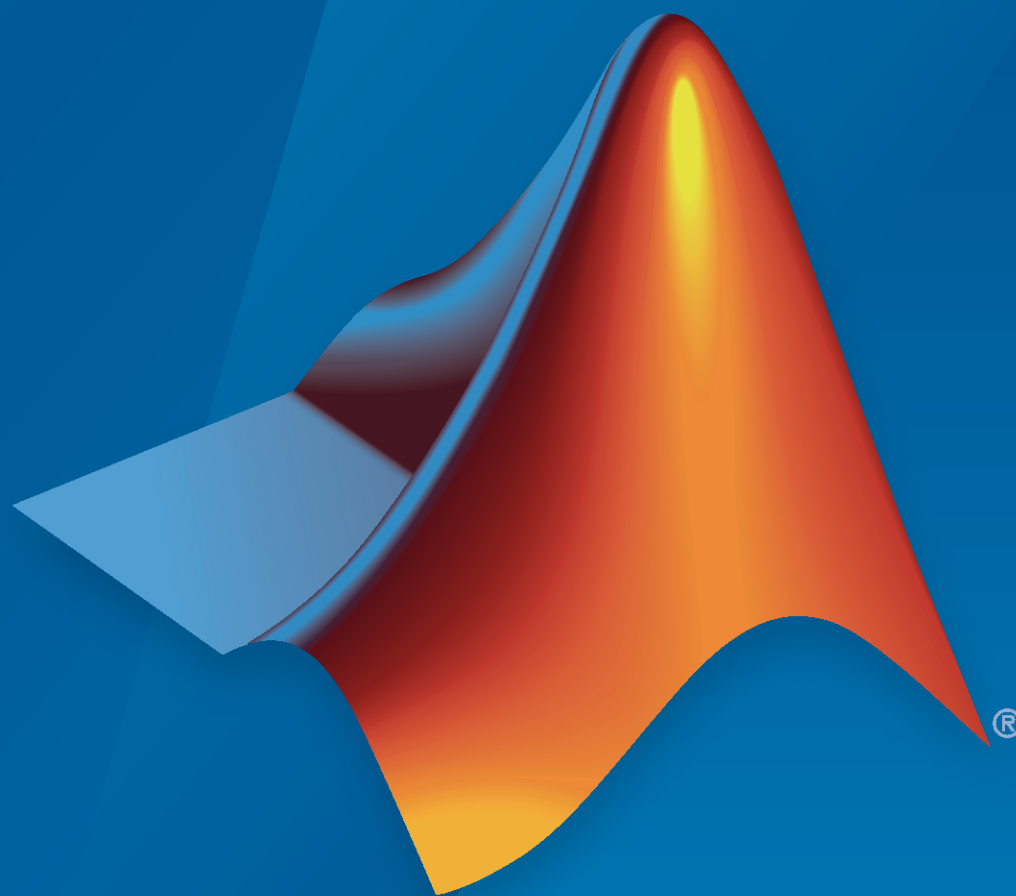# System Identification Toolbox™
Getting Started Guide

Lennart Ljung

MATLAB®&SIMULINK®

MathWorks®

How to Contact MathWorks

Latest news: www.mathworks.com

Sales and services: www.mathworks.com/sales_and_services

User community: www.mathworks.com/matlabcentral

Technical support: www.mathworks.com/support/contact_us

Phone: 508-647-7000

The MathWorks, Inc.
1 Apple Hill Drive
Natick, MA 01760-2098

Revision History

Contents

Nonlinear Model Identification

4

Product Overview

- "System Identification Toolbox Product Description" on page 1-2
- "Acknowledgments" on page 1-3
- "System Identification Overview" on page 1-4
- "Related Products" on page 1-17

System Identification Toolbox Product Description

Create linear and nonlinear dynamic system models from measured input-output data

System Identification Toolbox provides MATLAB® functions, Simulink® blocks, and an app for constructing mathematical models of dynamic systems from measured input-output data. It lets you create and use models of dynamic systems not easily modeled from first principles or specifications. You can use time-domain and frequency-domain input-output data to identify continuous-time and discrete-time transfer functions, process models, and state-space models. The toolbox also provides algorithms for embedded online parameter estimation.

The toolbox provides identification techniques such as maximum likelihood, prediction-error minimization (PEM), and subspace system identification. To represent nonlinear system dynamics, you can estimate Hammerstein-Weiner models and nonlinear ARX models with wavelet network, tree-partition, and sigmoid network nonlinearities. The toolbox performs grey-box system identification for estimating parameters of a user-defined model. You can use the identified model for system response prediction and plant modeling in Simulink. The toolbox also supports time-series data modeling and time-series forecasting.

Key Features

- Transfer function, process model, and state-space model identification using time-domain and frequency-domain response data
- Autoregressive (ARX, ARMAX), Box-Jenkins, and Output-Error model estimation using maximum likelihood, prediction-error minimization (PEM), and subspace system identification techniques
- Online model parameter estimation
- Time-series modeling (AR, ARMA) and forecasting
- Identification of nonlinear ARX models and Hammerstein-Weiner models with input-output nonlinearities such as saturation and dead zone
- Linear and nonlinear grey-box system identification for estimation of user-defined models
- Delay estimation, detrending, filtering, resampling, and reconstruction of missing data

Acknowledgments

System Identification Overview

System identification is a methodology for building mathematical models of dynamic systems using measurements of the input and output signals of the system.

The process of system identification requires that you:

- Measure the input and output signals from your system in time or frequency domain.
- Select a model structure.
- Apply an estimation method to estimate values for the adjustable parameters in the candidate model structure.
- Evaluate the estimated model to see if the model is adequate for your application needs.

Dynamic Systems and Models

In a dynamic system, the values of the output signals depend on both the instantaneous values of the input signals and also on the past behavior of the system. For example, a car seat is a dynamic system —the seat shape (settling position) depends on both the current weight of the passenger (instantaneous value) and how long the passenger has been riding in the car (past behavior).

A model is a mathematical relationship between the input and output variables of the system. Models of dynamic systems are typically described by differential or difference equations, transfer functions, state-space equations, and pole-zero-gain models.

You can represent dynamic models in both continuous-time and discrete-time form.

An often-used example of a dynamic model is the equation of motion of a spring-mass-damper system. As the following figure shows, the mass moves in response to the force $F(t)$ applied on the base to which the mass is attached. The input and output of this system are the force $F(t)$ and displacement $y(t)$, respectively.
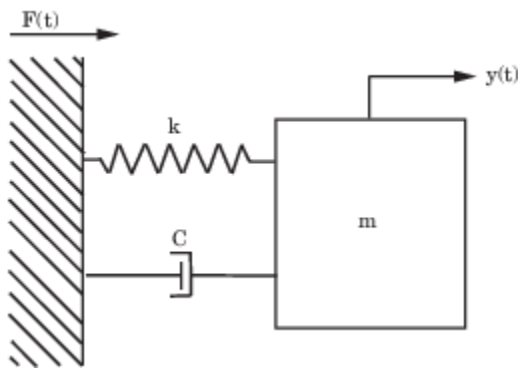
Continuous-Time Dynamic Model Example

You can represent the same physical system as several equivalent models. For example, you can represent the mass-spring-damper system in continuous time as a second-order differential equation:

$$m\frac{d^2y}{dt^2} + c\frac{dy}{dt} + ky(t) = F(t)$$

Here, m is the mass, k is the stiffness constant of the spring, and c is the damping coefficient. The solution to this differential equation lets you determine the displacement of the mass $y(t)$, as a function of external force $F(t)$ at any time t for known values of constant m, c, and k.

Consider the displacement $y(t)$ and velocity

$$v(t) = \frac{dy(t)}{dt}$$

as state variables:

$$x(t) = \begin{bmatrix} y(t) \\ v(t) \end{bmatrix}$$

You can express the previous equation of motion as a state-space model of the system:

$$\frac{dx}{dt} = Ax(t) + BF(t)$$

$$y(t) = Cx(t)$$

The matrices A, B, and C are related to the constants m, c, and k as follows:

$$A = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{c}{m} \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & \frac{1}{m} \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

You can also obtain a transfer function model of the spring-mass-damper system by taking the Laplace transform of the differential equation:

$$G(s) = \frac{Y(s)}{F(s)} = \frac{1}{(ms^2 + cs + k)}$$

Here, s is the Laplace variable.

Discrete-Time Dynamic Model Example

Suppose you can observe only the input and output variables $F(t)$ and $y(t)$ of the mass-spring-damper system at discrete time instants $t = nT_s$, where T_s is a fixed time interval and $n = 0, 1, 2, \ldots$. The variables are said to be *sampled* with sample time T_s. Then, you can represent the relationship between the sampled input-output variables as a second-order difference equation, such as

$$y(t) + a_1 y(t - T_s) + a_2 y(t - 2T_s) = bF(t - T_s)$$

Often, for simplicity, T_s is taken as one time unit, and the equation can be written as

$$y(t) + a_1 y(t-1) + a_2 y(t-2) = bF(t-1)$$

Here, a_1 and a_2 are the model parameters. The model parameters are related to the system constants m, c, and k, and the sample time T_s.

This difference equation shows the dynamic nature of the model. The displacement value at the time instant t depends not only on the value of force F at a previous time instant, but also on the displacement values at the previous two time instants $y(t–1)$ and $y(t–2)$.

You can use this equation to compute the displacement at a specific time. The displacement is represented as a weighted sum of the past input and output values:

$$y(t) = bF(t-1) - a_1 y(t-1) - a_2 y(t-2)$$

This equation shows an iterative way of generating values of the output $y(t)$ starting from initial conditions $y(0)$ and $y(1)$ and measurements of input $F(t)$. This computation is called simulation.

Alternatively, the output value at a given time t can be computed using the *measured* values of output at the previous two time instants and the input value at a previous time instant. This computation is called prediction. For more information on simulation and prediction using a model, see topics on the "Simulation and Prediction" page.

You can also represent a discrete-time equation of motion in state-space and transfer-function forms by performing the transformations similar to those described in "Continuous-Time Dynamic Model Example" on page 1-4.

Use Measured Data in System Identification

System identification uses the input and output signals you measure from a system to estimate the values of adjustable parameters in a given model structure. You can build models using time-domain input-output signals, frequency response data, time -series signals, and time-series spectra.

To obtain a good model of your system, you must have measured data that reflects the dynamic behavior of the system. The accuracy of your model depends on the quality of your measurement data, which in turn depends on your experimental design.

Time-Domain Data

Time-domain data consists of the input and output variables of the system that you record at a uniform sampling interval over a period of time.

For example, if you measure the input force $F(t)$ and mass displacement $y(t)$ of the spring-mass-damper system illustrated in "Dynamic Systems and Models" on page 1-4 at a uniform sampling frequency of 10 Hz, you obtain the following vectors of measured values:

$$u_{meas} = [F(T_s), F(2T_s), F(3T_s), ..., F(NT_s)]$$
$$y_{meas} = [y(T_s), y(2T_s), y(3T_s), ..., y(NT_s)]$$

Here, $T_s = 0.1$ seconds and NT_s is the time of the last measurement.

If you want to build a discrete-time model from this data, the data vectors u_{meas} and y_{meas} and the sample time T_s provide sufficient information for creating such a model.

If you want to build a continuous-time model, you must also know the intersample behavior of the input signals during the experiment. For example, the input can be piecewise constant (zero-order hold) or piecewise linear (first-order hold) between samples.

Frequency-Domain Data

Frequency-domain data represents measurements of the system input and output variables that you record or store in the frequency domain. The frequency-domain signals are Fourier transforms of the corresponding time-domain signals.

Frequency-domain data can also represent the frequency response of the system, represented by the set of complex response values over a given frequency range. The frequency response describes the outputs to sinusoidal inputs. If the input is a sine wave with frequency ω, then the output is also a sine wave of the same frequency, whose amplitude is $A(\omega)$ times the input signal amplitude and a phase shift of $\Phi(\omega)$ with respect to the input signal. The frequency response is $A(\omega)e^{(i\Phi(\omega))}$.

In the case of the mass-spring-damper system, you can obtain the frequency response data by using a sinusoidal input force and measuring the corresponding amplitude gain and phase shift of the response over a range of input frequencies.

You can use frequency-domain data to build both discrete-time and continuous-time models of your system.

Data Quality Requirements

System identification requires that your data capture the important dynamics of your system. Good experimental design ensures that you measure the right variables with sufficient accuracy and duration to capture the dynamics you want to model. In general, your experiment must:

- Use inputs that excite the system dynamics adequately. For example, a single step is seldom enough excitation.
- Measure data long enough to capture the important time constants.
- Set up a data acquisition system that has a good signal-to-noise ratio.
- Measure data at appropriate sampling intervals or frequency resolution.

You can analyze the data quality before building the model using the functions and techniques described in "Analyze Data". For example, you can analyze the input spectra to determine if the input signals have sufficient power over the bandwidth of the system. To get analysis and processing recommendations for your specific data, use `advice`.

You can also analyze your data to determine peak frequencies, input delays, important time constants, and indication of nonlinearities using nonparametric analysis tools in this toolbox. You can use this information for configuring model structures for building models from data. For more information, see:

- "Correlation Models"
- "Frequency-Response Models"

Build Models from Data

Model Structure

A model structure is a mathematical relationship between input and output variables that contains unknown parameters. Examples of model structures are transfer functions with adjustable poles and zeros, state-space equations with unknown system matrices, and nonlinear parameterized functions.

The following difference equation represents a simple model structure:

$$y(k) + ay(k-1) = bu(k)$$

Here, a and b are adjustable parameters.

The system identification process requires that you choose a model structure and apply the estimation methods to determine the numerical values of the model parameters.

You can use one of the following approaches to choose the model structure:

- You want a model that is able to reproduce your measured data and is as simple as possible. You can try various mathematical structures available in the toolbox. This modeling approach is called black-box modeling.
- You want a specific structure for your model, which you might have derived from first principles, but do not know numerical values of its parameters. You can represent the model structure as a set of equations or as a state-space system in MATLAB and estimate the values of its parameters from data. This approach is known as grey-box modeling.

Estimate Model Parameters

The System Identification Toolbox software estimates model parameters by minimizing the error between the model output and the measured response. The output y_{model} of the linear model is given by

$$y_{\text{model}}(t) = Gu(t)$$

Here, G is the transfer function.

To determine G, the toolbox minimizes the difference between the model output $y_{\text{model}}(t)$ and the measured output $y_{\text{meas}}(t)$. The minimization criterion is a weighted norm of the error, $v(t)$, where

$$v(t) = y_{meas}(t) - y_{model}(t).$$

$y_{model}(t)$ is one of the following:

- Simulated response ($Gu(t)$ of the model for a given input $u(t)$
- Predicted response of the model for a given input $u(t)$ and past measurements of the output $(y_{meas}(t\text{-}1), y_{meas}(t\text{-}2),...)$

Accordingly, the error $v(t)$ is called the simulation error or prediction error. The estimation algorithms adjust parameters in the model structure G such that the norm of this error is as small as possible.

Configure Parameter Estimation Algorithm

You can configure the estimation algorithm by:

- Configuring the minimization criterion to focus the estimation in a desired frequency range, for example, to put more emphasis at lower frequencies and deemphasize higher frequency noise contributions. You can also configure the criterion to target the intended application needs for the model, such as simulation or prediction.

- Specifying optimization options for iterative estimation algorithms.

  The majority of estimation algorithms in this toolbox are iterative. You can configure an iterative estimation algorithm by specifying options, such as the optimization method and the maximum number of iterations.

For more information about configuring the estimation algorithm, see "Options to Configure the Loss Function" and the topics for estimating specific model structures.

Black-Box Modeling

Select Black-Box Model Structure and Order

Black-box modeling is useful when your primary interest is in fitting the data regardless of a particular mathematical structure of the model. The toolbox provides several linear and nonlinear black-box model structures, which have traditionally been useful for representing dynamic systems. These model structures vary in complexity depending on the flexibility you need to account for the dynamics and noise in your system. You can choose one of these structures and compute its parameters to fit the measured response data.

Black-box modeling is usually a trial-and-error process, where you estimate the parameters of various structures and compare the results. Typically, you start with the simple linear model structure and progress to more complex structures. You might also choose a model structure because you are more familiar with this structure or because you have specific application needs.

The simplest linear black-box structures require the fewest options to configure:

- Transfer function, with a given number of poles and zeros
- Linear ARX model, which is the simplest input-output polynomial model
- State-space model, which you can estimate by specifying the number of model states

Estimation of some of these structures also uses noniterative estimation algorithms, which further reduces complexity.

You can configure a model structure using the model order. The definition of model order varies depending on the type of model you select. For example, if you choose a transfer function representation, the model order is related to the number of poles and zeros. For state-space representation, the model order corresponds to the number of states. In some cases, such as for linear ARX and state-space model structures, you can estimate the model order from the data.

If the simple model structures do not produce good models, you can select more complex model structures by:

- Specifying a higher model order for the same linear model structure. A higher model order increases the model flexibility for capturing complex phenomena. However, an unnecessarily high order can make the model less reliable.

- Explicitly modeling the noise by including the $He(t)$ term, as shown in the following equation.

  $y(t) = Gu(t) + He(t)$

Here, *H* models the additive disturbance by treating the disturbance as the output of a linear system driven by a white noise source *e*(*t*).

Using a model structure that explicitly models the additive disturbance can help to improve the accuracy of the measured component *G*. Furthermore, such a model structure is useful when your main interest is using the model for predicting future response values.

- Using a different linear model structure.

  See "Linear Model Structures".
- Using a nonlinear model structure.

  Nonlinear models have more flexibility in capturing complex phenomena than linear models of similar orders. See "Nonlinear Model Structures".

Ultimately, you choose the simplest model structure that provides the best fit to your measured data. For more information, see "Estimating Linear Models Using Quick Start" on page 3-14.

Regardless of the structure you choose for estimation, you can simplify the model for your application needs. For example, you can separate out the measured dynamics (*G*) from the noise dynamics (*H*) to obtain a simpler model that represents just the relationship between *y* and *u*. You can also linearize a nonlinear model about an operating point.

Use Nonlinear Model Structures

A linear model is often sufficient to accurately describe the system dynamics and, in most cases, a best practice is to first try to fit linear models. If the linear model output does not adequately reproduce the measured output, you might need to use a nonlinear model.

You can assess the need to use a nonlinear model structure by plotting the response of the system to an input. If you notice that the responses differ depending on the input level or input sign, try using a nonlinear model. For example, if the output response to an input step up is faster than the response to a step down, you might need a nonlinear model.

Before building a nonlinear model of a system that you know is nonlinear, try transforming the input and output variables such that the relationship between the transformed variables is linear. For example, consider a system that has current and voltage as inputs to an immersion heater, and the temperature of the heated liquid as an output. The output depends on the inputs through the power of the heater, which is equal to the product of current and voltage. Instead of building a nonlinear model for this two-input and one-output system, you can create a new input variable by taking the product of the current and voltage and building a linear model that describes the relationship between power and temperature.

If you cannot determine variable transformations that yield a linear relationship between input and output variables, you can use nonlinear structures such as nonlinear ARX or Hammerstein-Wiener models. For a list of supported nonlinear model structures and when to use them, see "Nonlinear Model Structures".

Black-Box Estimation Example

You can use the System Identification app or commands to estimate linear and nonlinear models of various structures. In most cases, you choose a model structure and estimate the model parameters using a single command.

Consider the mass-spring-damper system described in "Dynamic Systems and Models" on page 1-4. If you do not know the equation of motion of this system, you can use a black-box modeling approach to

build a model. For example, you can estimate transfer functions or state-space models by specifying the orders of these model structures.

A transfer function is a ratio of polynomials:

$$G(s) = \frac{\left(b_0 + b_1 s + b_2 s^2 + ...\right)}{\left(1 + f_1 s + f_2 s^2 + ...\right)}$$

For the mass-spring damper system, this transfer function is

$$G(s) = \frac{1}{\left(ms^2 + cs + k\right)}$$

which is a system with no zeros and 2 poles.

In discrete-time, the transfer function of the mass-spring-damper system can be

$$G\left(z^{-1}\right) = \frac{bz^{-1}}{\left(1 + f_1 z^{-1} + f_2 z^{-2}\right)}$$

where the model orders correspond to the number of coefficients of the numerator and the denominator (nb = 1 and nf = 2) and the input-output delay equals the lowest order exponent of z^{-1} in the numerator (nk = 1).

In continuous time, you can build a linear transfer function model using the `tfest` command.

`m = tfest(data,2,0)`

Here, `data` is your measured input-output data, represented as an `iddata` object, and the model order is the set of number of poles (2) and the number of zeros (0).

