta Medium=1.877649e-03
Error_local theta High=2.521139e+00 Error_mean theta High=2.414458e+00	Error_local theta High=2.079608e-01 Error_mean theta High=6.193502e-02

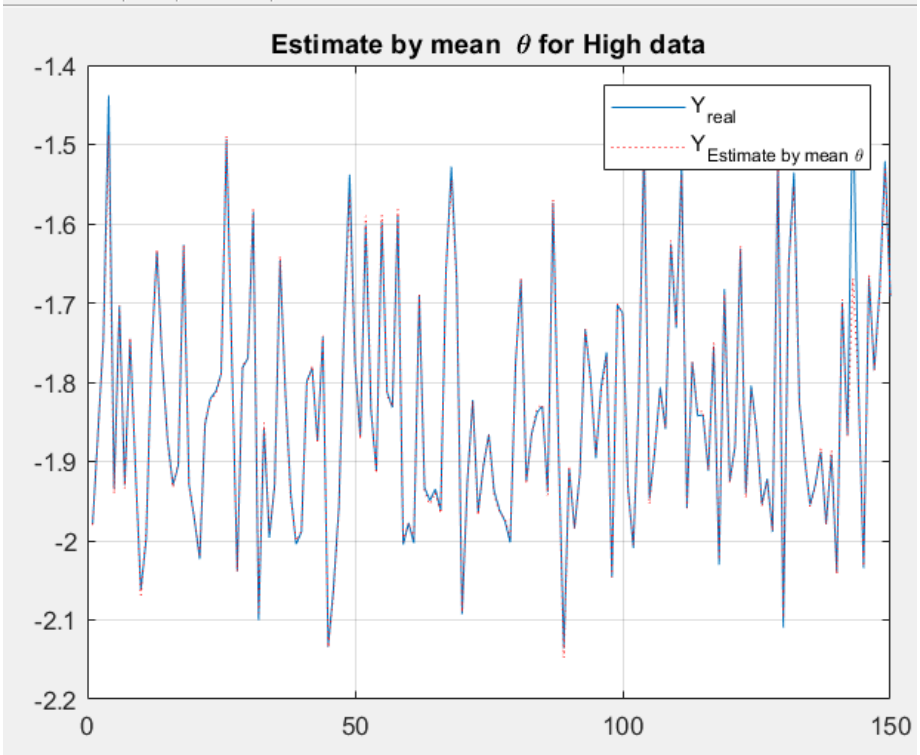
We can say, from the above table, that our results and the amount of error thrown by the algorithm has been reduced.



The plot of output data and the actual data using the mean estimated parameters for low noise(less data)



The plot of output data and the actual data using the mean estimated parameters for Medium noise(less data)



The plot of output data and the actual data using the mean estimated parameters for High noise (less data)

Conclusion: we can conclude from this part that once the number of data points is smaller, the algorithm has an easier task to perform (And this small error is not necessarily a good thing, the algorithm might have just simply overfit. Usually a larger dataset will reduce the odds of overfitting.

Question 1.1.D

When you are performing least squares estimation with an unknown number of regressors, you are dealing with a statistical problem that falls under the category of variable selection or model selection. In such cases, you want to choose the most appropriate subset of regressors from a pool of potential predictors, while avoiding overfitting or underfitting the model.

There are several strategies to address this problem:

1. Stepwise Regression:

- **Forward Selection**: Start with an empty model and iteratively add predictors that improve the model's fit (e.g., based on the reduction in the residual sum of squares or an information criterion like AIC or BIC).
- **Backward Elimination**: Start with a model that includes all available predictors and iteratively remove the predictors that contribute the least to the model.
- **Stepwise**: Combine forward and backward selection by iteratively adding and removing predictors based on certain criteria.

Which we will put into use the **FS** and **BE** in the next questions

2. LASSO (Least Absolute Shrinkage and Selection Operator): LASSO is a regularization technique that adds a penalty term to the least squares objective function, encouraging some of the regression coefficients to be exactly zero. This leads to automatic variable selection. LASSO can be particularly useful when you have a large number of potential predictors.
3. Ridge Regression: Ridge regression is another regularization technique that adds a penalty term to the least squares objective function. While it doesn't perform variable selection by setting coefficients to zero, it can help mitigate multicollinearity and shrink coefficients toward zero, making the model more robust.
4. Elastic Net: Elastic Net is a combination of LASSO and Ridge regression, which includes both L1 (LASSO) and L2 (Ridge) regularization terms. It allows for variable selection while also addressing multicollinearity.
5. Information Criteria: Use information criteria like AIC (Akaike Information Criterion) or BIC (Bayesian Information Criterion) to compare different

models with varying numbers of predictors. These criteria penalize the inclusion of unnecessary predictors and can help you select the best subset of regressors.

6. Cross-Validation: Perform cross-validation to assess the performance of different models with varying numbers of predictors. Cross-validation helps you estimate the out-of-sample predictive accuracy of each model and can guide you in selecting the right subset of regressors.
7. Expert Knowledge: Sometimes, domain expertise can play a crucial role in selecting the most relevant predictors. Subject-matter experts may be able to provide guidance on which variables are likely to be important.

Conclusion: As I mentioned this problem is a stochastical problem, and some solutions involve trial and error (such as BE, FS and cross-validation), in which we assign different model for the system and measure the error from train or test data)

Question 1.1.E

Yes, you can choose the regressors without performing the estimation by using various variable selection techniques or prior knowledge about the predictors. These methods allow you to identify a subset of relevant variables without fitting the regression model first. Here are some common approaches:

1. **Domain Knowledge**: If you have a strong understanding of the subject matter, you can use your domain knowledge to select potential predictors. Expert opinions or theoretical considerations may guide you in choosing relevant variables.
2. **Feature Importance**: You can calculate feature importance scores for each predictor based on their correlation, information gain, or other relevant metrics. This helps identify variables that have a significant impact on the target variable.
3. **Filter Methods**: Filter methods use statistical tests, such as correlation or mutual information, to rank predictors in terms of their relevance to the target variable. You can then choose the top-ranked predictors.
4. **Embedded Methods**: Some machine learning algorithms, like decision trees or random forests, provide variable importances as part of their modeling process. You can use these importances to select relevant predictors without running a separate regression.
5. **Principal Component Analysis (PCA):** PCA can be used to reduce the dimensionality of your predictor space by transforming the original predictors into a set of linearly uncorrelated variables (principal components). You can then select a subset of these components to use in your regression.

Implementation Question 1.2

Ok, this time we want to choose the training data from range $[0, 0.8]$ and give the algorithm the test data from range [0.8,1] to see what happens. So we first need to choose the input data in range $[0, 0.8]$ and it's respective y. After running the algorithm we get this results:

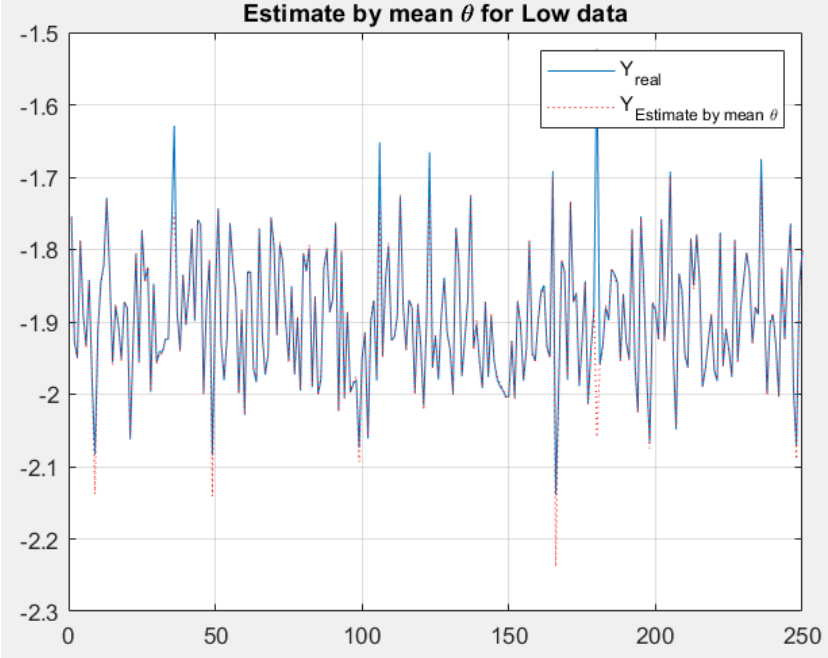


Fig The plot of output data and the actual data using the mean estimated parameters for low noise

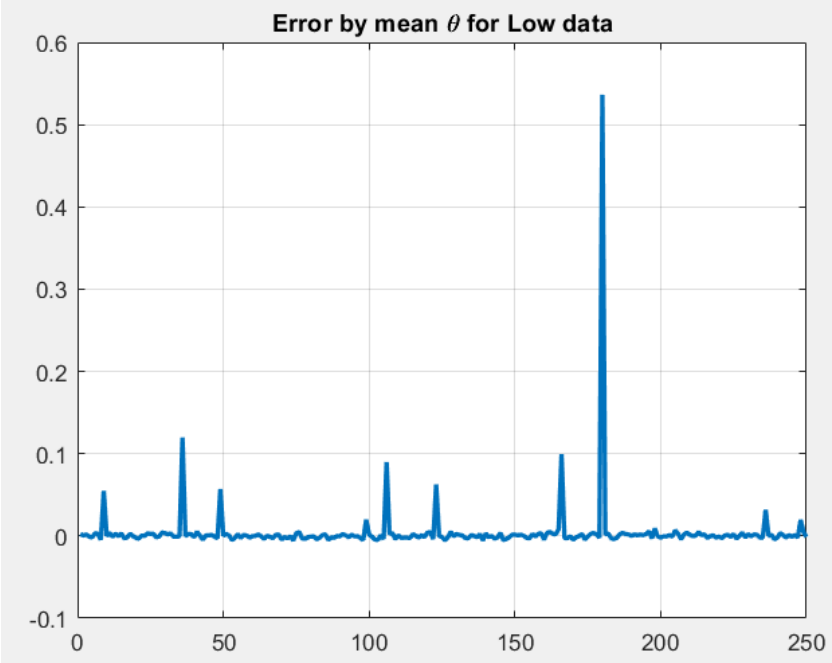
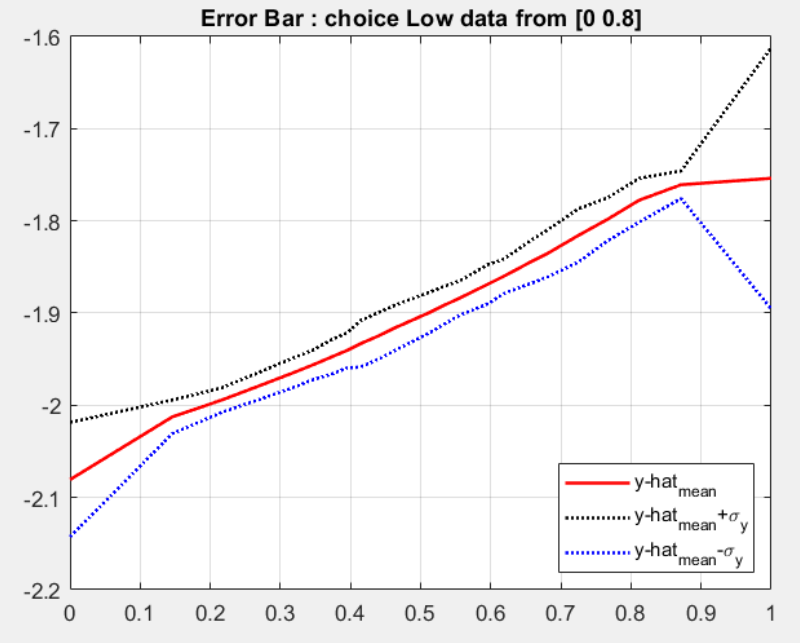
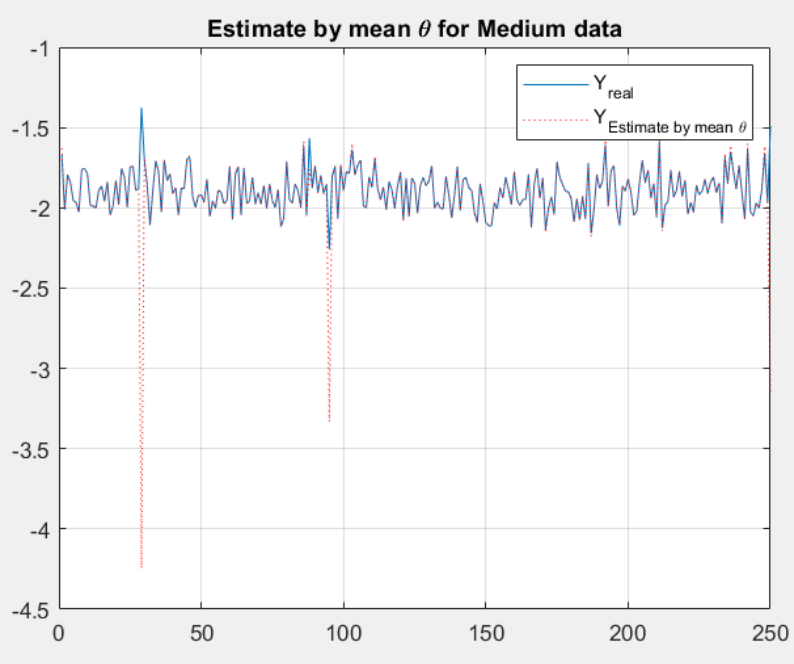


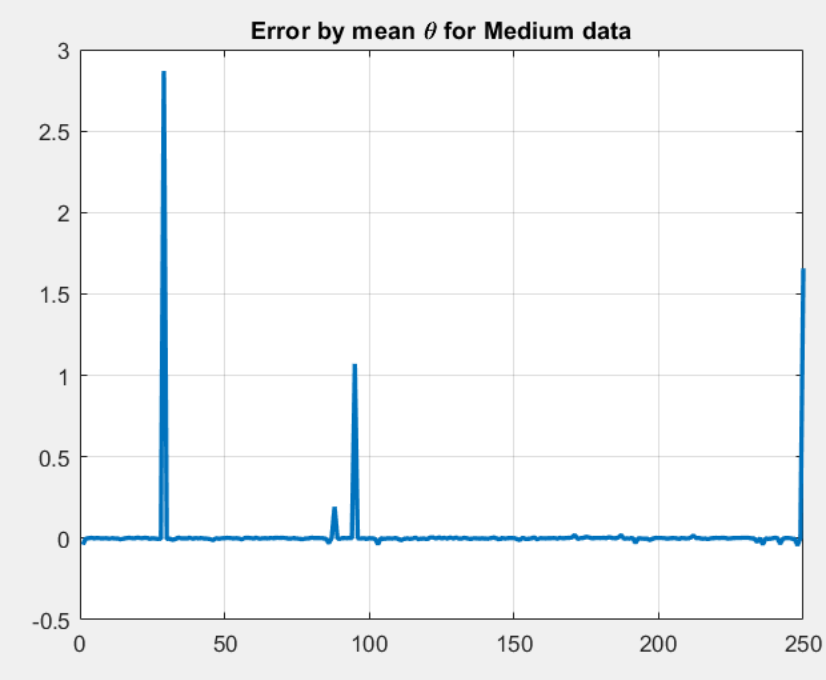
Fig Error plot for the average estimated parameters and Low noise



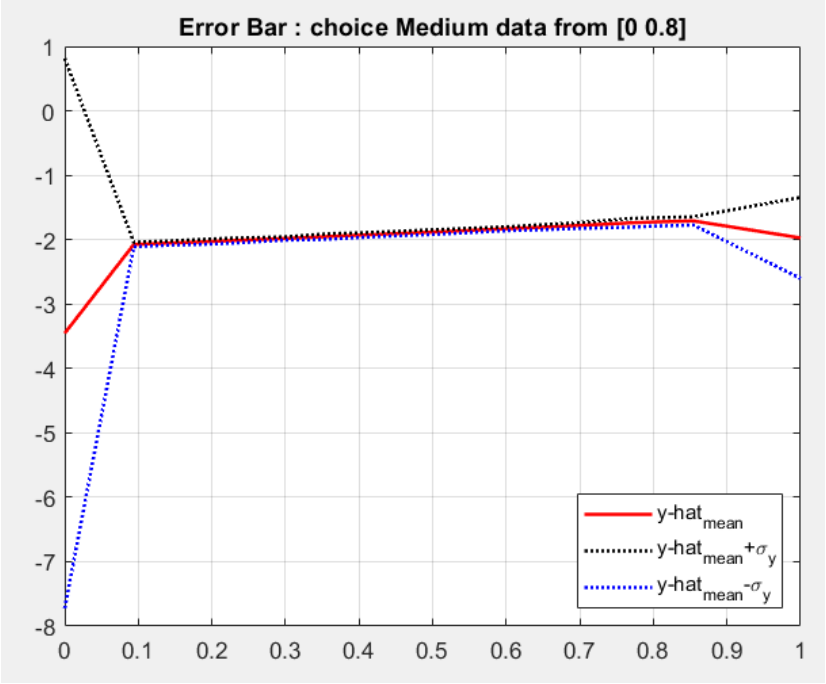
error bar for estimated parameters output Low Noise



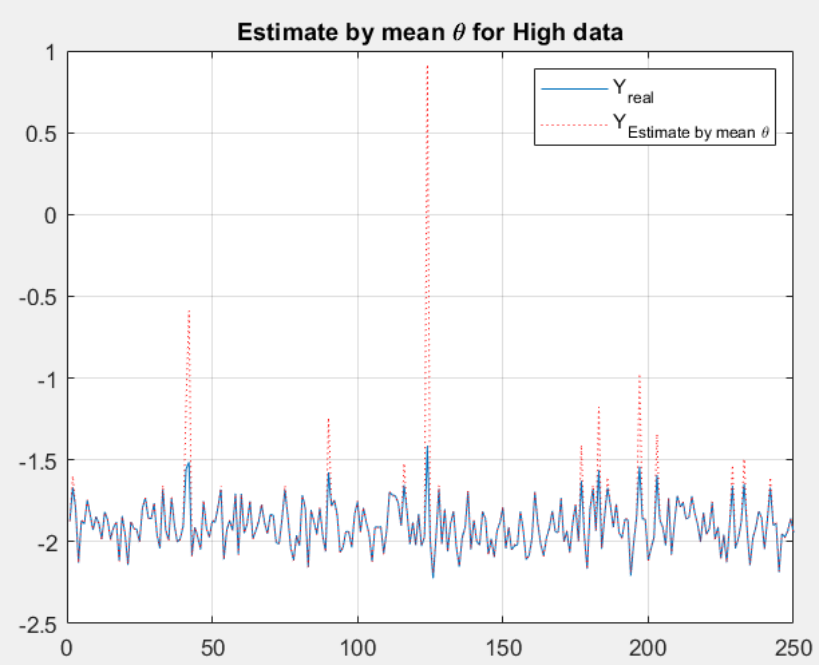
The plot of output data and the actual data using the mean estimated parameters for medium noise



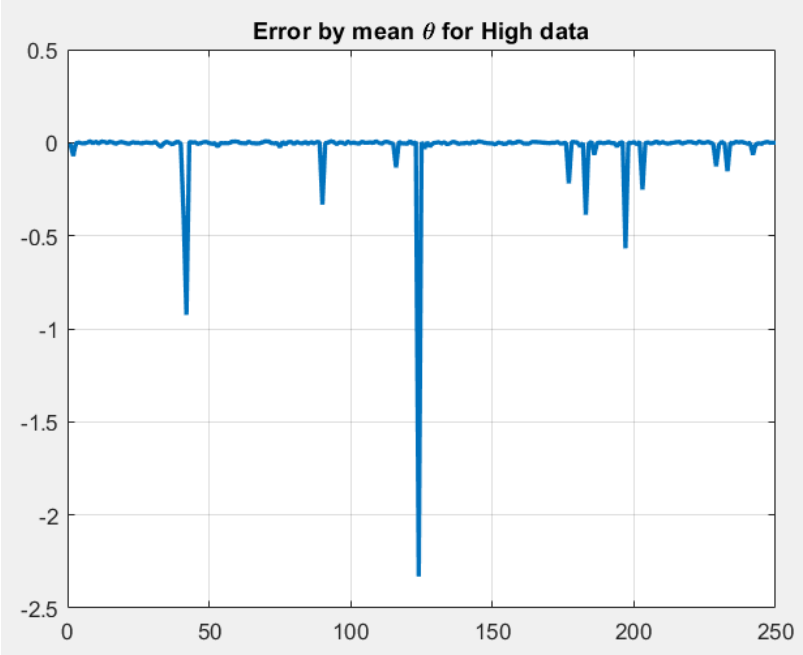
Error plot for the average estimated parameters and Medium noise



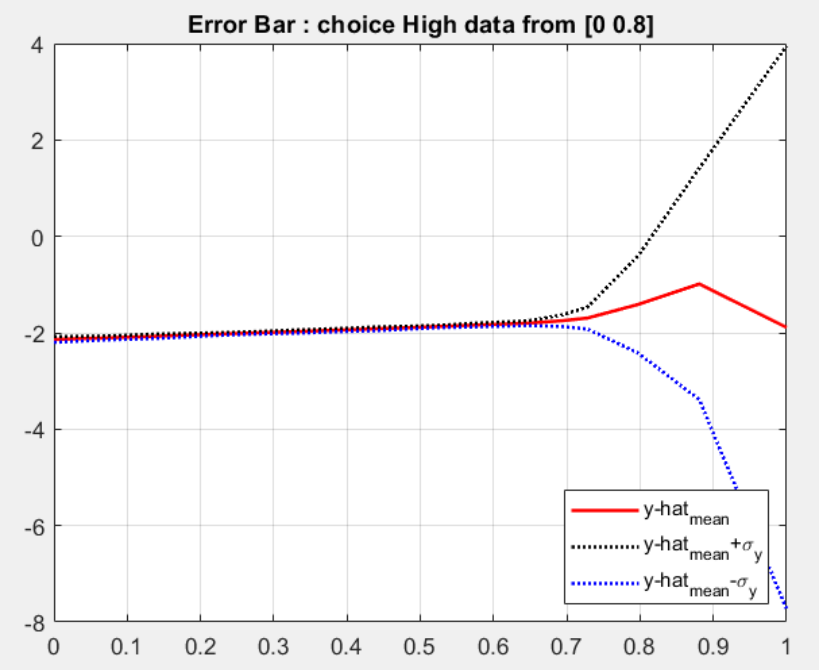
error bar for estimated parameters output Medium Noise



The plot of output data and the actual data using the mean estimated parameters for High noise



Error plot for the average estimated parameters and High noise



error bar for estimated parameters output High Noise

theta =

1.0e+03 *

-0.005141448447581

0.048843743450541

-0.343773185736564

1.404943377890097

-3.615502018332603

5.979055406422101

-6.169261787037079

3.612568733198005

-0.915732330470984

Error_local theta=4.227543e-01

Error_mean theta=3.335390e-01

theta =

1.0e+03 *

-0.004129613458507

0.036766044539925

-0.310616885600780

1.515600061538819

-4.530808464859740

8.433177900238235

-9.497888767892192

5.906281418450310

-1.552626339276072

```
Error_local theta=1.838558e+02
Error_mean theta=1.217466e+01

theta =
   1.0e+04 *
  -0.000367140906822
   0.003586747374241
  -0.035621414077967
   0.187294538887572
  -0.568863269722090
   1.035269389991960
  -1.111061767318762
   0.645929847060302
  -0.156075263440822

Error_local theta=3.763700e+02
Error_mean theta=7.217712e+00
```

From the figures, the estimated values for θ's and error values, it's clear that the algorithm failed to estimate the correct values for the system. So we can conclude that the data we want to feed the algorithm, must be representative of the system. And set aside the error and estimated parameters, we see the **error bars are off the chart! Which indicate the estimation at the beginning and at the end of the selected range is not trustworthy!**

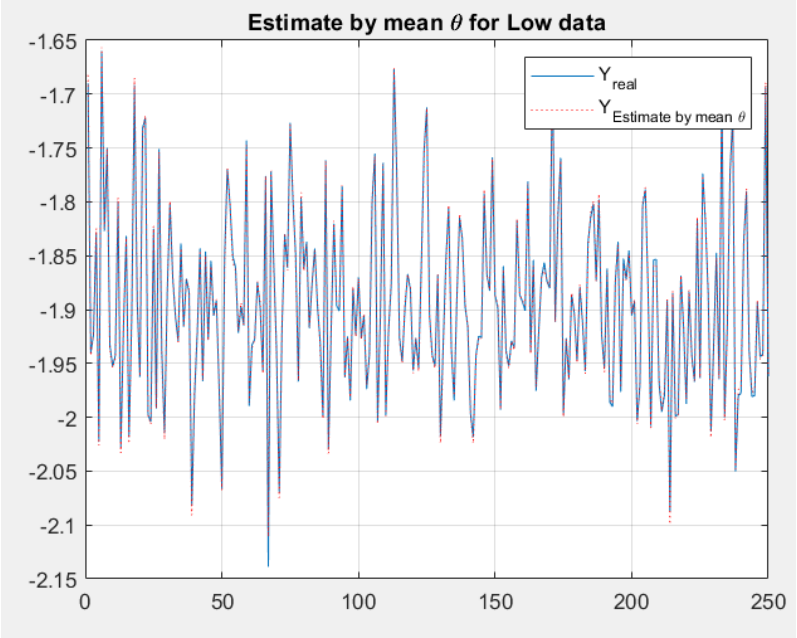
Implementation Question 1.3

As you know the sliding window method is a method used in LTV system identification. In these cases because of the changes in the operation point of the system, the priori data is not as important as fresh data. Using the sliding window method we use a window of size N (N priori data points) to estimate the system parameters.

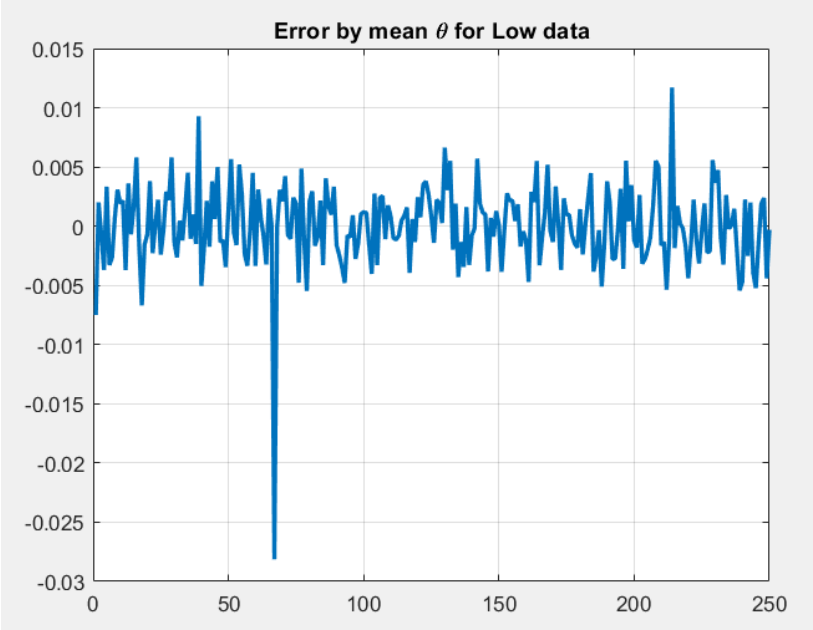
We expect no big difference because the system is LTI, but because of dividing the system to appropriate subsystems, we might get smaller error values. We take a window size of 150 data points.

