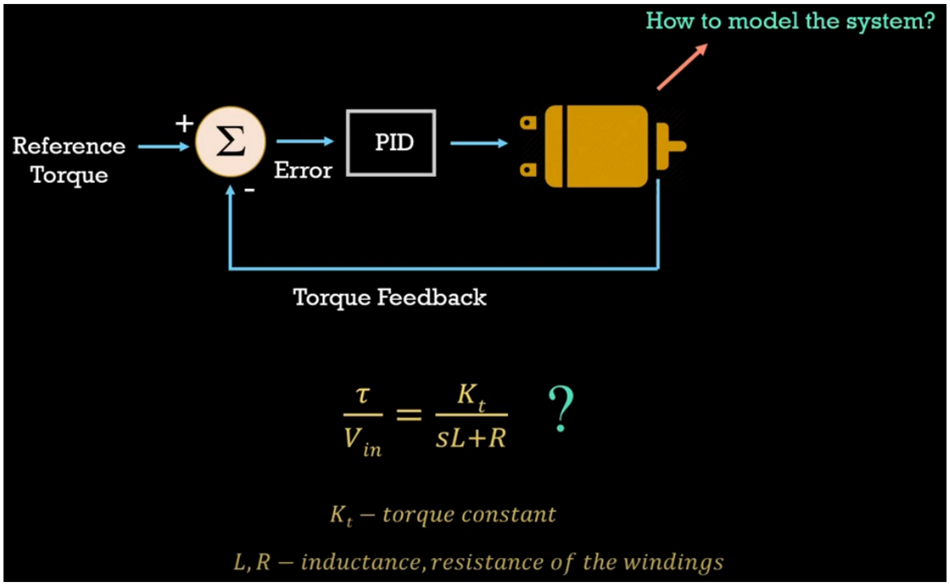
Part4 How to model a System?

Saturday, July 13, 2024 4:44 PM



in this video we are going to see different techniques to model a system

for example we saw in the previous video that for a dc motor the dot 2 voltage transfer function is kt upon sl plus r where kt is a torque constant l is the inductance and r is the resistance of the motor

but how do i find this transfer function let us find out now before you design any controller you should definitely know how your system behaves or what is the model of the system

but as a control systems engineer you might come across different hardware and different situations that are not very straightforward

one of the possibilities is you already know how the system works

for example you know how this dc motor works
in other words you know what are the underlying equations that govern the system

in this case you can write down one equation for the electrical quantities one equation for the mechanical quantities and a third equation which links the current to the torque now you also know the values of r l j etc

this is the best possible scenario you just substitute these values and find the transfer function by taking the laplace transform

i have written down the torque to voltage and speed to voltage transfer function and there you have the model of your system drop done

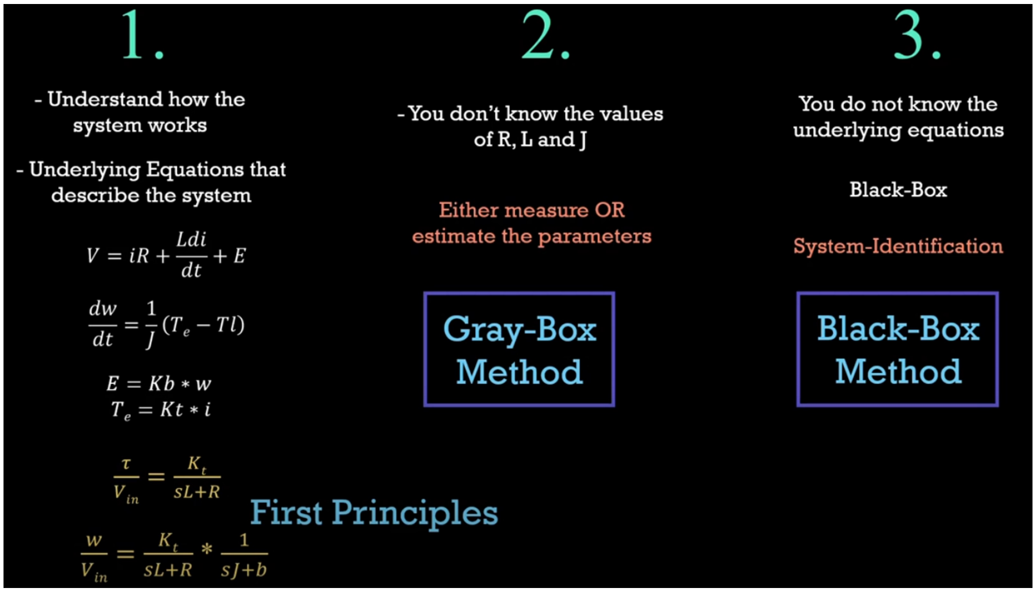
you can tune the controller now but it is not always that easy

you might know how the system works that is you know what are the underlying equations but you do not know the values of r l and j

in this case either you will have to measure the parameters or you will have to estimate them but there is one more case where you do not know anything about the system

it is a black box studio in this case you will have to do the task of system identification
these three scenarios have names to them:
-- the first one is called modeling from first principles
-- the second one is called gray box modelling and
-- the third one is the black box method in this video we'll see both the gray box and black box methods

we'll start with the gray



Gray-box method

box method you know something about the system but not everything in the dc motor case you do not know the values of r l and j we will first see how to measure these values

i can give a small voltage to the dc motor and check the final steady state current value the resistance of the motor is nothing but v by i and in this case it is 2 volt by 1 amp which is equal to 2 ohms we have found the resistance

now we know the time constant of the current waveform is given by the value l by r so we can check how fast the current waveform is rising we know in one time constant the current would rise to 63.2 percent of the final value we can just check out how much time it has taken and then equate it to l by r and find the inductance

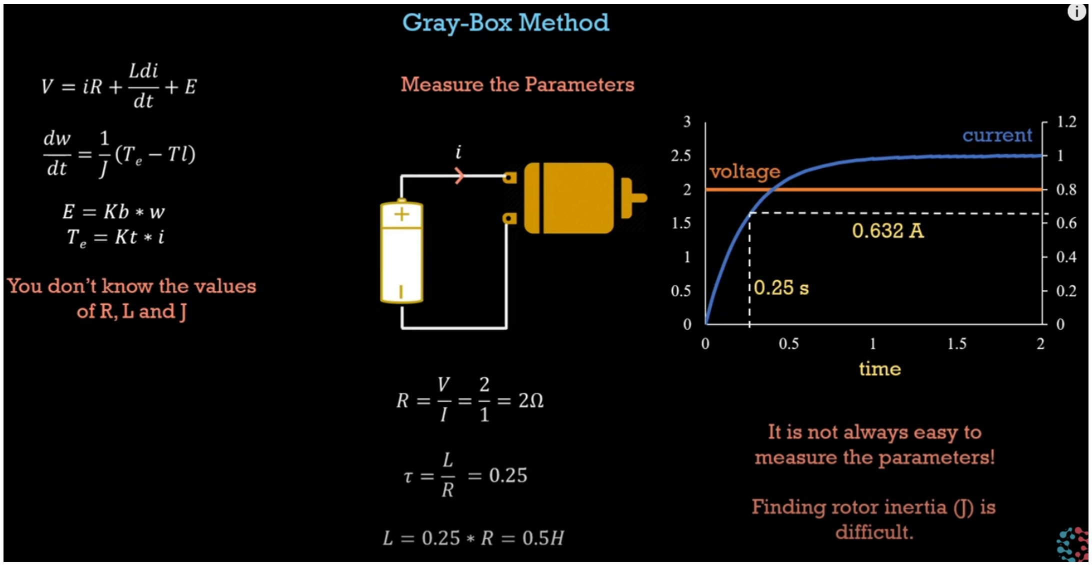
it's easy right yes it was easy in this case but this technique will only apply to the dc motor case. if your system is different, for example, you are tuning a heat exchanger control system then you have to come up with your own measurement techniques

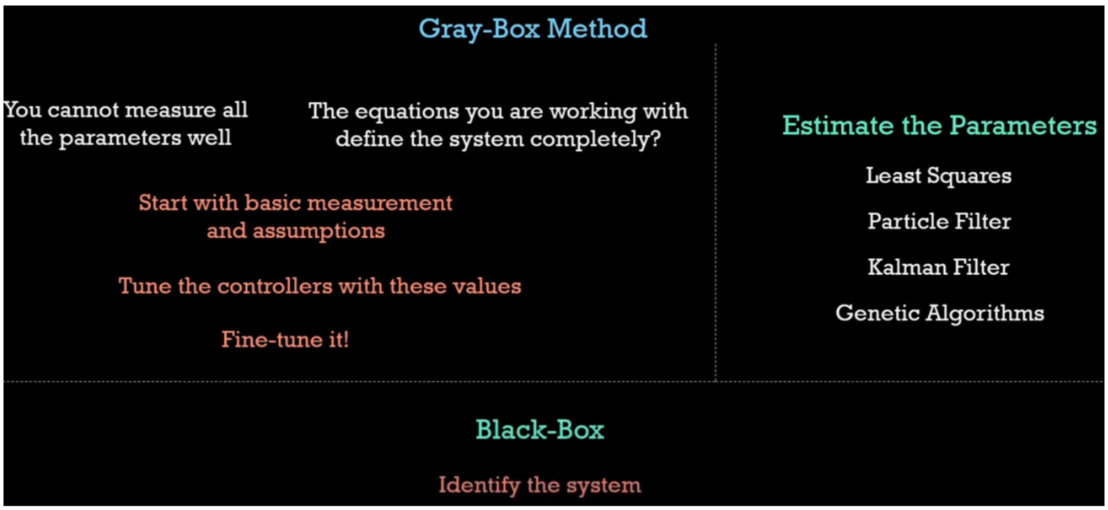
and it is not always easy to measure the parameters for example you still have to find the inertia of the motor it is not straightforward to do that and it also depends on what is attached to the shaft of the motor this is the main drawback of the grey box modeling technique

first, you cannot always measure all the parameters second, you're not even sure whether your equation have included all the intricacies of the system

so what do you do you can move ahead with the basic calculation of these parameters tune the controller based on these values and then fine-tune it later

we'll see how to do the fine tuning in the next video but we have another option if you cannot measure the parameters you estimate them now there are a lot of ways to do this i will attach a link in the description where you can check out how to do parameter estimation in matlab and finally just treat it as a black box and let us identify the system from scratch





Black-Box method (System Identification)

this will bring us to the black box method or the system identification method

you do not know anything about the system so how do you identify it well we will use the matlab system identification toolbox to do it

the idea is you give some input voltage to the dc motor and you measure the speed of the motor you know the input you know the output and then you try to guess what the system is

