DATA-BASED MODELING OF ENGINE THROTTLE VALVE DYNAMICS

TABLE OF CONTENTS

This document describes nonlinear modeling of the engine throttle dynamics using data centric approaches available in System Identification Toolbox™ . Two approaches are described:

1. *Black box modeling*: case where you cannot derive the exact mathematical representation of the system from physical considerations; the form of the model as well as the values of its coefficients is extracted from data.
2. *Grey-box modeling*: the equations of motion relating the input and output variables are known, but the values of various physical constants in the equations are unknown; the data is then used to find the values of those unknowns only.

The emphasis is on the black box modeling approach. It will be shown that even though no a priori knowledge of model structure is required, it is often helpful to have some intuition about the nature of the system and to use this knowledge to fine-tune the configuration of model structures.

The following example were created using R2010a release. Some of the "Best Fit" value might be slightly different depending on the version of MATLAB® you are using.

INTRODUCTION TO PROBLEM

The throttle controls the air mass flow into the intake manifold of an engine. The throttle body contains a butterfly valve that opens when the driver presses down on the accelerator pedal. This lets more air enter the cylinders and causes the engine to produce more torque.

A DC motor controls the opening angle of the butterfly valve. There is also a spring attached to the valve to return it to its default position (15 degrees open) when the DC motor is de-energized. The amount of rotation of the valve is limited to approximately 90 degrees. Therefore, if a large command input is applied to the motor, the valve hits the hard stops preventing it from rotating further.

The task here is to create a mathematical representation of the dynamics of the DC motor, throttle valve combination. The model should describe the relationship between the input command to the driving motor (normalized voltage) and the resulting angle of the throttle valve (degrees).

FIGURE 1: A THROTTLE VALVE

DATA BASED MODELING APPROACH

We set up an experiment to measure the input and output signals of this system. The data was collected for several input profiles such as step signals of low and high amplitudes, pulse signals and multi-step signals. This resulted in a set of 12 input-output data sets all measured as a sampling frequency of 1kHz. Plots of the five chosen data sets used for modeling are shown below.

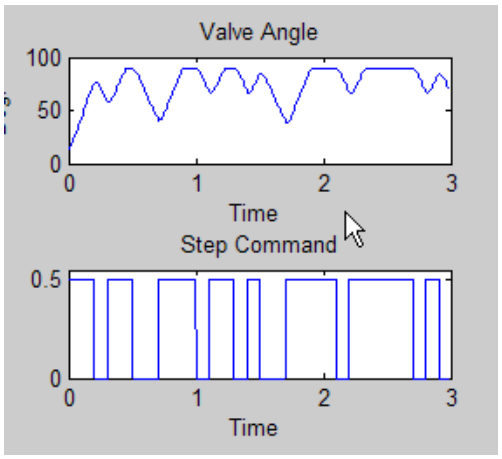


FIGURE 2: DATA SET 1

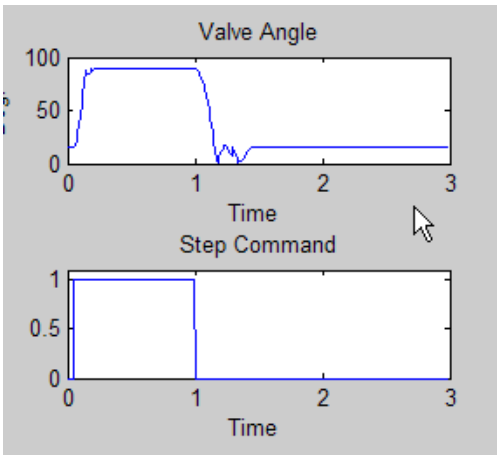


FIGURE 3: DATA SET 2

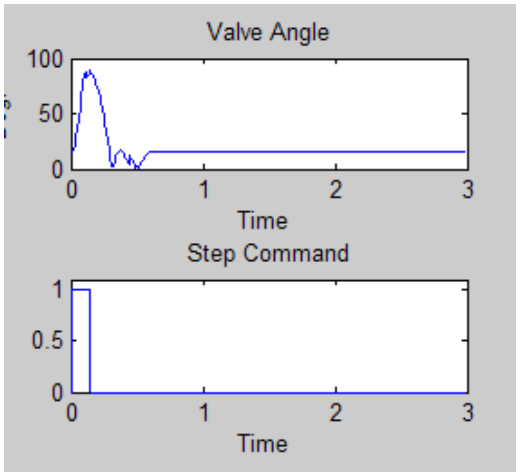


FIGURE 4: DATA SET 3

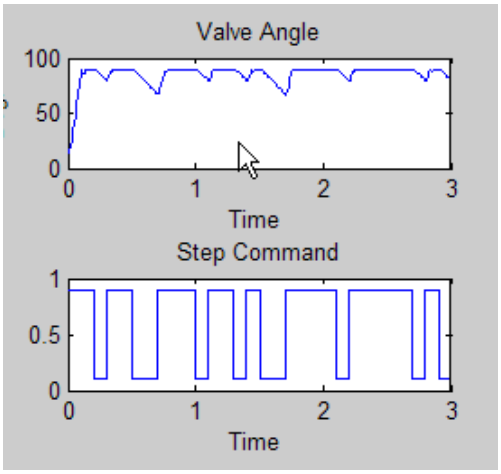


FIGURE 5: DATA SET 4

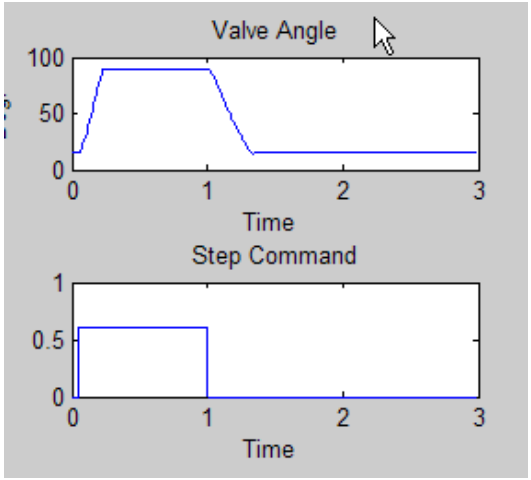


FIGURE 6: DATA SET 5

For our analysis, we divided the data into estimation and validation sets. Thus, the data sets 1, 2 and 3 were used for creation of models, while the data sets 4 and 5 were used for validating the performance of the model. The data sets were also down-sampled to a frequency of 100 Hz.

PROCESS OF DATA BASED MODEL CREATION

A data-centric approach to modeling is summarized by the flowchart in Figure 7. More detailed descriptions can be found in product documentations and textbooks; for example, see:

- http://www.mathworks.com/access/helpdesk/help/toolbox/ident/gs/brav7fy.html
- System Identification – Theory for the User, Lennart Ljung, Prentice Hall, 2nd ed., Section 1.4

As illustrated by this figure, the process of obtaining *models from data* calls for a systematic approach. You begin by designing a good experiment, denoting input/output signals of interest and recording their values with sufficient accuracy. Optionally, you perform some post-acquisition processing, such as choosing data segments with no outliers, filtering and down-sampling. The "massaged" data is then used to create a variety of models usually starting with the simplest (linear, low order) and then gradually trying more complex forms (higher orders, nonlinear forms) if required. You revisit your previous actions iteratively until good results are obtained. A "good result" mainly means you find a model that can reproduce the measured outputs when the same input signals are used. Tests of model's fidelity/performance are usually performed on a different data set than the one(s) used for estimation.

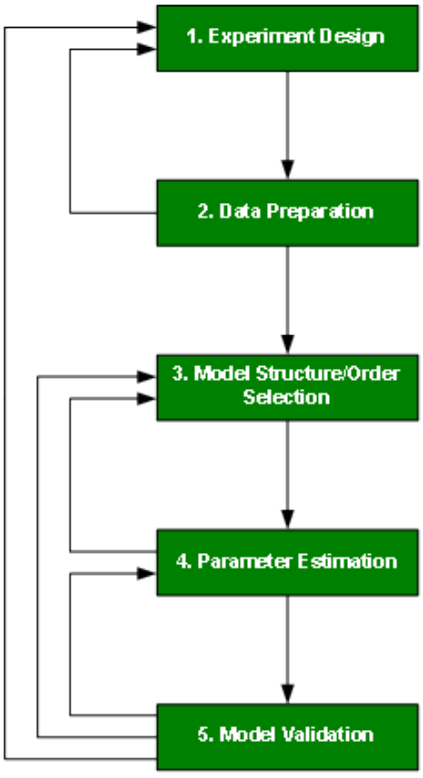


FIGURE 7: DATA BASED MODELING APPROACH

Keeping models as simple as possible is important – it not only makes the model more reliable (less variability in estimated parameter values), but also lowers the implementation costs, were you to deploy this model on actual hardware.

BLACK BOX MODELING APPROACH

We first try out a black box approach to modeling wherein we start with little or no knowledge of the nature of the system. This approach boils down to trying various model structures available in products such as System Identification Toolbox™ and Neural Network Toolbox™, configuring their exact forms (step 3), estimating the values of their coefficients (parameters) using data (step 4) and then comparing/contrasting them. The task ends with choosing one of these models that appears to emulate the observed behavior (output data values) reasonably well without being too complex. This approach is, to a certain extent, a trial and error approach. Hence some patience and time is required to try out various forms and estimating their parameters. In the following we show how one can go about choosing and configuring various model structures using physical insights and desire for simplicity. But before we do that, it would be useful to familiarize ourselves with some of the nonlinear model structures that are commonly employed for such tasks.

UNDERSTANDING NONLINEAR MODEL STRUCTURES

So that the whole modeling approach does not look like black magic, let us look a bit into their structures and where they come from. To begin with, you might be familiar with some forms of linear models. The underlying mathematical form of a linear model is a differential equation, or in discrete-time case, a difference equation. For example, the equation of motion of a mass-spring-damper system is a second order differential equation:

$$m\, d^2y/dt^2 + c\, dy/dt + k\, y(t) = F(t) \qquad \ldots\ldots\ldots\ldots (1)$$

where m is the mass, k the spring's stiffness constant and c the damping coefficient. y(t) is the displacement of the mass and F(t) is the input force. If you perform Laplace transform of Equation (1), you get the familiar transfer function form:

$$G(s) = X(s)/F(s) = 1/(ms^2+cs+k) \qquad \ldots\ldots\ldots\ldots (2)$$

Similarly, if you designate the displacement and velocity as state-variables, you can represent the equation of motion in state-space form as:

$$dX/dt = A\,X(t) + B\,F(t)$$

$$x(t) = C\,X(t)$$

where X(t) = [y(t); v(t)] is the vector of model states. The matrices A, B, and C are related to the constants m, c and k.

If you discretize the differential equation above using backward Euler formula: dx/dt = (x(t)-x(t-Ts))/Ts, you get a difference equation that shows the relationship between the uniformly sampled values of input F(kTs) and output y(kTs), k = 0, 1, 2, …; Ts = sampling interval:

$$y(kTs) + a_1\, y((k-1)Ts) + a_2\, y((k-2)Ts) = b_0\, F(kTs)$$

Using Ts = 1 for notational simplicity and moving the delayed terms to RHS, we have the following difference equation:

$$\mathbf{y(k) = -\, a_1\, y(k-1) - a_2 y(k-2) + b_0\, F(k),\ k = 0, 1, \ldots} \qquad \ldots\ldots\ldots\ldots (3)$$

The above equation is quite instructive: it shows how the value of the output at any time k can be computed as *a weighted sum* of current value of input and past values of output. In an identification task, you have to determine the values of these weights – a1, a2 and b0 such that the model's output y(k) is as close as possible to its measured values.

NONLINEAR MODELS AS EXTENSION OF LINEAR ONES

NONLINEAR ARX MODELS

In the difference equation (3) above, the output y(k) is a weighted sum of variables y(k-1), y(k-2) and F(k). In Statistics and System Identification literature, these variables are called *regressors*. Thus, the above equation is en example of a *linear-in-regressors model*: the output is a linear combination of three regressors. This formulation forms the basis of nonlinear ARX models that we will use to model the throttle dynamics. Nonlinear ARX models use a nonlinear combination of

regressors to compute the output values. To see how this happens, inspect Equation (3) again. If you wanted to "extend" or "generalize" this equation to make it more flexible, you have two options:

1. Use more complicated regressors: the regressors need not be simple time-delayed versions of the input/output variables. They can be more complicated forms such as |(F(k)|, y(k-1)*y(k-5), F(k-2)*y(k-3)^2, min(y(k-1), 90) etc. System Identification Toolbox allows you to create and use any arbitrary forms of regressors. Sometimes, you create regressors based on your knowledge of the physics of the system. We shall explore that in the following sections.
2. In place of a simple weighted sum of regressors, use more complex, nonlinear mappings – y(k) = f(R(k)), there R(k) denotes the set of regressors, and f() is a nonlinear function. For example, f() could be a sum of 10 sigmoid functions each using a different amplification, dilation and translation factors. This way, your nonlinear model becomes what is called a "sigmoid network". Similarly you can have "wavelet networks", "neural networks" and other forms.

System Identification Toolbox and Neural Network Toolbox allow you to create a variety of nonlinear models using above generalizations. However, use of arbitrary regressors is supported only by System Identification Toolbox. Also, both products offer a slightly different list of nonlinear functions f() to use. For example, System Identification Toolbox lets you use wavelet networks, sigmoid networks and binary tree (among others), while Neural Network Toolbox lets you use sigmoid and tansig networks (among others).

HAMMERSTEIN-WIENER MODELS

System Identification Toolbox also offers a whole different class of nonlinear models known as models with I/O nonlinearities, or *Hammerstein-Wiener* models. As before the best way to understand them is to view them as extensions of linear models. We are all familiar with block diagram representation of systems. Thus if you represent the linear transfer function (Equation (2)) by a linear block, we have:

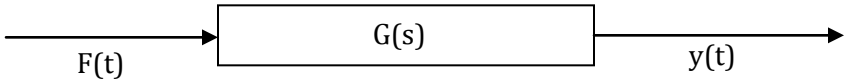


FIGURE 8: BLOCK DIAGRAM OF A LINEAR MODEL

The above block diagram may be represented by using, for example, a transfer function block in Simulink®. One way to extend this form to handle complex behavior is to add nonlinear elements at the input and/or output ports:

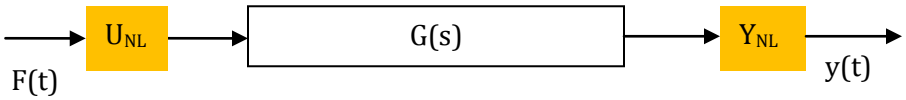


FIGURE 9: HAMMERSTEIN-WIENER MODEL

where U_{NL} refers to a nonlinear function at the input port and Y_{NL} refers to the nonlinear function at the output port. You need not have both U_{NL} and Y_{NL} simultaneously present. This form allows convenient extension of linear models to a nonlinear form. For example if you have a linear system

that you are modeling, but the sensors you are using it measure its behavior has nonlinearities at high gains (such as saturation), you can use a Wiener model that uses a linear block combined with a saturation block at its output port. The forms of these nonlinear functions are often guided by physical insight, although you can also use arbitrary forms to capture nonlinear effects whose nature might be unknown. The parameters of the composite form can be calculated in one shot using estimation routines in System Identification Toolbox. For more information, see:

http://www.mathworks.com/access/helpdesk/help/toolbox/ident/ug/bq2ix15.html

ESTIMATING MODELS USING SYSTEM IDENTIFICATION TOOL

First, we try out some model structures available in System Identification Toolbox. For this, we launch the GUI called System Identification Tool (Figure 10).