We run the algorithm:

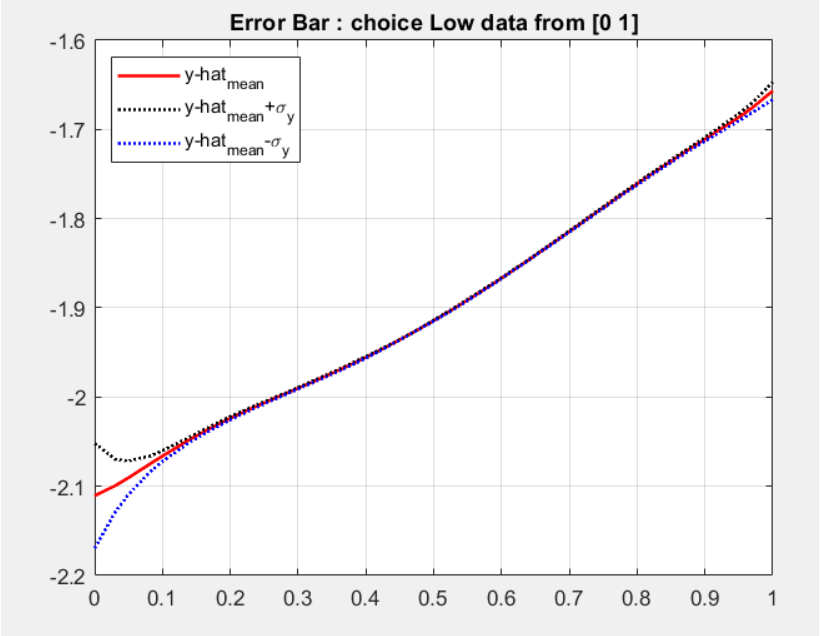
Low Nosie:



Plot of output data using sliding window and the actual data using the mean estimated parameters for Low noise

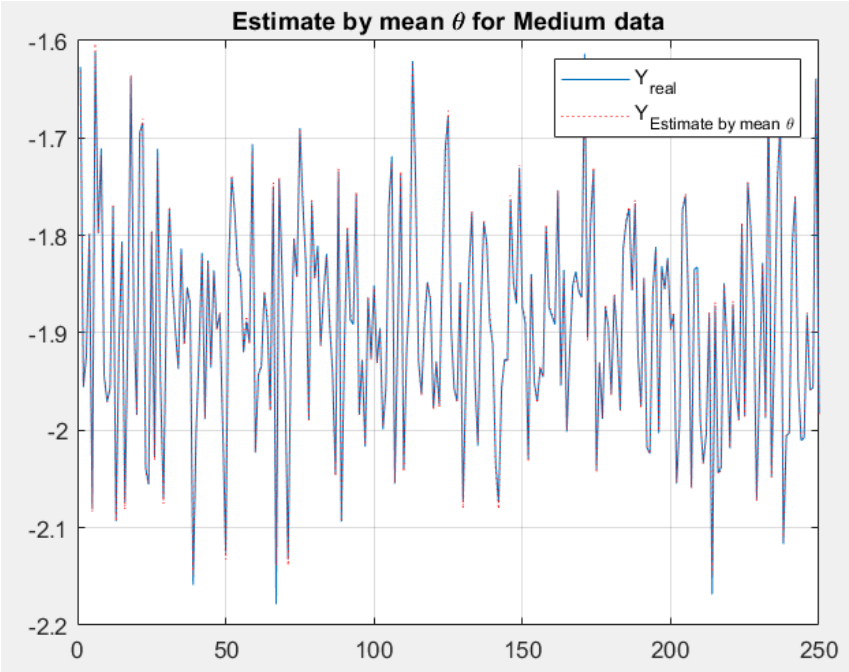


Error plot for the average estimated parameters using the sliding window method and Low noise

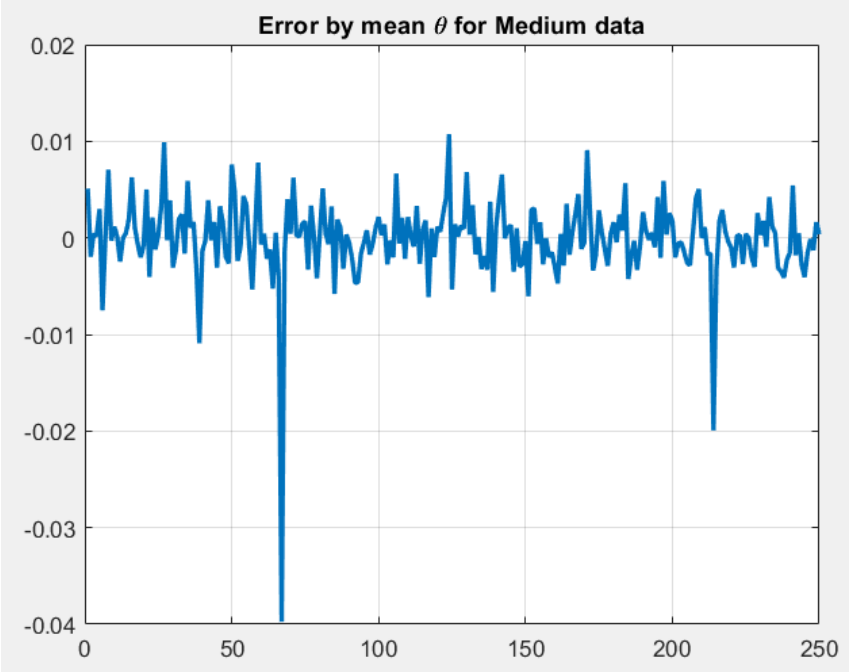


Error bar for estimated parameters output using sliding window method and Low Noise

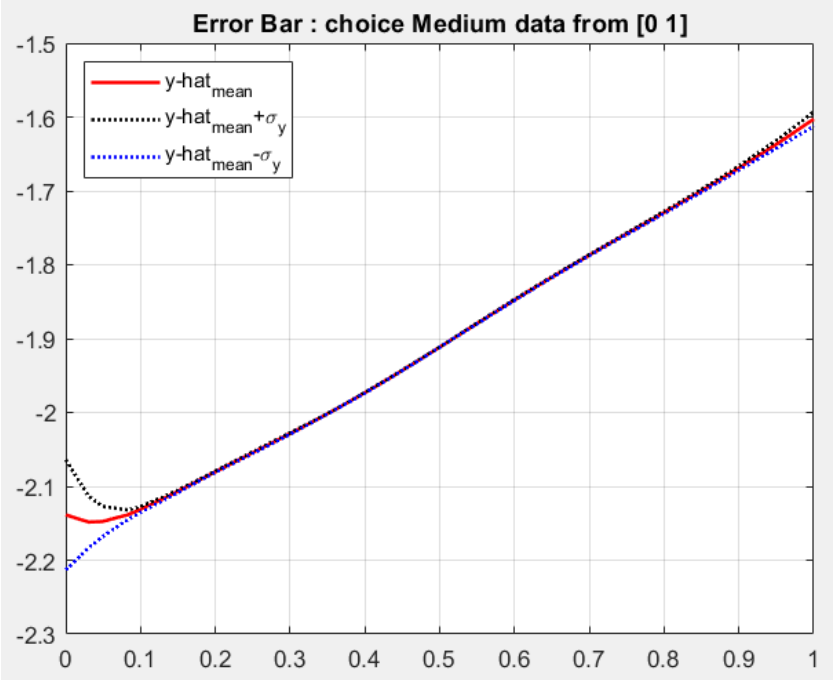
Medium Nosie:



Plot of output data using sliding window and the actual data using the mean estimated parameters for Medium noise

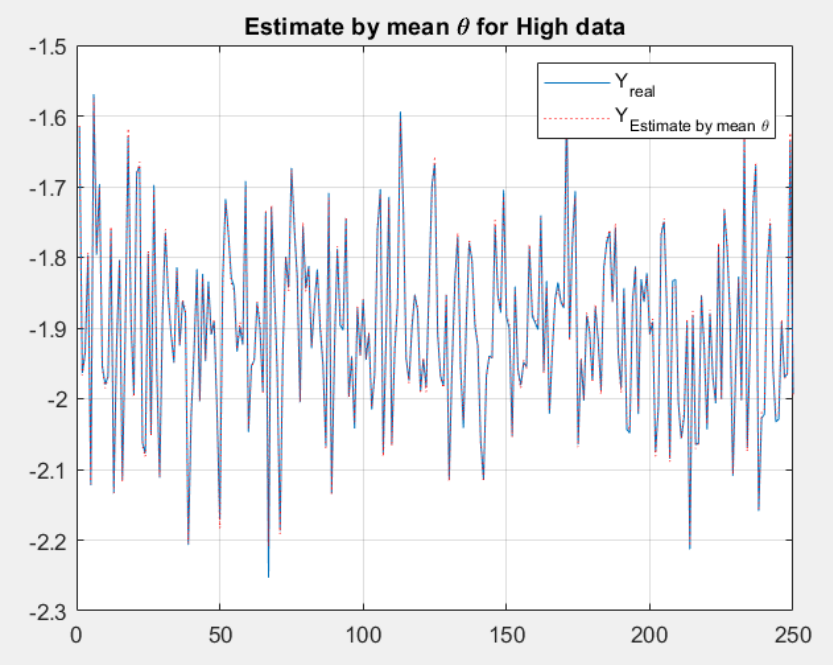


Error plot for the average estimated parameters using the sliding window method and Medium noise

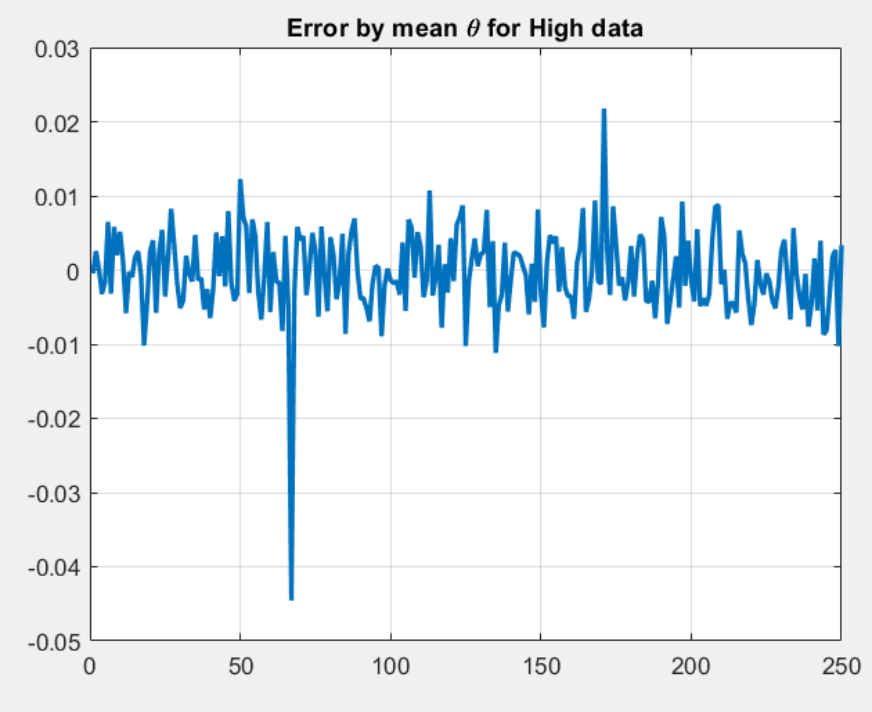


Error bar for estimated parameters output using sliding window method and Medium Noise

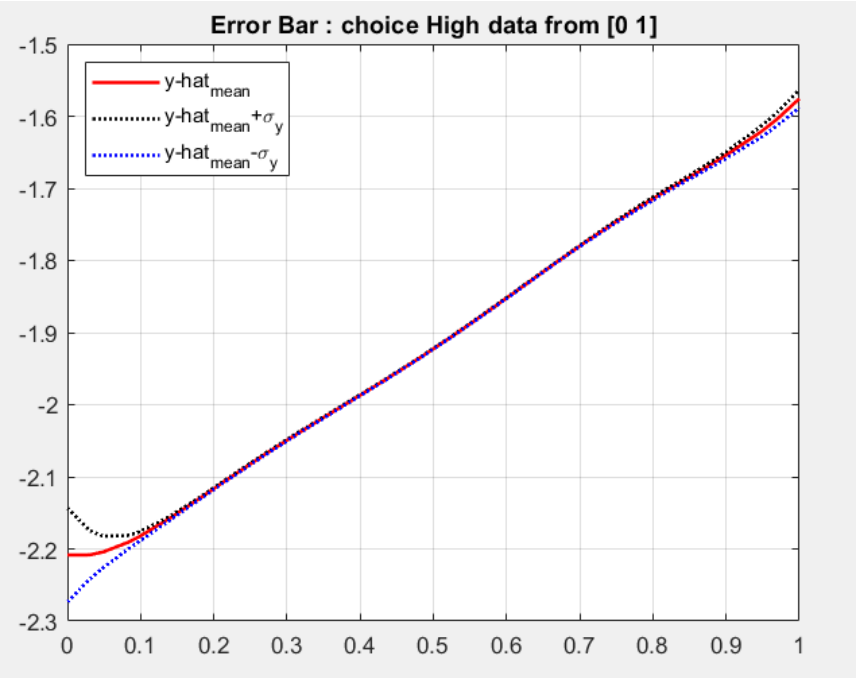
High Noise:



Plot of output data using sliding window and the actual data using the mean estimated parameters for High noise



Error plot for the average estimated parameters using the sliding window method and High noise



Error bar for estimated parameters output using sliding window method and High Noise

theta =

1.0e+03 *

-0.001494150182244

-0.014487959244253

```
  0.130333450515270
 -0.599522152509274
  1.626943107131691
 -2.707991382537023
  2.731562685337569
 -1.537467971832252
  0.371046268070548

Error_mean theta Low=2.956444e-03

theta =
   1.0e+03 *
 -0.000475140545652
 -0.031837693659457
  0.244526736180061
 -1.004193599748299
  2.465039049358056
 -3.707521853348647
  3.348230574296632
 -1.667273972304686
  0.352104662487143

Error_mean theta Medium=4.329809e-03

theta =
   1.0e+03 *
 -0.001745590446618
 -0.006754231251578
  0.027377642430281
  0.014639773614535
 -0.346608032371941
  0.977152663671578
 -1.273908248051115
  0.812936705396655
 -0.204266384800176

Error_mean theta High=7.791280e-03
```

We see the error has reduced and beside that we see the estimated values are close to the actual values, so we can say that the sliding window algorithm is a win. And error bar is tighter. And pay attention that sliding window algorithm is used more in online parameter estimation that offline usage.

Implementation Question 1.4

The main idea behind WLS algorithm is that if amongst training data there are some data points that are not so reliable we have to give them less importance. And we do so by assigning them a coefficient, which we know as weight, and this approach is somehow effective because we don't simply get rid of the data, we assign it a smaller weight. As we know, a value smaller than 1, gets smaller every time it is multiplied in itself.

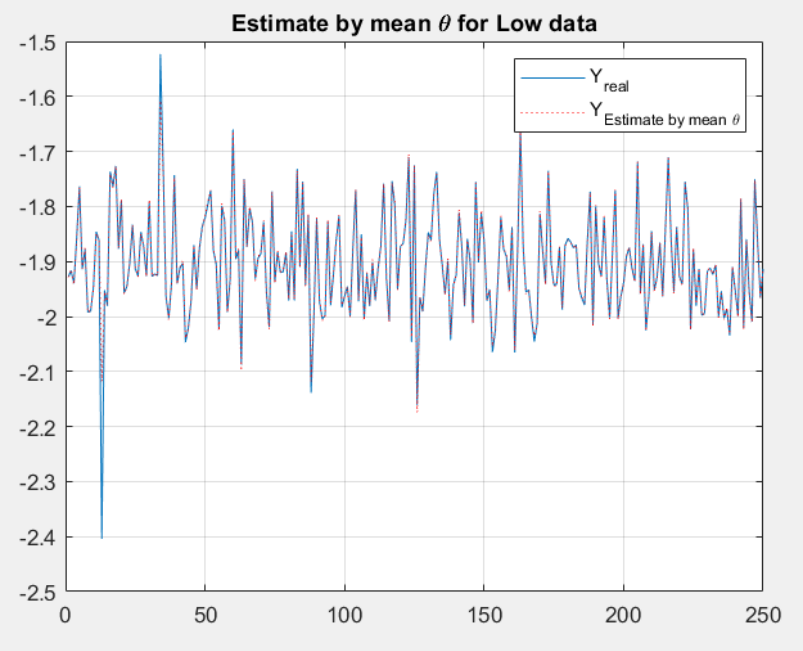
The formula for WLS is as below:

$$\hat{\theta} = (X^T W X)^{-1} X^T W Y$$

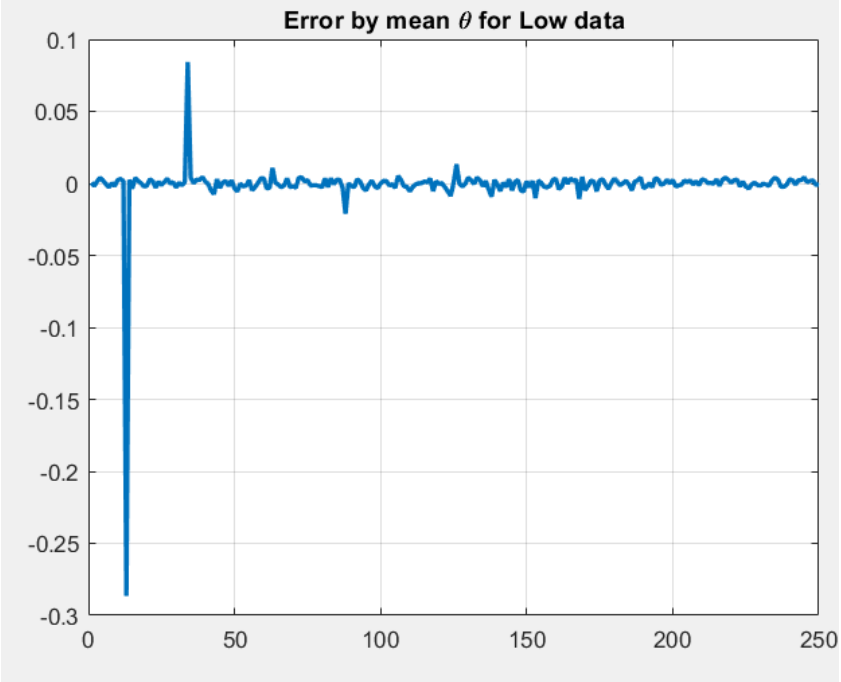
We take the following W(weight) and the parameter $\lambda = 0.95$:

$$W = \begin{bmatrix} \ddots & 0 & 0 & 0 & 0 \\ 0 & \lambda^3 & 0 & 0 & 0 \\ 0 & 0 & \lambda^2 & 0 & 0 \\ 0 & 0 & 0 & \lambda & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

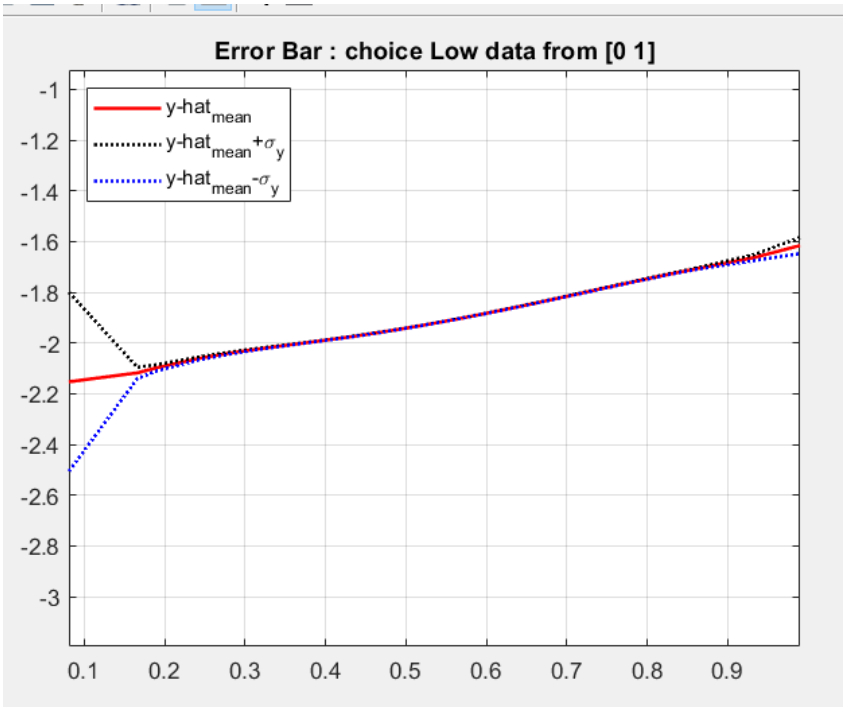
Low Noise



Plot of output data using WLS algorithm and the actual data using the mean estimated parameters for Low Noise

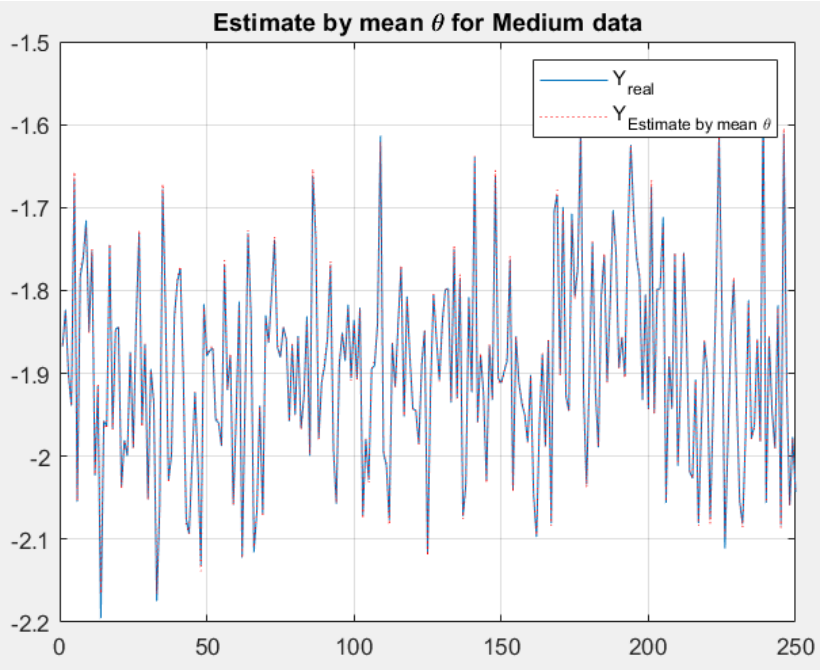


Error plot for the average estimated parameters using the WLS method and Low Noise

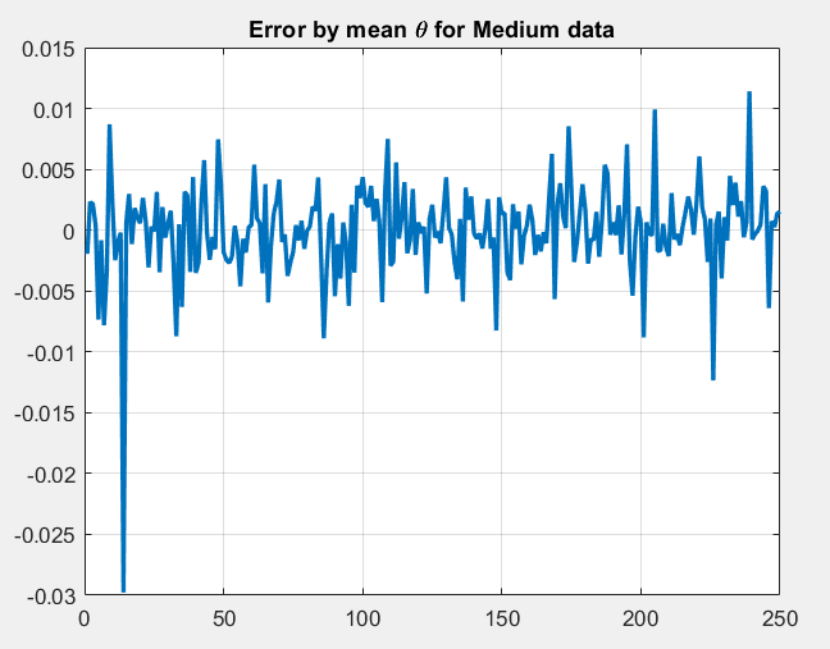


Error bar for estimated parameters output using WLS method and Low Noise

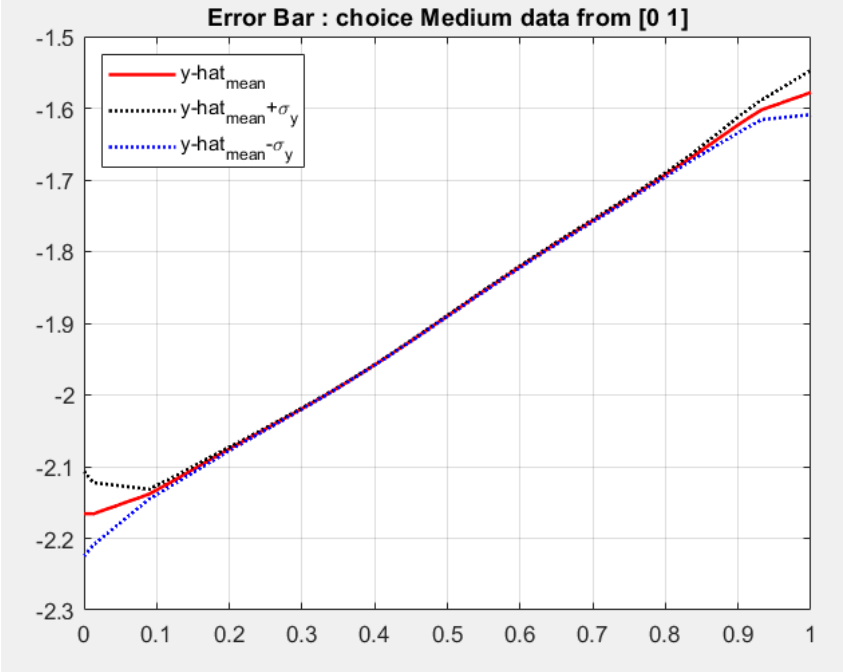
Medium Noise



Plot of output data using WLS algorithm and the actual data using the mean estimated parameters for Medium Noise

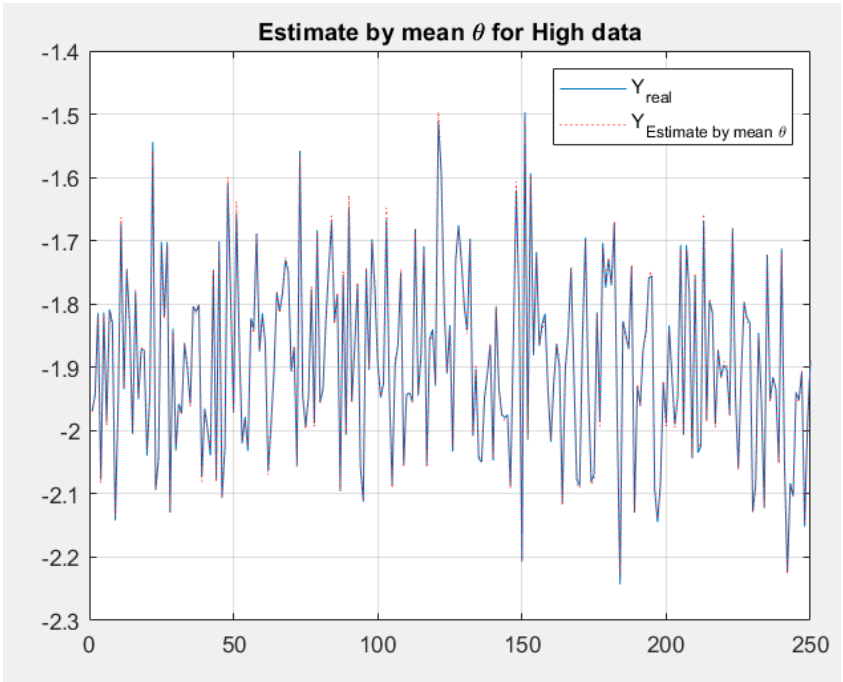


Error plot for the average estimated parameters using the WLS method and Medium Noise