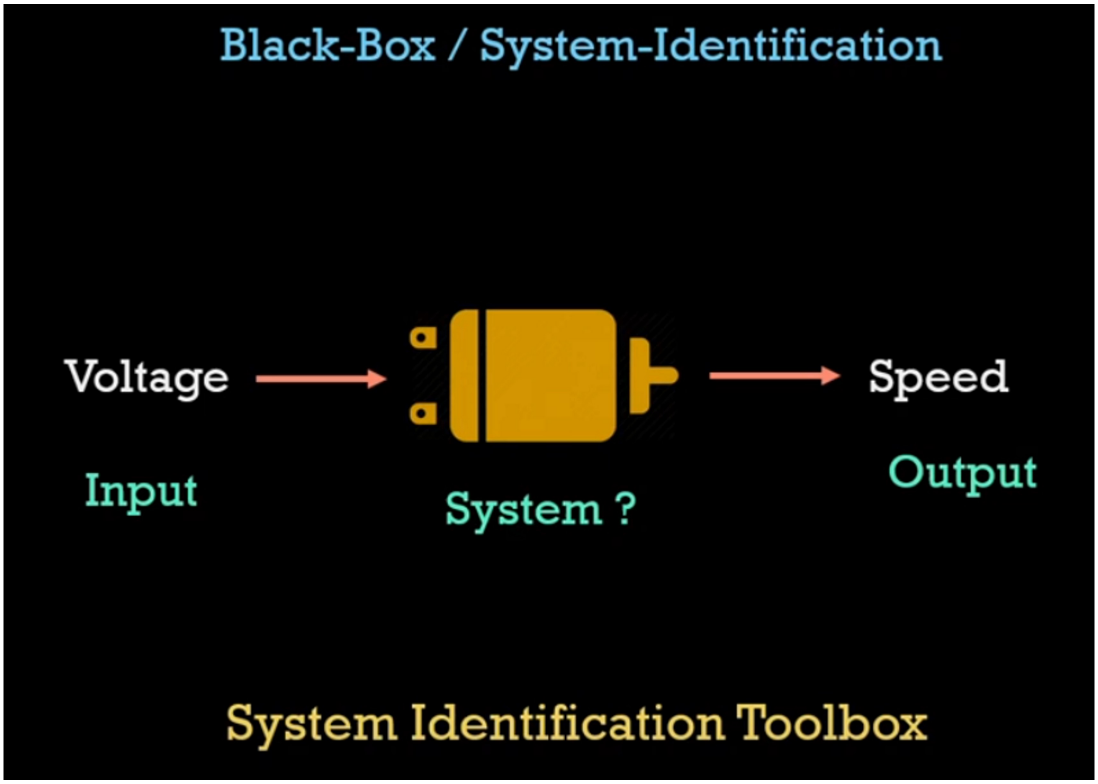
matlab has done a good job of guessing the system in the system identification toolbox instead of using matlab you can also check out some system identification algorithms

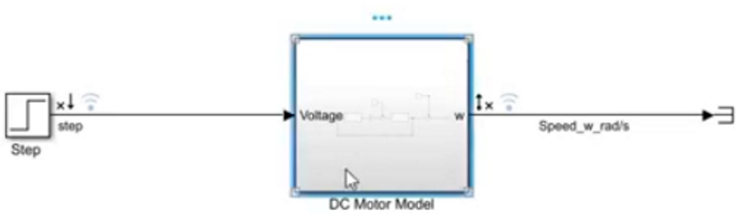
if you want to do it in real time but that is not the subject of this video let us go to simulink now

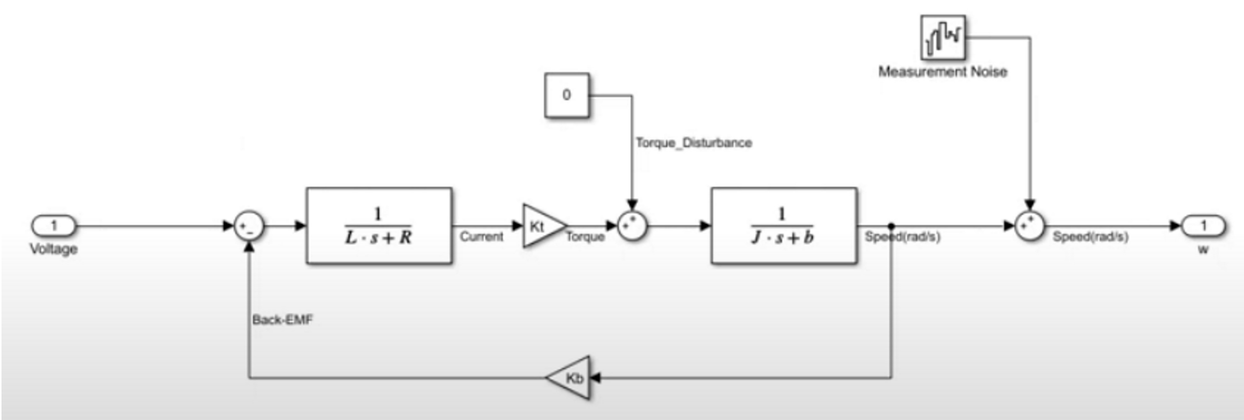
here i have modeled the dc motor using the same equations that we saw earlier ideally i would like to do system identification on a real motor but since i do not have one right now let us assume this model is the real motor

we will give a step input to it and we will check the output let us run the simulation and check out the signals in the data inspector we can see how the speed rises to a one volt step input

we have also added measurement noise to the speed as this is how the sensors generally behave considering that these were the input and the outputs that you would get from a real system







we will use them in the system identification tool first we will log these values into the workspace and then open the system identification toolbox the next step is to import the time domain data that we logged now you can also filter the
noise using preprocessor

but we will not do that now let's see how the system identification works with noisy data

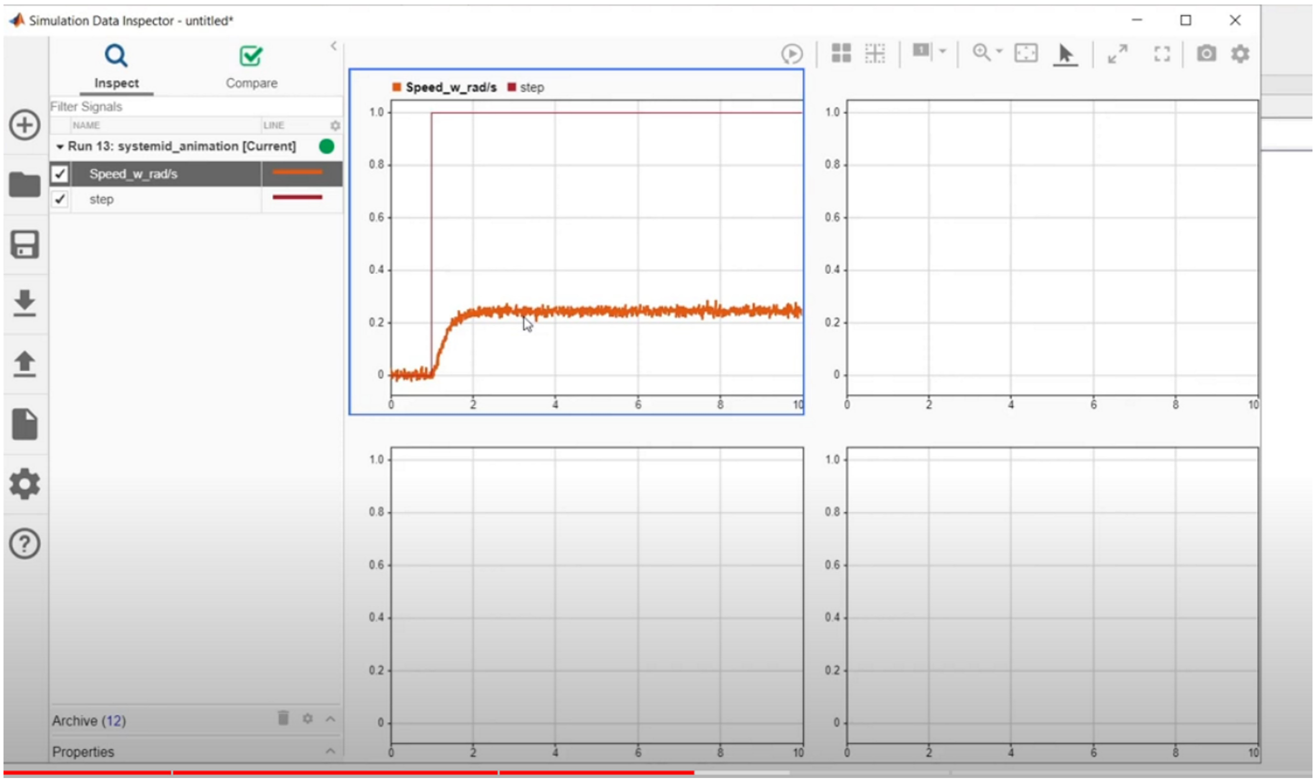
we will set the number of poles as 2 and the number of zeros are zero this is where it's helpful to have some knowledge of the system rather than assuming it as a complete black box

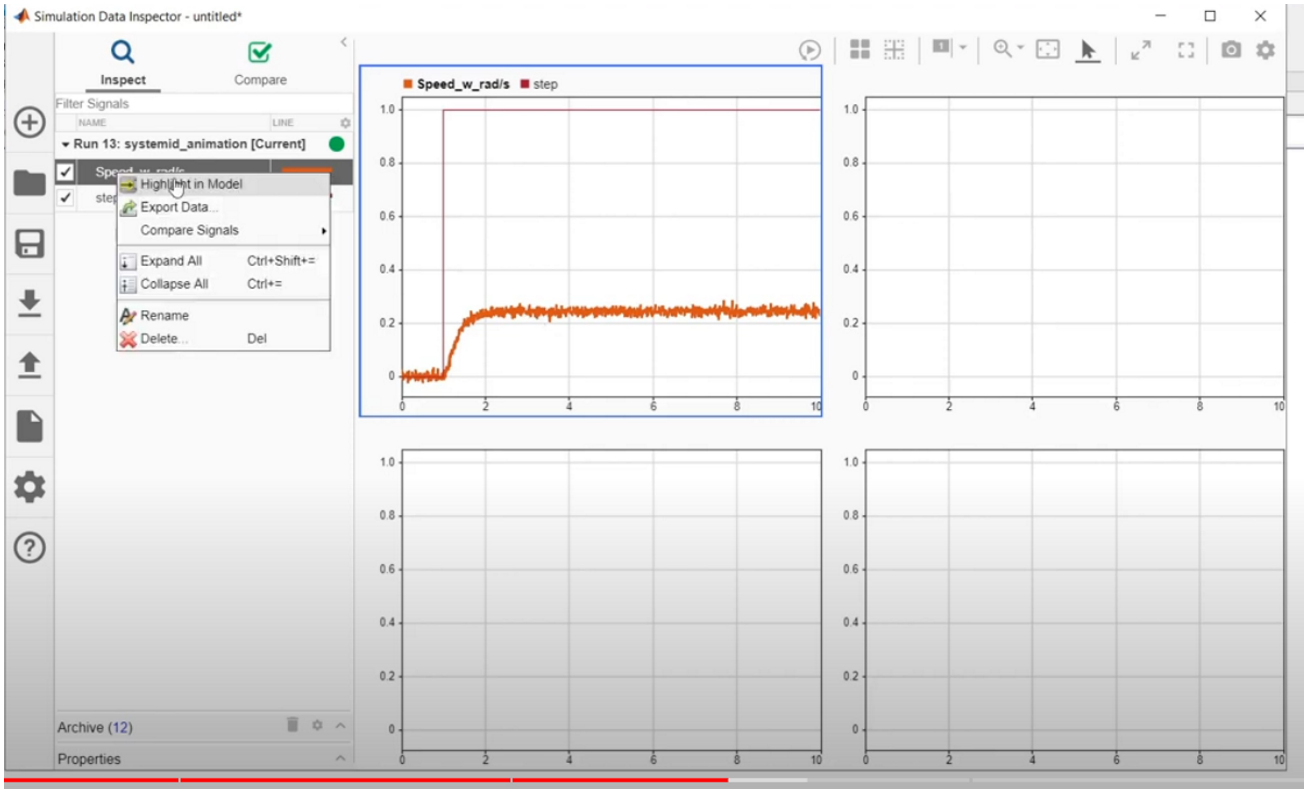
if you did not know anything about the system then you have to try out different combinations of poles and zeroes and see which model resembles closest to the original system