Similarly, you can build a discrete-time model Output Error structure using the `oe` command.

`m = oe(data,[1 2 1])`

The model order is [nb nf nk] = [1 2 1]. Usually, you do not know the model orders in advance. Try several model order values until you find the orders that produce an acceptable model.

Alternatively, you can choose a state-space structure to represent the mass-spring-damper system and estimate the model parameters using the `ssest` or the `n4sid` command.

`m = ssest(data,2)`

Here, the second argument 2 represents the order, or the number of states in the model.

In black-box modeling, you do not need the equation of motion for the system — only a guess of the model orders.

For more information about building models, see "Steps for Using the System Identification App" and "Model Estimation Commands".

Grey-Box Modeling

In some situations, you can deduce the model structure from physical principles. For example, the mathematical relationship between the input force and the resulting mass displacement in the spring-

mass-damper system illustrated in "Dynamic Systems and Models" on page 1-4 is well known. In state-space form, the model is given by

$$\frac{dx}{dt} = Ax(t) + BF(t)$$

$$y(t) = Cx(t)$$

where $x(t) = [y(t); v(t)]$ is the state vector. The coefficients A, B, and C are functions of the model parameters:

A = [0 1; –k/m –c/m]

B = [0; 1/m]

C = [1 0]

Here, you fully know the model structure but do not know the values of its parameters—m, c, and k.

In the grey-box approach, you use the data to estimate the values of the unknown parameters of your model structure. You specify the model structure by a set of differential or difference equations in MATLAB and provide some initial guess for the unknown parameters specified.

In general, you build grey-box models by:

1   Creating a template model structure.
2   Configuring the model parameters with initial values and constraints (if any).
3   Applying an estimation method to the model structure and computing the model parameter values.

The following table summarizes the ways you can specify a grey-box model structure.

Grey-Box Structure Representation	Learn More
Represent the state-space model structure as a structured `idss` model object and estimate the state-space matrices A, B, and C. You can compute the parameter values, such as m, c, and k, from the state space matrices A and B. For example, $m = 1/B(2)$ and $k = –A(2,1)m$.	• "Estimate State-Space Models with Canonical Parameterization" • "Estimate State-Space Models with Structured Parameterization"
Represent the state-space model structure as an `idgrey` model object. You can directly estimate the values of parameters m, c, and k.	"Grey-Box Model Estimation"

Evaluate Model Quality

After you estimate the model, you can evaluate the model quality by:

• "Compare Model Response to Measured Response" on page 1-13
• "Analyze Residuals" on page 1-14
• "Analyze Model Uncertainty" on page 1-14

Ultimately, you must assess the quality of your model based on whether the model adequately addresses the needs of your application. For information about other available model analysis techniques, see "Model Analysis".

If you do not get a satisfactory model, you can iteratively improve your results by trying a different model structure, changing the estimation algorithm settings, or performing additional data processing. If these changes do not improve your results, you might need to revisit your experimental design and data gathering procedures.

Compare Model Response to Measured Response

Typically, you evaluate the quality of a model by comparing the model response to the measured output for the same input signal.

Suppose you use a black-box modeling approach to create dynamic models of the spring-mass damper system. You try various model structures and orders, such as:

```
model1 = arx(data, [2 1 1]);
model2 = n4sid(data, 3)
```

You can simulate these models with a particular input and compare their responses against the measured values of the displacement for the same input applied to the real system. The following figure compares the simulated and measured responses for a step input.
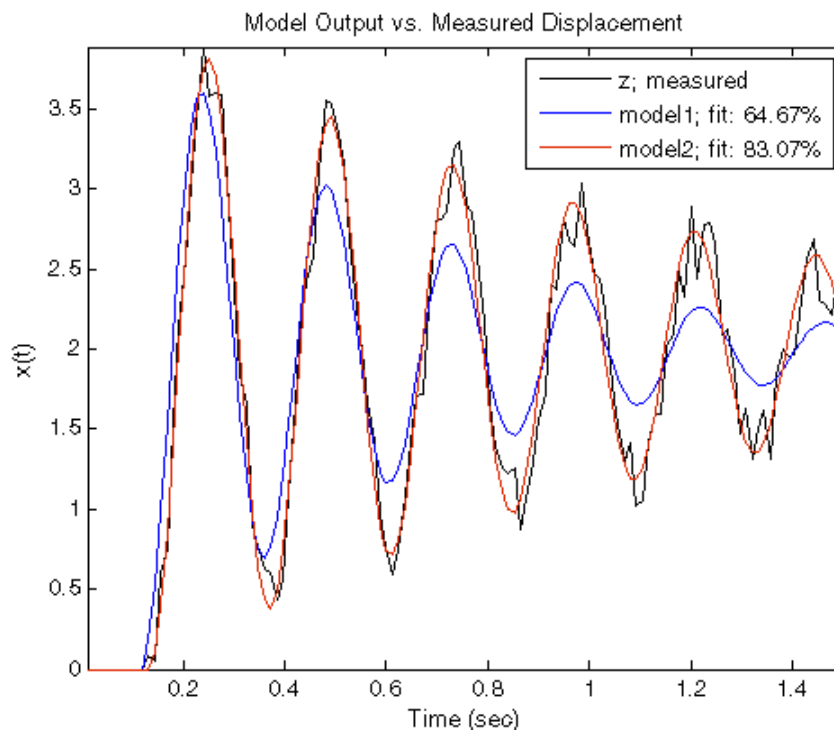
The figure indicates that `model2` is better than `model1` because `model2` better fits the data (65% vs. 83%).

The fit percentage indicates the agreement between the model response and the measured output: 100 means a perfect fit, and 0 indicates a poor fit (that is, the model output has the same fit to the measured output as the mean of the measured output).

For more information, see topics on the "Compare Output with Measured Data" page.

Analyze Residuals

The System Identification Toolbox software lets you perform residual analysis to assess the model quality. Residuals represent the portion of the output data not explained by the estimated model. A good model has residuals uncorrelated with past inputs.

For more information, see the topics on the "Residual Analysis" page.

Analyze Model Uncertainty

When you estimate the model parameters from data, you obtain their nominal values that are accurate within a confidence region. The size of this region is determined by the values of the parameter uncertainties computed during estimation. The magnitude of the uncertainties provide a measure of the reliability of the model. Large uncertainties in parameters can result from unnecessarily high model orders, inadequate excitation levels in the input data, and a poor signal-to-noise ratio in measured data.

You can compute and visualize the effect of parameter uncertainties on the model response in the time and frequency domains using pole-zero maps, Bode response plots, and step response plots. For example, in the following Bode plot of an estimated model, the shaded regions represent the uncertainty in amplitude and phase of the frequency response of the model, computed using the uncertainty in the parameters. The plot shows that the uncertainty is low only in the 5 to 50 rad/s frequency range, which indicates that the model is reliable only in this frequency range.
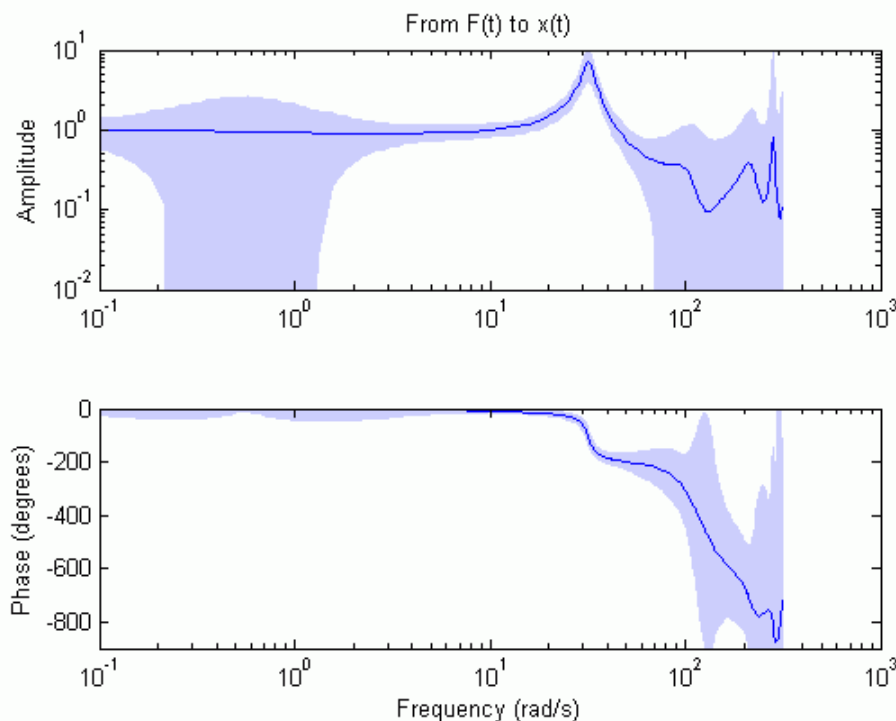
For more information, see "Compute Model Uncertainty".

Resources

The System Identification Toolbox documentation provides you with the necessary information to use this product. Additional resources are available to help you learn more about specific aspects of system identification theory and applications.

The following book describes methods for system identification and physical modeling:

- Ljung, Lennart, and Torkel Glad. *Modeling of Dynamic Systems*. Prentice Hall Information and System Sciences Series. Englewood Cliffs, NJ: PTR Prentice Hall, 1994.

These books provide detailed information about system identification theory and algorithms:

- Ljung, Lennart. *System Identification: Theory for the User*. Second edition. Prentice Hall Information and System Sciences Series. Upper Saddle River, NJ: PTR Prentice Hall, 1999.
- Söderström, Torsten, and Petre Stoica. *System Identification*. Prentice Hall International Series in Systems and Control Engineering. New York: Prentice Hall, 1989.

For information about working with frequency-domain data, see the following book:

- Pintelon, Rik, and Johan Schoukens. *System Identification. A Frequency Domain Approach*. Hoboken, NJ: John Wiley & Sons, 2001. https://doi.org/10.1002/0471723134.

For information on nonlinear identification, see the following references:

- Sjöberg, Jonas, Qinghua Zhang, Lennart Ljung, Albert Benveniste, Bernard Delyon, Pierre-Yves Glorennec, Håkan Hjalmarsson, and Anatoli Juditsky. "Nonlinear Black-Box Modeling in System Identification: A Unified Overview." *Automatica* 31, no. 12 (December 1995): 1691–1724. https://doi.org/10.1016/0005-1098(95)00120-8.
- Juditsky, Anatoli, Håkan Hjalmarsson, Albert Benveniste, Bernard Delyon, Lennart Ljung, Jonas SjÖberg, and Qinghua Zhang. "Nonlinear Black-Box Models in System Identification: Mathematical Foundations." *Automatica* 31, no. 12 (December 1995): 1725–50. https://doi.org/10.1016/0005-1098(95)00119-1.
- Zhang, Qinghua, and Albert Benveniste. "Wavelet Networks." *IEEE Transactions on Neural Networks* 3, no. 6 (November 1992): 889–98. https://doi.org/10.1109/72.165591.
- Zhang, Qinghua. "Using Wavelet Network in Nonparametric Estimation." *IEEE Transactions on Neural Networks* 8, no. 2 (March 1997): 227–36. https://doi.org/10.1109/72.557660.

For more information about systems and signals, see the following book:

- Oppenheim, Alan V., and Alan S. Willsky, *Signals and Systems*. Upper Saddle River, NJ: PTR Prentice Hall, 1985.

The following textbook describes numerical techniques for parameter estimation using criterion minimization:

- Dennis, J. E., Jr., and Robert B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Upper Saddle River, NJ: PTR Prentice Hall, 1983.

See Also

More About

- "Get Started with System Identification Toolbox"

Related Products

The following table summarizes MathWorks® products that extend and complement the System Identification Toolbox software. For current information about these and other MathWorks products, point your Web browser to:

`www.mathworks.com`

Product	Description
"Control System Toolbox"	Provides extensive tools to analyze plant models created in the System Identification Toolbox software and to tune control systems based on these plant models. You can use the identified models directly for advanced linear analysis and control design tasks — no conversion of the format required.
"Model Predictive Control Toolbox"	Uses the linear plant models created in the System Identification Toolbox software for predicting plant behavior that is optimized by the model-predictive controller.
"Deep Learning Toolbox"	Provides flexible neural-network structures for estimating nonlinear models using the System Identification Toolbox software.
"Optimization Toolbox"	When this toolbox is installed, you have the option of using the `lsqnonlin` optimization algorithm for linear and nonlinear identification.
"Robust Control Toolbox"	Provides tools to design multiple-input and multiple-output (MIMO) control systems based on plant models created in the System Identification Toolbox software. Helps you assess robustness based on confidence bounds for the identified plant model.
"Signal Processing Toolbox"	Provides additional options for: • Filtering (The System Identification Toolbox software provides only the fifth-order Butterworth filter.) • Spectral analysis After using the advanced data processing capabilities of the Signal Processing Toolbox™ software, you can import the data into the System Identification Toolbox software for modeling.
"Simulink"	Provides System Identification blocks for simulating the models you identified using the System Identification Toolbox software. Also provides blocks for model estimation.

Using This Product

When to Use the App vs. the Command Line

After installing the System Identification Toolbox product, you can start the System Identification app or work at the command line.

You can work either in the app or at the command line to preprocess data, and estimate, validate, and compare models.

The following operations are available only at the command line:

- Generating input and output data (see `idinput`).
- Estimating coefficients of linear and nonlinear ordinary differential or difference equations (grey-box models).
- Using recursive online estimation methods. For more information, see topics about estimating linear models recursively on the "Online Estimation" page.
- Converting between continuous-time and discrete-time models (see `c2d` and `d2c` reference pages).
- Converting models to Control System Toolbox™ LTI objects (see `ss`, `tf`, and `zpk`).

Note Conversions to LTI objects require the Control System Toolbox software.

New users should start by using the app to become familiar with the product. To open the app, on the **Apps** tab of MATLAB desktop, in the **Apps** section, click **System Identification**. Alternatively, type `systemIdentification` in the MATLAB Command Window.

To work at the command line, type the commands directly in the MATLAB Command Window. For more information about a command, type `doc command_name` in the MATLAB Command Window.

See Also

More About

- "System Identification Workflow" on page 2-3
- "Commands for Model Estimation" on page 2-5
- "Working with System Identification App"

System Identification Workflow

System identification is an iterative process, where you identify models with different structures from data and compare model performance. Ultimately, you choose the simplest model that best describes the dynamics of your system.

Because this toolbox lets you estimate different model structures quickly, you should try as many different structures as possible to see which one produces the best results.

A system identification workflow might include the following tasks:

1 Process data for system identification by:

- Importing data into the MATLAB workspace.
- Representing data in the System Identification app or as an `iddata` or `idfrd` object in the MATLAB workspace.
- Plotting data to examine both time- and frequency-domain behavior.

  To analyze the data for the presence of constant offsets and trends, delay, feedback, and signal excitation levels, you can also use the `advice` command.
- Preprocessing data by removing offsets and linear trends, interpolating missing values, filtering to emphasize a specific frequency range, or resampling (interpolating or decimating) using a different time interval.

2 Identify linear or nonlinear models:

- Frequency-response models
- Impulse-response models
- Low-order transfer functions (process models)
- Input-output polynomial models
- State-space models
- Transfer function models
- Nonlinear black-box models
- Ordinary difference or differential equations (grey-box models)

3 Validate models.

  When you do not achieve a satisfactory model, try a different model structure and order or try another identification algorithm. In some cases, you can improve results by including a noise model.

  You might need to preprocess your data before doing further estimation. For example, if there is too much high-frequency noise in your data, you might need to filter or decimate (resample) the data before modeling.

4 Postprocess models:

- Transform between continuous- and discrete-time domains
- Transform between model representations
- Extract numerical model data
- Subreference, concatenate and merge models

- Linearize nonlinear models

5 Use identified models for:

- "Simulation and Prediction"
- Control design for the estimated plant using other MathWorks products.

  You can import an estimated linear model into Control System Toolbox, Model Predictive Control Toolbox™, Robust Control Toolbox™, or Simulink software.

- As dynamic blocks in Simulink

For online applications, you can perform online estimation.

Commands for Model Estimation

The following tables summarize System Identification Toolbox commands for offline and online estimation. For detailed information about using each command, see the corresponding reference page.

You can compile all the estimation commands using MATLAB Compiler™ software. Using MATLAB Coder™ software, you can only generate C and C++ code for *online* estimation commands, except for `rpem`, `rplr`, and `segment`.

Commands for Offline Estimation

Model Type	Estimation Commands
Transfer function models	`tfest`
Process models (low-order transfer functions expressed in time-constant form)	`procest`
Linear input-output polynomial models	`armax` (ARMAX and ARIMAX models) `arx` (ARX and ARIX models) `bj` (BJ only) `iv4` (ARX only) `ivx` (ARX only) `oe` (OE only) `polyest` (for all models)

State-space models	`n4sid`
	`ssest`
	`ssregest`

Frequency-response models	`etfe`
	`spa`
	`spafdr`

Correlation models	`cra` `impulseest`

Linear time-series models	`ar`
	`arx` (for multiple outputs)
	`ivar`

Linear grey-box models	`greyest`
Nonlinear ARX models	`nlarx`
Hammerstein-Wiener models	`nlhw`
Nonlinear grey-box models	`nlgreyest`
Linear and nonlinear models	`pem`

Commands for Online Estimation

Model Type	Estimation Commands

Linear input-output polynomial models	`recursiveARX`
	`recursiveARMAX`
	`recursiveOE`
	`recursiveBJ`

Linear time-series models	`recursiveAR` `recursiveARMA`
Model that is linear in parameters	`recursiveLS`

Linear polynomial models	`rpem`
	`rplr`
	`segment` (AR, ARMA, ARX, and ARMAX models only)

See Also

More About

- "What Is Online Estimation?"

Linear Model Identification

Identify Linear Models Using System Identification App

Introduction

Objectives

Estimate and validate linear models from single-input/single-output (SISO) data to find the one that best describes the system dynamics.

After completing this tutorial, you will be able to accomplish the following tasks using the System Identification app:

- Import data arrays from the MATLAB workspace into the app.
- Plot the data.
- Process data by removing offsets from the input and output signals.
- Estimate, validate, and compare linear models.
- Export models to the MATLAB workspace.

Note The tutorial uses time-domain data to demonstrate how you can estimate linear models. The same workflow applies to fitting frequency-domain data.

This tutorial is based on the example in section 17.3 of *System Identification: Theory for the User*, Second Edition, by Lennart Ljung, Prentice Hall PTR, 1999.

Data Description

This tutorial uses the data file `dryer2.mat`, which contains single-input/single-output (SISO) time-domain data from Feedback Process Trainer PT326. The input and output signals each contain 1000 data samples.

This system heats the air at the inlet using a mesh of resistor wire, similar to a hair dryer. The input is the power supplied to the resistor wires, and the output is the air temperature at the outlet.

Preparing Data for System Identification

Loading Data into the MATLAB Workspace

Load the data in `dryer2.mat` by typing the following command in the MATLAB Command Window:

```
load dryer2
```

This command loads the data into the MATLAB workspace as two column vectors, `u2` and `y2`, respectively. The variable `u2` is the input data and `y2` is the output data.

Opening the System Identification App

To open the System Identification app, type the following command in the MATLAB Command Window:

```
systemIdentification
```

The default session name, `Untitled`, appears in the title bar.
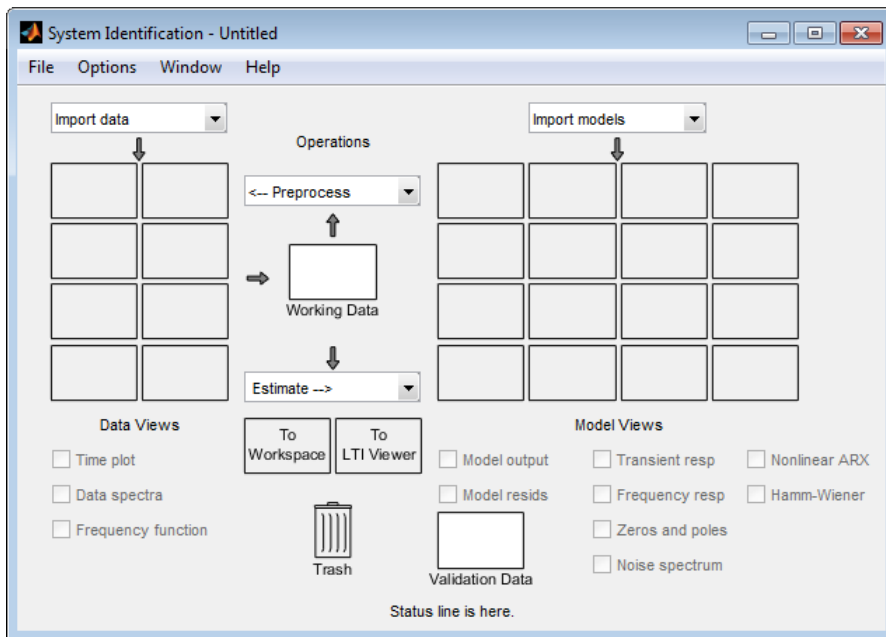
Importing Data Arrays into the System Identification App

You can import the single-input/single-output (SISO) data from a sample data file `dryer2.mat` into the app from the MATLAB workspace.

