Parameter Estimation

Control Engineering Laboratory Report.
Experiment No. 1.

Name: Pragyaditya Das.
Roll Number: 110113062.

Attestation:
Date:

Problem Statement:

1. To estimate the parameter of a,
   a. First Order Transfer Function.
   b. First Order Transfer Function, with a given Delay.
   c. Second Order Transfer Function.
2. To find the coefficients of a polynomial Equation, using the given Dataset.
3. To estimate the parameter of a transfer function using the following two method.
   a. Data Driven Model.
   b. Transfer Function Model.

Software Used:

MATLAB R2015a

Theory:

Motivation:

The basis of innumerable simulations and analysis of Dynamic Systems, for our case in particular, Control Systems, are Mathematical Models. Differential Equations play an important role in the development of this models, as they can be easily related to the Physical Principles of the System and their respective transfer function. Before the model is determined, we can regard the System as a "Black Box" with some Process Equations given to us.

There are many methods used for Parameter Estimation. Such as the Least Square Method, Adaptive Control and Moving window method of Parameter Estimation. We will concentrate on the method of Least Squares (LS method). There are multiple advantages of using the LS method. One of the salient advantage is, this method can be used for the parameter estimation of both Dynamic and Static Models.

Mathematics behind Least Squares Method:

The Method of Least Squares is a way of "solving" an over determined system (More equations than unknowns). Because the system is over determined, it is mostly inconsistent.

$$Ax = b \qquad (1)$$

Here A is a rectangular matrix of Dimensions $m \times n$ with *more equations than unknowns* (when $m > n$).

Historically, the method of least square was used by *Gauss and Legendre* to solve problems in astronomy and geodesy. The method was published by *Legendre* in 1805, where he used the method to determine the orbits of comets. However, *Gauss* had already used this method to determine the orbit of Asteroid *Céres*, in 1801. Interesting fact about it is, the method of Gaussian Elimination using Pivots was introduced in this same paper.

The reason why more equations than unknowns arise in such problems is that repeated measurements are taken to minimize errors. This in turn, produces an over determined system of linear equations.

Let us now look at an illustration for more concrete proof of concept.

Suppose, we observe the motion of a small object, assimilated to a point, in the plane.

From our observations, we suspect that this point moves along a straight line, say the equation is

$$y = dx + c$$

Suppose, we observed the moving point at three different locations

$$(x_1, y_1), (x_2, y_2) \; and \; (x_3, y_3).$$

Then we should have the following sets of equations,

$$y_1 = dx_1 + c$$
$$y_2 = dx_2 + c$$
$$y_3 = dx_3 + c$$

If there were no errors in our measurements, these equations would be compatible, and c *and* d would be determined by only two of the equations. However, in the presence of errors, the system may be inconsistent.

The *idea of least square is to find* (c, d) *such that it minimizes the sum of squares of the errors.*

For our illustration, we intend to minimize the following expression,

$$\sum_{i=1}^{3}(c_i + dx_i - y_i).$$

In general, for a system of dimension $m \times n$, written as $Ax = b$, *Gauss and Legendre* discovered that there are solutions x minimizing

$$||Ax - b||^2 \qquad (2)$$

and that these solutions are given by the square $n \times n$ system,

$$A^T Ax = A^T b. \qquad (3)$$

Furthermore, *when the columns of A are linearly independent*, it turns out that $A^T A$ is invertible, so x is unique and given by

$$x = (A^T A)^{-1} A^T b. \qquad (4)$$

We must note that $A^T A$ is a symmetric matrix, one of the so called *normal equations* of a least squares problem.

For instance, the normal equations for the above problem are,

$$\begin{pmatrix} 3 & x_1 + x_2 + x_3 \\ x_1 + x_2 + x_3 & x_1^2 + x_2^2 + x_3^2 \end{pmatrix} = \begin{pmatrix} y_1 + y_2 + y_3 \\ x_1 y_1 + x_2 y_2 + x_3 y_3 \end{pmatrix}.$$

In fact, given any real $m \times n$ matrix A, there is always a unique x^+ of minimum norm that minimizes $||Ax - b||^2$, even when the columns of A are linearly dependent.

It turns out that the minimum norm least squares solution x^+ can be found in terms of the *pseudo-inverse* $A^+$of A, which is itself obtained from a SVD of A.

We define pseudo inverse a follows,

If $A = VDU^T$, with

$$D = diag\,(\lambda_1, \dots, \lambda_{r,} \dots, 0, \dots 0),$$

Where D is an $m \times n$ matrix and $\lambda_i > 0$, letting

$$D^+ = diag\left(\frac{1}{\lambda_1}, \dots, \frac{1}{\lambda_r}, \dots, 0, \dots 0\right),$$

an $n \times m$ matrix, the pseudo-inverse of A is defined as

$$A^+ = UD^+U^T \qquad (5)$$

Application in of Least Square method in Parameter Estimation in Control Engineering:

Suppose,$y^m = \beta_0 + u\beta_1$, is a straight line best fit of a given experimental data with some error, ε. Now, our task is to determine the coefficients of β_0 and β_1 to give the best fit of the model to the data. Our objective is to minimize, ε (minimize the cost function, V).

$$V = \sum_{i=1}^{r} \varepsilon_i^2, r = number\ of\ data.$$
$$\varepsilon_i = y_i - y_i^m$$

Therefore, $$V = \sum_{i=1}^{r} \quad (y_i \ - \beta_0 + u_i\beta_1)^{\ 2}$$

To minimize V w.r.t. β_0 and β_1, set
$$\frac{\partial V}{\partial \beta_0} = 0\ and\ \ \frac{\partial V}{\partial \beta_1} = 0.$$
Solving for $\beta_0\ and\ \beta_1$,
$$\beta_0 = \frac{\sum y_i \sum u_i^2 - \sum u_i \sum u_i y_i}{r \sum u_i^2 - (\sum u_i)^2}$$
$$\beta_1 = \frac{r \sum y_i u_i - \sum y_i \sum u_i}{r \sum u_i^2 - (\sum u_i)^2}$$
Model Form, $y_m = \beta_0 + \beta_1 u_1 + \cdots + \beta_m u_m$. To write this in vector form, we eliminate the observation β_0 from our experimental set.