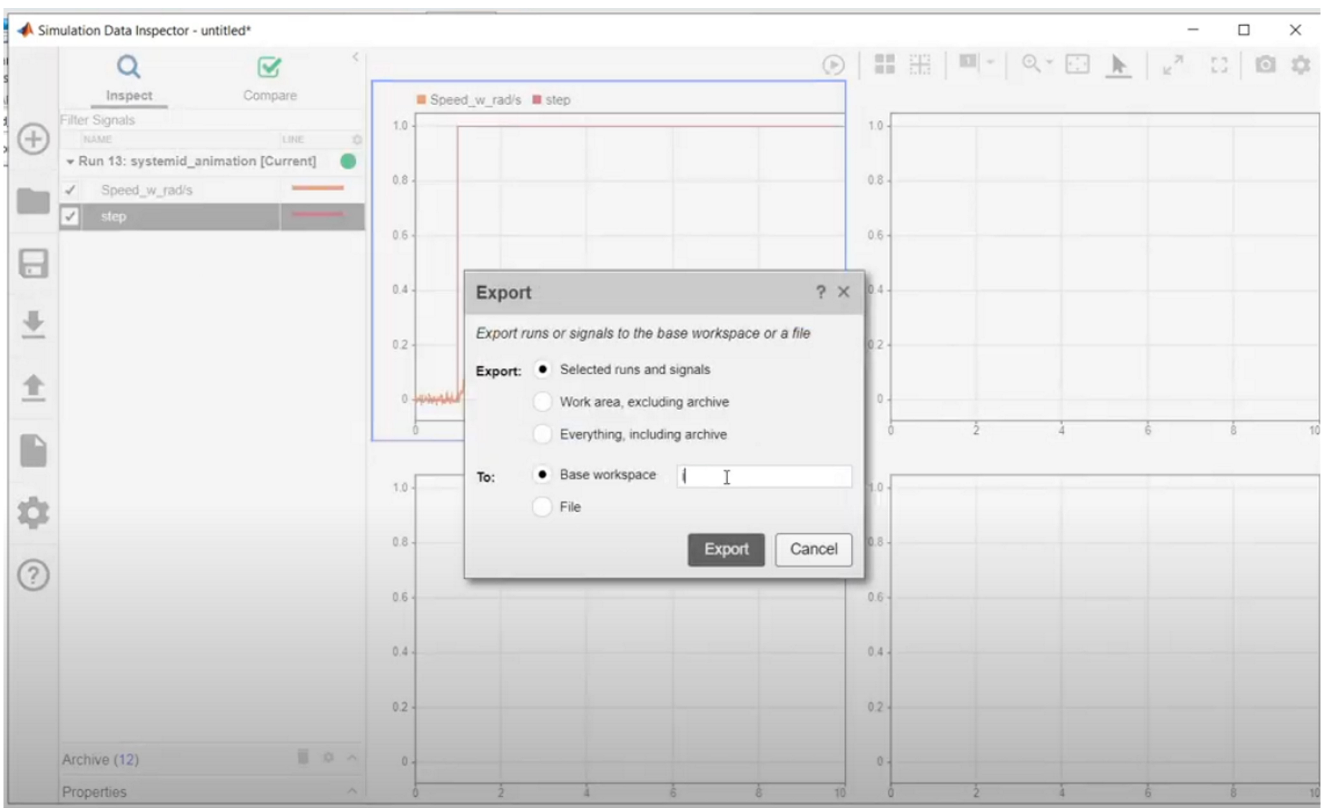
it shows me that there is an 87 percent fit that's great let us add this transfer function to a simulink file run the simulation and compare it with the model output

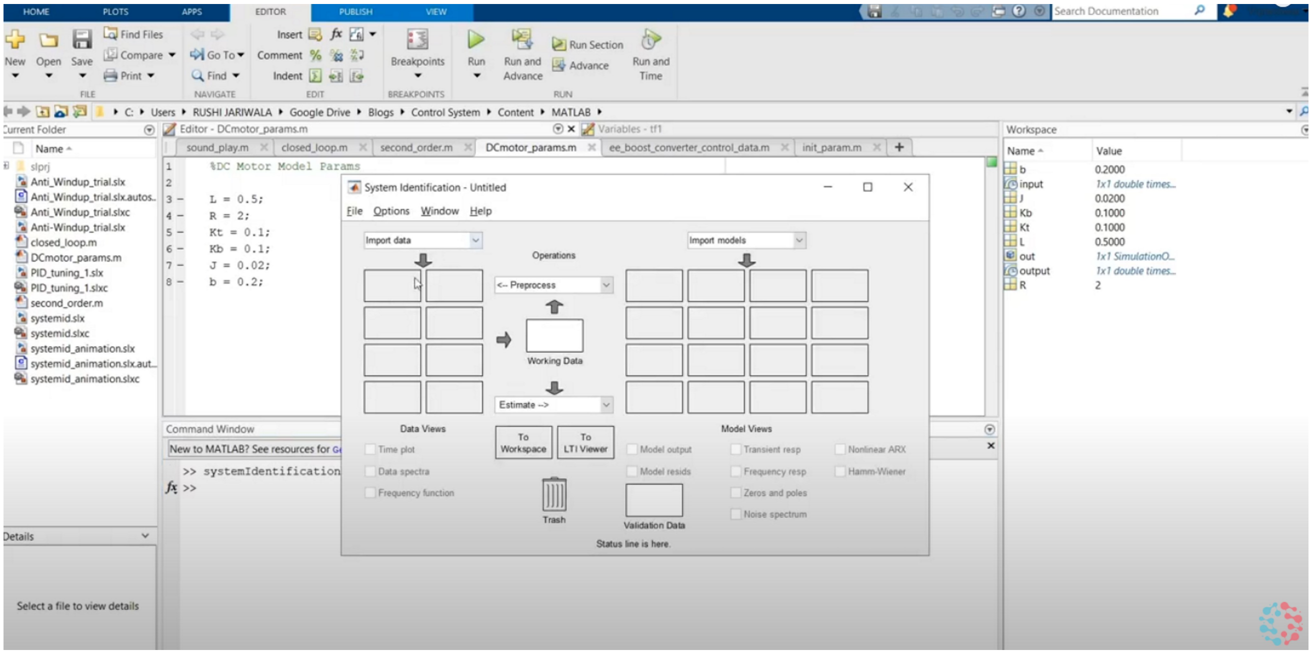
this is a good match the output from the derived model is very close to the output from the original model but we should also validate our new model with different inputs so as to confirm that the new model behaves reasonably well

for all kinds of inputs let us run the simulation again it matches well so system identification works for Non- Linear Systems us but we have a catch what we have assumed here is that the system is linear

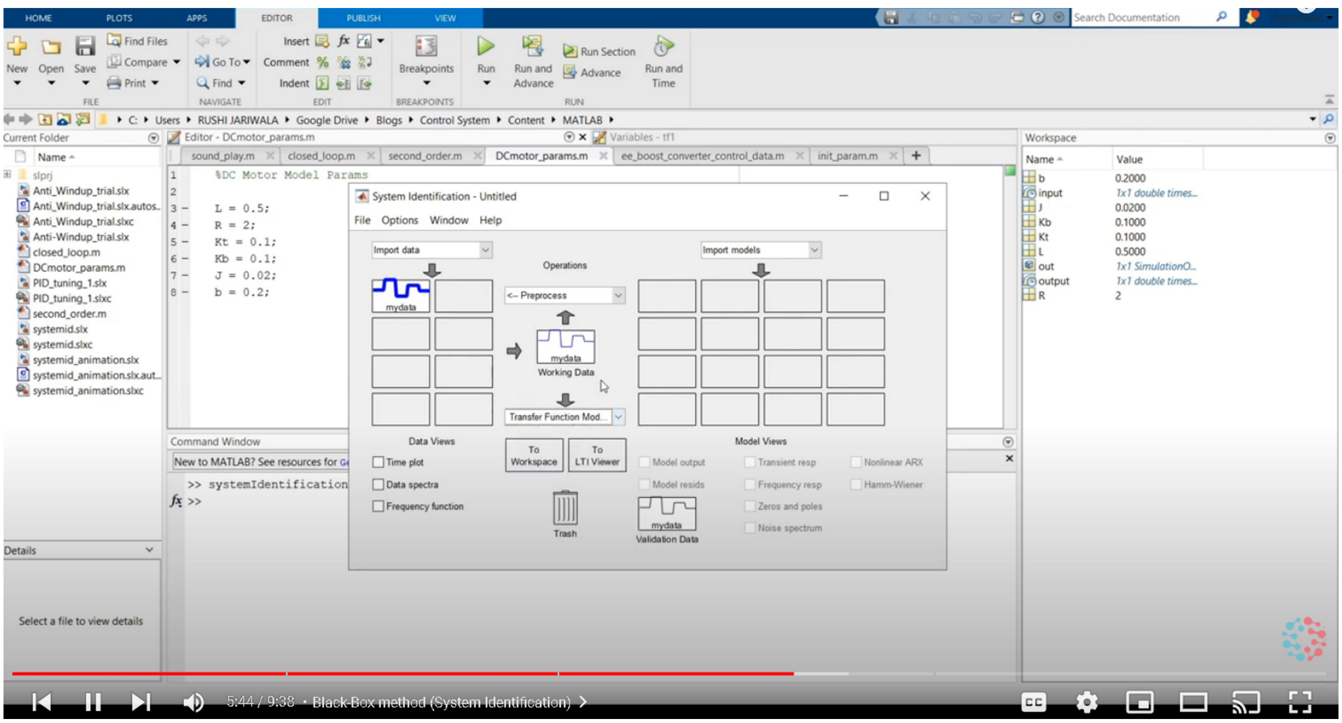



Simulation Data Inspector - untitled*
Inspect
Compare
Filter Signals
NAME
LINE
Run 13: systemid_animation [Current]
Highlight in Model
Export Data...
Compare Signals
Expand All Ctrl+Shift+=
Collapse All Ctrl+=
Rename
Delete... Del
Speed_w_rad/s
step
Archive (12)
Properties

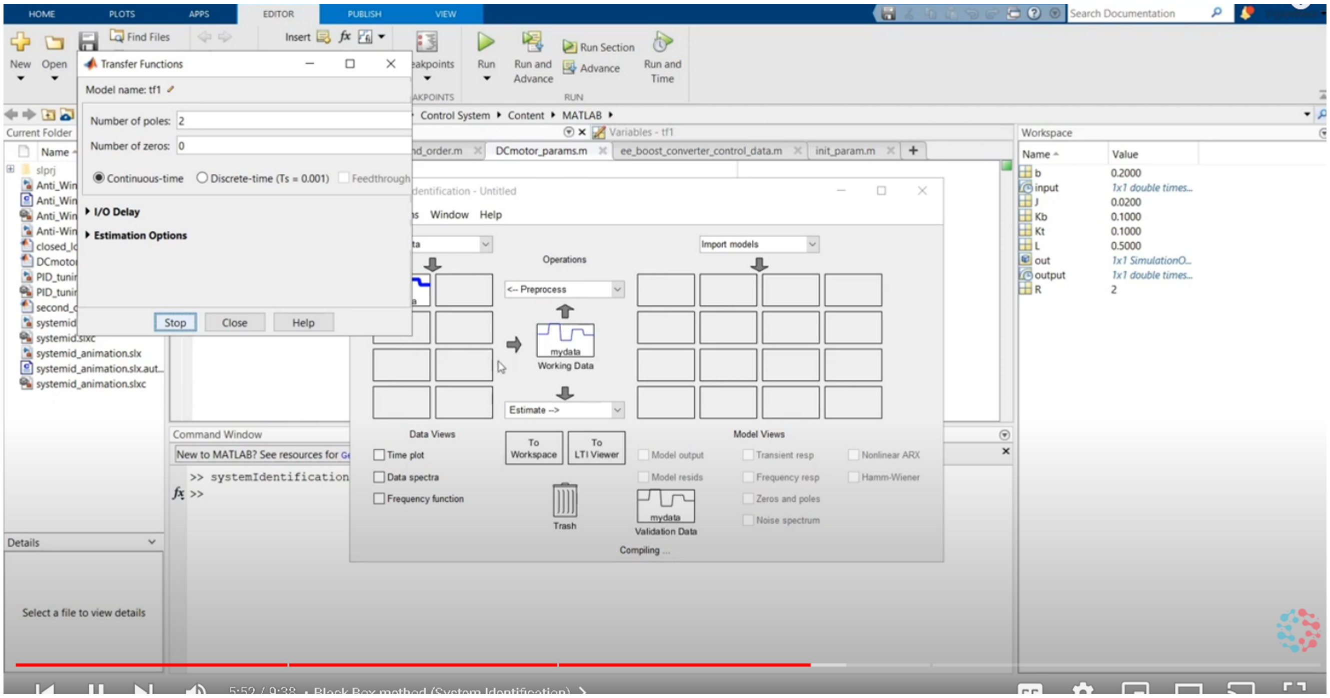

Simulation Data Inspector - untitled*
Inspect
Compare
Filter Signals
Run 13: systemid_animation [Current]
Speed_w_rad/s
step
Export
Export runs or signals to the base workspace or a file
Export:
Selected runs and signals
Work area, excluding archive
Everything, including archive
To:
Base workspace
File
Export
Cancel
Archive (12)
Properties