You must have already loaded the sample data into MATLAB, as described in "Loading Data into the MATLAB Workspace" on page 3-2, and opened the System Identification app, as described in "Opening the System Identification App" on page 3-2.

To import data arrays into the System Identification app:

1 Select **Import data > Time domain data**. This action opens the Import Data dialog box.
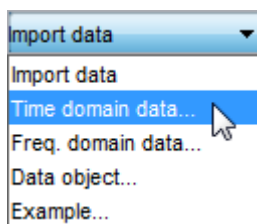
2 In the Import Data dialog box, specify the following options:

- **Input** — Enter `u2` as the name of the input variable.
- **Output** — Enter `y2` as the name of the output variable.
- **Data name** — Change the default name to `data`. This name labels the data in the System Identification app after the import operation is completed.
- **Starting time** — Enter `0` as the starting time. This value designates the starting value of the time axis on time plots.

- **Sample Time** — Enter `0.08` as the time between successive samples in seconds. This value is the actual sample time in the experiment.

The Import Data dialog box now resembles the following figure.
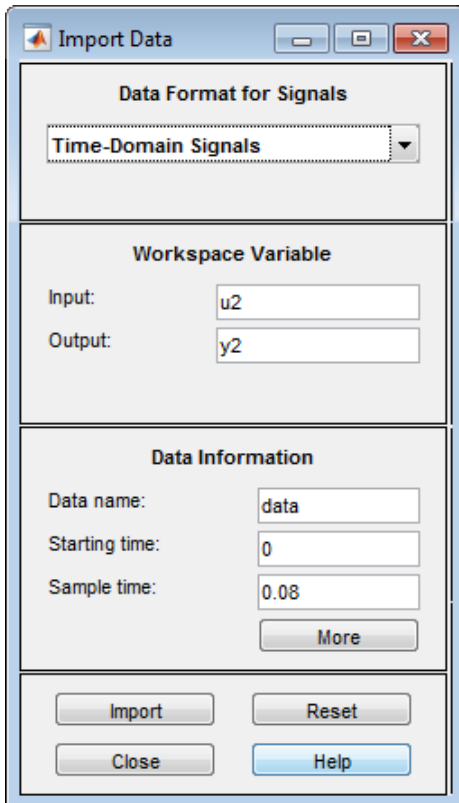
3   In the **Data Information** area, click **More** to expand the dialog box and specify the following options:

Input Properties

- **InterSample** — Accept the default `zoh` (zero-order hold) to indicate that the input signal was piecewise-constant between samples during data acquisition. This setting specifies the behavior of the input signals between samples when you transform the resulting models between discrete-time and continuous-time representations.
- **Period** — Accept the default `inf` to specify a nonperiodic input.

Note   For a periodic input, enter the whole number of periods of the input signal in your experiment.

Channel Names

- **Input** — Enter `power`.

Tip   Naming channels helps you to identify data in plots. For multivariable input and output signals, you can specify the names of individual **Input** and **Output** channels, separated by commas.

- **Output** — Enter `temperature`.

Physical Units of Variables

- **Input** — Enter `W` for power units.

Tip When you have multiple inputs and outputs, enter a comma-separated list of **Input** and **Output** units corresponding to each channel.

- **Output** — Enter `^oC` for temperature units.

Notes — Enter comments about the experiment or the data. For example, you might enter the experiment name, date, and a description of experimental conditions. When you estimate models from this data, these models inherit your notes.

The expanded Import Data dialog box now resembles the following figure.

4   Click **Import** to add the data to the System Identification app. The app displays an icon to represent the data.
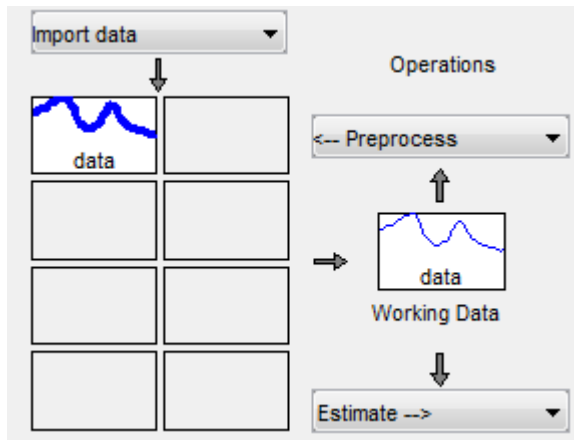
5   Click **Close** to close the Import Data dialog box.

Plotting and Processing Data

In this portion of the tutorial, you evaluate the data and process it for system identification. You learn how to:

- Plot the data.
- Remove offsets from the data by subtracting the mean values of the input and the output.
- Split the data into two parts to use one part model estimation and the other part for model validation.

The reason you subtract the mean values from each signal is because, typically, you build linear models that describe the responses for deviations from a physical equilibrium. With steady-state data, it is reasonable to assume that the mean levels of the signals correspond to such an equilibrium. Thus, you can seek models around zero without modeling the absolute equilibrium levels in physical units.

You must have already imported data into the System Identification app, as described in "Importing Data Arrays into the System Identification App" on page 3-3.

To plot and process the data:

1   Select the **Time plot** check box to open the Time Plot. If the plot window is empty, click the `data` icon in the System Identification app.
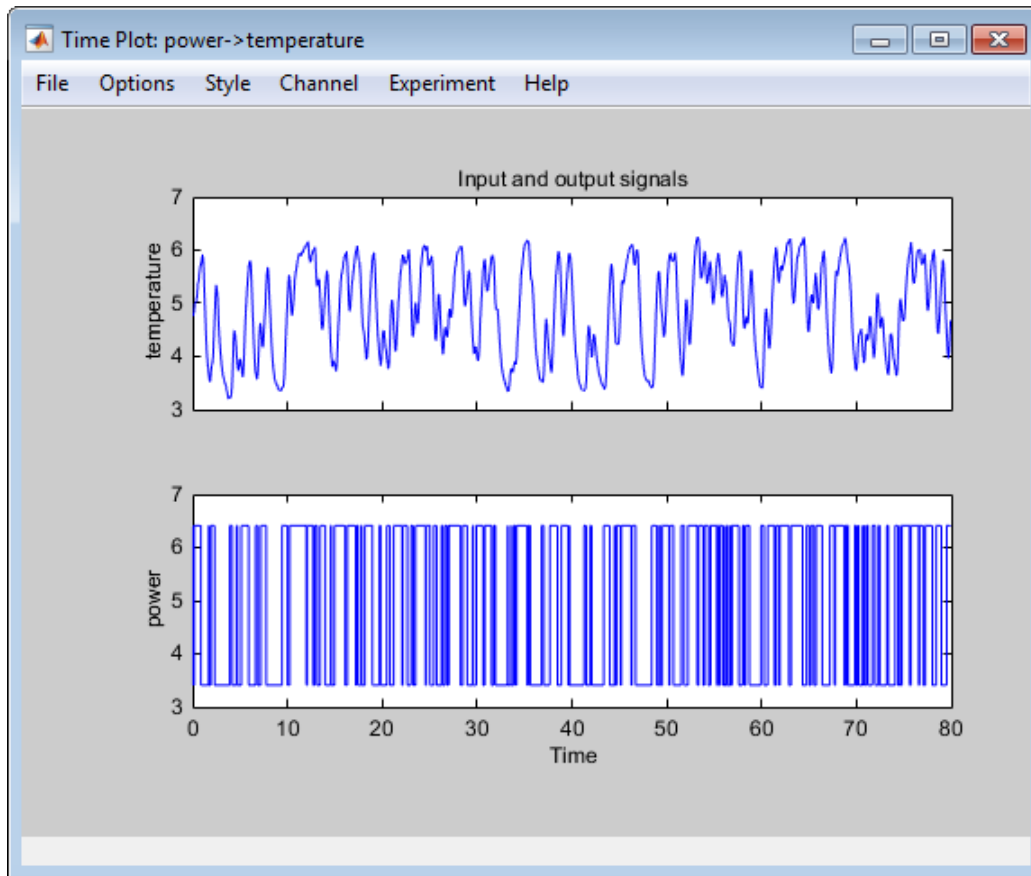
The top axes show the output data (temperature), and the bottom axes show the input data (power). Both the input and the output data have nonzero mean values.

2   Subtract the mean input value from the input data and the mean output value from the output data. In the System Identification app, select **<--Preprocess > Remove means**.
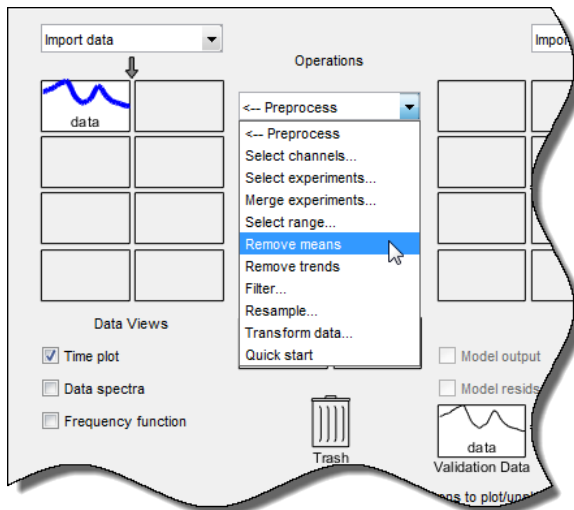
This action adds a new data set to the System Identification app with the default name `datad` (the suffix *d* means *detrend*), and updates the Time Plot window to display both the original and the detrended data. The detrended data has a zero mean value.
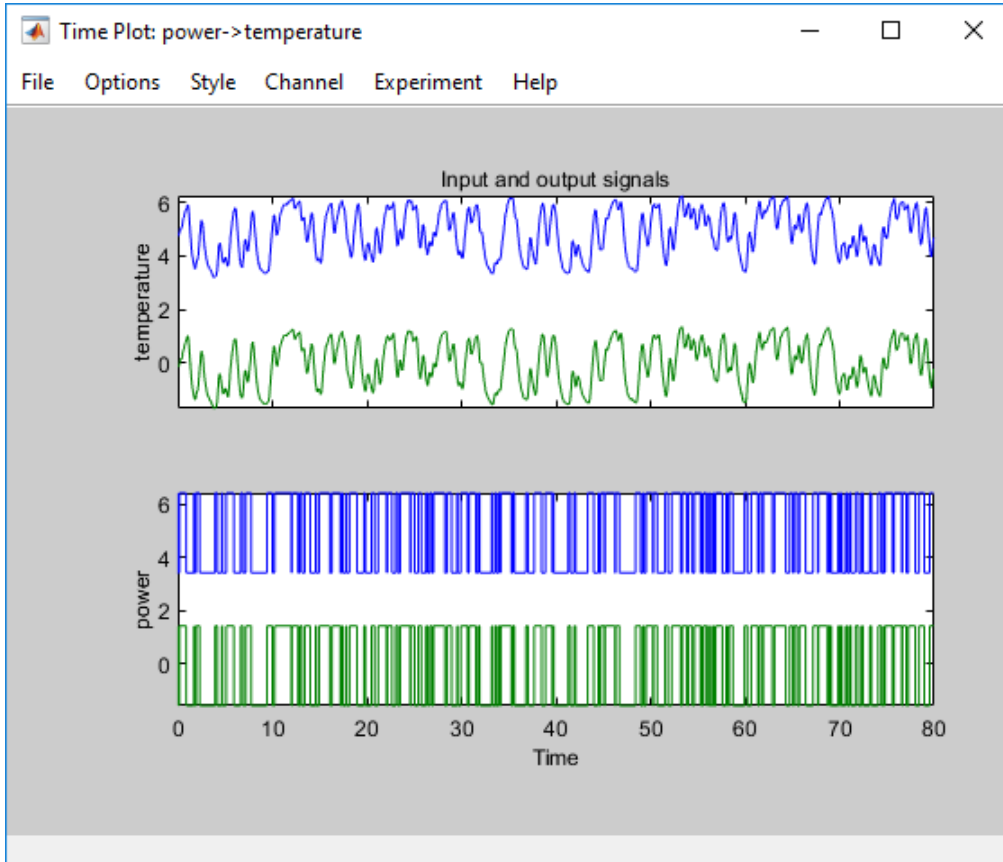
3 Specify the detrended data to be used for estimating models. Drag the data set `datad` to the **Working Data** rectangle.

Alternatively, right-click the `datad` icon to open the Data/model Info dialog box.
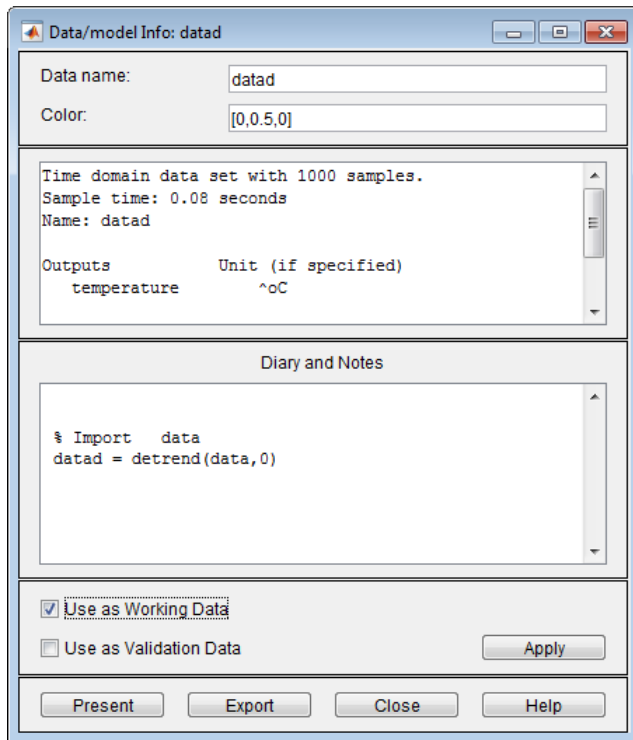
Select the **Use as Working Data** check-box. Click **Apply** and then **Close**. This action adds `datad` to the **Working Data** rectangle.

4 Split the data into two parts and specify the first part for model estimation, and the second part for model validation.

a Select **<--Preprocess > Select range** to open the Select Range window.
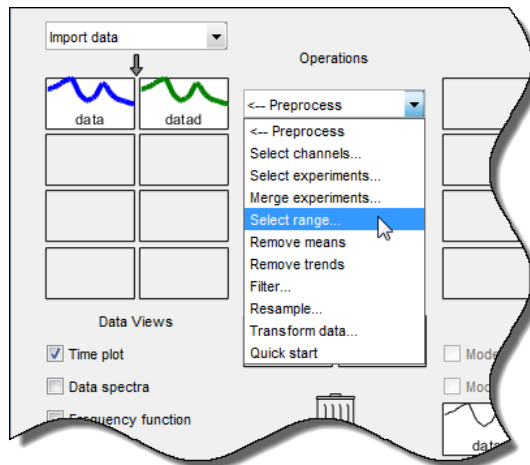
b In the Select Range window, create a data set containing the first 500 samples. In the **Samples** field, specify `1 500`.

> **Tip** You can also select data samples using the mouse by clicking and dragging a rectangular region on the plot. If you select samples on the input-channel axes, the corresponding region is also selected on the output-channel axes.

c In the **Data name** field, type the name `data_est`.
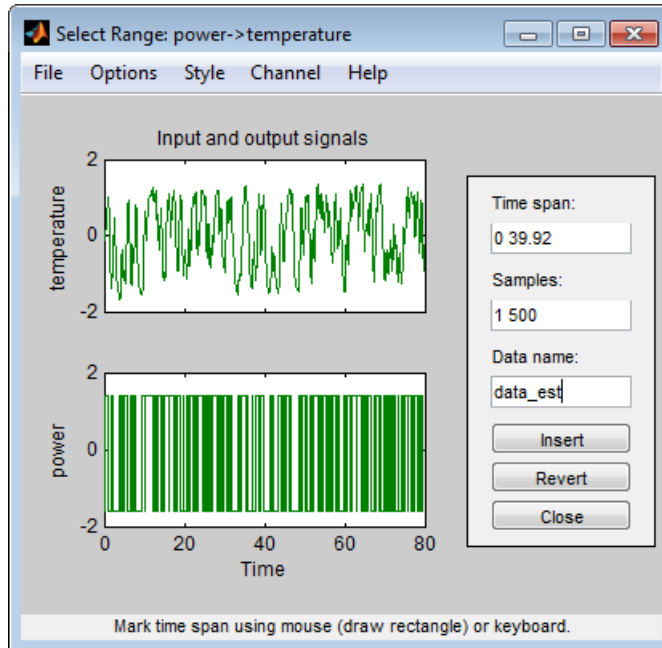
d Click **Insert** to add this new data set to the System Identification app to be used for model estimation.

e Repeat this process to create a second data set containing a subset of the data to use for validation. In the Select Range window, specify the last 500 samples in the **Samples** field. Type the name `data_val` in the **Data name** field. Click **Insert** to add this new data set to the System Identification app.
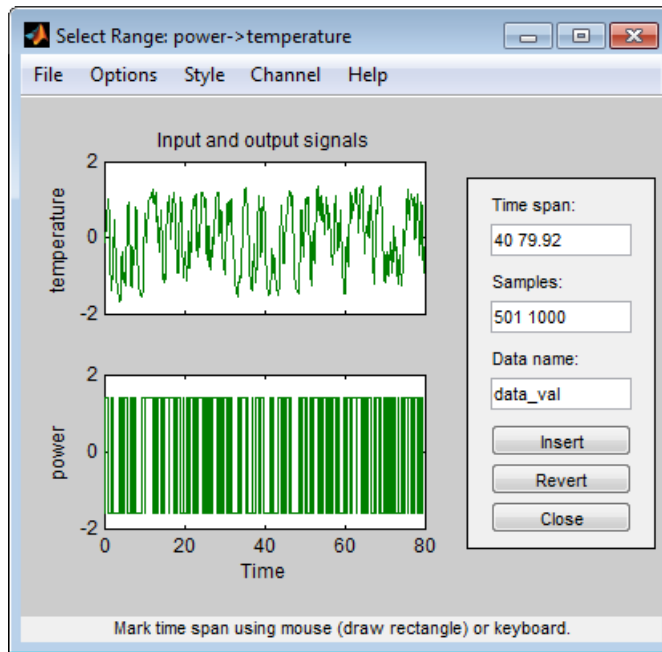
f Click **Close** to close the Select Range window.

5 In the System Identification app, drag and drop `data_est` to the **Working Data** rectangle, and drag and drop `data_val` to the **Validation Data** rectangle.

6 To get information about a data set, right-click its icon. For example, right-click `data_est` to open the Data/model Info dialog box.
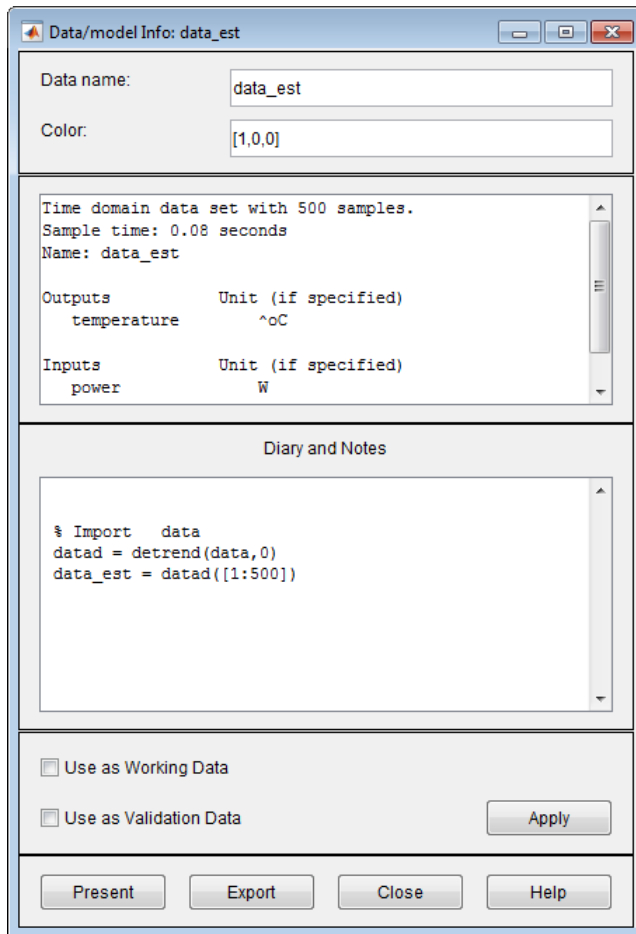
You can also change certain values in the Data/model Info dialog box, including:

*   Changing the name of the data set in the **Data name** field.