Thus
$$y_m = (u_1, u_2\ , \dots, u_m)(\beta_1, \beta_2, \dots, \beta_m)^T = u^T\beta$$

If there are r sets of measurements, then,
$$Y_{r\times1} = U_{r\times m}\beta_{m\times1}.$$
For convenience, we shall***, from now on use $\boldsymbol{\varphi}$ in place of $\boldsymbol{u}$ and $\boldsymbol{\theta}$ in place of $\boldsymbol{\beta}$***.
Thus,
$$Y\ = \varphi\theta$$

In an alternative approach, we can also do the following to define Least Square method,

Error, $E = Y - \ \varphi\theta.$

Thus, $E^T = \ (Y - \ \varphi\theta)^T.$

Now,
$$\frac{1}{2}E^TE = \frac{1}{2}(Y - \ \varphi\theta)^T(Y - \ \varphi\theta).$$
$$P = \frac{1}{2}(Y - \ \varphi\theta)^T(Y - \ \varphi\theta).$$
$$P = \frac{1}{2}[Y^TY - Y^T\varphi\theta - \varphi^T\theta^TY - \varphi^T\theta^T\varphi\theta].$$
Here,
$$\theta = (\varphi^T\varphi)^{-1}\varphi^TY. \text{ (From (4)).}$$

<u>Procedure:</u>

1. Construct the block diagrams of the systems, wherever necessary.
2. Add an additional **ToFile block** on the input and output site to capture the data generated by the system when it runs.
3. After saving the data, load it into the workspace.
4. Splice the dataset as intended for the analysis.
5. Save the variables with the intended.

6. Start the **System Identification Toolbox**, using the **ident** command on MATLAB.
7. Import the data into it (Time domain). Take care about the Sampling rate.
8. Estimate it as intended.
9. Use the **MATLAB polyfit** function to fit the given data into a straight line.

Simulation and Results:

Experiment 1:

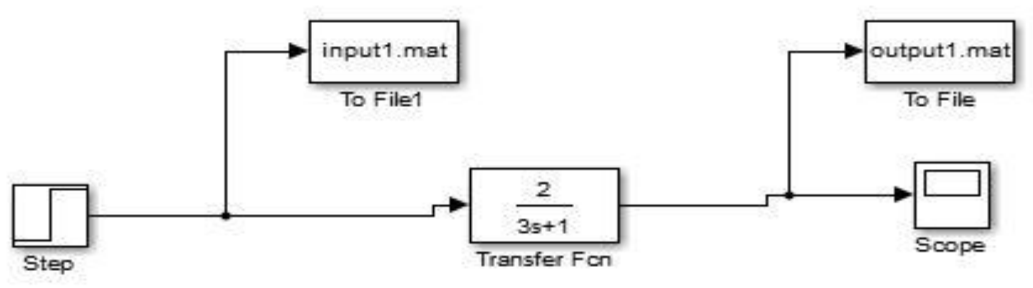
a.

given transfer function,

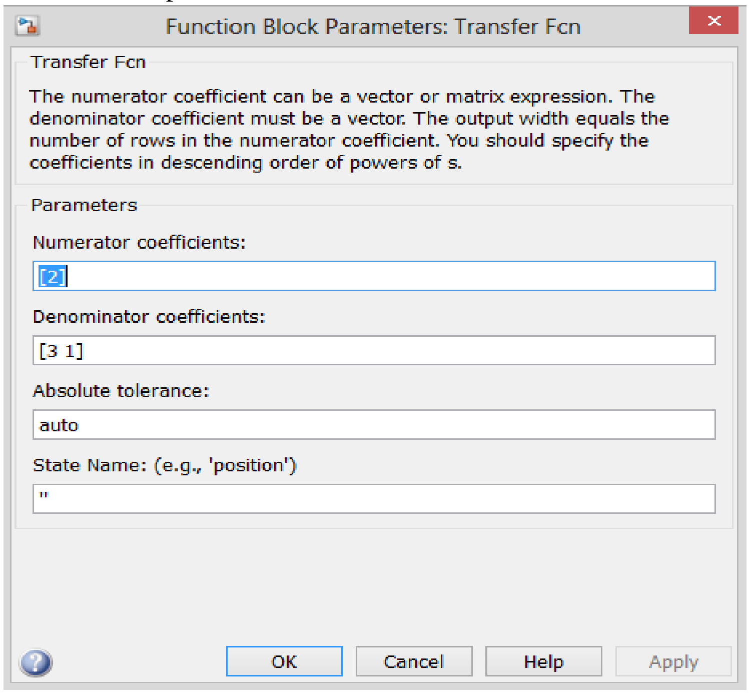
$$\frac{2}{3s+1}$$

this transfer function is a simple first transfer function without any delay.

Now, we construct the block diagram on Simulink.



Here are the specifications of the Transfer Function block,



Function Block Parameters: Transfer Fcn

Transfer Fcn

The numerator coefficient can be a vector or matrix expression. The denominator coefficient must be a vector. The output width equals the number of rows in the numerator coefficient. You should specify the coefficients in descending order of powers of s.