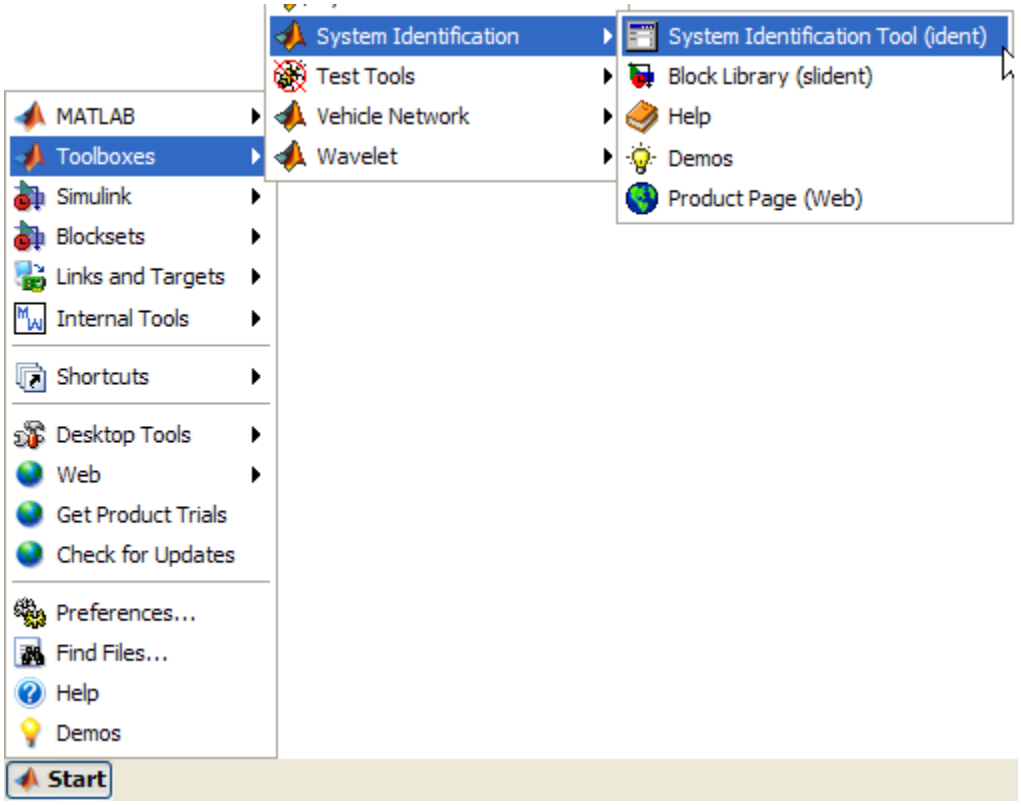


FIGURE 10: LAUNCHING THE GUI

When the GUI comes up, we begin our identification task by importing input and output data sets (Figure 11). The import dialog asks for variable names of input and output data vectors that must be available in the MATLAB workspace before importing. This results in several icons appearing in the data board of the GUI, each representing a data set (Figure 12)

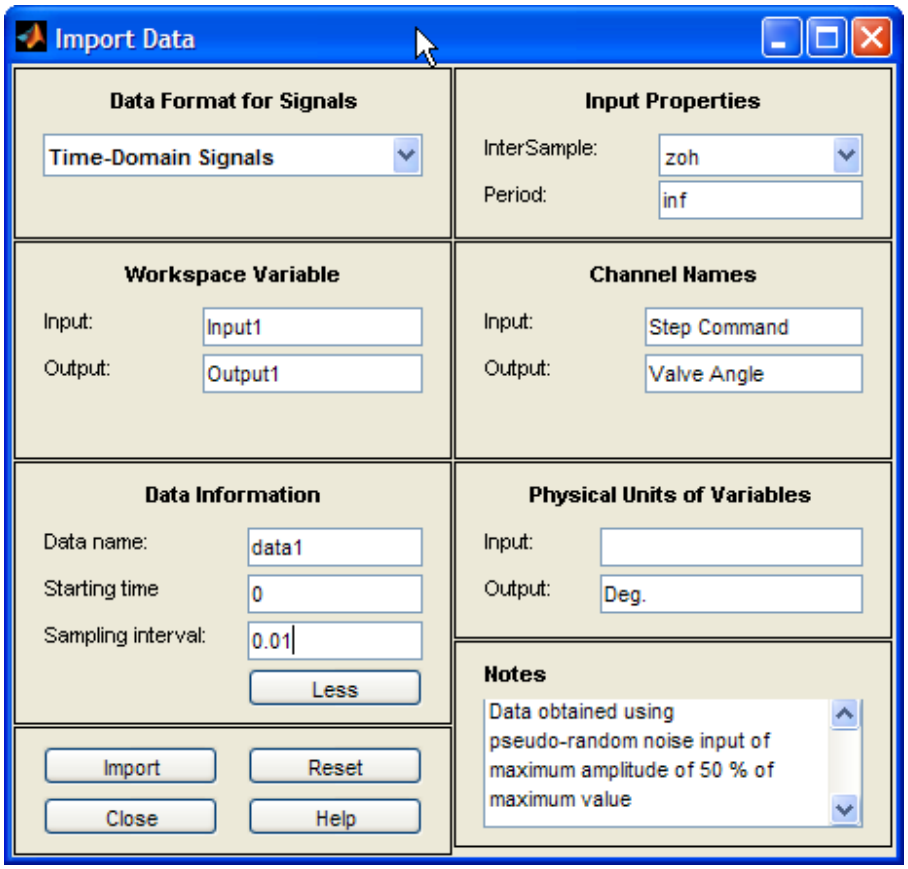


FIGURE 11: DATA IMPORT DIALOG

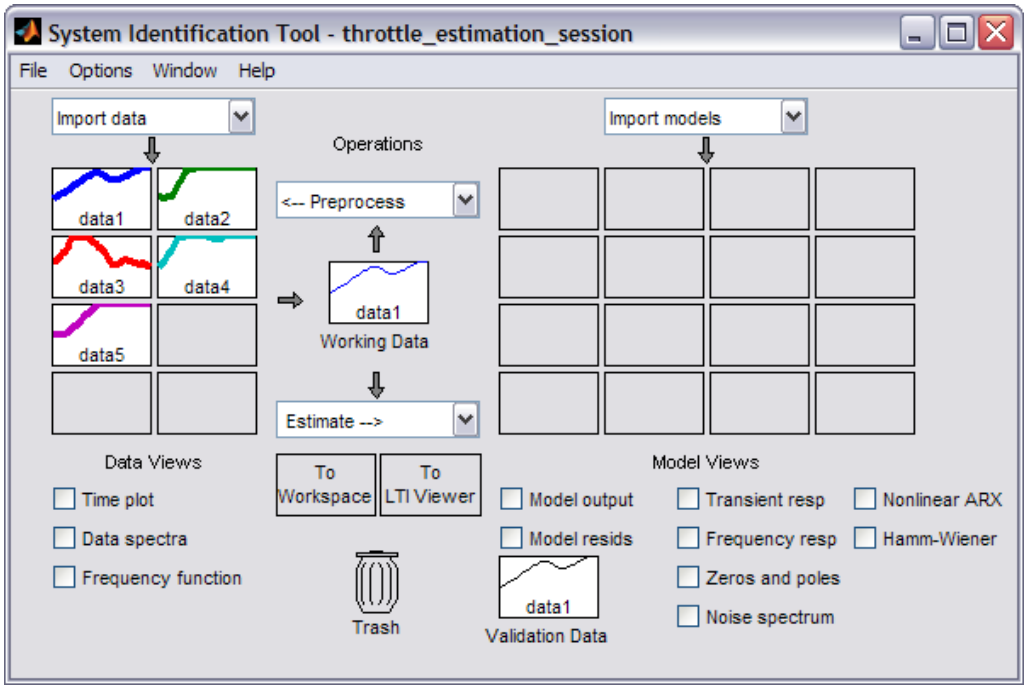


FIGURE 12: GUI WITH IMPORTED DATA SETS

After importing designate one dataset as Working Data (data set on which estimation will be performed) and another one as Validation Data (data test to be used for checking the quality of model). It is instructive to initially use the same data set for estimation and validation. This way, you can quickly check how good a particular estimated model fits the estimation data. Once you find a model that performs reasonably well on estimation data, you can test it with other data sets.

A System Identification session with all the data sets (as well as all the results generated using subsequent sections) has already been created. The user can load the following session as the starting point: throttle_estimation_session.sid.

ESTIMATING LINEAR MODELS

We begin by estimating linear models. It is recommended to try linear ones before nonlinear models. This exercise reveals the nature of the system, gives you an idea of order of the model (number of regressors etc) and the linear model can often be reused as a component of the nonlinear model. For linear model estimation we need a data set that does not excite the nonlinear behavior (the hard stop at 90 degrees) of the system significantly. The closest such data is data3. However, for zero inputs value, the output signal still shows a 15 degree offset; a linear model would not be able to capture this offset. So we first subtract this offset from the output data and then import it (subtraction of arbitrary offsets must be done in MATLAB Command Window before importing; you can't do this in GUI). The data set with no offset is called data3lin (see Figure 13 where its plot is compared against the original data). We set this data as Working Data and Validation Data.

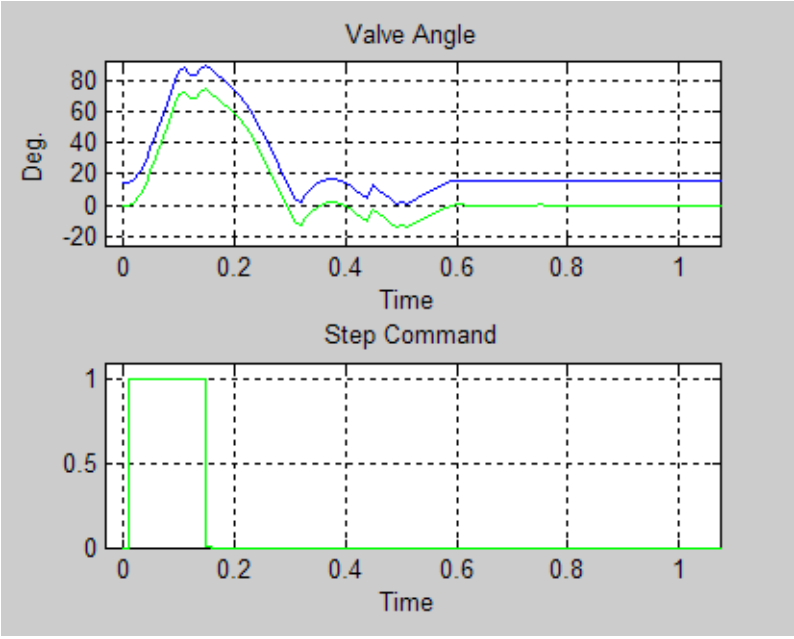


FIGURE 13: DATA FOR LINEAR MODEL ESTIMATION (GREEN)

To choose linear model structure, use the "Linear parametric models " option from the Estimation popup menu:

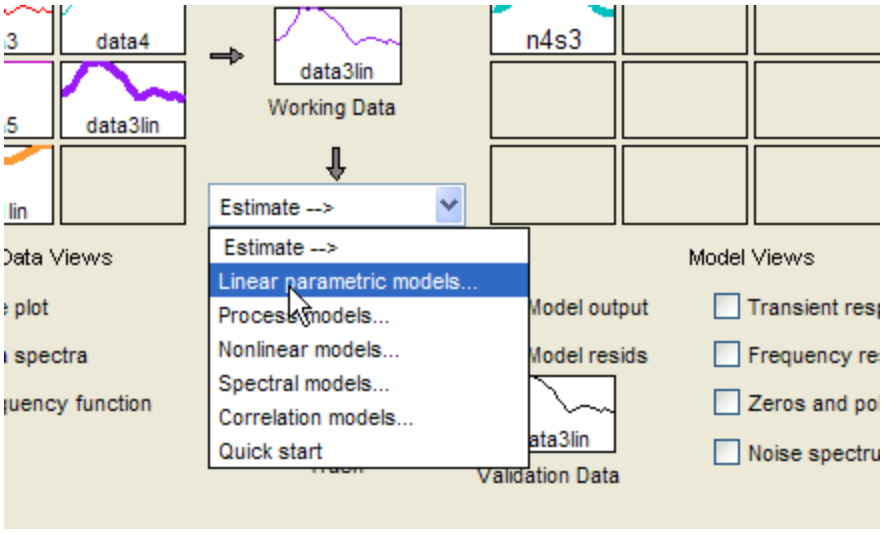


FIGURE 14: OPENING LINEAR MODEL ESTIMATION DIALOG

In the resulting dialog, choose various model structures such as ARX, OE and State space. Set the Focus to Simulation (since our goal is to maximize fit to measured throttle angle data). For each structure, try various orders starting with small numbers. For ARX and State-space structure, you also have the option of searching for best orders automatically. A quick exercise shows that model orders in the vicinity of 2 give good results. Thus a transfer function with 2 poles (OE model with orders [2 2 0], ARX model with na = 2, nb= 1 or 2, nk = 0 or 1) or a state-space model with 2 states work well.

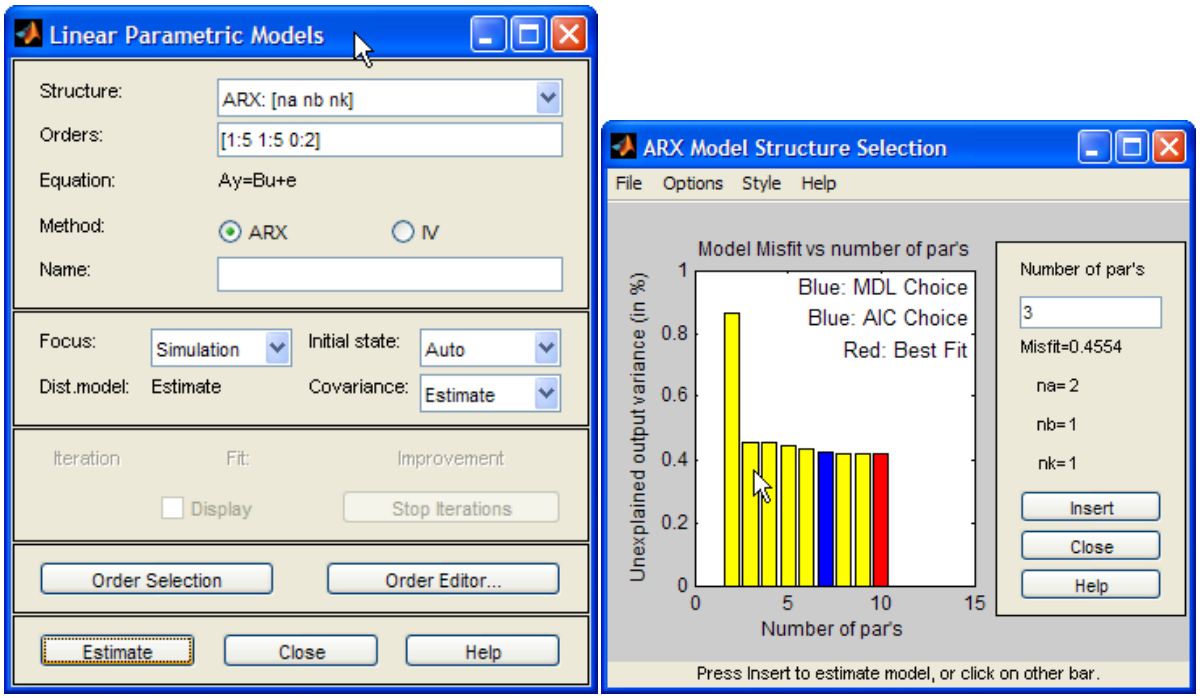


FIGURE 15: ORDER SELECTION FOR ARX MODELS; ORDER [2 1 1] REPRESENTED BY SECOND BAR FROM LEFT SEEM TO A GOOD COMPROMISE.

The simulated response of each model can be compared against the measured output value by clicking on the Model output checkbox in the GUI. This creates a plot showing the simulated responses of all models superimposed over measured output values, along with the corresponding fit values in percent. The % fit indicates the agreement between the model response and the measured output: 100% means a perfect fit, and 0% indicates a poor fit (0% means that the model output has the same fit to the measured output as the mean of the measured output; the fit values can be negative if the model does worse than fitting a straight line to data!).