*   Changing the color of the data icon in the **Color** field. You specify colors using RGB values (relative amounts of red, green, and blue). Each value is between 0 and 1. For example, [1,0,0] indicates that only red is present, and no green and blue are mixed into the overall color.

*   Viewing or editing the commands executed on this data set in the **Diary and Notes** area. This area contains the command-line equivalent to the processing you performed using the System Identification app. For example, as shown in the Data/model Info: estimate window, the data_est data set is a result of importing the data, detrending the mean values, and selecting the first 500 samples of the data.

```
% Import   data
datad = detrend(data,0)
data_est = datad([1:500])
```

For more information about these and other toolbox commands, see the corresponding reference pages.

The Data/model Info dialog box also displays the total number of samples, the sample time, and the output and input channel names and units. This information is not editable.

Tip As an alternative shortcut, you can select **Preprocess > Quick start** from the System Identification app to perform all of the data processing steps in this tutorial.

Learn More

For information about supported data processing operations, such as resampling and filtering the data, see "Preprocess Data".

Saving the Session

After you process the data, as described in "Plotting and Processing Data" on page 3-6, you can delete any data sets in the window that you do not need for estimation and validation, and save your session. You can open this session later and use it as a starting point for model estimation and validation without repeating these preparatory steps.

You must have already processed the data into the System Identification app, as described in "Plotting and Processing Data" on page 3-6.

To delete specific data sets from a session and save the session:

1   In the System Identification app:

    a   Drag and drop the `data` data set into **Trash**.

    b   Drag and drop the `datad` data set into **Trash**.

    Alternatively, you can press the **Delete** key on your keyboard to move the data sets to **Trash**.

Note Moving items to the **Trash** does not delete them. To permanently delete items, select **Options > Empty trash**.

The following figure shows the System Identification app after moving the items to **Trash**.
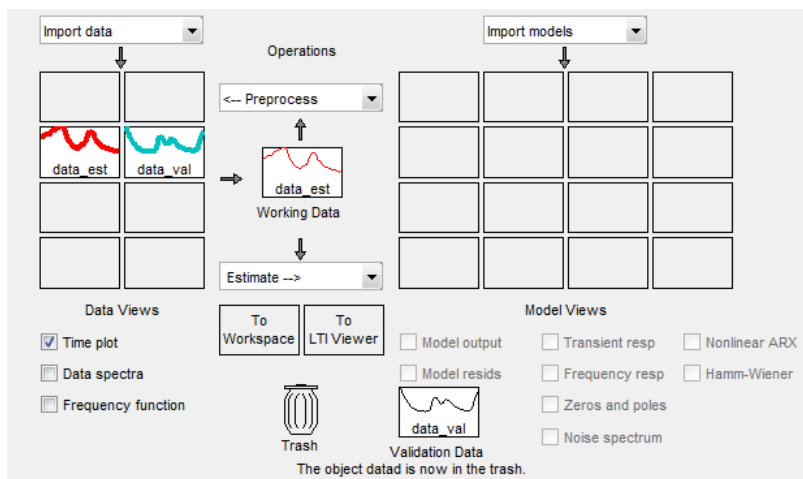
2   Drag and drop the `data_est` and `data_val` data sets to fill the empty rectangles, as shown in the following figure.
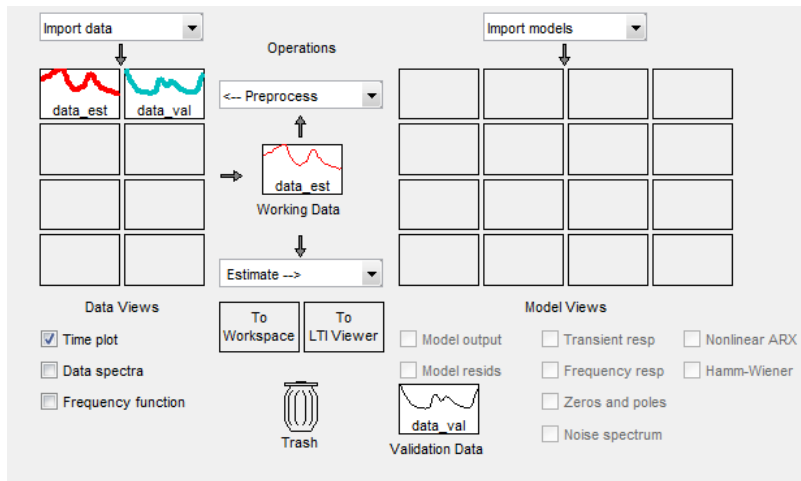
3. Select **File** > **Save session as** to open the Save Session dialog box, and browse to the folder where you want to save the session file.

4. In the **File name** field, type the name of the session `dryer2_processed_data`, and click **Save**. The resulting file has a `.sid` extension.

Tip You can open a saved session when starting the System Identification app by typing the following command at the MATLAB prompt:

```
systemIdentification('dryer2_processed_data')
```

For more information about managing sessions, see "Starting and Managing Sessions".

Estimating Linear Models Using Quick Start

How to Estimate Linear Models Using Quick Start

You can use the Quick Start feature in the System Identification app to estimate linear models. Quick Start might produce the final linear models you decide to use, or provide you with information required to configure the estimation of accurate parametric models, such as time constants, input delays, and resonant frequencies.

You must have already processed the data for estimation, as described in "Plotting and Processing Data" on page 3-6.

In the System Identification app , select **Estimate > Quick start**.

This action generates plots of step response, frequency-response, and the output of state-space and polynomial models. For more information about these plots, see "Validating the Quick Start Models" on page 3-16.
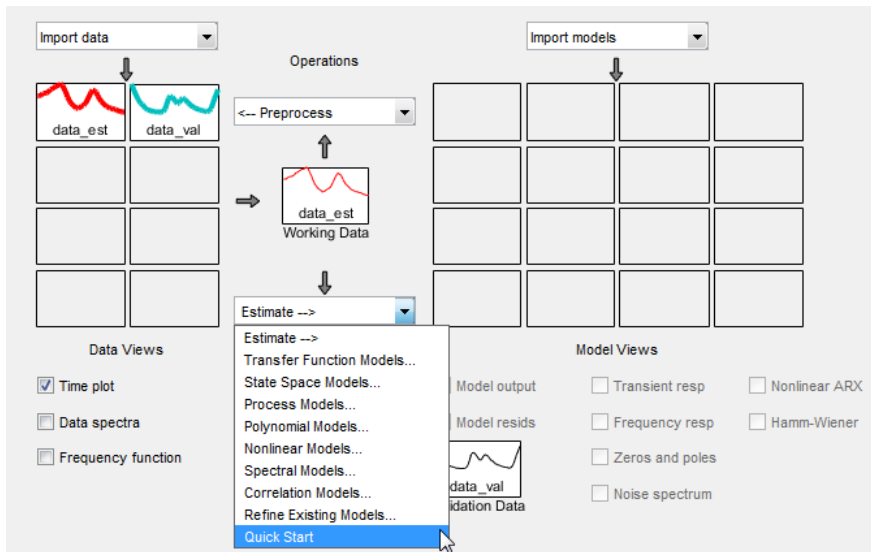
Types of Quick Start Linear Models

Quick Start estimates the following four types of models and adds the following to the System Identification app with default names:

- `imp` — Step response over a period of time using the `impulseest` algorithm.
- `spad` — Frequency response over a range of frequencies using the `spa` algorithm. The frequency response is the Fourier transform of the impulse response of a linear system.

  By default, the model is evaluated at 128 frequency values, ranging from 0 to the Nyquist frequency.
- `arxqs` — Fourth-order autoregressive (ARX) model using the `arx` algorithm.

  This model is parametric and has the following structure:

$$y(t) + a_1 y(t-1) + ... + a_{na} y(t - n_a) =$$
$$\qquad b_1 u(t - n_k) + ... + b_{nb} u(t - n_k - n_b + 1) + e(t)$$

  $y(t)$ represents the output at time t, $u(t)$ represents the input at time t, n_a is the number of poles, n_b is the number of b parameters (equal to the number of zeros plus 1), n_k is the number of samples before the input affects output of the system (called the *delay* or *dead time* of the model), and $e(t)$ is the white-noise disturbance. System Identification Toolbox software estimates the parameters $a_1...a_n$ and $b_1...b_n$ using the input and output data from the estimation data set.

  In `arxqs`, $n_a = n_b = 4$, and n_k is estimated from the step response model `imp`.
- `n4s3` — State-space model calculated using `n4sid`. The algorithm automatically selects the model order (in this case, 3).

  This model is parametric and has the following structure:

$$\frac{dx}{dt} = Ax(t) + Bu(t) + Ke(t)$$
$$y(t) = Cx(t) + Du(t) + e(t)$$

y(t) represents the output at time *t*, *u(t)* represents the input at time *t*, *x* is the state vector, and *e(t)* is the white-noise disturbance. The System Identification Toolbox product estimates the state-space matrices *A*, *B*, *C*, *D*, and *K*.

Note The Quick Start option does not create a transfer function model or a process model which can also be good starting model types.

Validating the Quick Start Models

Quick Start generates the following plots during model estimation to help you validate the quality of the models:

- Step-response plot
- Frequency-response plot
- Model-output plot

You must have already estimated models using Quick Start to generate these plots, as described in "How to Estimate Linear Models Using Quick Start" on page 3-14.

Step-Response Plot

The following step-response plot shows agreement among the different model structures and the measured data, which means that all of these structures have similar dynamics.

Tip If you closed the plot window, select the **Transient resp** check box to reopen this window. If the plot is empty, click the model icons in the System Identification app window to display the models on the plot.
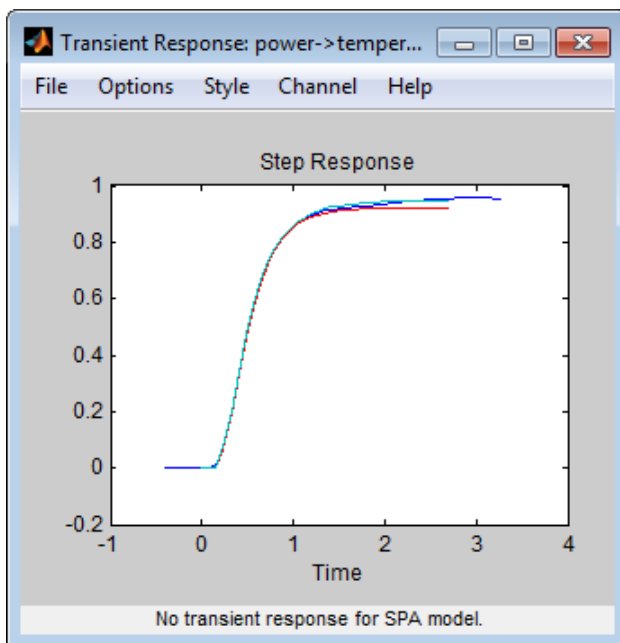
Step Response for imp, arxqs, and n4s3

Tip You can use the step-response plot to estimate the dead time of linear systems. For example, the previous step-response plot shows a time delay of about 0.25 s before the system responds to the input. This response delay, or *dead time*, is approximately equal to about three samples because the sample time is `0.08` s for this data set.

Frequency-Response Plot

The following frequency-response plot shows agreement among the different model structures and the measured data, which means that all of these structures have similar dynamics.

Tip If you closed this plot window, select the **Frequency resp** check box to reopen this window. If the plot is empty, click the model icons in the System Identification app window to display the models on the plot.
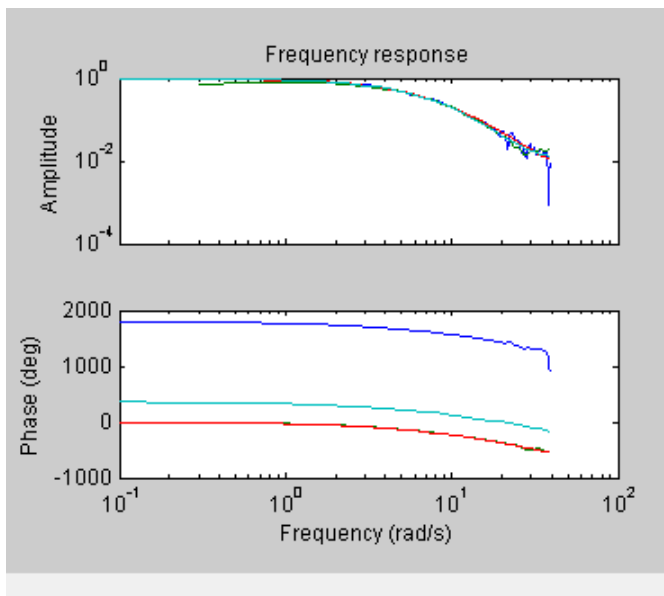
Frequency Response for Models imp, spad, arxqs, and n4s3

Model-Output Plot

The Model Output window shows agreement among the different model structures and the measured output in the validation data.

Tip If you closed the Model Output window, select the **Model output** check box to reopen this window. If the plot is empty, click the model icons in the System Identification app window to display the models on the plot.
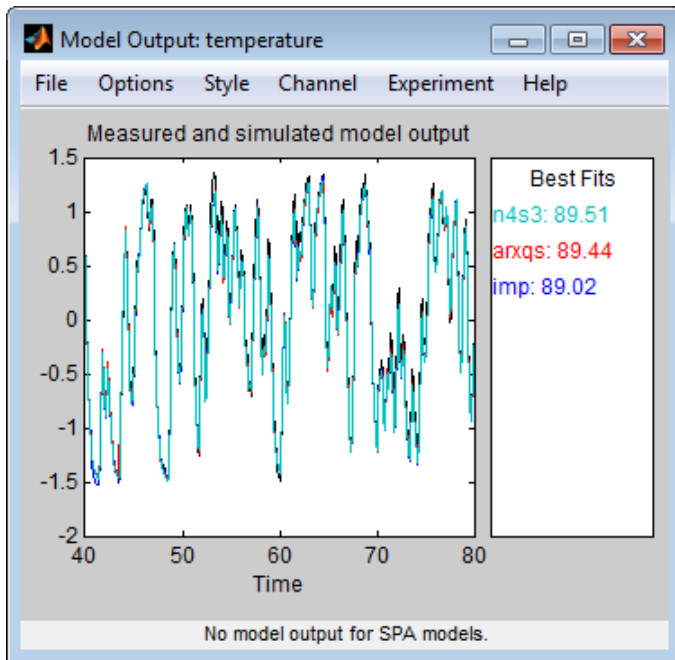
Measured Output and Model Output for Models imp, arxqs, and n4s3

The model-output plot shows the model response to the input in the validation data. The fit values for each model are summarized in the **Best Fits** area of the Model Output window. The models in the **Best Fits** list are ordered from best at the top to worst at the bottom. The fit between the two curves is computed such that 100 means a perfect fit, and 0 indicates a poor fit (that is, the model output has the same fit to the measured output as the mean of the measured output).

In this example, the output of the models matches the validation data output, which indicates that the models seem to capture the main system dynamics and that linear modeling is sufficient.

Tip To compare predicted model output instead of simulated output, select this option from the **Options** menu in the Model Output window.

Estimating Linear Models

Strategy for Estimating Accurate Models

The linear models you estimated in "Estimating Linear Models Using Quick Start" on page 3-14 showed that a linear model sufficiently represents the dynamics of the system.

In this portion of the tutorial, you get accurate parametric models by performing the following tasks:

1 Identifying initial model orders and delays from your data using a simple, polynomial model structure (ARX).

2 Exploring more complex model structures with orders and delays close to the initial values you found.

The resulting models are discrete-time models.

Estimating Possible Model Orders

To identify black-box models, you must specify the model order. However, how can you tell what model orders to specify for your black-box models? To answer this question, you can estimate simple polynomial (ARX) models for a range of orders and delays and compare the performance of these models. You choose the orders and delays that correspond to the best model fit as an initial guess for more accurate modeling using various model structures such as transfer function and state-space models.

About ARX Models

For a single-input/single-output system (SISO), the ARX model structure is:

$$y(t) + a_1 y(t - 1) + ... + a_{na} y(t - n_a) =$$
$$b_1 u(t - n_k) + ... + b_{nb} u(t - n_k - n_b + 1) + e(t)$$

y(t) represents the output at time *t*, *u(t)* represents the input at time *t*, n_a is the number of poles, n_b is the number of zeros plus 1, n_k is the input delay—the number of samples before the input affects the system output (called *delay* or *dead time* of the model), and *e(t)* is the white-noise disturbance.

You specify the model orders n_a, n_b, and n_k to estimate ARX models. The System Identification Toolbox product estimates the parameters $a_1...a_n$ and $b_1...b_n$ from the data.

How to Estimate Model Orders

1   In the System Identification app, select **Estimate > Polynomial Models** to open the Polynomial Models dialog box.
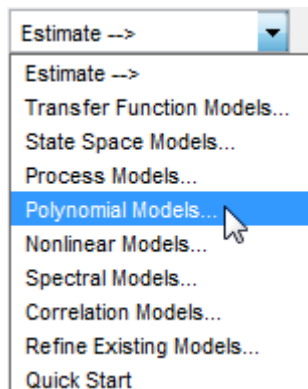
2   From the **Structure** list, select ARX: [na nb nk]. By default, this is already selected.
3   Edit the **Orders** field to try all combinations of poles, zeros, and delays, where each value is from 1 to 10:

    [1:10 1:10 1:10]
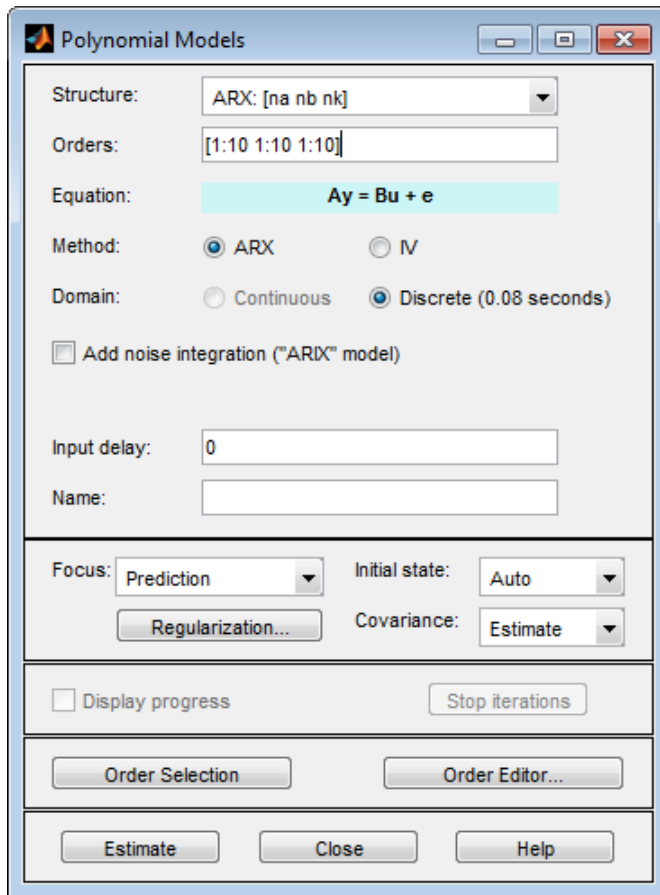
4 Click **Estimate** to open the ARX Model Structure Selection window, which displays the model performance for each combination of model parameters.

You use this plot to select the best-fit model.

- The horizontal axis is the total number of parameters — $n_a + n_b$.

- The vertical axis, called **Unexplained output variance (in %)**, is the portion of the output not explained by the model—the ARX model prediction error for the number of parameters shown on the horizontal axis.

  The *prediction error* is the sum of the squares of the differences between the validation data output and the model one-step-ahead predicted output.

- n_k is the delay.