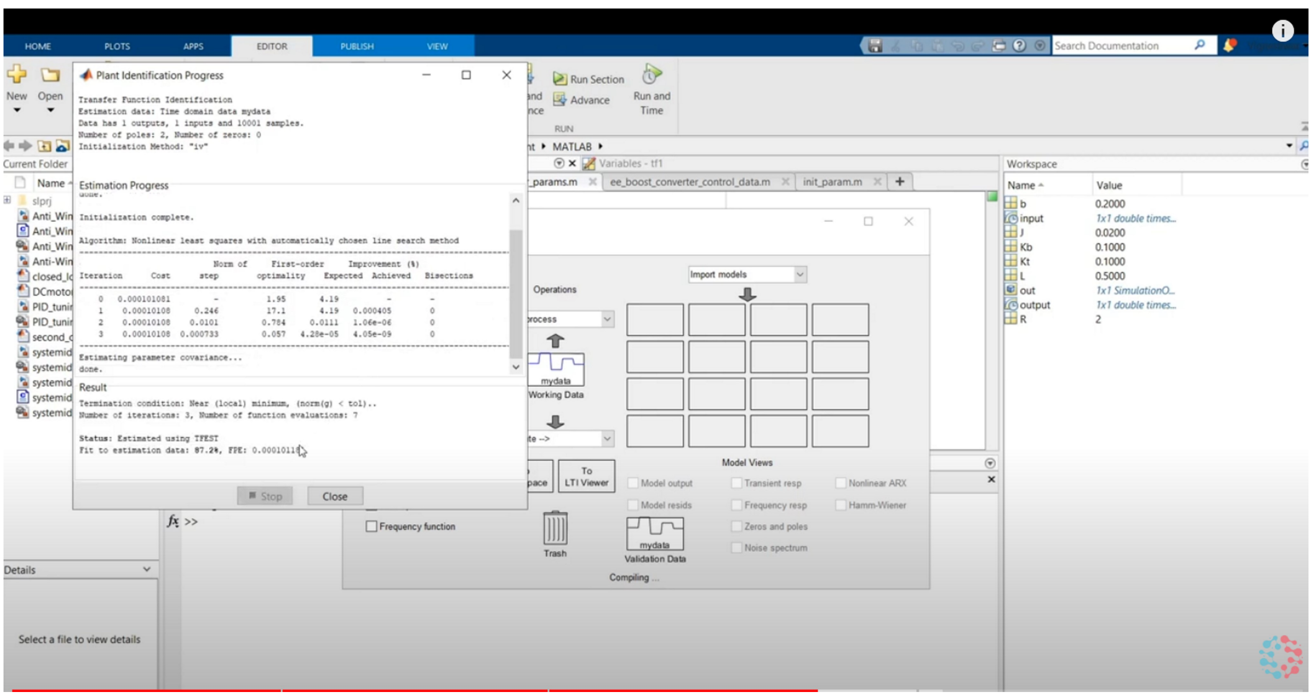


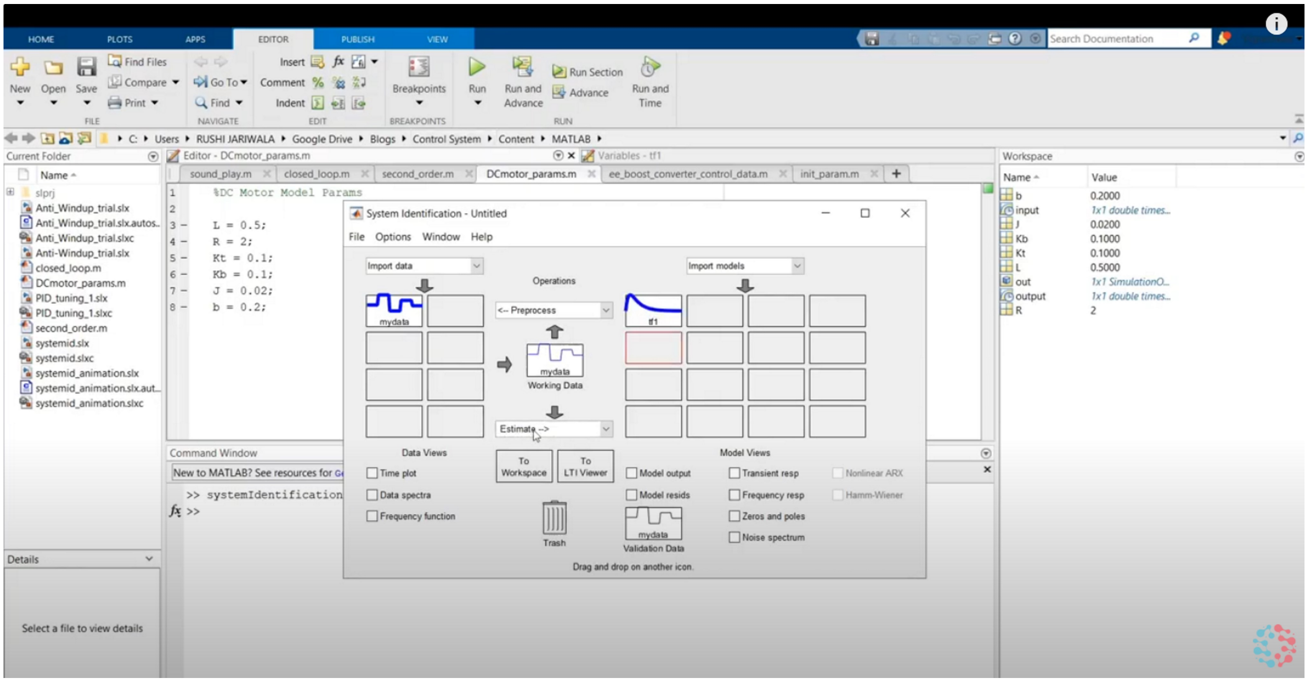
Editor - DCmotor_params.m
%DC Motor Model Params
L = 0.5;
R = 2;
Kt = 0.1;
Kb = 0.1;
J = 0.02;
b = 0.2;
System Identification - Untitled
File Options Window Help
Import data
Import models
Operations
<-- Preprocess
Working Data
Estimate -->
Data Views
Time plot
Data spectra
Frequency function
To Workspace
To LTI Viewer
Trash
Model Views
Model output
Model resids
Transient resp
Frequency resp
Zeros and poles
Noise spectrum
Nonlinear ARX
Hamm-Wiener
Validation Data
Status line is here.
Command Window
>> systemIdentification
Workspace
b 0.2000
input 1x1 double times...
J 0.0200
Kb 0.1000
Kt 0.1000
L 0.5000
out 1x1 SimulationO...
output 1x1 double times...
R 2