Parameters

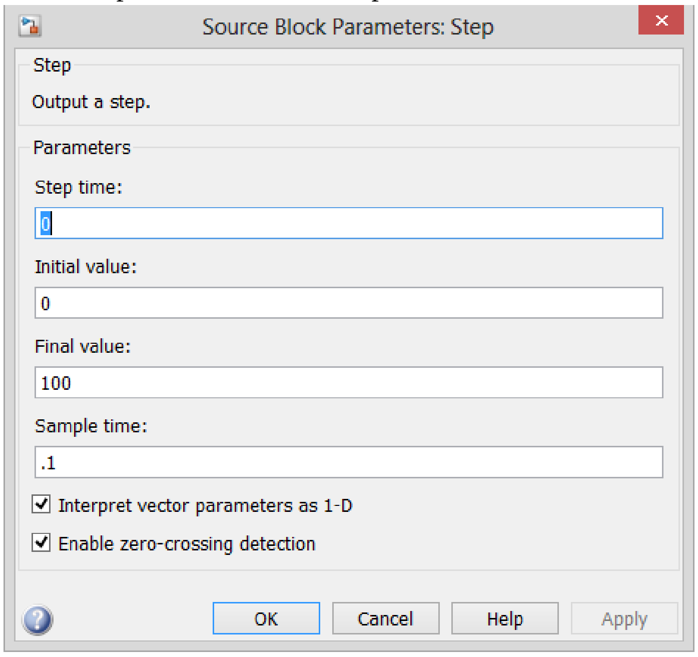
Numerator coefficients: [2]

Denominator coefficients: [3 1]

Absolute tolerance: auto

State Name: (e.g., 'position') ''

And the specifications of the input block is,


Source Block Parameters: Step

Step

Output a step.

Parameters

Step time:

0

Initial value:

0

Final value:

100

Sample time:

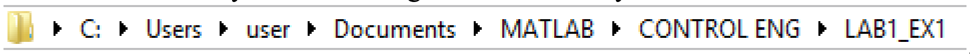
.1

☑ Interpret vector parameters as 1-D

☑ Enable zero-crossing detection

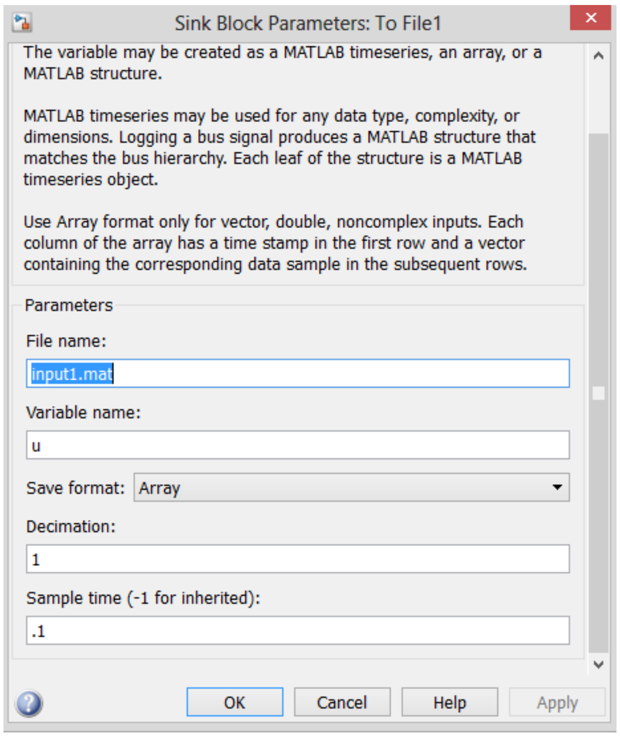
For all the other simulations we do, the settings remain the same.

Now, after the SIMULINK model is set, we set up the ToFile block. For this block to work properly, we need to change the working directory of MATLAB.
There is an error if you don't change it. It is basically an administrative error.


C: ▸ Users ▸ user ▸ Documents ▸ MATLAB ▸ CONTROL ENG ▸ LAB1_EX1

, this is my directory. I suggest you to do the same.

The ToFile block specifications are as follows:



The variable may be created as a MATLAB timeseries, an array, or a MATLAB structure.

MATLAB timeseries may be used for any data type, complexity, or dimensions. Logging a bus signal produces a MATLAB structure that matches the bus hierarchy. Each leaf of the structure is a MATLAB timeseries object.

Use Array format only for vector, double, noncomplex inputs. Each column of the array has a time stamp in the first row and a vector containing the corresponding data sample in the subsequent rows.

Parameters

File name: input1.mat

Variable name: u

Save format: Array

Decimation: 1

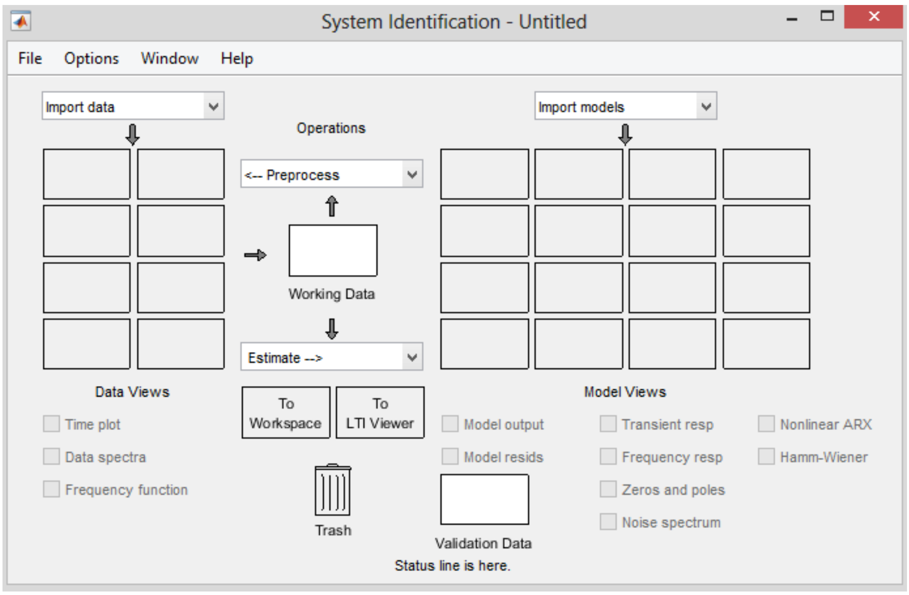
Sample time (-1 for inherited): .1

Now, we run the model. We can see that the input.mat and output.mat files are saved in the specified folder.
We then splice the data files, to get the specified values in column form.
Then we start the System Identification Tool box, by the command ***ident.***
The screen shot is as follows:

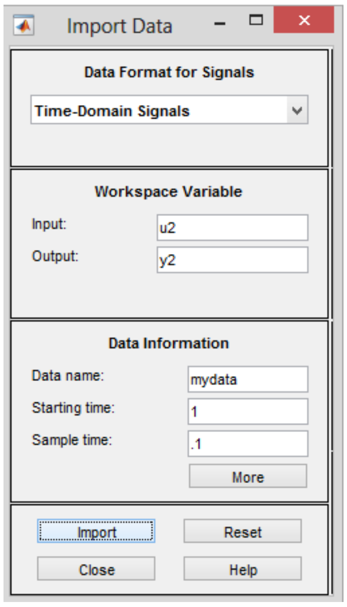
```
>> load('input1.mat')
>> load('output1.mat')
>> u1 = u(2,:);
>> y1 = y(2,:);
>> y2 = y1';
>> u2 = u1';
>> ident
```

[NOTE : This steps will be used for all the three cases of Parameter Estimation.]

After the ***ident*** command is executed in the command window. The System Identification toolbox will start.



Now, we follow the following steps,
Import data > Time Domain > Select the Input (u2) and output (y2) from the workspace > set "Sample Time" as 0.1 >Keep the name as "mydata" > Click on "Import" > In "Estimate", select "Process Model".> Adjust the specifications (Uncheck Delay and Set the no. of poles) > Click on "Estimate" > See the result.


Import Data

Data Format for Signals

Time-Domain Signals

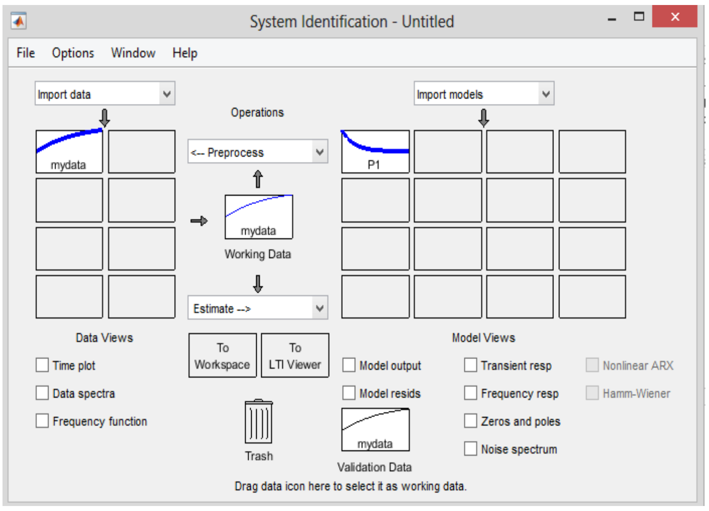
Workspace Variable

Input:	u2
Output:	y2

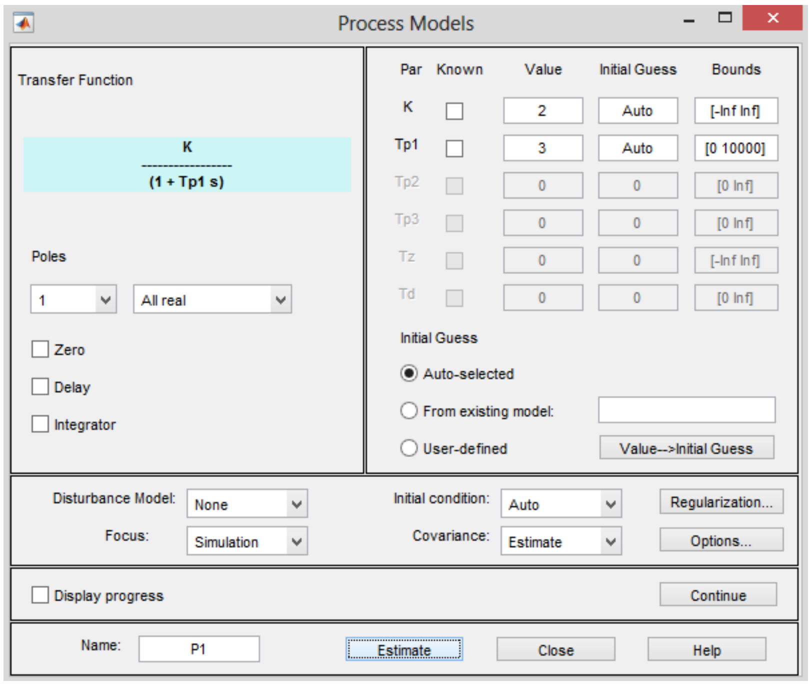
Data Information

Data name:	mydata
Starting time:	1
Sample time:	.1

Import Data.



Take the data for estimation.