Three rectangles are highlighted on the plot in green, blue, and red. Each color indicates a type of best-fit criterion, as follows:

- Red — Best fit minimizes the sum of the squares of the difference between the validation data output and the model output. This rectangle indicates the overall best fit.

- Green — Best fit minimizes Rissanen MDL criterion.

- Blue — Best fit minimizes Akaike AIC criterion.

In this tutorial, the **Unexplained output variance (in %)** value remains approximately constant for the combined number of parameters from 4 to 20. Such constancy indicates that model performance does not improve at higher orders. Thus, low-order models might fit the data equally well.

Note When you use the same data set for estimation and validation, use the MDL and AIC criteria to select model orders. These criteria compensate for overfitting that results from using too many parameters. For more information about these criteria, see the `selstruc` reference page.

---

5   In the ARX Model Structure Selection window, click the red bar (corresponding to 15 on the horizontal axis), and click **Insert**. This selection inserts n_a=6, n_b=9, and n_k=2 into the Polynomial Models dialog box and performs the estimation.

This action adds the model `arx692` to the System Identification app and updates the plots to include the response of the model.

> **Note** The default name of the parametric model contains the model type and the number of poles, zeros, and delays. For example, `arx692` is an ARX model with n_a=6, n_b=9, and a delay of two samples.

6   In the ARX Model Structure Selection window, click the third bar corresponding to 4 parameters on the horizontal axis (the lowest order that still gives a good fit), and click **Insert**.

- This selection inserts n_a=2, n_b=2, and n_k=3 (a delay of three samples) into the Polynomial Models dialog box and performs the estimation.
- The model `arx223` is added to the System Identification app and the plots are updated to include its response and output.

7   Click **Close** to close the ARX Model Structure Selection window.

8   Click **Close** to close the Polynomial Models dialog box.

Identifying Transfer Function Models

By estimating ARX models for different order combinations, as described in "Estimating Possible Model Orders" on page 3-19, you identified the number of poles, zeros, and delays that provide a good starting point for systematically exploring different models.

The overall best fit for this system corresponds to a model with six poles, nine zeros, and a delay of two samples. It also showed that a low-order model with n_a = 2 (two poles), n_b = 2 (one zero), and n_k = 3 (input-output delay) also provides a good fit. Thus, you should explore model orders close to these values.

In this portion of the tutorial, you estimate a transfer function model.

About Transfer Function Models

The general transfer function model structure is:

$$Y(s) = \frac{num(s)}{den(s)} U(s) + E(s)$$

$Y(s)$, $U(s)$ and $E(s)$ represent the Laplace transforms of the output, input and error, respectively. $num(s)$ and $den(s)$ represent the numerator and denominator polynomials that define the relationship between the input and the output. The roots of the denominator polynomial are referred to as the model *poles*. The roots of the numerator polynomial are referred to as the model *zeros*.

You must specify the number of poles and zeros to estimate a transfer function model. The System Identification Toolbox product estimates the numerator and denominator polynomials, and input/output delays from the data.

The transfer function model structure is a good choice for quick estimation because it requires that you specify only 2 parameters to get started: `np` is the number of poles and `nz` is the number of zeros.

How to Estimate Transfer Function Models

1   In the System Identification app, select **Estimate** > **Transfer Function Models** to open the Transfer Functions dialog box.
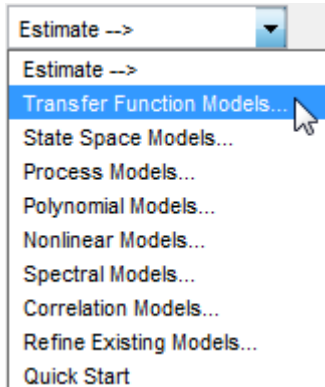
2   In the Estimate Transfer Functions dialog box, specify the following options:

- **Number of poles** — Leave the default value 2 to specify a second order function, for two poles.
- **Number of zeros** — Leave the default value 1.
- **Continuous-time** — Leave this checked.

3   Click **Delay** to expand the input/output delay specification area.

By estimating ARX models for different order combinations, as described in "Estimating Possible Model Orders" on page 3-19, you identified a 3 sample delay (`nk = 3`). This delay translates to a continuous-time delay of `(nk-1)*Ts`, which is equal to 0.16 seconds.

Specify **Delay** as `0.16` seconds. Leave **Fixed** checked.

Use the default Estimation Options. By default, the app assigns the name `tf1` to the model. The dialog box should look like this.
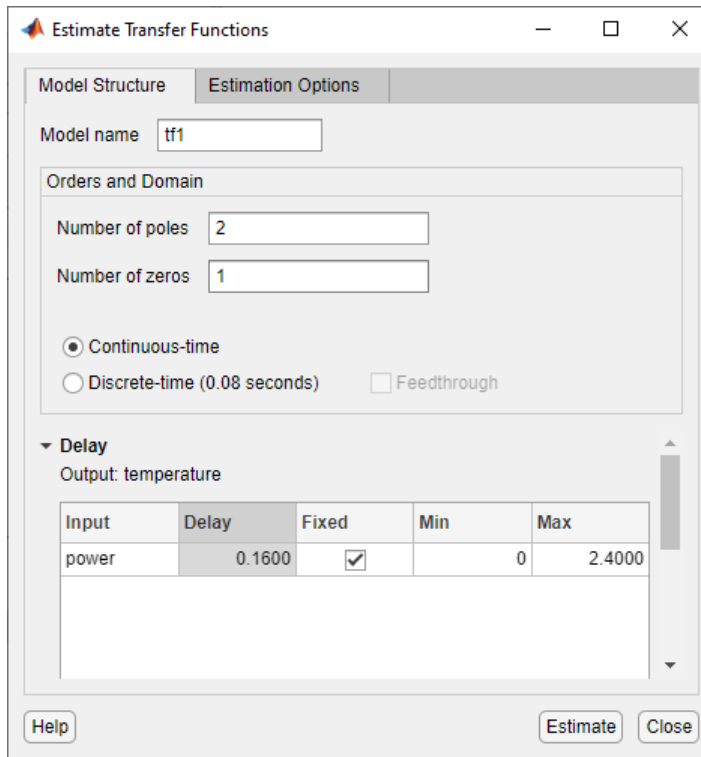
4 Click **Estimate** to add a transfer function model called `tf1` to the System Identification app. You can view the output of the estimation of the transfer function model in comparison with the estimations of other models, in the Model output window.
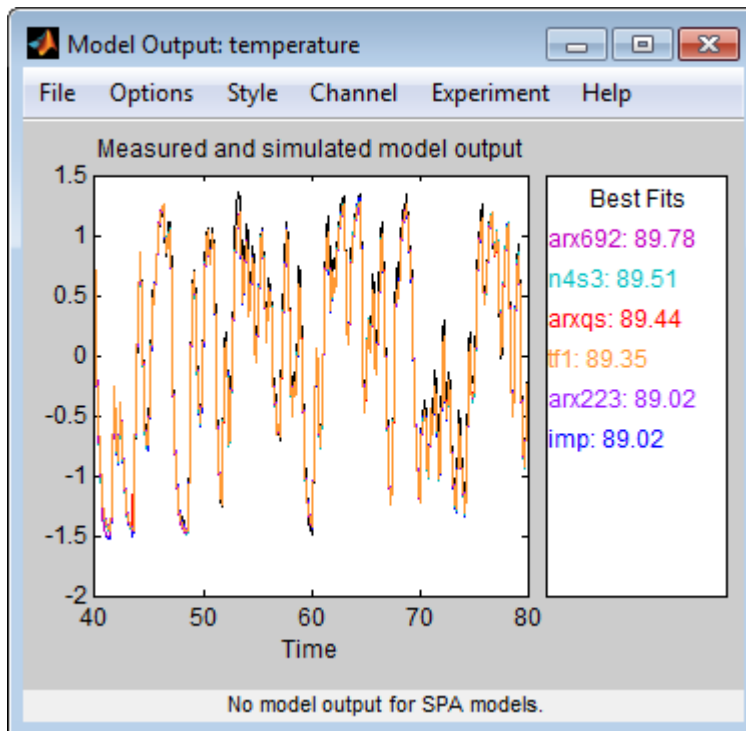
> **Tip** If you closed the Model Output window, you can regenerate it by selecting the **Model output** check box in the System Identification app. If the new model does not appear on the plot, click the model icon in the System Identification app to make the model active.

5 Click **Close** to close the Transfer Functions dialog box.

Learn More

To learn more about identifying transfer function models, see "Transfer Function Models".

Identifying State-Space Models

By estimating ARX models for different order combinations, as described in "Estimating Possible Model Orders" on page 3-19, you identified the number of poles, zeros, and delays that provide a good starting point for systematically exploring different models.

The overall best fit for this system corresponds to a model with six poles, nine zeros, and a delay of two samples. It also showed that a low-order model with n_a=2 (two poles), n_b=2 (one zero), and n_k=3 (input-output delay) also provides a good fit. Thus, you should explore model orders close to these values.

In this portion of the tutorial, you estimate a state-space model.

About State-Space Models

The general state-space model structure (innovation form) is:

$$\frac{dx}{dt} = Ax(t) + Bu(t) + Ke(t)$$

$$y(t) = Cx(t) + Du(t) + e(t)$$

y(t) represents the output at time *t*, *u(t)* represents the input at time *t*, *x(t)* is the state vector at time *t*, and *e(t)* is the white-noise disturbance.

You must specify a single integer as the model order (dimension of the state vector) to estimate a state-space model. The System Identification Toolbox product estimates the state-space matrices *A*, *B*, *C*, *D*, and *K* from the data.

The state-space model structure is a good choice for quick estimation because it requires that you specify only the number of states (which equals the number of poles). You can optionally also specify the delays and feedthrough behavior.

How to Estimate State-Space Models

1 In the System Identification app, select **Estimate** > **State Space Models** to open the State Space Models dialog box.
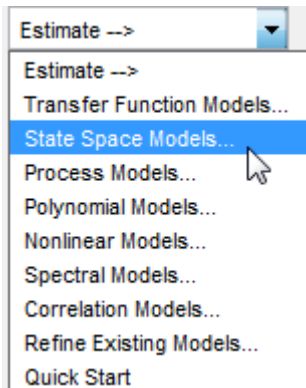
2 In the **Specify value** field of the **Model Structure** tab, specify the model order. Type 6 to create a sixth-order state-space model.

This choice is based on the fact that the best-fit ARX model has six poles.

Tip Although this tutorial estimates a sixth-order state-space model, you might want to explore whether a lower-order model adequately represents the system dynamics.

The dialog box should look like this:
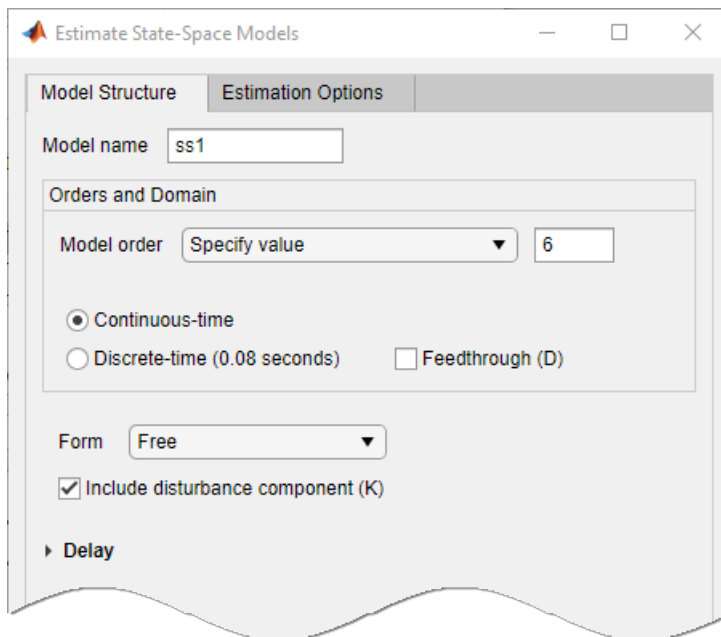
3 Select the **Estimation Options** tab to display additional options.

4 Change **Estimation Focus** to Simulation to optimize the model to use for output simulation.

The State Space Models dialog box looks like the following figure.
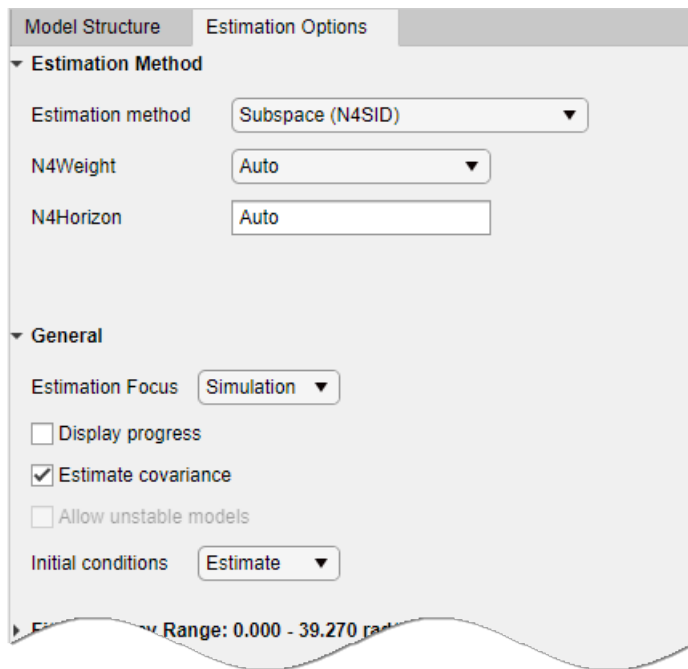
5 Click **Estimate** to add a state-space model called `ss1` to the System Identification app.

You can view the output of the estimation of the state-space model in comparison with the estimations of other models, in the Model output window.
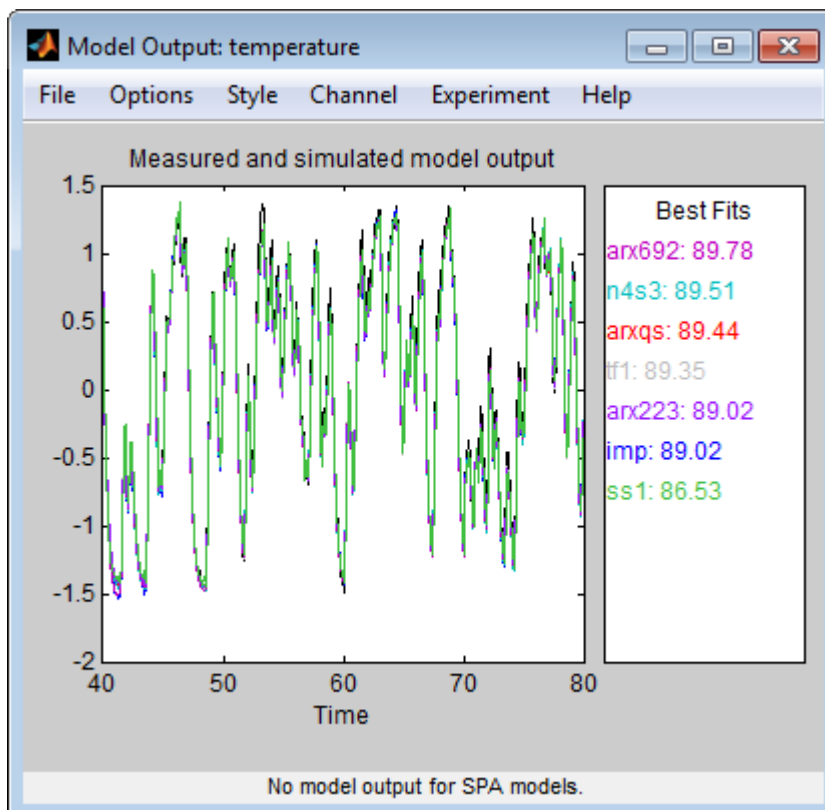
Tip If you closed the Model Output window, you can regenerate it by selecting the **Model output** check box in the System Identification app. If the new model does not appear on the plot, click the model icon in the System Identification app to make the model active.

---

6 Click **Close** to close the State Space Models dialog box.

Learn More

To learn more about identifying state-space models, see "State-Space Models".

Identifying ARMAX Models

By estimating ARX models for different order combinations, as described in "Estimating Possible Model Orders" on page 3-19, you identified the number of poles, zeros, and delays that provide a good starting point for systematically exploring different models.

The overall best fit for this system corresponds to a model with six poles, nine zeros, and a delay of two samples. It also showed that a low-order model with n_a=2 (two poles), n_b=2 (one zero), and n_k=3 also provides a good fit. Thus, you should explore model orders close to these values.

In this portion of the tutorial, you estimate an ARMAX input-output polynomial model.

About ARMAX Models

For a single-input/single-output system (SISO), the ARMAX polynomial model structure is:

$$y(t) + a_1 y(t-1) + ... + a_{na} y(t-n_a) =$$
$$b_1 u(t-n_k) + ... + b_{nb} u(t-n_k-n_b+1) +$$
$$e(t) + c_1 e(t-1) + ... + c_{nc} e(t-n_c)$$

$y(t)$ represents the output at time t, $u(t)$ represents the input at time t, n_a is the number of poles for the dynamic model, n_b is the number of zeros plus 1, n_c is the number of poles for the disturbance model, n_k is the number of samples before the input affects output of the system (called the *delay* or *dead time* of the model), and $e(t)$ is the white-noise disturbance.

Note The ARMAX model is more flexible than the ARX model because the ARMAX structure contains an extra polynomial to model the additive disturbance.

You must specify the model orders to estimate ARMAX models. The System Identification Toolbox product estimates the model parameters $a_1...a_n$, $b_1...b_n$, and $c_1...c_n$ from the data.

How to Estimate ARMAX Models

1 In the System Identification app , select **Estimate > Polynomial Models** to open the Polynomial Models dialog box.
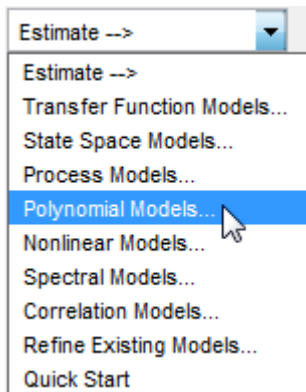
2 From the **Structure** list, select ARMAX: [na nb nc nk] to estimate an ARMAX model.
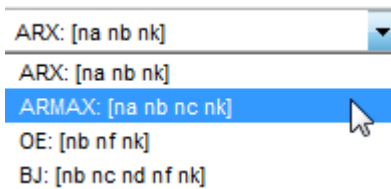
3 In the **Orders** field, set the orders *na*, *nb*, *nc*, and *nk* to the following values:

[2 2 2 2]

The app assigns the name to the model amx2222, by default, visible in the **Name** field.
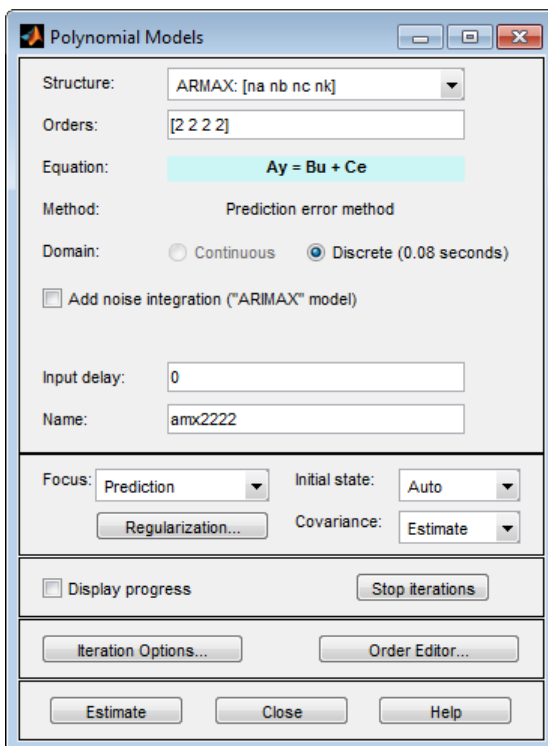
4 Click **Estimate** to add the ARMAX model to the System Identification app.