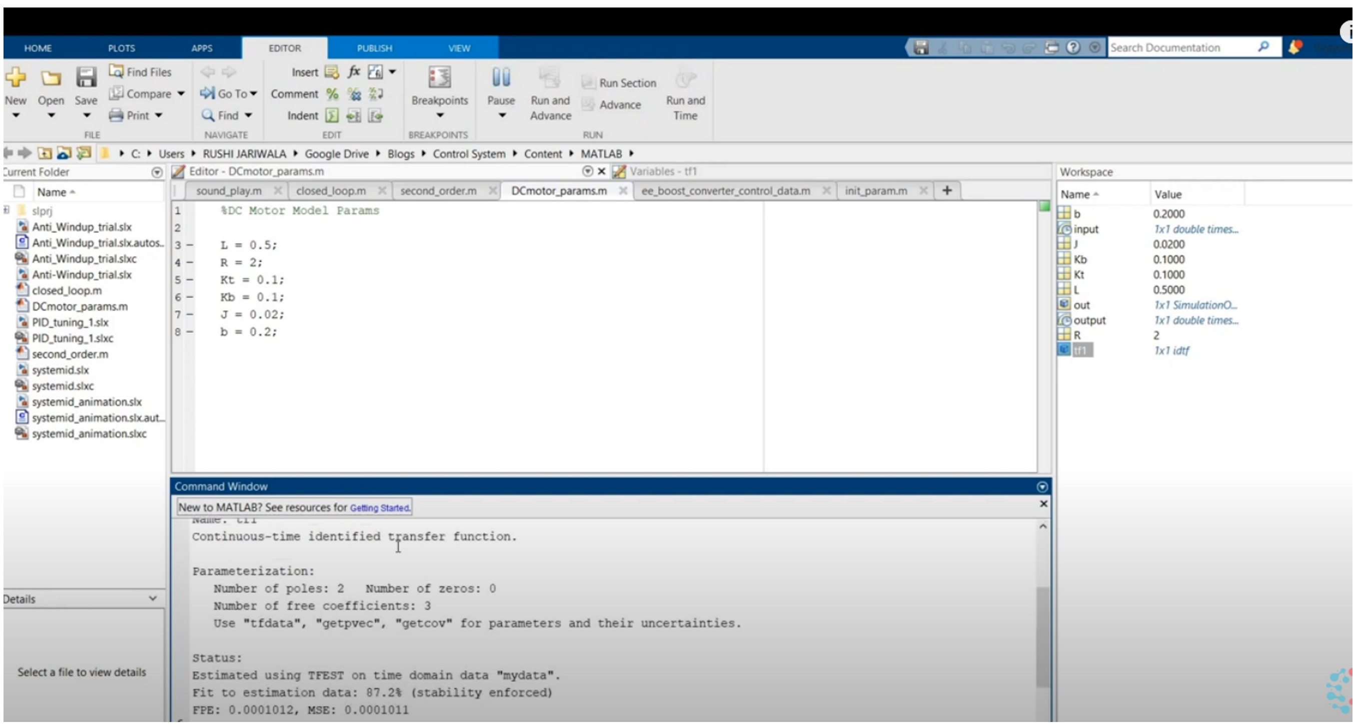


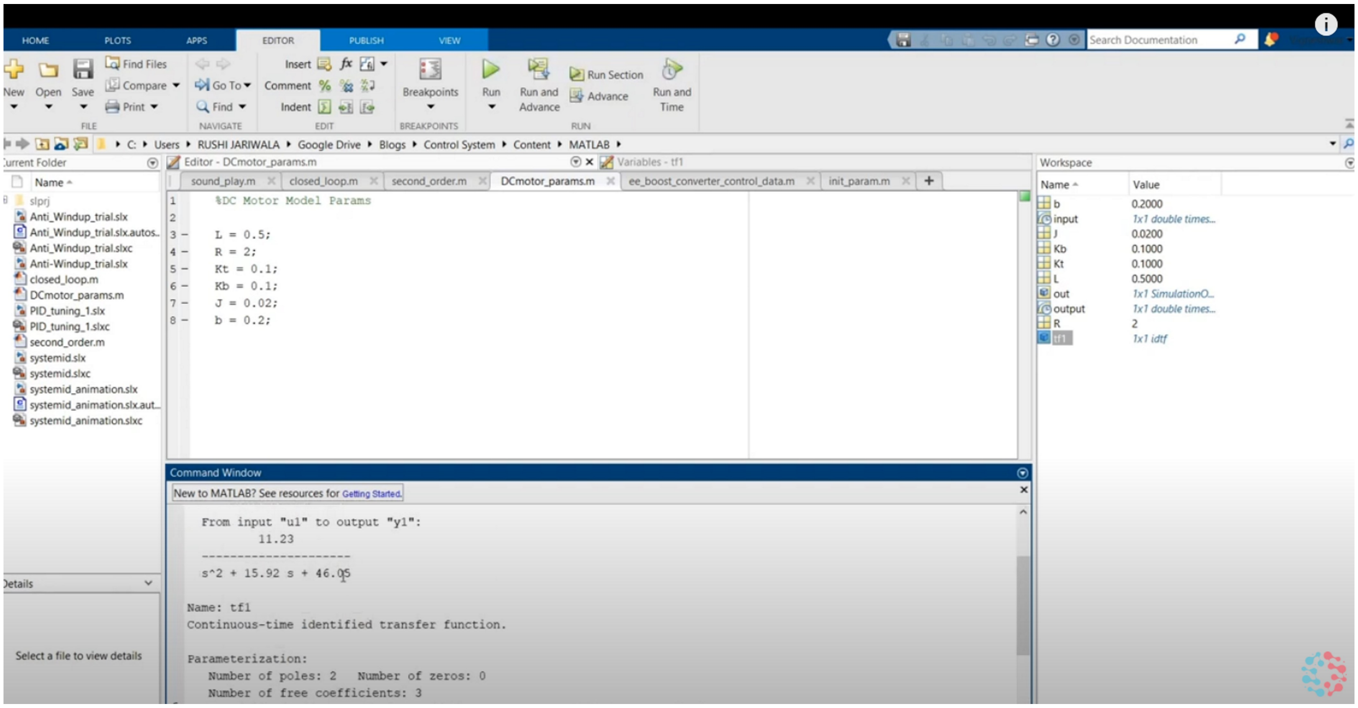
Editor - DCmotor_params.m
%DC Motor Model Params
L = 0.5;
R = 2;
Kt = 0.1;
Kb = 0.1;
J = 0.02;
b = 0.2;
System Identification - Untitled
File Options Window Help
Import data
Import models
mydata
Operations
<-- Preprocess
Working Data
Transfer Function Mod...
Data Views
Time plot
Data spectra
Frequency function
To Workspace
To LTI Viewer
Trash
Model Views
Validation Data
Command Window
>> systemIdentification
5:44 / 9:38 • Black-Box method (System Identification)


Transfer Functions
Model name: tf1
Number of poles: 2
Number of zeros: 0
Continuous-time
Discrete-time (Ts = 0.001)
Feedthrough
I/O Delay
Estimation Options
Stop
Close
Help
Operations
<-- Preprocess
Working Data
Estimate -->
Import models
Data Views
Time plot
Data spectra
Frequency function
To Workspace
To LTI Viewer
Trash
Model Views
Model output
Model resids
Transient resp
Frequency resp
Zeros and poles
Noise spectrum
Nonlinear ARX
Hamm-Wiener
Validation Data
Compiling ...
>> systemIdentification
Workspace
b 0.2000
J 0.0200
Kb 0.1000
Kt 0.1000
L 0.5000
R 2

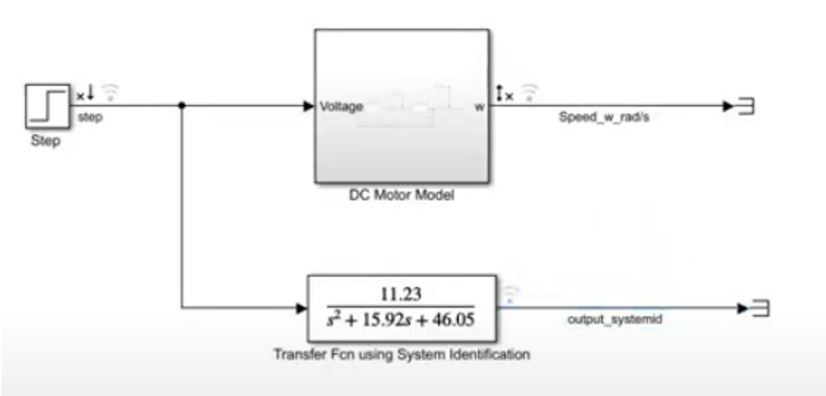

Plant Identification Progress
Transfer Function Identification
Estimation data: Time domain data mydata
Data has 1 outputs, 1 inputs and 10001 samples.
Number of poles: 2, Number of zeros: 0
Initialization Method: "iv"
Estimation Progress
Initialization complete.
Algorithm: Nonlinear least squares with automatically chosen line search method
Iteration Cost Norm of step First-order optimality Improvement (%) Expected Achieved Bisections
0 0.000101081 - 1.95 4.19 - -
1 0.00010108 0.246 17.1 4.19 0.000405 0
2 0.00010108 0.0101 0.784 0.0111 1.06e-06 0
3 0.00010108 0.000733 0.057 4.28e-05 4.05e-09 0
Estimating parameter covariance...
done.
Result
Termination condition: Near (local) minimum, (norm(g) < tol)..
Number of iterations: 3, Number of function evaluations: 7
Status: Estimated using TFEST
Fit to estimation data: 87.2%, FPE: 0.000101
Stop
Close


System Identification - Untitled
File Options Window Help
Import data
Operations
Import models
mydata
<-- Preprocess
tf1
mydata
Working Data
Estimate -->
Data Views
To Workspace
To LTI Viewer
Model Views
Time plot
Data spectra
Frequency function
Trash
Model output
Model resids
mydata
Validation Data
Transient resp
Frequency resp
Zeros and poles
Noise spectrum
Nonlinear ARX
Hamm-Wiener
Drag and drop on another icon.
%DC Motor Model Params
L = 0.5;
R = 2;
Kt = 0.1;
Kb = 0.1;
J = 0.02;
b = 0.2;
>> systemIdentification

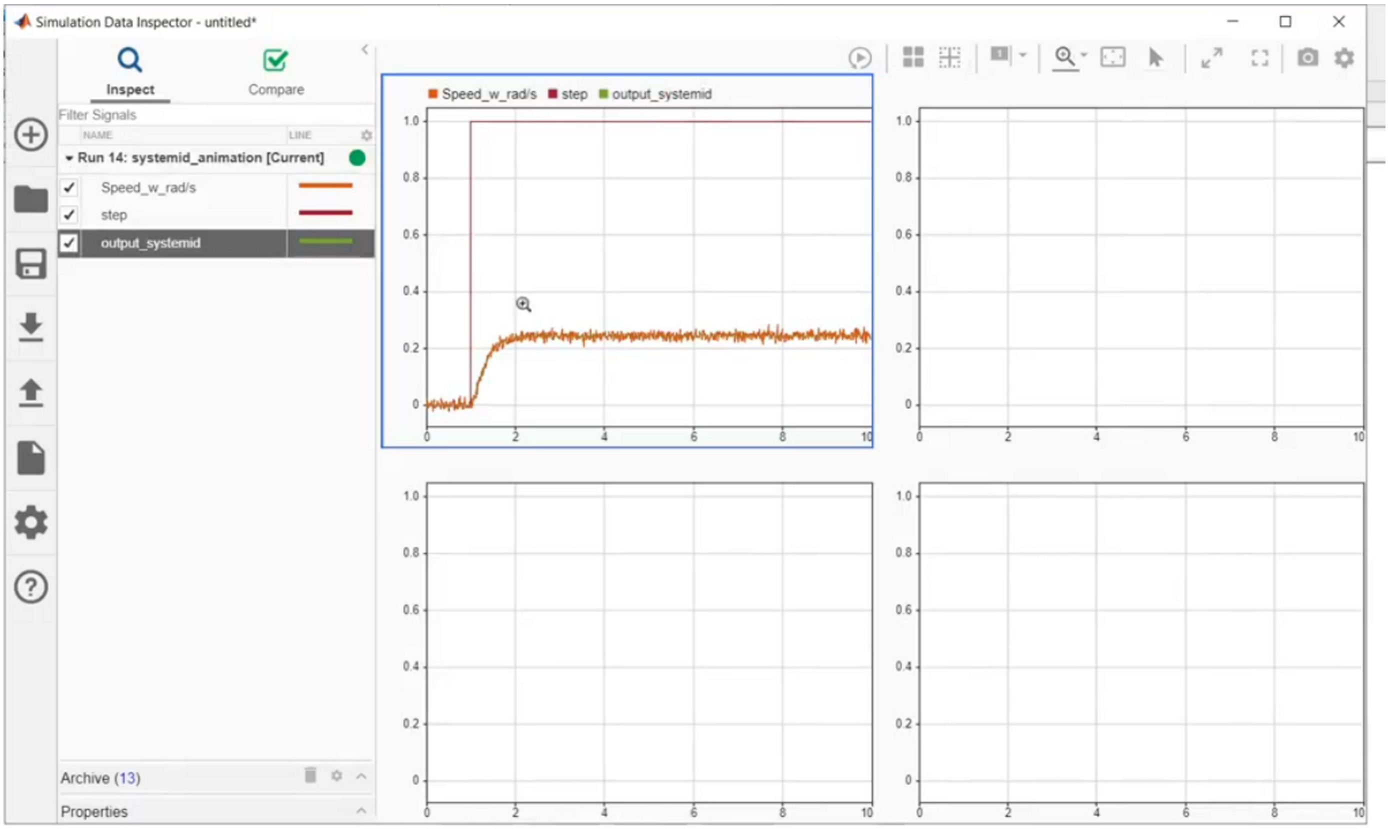

%DC Motor Model Params
L = 0.5;
R = 2;
Kt = 0.1;
Kb = 0.1;
J = 0.02;
b = 0.2;
Command Window
Continuous-time identified transfer function.
Parameterization:
Number of poles: 2 Number of zeros: 0
Number of free coefficients: 3
Use "tfdata", "getpvec", "getcov" for parameters and their uncertainties.
Status:
Estimated using TFEST on time domain data "mydata".
Fit to estimation data: 87.2% (stability enforced)
FPE: 0.0001012, MSE: 0.0001011



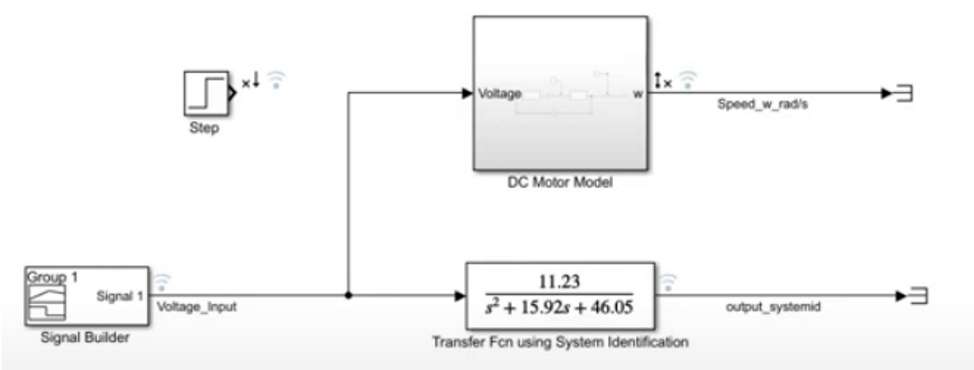
Editor - DCmotor_params.m
sound_play.m
closed_loop.m
second_order.m
DCmotor_params.m
ee_boost_converter_control_data.m
init_param.m
%DC Motor Model Params
L = 0.5;
R = 2;
Kt = 0.1;
Kb = 0.1;
J = 0.02;
b = 0.2;
Command Window
New to MATLAB? See resources for Getting Started.
From input "u1" to output "y1":
11.23
----
s^2 + 15.92 s + 46.05
Name: tf1
Continuous-time identified transfer function.
Parameterization:
Number of poles: 2   Number of zeros: 0
Number of free coefficients: 3
Workspace
Name
Value
b
0.2000
input
1x1 double times...
J
0.0200
Kb
0.1000
Kt
0.1000
L
0.5000
out
1x1 SimulationO...
output
1x1 double times...
R
2
tf1
1x1 idtf
Current Folder
slprj
Anti_Windup_trial.slx
Anti_Windup_trial.slxc
Anti-Windup_trial.slx
closed_loop.m
DCmotor_params.m
PID_tuning_1.slx
PID_tuning_1.slxc
second_order.m
systemid.slx
systemid.slxc
systemid_animation.slx
systemid_animation.slxc
Select a file to view details