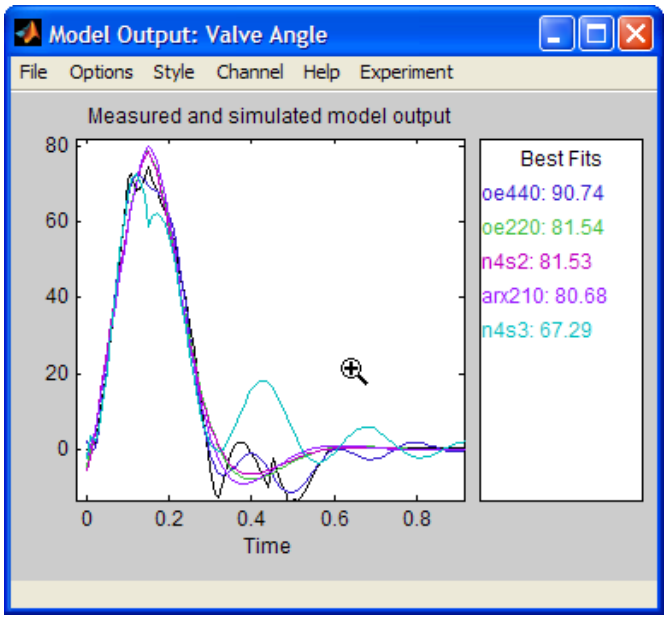


FIGURE 16: RESPONSE OF VARIOUS LINEAR MODELS COMPARED TO ESTIMATION DATA

If we replace the Validation data with some other data set (say, data2), we find that the linear model is unable to emulate the measured output. This is not surprising since data2 exhibits nonlinear behavior:

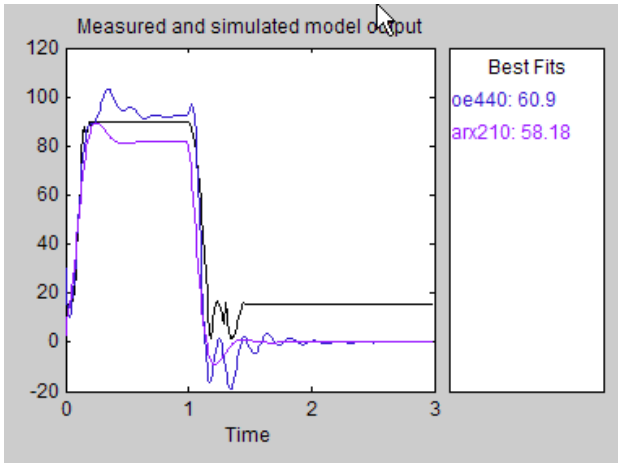


FIGURE 17: VALIDATING MODELS AGAINST DATA2

Note that in the linear region of the operation of the system, when the throttle angle is between 15 and 90 degrees, the linear model is still able to capture the response. However, the linear models do

not know how to describe the hard stop nonlinearity at 90 degrees or the non-zero offset (15 degrees) when no input signal is present. Hence for those values, the models' responses does not match the measured value.

Note that even though we did not use the notion of regressors when creating the linear models, the regressors do exist. For example, the regressors used by the ARX model arx210 are y(t-1), y(t-2) and u(t), where y refers to the output signal (throttle angle) and u to the input (step command operating the DC motor that moves the valve).

This concludes our preliminary exercise with linear model estimation. We will use the knowledge of the model orders from this exercise as well as the linear models themselves to create various nonlinear models.

ESTIMATING NONLINEAR ARX MODELS

Going from linear to nonlinear models, the simplest form we can try is one that describes the output as a linear combination of its regressors plus an offset. To begin with, the regressors can be same as those for the linear ARX model above (the model arx210). Hence the only aspect in which this model is different from the linear one is that it is able to capture the signal offsets. To do this, we launch the "Nonlinear Models" estimation dialog from the Estimate popup menu of the GUI:

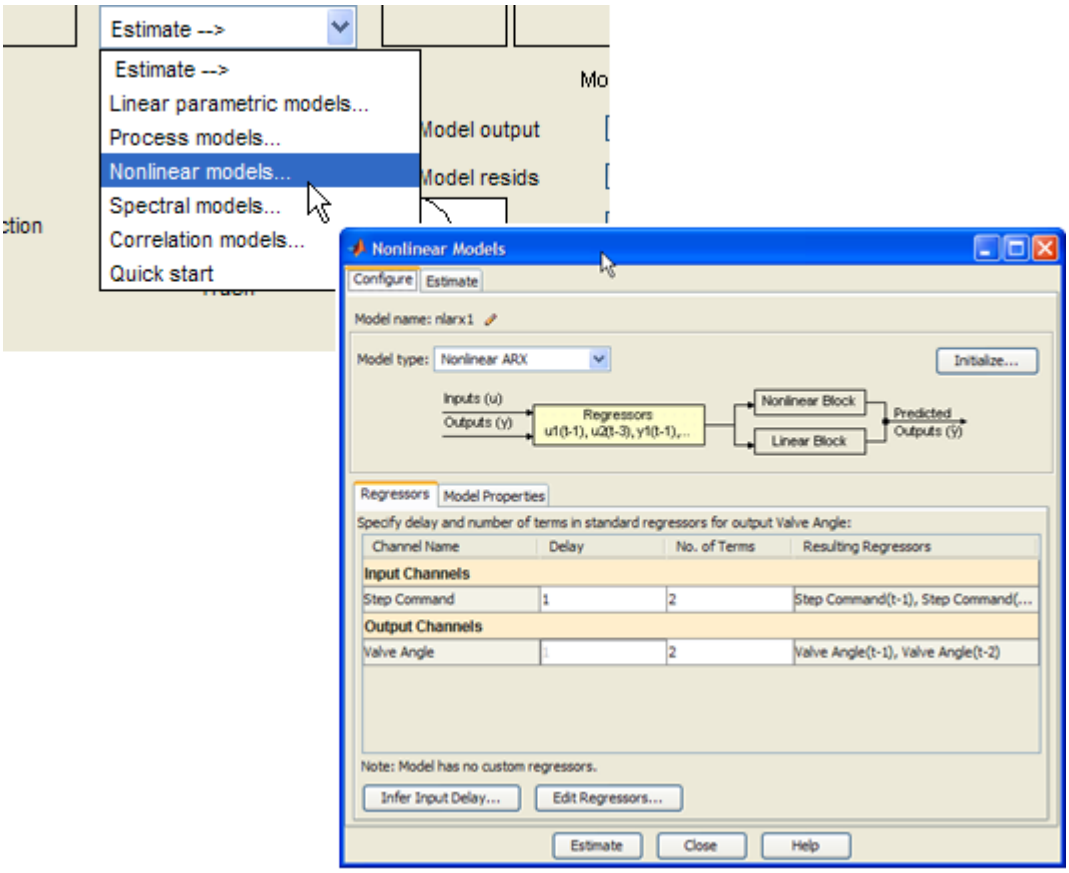


FIGURE 18: LAUNCHING INTERFACE FOR NONLINEAR MODELS ESTIMATION

This dialog allows us to create Nonlinear ARX and Hammerstein-Wiener models. In its default configuration, the dialog allows creation of Nonlinear ARX models; you can change this using the

Model type popup menu. The structure of Nonlinear ARX models is shown by a block diagram at the top; the model computes regressors using input and output data and then uses the regressors in a parallel set of linear and nonlinear functions to compute the response of the model. The tabs labeled "Regressors" and "Model Properties" beneath the diagram allow configuration of regressors (types, how many etc), and the nature of the linear and nonlinear functions (sigmoid function, wavelet function etc, the properties of chosen function etc), respectively.

To achieve a form similar to the linear ARX model arx210, we do the following:

1. Set the model orders in the regressor tab to [2 1 0], that is, set number of output regressors to 2, the number of input regressors to 1 and input delay to 0.
2. Under the Model Properties tab, choose the nonlinearity to be "none". This removes the nonlinear block from the model structure, so that the equation for output y(t) as a function of its regressors becomes:

   $$y(t) = \alpha_1 y(t-1) + \alpha_2 y(t-2) + \beta u(t) + d$$

   where y is the output (valve angle), u is the input (step command) and $\alpha_1, \alpha_2, \beta$, and d are the parameters to be estimated. d represents the output offset that the linear models could not account for. The model diagram in the GUI shows that the nonlinear function is not active.

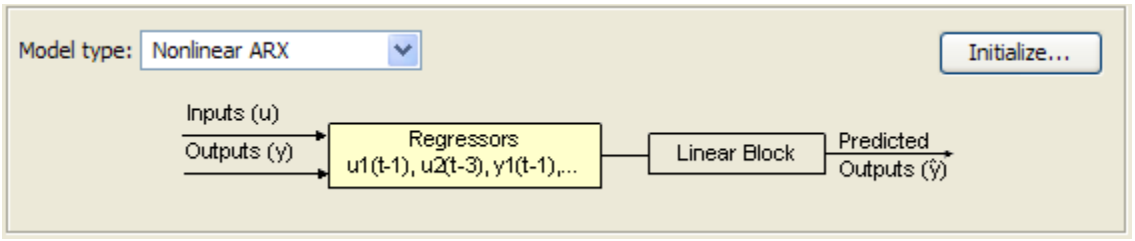


FIGURE 19: STRUCTURE OF A NONLINEAR ARX MODEL CONTAINING NO NONLINEAR FUNCTION

To perform estimation that minimizes simulation error, click on the "Algorithm Options…" button on the Estimate tab and set the Focus to "simulation".

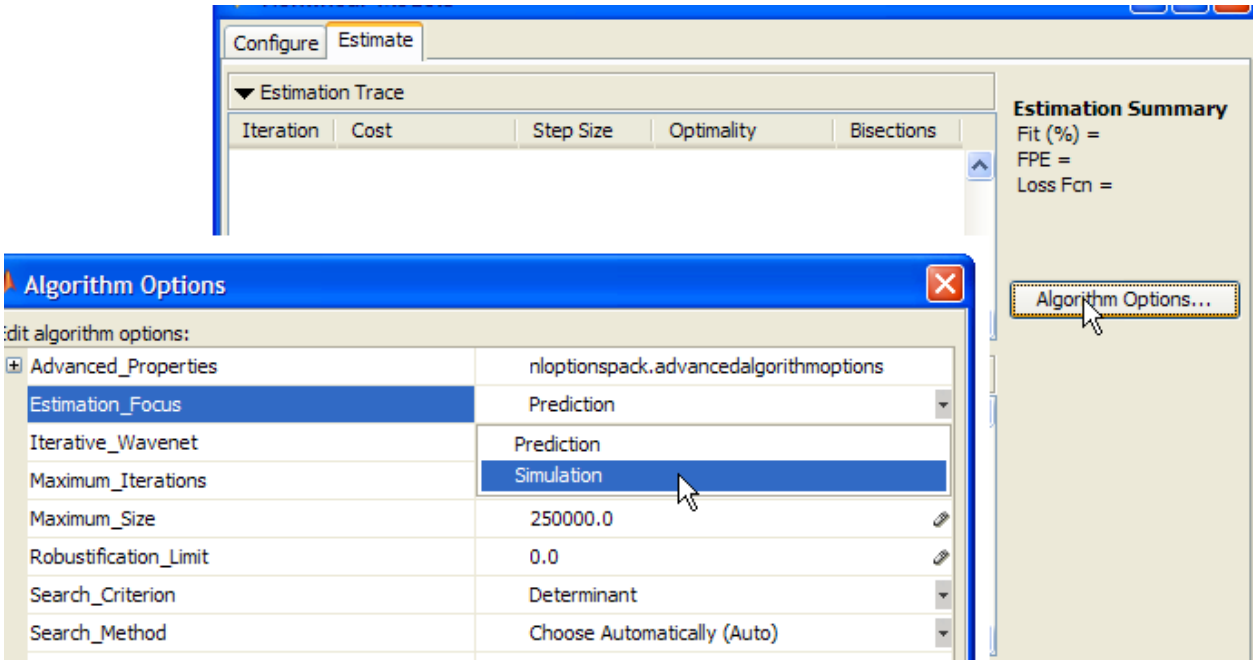


FIGURE 20: SETTING ESTIMATION ALGORITHM OPTIONS

We now set data3 as Working Data as well as Validation Data, and perform estimation by pressing the Estimate button. The resulting model ("nlarx1") behaves well on this data containing offsets. The linear model arx210, by contrast does not perform well when data contains offsets:

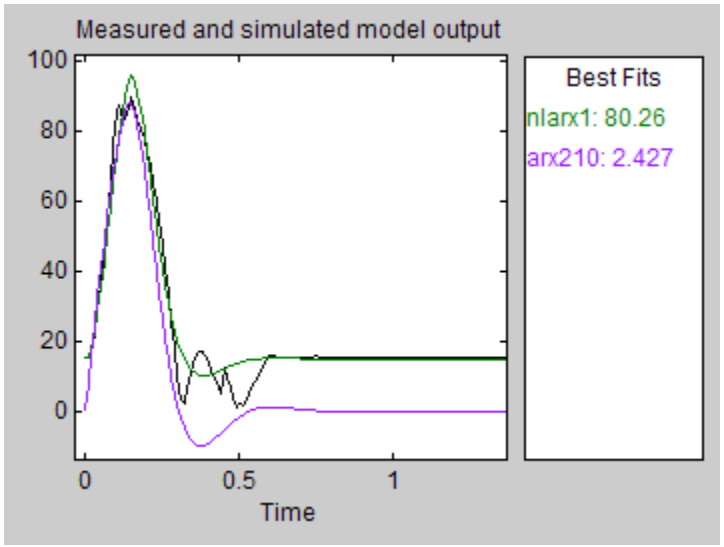


FIGURE 21: COMPARING RESPONSES OF MODELS TO MEASURED RESPONSE IN DATA3; NOTE THAT THESE MODELS WERE ESITMATED USING DATA3 TOO

However, if you drag the any other data set, such as data1 into the Validation Data box, you will quickly see that the fits are not good. Thus our simple nonlinear model that uses a linear combination of simple regressors (the regressors are simply time delayed I/O variables) plus offset is not powerful enough to capture the nonlinear effects:

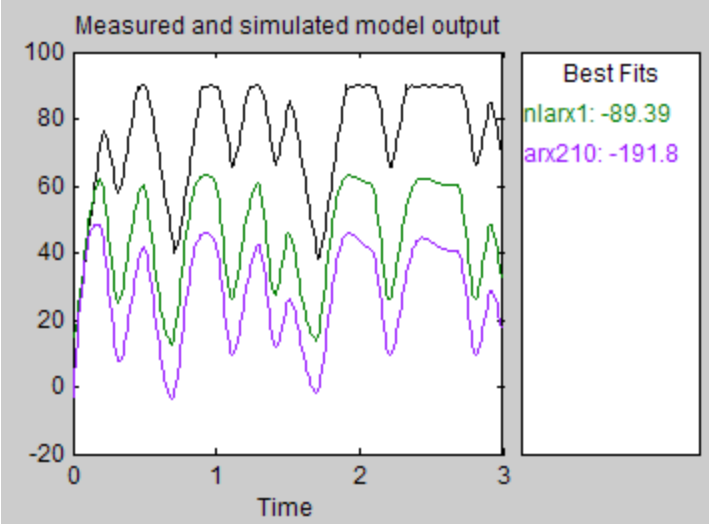


FIGURE 22: VALIDATING MODELS CREATED USING DATA3 AGAINST DATA SET # 1 (DATA1)

Hence we need more complex forms. In the section "UNDERSTANDING NONLINEAR MODEL STRUCTURES" we saw how nonlinear ARX models can be considered as logical extensions of linear ones. What kind of extensions could we try? We will try two approaches:

1. Adding more complex regressors motivated by physical insight.
2. Trying various nonlinear functions in a trial-and-error approach.