Process Models

Transfer Function

$$\frac{K}{(1 + Tp1\ s)}$$

Poles: 1, All real

- [ ] Zero
- [ ] Delay
- [ ] Integrator

Par	Known	Value	Initial Guess	Bounds
K	☐	2	Auto	[-Inf Inf]
Tp1	☐	3	Auto	[0 10000]
Tp2	☐	0	0	[0 Inf]
Tp3	☐	0	0	[0 Inf]
Tz	☐	0	0	[-Inf Inf]
Td	☐	0	0	[0 Inf]

Initial Guess
- ● Auto-selected
- ○ From existing model:
- ○ User-defined

Disturbance Model: None
Focus: Simulation
Initial condition: Auto
Covariance: Estimate

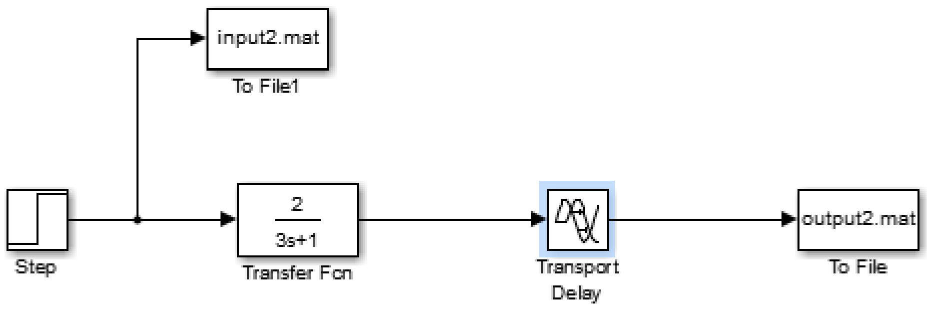
- [ ] Display progress

Name: P1

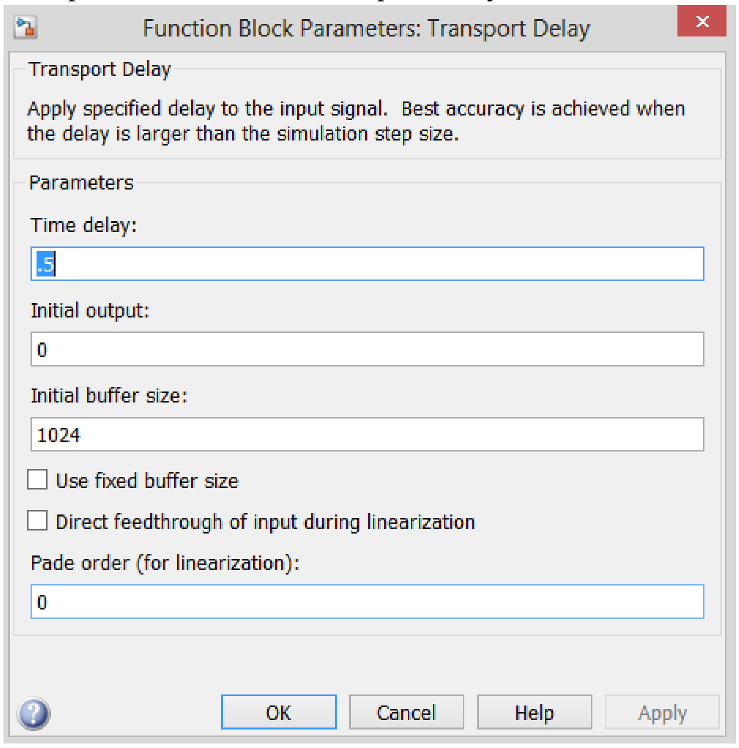
Result.

Now, we have other two questions, they are solved in the similar way, but with very minute modifications.

<u>For experiment 1(b)</u>, we add a delay transport after the transfer function block. The block design is follows:



The specifications for the Transport Delay block is as follows:


Function Block Parameters: Transport Delay

Transport Delay

Apply specified delay to the input signal. Best accuracy is achieved when the delay is larger than the simulation step size.

Parameters

Time delay: .5

Initial output: 0

Initial buffer size: 1024

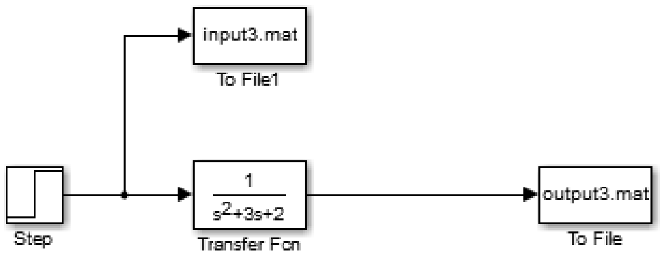
☐ Use fixed buffer size

☐ Direct feedthrough of input during linearization

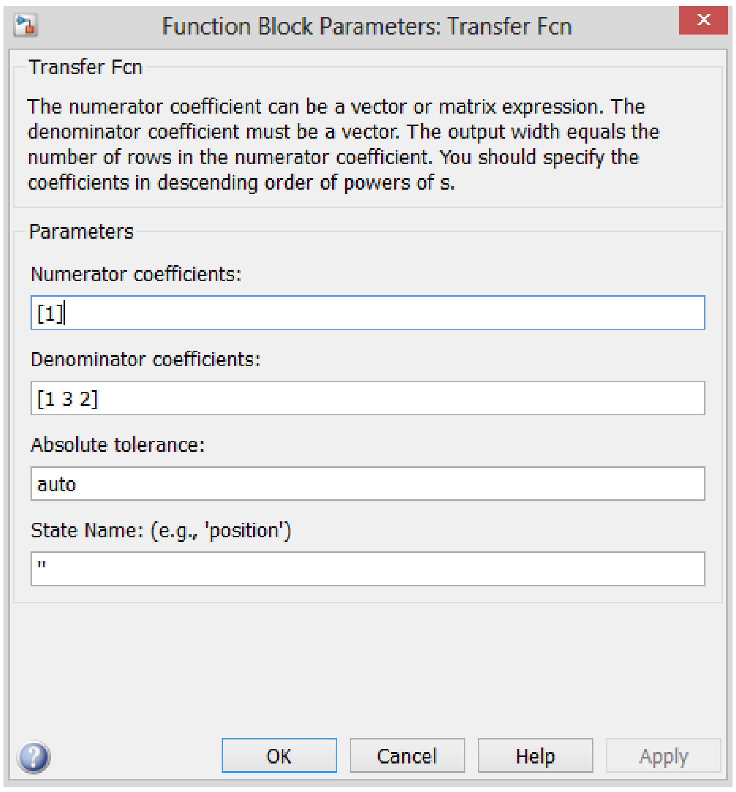
Pade order (for linearization): 0

Now, we follow the same steps as mentioned for Experiment 1(a).