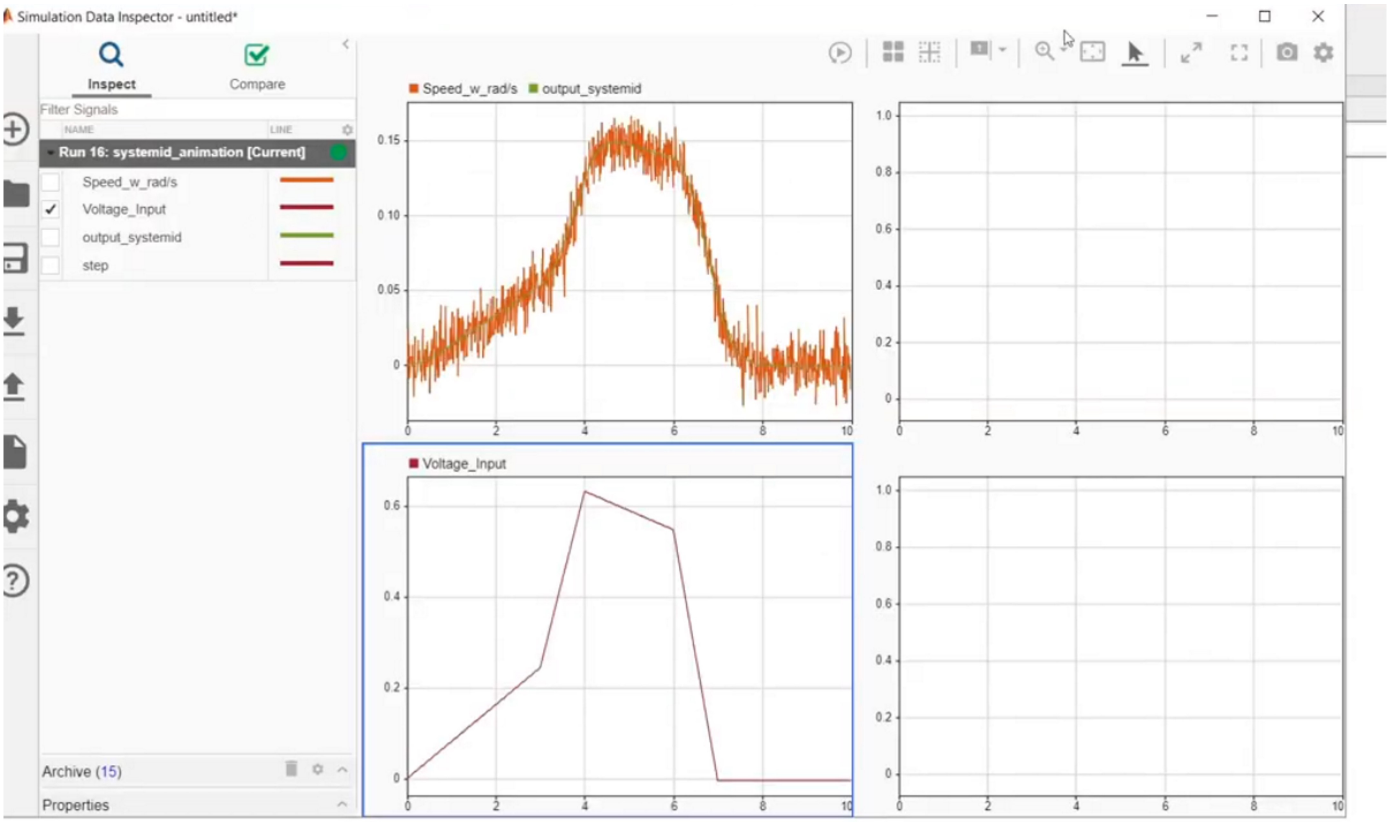
step
Step
Voltage
w
DC Motor Model
Speed_w_rad/s
11.23
$s^2 + 15.92s + 46.05$
Transfer Fcn using System Identification
output_systemid

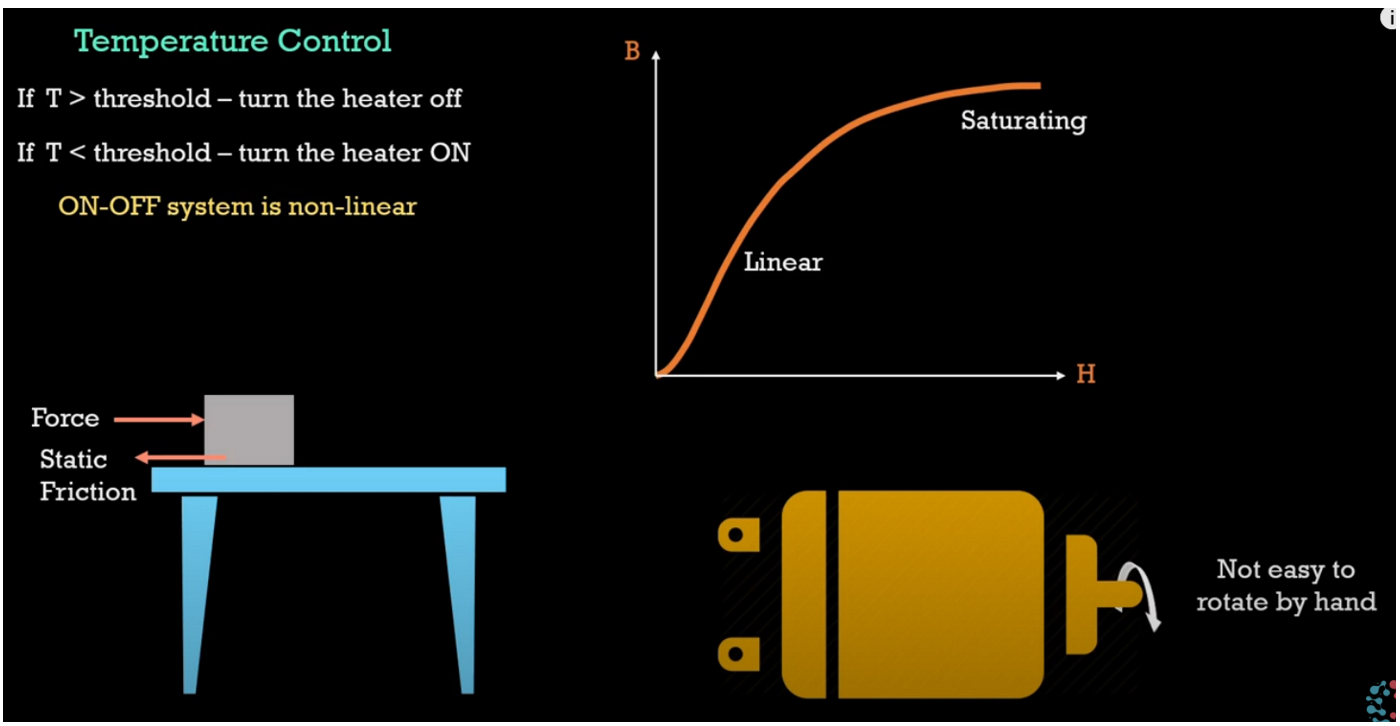


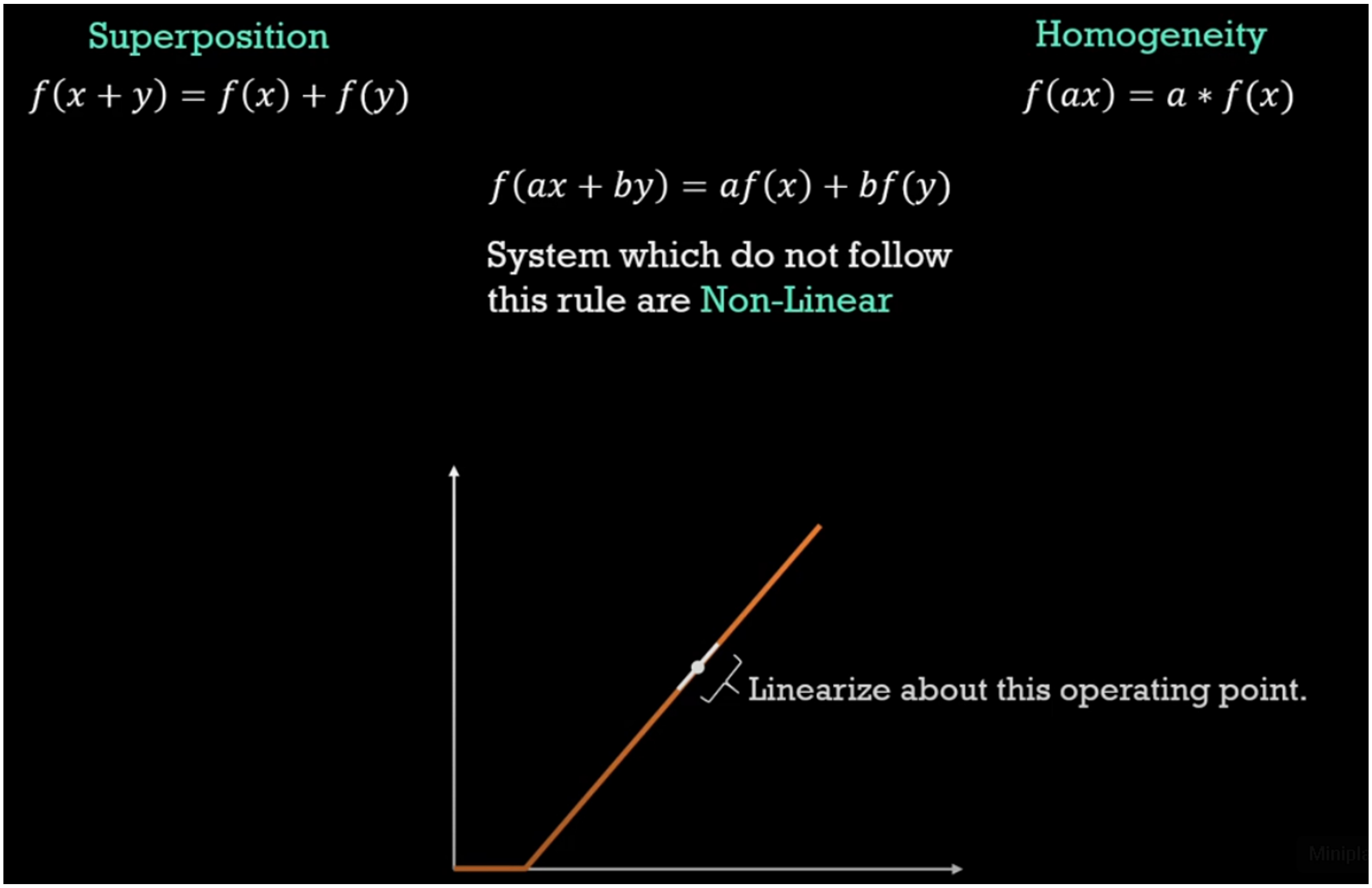
Simulation Data Inspector - untitled*
Inspect
Compare
Filter Signals
NAME
LINE
Run 14: systemid_animation [Current]
Speed_w_rad/s
step
output_systemid
Archive (13)
Properties
Speed_w_rad/s
step
output_systemid