For creating our own *custom* regressor, we examine the physics governing this system. The overall system seems to be behaving like a second order system (at least in linear range). When the throttle angle becomes 90%, there is a hard stop which can be described by a very hard spring that resists the throttle angle to be greater than 90 degrees. In addition, the offset is 15 degrees, which is always there in absence of inputs. This can similarly be captured using a spring force that is activated only when the angle becomes less than 15 degrees. Thus the nonlinear resistive force, which comes into play only when the throttle angle is either less than 15 or greater than 90 degrees can be expressed as: $F_{nl}(t) = K_1*(\max(y(t),90)-90) + K_2*(\min(y(t),15)-15)$, where y(t) is the valve angle at time t. For simplicity we can assume $K_1=K_2$, so that the nonlinear spring force becomes: $K_1*(\max(y(t), 90) + \min(y(t),15)-105)$. We can include the formula "max(y(t), 90) + min(y(t),15)-105" as a regressor in the model and the job of the estimation method will be determine the value of K1, in addition to previous coefficients. This can be achieved as follows:

(a) Click on the Regressors tab under the Configure tab of the Nonlinear Models dialog. Then click on Edit Regressors button. This launches a dialog called Model Regressors that lets us view and edit the model's regressors.
(b) Expand the Custom Regressors tab in this dialog and click on the Create… button:

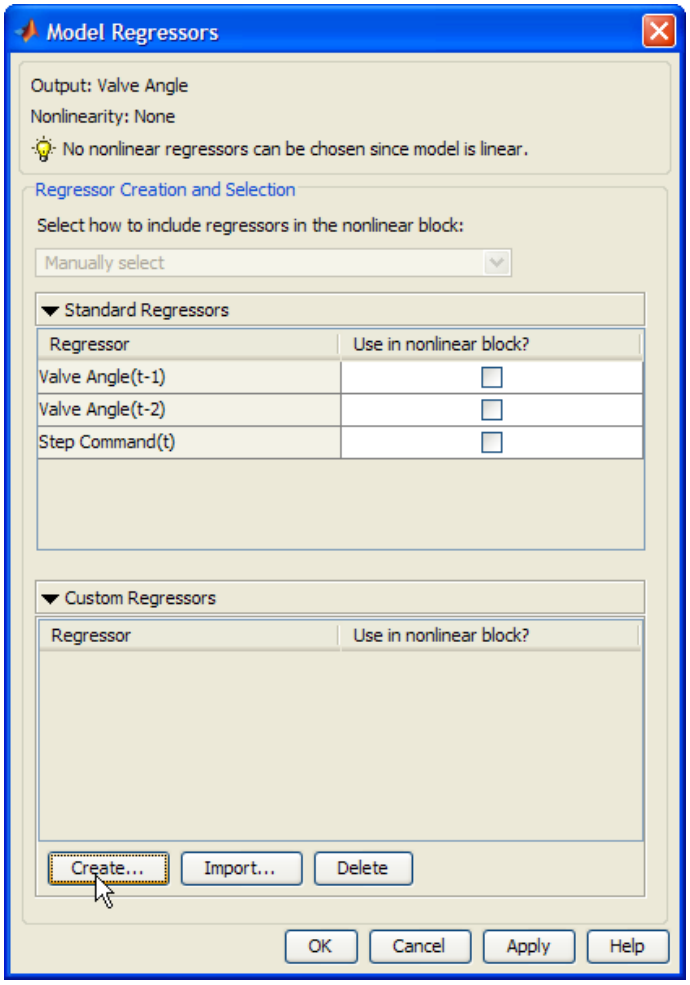


FIGURE 23: MODEL REGRESSORS DIALOG. YOU CAN CREATE CUSTOM REGRESSORS FROM HERE

(c) In the resulting custom regressor dialog, enter the expression "max(Valve Angle(t-1),90)+min(Valve Angle(t-1),15)-105". Then press the "Add" button. This adds a custom regressor using the above formula to the model. A delay of 1 sample in the above formula results from discretization effects. Typically, the discrete lag would not be suggested clearly by physical considerations and you should try out a few different values. Note that the lag in output variables has to be at least 1 to maintain causality.

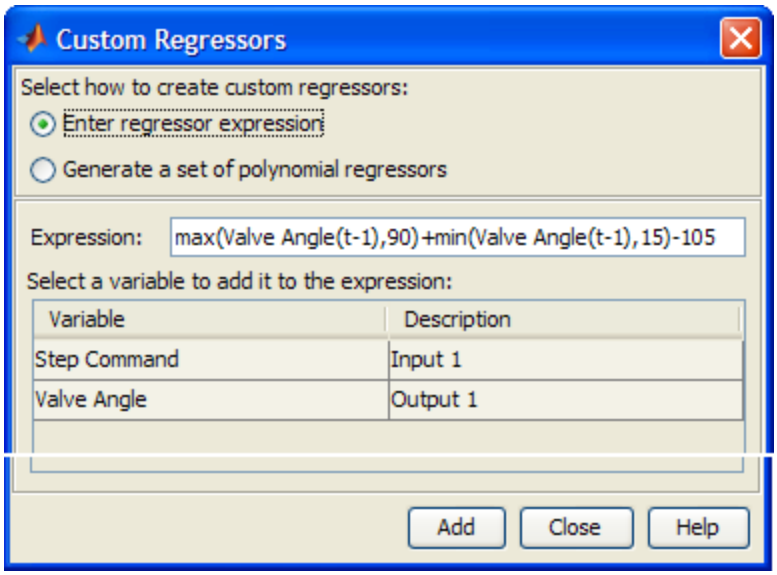


FIGURE 24: DIALOG THAT CREATES CUSTOM REGRESSORS.

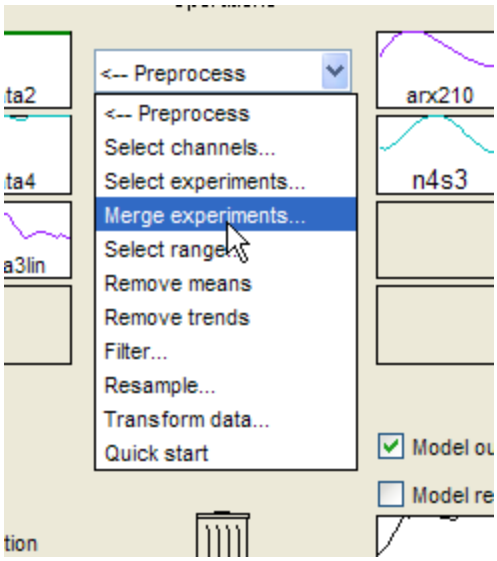
(d) Close the Custom Regressors Dialog. Click OK on Model Regressors dialog to accept changes to regressor list and close it as well.

The model is now configured to use a linear-in-regressors formula (no nonlinear functions yet) containing a custom-made regressor and output offset, described by the following formula:

$$y(t) = \alpha_1\, y(t\text{-}1) + \alpha_2\, y(t\text{-}2) + \alpha_3\, (\max(y(t\text{-}1), 90) + \min(y(t\text{-}1),15)\text{-}105) + \beta\, u(t) + d$$

The term in green is the new addition compared to previous model. The parameters of this model are α_1, α_2, α_3, β, and d. In order to estimate this model, we will now use data that excites the nonlinear behavior more strongly. The time plots of data1 and data2 suggest that they may be good candidates. So for the subsequent estimations, we will use a combination of these two data sets. The other data sets (data3, data4, data5) will be used for final validation of the quality of the models.

In order to use multiple data sets for estimation, we must first combine them. To do so, choose the Merge experiments … option from the Preprocess popup of the main GUI:



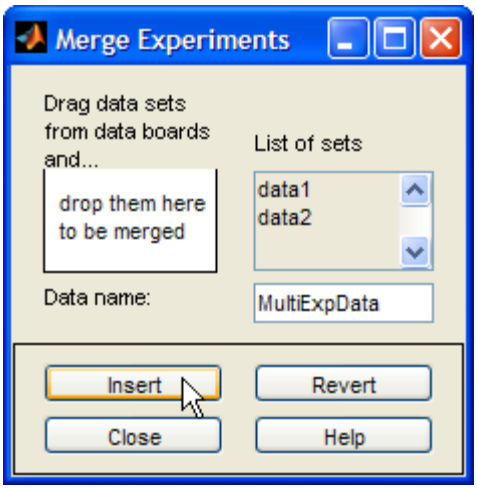


FIGURE 25: MERGING DATA EXPERIMENTS

Drag and drop the data icons data1 and data2 into the Merge Experiments drop area, set the name to MultiExpData and then hit Insert. This causes a new data icon named "MultiExpData" to appear in the data board of the GUI. Designate this data as both estimation and validation data by dragging its icon onto the Working Data and Validation Data boxes.

Click on Estimate button in the Nonlinear Models window. The resulting model "nlarx2" has good fit to both data sets contained in MultiExpData.

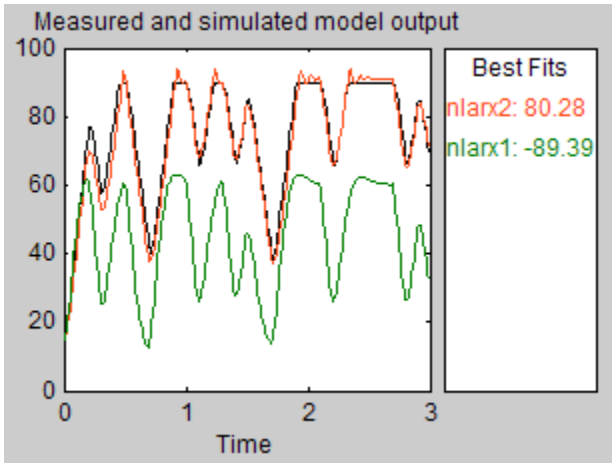


FIGURE 26: FIT TO ESTIMATION DATA (EXP. 1)

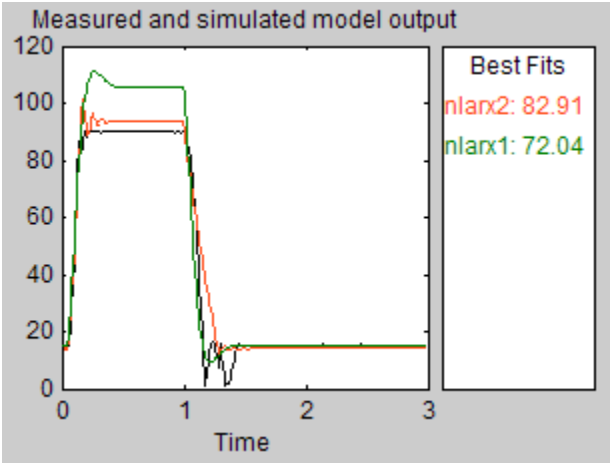


FIGURE 27: FIT TO ESTIMATION DATA (EXP. 2)

Furthermore, it validates reasonably well with data4 and data5, as shown in figures 28 and 29.

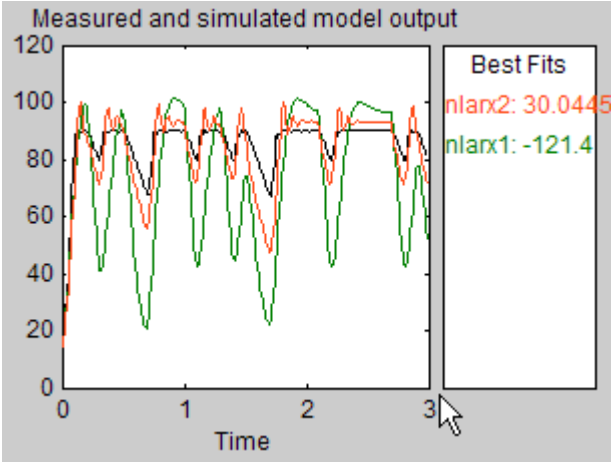


FIGURE 28: FIT TO VALIDATION DATA4

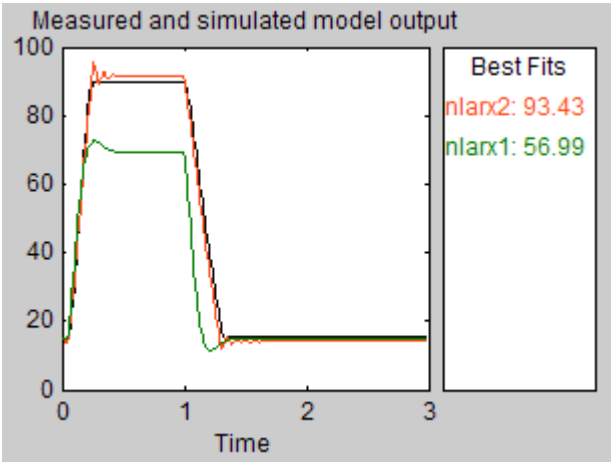


FIGURE 29: FIT TO VALIDATION DATA45

Note that in this document, we are only showing some typical results; it is entirely possible that you could get better results by tweaking the regressor formula or using different model orders . Also note that coming up with "best possible" results requires several trials and you might also have to play with estimation algorithm options, such as choice of search method, maximum number of iterations allowed, convergence tolerance etc.

Examination of model output plots reveals that where there are hard stops (angle = 90 degrees), the model's response shows oscillations which can be small or large. Close to points of transition to the nonlinear behavior (angle going from <90 to 90, or from >15 to 15 degrees), the fit is not very good. The model seems to be lacking sufficient flexibility to capture such fast transitional effects.

Could the fits be improved further? So far, we have only tried linear combination of regressors. So another logical generalization is to use nonlinear functions of regressors. System Identification Toolbox offers several choices for nonlinear functions such as binary tree (called "tree partition" in the product), wavelet network, sigmoid network and neural network. Most of these forms are a series expansion of some "unit function" such as a wavelet ($\exp(-|x|^2)$), sigmoid function ($1/(\exp(-x)+1)$) etc. The approach we have to take is try them out in succession and use one that seems to give better results. Also, for each choice of nonoinear function, there are some configuration parameters that we can tweak, the most important one being the number of terms to be used in the series expansion. In the following, we only show the some of the results (including the best we got after some trial and error) to showcase our approach.

(a) In the Nonlinear Models dialog, select the Configure tab and then click on the "Model Properties" tab. Set the Nonlinearity to Wavelet Network.
(b) Then click on the Regressors tab and click on the Edit Regressors… button to launch the Model Regressors dialog. If the regressors listed in the two tables in there are not already selected, select them all. This can be done quickly by choosing the "All" option from the topmost popup menu on this dialog. Finally, the dialog should look like this:

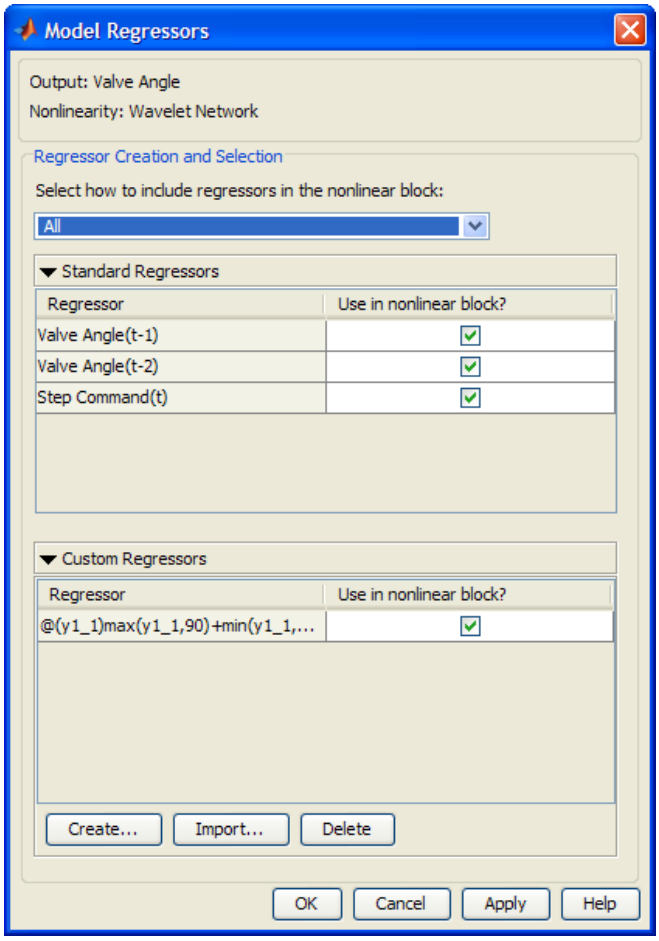


FIGURE 30: REGRESSOR DIALOG WITH ALL REGRESSORS SELECTED

(c) Click OK to accept changes in Regressor dialog. The model configuration is now complete. At this point, the model's structure is:

$$y(t) = \alpha_1 y(t-1) + \alpha_2 y(t-2) + \alpha_3 r(t-1) + \beta u(t) + d + f(y(t-1), y(t-2), u(t), r(t-1), \theta)$$

where r(t) is custom regressor function = max(y(t), 90) + min(y(t),15)-105, and f() is a parameterized Wavelet Network (if you want to know what a wavelet network looks like, visit the product documentation, online at: http://www.mathworks.com/access/helpdesk/help/toolbox/ident/ref/wavenet.html). The parameters of this model are: α_1, α_2, α_3, β, d and those used by wavelet network – θ (θ is a vector of parameters, usually a long list).