<u>For experiment 1(c)</u>, we add a delay transport after the transfer function block. The block design is follows:



Here, the transfer function block changes. The specifications of it are as follows:



Transfer Fcn

The numerator coefficient can be a vector or matrix expression. The denominator coefficient must be a vector. The output width equals the number of rows in the numerator coefficient. You should specify the coefficients in descending order of powers of s.

Parameters

Numerator coefficients: [1]

Denominator coefficients: [1 3 2]

Absolute tolerance: auto

State Name: (e.g., 'position') ''

Other than this, the entire approach remains the same.

<u>Experiment 2:</u>

k	0	1	2	3	4	5	6
U(k)	1	1	1	1	1	1	1
Y(k)	1	1/2	3/4	5/8	11/16	21/32	43/64

We need to find a and b for this equation,

$$y_k = a y_{k-1} + b u_{k-1}$$

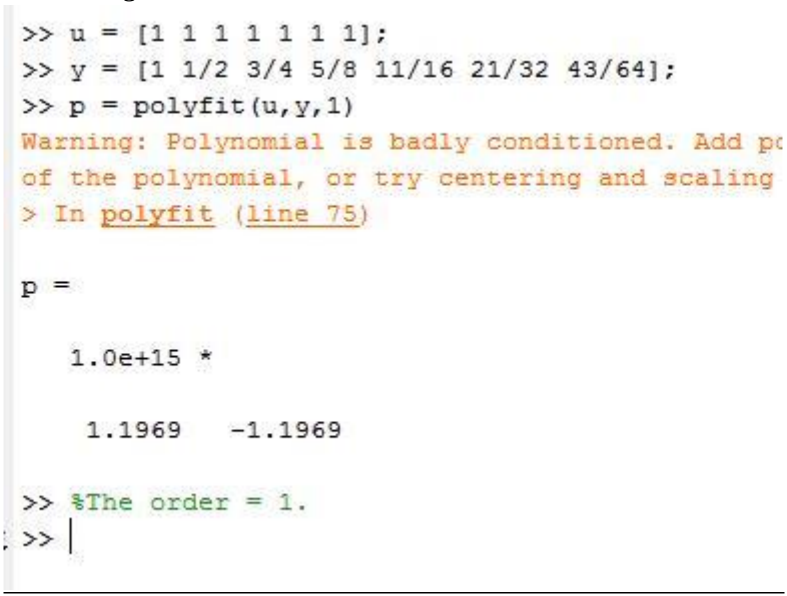
<u>Steps Involved:</u>

1. we load the given U (k) and Y (k) data into MATLAB. We treat these data points as row matrices U and Y.

2.Now, we plot the U vs Y, and we indicate the data points as *.

3. Now, we use a MATLAB function called **Polyfit**, this function is used to find the values of coefficients of a polynomial, so that the polynomial gives a best fit solution.

The coding is as follow:

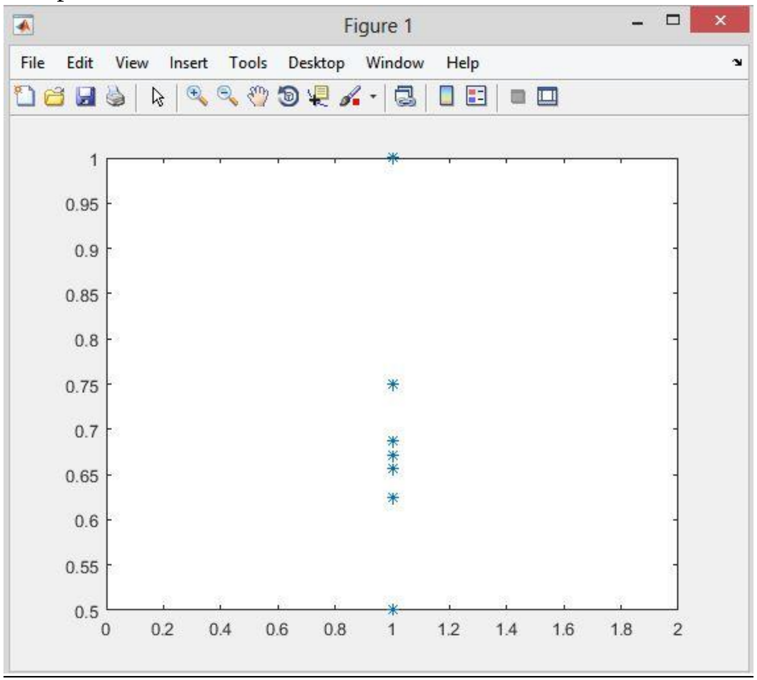


```
>> u = [1 1 1 1 1 1 1];
>> y = [1 1/2 3/4 5/8 11/16 21/32 43/64];
>> p = polyfit(u,y,1)
Warning: Polynomial is badly conditioned. Add po
of the polynomial, or try centering and scaling
> In polyfit (line 75)

p =

   1.0e+15 *

    1.1969   -1.1969

>> %The order = 1.
>> |
```