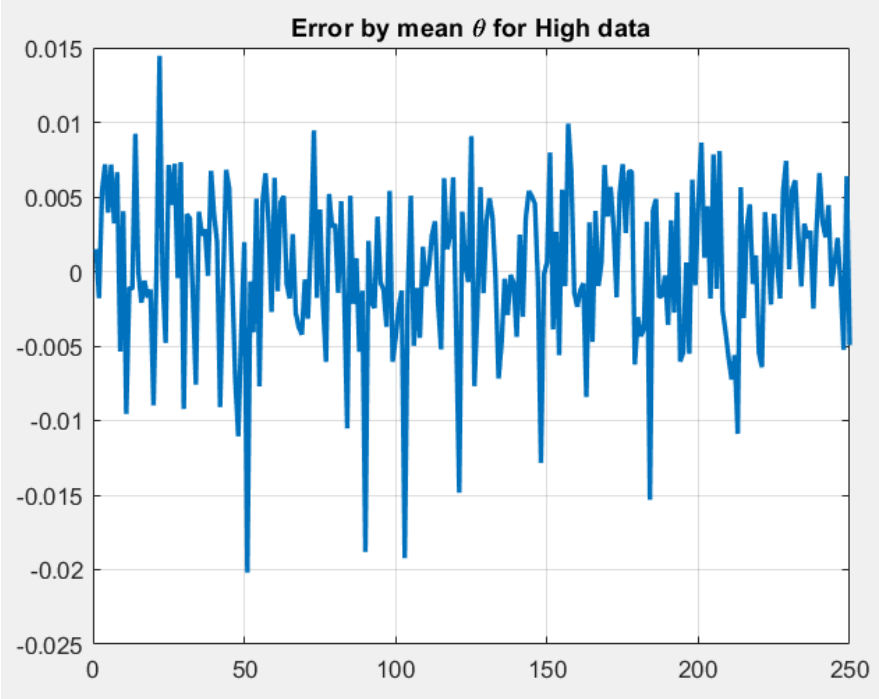


Error bar for estimated parameters output using WLS method and Medium Noise

High Noise:

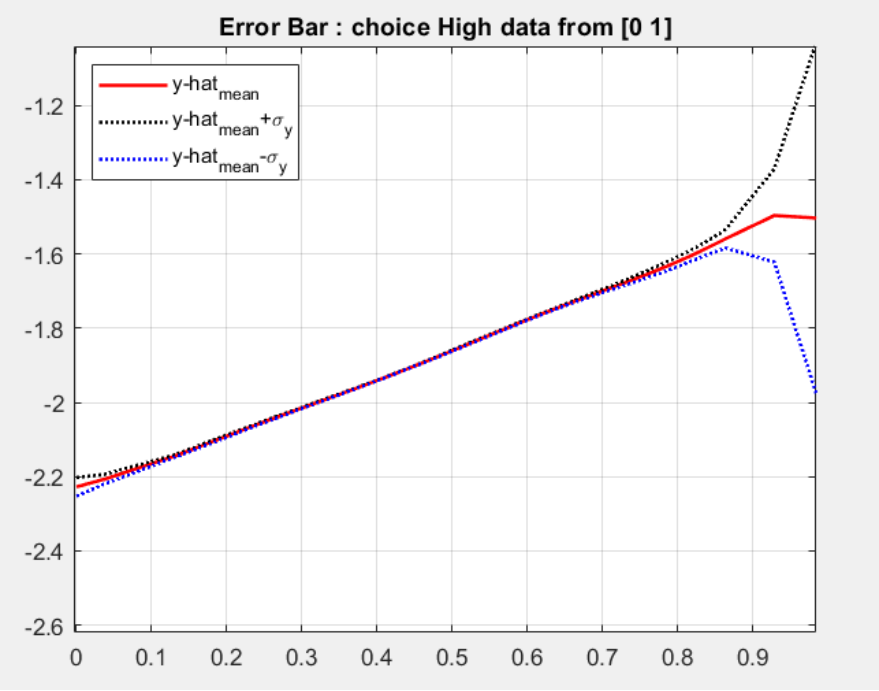


Plot of output data using WLS algorithm and the actual data using the mean estimated parameters for High Noise



Error plot for the average estimated parameters using the WLS method and High Noise

Fig



Error bar for estimated parameters output using WLS method and High Noise

```
theta =
   1.0e+03 *
  -0.002117778290675
  -0.003124298168246
   0.044608797367157
  -0.243622499216532
   0.726830220856761
  -1.276708723004713
   1.327726793844618
  -0.759242332984813
   0.184251484411555

Error_mean theta Low=9.191738e-02

theta =
   1.0e+03 *
  -0.001415691068829
  -0.014869139053902
   0.113365154144692
  -0.437559935932003
   0.970408482406713
  -1.244210593926493
   0.871010496908998
  -0.276676191465679
   0.018179377979745

Error_mean theta Medium=3.508943e-03

theta =
   1.0e+03 *
  -0.002854230804155
   0.013422447258654
  -0.126727853713973
   0.663516390055868
  -2.004079764627066
   3.621385333761876
  -3.857850138003087
   2.231342127284846
  -0.539728123660457

Error_mean theta High=6.964315e-03
```

As is obvious the WLS algorithm has managed to estimate the parameters better and make the error smaller.

Implementation Question 1.5

LS estimation method is an offline algorithm. If you want to make the algorithm online, one thing to consider is recursive LS algorithm or RLS, Which we utilize a covariance matrix and a recursive equation to update our information. The equations are as follows:

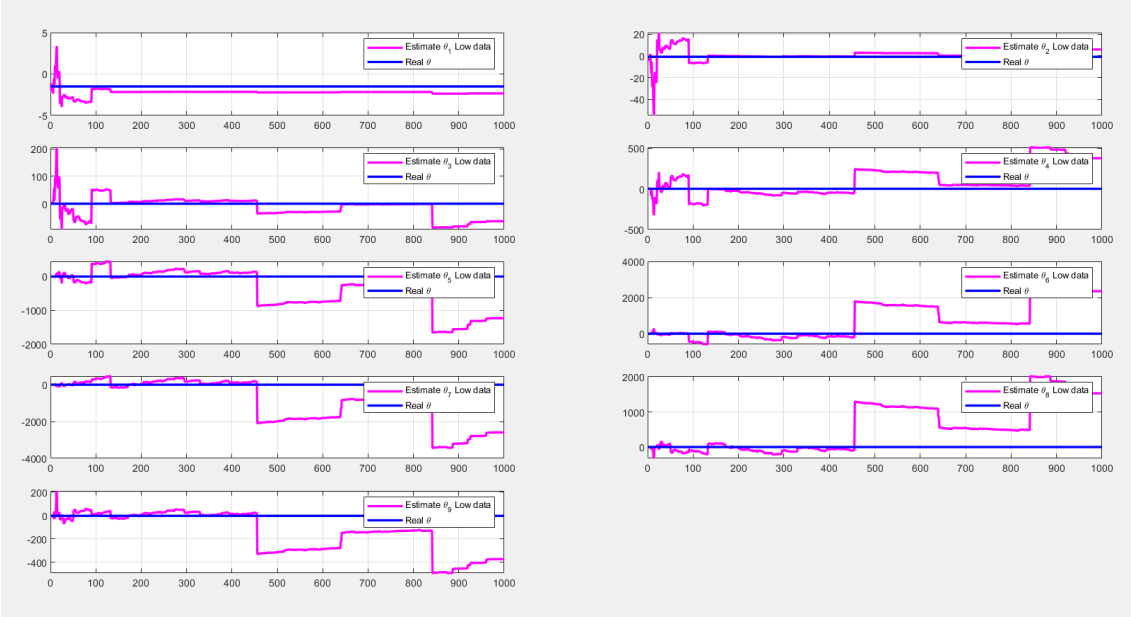
$$\widehat{\theta}(t+1) = \widehat{\theta}(t) - P(t+1)U(t+1)\varepsilon(t+1)$$

$$P(t+1) = P(t) - \frac{P(t)U(t+1)}{1 + U^T(t+1)P(t)U(t+1)}$$

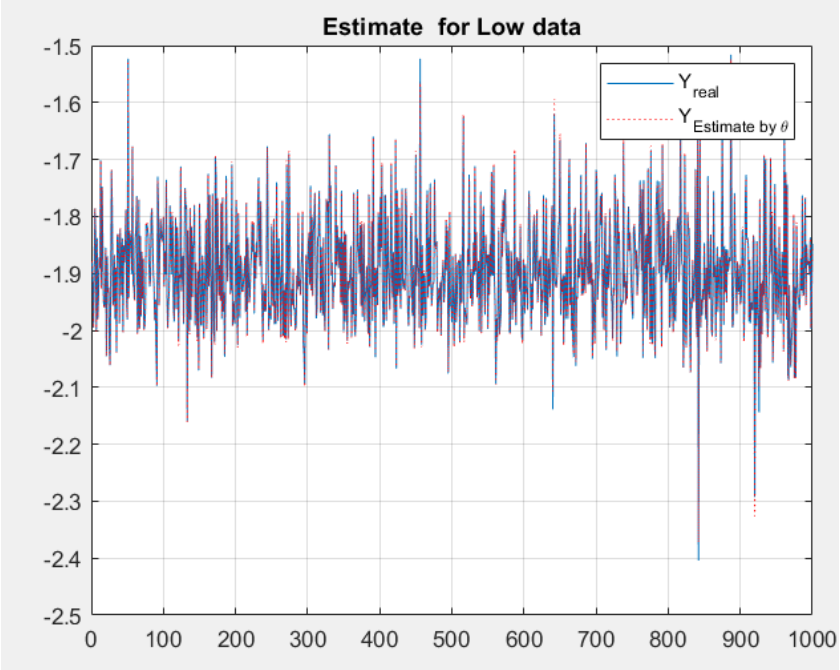
$$\varepsilon(t+1) = U^T(t+1)\hat{\theta}(t) - Y(t+1)$$

If we reduce the estimation error, instead of the actual error, we can converge to the optimal parameters even faster:

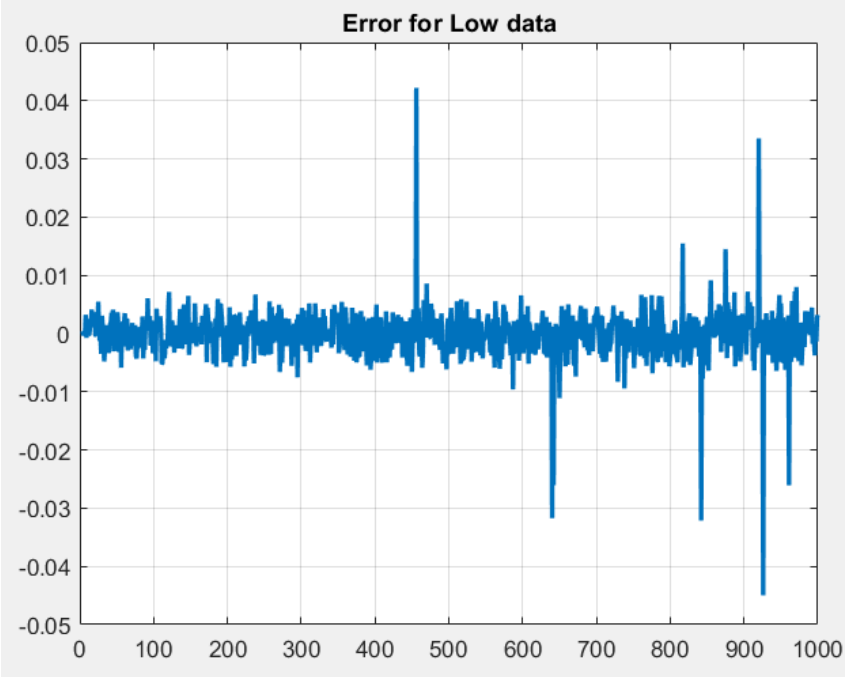
$$\hat{\theta}_0 = zeros(10,1)\,, P(0) = 10^{11} I$$



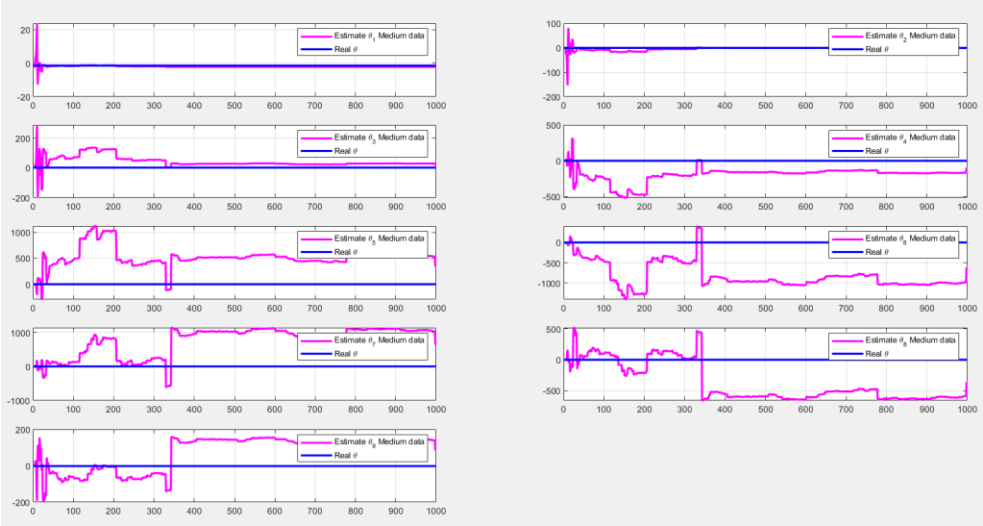
Convergence plots of parameters for Low Noise



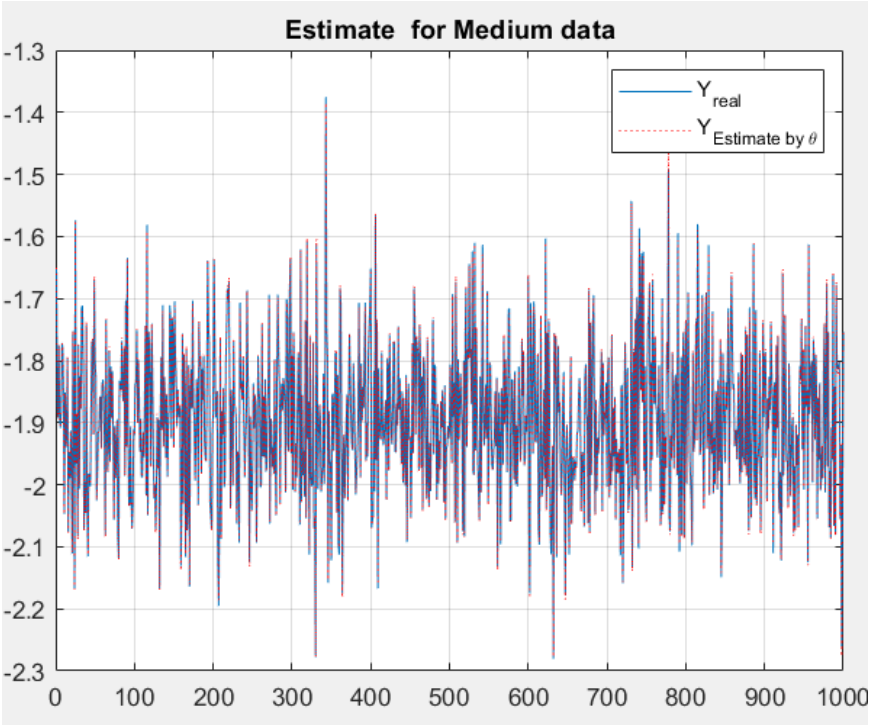
Plot of output data using WLS algorithm and the actual data using the mean estimated parameters for Low Noise



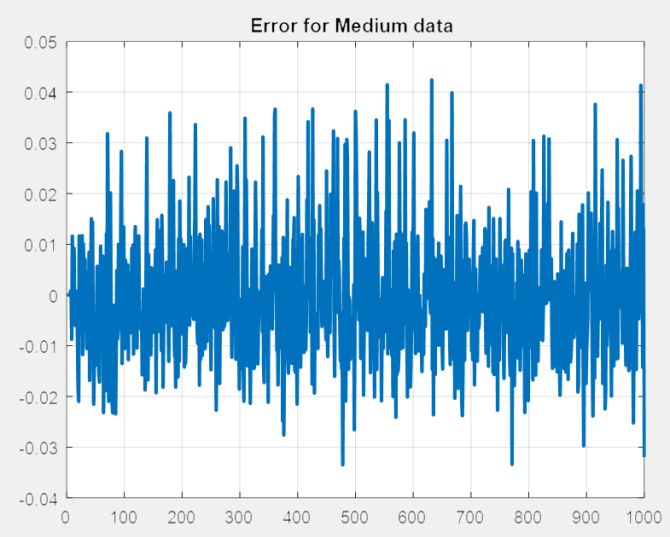
Error plot for the average estimated parameters using the WLS method and Low Noise



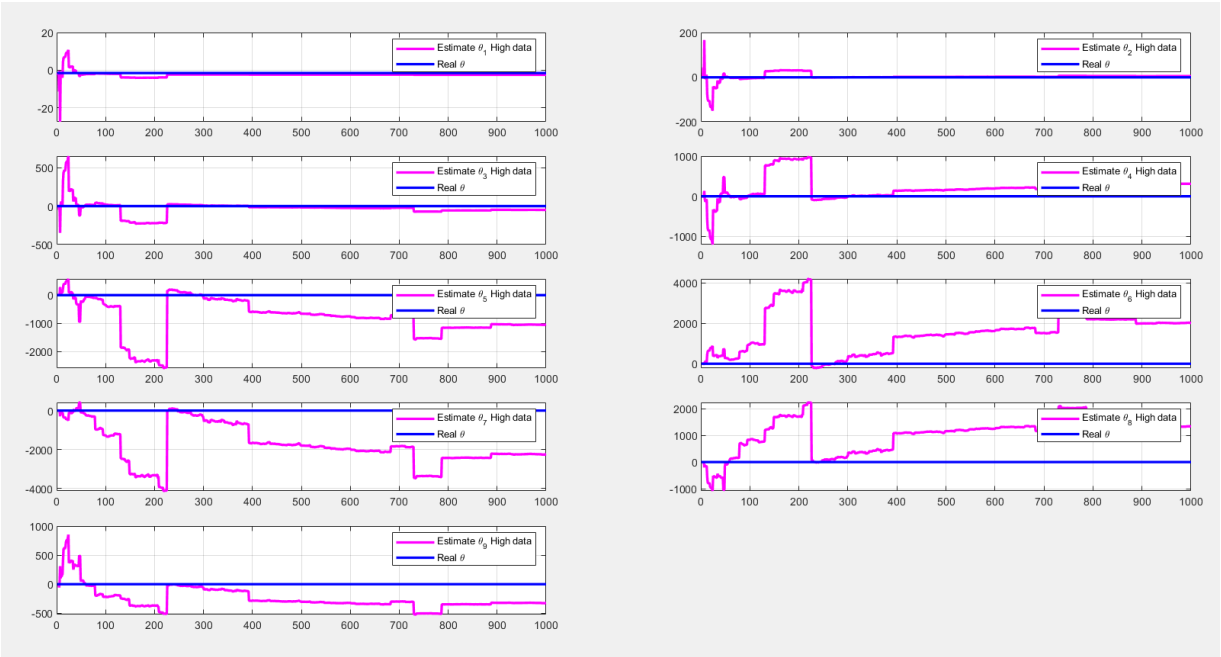
Convergence plots of parameters for WLS method and Medium Noise



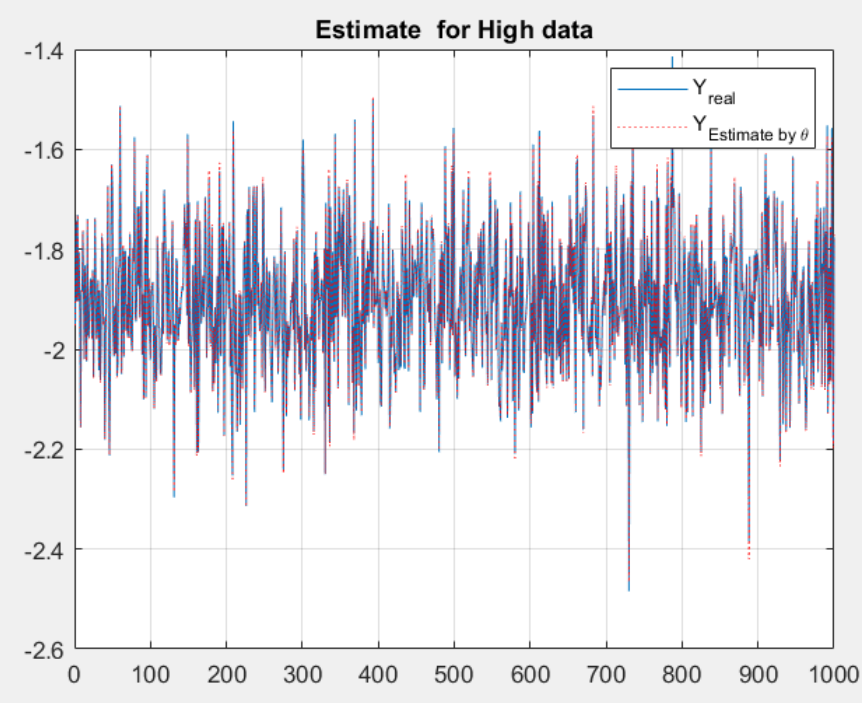
Plot of output data using WLS algorithm and the actual data using the mean estimated parameters for Medium Noise



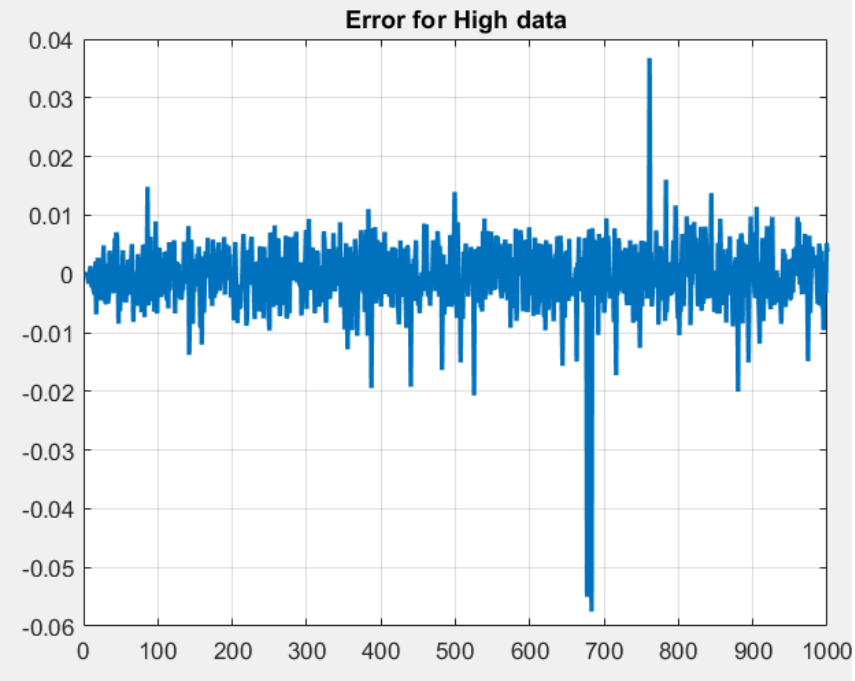
Error plot for the average estimated parameters using the WLS method and Medium Noise



Convergence plots of parameters for WLS method and High Noise



Plot of output data using WLS algorithm and the actual data using the mean estimated parameters for High Noise

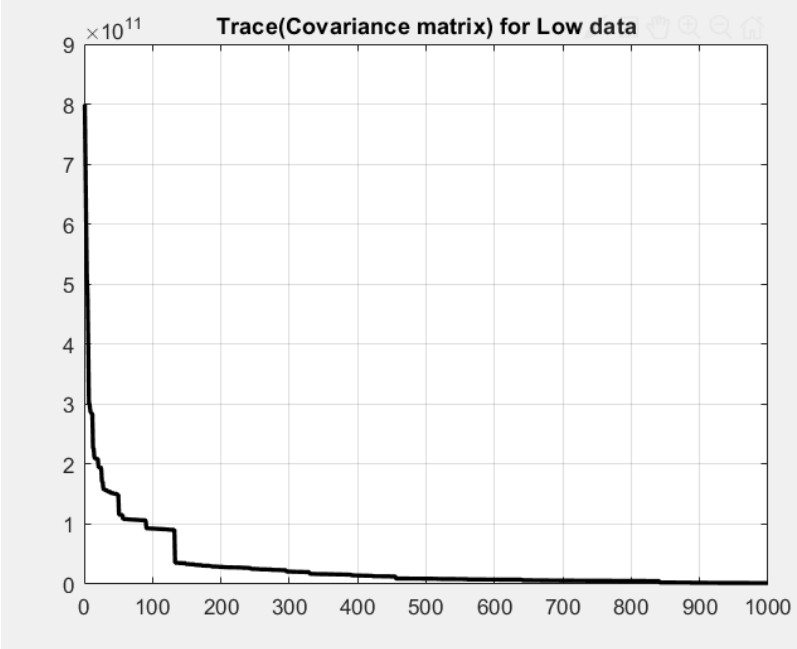


Error plot for the average estimated parameters using the WLS method and High Noise

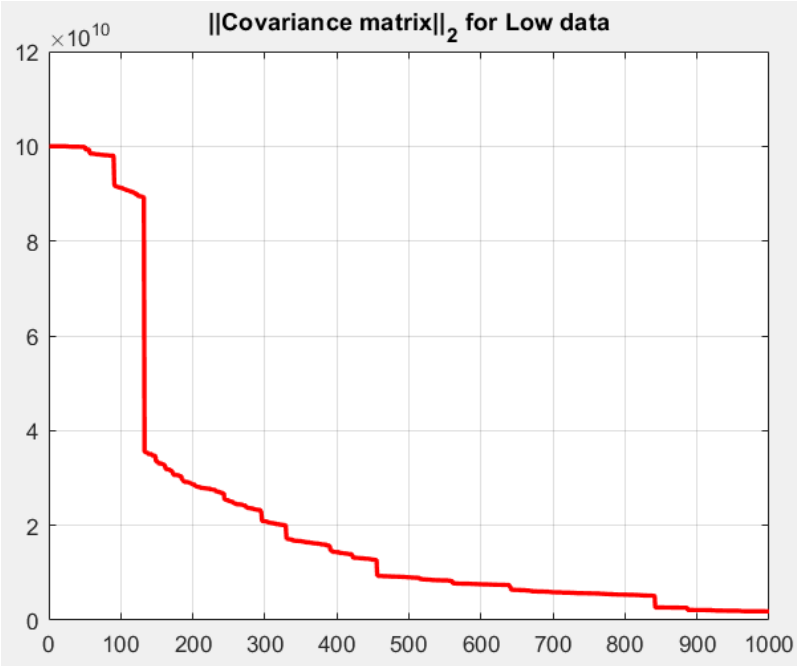
As is shown in the figures above, we have a relatively good convergence between actual parameters and the estimated parameters, and the estimated output

follows the real output very well. So we can conclude the algorithm is working properly.