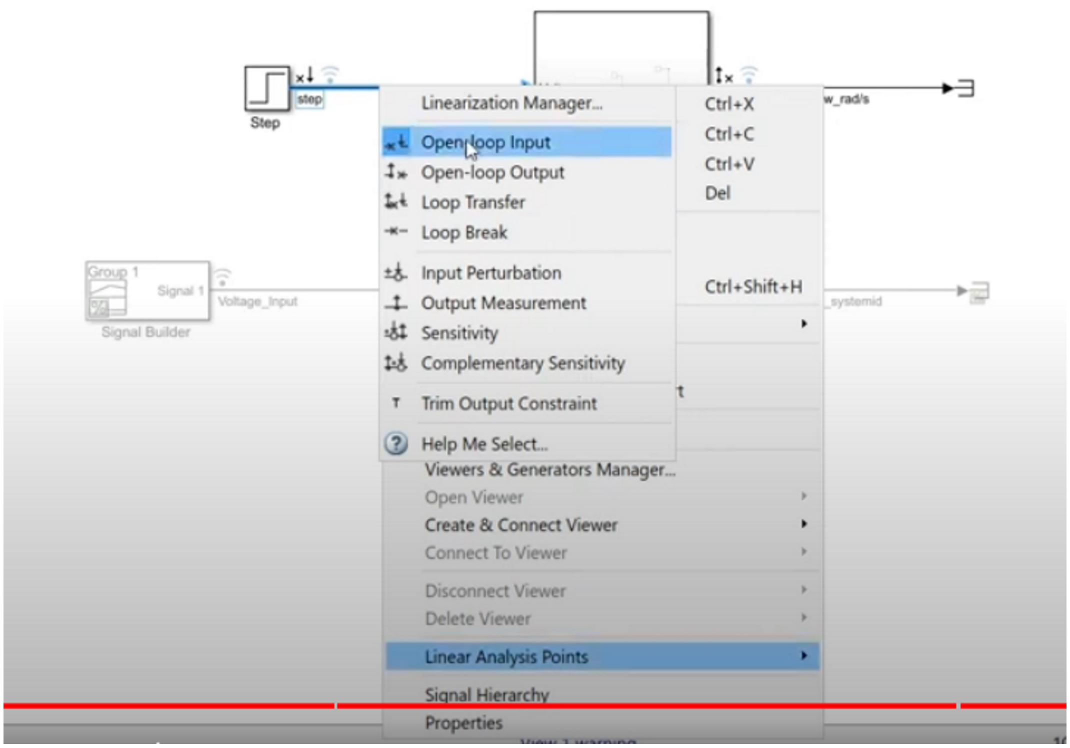


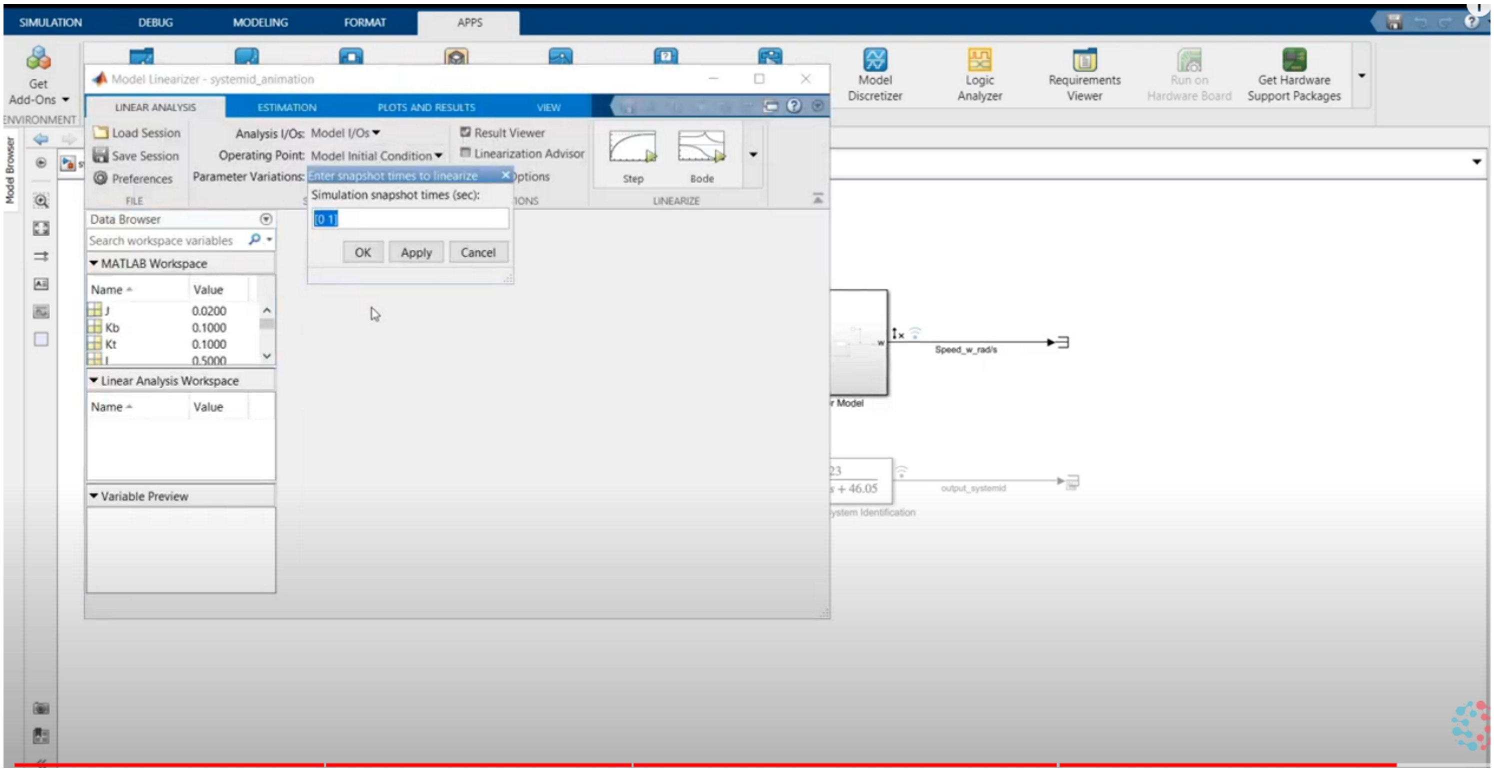
Step
Voltage
w
DC Motor Model
Speed_w_rad/s
Group 1
Signal 1
Signal Builder
Voltage_Input
11.23
$s^2 + 15.92s + 46.05$
Transfer Fcn using System Identification
output_systemid

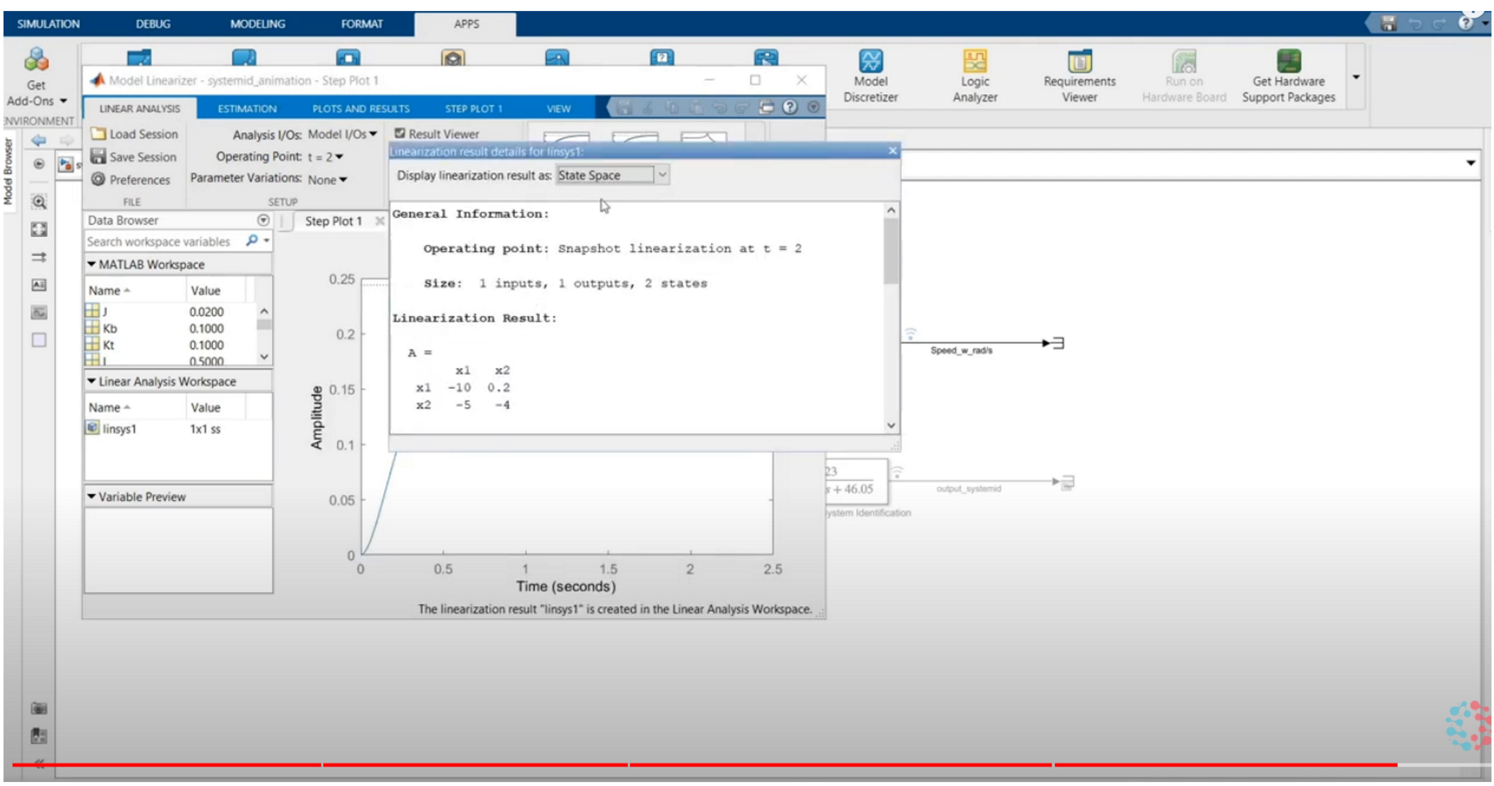

Simulation Data Inspector - untitled*
Inspect
Compare
Filter Signals
NAME
LINE
Run 16: systemid_animation [Current]
Speed_w_rad/s
Voltage_Input
output_systemid
step
Archive (15)
Properties
Speed_w_rad/s
output_systemid
Voltage_Input

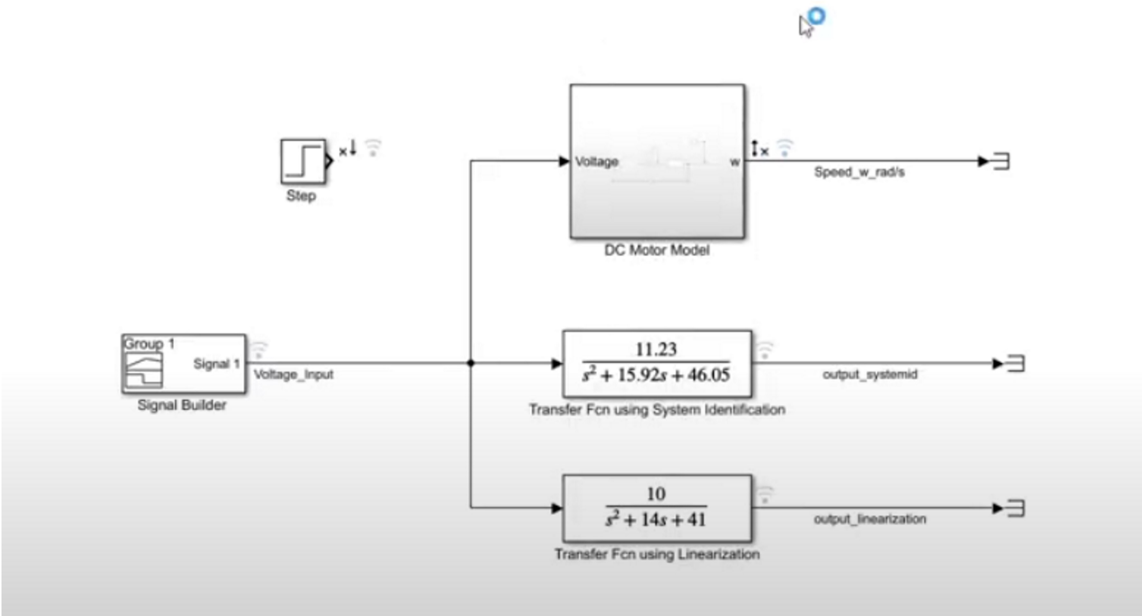

Temperature Control
If T > threshold – turn the heater off
If T < threshold – turn the heater ON
ON-OFF system is non-linear
B
Saturating
Linear
H
Force
Static
Friction
Not easy to
rotate by hand