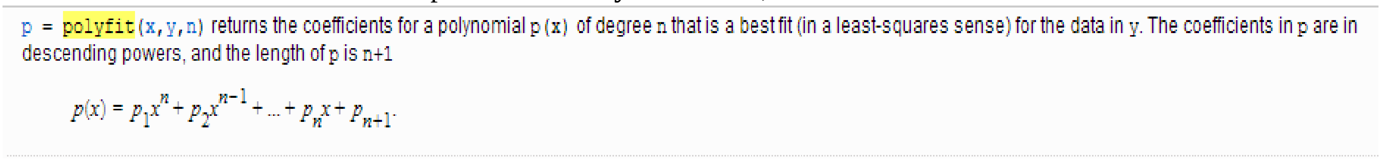
If we do:

`plot(u,y,’*’).`

We will see that the given dataset is very badly conditioned, and hence the large value of a and b.
The plot is as follows,



Here is the official MATLAB description of the **Polyfit** function,



`p = polyfit(x,y,n)` returns the coefficients for a polynomial `p(x)` of degree `n` that is a best fit (in a least-squares sense) for the data in `y`. The coefficients in `p` are in descending powers, and the length of `p` is `n+1`

$$p(x) = p_1x^n + p_2x^{n-1} + \ldots + p_nx + p_{n+1}.$$

Experiment 3:

Given transfer function,

$$\frac{2}{3s+1}$$

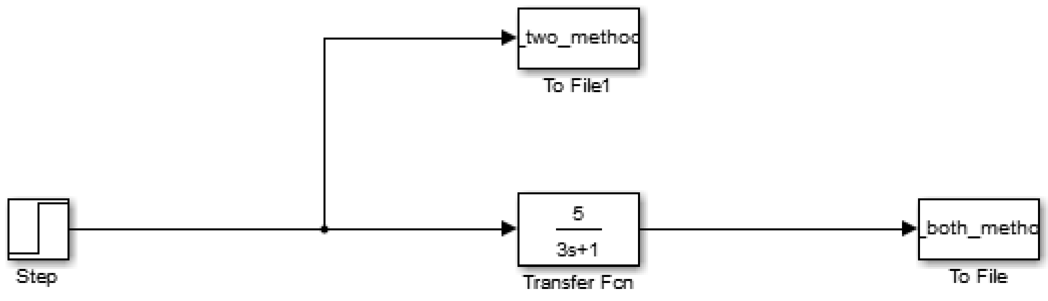
this transfer function is a simple first transfer function without any delay.

We shall solve this problem using two approaches, namely,

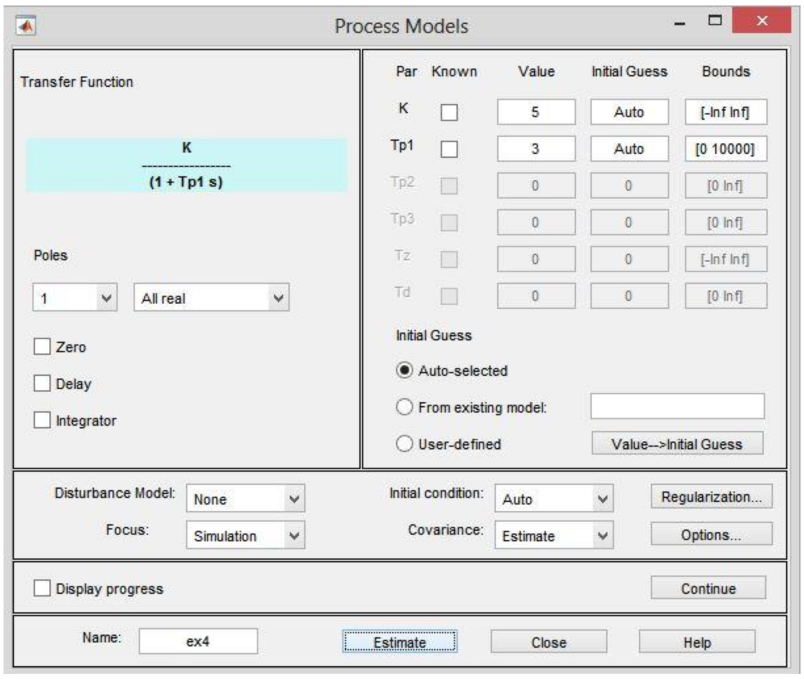
1. Transfer Function Model – Data Driven.
2. Minimizing the Mean Square Error.

For the first method, we follow the given steps,
a. Create the SIMULINK model.
b. Store the data in input.mat and output.mat files.
c. Analyse the stored files, splice the data (as done in Experiment 1(a)).
d. Start the System Identification Toolbox, import the data and estimate the Transfer function model.

The SIMULINK model and the output of the System Identification toolbox is as follows,



SIMULINK model.


Process Models

Transfer Function

K / (1 + Tp1 s)

Poles: 1, All real

Zero [ ], Delay [ ], Integrator [ ]

Par	Known	Value	Initial Guess	Bounds
K	☐	5	Auto	[-Inf Inf]
Tp1	☐	3	Auto	[0 10000]
Tp2	☐	0	0	[0 Inf]
Tp3	☐	0	0	[0 Inf]
Tz	☐	0	0	[-Inf Inf]
Td	☐	0	0	[0 Inf]