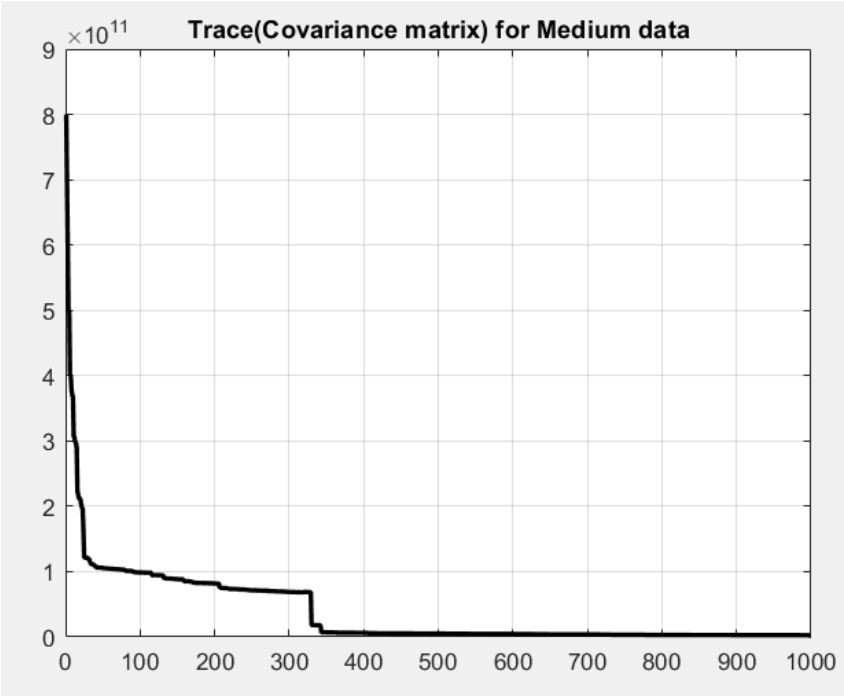
In this simulation, always a small value is reduced form the positive definite matrix P, as a result matrix P is going towards zero, we can prove that by plotting the norm of the matrix



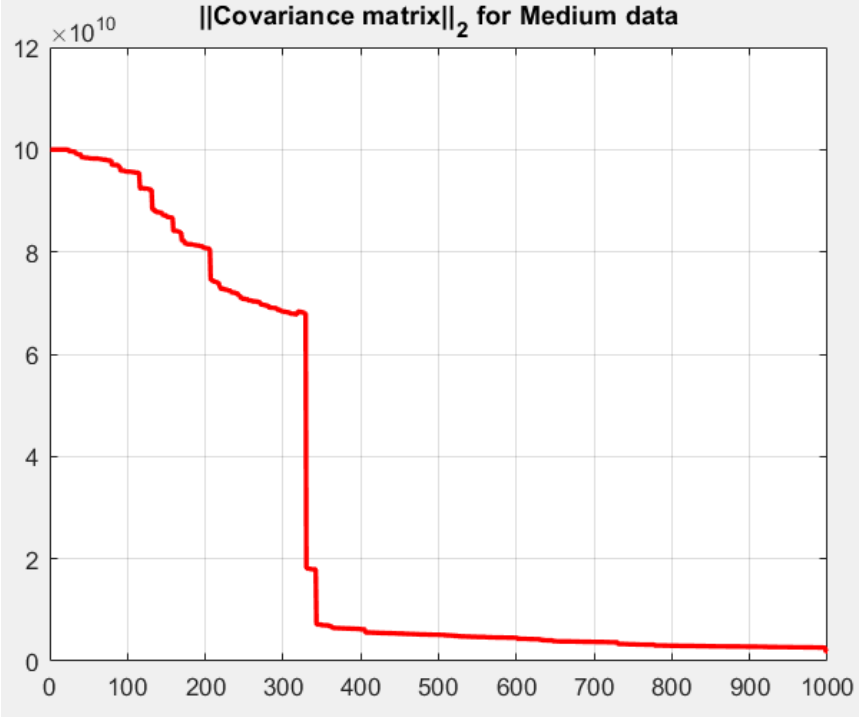
Plot of trace of covariance matrix for Low Noise



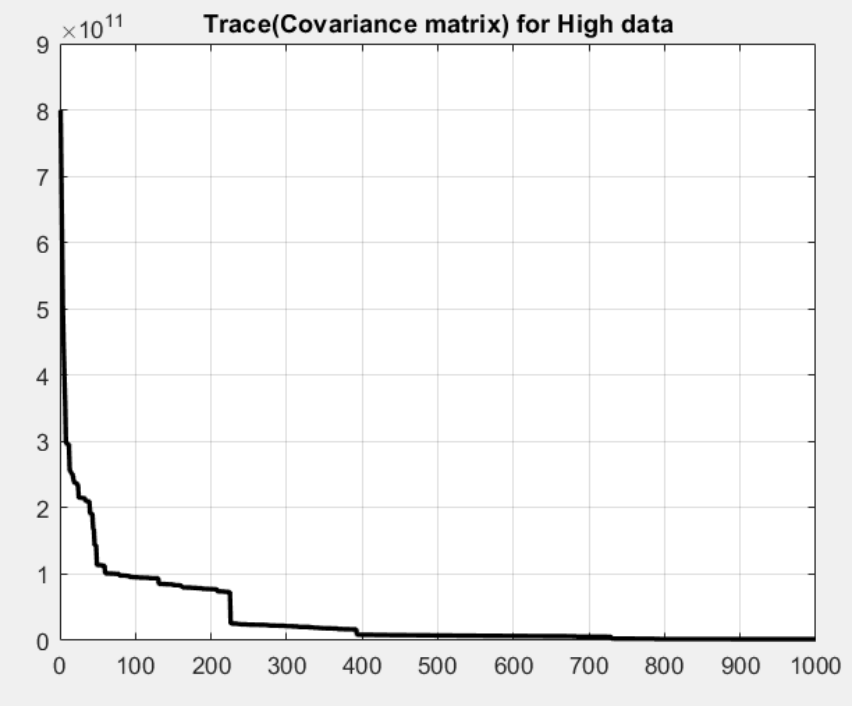
Plot of Norm of covariance matrix for Low Noise



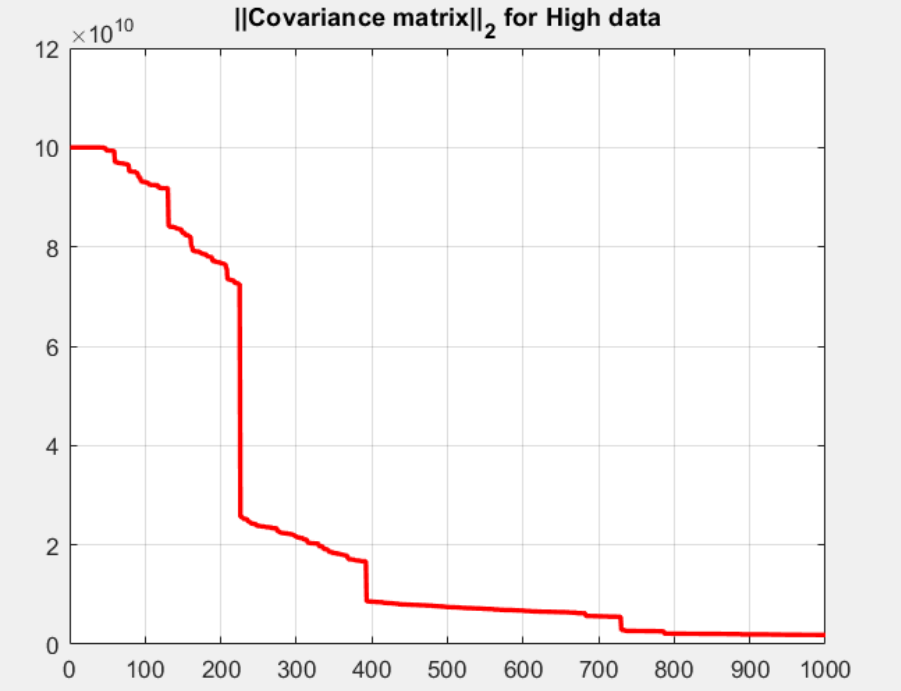
Plot of trace of covariance matrix for Medium Noise



Plot of Norm of covariance matrix for Medium Noise



Plot of trace of covariance matrix for High Noise



Plot of Norm of covariance matrix for High Noise

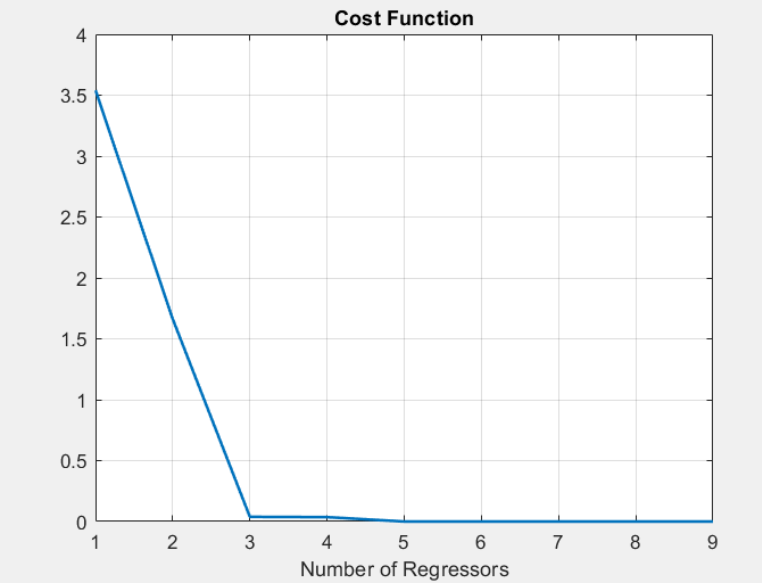
Implementation Question 1.6

Forward Selection

One of the methods of subset selection is **Forward selection,** that selects a subset of main regressors from a set of n regressors. The process is defined by:

1. With the training data, we carry out n uni-regressor estimations and use the test data to calculate the error (evaluate each estimation) and choose the regressor that has the least error as the first main regressor.
2. Now with the chosen regressor from the previous step we perform $n-1$ two-regressor estimations and using the test data we evaluate the estimations, and choose the regressor with the least error, now we have our second main regressor.
3. With the regressors chosen from the last two steps, perform $n-k+1$ k-regressors estimations and using the test data we evaluate the estimations and choose the one with the least error.
4. Do this until you reach the desired error value, or until you see no noticeable changes in error.

Note: Here we overlook the Nosie and the cost function is the error between the actual value and the test output.



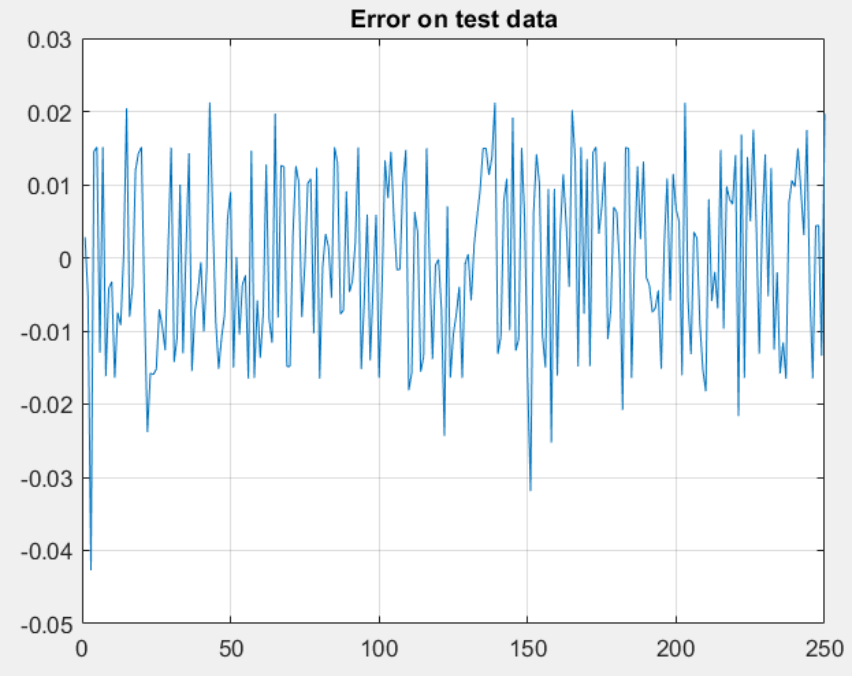
Plot for the cost based on number of regressors in FS algorithm

It's clear that after just 3 steps the error reaches a negligible value.
For this specific case the without noise we have the order listed below:

Order Of Important Regressors Forward Selection =**1 2 4 9** 7 3 6 5 8
Choosing 1, 2, 4 and 9 as the main regressors, we get a pretty decent error which is **literal zero.**
And the output equation will become this:

$$y = -1.5 - 0.8u + 0.01u^3 - 1.7u^8$$

Choosing these main regressors the cost function will become zero and this shows that the **Forward Selection** method does excellent in the absence of noise. We can choose another regressor but nothing significant happens and the computational cost might go up.

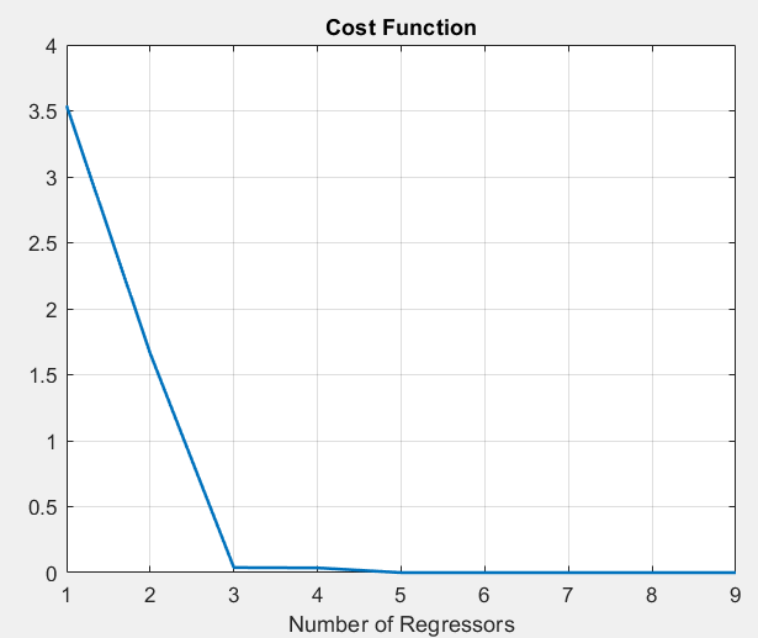


Plot of error between estimated output and the actual output in FS

Estimation on the test data using the main regressors.
MSE of Error in Test Data =
1.4395e-04
The error is very small, this method is awesome☺

Low Noise:
Order Of Important Regressors Forward Selection = 1 2 4 9 7 8 6 5 3
The order in which we going to select the main regressors is **1, 2, 4, 9 and 7.**
The errors we get from each step is listed below:
Cost Function =
3.5396 1.6617 0.0384 0.0359 0.0002 0.0002 0.0002 0.0002 0.0002

Low Noise:

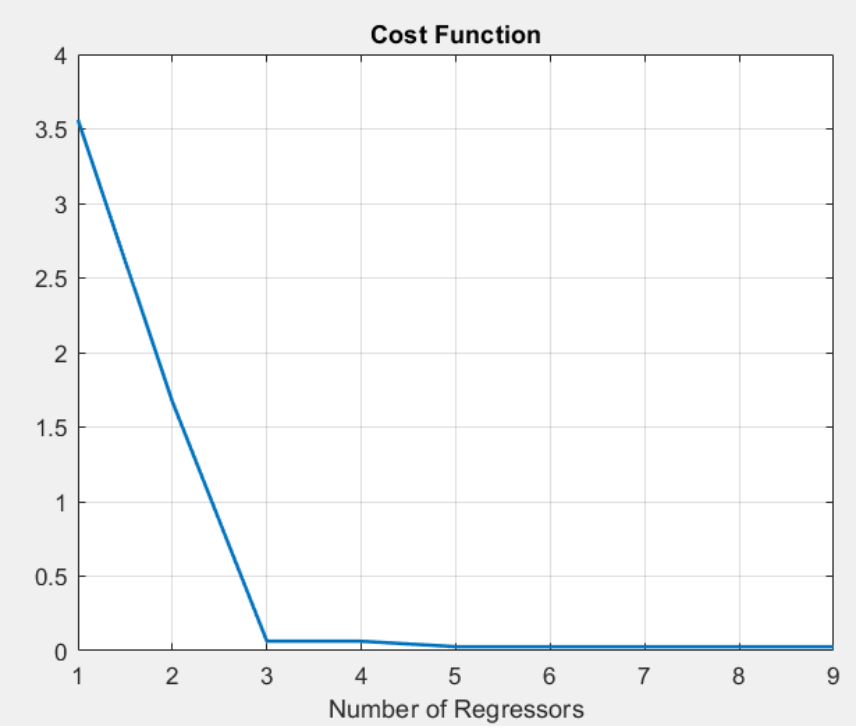


Plot for the cost based on number of regressors in FS algorithm and Low noise

Plot of error between estimated output and the actual output in FS and Low noise

The error is a small amount so is the noise. So we can say the algorithm has worked properly.

Medium Nosie:



Plot for the cost based on number of regressors in FS algorithm and Medium noise

Plot of error between estimated output and the actual output in FS and Medium noise

Cost Function =

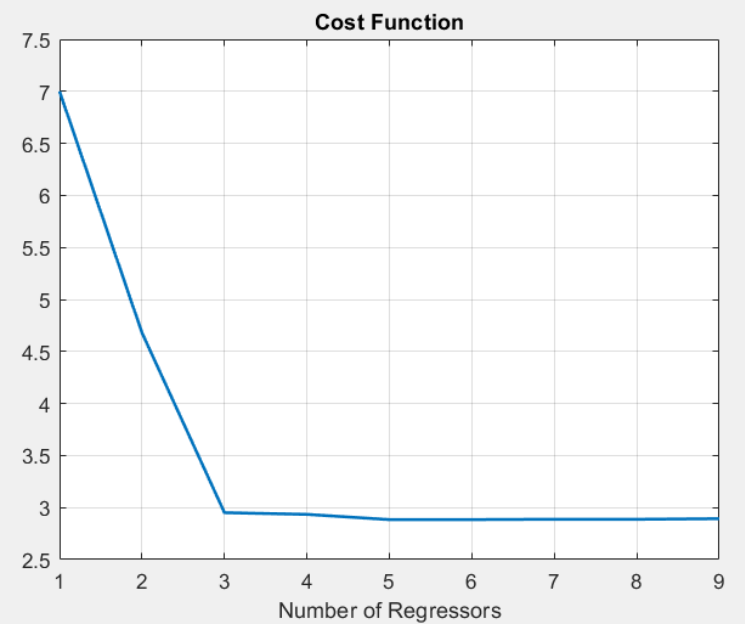
3.5604 1.6704 0.0639 0.0637 0.0274 0.0273 0.0274 0.0274 0.0274

Order Of Important Regressors Forward Selection =**1 2 4 9 7** 3 8 6 5

MSE of Error in Test Data = 2.5462e-04

High Noise:

Plot for the cost between estimated output and the actual output in FS algorithm and High noise



Plot of error based on the number of regressors in FS and High noise

Order Of Important Regressors Forward Selection = **1 2 5 9 7** 3 4 6 8
Cost Function =
7.0010 4.6798 2.9508 2.9344 2.8841 2.8843 2.8874 2.8871 2.8930
MSE of Error in Test Data =
0.0117

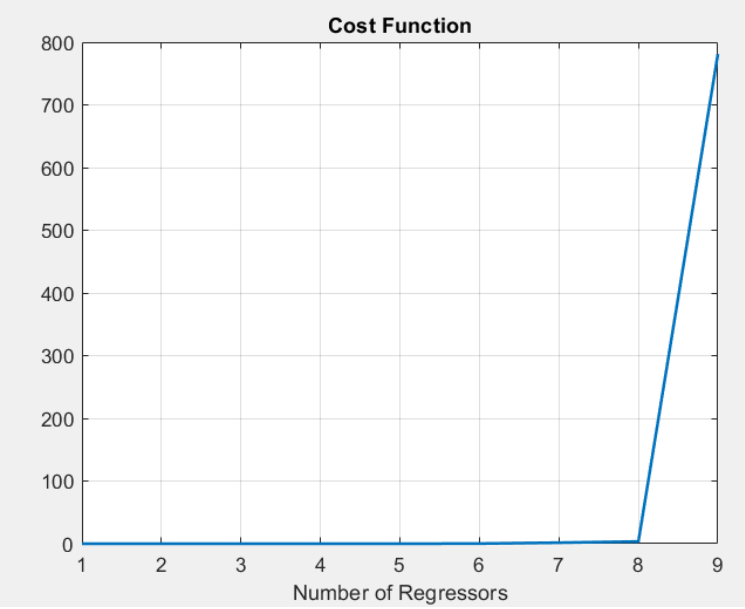
Backward Elimination:

One of the methods of subset selection is backward elimination for selecting a subset of main regressors from a set of n regressors. Which we discuss below:

1. First we implement a n-regressor estimation and evaluate the algorithm.
2. Then perform n (n-1)-regressor estimation, in which each time we eliminate a regressor, and evaluate each of the estimation using the test data to calculate the error and choose the one estimation that has the least error reduction, the eliminated regressor is that.
3. Then we preform $n-k+1$, (n-k)-regressor estimate, which in each estimation we eliminate one regressor. Evaluate the estimations and choose the one with the least error reduction, the eliminated regressor is that.
4. We carry on till we get the desired error or the error doesn't go down any further.

Without Noise:

We evaluate the estimation with the test data.



Plot of error based on the number of regressors in BE and without Noise

Order of delete Regressors in Backward Elimination =**5 8 6 3 4** 9 7 2 1
5, 8, 6, 3 and 4 will be eliminated first. **So our main regressors are 9, 7, 2 and 1**.
Equation for the output will be as below:

$$y = -1.5 - 0.8u + 2.25u^6 - 1.7u^8$$

Or if we exclude the 4th regressor we can get the following equation (then main regressors are 4, 9, 7, 2 and 1):

$$y = -1.5 - 0.8u + 0.01u^3 + 2.25u^6 - 1.7u^8$$

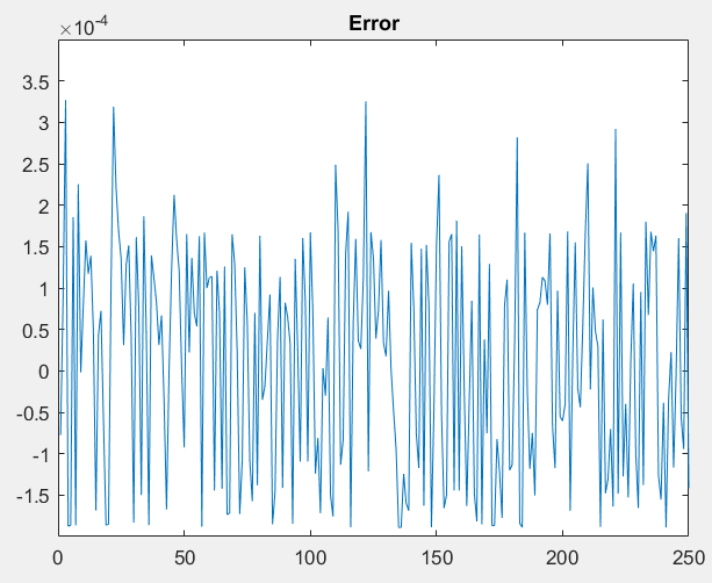
Cost Function =

0.0000 0.0000 0.0000 0.0000 0.0000 0.2298 1.6634 3.5403 781.0697

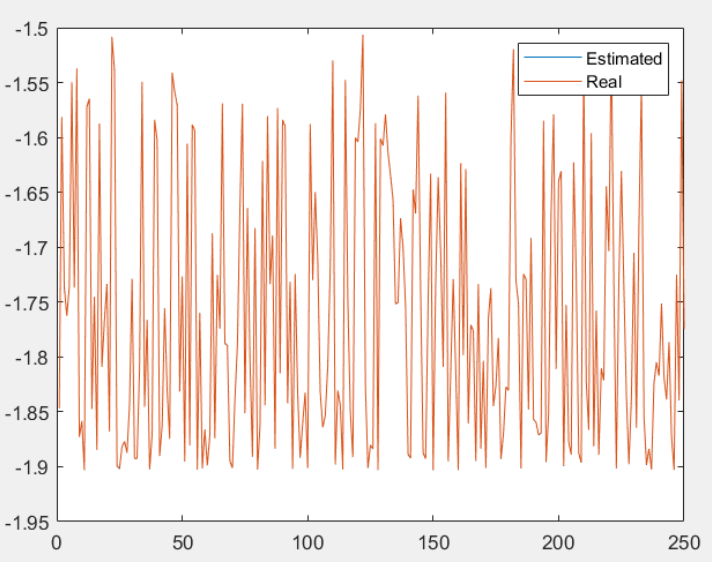
MSE of Error in Test Data =

1.8112e-08

If we perform the estimation using 4 main regressors, the results will be:



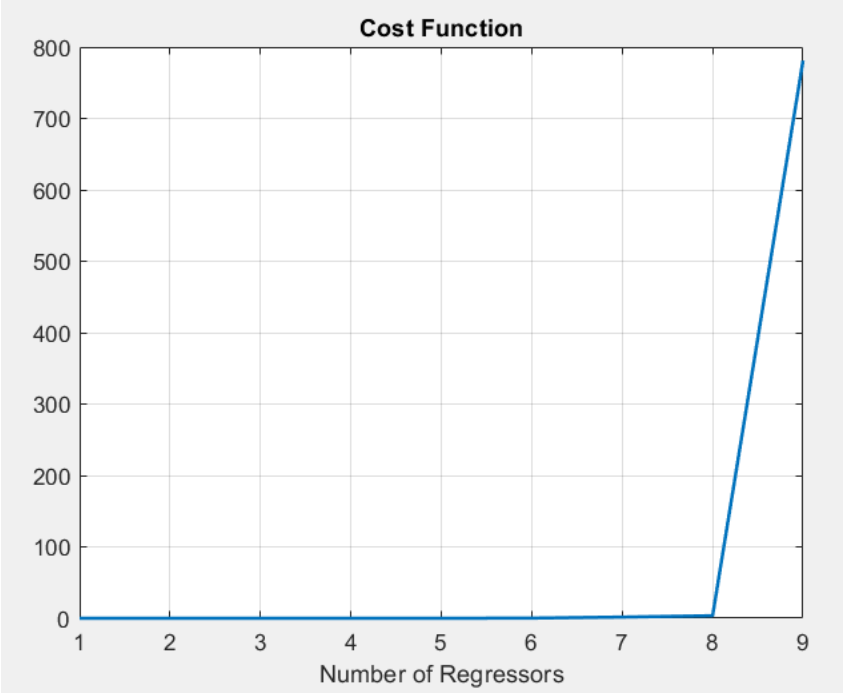
Plot of error between estimated output and the actual output in BE and Without Noise



Comparison between estimated output and the real output Without Noise

The error and noise look alike. So it is indicative of a good estimation in the absence of noise.