Note that the custom regressor is still active. Also, the model uses both a linear function (weighted sum of regressors plus offset) and a nonlinear function (wavelet network) as its components. We can remove the linear term as well as the custom regressor if required (shown later in the context of a sigmoid network)

(d) Click on the Estimate tab, and select the "Algorithm Options". Set the Interative_Wavenet option to "On".

(e) Press the Estimate button on the Nonlinear Models dialog. This creates a new model ("nlarx3").

The model does not perform well as show in by fit to the first data set in MultiExpData:

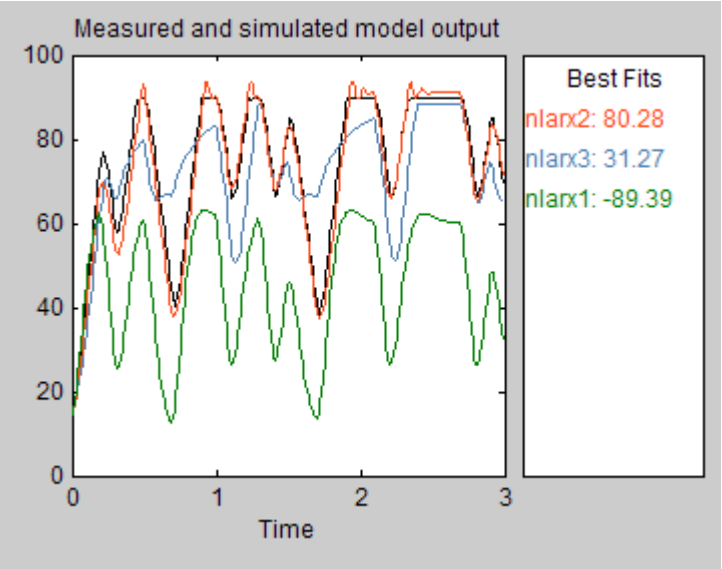


FIGURE 31: FIT TO ESTIMATION DATA (EXP. 1); MODEL NLARX3 DOES NOT FIT WELL.

It appears the wavelet network in unable to capture the underlying dynamics of the system. Next we try a Sigmoid Network:

(a) Change the Nonlinearity to Sigmoid Network
(b) Set the Number of units in nonlinear block to 5.
(c) Press the Estimate button.

This adds a new model called "nlarx4" to the model board of the main GUI. This model seems to be performing better than all other models tried before on estimation data, but not so on some of the validation data (especially data4). In the following figures, the fits to estimation and validation data sets is shown. Overall,use of Sigmoid Network looks more promising.

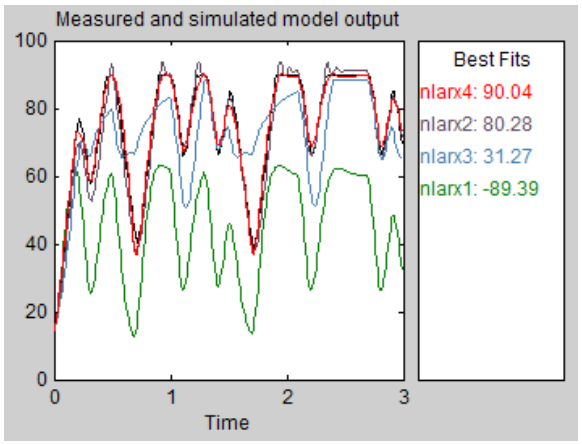


FIGURE 32: FIT TO ESTIMATION DATA (EXP. 1)

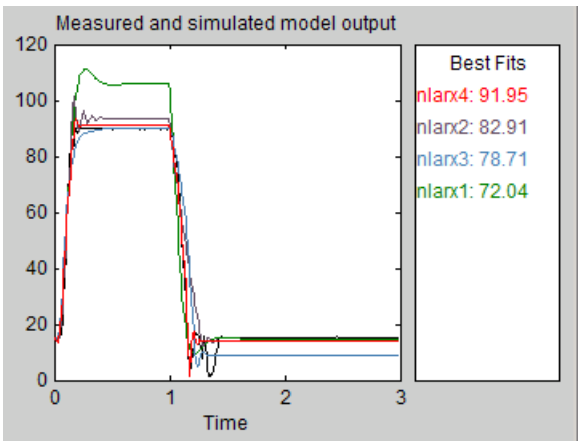


FIGURE 33: FIT TO ESTIMATION DATA (EXP. 2)

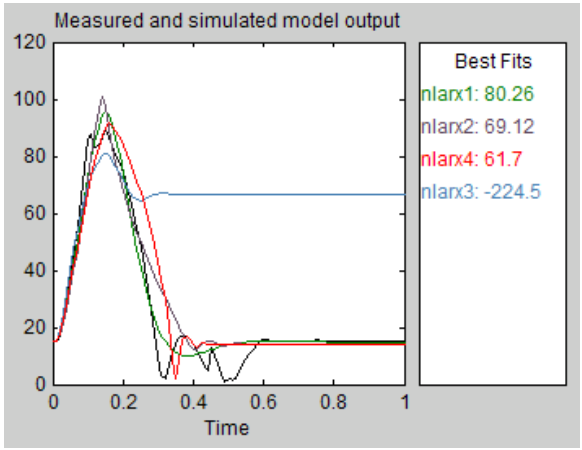


FIGURE 34: FIT TO DATA3

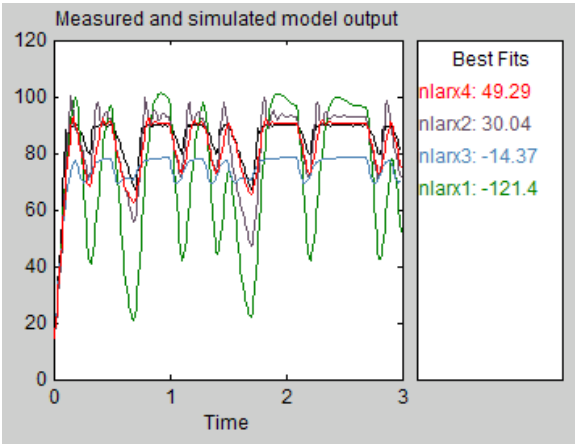


FIGURE 35: FIT TO DATA4

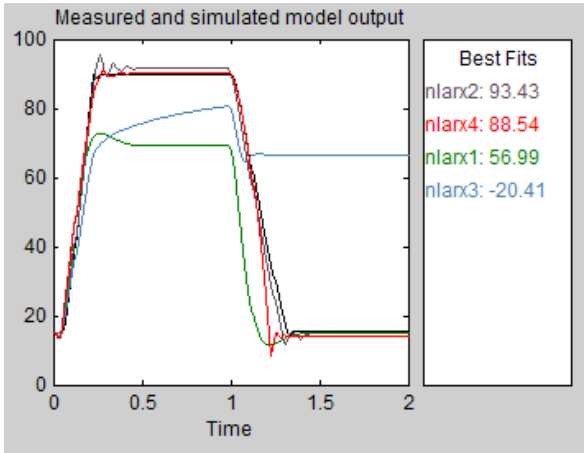


FIGURE 36: FIT TO DATA5

Next we explore if we can get even better results (nlarx4 did poorly on data4). After some trial and error, a model of order [3 3 0] and using no custom regressors with sigmoid network seems to work pretty well. This configuration can be achieved as follows:

(a) In the "Nonlinear Models" window, under Regressors tab, set Delay for input channel (Step Command) to zero. Set number of terms for both inut and output channels to 3.
(b) Cick on the "Edit Regressors…" button to launch the Model Regressors dialog. In the Model Regressors dialog, select the custom regressor we had added earlier, and delete it. We do this to investigate if using a nonlinear function with only the standard regressors (those that are simply the time-shifted I/O variables) is going to be sufficient. When we have a nonlinear function (sigmoid network) in the model, do we also need complex forms of regressors? Let us find out.
(c) Press OK to accept changes to model regressors and close the Model Regressors dialog.
(d) Set the Number of units in nonlinear block to 5.
(e) Under Algorithm options, set Maximum_Iterations to 90 (estimation progress in the Estimation Trace area showed that the error reduces slowly and may therefore require a large number of iterations). To do the large number of iterations, this run will take several minutes.
(f) Then press Estimate button to estimate the parameters of the model. This creates a model called "nlarx5". Note that the equation of motion for this model is:

$$y(t) = \alpha_1\, y(t-1) + \alpha_2\, y(t-2) + \beta\, u(t) + d + g(y(t-1), y(t-2), u(t), \theta)$$

where g() is a Sigmoid Network function (see: http://www.mathworks.com/access/helpdesk/help/toolbox/ident/ref/sigmoidnet.html). The parameters of this model are: $\alpha_1, \alpha_2, \alpha_3, \beta$, d and those used by sigmoid network – θ. Note that θ is not a single parameter; it represents a set of several ones used by the sigmoid network.

The model nlarx5 fits all the data sets quite well, as shown in figures below. This is the best model we have achieved after trying various types of configurations.

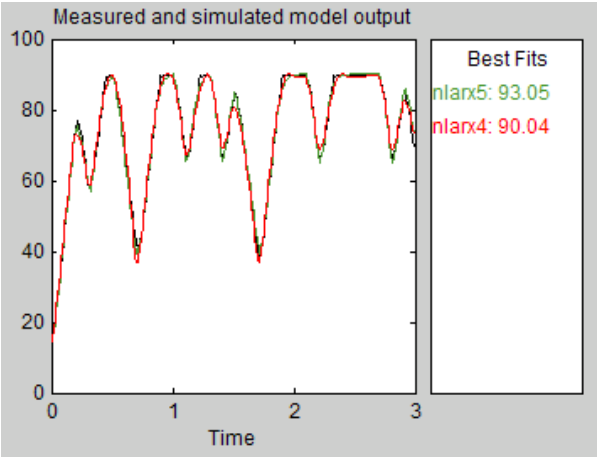


FIGURE 37: FIT TO ESTIMATION DATA (EXP. 1)

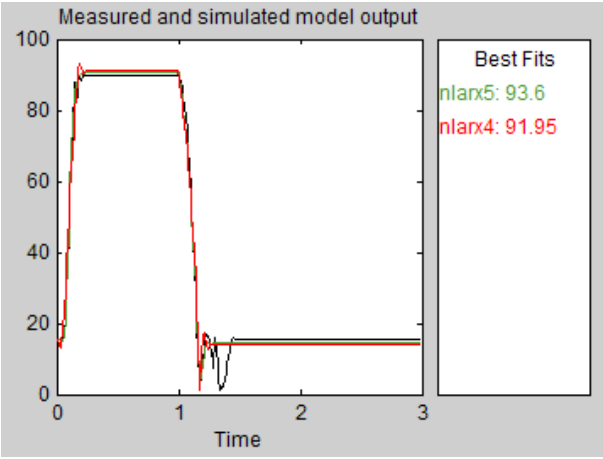


FIGURE 38: FIT TO ESTIMATION DATA (EXP. 2)

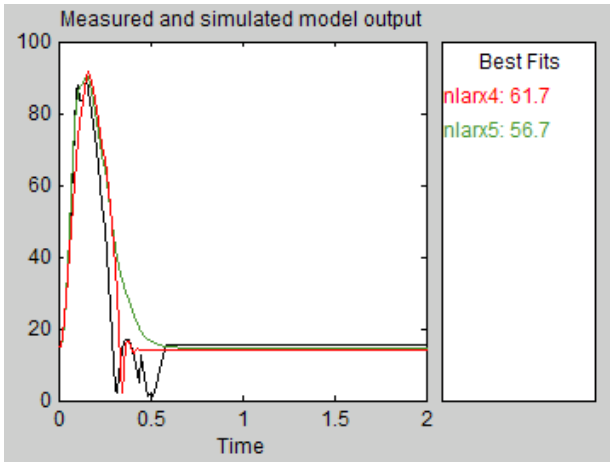


FIGURE 39: FIT TO DATA3

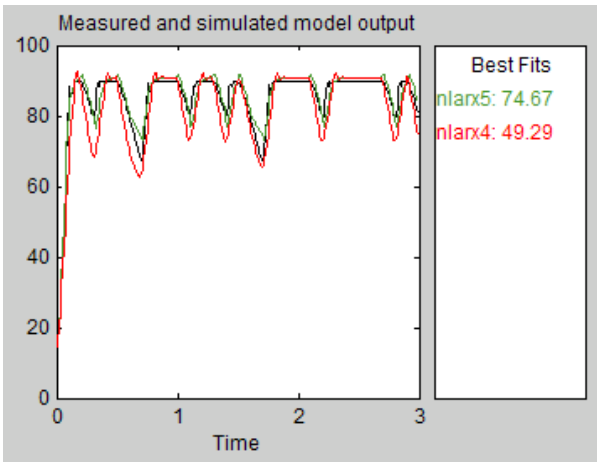


FIGURE 40: FIT TO DATA4

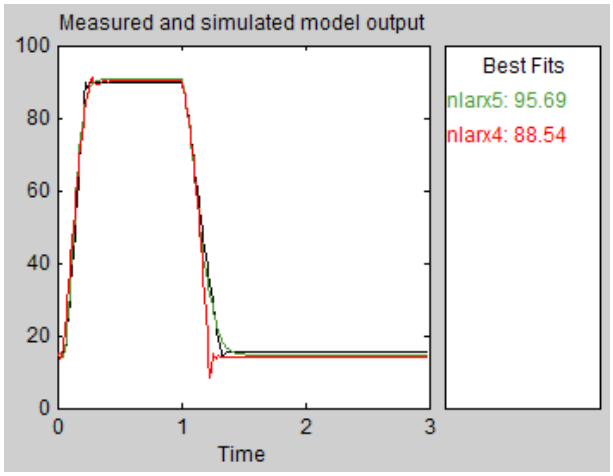


FIGURE 41: FIT TO DATA5

ESTIMATING HAMMERSTEIN-WIENER MODELS

Next, we explore the Hammerstein-Wiener class of models. As described earlier, these models are series connections of nonlinear functions with linear dynamic models.

The choice of nonlinear functions to use at the input and output port of the linear model can be made arbitrarily in the spirit of black box modeling, or can be guide by physical insight. We know intuitively that the system has a linear range. At angles less than 15 degrees and greater than 90 degrees resistive forces kick in. The ones at 90 degrees are strong and can be considered almost as a complete *hard stop*. Such phenomena can be described by a saturation function. Thus adding a saturation function at the output of the model should capture at least the 15 and 90 degree steady-state signal levels present in the output, if not the transient behavior. To implement such a configuration, we proceed as follows:

(a) In the Nonlinear Models dialog, select the Configure tab. Then select Hammerstein-Wiener as the Model Type using the popup menu above the diagram. You will see two tabs titled- “I/O Nonlinearity” and “Linear Block”.