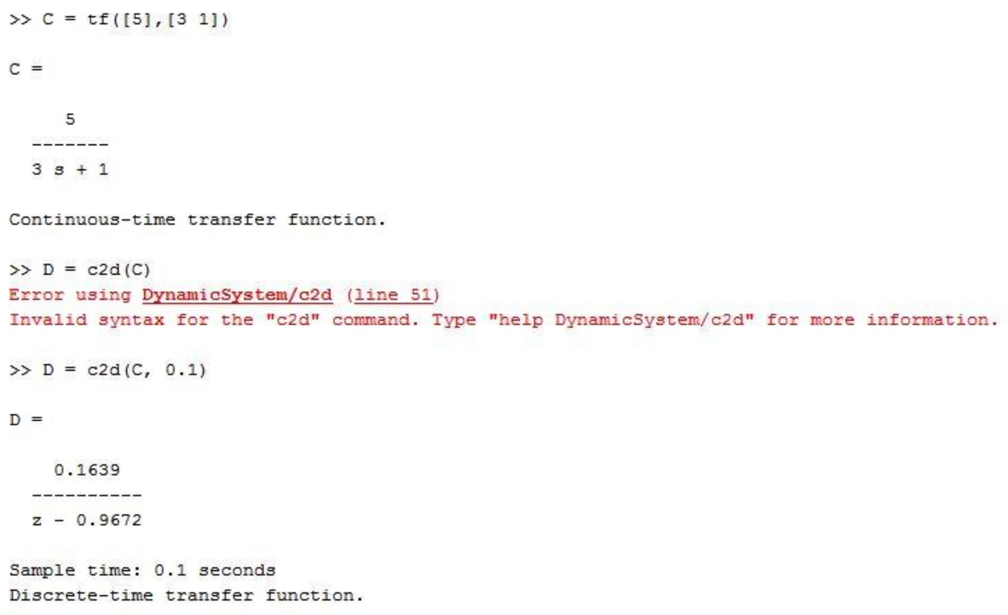
Initial Guess: [✓ Auto-selected] From existing model: User-defined

Disturbance Model: None
Focus: Simulation
Initial condition: Auto
Covariance: Estimate

Display progress [ ]

Name: ex4

System Identification Toolbox result.



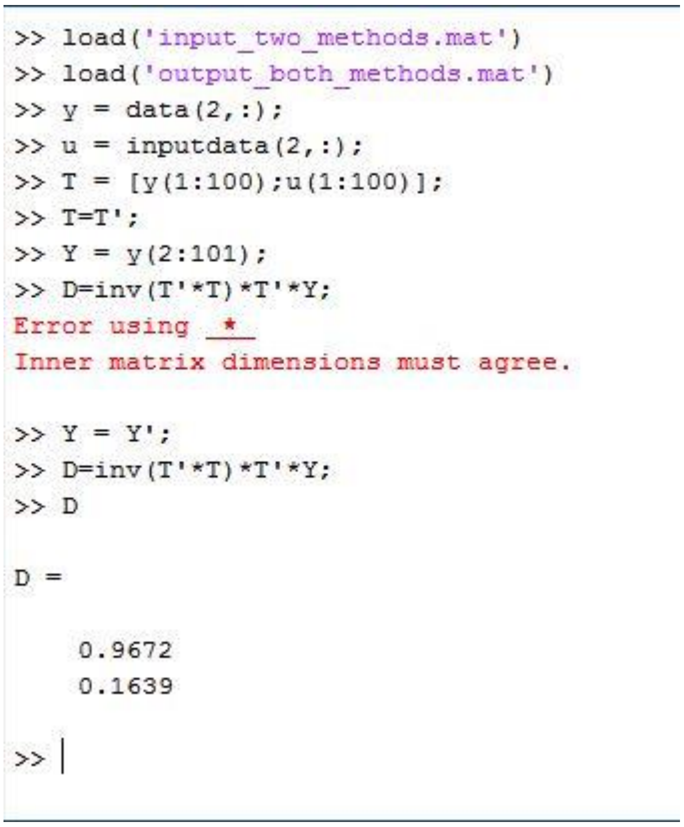
```
>> C = tf([5],[3 1])

C =

      5
  -------
  3 s + 1

Continuous-time transfer function.

>> D = c2d(C)
Error using DynamicSystem/c2d (line 51)
Invalid syntax for the "c2d" command. Type "help DynamicSystem/c2d" for more information.

>> D = c2d(C, 0.1)

D =

    0.1639
  ----------
  z - 0.9672

Sample time: 0.1 seconds
Discrete-time transfer function.
```

(In this step, we convert the transfer function from s-domain to z-domain).

Now, we go into the second method. In this method we use pure Mathematics to find the model. These are the following steps:

1. Load the data generated from the SIMULINK model into the Workspace of MATLAB.
2. Splice each dataset to get the output row matrix.
3. Create a matrix, with both the rows of input and output data.
4. Get the Transpose of the Matrix.
5. Now, we compute the pseudo-inverse of the matrix (equation (5)).
6. Now multiply it with the output matrix, this will generate the output matrix in 2X1 form.
7. These are the parameters of the transfer function.

The Maths used is as follows:



```
>> load('input_two_methods.mat')
>> load('output_both_methods.mat')
>> y = data(2,:);
>> u = inputdata(2,:);
>> T = [y(1:100);u(1:100)];
>> T=T';
>> Y = y(2:101);
>> D=inv(T'*T)*T'*Y;
Error using  *
Inner matrix dimensions must agree.

>> Y = Y';
>> D=inv(T'*T)*T'*Y;
>> D

D =

    0.9672
    0.1639

>> |
```

We see that the parameters are estimated with a very high extent of accuracy.
Thus, the experiment is successful.

Conclusion:

Thus, in conclusion we have used the System Identification toolbox of MATLAB to estimate the parameter of transfer functions of various types.
We have also the best fit coefficients of a polynomial using **Polyfit** function and inferred that the given dataset is very badly conditioned.
We estimated the Parameters of a given transfer function using the Transfer Function-Data Driven model and the Minimum Mean Square Error Model. We inferred that both the methods given almost the same result (4 decimal points).