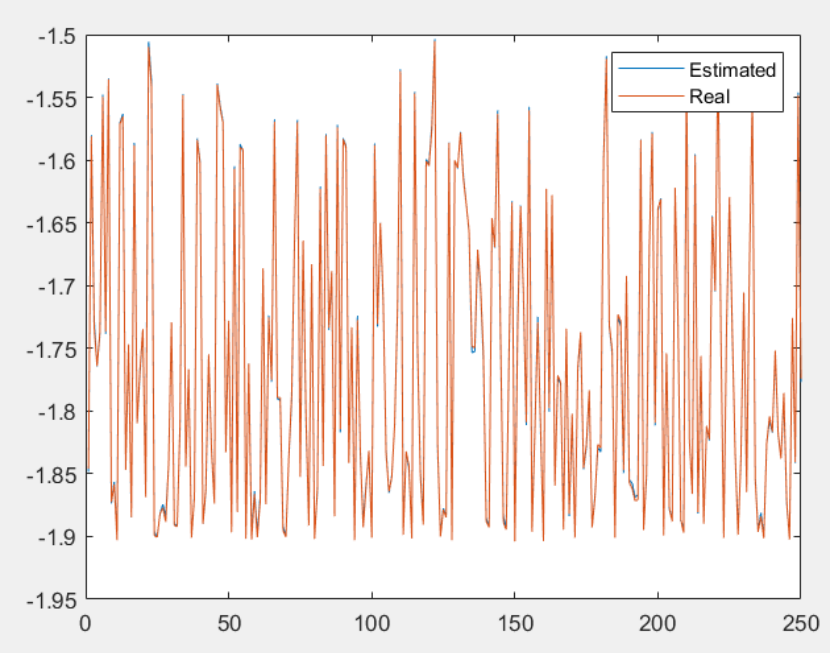
Low Noise:



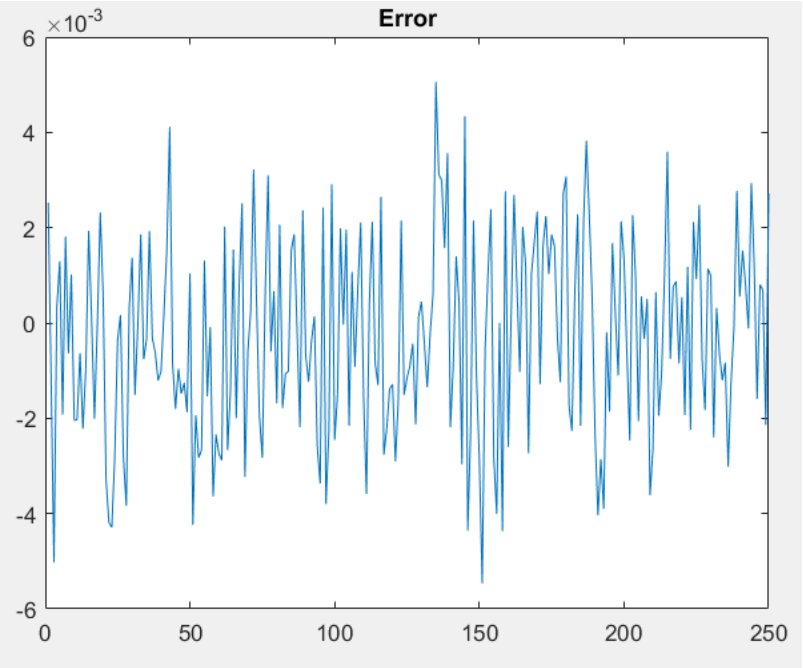
Plot of error based on the number of regressors in BE and with Low Noise

Order of delete Regressors in Backward Elimination =9 3 4 5 6 **8 7 2 1**
Cost Function = 0.0002 0.0002 0.0002 0.0003 0.0011 0.2301 1.6617 3.5396 781.0908
MSE of Error in Test Data = 4.2552e-06

Choosing 8, 7, 2 and 1 as the main regressors and conducting the estimation we get the results:



Comparison between estimated output and the real output With Low Noise

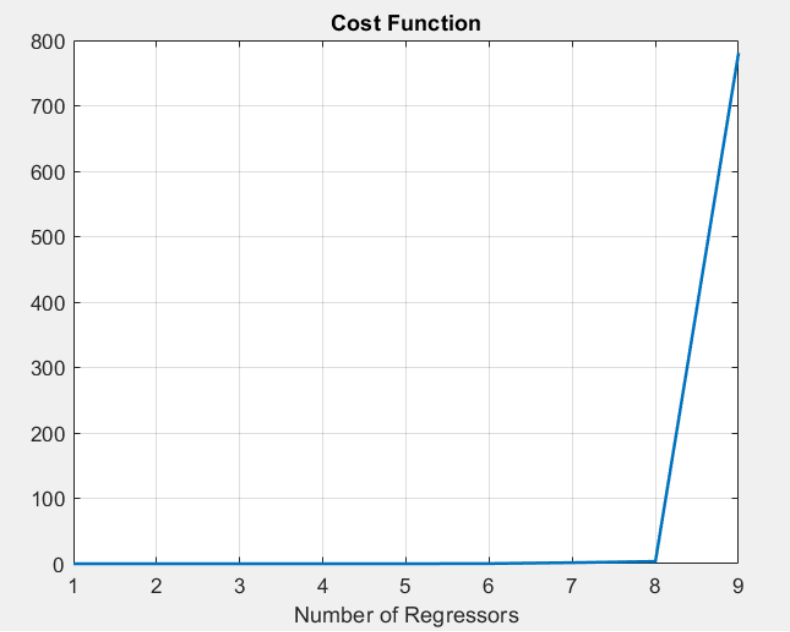


Plot of error between estimated output and the actual output in BE and With Low Noise

The error looks like noise. This is a good sign☺

The performance of the BE in the presence of Low Noise quite acceptable.

Medium Noise:

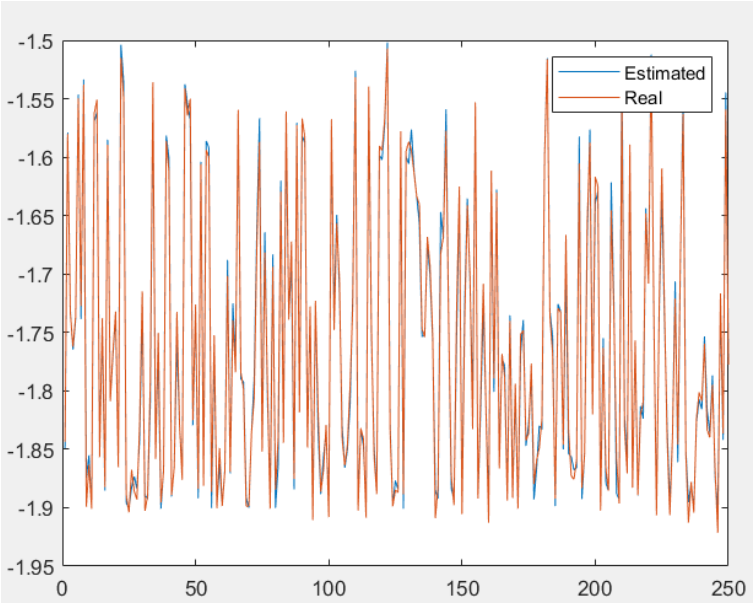


Plot of error based on the number of regressors in BE and with Medium Noise

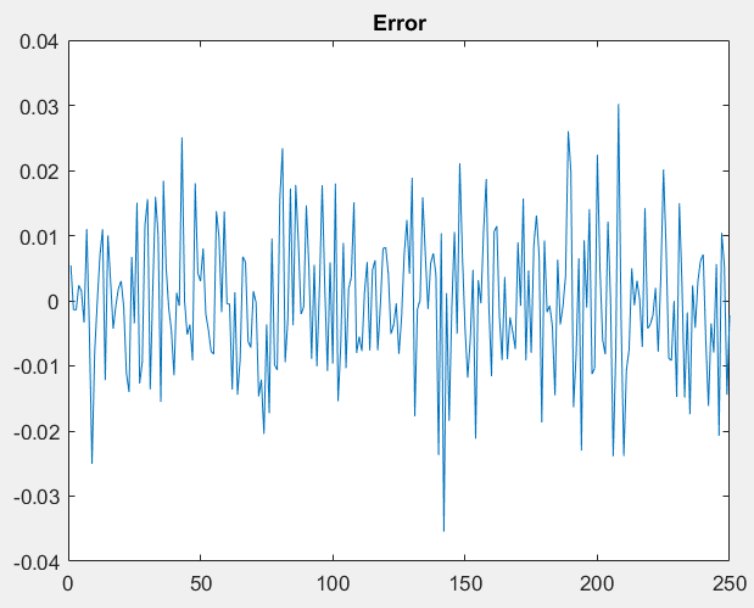
Order of delete Regressors in Backward Elimination = 3 9 5 4 6 **8 7 2 1**
Cost Function =
0.0274 0.0274 0.0274 0.0274 0.0284 0.2504 1.6704 3.5604 780.6181
MSE of Error in Test Data = 1.1353e-04

Let's not drag it out, we keep It simple then.

As before we eliminate the first five and the rest will be our main regressors.



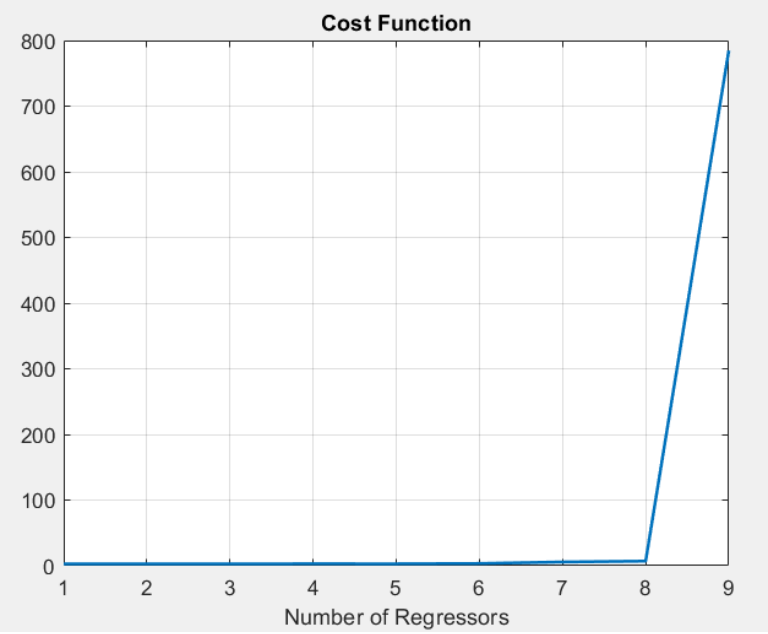
Comparison between estimated output and the real output With Medium Noise



Plot of error between estimated output and the actual output in BE and With Medium Noise

The estimation is relatively good in the presence of medium density noise.

High Noise:



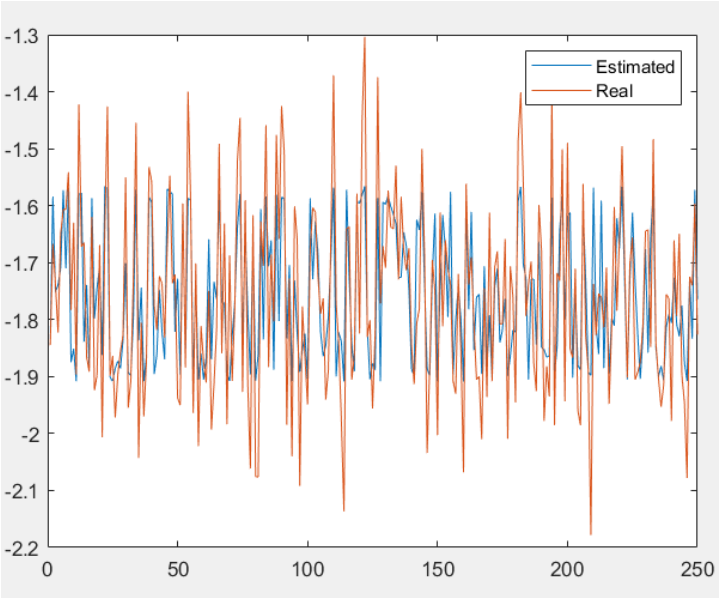
Plot of error based on the number of regressors in BE and with High Noise

Order of delete Regressors in Backward Elimination = 9 2 8 4 5 **7 6 3 1**
Cost Function =

2.8847 2.8788 2.8984 2.9277 2.9134 3.4007 5.6960 7.0010 784.6914

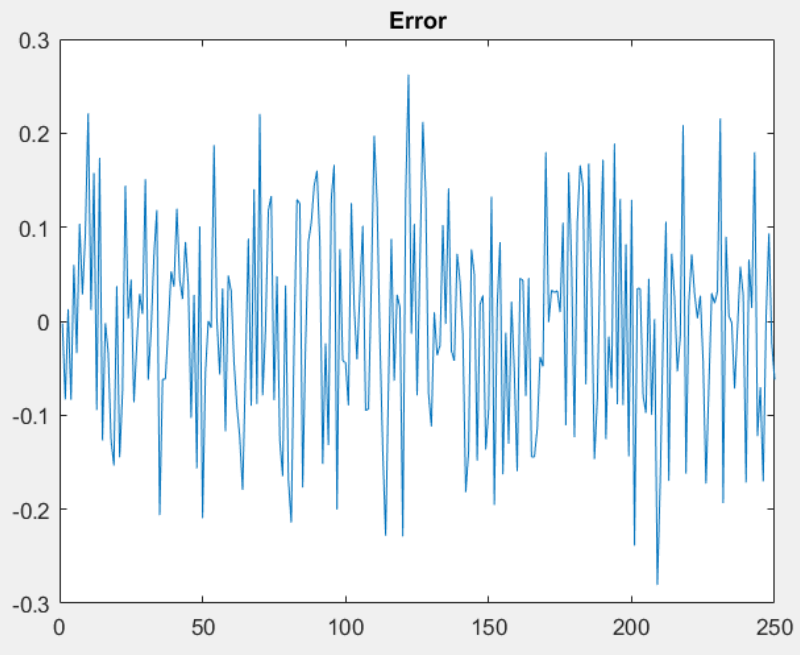
MSE of Error in Test Data = 0.0117

Eliminate the first five and the rest will be our main regressors, which are 7, 6, 3 and 1.



Comparison between estimated output and the real output With High Noise

Not a bad estimation we see above.



Plot of error between estimated output and the actual output in BE and With High Noise

The noise is like above.

It’s noticeable that with even subset selection the noise is still high. In comparison to the FS method, the cost values are almost equal with little to zero difference.

The BE method in comparison to FS, on noisy data, has bigger errors. But when we don’t have the system equations both methods are good. But there is a difference between in the order of main regressors. And both methods, because of non-orthogonality of regressors are sub optimal. And in this case because the number of regressors are small the computational cost for FS is smaller than BE, so FS is proposed for such problems.

Implementation Question 1.7

In BS and FS because of the non-orthogonality between regressors, the optimized parameters are different in each step. If we could do something that makes the regressors orthogonal, this will decouple parameters, and we can calculate the value for only one new parameter independent of others in each step. And the computational cost in this method is the orthogonalization. The important point is that the resulting orthogonal vectors are not just vectors, with this transformation the physical vectors v change their physical concepts.

OLS algorithm is as below:

1. Using each regressor we perform a one-regressor estimation. Based on the value of I (bigger) we choose the main regressors. Which is basically this:

$$
\begin{aligned}
&V_1^i = x_i && i \in \{1,2,\dots,n\}\\
&\eta_1^i = \frac{V_1^{iT} Y}{V_1^{iT} V_1^i} && i \in \{1,2,\dots,n\}\\
&err_1^i = \left(\eta_1^i\right)^2 \left\|V_1^i\right\|^2 && i \in \{1,2,\dots,n\}\\
&i_1 = \arg\max\left\{err_1^i\right\}\\
&V_1 = V_1^{i_1} = x_{i_1}
\end{aligned}
$$

2. Then using the Gram-Schmidt algorithm, we orthogonalize the space. We create a set of new orthogonal vectors in the space using the non-orthogonal vectors. It is carried out using the algorithm below:

for $\mathrm{k} = 2,3,\dots,\mathrm{n}_s$

$$
\begin{aligned}
&\text{for } i \in \{1,2,\dots,n\} - \{i_1, i_2, \dots, i_{k-1}\}\\
&\alpha_j{}^i = \frac{V_j{}^T x_i}{V_j{}^T V_j} \; j \in \{1,2,\dots,k-1\}\\
&V_j{}^i = x_i - \sum_{j=1}^{k-1} \alpha_j{}^i V_j \;(\text{Gram} - \text{Schmidt})\\
&\eta_k{}^i = \frac{V_k{}^{iT} Y}{V_k{}^{iT} V_k{}^i}\\
&\text{err}_k{}^i = \left(\eta_k{}^i\right)^2 \left\|V_k{}^i\right\|^2
\end{aligned}
$$

end i

$$i_k = \arg\max\{\text{err}_k{}^{i}\}$$

$$V_k = V_k{}^{i_k} = x_{i_k} - \sum_{j=1}^{k-1} \alpha_{jk}{}^{i_k} V_j$$

end k

if we finish the OLS algorithm the parameters estimated by this algorithm will be:

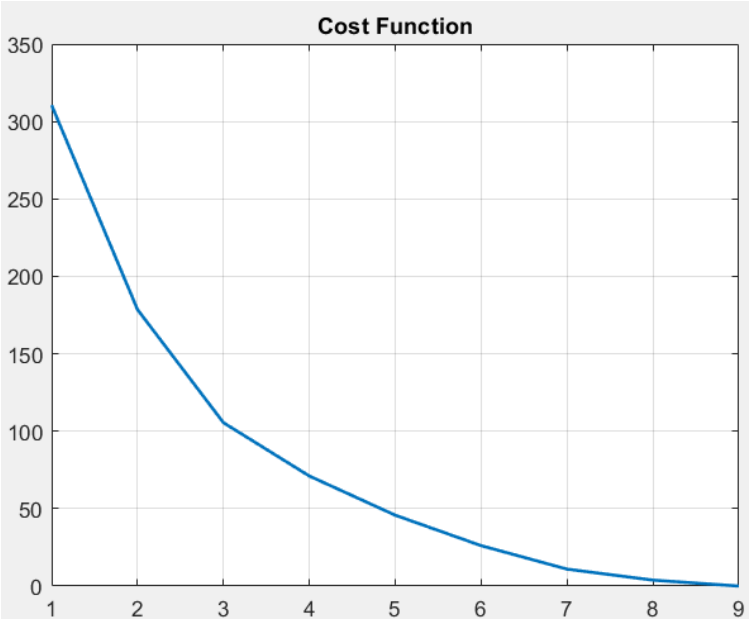
$$\theta = [-1.5 \quad -0.8000 \quad 1.2e-17 \quad 0.0100 \quad -9.7e-17 \quad -0.6500 \quad 2.2500 \quad 3.6e-15 \quad -1.700]$$

FS-OLS Algorithm

The parameters estimated here are much more close to real values.

Now if we give the FS algorithm the orthogonal space, the main regressors are as below:

Order Of Important Regressors Forward Selection (In orthogonal space) =3 4 5 6 7 2 8 9 1

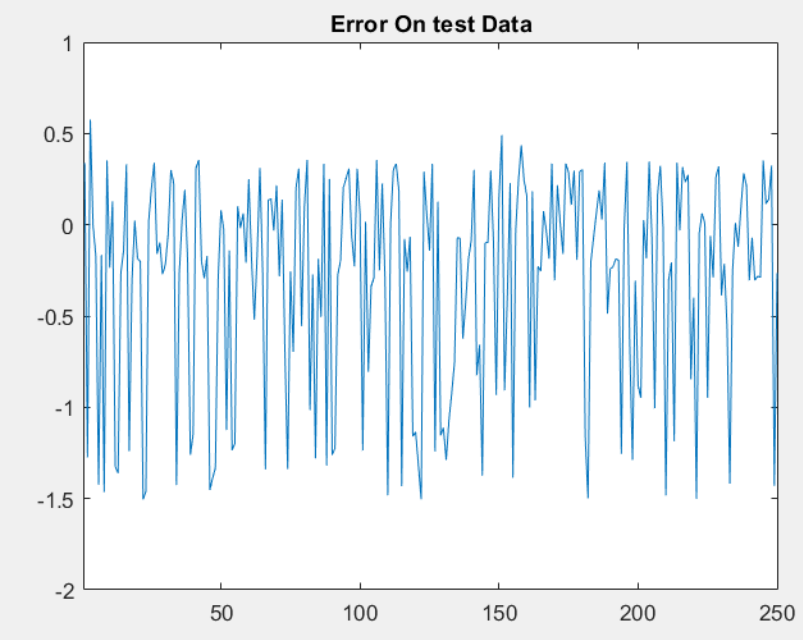


Plot of error based on the number of regressors in FS

Cost Function =

310.5122 178.5211 105.5148 71.0633 45.7586 26.0943 10.9367 3.8353 0.0000

We choose the three bold regressors we achieve an error as above. With only three regressors, we can estimate the system with reasonable accuracy. We can add a fourth regressor but nothing significant will take place, in comparison to the computational cost.



Plot of error between estimated output and the actual output in OLS-FS

MSE of Error in Test Data = 0.4221

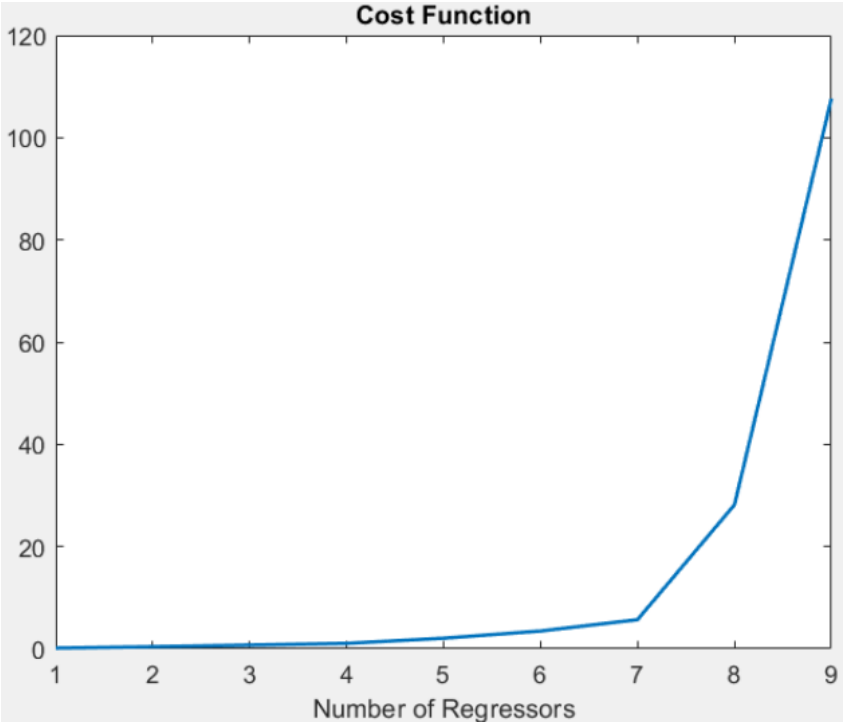
In the code implementation of the problem, just adding noise from choices noise1, noise2 and noise3 would suffice. We just avoided doing it here.

BE-OLS Algorithm

If we give the algorithm the orthogonal space, the order of regressor elimination will be a below:

Order Regressors Backward Elimination (In orthogonal space)=9 8 7 1 6 5 **4 3 2**

The regressors 9, 8, 7, 1, 6 and 5 are going to be eliminated.
And regressors 4, 3 and 2 are the main regressors.



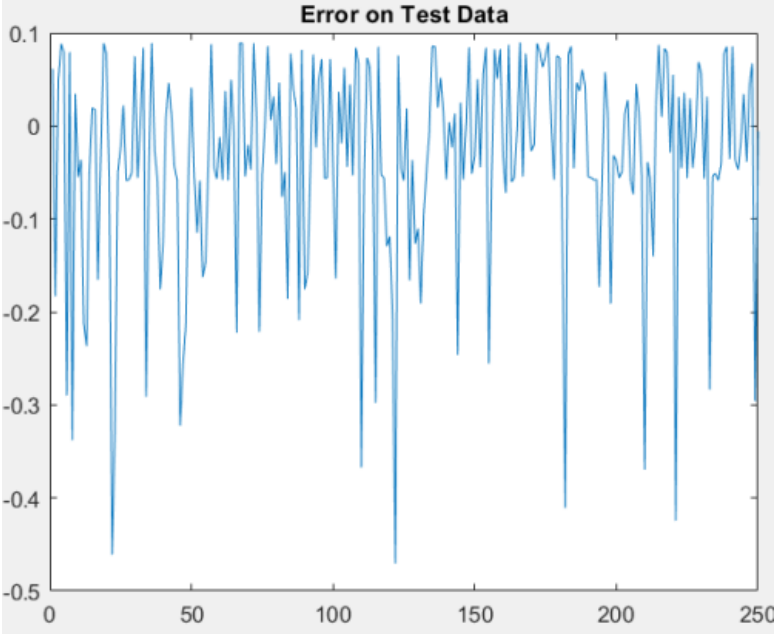
Plot of error based on the number of regressors in BS

Cost Function =

3.8353e-31 10.9367e-30 26.0943 e-30 45.7586e-29 71.0633e-07 1.0234e-04 0.0059 0.5122 32.5418

By elimination of six regressors we have the cost a little reduced, so if we eliminate a smaller number of regressors, not much good is going to be done to the model, and we keep the last three regressors in orthogonal space.

There might be something wrong with either my FS OLS or BE OLS that I get very different results, I spent a lot of time but I couldn't figure it out.



Plot of error between estimated output and the actual output in OLS-BE

In the presence of High Noise the order will be as this:

Order Regressors Backward Elimination (In orthogonal space) = 8 9 7 1 5 6 **4 3 2**

And the $MSE\ (Error\ on\ Test\ Data) = 0.0255$ the MSE is increased, and we have two changes in the order of elimination of regressors. We can deduce that in presence of noise the FS method is better, because it's more consistent, and has a slight better performance.

As can be seen, both BE and FS have almost the same outcome, and let aside my mis-implementation☺ they should yield to the same results, because the space is orthogonal.

And don't forget a regressor in the orthogonal space could be composed of different regressors. And cause the space is orthogonal the algorithm goes towards optimality.

Implementation Question 1.8

The Idea behind the Ridge Regression fir distinguishing between main and secondary regressors, is that in the parameter estimation formula we make some changes:

$$\hat{\theta} = (X^T X + \alpha I)^{-1} X^T Y$$

Change the parameter α and examine the changes in the estimated parameters. Those parameters that have a slight change to α are actually the parameters related to the main regressors. For example for this case the trend for changes in α is as below:

$$\alpha_i = i^{0.001} \quad , \quad i = 0,1,2,\dots.,1000$$

For $i = 1000$ we record the changes in the estimated parameters.

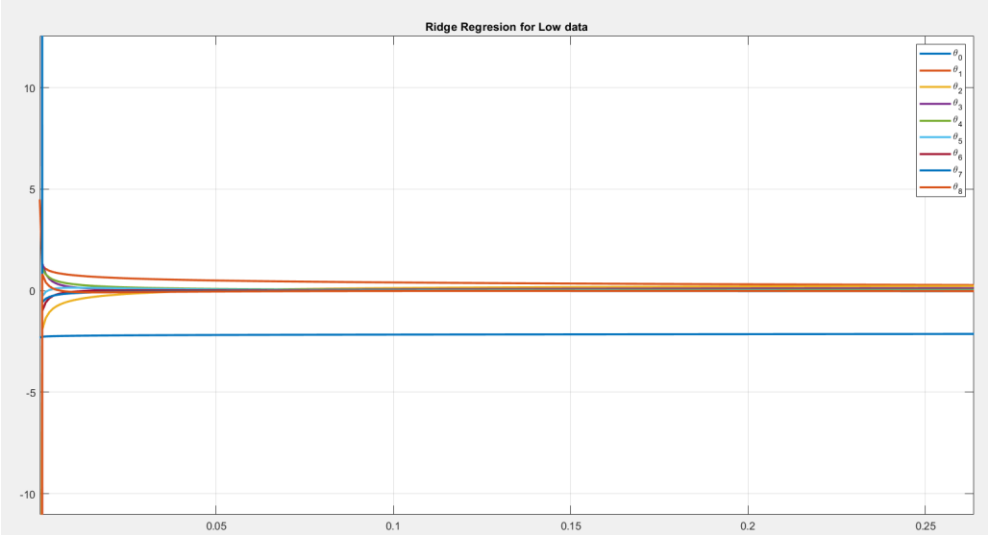
$$\widehat{\theta_i} = (X^T X + \alpha_i I)^{-1} X^T Y$$

Then for each member of the parameter vector, evaluate the dispersion and plot it. Depending on the range and trend of changes in α, it's possible that a set of different regressors are the main regressors.