(b) Select the Linear Block tab. Set the model orders as follows: B order: 3, F order: 3, Delay: 0; by some trail and error, this choice of orders seems to work better.

(c) In the I/O Nonlinearity tab, set the Input Nonlinearity to None. Set the Output Nonlinearity to Saturation. At this point the dialog looks like this:

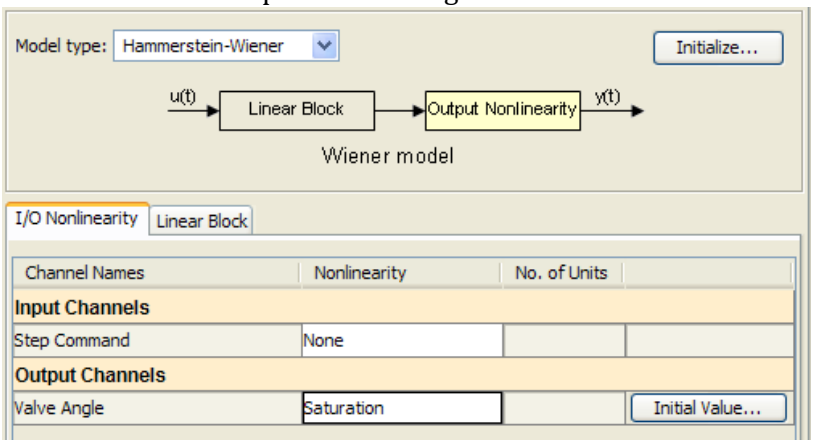


FIGURE 42: CONFIGURATION FOR WIENER MODEL ESTIMATION

(d) Click on the "Initial Value" button for setting the initial values of the saturation nonlinearity. This launches a dialog wherein you can specify initial guess values for the lower and upper limits of saturation. Set the lower and upper limits to 15 and 90 respectively (see Figure below)

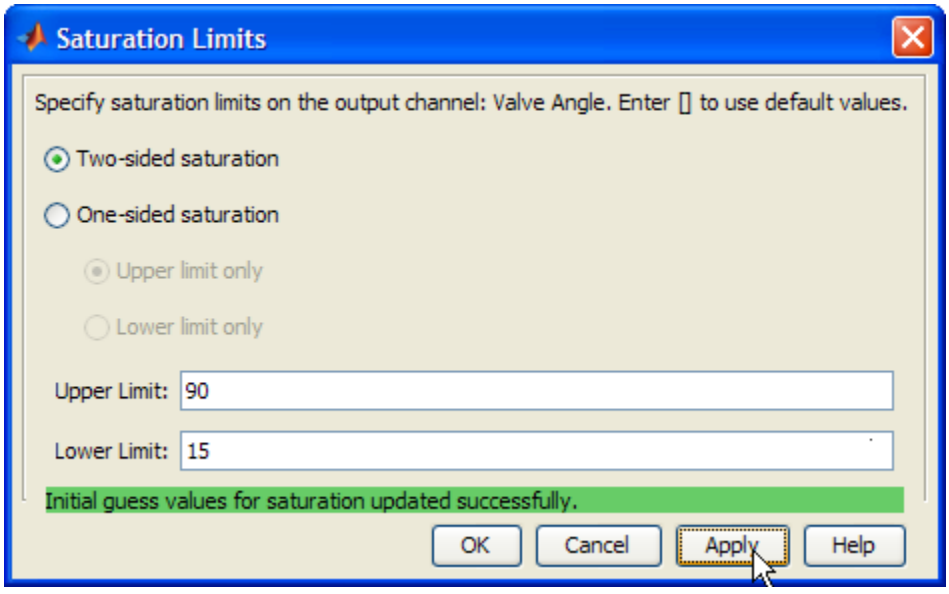


FIGURE 43: SPECIFYING SATURATION LIMITS

NOTE: you must set linear block orders before setting the initial values of the nonlinearity, otherwise those values may be reset.

(e) Open Algorithm Options dialog and set the "Initial_State" option to "Estimate". This will cause the initial conditions to be estimated together with model parameters. Estimating

initial conditions can sometimes have beneficial effect on accurate determination of model parameters.

(f) Click on the Estimate button.

This creates a new Hammerstein-Wiener model "nlhw1", an icon for which appears in the model board. Mathematically, this is just a Wiener model since the model has only output nonlinearity. The block diagram of this model is:

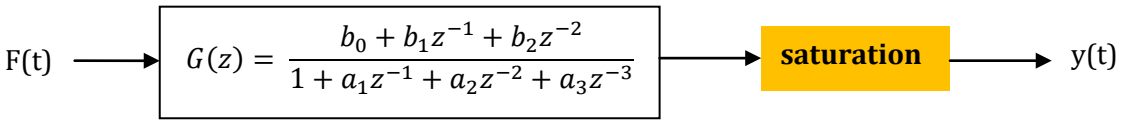


FIGURE 44: WIENER MODEL OF THROTTLE DYNAMICS

The comparison of the model's response to estimation data is as follows:

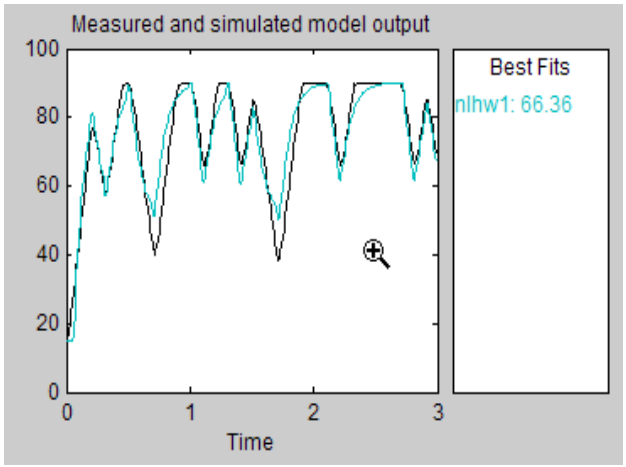


FIGURE 45: FIT TO ESTIMATION DATA (EXP. 1)

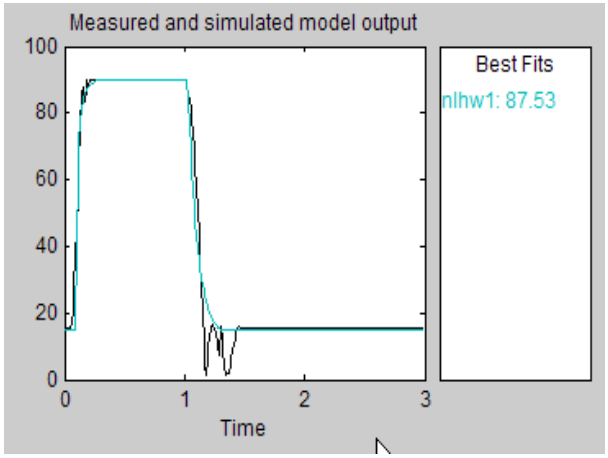


FIGURE 46: FIT TO ESTIMATION DATA (EXP. 2)

Furthermore, it validates reasonably well with data3 and data4, but not to Data5 as shown in Figures 47-49.

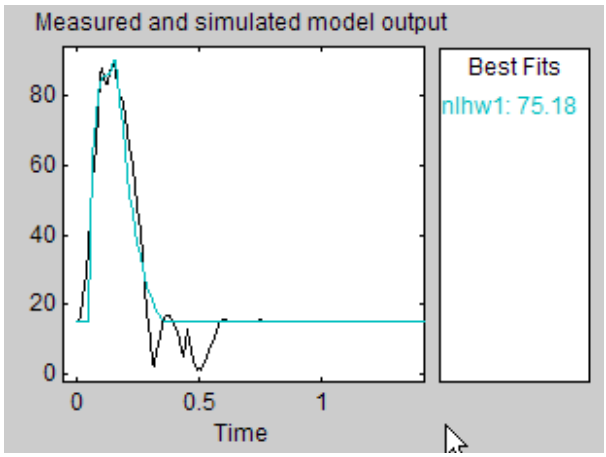


FIGURE 47: FIT TO DATA3

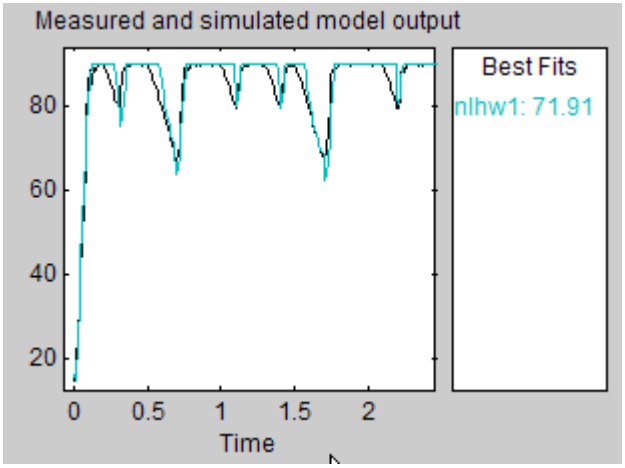


FIGURE 48: FIT TO DATA4

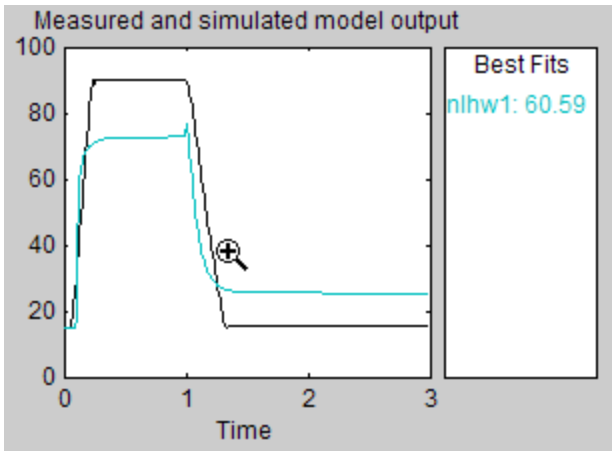
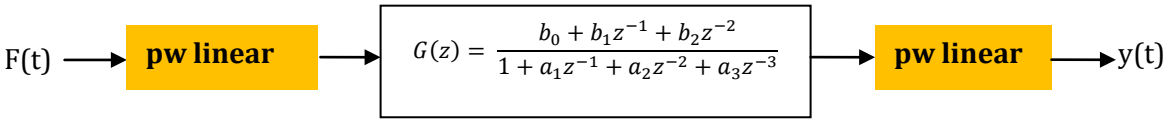


FIGURE 49: FIT TO DATA5

Can we do better? In the spirit of "black box" modeling, we shall use generic nonlinear functions for input/output nonlinearities. These nonlinear functions are not derived from physical insight. Some trial and error reveals that a model of order [nb, nf, nk] = [3 3 0] and piecewise linear functions for both input and output nonlinearities works quite well. This is how we configure this form:

1. Under the I/O Nonlinearity tab, set both Input and Output Nonlinearity to "Piecewise Linear".
2. Under Linear Block, set orders to 3, 3 and 0 respectively for B order, F order and Input Delay.
3. Set the Initial_State option under Algorithm Options to 'Zero'. It turns out keeping initial conditions fixed at zero leads to better results for this configuration.
4. Under Algorithm options, set Maximum_Iterations to 200.

The form of the model is as shown below:



Press the Estimate button to estimate the above model. This creates a model named "nlhw2" in the Model Board. The response of this model, compared to data and the previous model nlhw1 is show below. As observed, the model performs quite well for all the data sets.

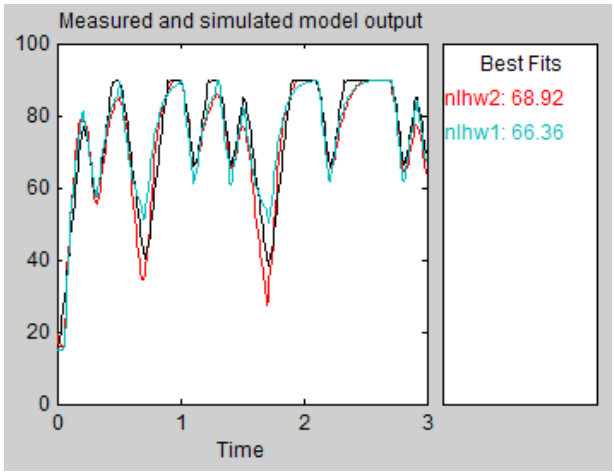


FIGURE 50: FIT TO ESTIMATION DATA (EXP. 1)

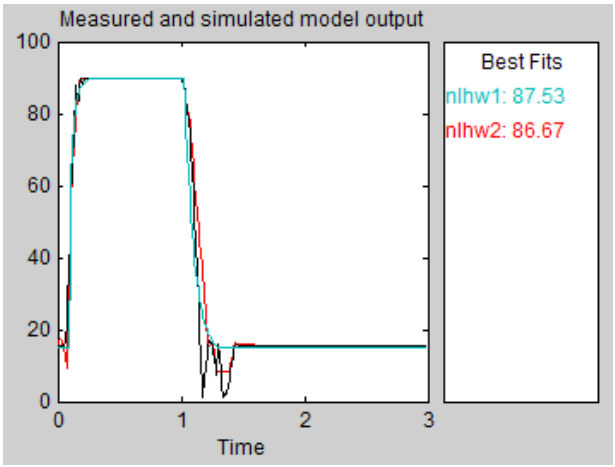


FIGURE 51: FIT TO ESTIMATION DATA (EXP. 2)

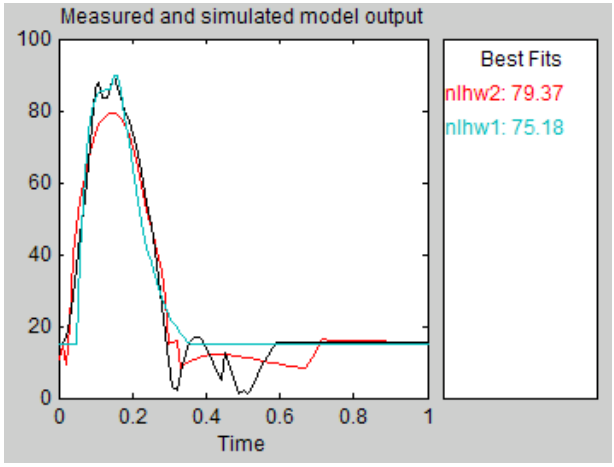


FIGURE 52: FIT TO DATA3

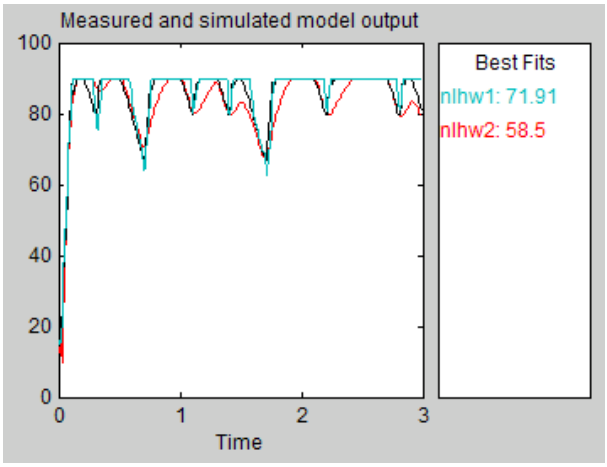


FIGURE 53: FIT TO DATA4

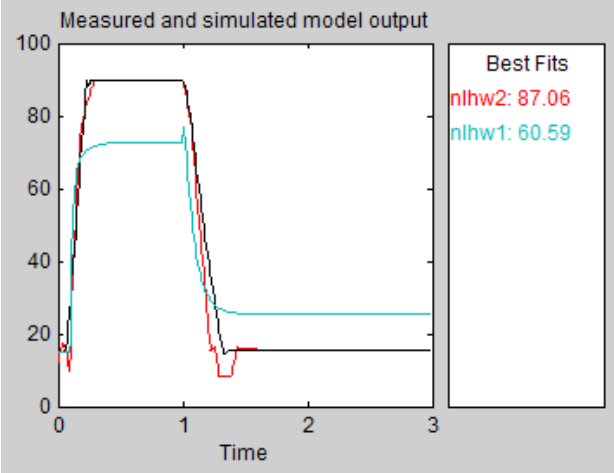


FIGURE 54: FIT TO DATA5

Model nlhw2 is more or less same as nlhw1 in its data fitting abilities, although on data5 it seems to provide better fit (90% vs. 60%).