5 Repeat steps 3 and 4 using higher **Orders** 3 3 2 2. These orders result in a model that fits the data almost as well as the higher order ARX model `arx692`.
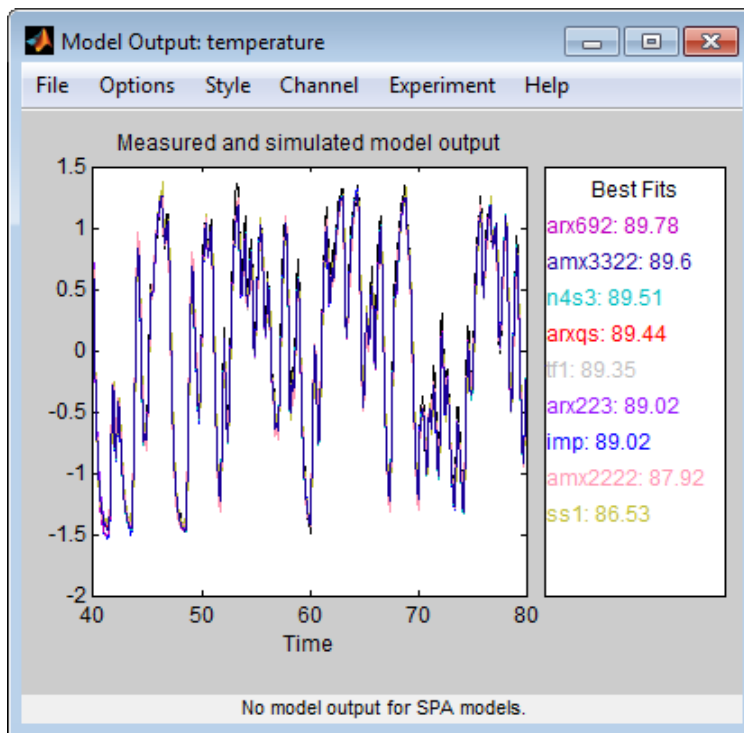
Tip If you closed the Model Output window, you can regenerate it by selecting the **Model output** check box in the System Identification app. If the new model does not appear on the plot, click the model icon in the System Identification app to make the model active.

6 Click **Close** to close the Polynomial Models dialog box.

Learn More

To learn more about identifying input-output polynomial models, such as ARMAX, see "Input-Output Polynomial Models".

Choosing the Best Model

You can compare models to choose the model with the best performance.

You must have already estimated the models, as described in "Estimating Linear Models" on page 3-18.

Summary of Models

The following figure shows the System Identification app, which includes all the estimated models in "Estimating Linear Models" on page 3-18.
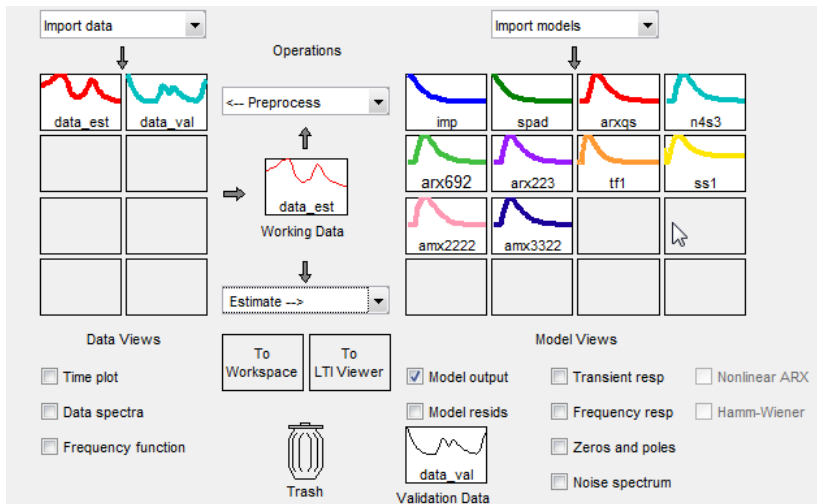
Examining the Model Output

Examine the model output plot to see how well the model output matches the measured output in the validation data set. A good model is the simplest model that best describes the dynamics and successfully simulates or predicts the output for different inputs. Models are listed by name in the **Best Fits** area of the Model Output plot. Note that one of the simpler models, `amx3322`, produced a similar fit as the highest-order model you created, `arx692`.
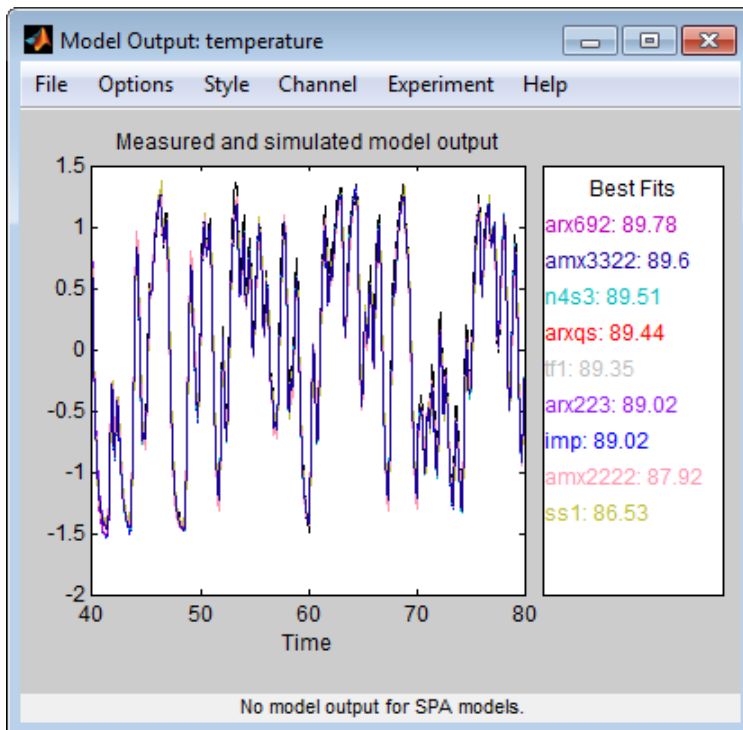
Tip If you closed the Model Output window, you can regenerate it by selecting the **Model output** check box in the System Identification app. If the new model does not appear on the plot, click the model icon in the System Identification app to make the model active.

To validate your models using a different data set, you can drag and drop this data set into the **Validation Data** rectangle in the System Identification app. If you transform validation data into the frequency domain, the Model Output plot updates to show the model comparison in the frequency domain.

To get a closer look at how well these models fit the data, magnify a portion of the plot by clicking and dragging a rectangle around the region of interest, as shown in the following figure.
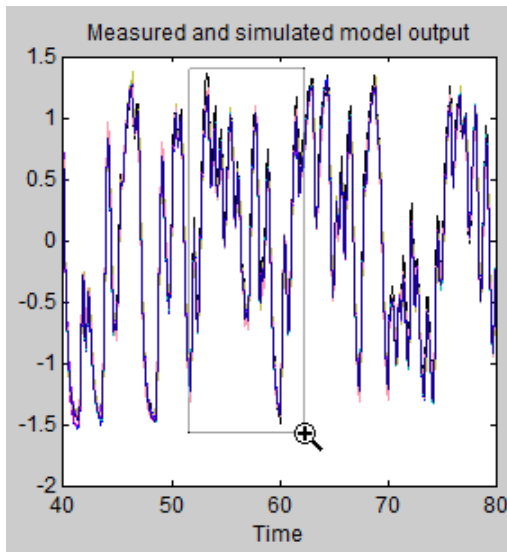
Releasing the mouse magnifies this region and shows that the output of all models matches the validation data well.
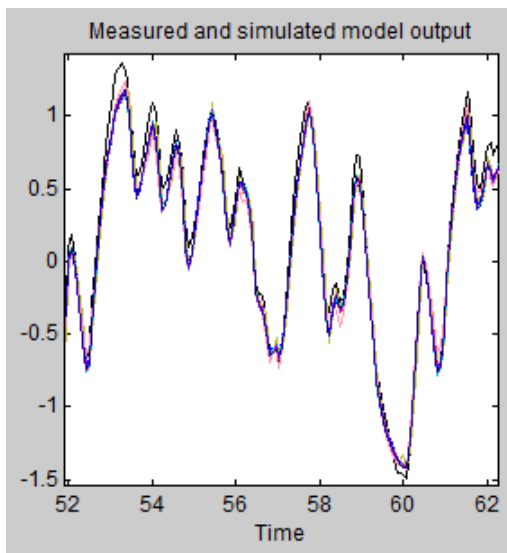
Viewing Model Parameters

Viewing Model Parameter Values

You can view the numerical parameter values for each estimated model.

You must have already estimated the models, as described in "Estimating Linear Models" on page 3-18.

To view the parameter values of the model amx3322, right-click the model icon in the System Identification app. The Data/model Info dialog box opens.
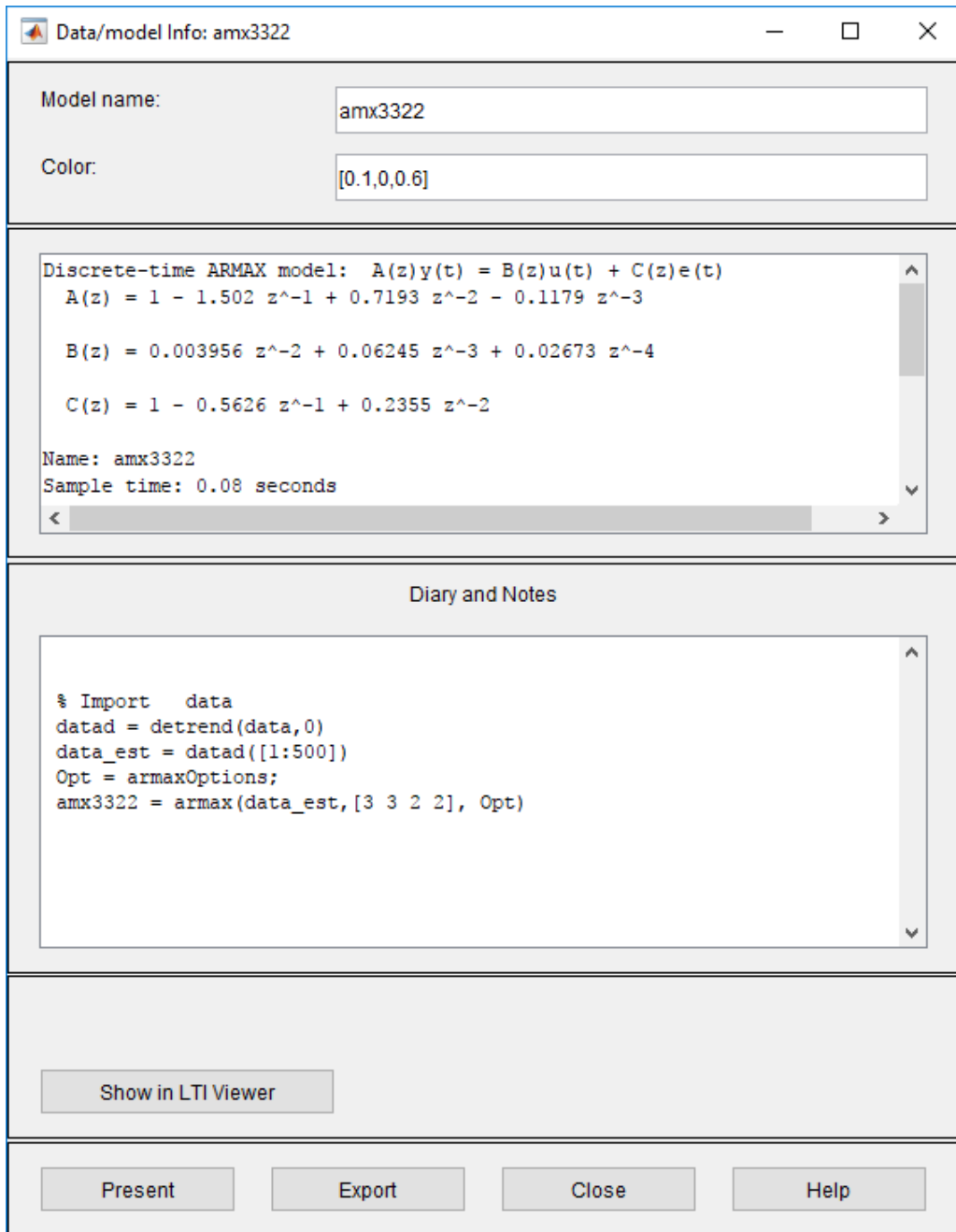
The noneditable area of the Data/model Info dialog box lists the parameter values correspond to the following difference equation for your system:

$$y(t) - 1.502y(t-1) + 0.7193y(t-2) - 0.1179y(t-3) =$$
$$0.003956u(t-2) + 0.06245u(t-3) + 0.02673u(t-4) + e(t) - 0.5626e(t-1) + 0.2355e(t-2)$$

Note The coefficient of *u(t-2)* is not significantly different from zero. This lack of difference explains why delay values of both 2 and 3 give good results.

Parameter values appear in the following format:

$$A(z) = 1 + a_1 z^{-1} + \ldots + a_{na} z^{-na}$$

$$B(z) = b_1 z^{-nk} + \ldots + b_{nb} z^{-nb-nk+1}$$

$$C(z) = 1 + c_1 z^{-1} + \ldots + c_{nc} z^{-nc}$$

The parameters appear in the ARMAX model structure, as follows:

$$A(q)y(t) = B(q)u(t) + C(q)e(t)$$

which corresponds to this general difference equation:

$$y(t) + a_1 y(t-1) + \ldots + a_{na} y(t - n_a) =$$
$$b_1 u(t - n_k) + \ldots + b_{nb} u(t - n_k - n_b + 1) +$$
$$e(t) + c_1 e(t-1) + \ldots + c_{nc} e(t - n_c)$$

y(t) represents the output at time *t*, *u(t)* represents the input at time *t*, n_a is the number of poles for the dynamic model, n_b is the number of zeros plus 1, n_c is the number of poles for the disturbance model, n_k is the number of samples before the input affects output of the system (called the *delay* or *dead time* of the model), and *e(t)* is the white-noise disturbance.

Viewing Parameter Uncertainties

You can view parameter uncertainties of estimated models.

You must have already estimated the models, as described in "Estimating Linear Models" on page 3-18.

To view parameter uncertainties, click **Present** in the Data/model Info dialog box, and view the model information at the MATLAB prompt.

```
amx3322 =
Discrete-time ARMAX model:  A(z)y(t) = B(z)u(t) + C(z)e(t)

  A(z) = 1 - 1.502 (+/- 0.05982) z^-1 + 0.7193 (+/- 0.0883) z^-2
                                - 0.1179 (+/- 0.03462) z^-3


  B(z) = 0.003956 (+/- 0.001551) z^-2 + 0.06245 (+/- 0.002372) z^-3
                              + 0.02673 (+/- 0.005651) z^-4


  C(z) = 1 - 0.5626 (+/- 0.07322) z^-1 + 0.2355 (+/- 0.05294) z^-2

Name: amx3322
Sample time: 0.08 seconds

Parameterization:
   Polynomial orders:    na=3   nb=3   nc=2   nk=2
   Number of free coefficients: 8
```

```
   Use "polydata", "getpvec", "getcov" for parameters and their uncertainties.

Status:
Termination condition: Near (local) minimum, (norm(g) < tol).
Number of iterations: 5, Number of function evaluations: 11

Estimated using POLYEST on time domain data "data_est".
Fit to estimation data: 95.3% (prediction focus)
FPE: 0.001596, MSE: 0.001546
More information in model's "Report" property.
```

The 1-standard deviation uncertainty for the model parameters is in parentheses next to each parameter value.
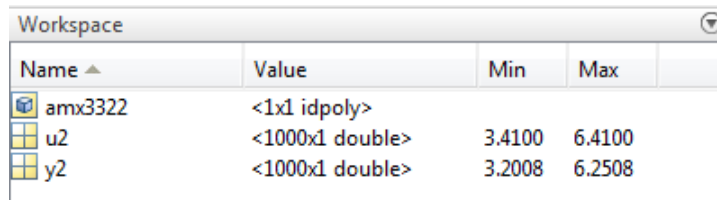
Exporting the Model to the MATLAB Workspace

The models you create in the System Identification app are not automatically available in the MATLAB workspace. To make a model available to other toolboxes, Simulink, and System Identification Toolbox commands, you must export your model from the System Identification app to the MATLAB workspace. For example, if the model is a plant that requires a controller, you can import the model from the MATLAB workspace into the Control System Toolbox product.

You must have already estimated the models, as described in "Estimating Linear Models" on page 3-18.

To export the `amx3322` model, drag it to the **To Workspace** rectangle in the System Identification app. Alternatively, click **Export** in the Data/model Info dialog box.

The model appears in the MATLAB Workspace browser.

Note This model is an `idpoly` model object.

After the model is in the MATLAB workspace, you can perform other operations on the model. For example, if you have the Control System Toolbox product installed, you might transform the model to a state-space object using:

```
ss_model=ss(amx3322)
```

Exporting the Model to the Linear System Analyzer

If you have the Control System Toolbox product installed, the **To Linear System Analyzer** rectangle appears in the System Identification app.

The Linear System Analyzer is a graphical user interface for viewing and manipulating the response plots of linear models. It displays the following plots:

- Step- and impulse-response
- Bode, Nyquist, and Nichols
- Frequency-response singular values
- Pole/zero
- Response to general input signals
- Unforced response starting from given initial states (only for state-space models)

To plot a model in the Linear System Analyzer, drag and drop the model icon to the **To Linear System Analyzer** rectangle in the System Identification app. Alternatively, click **Show in Linear System Analyzer** in the Data/model Info dialog box.

For more information about working with plots in the Linear System Analyzer, see "Linear System Analyzer Overview" (Control System Toolbox).

Identify Linear Models Using the Command Line

Introduction

Objectives

Estimate and validate linear models from multiple-input/single-output (MISO) data to find the one that best describes the system dynamics.

After completing this tutorial, you will be able to accomplish the following tasks using the command line:

- Create data objects to represent data.
- Plot the data.
- Process data by removing offsets from the input and output signals.
- Estimate and validate linear models from the data.
- Simulate and predict model output.

Note This tutorial uses time-domain data to demonstrate how you can estimate linear models. The same workflow applies to fitting frequency-domain data.

Data Description

This tutorial uses the data file `co2data.mat`, which contains two experiments of two-input and single-output (MISO) time-domain data from a steady-state that the operator perturbed from equilibrium values.

In the first experiment, the operator introduced a pulse wave to both inputs. In the second experiment, the operator introduced a pulse wave to the first input and a step signal to the second input.
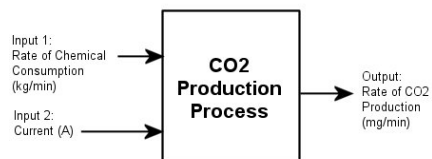
Preparing Data

Loading Data into the MATLAB Workspace

Load the data.

```
load co2data
```

This command loads the following five variables into the MATLAB Workspace:

- `Input_exp1` and `Output_exp1` are the input and output data from the first experiment, respectively.

- `Input_exp2` and `Output_exp2` are the input and output data from the second experiment, respectively.
- `Time` is the time vector from 0 to 1000 minutes, increasing in equal increments of `0.5` min.

For both experiments, the input data consists of two columns of values. The first column of values is the rate of chemical consumption (in kilograms per minute), and the second column of values is the current (in amperes). The output data is a single column of the rate of carbon-dioxide production (in milligrams per minute).