The plots of changes in the estimated parameters are plotted in these figures:

Note: I tried really hard to distinguish between different regressors but plots are a little too much indistinguishable☹. I mean this is the process of finding the main regressors please be a little nice, when you are grading my homework.

Small Noise Density:



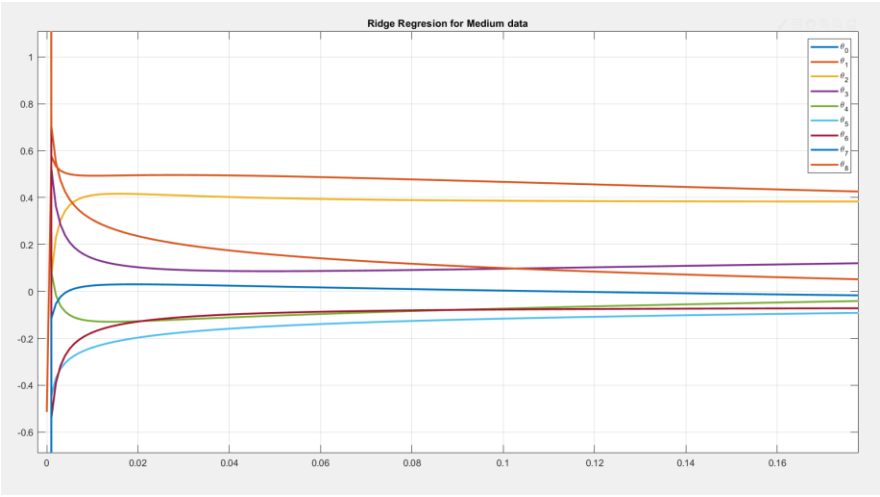
Plot of changes of regressors based on the parameter α and Low Noise

In this case it's a little hard to distinguish between main and secondary regressors. There might be a bug in my code implementation, but we proceed with the rest of the work.

We first need to omit the curves with the most variations.

For low noise density part, we have $\theta_4, \theta_5, \theta_7, \theta_8$ as the main regressors.

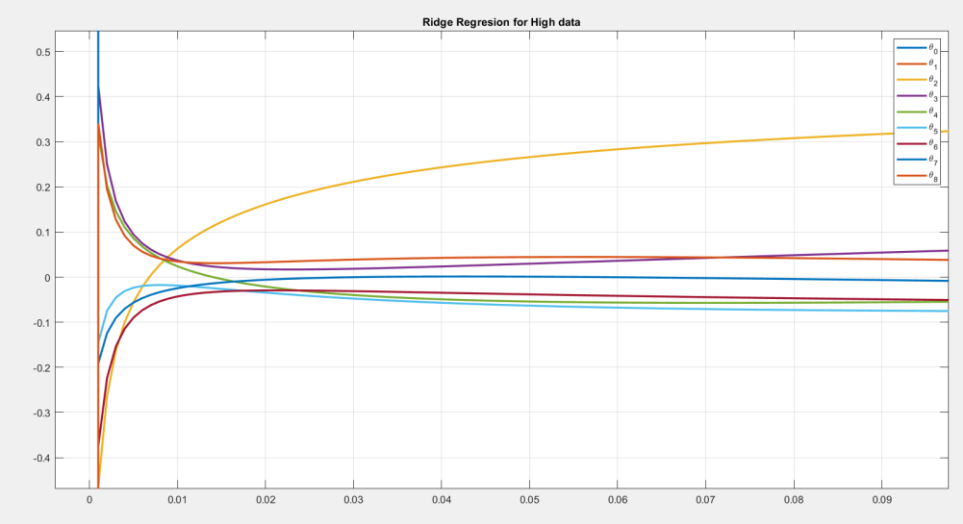
Medium Noise density:



Plot of changes of regressors based on the parameter α and Medium Noise

For Medium noise density part, we have $\theta_1, \theta_2, \theta_3, \theta_4$ as the main regressors.

High Noise Density:



Plot of changes of regressors based on the parameter α and High Noise

This time it's even harder! Based on the variations and eye examination, main regressors are, $\theta_3, \theta_4, \theta_5, \theta_6$.

Implementation Question 2

Implementation Question 2.1

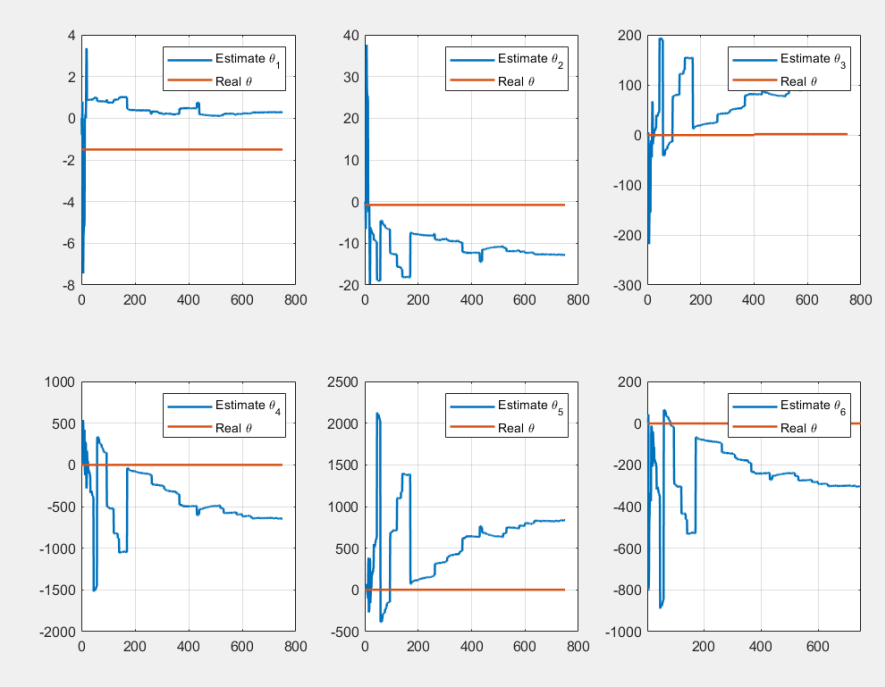
In this question we intend to see the negative effect of changing parameters over time on our RLS estimation and examine it. So all we need to do is we change the parameter θ_3 to a time varying function:

$$\theta_3 = 0.5 + tansig(g(t - 400))$$

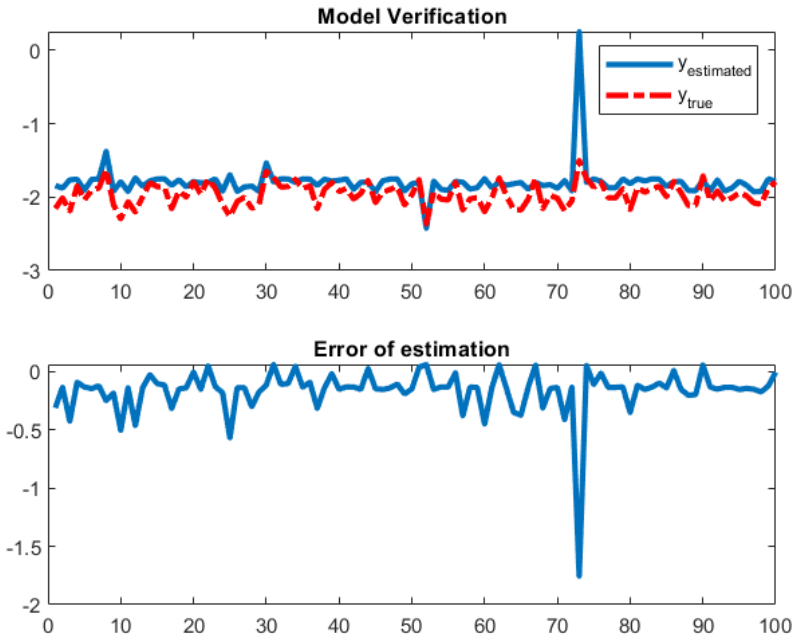
The regressor vector we are going to use in this part is :

$$\overline{x^q} = \begin{bmatrix} 1 & u_q & u_q^3 & u_q^5 & u_q^6 & u_q^8 \end{bmatrix}$$

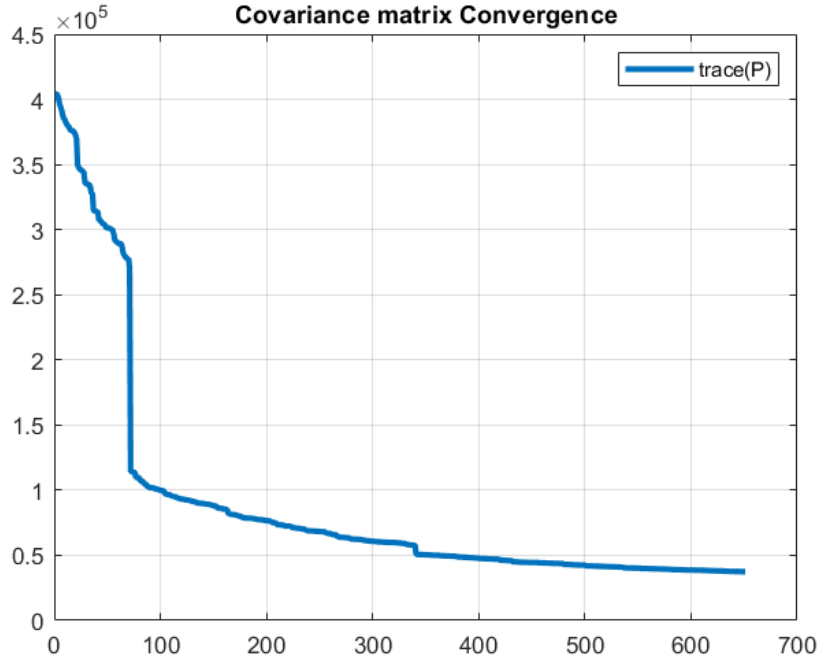
With g parameter equal to 1, the results are as follows:



Estimation of system parameters using θ_3



Estimated output, real output, estimated error and real error



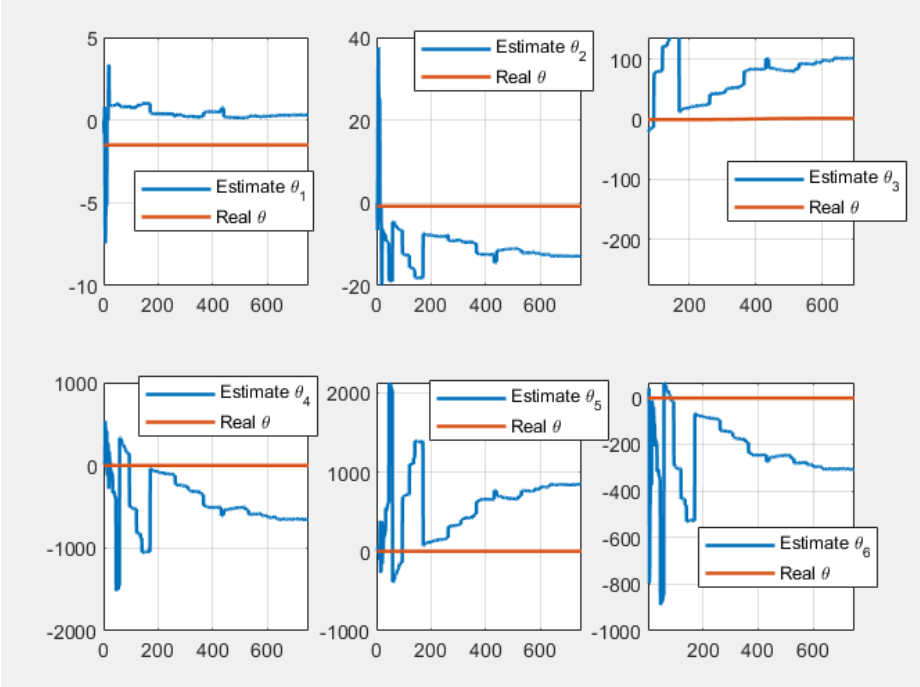
Trace of covariance matrix P for RLS estimation

When there are not stiff changes (before t = 400) estimation is getting closer and closer to the real values. Nut after t=400 we see the algorithm fails to follow changes. And beside that error plot is indicative of a not-so-good estimation.

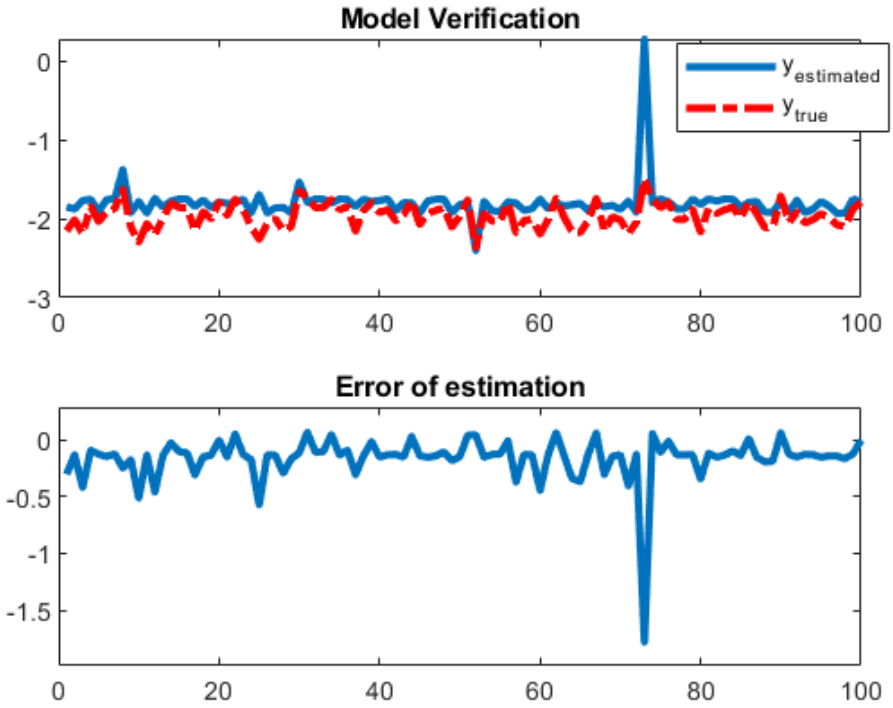
The reason behind the fact that after t=400 algorithm fails, is that the covariance matrix is fairly small so that the algorithm can't change the parameters. And after that eigenvalues are relatively small and this is another reason that the algorithm failed. We can say the more the eigenvalues are bigger, the better and faster the estimation and tracking will be conducted.

Implementation Question 2.2

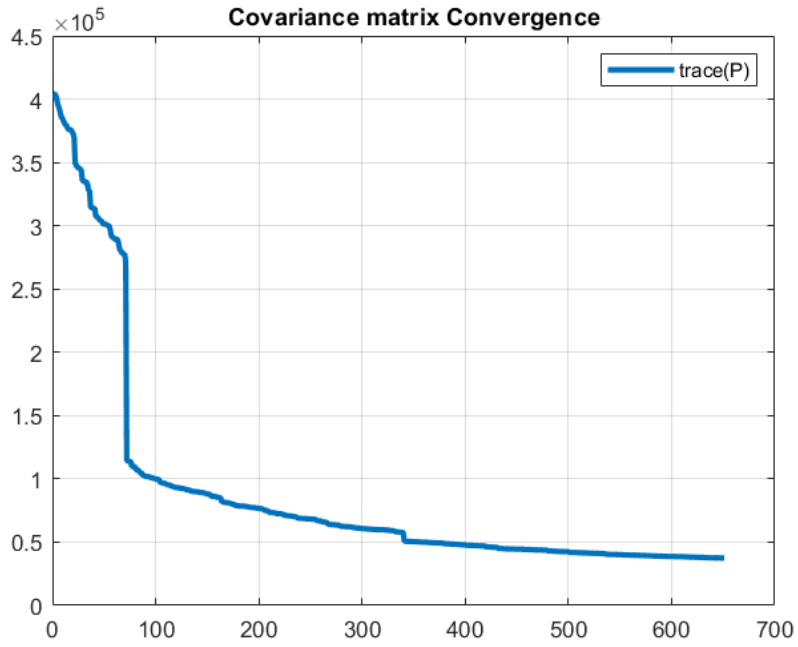
Now we examine the effect of a much lower g, which is $g = 0.01$.



Convergence of system parameters using θ_3 and $g = 0.01$



Output, estimated output, estimation error of the RLS algorithm using θ_3 and $g = 0.01$

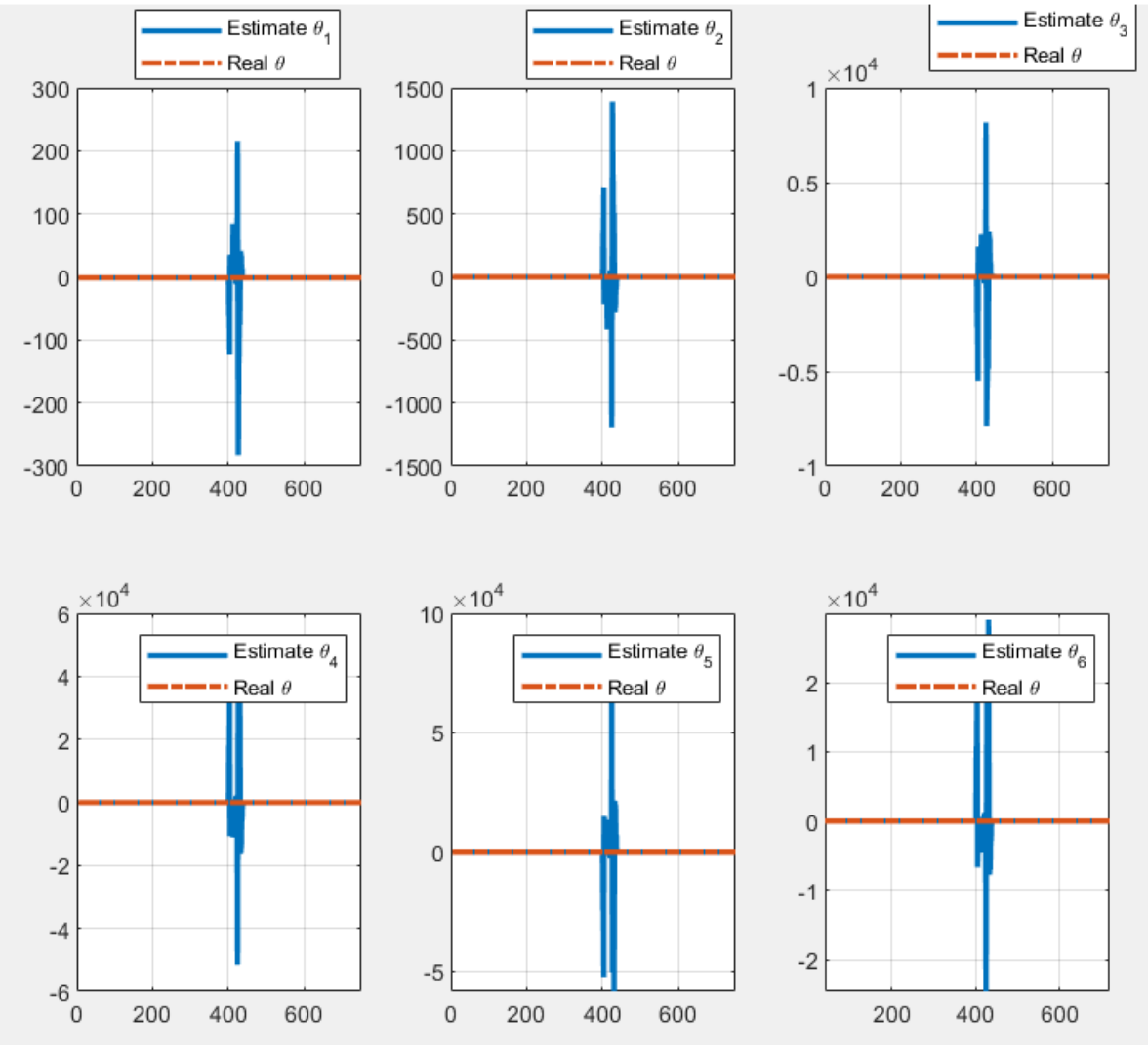


Trace of covariance matrix P, in RLS estimation and $g = 0.01$

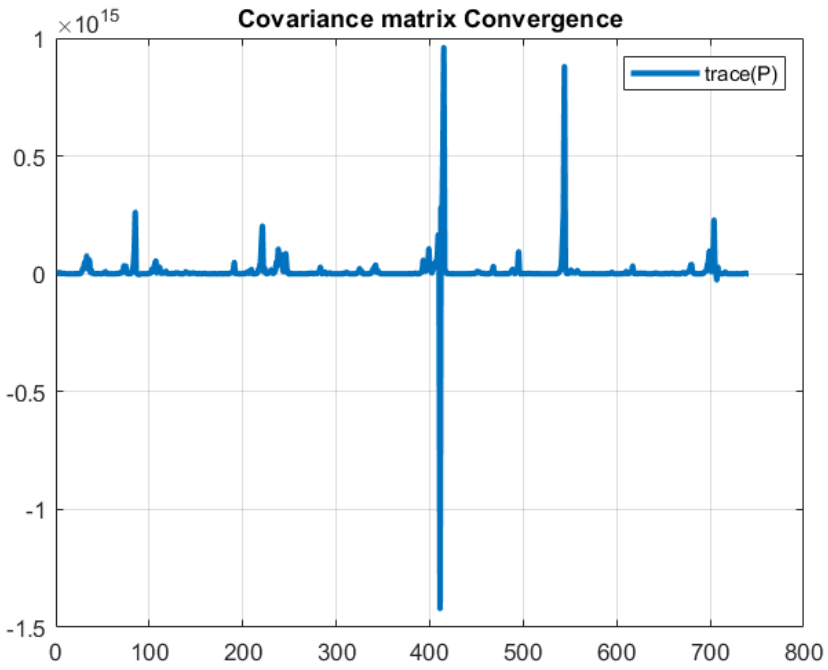
Conclusion: The more we change the g the worse the RLS estimation gets. Because g is actually the gain inside the function. $tansig$ is a continuous function that small changes inside the argument will result in smoother change. Increasing g will provoke the function's argument to grow faster and this action will result in the regressor changing faster. In this case, when the regressor is changing fast, the covariance matrix moves towards zero, the tracking takes place slower. Therefore changing g to a smaller amount will make the tracking faster and more acceptable.

Implementation Question 2.3

The FFRLS is a method that uses a forgetting factor to give more importance to recent data than the old data. And this factor makes thee covariance matrix move slower. And the consequent of this algorithm is that it forgets the old data faster and when there is a changing parameter in the system this might be useful. But the most important thing in this algorithm is choosing this forgetting factor, which is not an easy task. If the $\lambda = 0.3$ we get:



Estimated parameters using θ_3 and FFRLS method



Trace of covariance matrix P using FFRLS

Conclusion:

It can be seen using forgetting factor we can create an oscillating peak that's going to alleviate the changes in the parameter and properly follow the parameters changing and convergence to the actual value of parameters can be reached.

Implementation Question 2.4

In Kalman Filter we have:

$$\theta(t+1) = \theta(t) + K(t)\varepsilon(t)$$

$$\Sigma(t+1) = \Sigma(t) + Q - \frac{\sum(t)X(t)X^T(t)\Sigma(t)}{1 + X^T(t)\sum(t)X(t)}$$

$$K(t) = (\Sigma(t+1) - Q)x(t)$$

$\varepsilon(t)$ is the estimation error in this case that is calculated like below:

$$\varepsilon(t) = y(t+1) - x^T(t+1) \times \theta(t)$$

Q matrix in this relationship is a diagonal matrix which is the noise covariance. The main diagonal, we can determine each value on the main diagonal ourselves, if we know how that system parameter is behaving, we can assign a value for the corresponding parameter based on how fast that parameter changes.

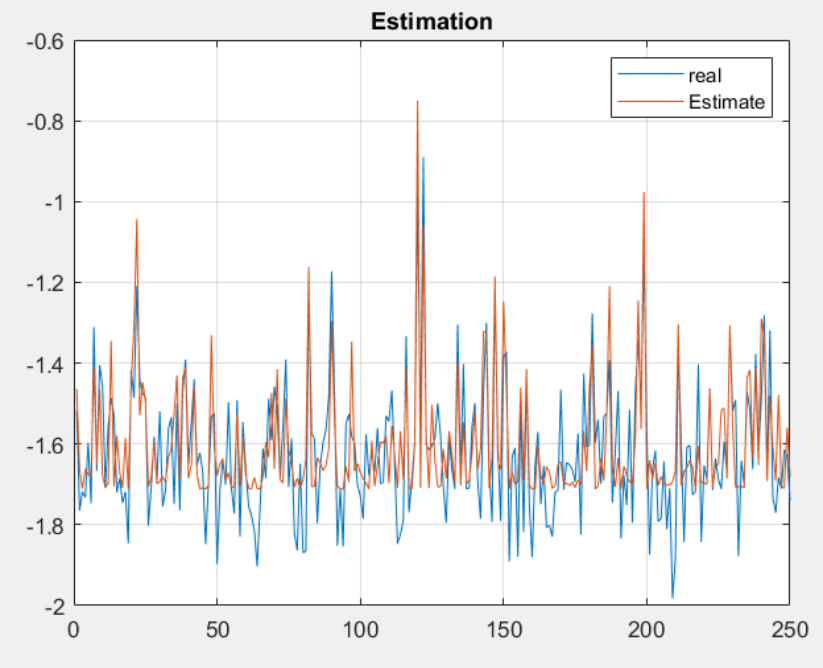
This is like applying forgetting factor for each individual parameter and actually this is the advantage of Kalman Filter algorithm compared to RLS, which only has one forgetting factor for all parameters. Matrix Q is the consequent of changing parameters.

Initial values for Kalman Filter are as below:

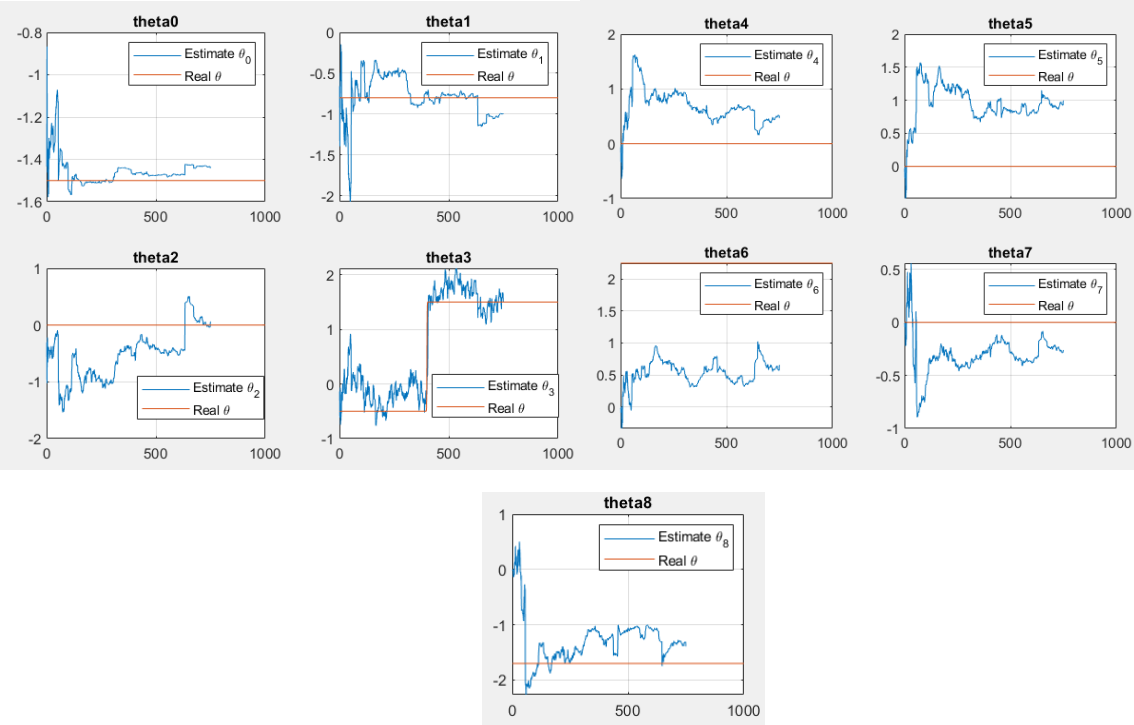
$$\Sigma(0) = 10^3 I, \theta(0) = \text{ zeros } (10,1)$$

In comparison to RLS and FFRLS, Kalman Filter is a better algorithm and yields to a better outcome with high variance noises. But if the parameters have aggressive changes the corresponding values in Q must be big enough, which this will result in matrix Σ becoming big and we don't get a good convergence.