This concludes our demonstration of some black box modeling approaches using System Identification Toolbox. This document described a GUI based approach to modeling. The same tasks can be in command line. The MATLAB file "throttledemo.m" shows how to create the same models without using the GUI. The essential elements you will need to be aware of are the following:

a) Data objects: data has to be packaged in an object called "iddata" when using System Identification Toolbox.
b) Estimation commands: nlarx, nlhw, train commands create nonlinear models from data
c) Nonlinear model objects: the models are represented by idnlarx, and idnlhw objects.

It was briefly mentioned at the end of section on linear model estimation that the linear models can be used directly as a component of the nonlinear model structure when estimating them. The GUI does not support this operation. However, in command line you can utilize the linear model as an input argument to estimation commands when creating nonlinear models. This is shown in command line demo throttledemo.m.

SUMMARIZING BLACK BOX MODELING APPROACH

The following things should be kept in mind:

1. Getting model from data is a systematic, multi-step process. You rarely get good models right away by throwing data at estimation routines. Especially for nonlinear modeling, make sure that the system has been sufficiently excited in the region of operation and that the data has sufficient number of samples.

2. When choosing a model structure, always begin with simpler forms such as low-order linear models. Use model order selection facility available with linear state-space and ARX models. This will quickly give you a feel for the complexity of the system (order of underlying dynamics) and whether you need nonlinear models.
3. Some special items to pay attention to:
    a. Dead time: the input to output delay is a quantity that can have significant effect on a model's quality. For most model structures, this delay is not estimated but needs to be specified in advanced (such as using "nk" in Nonlinear ARX models). Fortunately, delays can often be determined separately from physical considerations. Before starting with nonlinear modeling, try to ascertain of the system has large delays (larger than the data sample time); if so, try to determine its value accurately.
    b. Feedback: many times models do not give good results because the data corresponds to a system operating under feedback. If removal of feedback is possible, it could simplify your modeling approach. Otherwise, when using data containing output feedback, you have to pay attention to the model structures you use: when using linear models, use the ones that have a sufficiently flexible noise component. Do not use output-error type of models, or trust non-parametric modeling techniques.
    c. Outliers: If data contains missing values or outliers, it is best to remove them by segmenting data into several experiments and using segments free from these issues. Some other ways to handle missing data in System Identification Toolbox are:
        i. "Fill-in" missing values, by designating them as unknown parameters (see "misdata").
        ii. There is an estimation algorithm option called "LimitError" that can be set to a non-zero value (say, 1.6), which reduces the influence of outliers on model's parameter estimation.
4. To a large extent, the process of selecting a model structure, configuring its order and components (such as regressors/nonlinear function, or number of neurons/layers) and estimating its parameters is an iterative, trial/error approach. Be ready to try a variety of forms.
5. Always validate your models using independent data sets. For nonlinear models, your model may not validate if the validate data set corresponds to a different operating point or radically different input signal levels. With nonlinear models it is difficult to obtain a universal model that fits all types of data (unless configured based on physical intuition).
6. For modeling of processes that span across several operational regions, you may have to use a cluster of models with a way of combining or witching among them; that approach could work better than trying a create a single black box model directly from data.

GREY BOX MODELING APPROACH

Those who are familiar with the throttle valve device may not find it too difficult to derive an exact relationship between the command input and the valve angle purely from physical considerations. In fact, we used some of those considerations to arrive at the formula for custom regressor in the Nonlinear ARX modeling approach. The ability to write explicit equations for the system leads us to a slightly different modeling approach wherein the structure of the model is fixed, but only the

values of its parameters are computed using test data. To study this approach, let us investigate the dynamics of throttle valve system more closely.

The DC motor operating the valve responds very quickly to the step command. Hence compared to the throttle valve, the dynamics of the motor can be captured simply as a steady-state gain. The butterfly valve behaves like a mass-spring-damper system (second order), except for the hard stops. The hard stops can be described by a resistive hard spring. Hence the overall dynamics can be represented by a second order differential equation that uses a nonlinear resistive torsional spring:

$$J\ddot{y} + c\dot{y} + ky + K(\max(y, 90) - 90 + \min(y, 15) - 15) = bF(t)$$

where J is the rotational inertia of the throttle valve, c is its damping coefficient, k is its torsional spring constant (linear stiffness) and K is the stiffness constant related to the nonlinear resistive force. The constant b describes the system gain originating from the DC motor that transforms the step command input a real power signal that moves the throttle valve. The parameters of this model are the coefficients J, c, k, K and b. We can reduce one parameter by dividing the whole equation by J. Hence in the following discussions, assume J=1, or equivalently, that the constants c, k, K and b are parameters whose values have been normalized by that of J. The equation of motion is now:

$$\ddot{y} + c\dot{y} + ky + K(\max(y, 90) - 90 + \min(y, 15) - 15) = bF(t)$$

Now we move on to the task of estimating the values of the (normalized) constants *c*, *k*, *K* and *b* using test data. There are two ways of doing so:

(a) Simulink® centric approach: If Simulink is your preferred modeling platform, you can create the above system as a block diagram that uses some integrator, gain and saturation blocks. Then Simulink Design Optimization™ can be used to calculate the values of the unknown constants that fit chosen data set(s). This approach is discussed in the demo: http://www.mathworks.com/products/sl-design-optimization/demos.html?file=/products/demos/shipping/sldo/spe_engine_throttle.html

(b) MATLAB® centric approach: Write a MATLAB function (similar to an ODE file to be used for solving differential equations, or an objective file used in Optimization Toolbox for minimization)that represents the above equation: the function should return the derivative and output values as a function of model's parameters and current input values. This MATLAB function would look something like this:

```
function [dx, y] = throttleODE(t, x, F, c, k, K, b)
% ODE function for throttle body dynamics.
% Represent equation of motion by a set of first order equations (state-space)
%
%  dx: state derivatives at time t
%  y: output at time t
%
%  t: time value (scalar)
%  x: state vector at time t
%  F: input (step command) at time t
%  c, k, K, b: parameters to be estimated

g = max(90,x(1))-90+min(x(1),15)-15;
```

```
% State equations
dx(1) = x(2);
dx(2) = b*F - c*x(2) - k*x(1) - K*g;

% Output equation
y = x(1);

end
```

Use this function (throttleODE.m) to create what is called a "Nonlinear Grey Box" model of System Identification Toolbox. The parameters of this model can be estimated to fit chosen data set(s). This approach is discussed in the demo throttledemo.m. This demo shows that getting good parameter values is contingent upon using good initial guesses, which may not be known. Hence some trial and error is required for trying out various initial guesses.

The main difference between the grey-box approach and the black box modeling approach is that the grey box approach requires explicit knowledge of the system's dynamics. When you perform grey-box modeling, you must first create an exact mathematical representation of you system (in either MATLAB or Simulink) and *also prescribe initial guess values to all parameters*. Black box approaches, on the other hand, do not require initial guess values of model's parameters. Read the grey box modeling section of command line demo throttledemo.m for more details.

The grey-box approach is not facilitated by the GUI, since it requires you to write some code by hand. However, estimated models can be imported into the GUI so that they may be analyzed and compared against other models.

CONCLUDING REMARKS ON MODELING

In the preceding sections, we discussed several modeling approaches. In the end, we must choose one of these models as a representation of the throttle system. The obvious criterion for choosing a model is how well it fits the estimation and validation data sets. However, we also need to keep some other factors in mind:

1. Model's simplicity: A simpler model should be preferred over a model complex one, if their performances are in the same ballpark.
2. Not all models may be amenable to automatic code generation. See the section on "Code Generation" later on in this document.

In the following sections, we discuss how to make use of these models. That involves fetching the model's parameters, simulating the model, linearizing it, importing it as a block in Simulink and generating code for deployment on a target.

If you have created models using a GUI (System Identification Tool or Neural Network Tool), you should first export the desired model(s) from the GUI to MATLAB workspace.

VIEWING MODEL PARAMETERS

Many times in the description above, we showed the underlying equation of a model and the parameters it uses. It may be natural to ask what values were assigned to those parameters, how

can one fetch them? This information may be required if you are going to implement the model equation "by hand", such as in C for a hardware deployment. In the following we discuss how to query identified models for their parameters.

PARAMETERS OF A NONLINEAR ARX MODEL

A nonlinear ARX model created using System Identification Toolbox is represented by a dedicated object called "IDNLARX" object. This object contains the list of regressors used by the model and parameters of the linear and/or nonlinear function employed by the model.

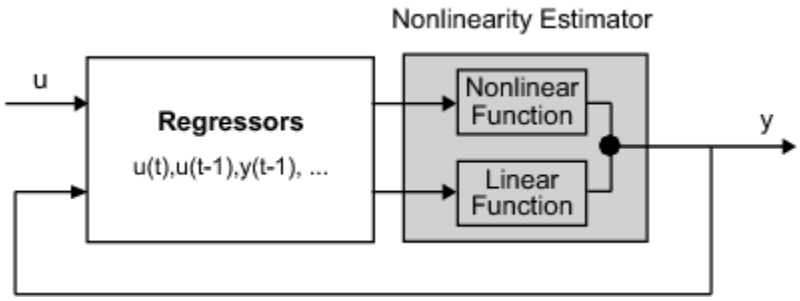


FIGURE 55: STRUCTURE OF A NONLINEAR ARX MODEL

1. Model regressors:
    a. The standard regressors, which are time-shifted I/O variables, are stored implicitly as model orders – na, nb and nk. For example, if na=2, nb=1 and nk=0, the regressors indicated by these orders are: y(t-1), y(t-2), u(t)
    b. Any custom regressors you specify manually (such as "max(y(t), 90) + min(y(t),15)-105" we used to capture hard stops at 15 and 90 degrees) are listed under a property called "CustomRegressors".
    c. To view a unified list of all regressors used by a model, run the "getreg" command, as in "getreg(nlarx2)". You will see the mathematical expressions for all the regressors printed in a list. For example getreg(nlarx1) shows the following:

```
Regressors:

Valve Angle(t-1)

Valve Angle(t-2)

Step Command(t)
```

2. Nonlinear function parameters: Model has a property called "Nonlinearity". This property contains all the parameters and settings associated with the linear and nonlinear functions employed by the model. For example, consider the model "nlarx4" which uses a sigmoid network with additional linear terms to transform regressors into output:

$$y(t) = \alpha_1\, y(t-1) + \alpha_2\, y(t-2) + \beta\, u(t) + d + g(y(t-1), y(t-2), u(t), \theta)$$

where g() is a single-layer sigmoid network expressed as a weighted sum of translated and dilated sigmoid functions. The parameters α_1, α_2 and β are related to the linear function, d is

an offset ("bias" in neural network terminology) and the set θ denoted all the coefficients associated with the sigmoid network. The values of all of these parameters are available under the "Nonlinearity" property. Typing "nlarx4.Nonlinearity" shows the following:

```
Sigmoid Network:
    NumberOfUnits: 5
      LinearTerm: 'on'
      Parameters: [1x1 struct]
```

NumberOfUnits refers to the number of terms in the series expansion for sigmoid network (number of "neurons" in neural network terminology). LinearTerm = 'on' means that the model uses the linear component (which is α_1 y(t-1)+ α_2 y(t-2) + β u(t)). "Parameters" is a structure containing all the parameter values. Typing "nlarx4.Nonlinearity.Parameters" displays the following:

```
ans =
        RegressorMean: [57.0526 56.9563 0.3247 -0.2828]
    NonLinearSubspace: [4x4 double]
        LinearSubspace: [4x4 double]
             LinearCoef: [4x1 double]
                Dilation: [4x5 double]
            Translation: [1.4403 -2.9774 0.4151 1.4286 17.2277]
           OutputCoef: [5x1 double]
         OutputOffset: 56.7809
```

To make sense of the various field names, you must know the formula for a sigmoid network, which can be found in product documentation (e.g., type "doc sigmoidnet"). In particular, LinearCoef contains the parameters α_1, α_2 and β, while OutputOffset refers to parameter d in the equation above.

By knowing the mathematical form of the model and by fetching the values of its various parameters, we can write code manually to simulate this model. For more details on querying the model and computing response by hand, see the documentation topic:

http://www.mathworks.com/access/helpdesk/help/toolbox/ident/ug/bq5o_xw-1.html#br7139m

PARAMETERS OF A HAMMERSTEIN-WIENER MODEL

As described before, a Hammerstein-Wiener model uses a series connection of input and/or output nonlinear functions with a dynamic linear system:

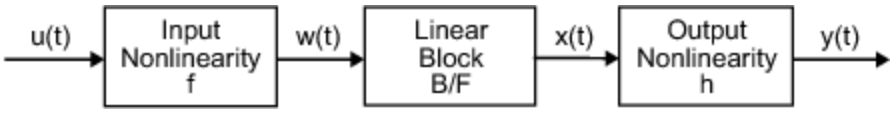


FIGURE 56: STRUCTURE OF A HAMMERSTEIN-WIENER MODEL

The model is represented by an "IDNLHW" object. This object stores the information of model configuration (types of nonlinear functions, order of linear model etc) and various parameter values as its properties.

The linear system is described by a state-space model (IDSS object) or a polynomial model (IDPOLY object). For single-input, single-output models, such as the case here, the linear model is represented by an IDPOLY object configured to describe a transfer function:

$$x(t) = B/F\ w(t)$$

where x(t) is the response of the linear block. The input (f) and output (h) nonlinear functions are similar to the nonlinear function used by a Nonlinear ARX model, and are represented by forms such as sigmoid network, wavelet network, piecewise linear function, saturation, deadzone etc.

1. Input nonlinear function (f): The settings and parameter values associated with the input nonlinearity is stored in the property "InputNonlinearity". For example, for the model nlhw1, typing "nlhw1.InputNonlinearity" displays:

   ```
   Unit Gain object (no property).
   ```

   The model nlhw1 has no input nonlinearity. This is achieved by setting the function to 1 (i.e., f(t) = 1). This configuration is represented in the model by a "unit gain" function. It has no properties associated with it.

2. Output nonlinear function (h): The settings and parameter values associated with the output nonlinearity is stored in the property "OutputNonlinearity". The format of the value of this property depends upon the form of the chosen nonlinear function. For example, for the model nlhw1, typing "nlhw1.OutputNonlinearity" displays:

   ```
   Saturation:
     LinearInterval: [15.0000 90.0000]
   ```

   The output nonlinearity is represented by a saturation function that clips the entering values below 15 or above 90. The mathematical expression for this function is h(t) = min(max(x,15), 90). The saturation function has only one property called "LinearInterval" that stores the lower and upper saturation limits. These values can be set manually or could be estimated automatically by running an estimation of the overall model.

3. Linear block: This is stored in the model property called "LinearModel". Typing "nlhw1.LinearModel" shows the equation for the linear block:

   ```
   Discrete-time IDPOLY model: y(t) = [B(q)/F(q)]u(t) + e(t)
   B(q) = -6.727 + 17.93 q^-1 - 11.2 q^-2

   F(q) = 1 - 2.405 q^-1 + 1.868 q^-2 - 0.4622 q^-3
   ```

Sampling interval: 0.01

The linear model coefficients B and F represent the numerator and denominator, respectively, of the linear (discrete-time) transfer function.