Plotting the Input/Output Data

Plot the input and output data from both experiments.

```
plot(Time,Input_exp1,Time,Output_exp1)
legend('Input 1','Input 2','Output 1')
figure
plot(Time,Input_exp2,Time,Output_exp2)
legend('Input 1','Input 2','Output 1')
```
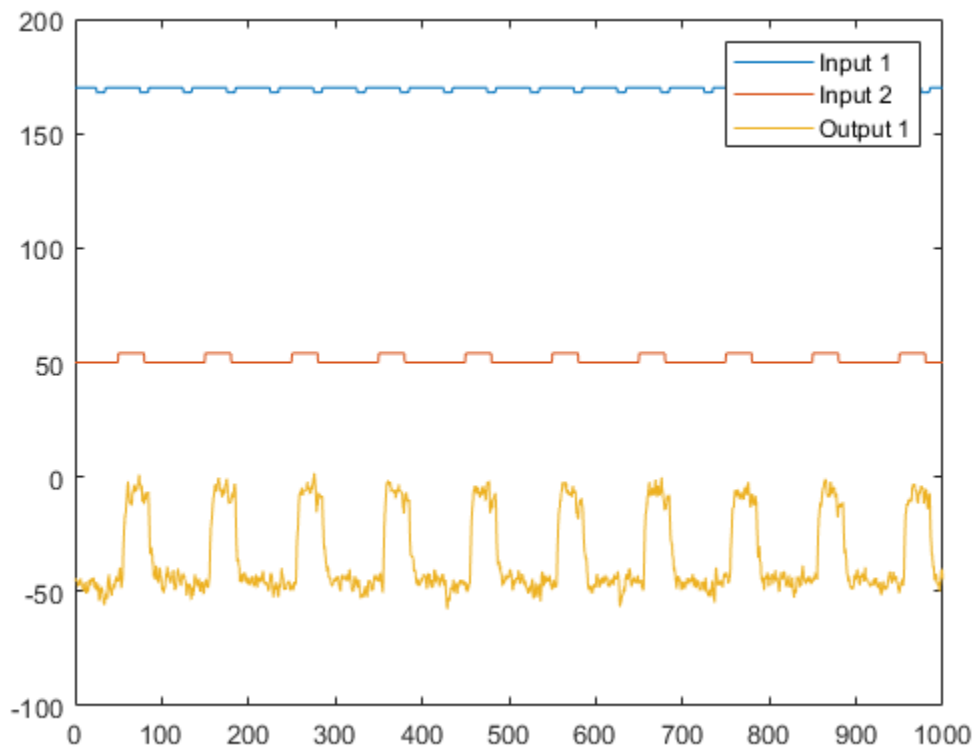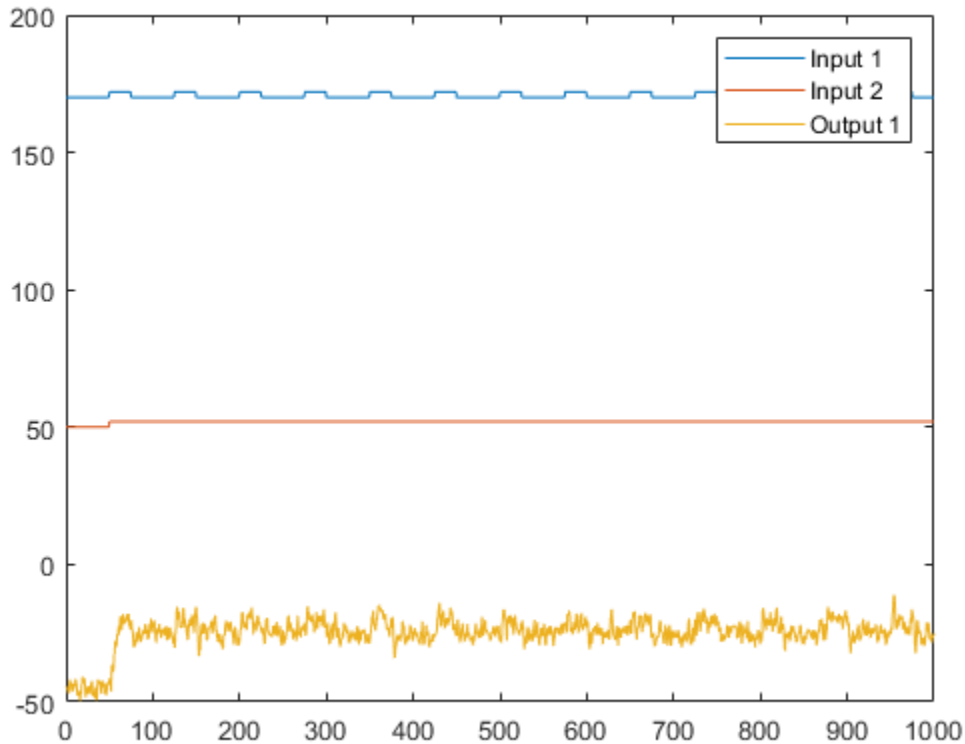
The first plot shows the first experiment, where the operator applies a pulse wave to each input to perturb it from its steady-state equilibrium.

The second plot shows the second experiment, where the operator applies a pulse wave to the first input and a step signal to the second input.

Removing Equilibrium Values from the Data

Plotting the data, as described in "Plotting the Input/Output Data" on page 3-38, shows that the inputs and the outputs have nonzero equilibrium values. In this portion of the tutorial, you subtract equilibrium values from the data.

The reason you subtract the mean values from each signal is because, typically, you build linear models that describe the responses for deviations from a physical equilibrium. With steady-state data, it is reasonable to assume that the mean levels of the signals correspond to such an equilibrium. Thus, you can seek models around zero without modeling the absolute equilibrium levels in physical units.

Zoom in on the plots to see that the earliest moment when the operator applies a disturbance to the inputs occurs after 25 minutes of steady-state conditions (or after the first 50 samples). Thus, the average value of the first 50 samples represents the equilibrium conditions.

Use the following commands to remove the equilibrium values from inputs and outputs in both experiments:

```
Input_exp1 = Input_exp1-...
    ones(size(Input_exp1,1),1)*mean(Input_exp1(1:50,:));
```

3-39

```
Output_exp1 = Output_exp1-...
    mean(Output_exp1(1:50,:));
Input_exp2 = Input_exp2-...
    ones(size(Input_exp2,1),1)*mean(Input_exp2(1:50,:));
Output_exp2 = Output_exp2-...
    mean(Output_exp2(1:50,:));
```

Using Objects to Represent Data for System Identification

The System Identification Toolbox data objects, `iddata` and `idfrd`, encapsulate data values and data properties into a single entity. You can use the System Identification Toolbox commands to conveniently manipulate these data objects as single entities.

In this portion of the tutorial, you create two `iddata` objects, one for each of the two experiments. You use the data from Experiment 1 for model estimation, and the data from Experiment 2 for model validation. You work with two independent data sets because you use one data set for model estimation and the other for model validation.

Note When two independent data sets are not available, you can split one data set into two parts, assuming that each part contains enough information to adequately represent the system dynamics.

Creating iddata Objects

You must have already loaded the sample data into the MATLAB workspace, as described in "Loading Data into the MATLAB Workspace" on page 3-37.

Use these commands to create two data objects, `ze` and `zv` :

```
Ts = 0.5; % Sample time is 0.5 min
ze = iddata(Output_exp1,Input_exp1,Ts);
zv = iddata(Output_exp2,Input_exp2,Ts);
```

`ze` contains data from Experiment 1 and `zv` contains data from Experiment 2. `Ts` is the sample time.

The `iddata` constructor requires three arguments for time-domain data and has the following syntax:

```
data_obj = iddata(output,input,sampling_interval);
```

To view the properties of an `iddata` object, use the `get` command. For example, type this command to get the properties of the estimation data:

```
get(ze)
```

```
ans =

  struct with fields:

            Domain: 'Time'
              Name: ''
        OutputData: [2001x1 double]
                 y: 'Same as OutputData'
        OutputName: {'y1'}
        OutputUnit: {''}
         InputData: [2001x2 double]
```

```
                u: 'Same as InputData'
        InputName: {2x1 cell}
        InputUnit: {2x1 cell}
           Period: [2x1 double]
      InterSample: {2x1 cell}
               Ts: 0.5000
           Tstart: []
  SamplingInstants: [2001x0 double]
         TimeUnit: 'seconds'
    ExperimentName: 'Exp1'
            Notes: {}
         UserData: []
```

To learn more about the properties of this data object, see the `iddata` reference page.

To modify data properties, you can use dot notation or the `set` command. For example, to assign channel names and units that label plot axes, type the following syntax in the MATLAB Command Window:

```matlab
% Set time units to minutes
ze.TimeUnit = 'min';
% Set names of input channels
ze.InputName = {'ConsumptionRate','Current'};
% Set units for input variables
ze.InputUnit = {'kg/min','A'};
% Set name of output channel
ze.OutputName = 'Production';
% Set unit of output channel
ze.OutputUnit = 'mg/min';

% Set validation data properties
zv.TimeUnit = 'min';
zv.InputName = {'ConsumptionRate','Current'};
zv.InputUnit = {'kg/min','A'};
zv.OutputName = 'Production';
zv.OutputUnit = 'mg/min';
```

You can verify that the `InputName` property of `ze` is changed, or "index" into this property:

```matlab
ze.inputname;
```

Tip Property names, such as `InputUnit`, are not case sensitive. You can also abbreviate property names that start with `Input` or `Output` by substituting u for `Input` and y for `Output` in the property name. For example, `OutputUnit` is equivalent to `yunit`.

Plotting the Data in a Data Object

You can plot `iddata` objects using the `plot` command.

Plot the estimation data.
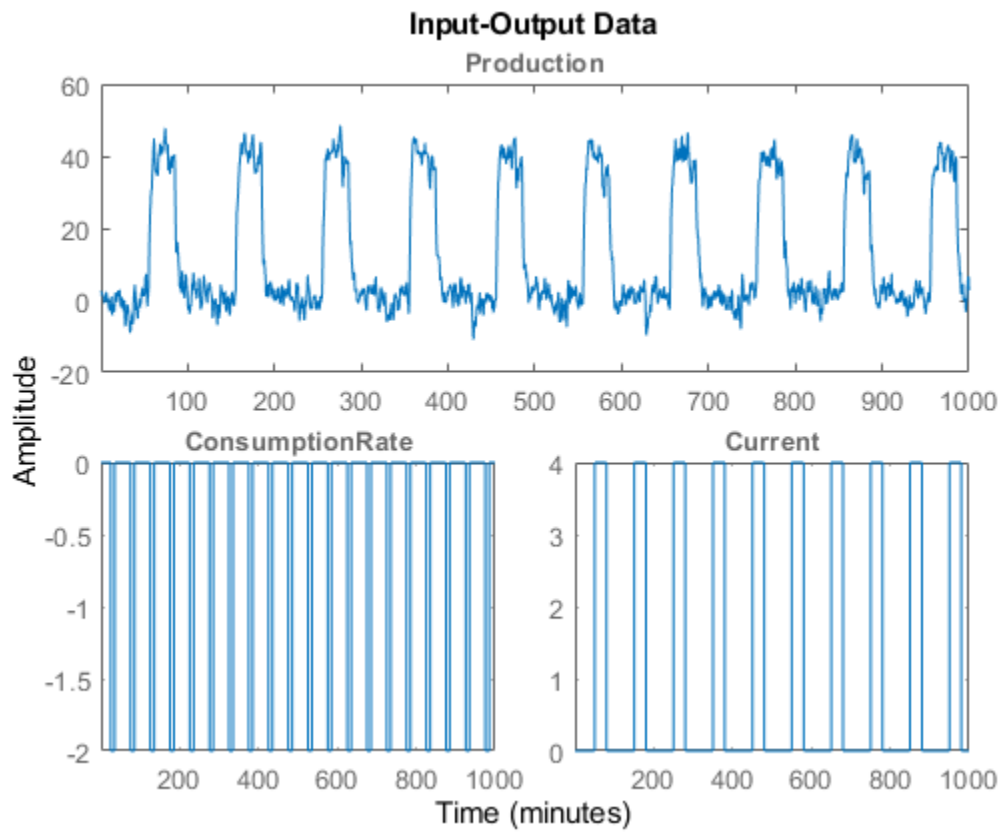
```matlab
plot(ze)
```

The bottom axes show inputs `ConsumptionRate` and `Current`, and the top axes show the output `ProductionRate`.

Plot the validation data in a new figure window.

```
figure    % Open a new MATLAB Figure window
plot(zv) % Plot the validation data
```
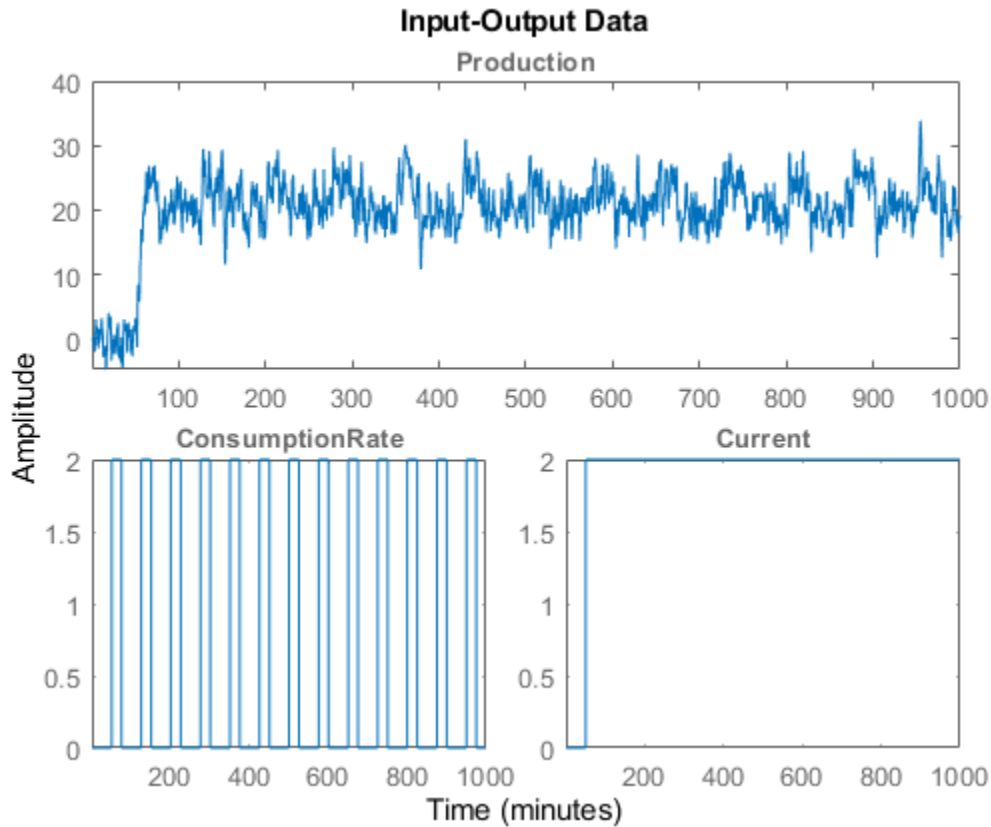
Zoom in on the plots to see that the experiment process amplifies the first input (`ConsumptionRate`) by a factor of 2, and amplifies the second input (`Current`) by a factor of 10.

Selecting a Subset of the Data

Before you begin, create a new data set that contains only the first 1000 samples of the original estimation and validation data sets to speed up the calculations.

```
Ze1 = ze(1:1000);
Zv1 = zv(1:1000);
```

For more information about indexing into `iddata` objects, see the corresponding reference page.

Estimating Impulse Response Models

Why Estimate Step- and Frequency-Response Models?

Frequency-response and step-response are *nonparametric* models that can help you understand the dynamic characteristics of your system. These models are not represented by a compact mathematical formula with adjustable parameters. Instead, they consist of data tables.

In this portion of the tutorial, you estimate these models using the data set `ze`. You must have already created `ze`, as described in "Creating iddata Objects" on page 3-40.

The response plots from these models show the following characteristics of the system:

- The response from the first input to the output might be a second-order function.
- The response from the second input to the output might be a first-order or an overdamped function.

Estimating the Frequency Response

The System Identification Toolbox product provides three functions for estimating the frequency response:

- `etfe` computes the empirical transfer function using Fourier analysis.
- `spa` estimates the transfer function using spectral analysis for a fixed frequency resolution.
- `spafdr` lets you specify a variable frequency resolution for estimating the frequency response.

Use `spa` to estimate the frequency response.

```
Ge = spa(ze);
```

Plot the frequency response as a Bode plot.

```
bode(Ge)
```
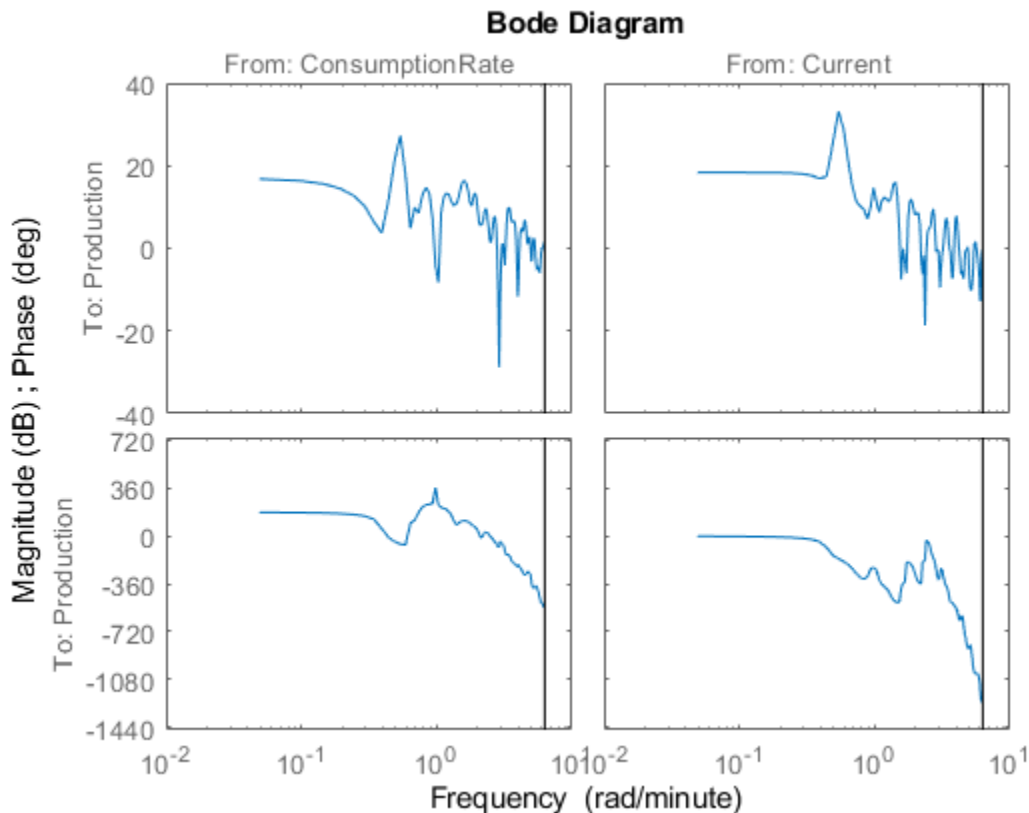

The amplitude peaks at the frequency of 0.54 rad/min, which suggests a possible resonant behavior (complex poles) for the first input-to-output combination - `ConsumptionRate` to `Production`.

In both plots, the phase rolls off rapidly, which suggests a time delay for both input/output combinations.

Estimating the Empirical Step Response

To estimate the step response from the data, first estimate a non-parametric impulse response model (FIR filter) from data and then plot its step response.

```
% model estimation
Mimp = impulseest(Ze1,60);

% step response
step(Mimp)
```
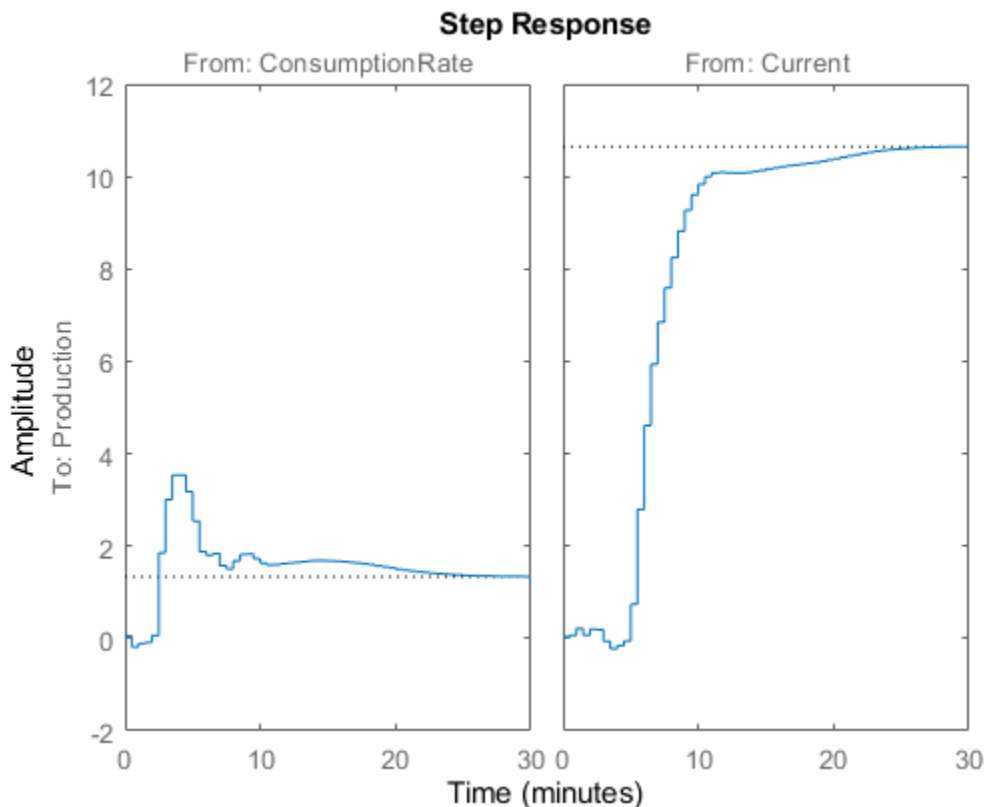

The step response for the first input/output combination suggests an overshoot, which indicates the presence of an underdamped mode (complex poles) in the physical system.