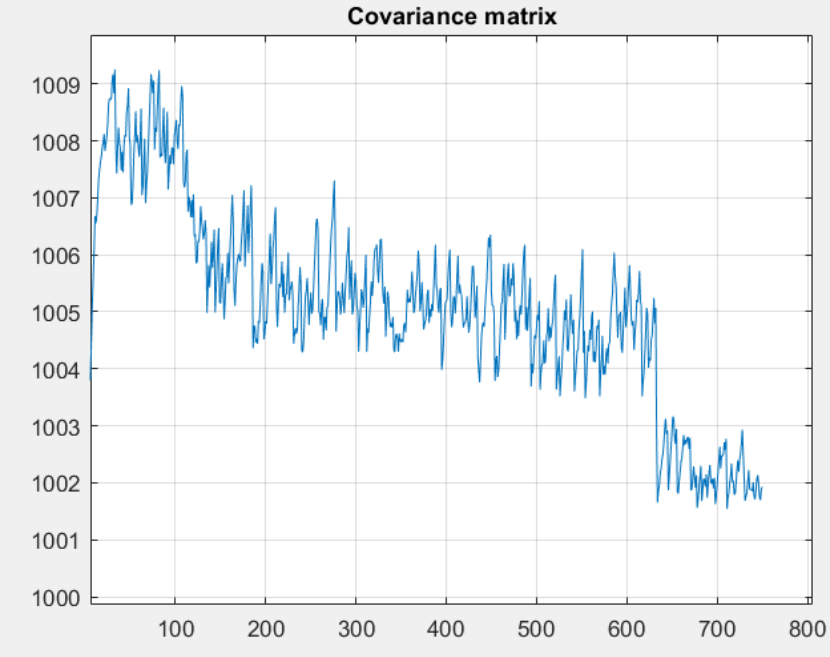
First case: g=1, High Noise:



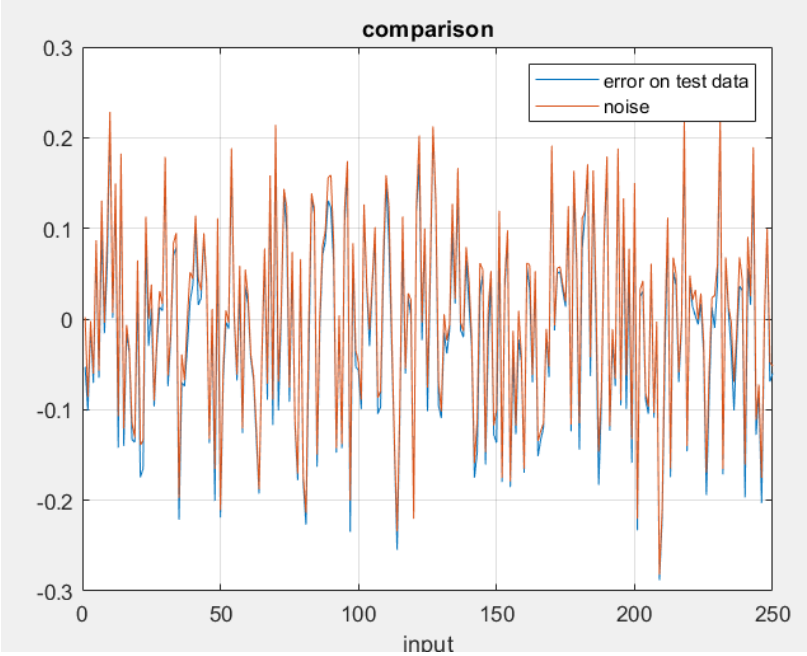
The actual output and the estimated output using Kalman Filter



Convergence of Parameters using the Kalman Filter



Norm of covariance matrix in Kalman Filter



Error from the test data and noise

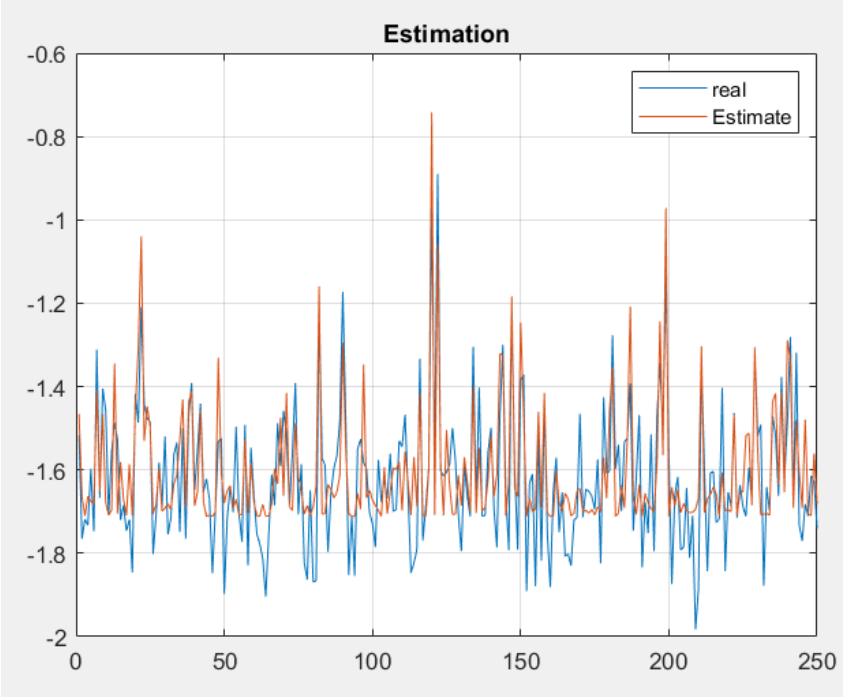
As can be seen from above the noise and error are pretty similar.

Mean squared normalized Error on test data = 0.0119

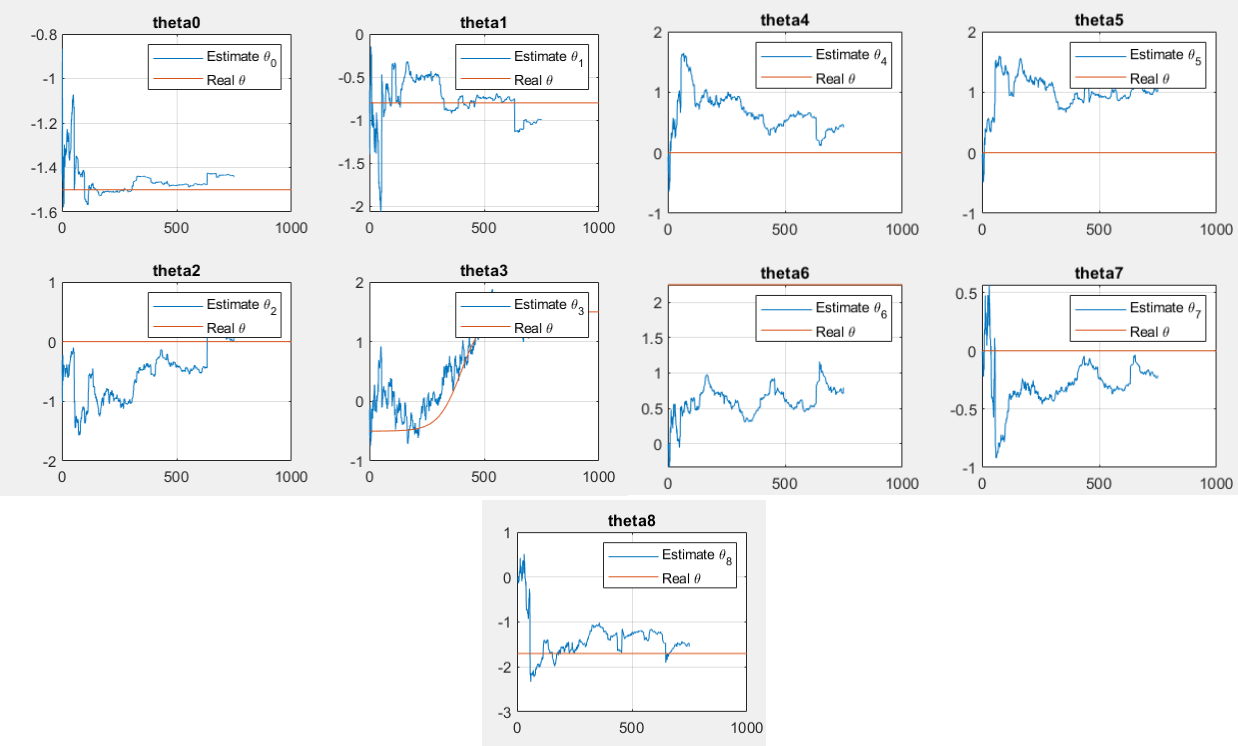
Mean of parameter errors = 0.5391

MSE of difference between error and noise = 0.0234

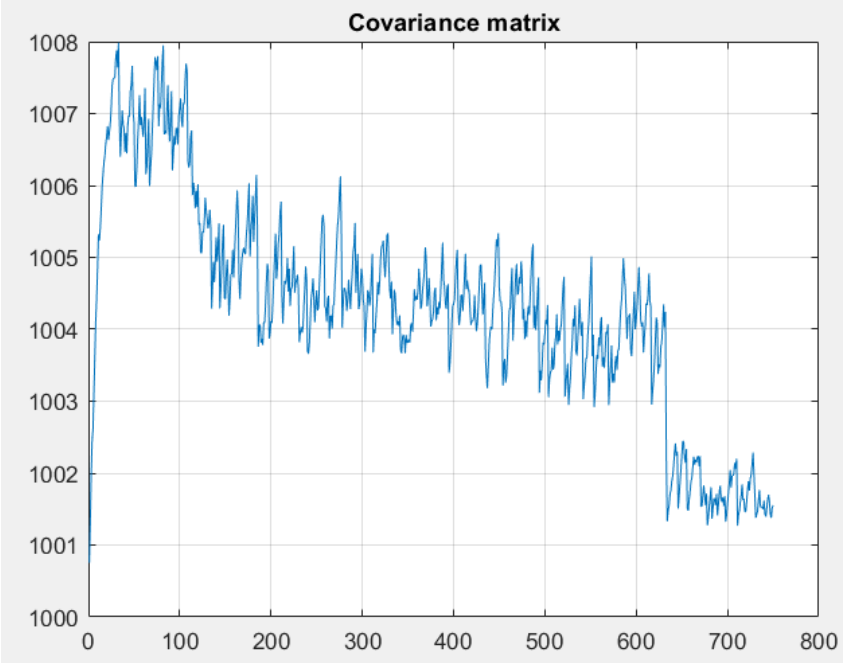
Second case: g = 0.01, High Noise



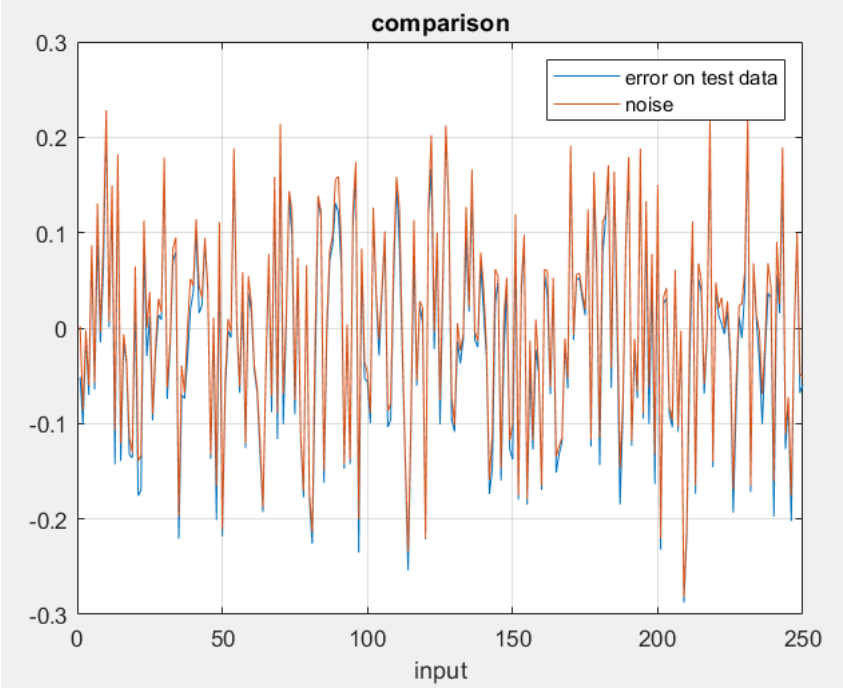
The output of the estimation and the real output g=0.01 and High Noise



Convergence of parameters using Kalman Filter, g = 0.01 and High noise



Norm of covariance matrix using Kalman Filter g=0.01 and High Noise



Error on the test data and the actual noise using Kalman filter g=0.01 and High Noise

According to the above Figure noise and error are pretty much similar

Mean squared normalized Error on test data = 0.0119

Mean of parameter errors = 0.4416

MSE of difference between error and noise = 0.0234

Analysis of Kalman Filter Method

Beside the fact that the error is small even for changing perimeter, the convergence is relatively good and the error and Nosie are identical as well. The difference between covariance of noise an error is very small for both cases (g = 1, High Noise and g = 0.01, High Noise).

And in sense of convergence, we see that for smaller g's, better convergence occurred, the estimated parameters are better and error becomes smaller.

From the covariance figures, we witness that the covariance doesn't apporch to zero fast, this means the estimation occurs better.

In comparison to RLS and FFRLS, the Kalman Filter method reduced **"Mean squared normalized Error on test"** drastically. And so does it for **"MSE of difference between error and noise"**

Implementation Question 3

Implementation Question 3.1

First we create 1000 data points with uniform distribution between [0,1] and split it between train and test, 75% and 25% respectively.

For data creation we use the $unifrand$ command in matlab, which create data with distribution. The more the power of regressors the less important those regressors get, so this means cause the data points are less than zero, if they are powered to a number bigger than one, they get smaller and the estimation for these regressors becomes less accurate.

For noise variance, we must choose a value proportionate to the amplitude of the regressors. If we choose a variance bigger than the amplitude of a regressor, that regressor is going to be estimated with a large error.

In this problem we take the following equation to be estimated:

$$y = -0.6X_1 + 2.25X_2 - 0.1X_3$$
$$X_3 = X_1 + X_2 + noise$$

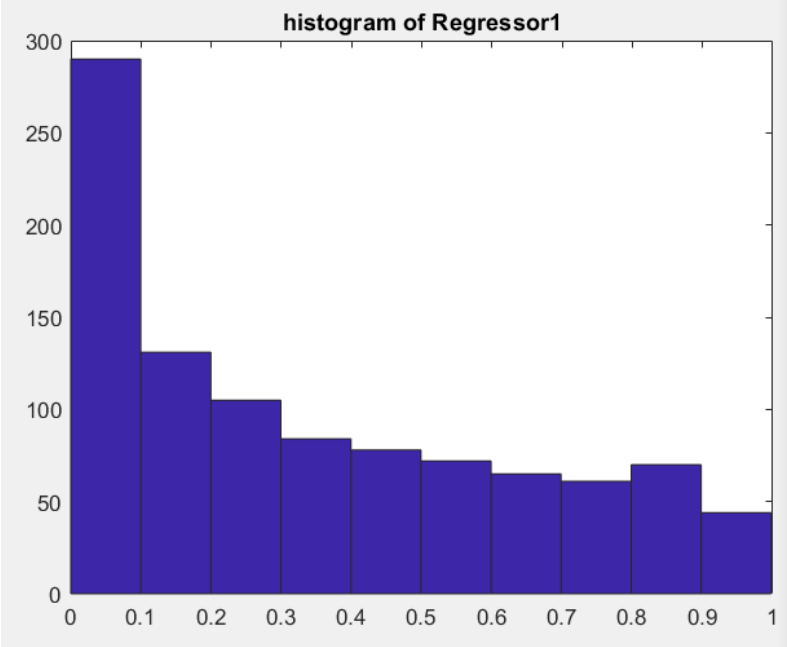
The vector of regressors is in the line below:

$$X = [X_1 \quad X_2 \quad X_3] = [u^2 \quad u^3 \quad u^2 + u^3]$$

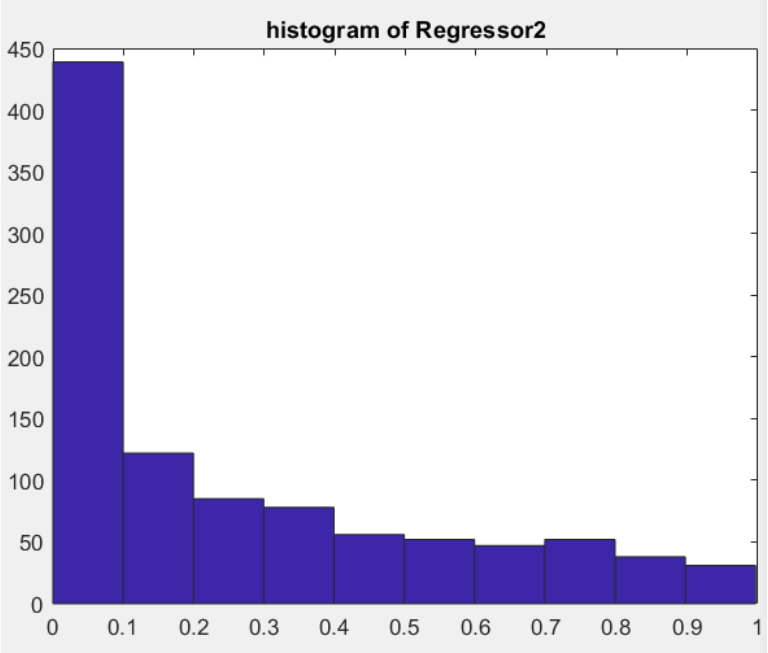
Taking the above into account, the equation would be:

$$y = -0.6u^2 + 2.25u^3 - 0.1(u^2 + u^3 + noise)$$

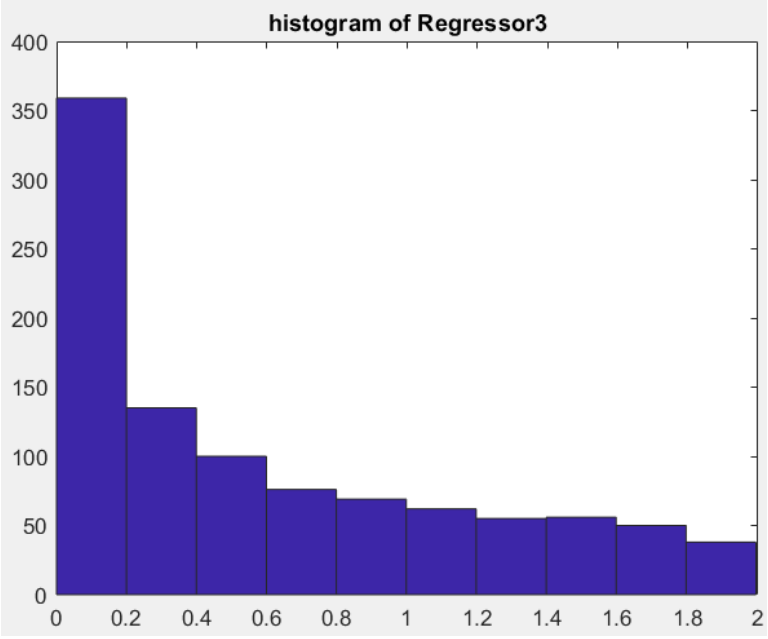
The diagram below is the histogram of the three regressors.



Histogram of the first regressor



Histogram of the second regressor



Histogram of the third regressor

From the histograms, it's seen that the majority of data points are in the range [0, 0.1], so the proper standard deviation will be 0.1, and the variance of 10^{-2} can

affect the data. Here we have noise, which is different than normal noise, so values bigger the before mentioned variance will be chosen.
The variance of output in the output without noise is 0.1643.
For white Gaussian noise we will have:
n1 = wgn(1,1000,-10);
var(noise) 0.0953
mena(noise) = $-3.3*10^{-13}$

The mean is a very small value and pretty close to zero.

$$\frac{\sigma_{y(noise)}}{\sigma_{y(normal)}} = 1.0057$$

The Power of this noise is $-10\ dbw$ or $0.095\ W$.
So we can consider this noise, low.

n2 = wgn(1,1000,1);
var(noise) 1.19
mena(noise) = $-4.1*10^{-17}$

$$\frac{\sigma_{y(noise)}}{\sigma_{y(normal)}} = 1.03$$

Power for this noise is $1\ dbW$ or $1.19\ W$.
Therefore this noise could be considered as medium.

n3 = wgn(1,1000,10);
var(noise) = 10.85
mena(noise) = $-1.1*10^{-19}$

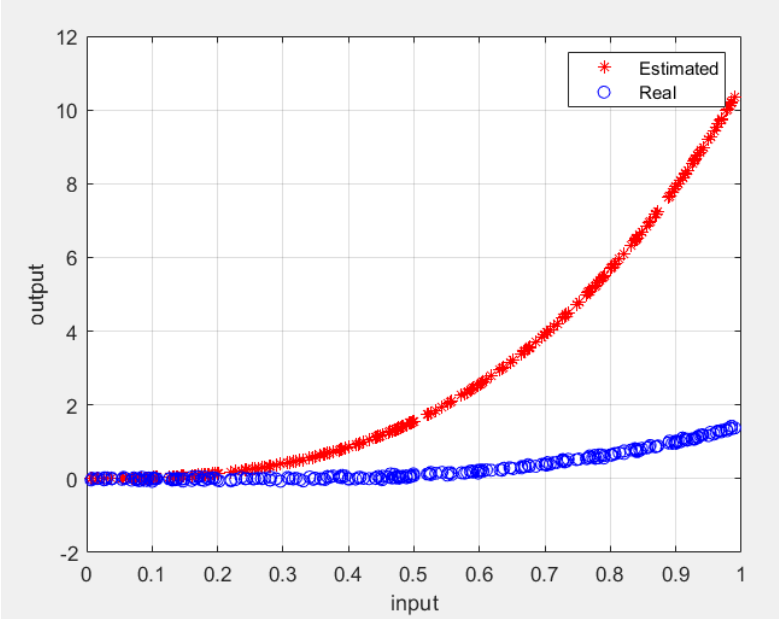
$$\frac{\sigma_{y(noise)}}{\sigma_{y(normal)}} = 1.3$$

Power for this noise is $10\ dbW$ or $10.85\ W$
As a mater of fact, this is considered High noise.

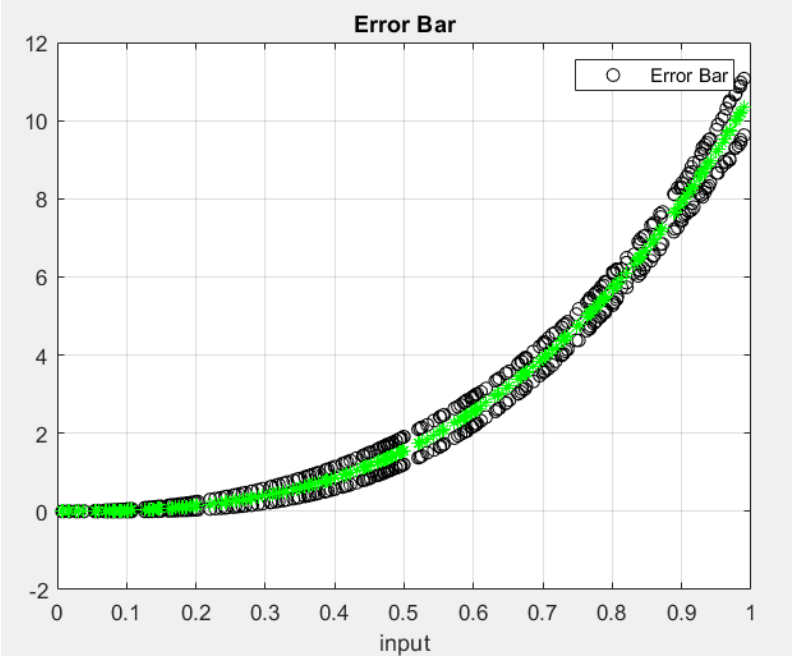
Implementation Question 3.2

Now perform LS estimation:

Low Noise:



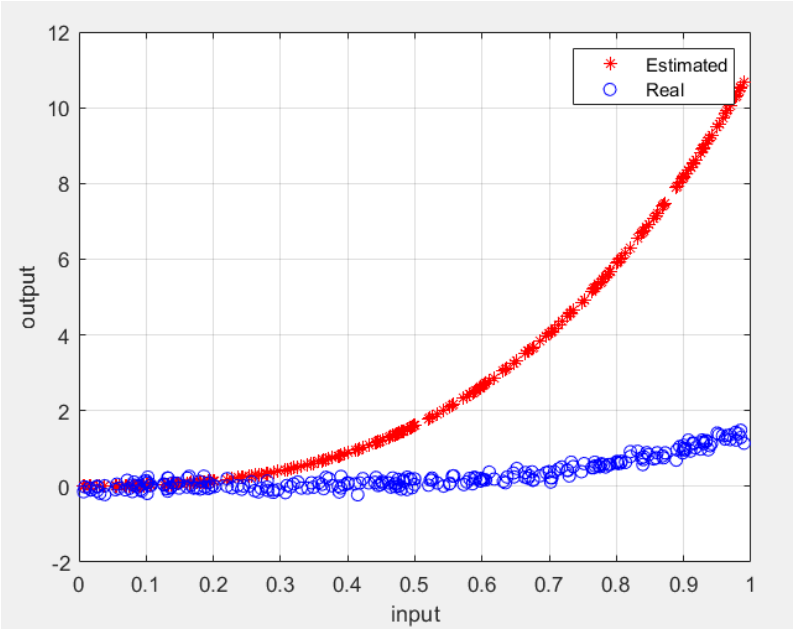
The output and the estimated output Low Noise



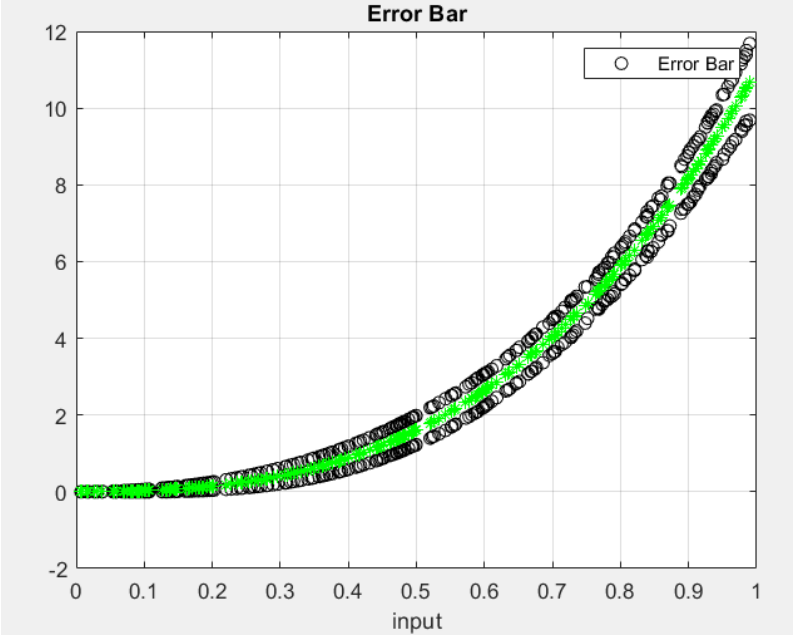
Error Bar Low Nosie

Mean squared normalized error = 14.2327

Medium Nosie:



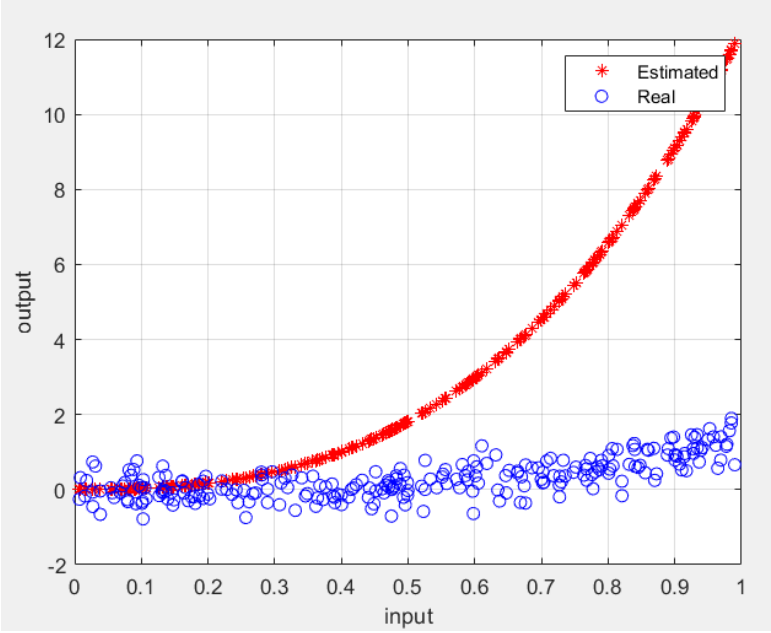
The output and the estimated output Medium Nosie



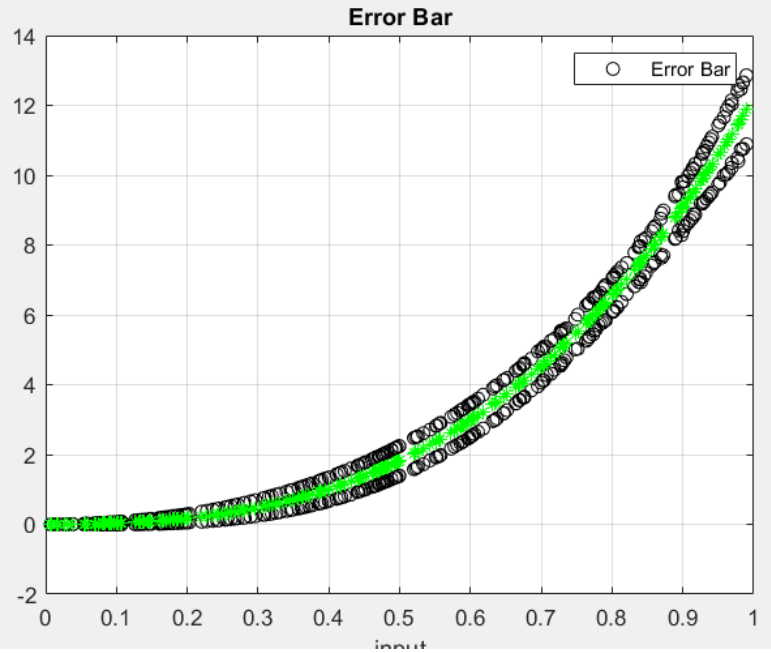
Error Bar Medium Nosie

Mean squared normalized error = 15.3640

High Noise



The output and the estimated output High Nosie



Error Bar High Nosie

Mean squared normalized error = 19.8770

In every case the estimation and the system output are way off! Because the $X^T X$ matrix is **near singular** and norm of this matrix is 6.5423e+15.