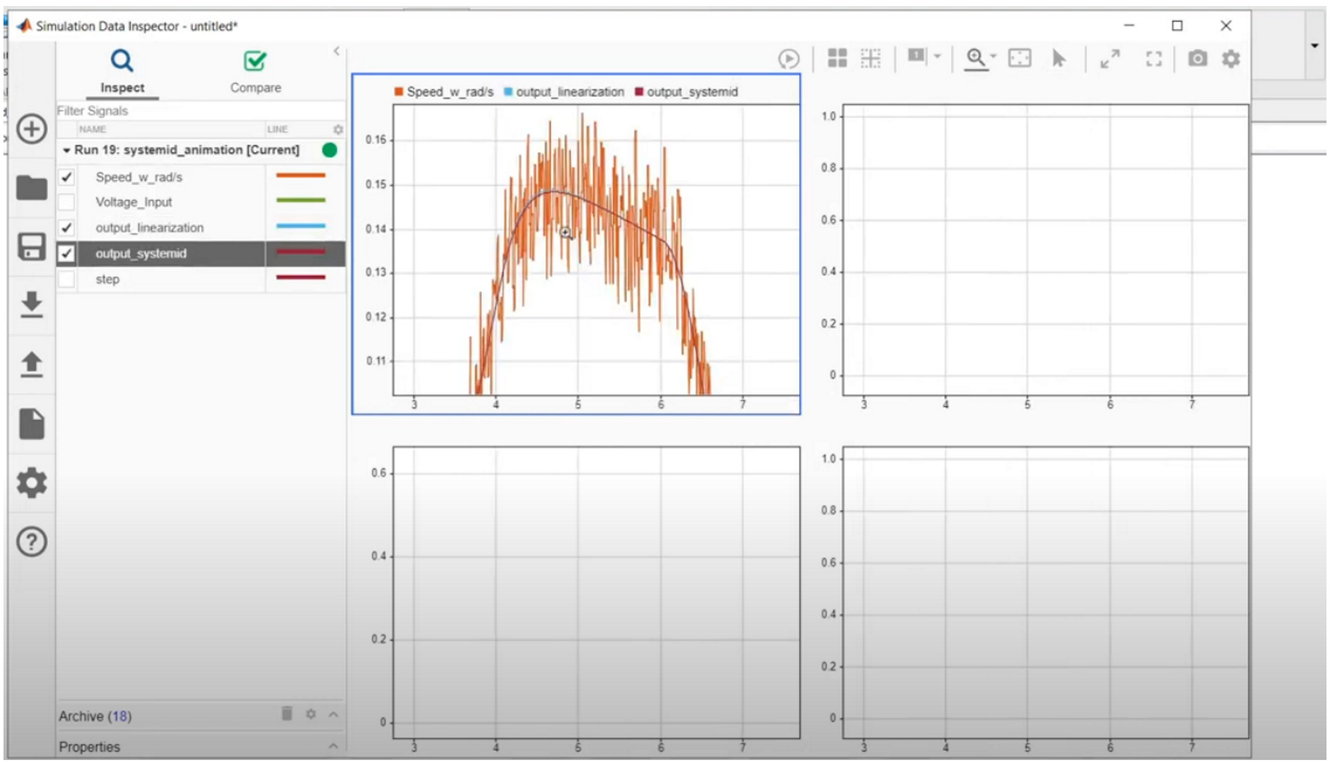

Superposition
$f(x+y) = f(x) + f(y)$
Homogeneity
$f(ax) = a * f(x)$
$f(ax+by) = af(x) + bf(y)$
System which do not follow this rule are Non-Linear
Linearize about this operating point.

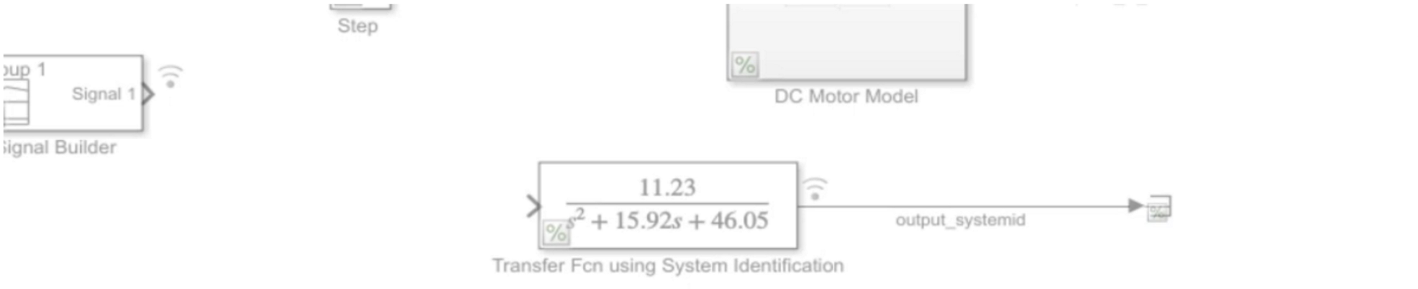

Step
step
w_rad/s
Linearization Manager...
Open-loop Input
Open-loop Output
Loop Transfer
Loop Break
Input Perturbation
Output Measurement
Sensitivity
Complementary Sensitivity
Trim Output Constraint
Help Me Select...
Ctrl+X
Ctrl+C
Ctrl+V
Del
Ctrl+Shift+H
Signal Builder
Voltage_Input
systemid
Viewers & Generators Manager...
Open Viewer
Create & Connect Viewer
Connect To Viewer
Disconnect Viewer
Delete Viewer
Linear Analysis Points
Signal Hierarchy
Properties

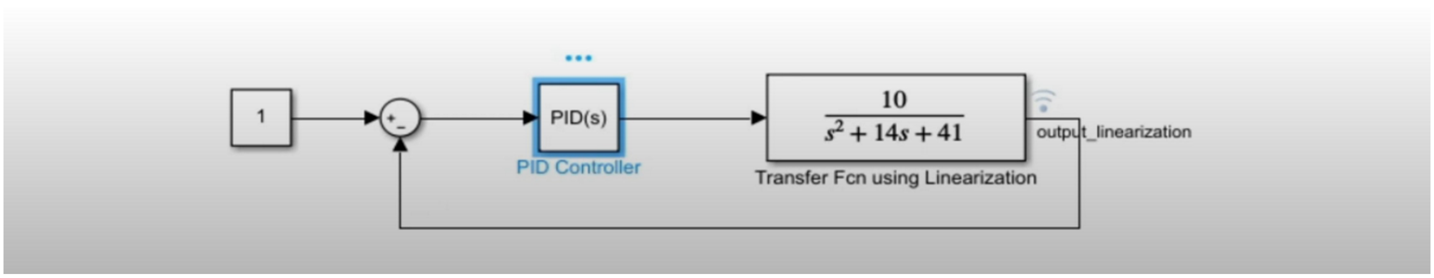

SIMULATION
DEBUG
MODELING
FORMAT
APPS
Model Linearizer - systemid_animation
LINEAR ANALYSIS
ESTIMATION
PLOTS AND RESULTS
VIEW
Load Session
Save Session
Preferences
Analysis I/Os: Model I/Os
Operating Point: Model Initial Condition
Parameter Variations:
Enter snapshot times to linearize
Simulation snapshot times (sec):
[0 1]
OK
Apply
Cancel
Result Viewer
Linearization Advisor
Step
Bode
LINEARIZE
Data Browser
Search workspace variables
MATLAB Workspace
Name
Value
J 0.0200
Kb 0.1000
Kt 0.1000
Linear Analysis Workspace
Variable Preview
Model Discretizer
Logic Analyzer
Requirements Viewer
Run on Hardware Board
Get Hardware Support Packages
Speed_w_rad/s
output_systemid


Model Linearizer - systemid_animation - Step Plot 1
LINEAR ANALYSIS
ESTIMATION
PLOTS AND RESULTS
STEP PLOT 1
VIEW
Analysis I/Os: Model I/Os
Operating Point: t = 2
Parameter Variations: None
Linearization result details for linsys1:
Display linearization result as: State Space
General Information:
Operating point: Snapshot linearization at t = 2
Size: 1 inputs, 1 outputs, 2 states
Linearization Result:
A =
x1 x2
x1 -10 0.2
x2 -5 -4
Step Plot 1
Linear Analysis Workspace
linsys1 1x1 ss
Amplitude
Time (seconds)
The linearization result "linsys1" is created in the Linear Analysis Workspace.


Step
Voltage
w
Speed_w_rad/s
DC Motor Model
Group 1
Signal 1
Signal Builder
Voltage_Input
$\frac{11.23}{s^2+15.92s+46.05}$
output_systemid
Transfer Fcn using System Identification
$\frac{10}{s^2+14s+41}$
output_linearization
Transfer Fcn using Linearization


Simulation Data Inspector - untitled*
Inspect
Compare
Filter Signals
NAME
LINE
Run 19: systemid_animation [Current]
Speed_w_rad/s
Voltage_Input
output_linearization
output_systemid
step
Archive (18)
Properties
Speed_w_rad/s
output_linearization
output_systemid





but not all systems are linear let us understand this statement using a few examples in the most basic temperature control system when the temperature exceeds the threshold you turn the heater off and when the temperature falls below the threshold you turn the heater on this on off system is non-linear in nature let us see the magnetization curve of a core we see that it is linear in some region and then it starts saturating luckily we operate in the linear region in most cases

and so we do not have to worry about the saturation but what about static friction if there is a heavy box kept on a table and you apply a force on it this box will not move unless a sufficient force is applied that exceeds the resistive force of the static friction

now consider a big motor you cannot rotate the shaft by the touch of your hand you have to apply sufficient torque for the motor to move this is the static friction and the cogging torque so you may never find a system that is completely linear

there will always be certain non-linearity which you have to take into account but how do you mathematically define non-linearity all systems that follow superposition and homogeneity are linear systems

if any one of this rule is violated then it is non-linear positive to see these equations so how do we control non-linear systems that would be an entire course on its own but in this video we'll see the simplest method let us linearize it at an operating point

for example in the case of static friction we are not worried much about the non-linear region as we know that the system is linear after that so we can select this operating point for example and then linearize this system around it

you might also have to linearize a system at multiple points if the system is highly non-linear i am not going into the mathematics of linearization

but let us get a quick look at how simulink can help us here we'll mark the input and outputs as point of linearization then we'll go to the model linearizer app and then we will select the operating point

we will select the step input and there we have the linearized transfer function

it is as easy as what you saw let us put this transfer function into simulink and see how it compares to the model we developed using first principles and using system identification pretty good i would say now we can complete a closed loop system add a pid controller and tune it for the models that we derived

in the next video we'll see how to tune this controller using different methods so we have seen a lot of different techniques to find the model of the system and i hope you do make use one of these in your

next project
see you next time