By knowing the mathematical form of the model and by fetching the values of its various parameters, we can write code manually to simulate this model. For more details on querying the model and computing response by hand, see the documentation topic:

http://www.mathworks.com/access/helpdesk/help/toolbox/ident/ug/bq2ix15.html#br73coo

PARAMETERS OF A NONLINEAR GREY BOX MODEL

The parameters of grey box models are explicitly defined. The display of model shows them. Their values can be fetched using "getpar" command. In the model object, they are stored in a property called "Parameters"; so typing "Model.Parameters" would return the list of parameters as a struct-array.

USING MODELS FOR SIMULATION AND ANALYSIS

A model is used for simulation and analysis. The nonlinear models created using System Identification Toolbox can be used in several ways:

- You can simulate them using given inputs in MATLAB (see "sim" command)
- You can linearize them about chosen operating points or input signals (see "linearize" and "linapp" commands); see also: http://www.mathworks.com/access/helpdesk/help/toolbox/ident/ug/brjukrq.html
- You can import them into Simulink using dedicated blocks. See System Identification Toolbox block library (*slident*) for models created using System Identification Toolbox, and Neural Network Toolbox block diagram generating function (*gensim*) and associated block library (*neural*) for neural network models.

SIMULATING IN MATLAB

The models created using the System Identification Tool can be simulated in the GUI itself using the Model output and Transient Response options. For other actions, you must export the model first to MATLAB workspace. You can do this by dragging the model icons from the model board of the GUI to its "To Workspace" box. Similarly, you can export data sets to the MATLAB base workspace. For example, after exporting "data1" and "nlarx4" to the base workspace, you can simulate the model's response to input signal in data1 as follows:

```
u = data1(:,[],:); % extract input signal

sim(nlarx4, u)     % simulate using zero initial conditions
```

Similarly, the data based models created using Neural Network Toolbox can be simulated using the "sim" command.

USING MODELS IN SIMULINK

Both System Identification Toolbox and Neural Network Toolbox provide blocks that allow you to incorporate estimated models into a Simulink model. This is useful when you have a Simulink model of a larger system, some of whose components are created using a data based modeling approach. For example, the throttle is a component of an engine. The model of an engine, represented by a large number of blocks and subsystems in Simulink, can use a block representing the throttle dynamics. System Identification Toolbox provides 4 blocks for representing identified models:

(a) Idmodel: block for identified linear models
(b) Nonlinear ARX Model: block for identified nonlinear ARX models
(c) Hammerstein-Wiener Model: block for identified Hammerstein-Wiener models

To simulate a model, simply drag its corresponding block from the block library into your model. For Neural Network models, the most convenient way is to use the "gensim" command that automatically creates a Simulink model for the network. However, the resulting model is difficult to simulate with time based input signals.

The following picture shows a Simulink model that simulates the responses of various identified models created using System Identification Toolbox. The models are simulated using the input from data set number 3 ("data3"). The responses of the models are overlaid on the measured response for same input. An offset of 15 degrees is added to the response of the linear model, since the linear model by itself does not account for 15 degree offset that exists in the response.

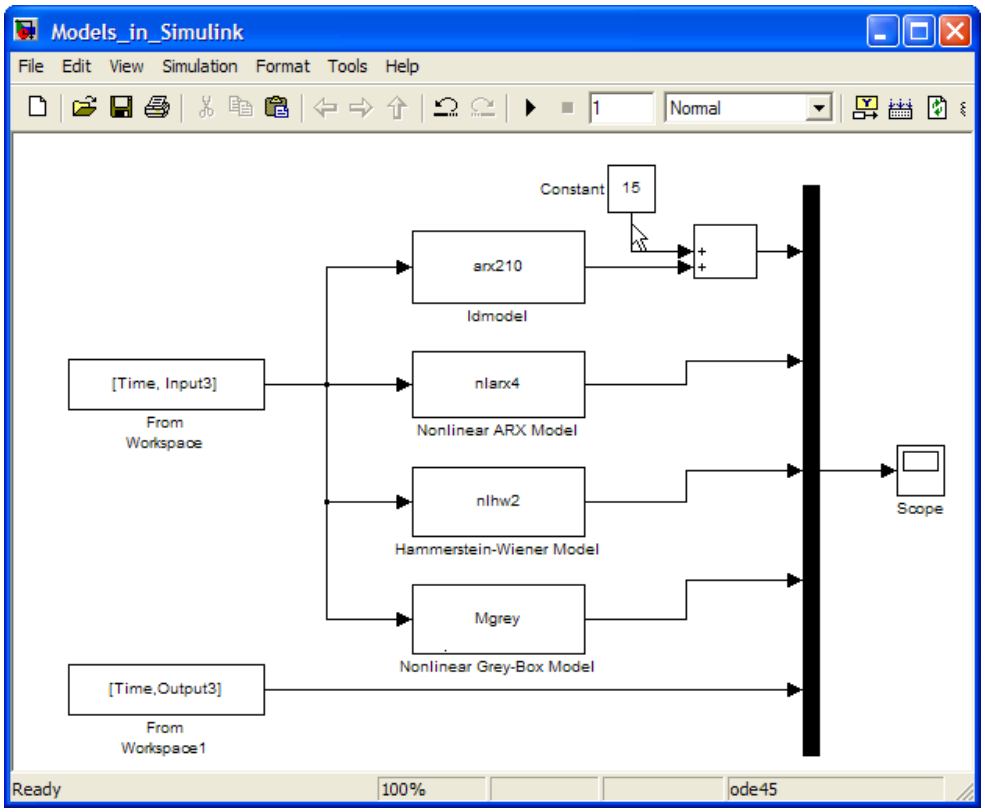


FIGURE 57: SIMULINK MODEL THAT USES THE IDENTIFIED MODELS

The responses in the Scope block for 1 second long simulation looks like this:

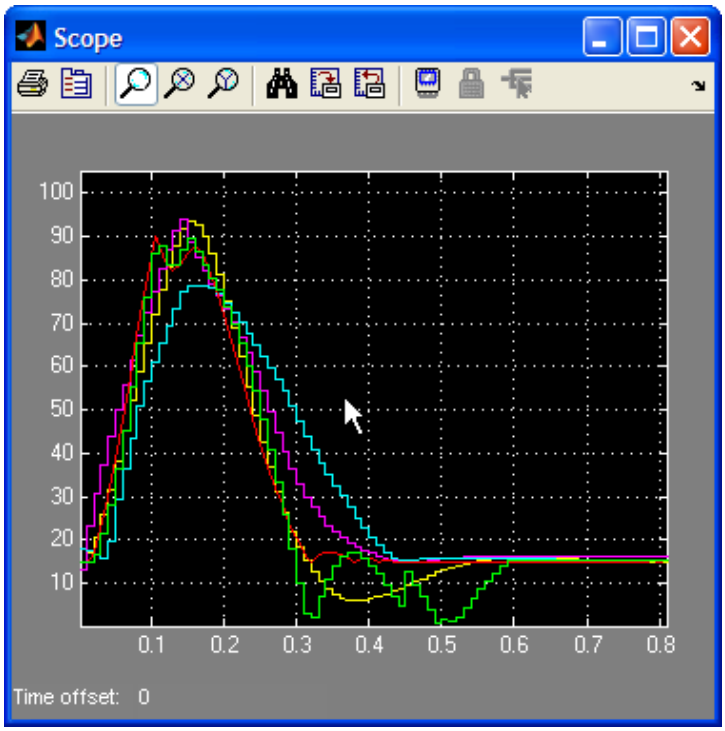


FIGURE 58: COMPARING MODEL RESPONSES TO MEASURED VALUES FOR A PARTICULAR INPUT SIGNAL

The green curve is for the measured response, while the others correspond to the responses of various models (yellow: linear model arx210, magenta: Nonlinear ARX model nlarx4, cyan: Hammerstein-Wiener model nlhw2 and red: Nonlinear grey-box model Mgrey).

Note: these responses may not match those seen in the GUI because of difference in initial conditions. In the GUI, the initial conditions are estimated to maximize the match to measured response. However, in implementation environment like Simulink, you should derive initial conditions from physical considerations - what was system doing before simulation started?

The block diagram structure of black box models might be of interest. If you look under the mask of the model blocks you can see how the input is transformed into outputs. For example, in case of a Nonlinear ARX model, the diagram under the hood shows how the output is computed in two stages: first the inputs and past output values are used to compute a set of regressors (the "UX2Reg" block) and then the regressors are transformed into output by the nonlinear function ("sigmoidnet").

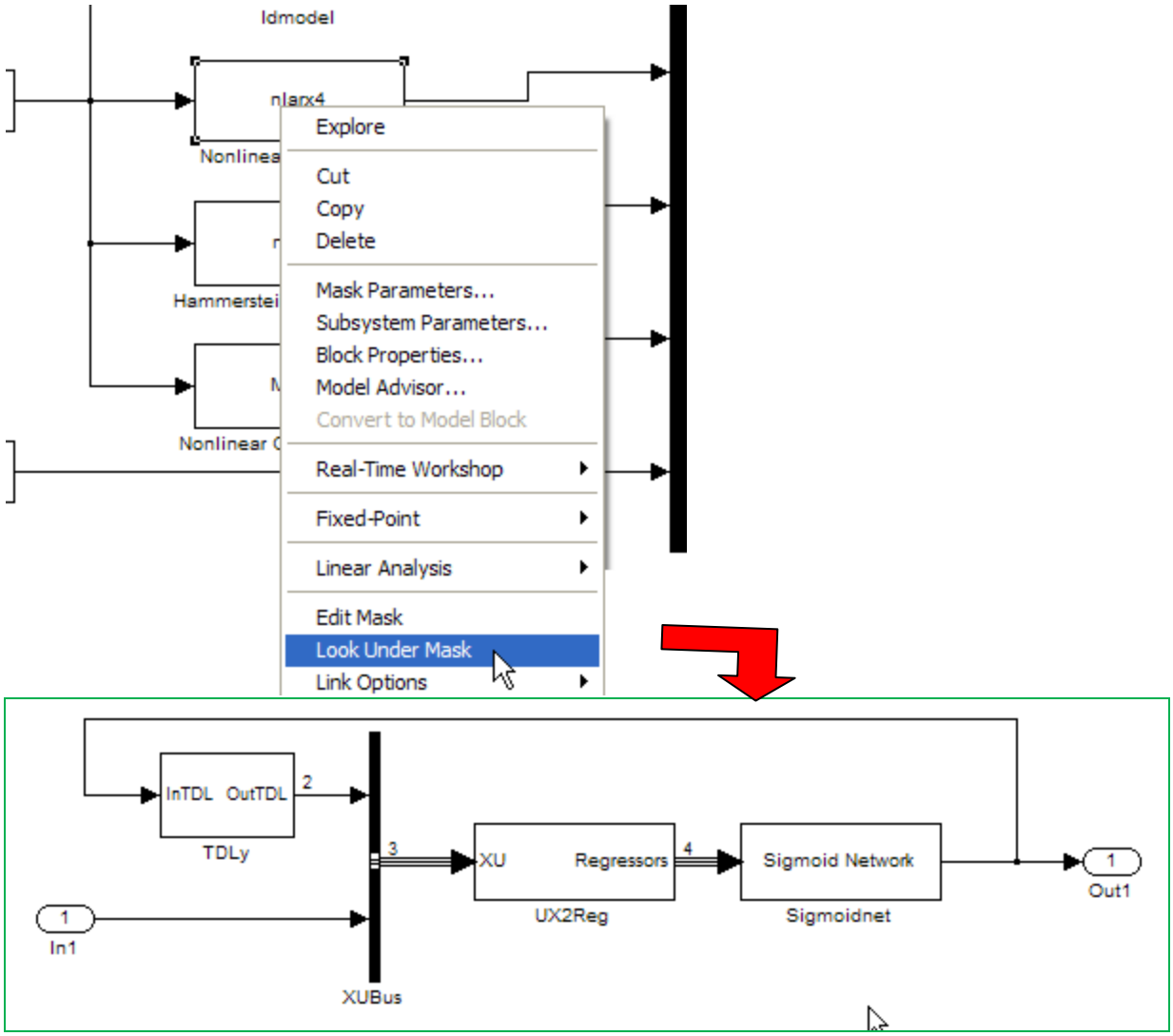


FIGURE 59: INPECTING DETAILED STRUCTURE OF A NONLINEAR ARX MODEL

CODE GENERATION

The model blocks for System Identification Toolbox support code generation with the following exceptions:

1. Nonlinear Grey Box model are not supported.
2. Nonlinear ARX models containing custom regressors can't be built using Real Time Workshop®. The reason is that evaluation of the formulas specified for custom regressors requires access to MATLAB. If you need to build such models, you can replace the "MATLAB Function" block used for custom regressors (named "EvalCustomReg") with Simulink's built-in blocks that generate the same answers. For example, in the model nlarx4 above, the

custom regressor employed has the formula: y(t) = max(x(t), 90) + min(x(t),15)-105. You can easily compute this result in Simulink by using saturation and summation blocks as shown below:

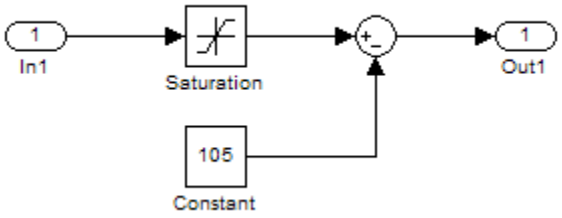


FIGURE 60: REPLACEMENT COMPONENT FOR CUSTOM REGRESSOR USED IN MODEL NLARX4

To achieve this replacement, you can proceed as follows:

(a) Look under the mask of the nonlinear ARX model that employs the custom regressor. Select all and copy
(b) Create a new Model (or subsystem) and paste the copied contents in there.
(c) Break library link for the Regressor block called "UX2Reg".
(d) Look under the mask of Regressor block called "UX2Reg". Delete EvalCustomReg block and replace it with a block that generates the custom regressor (here we have only one as a function of output signal).
(e) In the regressor block dialog, set the Model variable name to the name of the system ("nlarx4").

The automatically generated block diagram of a Neural Network model cannot be built using RTW.

FINAL NOTES

Some issues to be aware of:

CHOICE OF SAMPLE TIME

The sample time for modeling is chosen based on the nature of the system (its bandwidth) and a suitable length of data. The resulting models are discrete in time and have a fixed sample time (0.01 here). This need not be same as the sample time required for the generated code, which may be dictated by needs of the overall system or the hardware on which the code has to be executed. There are some ways to manage this difference:

1. In Simulink, rate transition blocks may be inserted around the model block to reconcile the model sample time with the simulation's fixed time step.
2. You can also estimate the model of a chosen form using data whose sample time is same as the desired sample time for code generation. If doing so, it is useful to keep in mind that the time delays (represented by "nk" parameter or "Input Delay" GUI option) are expressed in number of samples; it would have to be recalculated to be expressed as a multiple of the new sample time. For example, our models were created using a sample time of 0.01 sec. If the desired sample time is 0.001 sec, then the original value of nk would have to be

multiplied by 10. However, zero values may require extra consideration: in our examples, nk was zero indicating that for the sample time of 0.01 sec, the delay was insignificant. When the sample time is 0.001 sec, the delay need not be insignificant. Essentially, a choice of zero value only indicates that the delay is less than the sample time. If sample time should reduce, a zero value may have to be replaced by a non-zero value.

HOW TO CHOOSE INITIAL CONDITIONS ("INITIAL STATES") FOR SIMULATION

As mentioned before, the values of initial conditions should be given fresh thought; you should not simply use the values that were computed by the estimation algorithms (unless the same input signal is used for simulation and the simulation environment closely emulates the experimental set up used for collecting data). The values should be derived from physical considerations regarding the state of the system before start of simulation. Often, zero values or constant values denoting pre-existing equilibrium conditions are good choices. Look up documentation for definition of initial states for various model types.

REFERENCES

- Developing Models from Experimental Data using System Identification Toolbox (recorded webinar)
- System Identification Toolbox Documentation
- System Identification Toolbox product page at mathworks.com
- Books and papers relevant to system identification