Implementation Question 3.3

First Method:

RLS

First we try to use RLS, Regularized Least Square.

Rewrite the cost function as below:

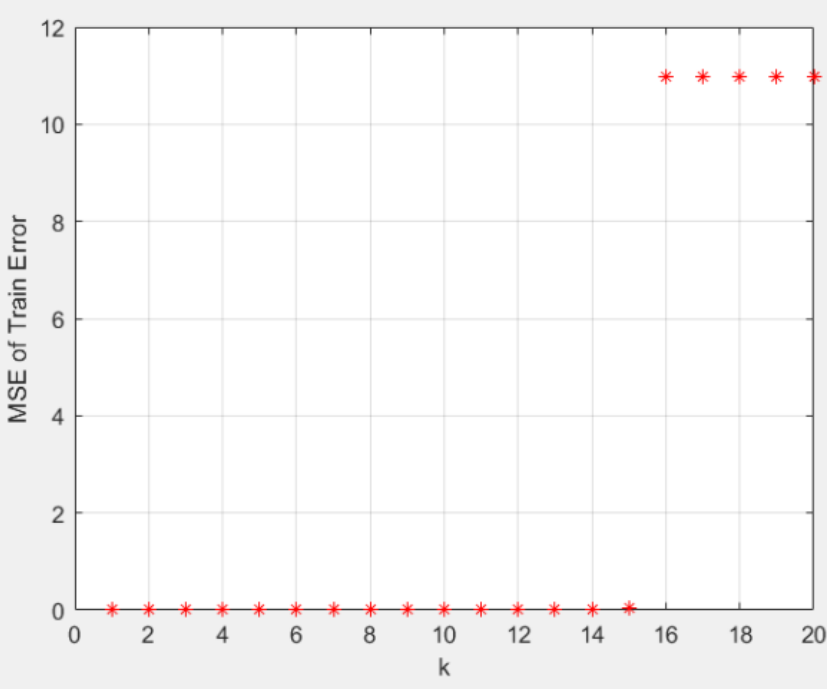
$$J = \sum_{k=1}^{N} e_k^2 + \alpha \sum_{l=1}^{n} \widehat{\theta_l^2}$$

So the estimation is performed via the formula below:

$$\hat{\theta} = (X^T X + \alpha I)^{-1} X^T Y$$

Choosing an appropriate α is an important step in RLS algorithm. If α is chosen a big value the $X^T X$ matrix is not near singular and this will result into the estimated parameters being less scattered from the actual parameters but having a bias. So we need to have a tradeoff between bias and being scattered.

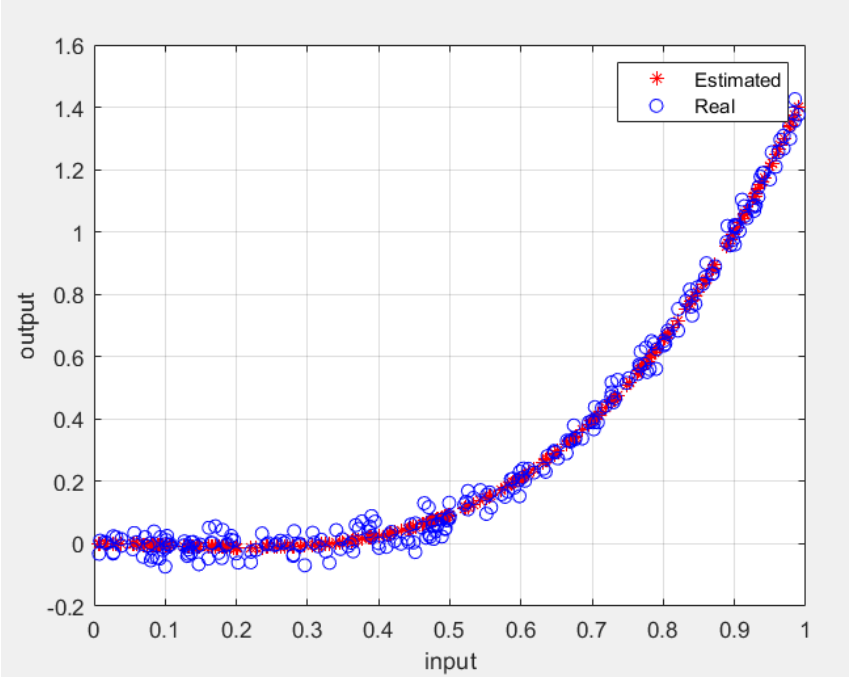
We take α parameter 10^{-k}, and plot the cost function for different values of k to choose an optimal value of k for α



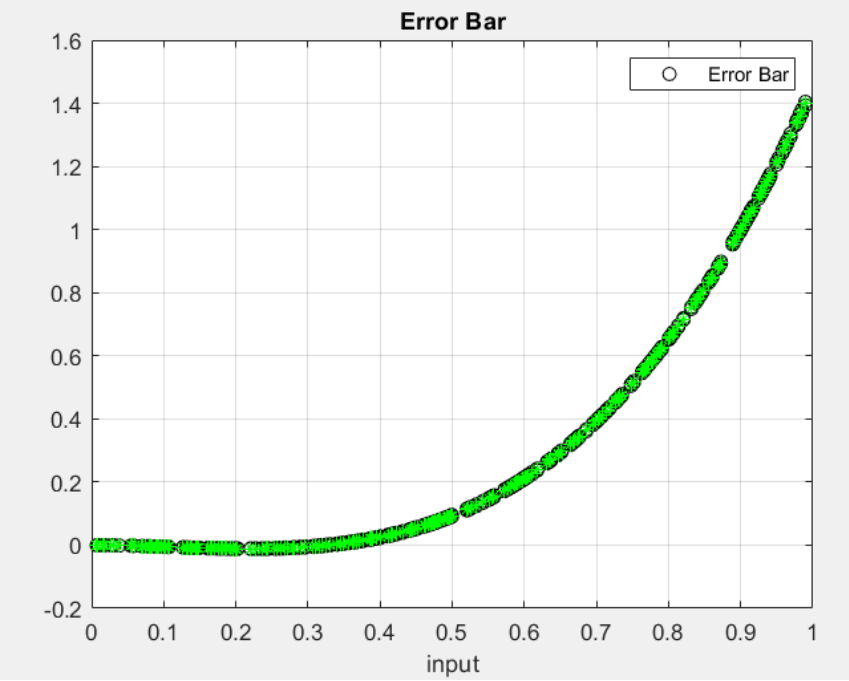
The MSE of Train error based on the values of k RLS method

$k = 13$ is the optimal value, so with α^{-13} we perform the estimation.

First Case (Low Noise):



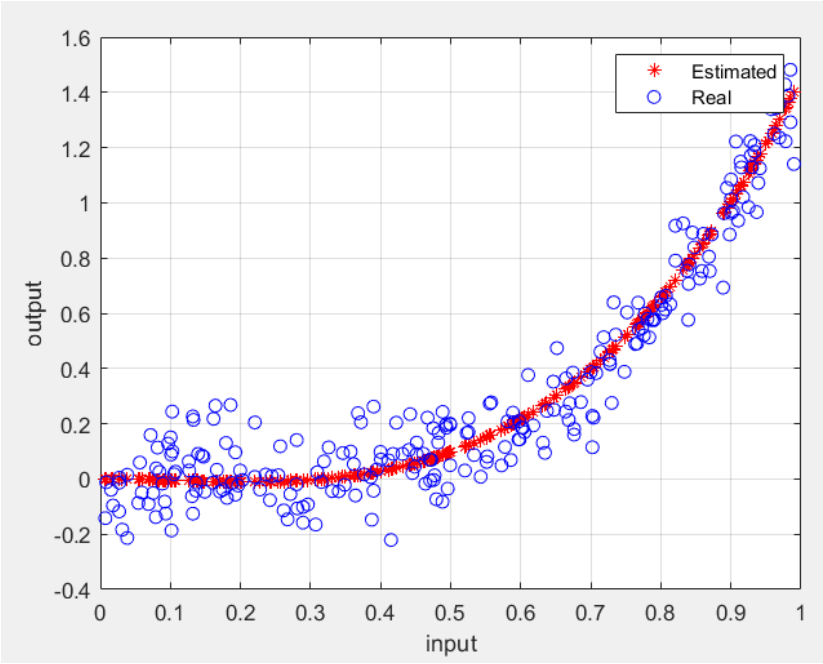
The output and the estimated output RLS Low Nosie



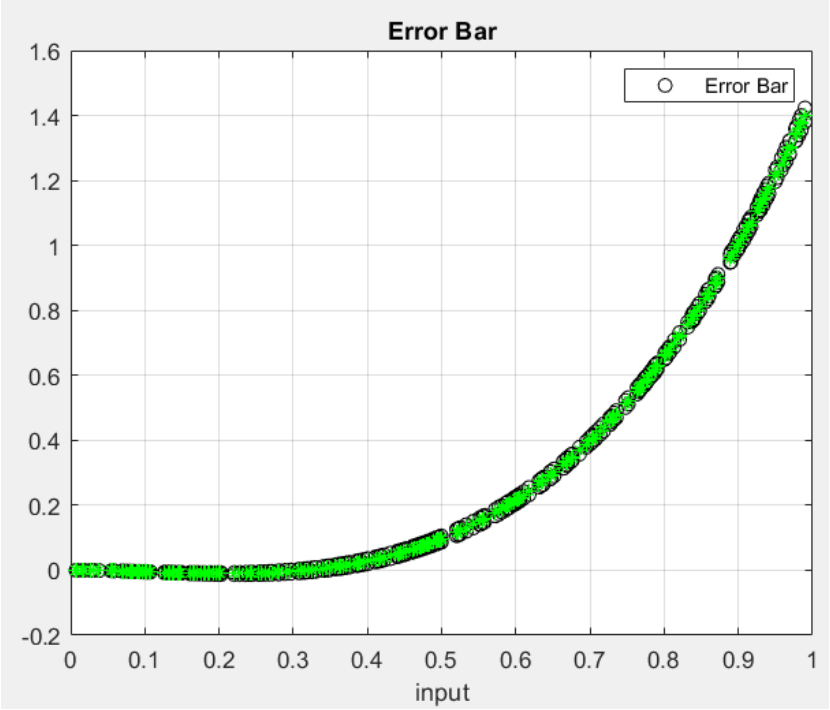
Error Bar RLS Low Noise

Mean squared normalized error = 9.0312e-04

Second case (Medium Noise):



The output and the estimated output RLS Medium Nosie

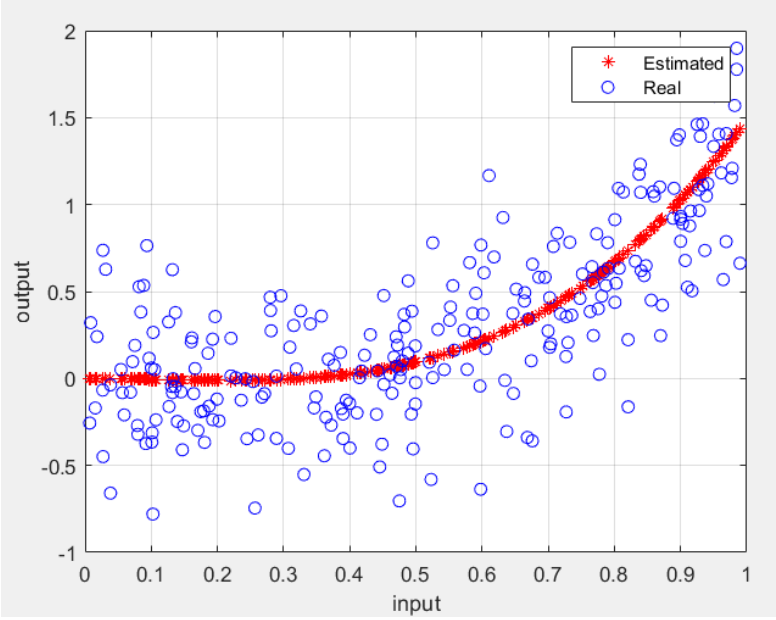


Error Bar RLS Medium Noise

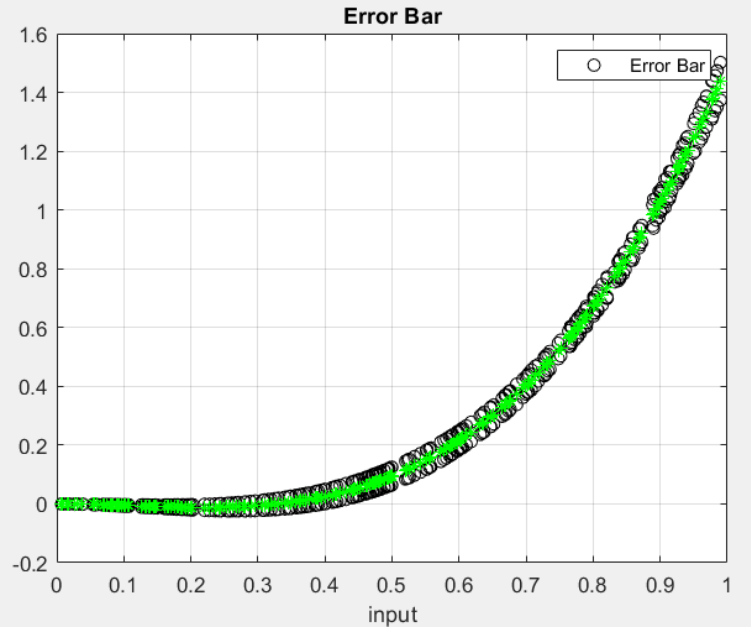
Mean squared normalized error = 0.0113

From the error bar we can say the estimation is performed with a high degree of confidence.

Third Case (High Noise):



The output and the estimated output RLS High Nosie



Error Bar RLS High Noise

Mean squared normalized error = 0.1080

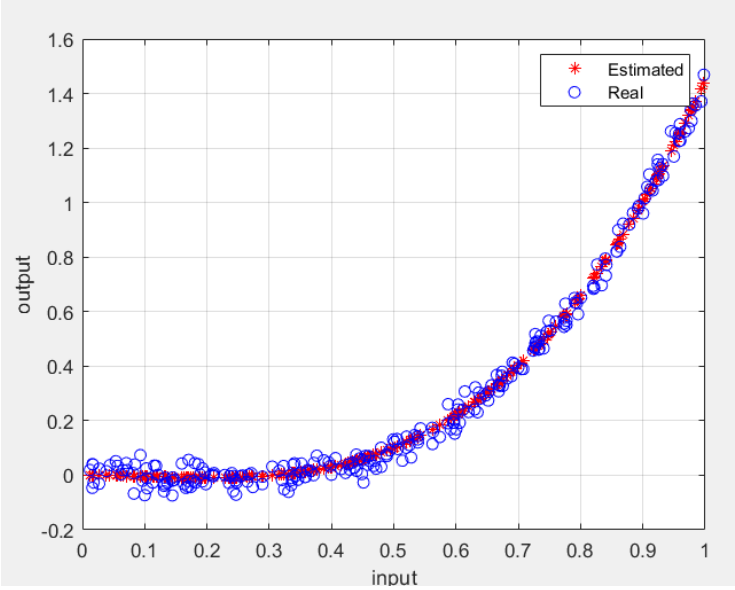
With this amount of noise, I think the algorithm works pretty well.

Second Method

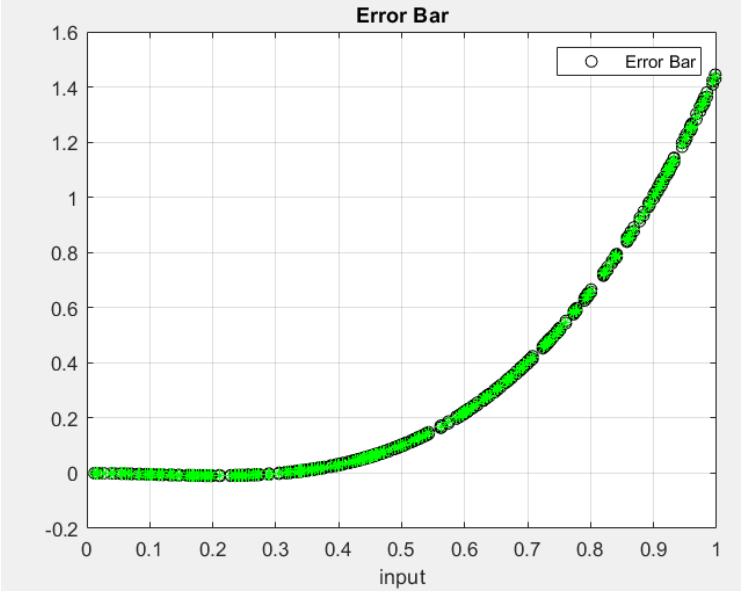
FFRLS

We first need to find a good forgetting factor which suits this problem. The FF is in range [0,1] and usually near 1. With trial and error we'll find the FF(say $\lambda = 0.99$). also with FF the covariance matrix grow big so at first we define a covariance matrix small (let's say 10^3).

Low Nosie

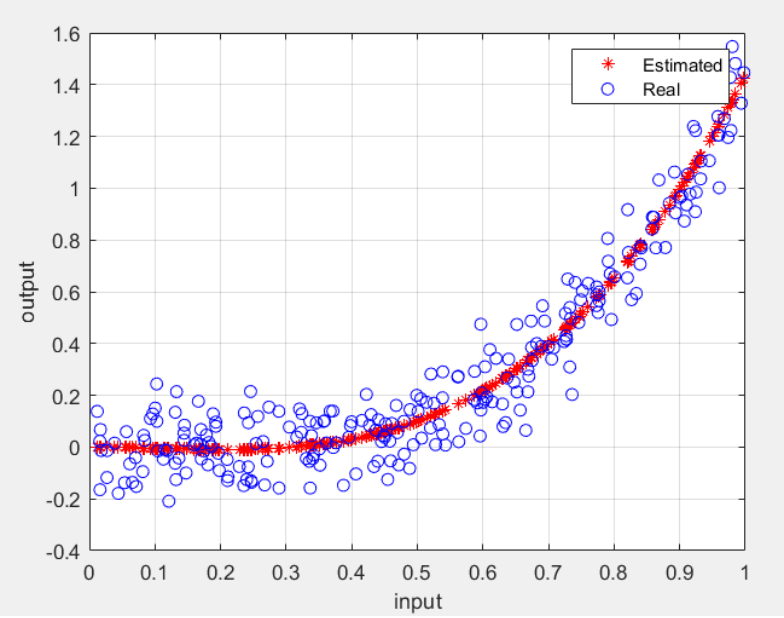


The output and the estimated output FFRLS Low Nosie

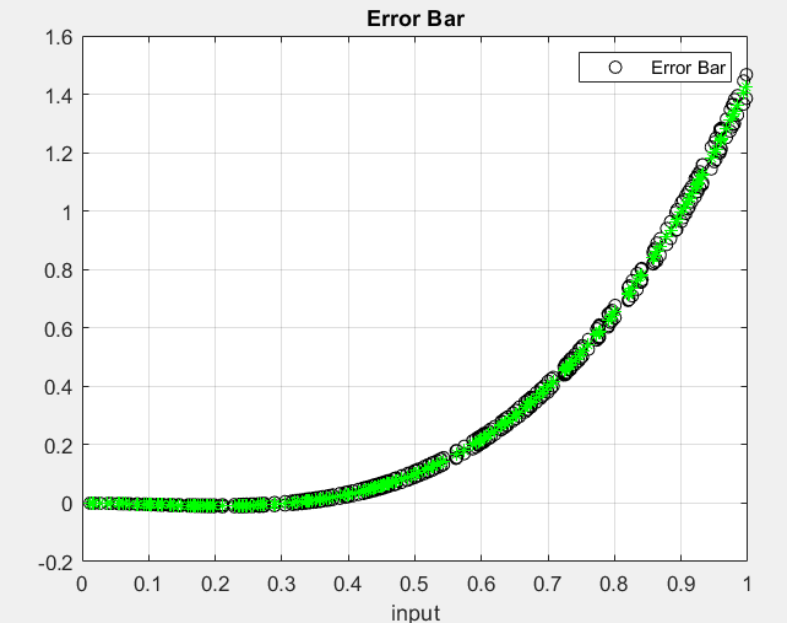


Error Bar FFRLS Low Nosie

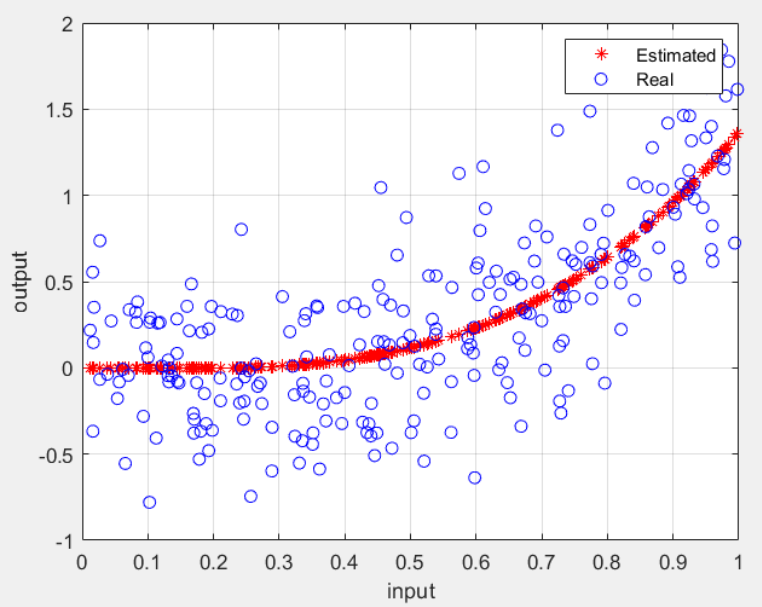
Medium Noise



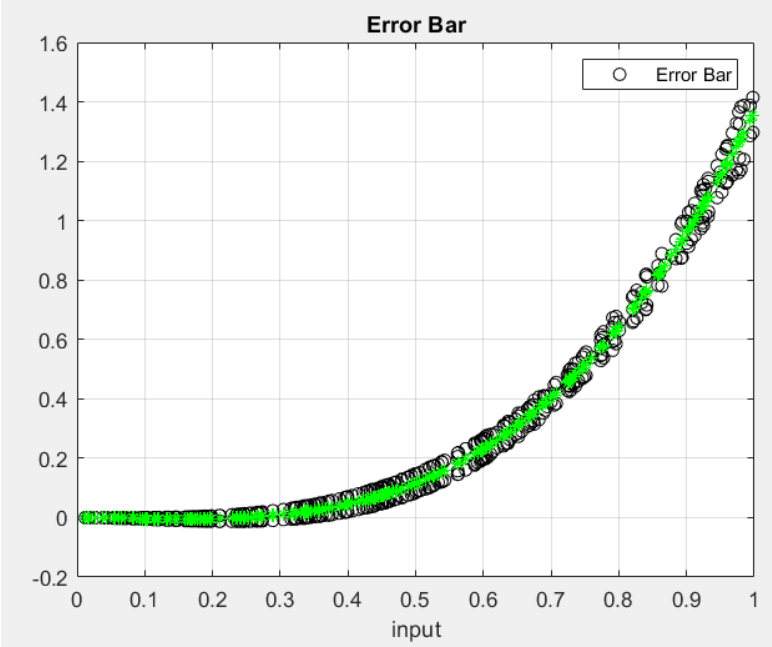
The output and the estimated output FFRLS Medium Nosie



Error Bar FFRLS Medium Nosie



The output and the estimated output FFRLS High Nosie



Error Bar FFRLS High Nosie

Analysis

From the part 2 of this question we can see with the noise the estimation fails to track the actual value of output, because the $X^T X$ matrix becomes near singular. So LS estimation for this specific case fails to estimate.

But as we add the regularization term to the solution, everything changes drastically, and error is reduced by a large amount. But don't forget that we first need to find the optimal α (or regularization parameter) to have an effective estimation.