The step response from the second input to the output shows no overshoot, which indicates either a first-order response or a higher-order response with real poles (overdamped response).

The step-response plot indicates a nonzero delay in the system, which is consistent with the rapid phase roll-off you got in the Bode plot you created in "Estimating the Empirical Step Response" on page 3-45.

Estimating Delays in the Multiple-Input System

Why Estimate Delays?

To identify parametric black-box models, you must specify the input/output delay as part of the model order.

If you do not know the input/output delays for your system from the experiment, you can use the System Identification Toolbox software to estimate the delay.

Estimating Delays Using the ARX Model Structure

In the case of single-input systems, you can read the delay on the impulse-response plot. However, in the case of multiple-input systems, such as the one in this tutorial, you might be unable to tell which input caused the initial change in the output and you can use the `delayest` command instead.

The command estimates the time delay in a dynamic system by estimating a low-order, discrete-time ARX model with a range of delays, and then choosing the delay that corresponding to the best fit.

The ARX model structure is one of the simplest black-box parametric structures. In discrete-time, the ARX structure is a difference equation with the following form:

$$y(t) + a_1 y(t-1) + \ldots + a_{na} y(t - n_a) =$$
$$b_1 u(t - n_k) + \ldots + b_{nb} u(t - n_k - n_b + 1) + e(t)$$

$y(t)$ represents the output at time t, $u(t)$ represents the input at time t, n_a is the number of poles, n_b is the number of b parameters (equal to the number of zeros plus 1), n_k is the number of samples before the input affects output of the system (called the *delay* or *dead time* of the model), and $e(t)$ is the white-noise disturbance.

`delayest` assumes that $n_a = n_b = 2$ and that the noise e is white or insignificant, and estimates n_k.

To estimate the delay in this system, type:

```
delayest(ze)
```

```
ans =

     5    10
```

This result includes two numbers because there are two inputs: the estimated delay for the first input is 5 data samples, and the estimated delay for the second input is 10 data samples. Because the sample time for the experiments is 0.5 min, this corresponds to a 2.5 -min delay before the first input affects the output, and a 5.0 -min delay before the second input affects the output.

Estimating Delays Using Alternative Methods

There are two alternative methods for estimating the time delay in the system:

- Plot the time plot of the input and output data and read the time difference between the first change in the input and the first change in the output. This method is practical only for single-input/single-output system; in the case of multiple-input systems, you might be unable to tell which input caused the initial change in the output.

- Plot the impulse response of the data with a 1-standard-deviation confidence region. You can estimate the time delay using the time when the impulse response is first outside the confidence region.

Estimating Model Orders Using an ARX Model Structure

Why Estimate Model Order?

Model order is one or more integers that define the complexity of the model. In general, model order is related to the number of poles, the number of zeros, and the response delay (time in terms of the number of samples before the output responds to the input). The specific meaning of model order depends on the model structure.

To compute parametric black-box models, you must provide the model order as an input. If you do not know the order of your system, you can estimate it.

After completing the steps in this section, you get the following results:

- For the first input/output combination: n_a=2, n_b=2, and the delay n_k=5.
- For the second input/output combination: n_a=1, n_b=1, and the delay n_k=10.

Later, you explore different model structures by specifying model-order values that are slight variations around these initial estimate.

Commands for Estimating the Model Order

In this portion of the tutorial, you use `struc`, `arxstruc`, and `selstruc` to estimate and compare simple polynomial (ARX) models for a range of model orders and delays, and select the best orders based on the quality of the model.

The following list describes the results of using each command:

- `struc` creates a matrix of model-order combinations for a specified range of n_a, n_b, and n_k values.
- `arxstruc` takes the output from `struc`, systematically estimates an ARX model for each model order, and compares the model output to the measured output. `arxstruc` returns the *loss function* for each model, which is the normalized sum of squared prediction errors.
- `selstruc` takes the output from `arxstruc` and opens the ARX Model Structure Selection window, which resembles the following figure, to help you choose the model order.

  You use this plot to select the best-fit model.

- The horizontal axis is the total number of parameters — $n_a + n_b$.
- The vertical axis, called **Unexplained output variance (in %)**, is the portion of the output not explained by the model—the ARX model prediction error for the number of parameters shown on the horizontal axis.

  The *prediction error* is the sum of the squares of the differences between the validation data output and the model one-step-ahead predicted output.
- n_k is the delay.

Three rectangles are highlighted on the plot in green, blue, and red. Each color indicates a type of best-fit criterion, as follows:

- Red — Best fit minimizes the sum of the squares of the difference between the validation data output and the model output. This rectangle indicates the overall best fit.
- Green — Best fit minimizes Rissanen MDL criterion.
- Blue — Best fit minimizes Akaike AIC criterion.

In this tutorial, the **Unexplained output variance (in %)** value remains approximately constant for the combined number of parameters from 4 to 20. Such constancy indicates that model performance does not improve at higher orders. Thus, low-order models might fit the data equally well.

Note When you use the same data set for estimation and validation, use the MDL and AIC criteria to select model orders. These criteria compensate for overfitting that results from using too many parameters. For more information about these criteria, see the `selstruc` reference page.

Model Order for the First Input-Output Combination

In this tutorial, there are two inputs to the system and one output and you estimate model orders for each input/output combination independently. You can either estimate the delays from the two inputs simultaneously or one input at a time.

It makes sense to try the following order combinations for the first input/output combination:

- n_a=2:5
- n_b=1:5
- n_k=5

This is because the nonparametric models you estimated in "Estimating Impulse Response Models" on page 3-43 show that the response for the first input/output combination might have a second-order response. Similarly, in "Estimating Delays in the Multiple-Input System" on page 3-46, the delay for this input/output combination was estimated to be 5.

To estimate model order for the first input/output combination:

1 Use `struc` to create a matrix of possible model orders.

```
NN1 = struc(2:5,1:5,5);
```

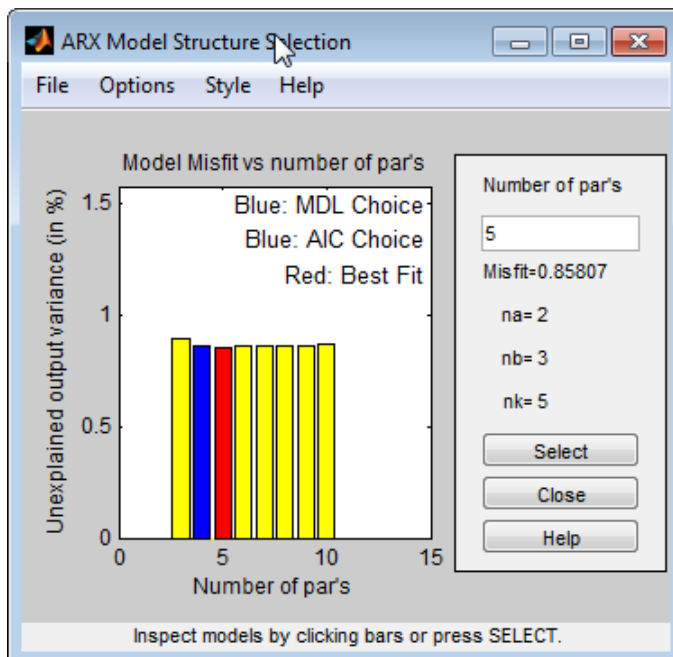**2** Use `selstruc` to compute the loss functions for the ARX models with the orders in `NN1`.

```
selstruc(arxstruc(ze(:,:,1),zv(:,:,1),NN1))
```

Note `ze(:,:,1)` selects the first input in the data.

This command opens the interactive ARX Model Structure Selection window.

> **Note** The Rissanen MDL and Akaike AIC criteria produces equivalent results and are both indicated by a blue rectangle on the plot.

The red rectangle represents the best overall fit, which occurs for n_a=2, n_b=3, and n_k=5. The height difference between the red and blue rectangles is insignificant. Therefore, you can choose the parameter combination that corresponds to the lowest model order and the simplest model.

3 Click the blue rectangle, and then click **Select** to choose that combination of orders:

n_a=2

n_b=2

n_k=5

4 To continue, press any key while in the MATLAB Command Window.

Model Order for the Second Input-Output Combination

It makes sense to try the following order combinations for the second input/output combination:

- n_a=1:3
- n_b=1:3
- n_k=10

This is because the nonparametric models you estimated in "Estimating Impulse Response Models" on page 3-43 show that the response for the second input/output combination might have a first-order response. Similarly, in "Estimating Delays in the Multiple-Input System" on page 3-46, the delay for this input/output combination was estimated to be 10.

To estimate model order for the second input/output combination:

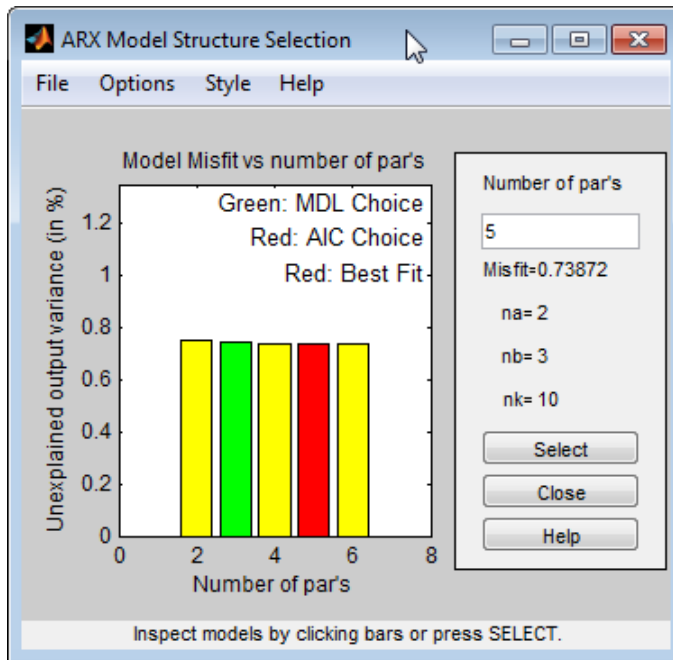**1** Use `struc` to create a matrix of possible model orders.

```
NN2 = struc(1:3,1:3,10);
```

2 Use `selstruc` to compute the loss functions for the ARX models with the orders in NN2.

```
selstruc(arxstruc(ze(:,:,2),zv(:,:,2),NN2))
```

This command opens the interactive ARX Model Structure Selection window.

Note The Akaike AIC and the overall best fit criteria produces equivalent results. Both are indicated by the same red rectangle on the plot.

The height difference between all of the rectangles is insignificant and all of these model orders result in similar model performance. Therefore, you can choose the parameter combination that corresponds to the lowest model order and the simplest model.

3 Click the yellow rectangle on the far left, and then click **Select** to choose the lowest order: n_a=1, n_b=1, and n_k=10.

4 To continue, press any key while in the MATLAB Command Window.

Estimating Transfer Functions

Specifying the Structure of the Transfer Function

In this portion of the tutorial, you estimate a continuous-time transfer function. You must have already prepared your data, as described in "Preparing Data" on page 3-37. You can use the following results of estimated model orders to specify the orders of the model:

- For the first input/output combination, use:

  - Two poles, corresponding to `na=2` in the ARX model.

  - Delay of 5, corresponding to `nk=5` samples (or 2.5 minutes) in the ARX model.

- For the second input/output combination, use:

  - One pole, corresponding to `na=1` in the ARX model

  - Delay of 10, corresponding to `nk=10` samples (or 5 minutes) in the ARX model.

You can estimate a transfer function of these orders using the `tfest` command. For the estimation, you can also choose to view a progress report by setting the `Display` option to `on` in the option set created by the `tfestOptions` command.

```
Opt = tfestOptions('Display','on');
```

Collect the model orders and delays into variables to pass to `tfest`.

```
np = [2 1];
ioDelay = [2.5 5];
```

Estimate the transfer function.

```
mtf = tfest(Ze1,np,[],ioDelay,Opt);
```

```
Transfer Function Identification
Estimation data: Time domain data Ze1
Data has 1 outputs, 2 inputs and 1000 samples.
Number of poles: [2 1], Number of zeros: [1 0], I/O delay: [2.5 5]
Initialization Method: "iv"
```

Estimation Progress

```
Initializing model parameters...
Initializing using 'iv' method...
done.

Initialization complete.

Algorithm: Nonlinear least squares with automatically chosen line search method
--------------------------------------------------------------------------------
-------
                          Norm of      First-order   Improvement (%)
Iteration      Cost       step         optimality    Expected  Achieved  Bisections
--------------------------------------------------------------------------------
-------
     0       16.4744        -          1.77e+03        2.94        -         -
     1       12.6675       14.1        1.94e+03        2.94       23.1       2
     ?       11 0002       20 2        1 00 .02        ? 60       5 20       4
```

Result

```
Termination condition: No improvement along the search direction with line search..
Number of iterations: 14, Number of function evaluations: 538

Status: Estimated using TFEST
Fit to estimation data: 78.92%, FPE: 14.2213
```

■ Stop   Close

View the model's coefficients.

```
mtf
```

```
mtf =

  From input "ConsumptionRate" to output "Production":
                 -1.382 s + 0.0008305
  exp(-2.5*s) * -------------------------
                 s^2 + 1.014 s + 5.447e-12

  From input "Current" to output "Production":
```

```
                5.93
   exp(-5*s) * ----------
                s + 0.5858

Continuous-time identified transfer function.

Parameterization:
   Number of poles: [2 1]    Number of zeros: [1 0]
   Number of free coefficients: 6
   Use "tfdata", "getpvec", "getcov" for parameters and their uncertainties.

Status:
Estimated using TFEST on time domain data "Ze1".
Fit to estimation data: 78.92%
FPE: 14.22, MSE: 13.97
```
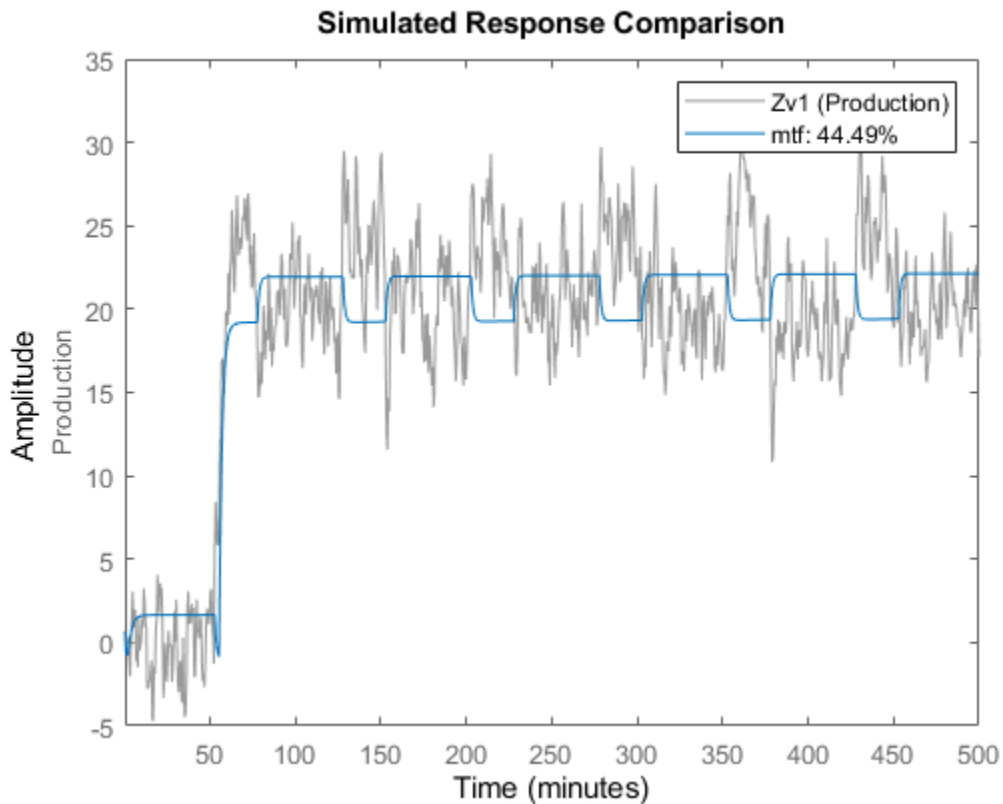
The model's display shows more than 85% fit to estimation data.

Validating the Model

In this portion of the tutorial, you create a plot that compares the actual output and the model output using the `compare` command.
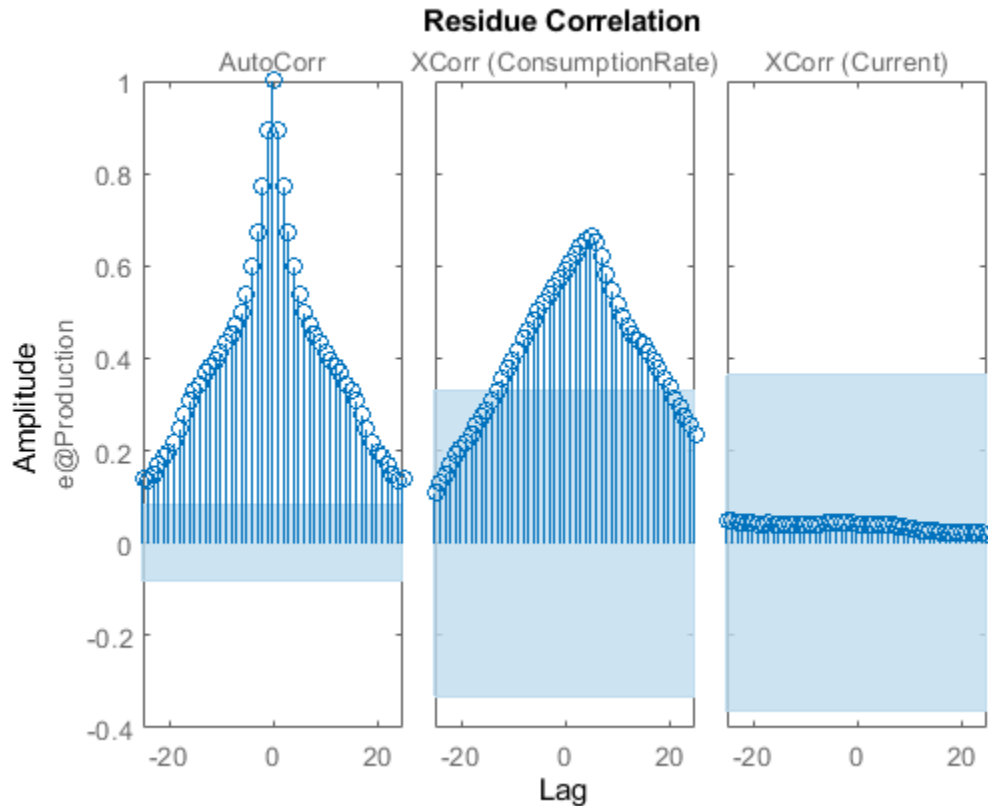
```
compare(Zv1,mtf)
```


The comparison shows about 60% fit.

Residual Analysis

Use the `resid` command to evaluate the characteristics of the residuals.

```
resid(Zv1,mtf)
```


The residuals show high degree of auto-correlation. This is not unexpected since the model `mtf` does not have any components to describe the nature of the noise separately. To model both the measured input-output dynamics and the disturbance dynamics you will need to use a model structure that contains elements to describe the noise component. You can use `bj`, `ssest` and `procest` commands, which create models of polynomial, state-space and process structures respectively. These structures, among others, contain elements to capture the noise behavior.

Estimating Process Models

Specifying the Structure of the Process Model

In this portion of the tutorial, you estimate a low-order, continuous-time transfer function (process model). the System Identification Toolbox product supports continuous-time models with at most three poles (which might contain underdamped poles), one zero, a delay element, and an integrator.

You must have already prepared your data, as described in "Preparing Data" on page 3-37.

You can use the following results of estimated model orders to specify the orders of the model:

- For the first input/output combination, use:

  - Two poles, corresponding to n_a=2 in the ARX model.
  - Delay of 5, corresponding to n_k=5 samples (or 2.5 minutes) in the ARX model.

- For the second input/output combination, use:

  - One pole, corresponding to n_a=1 in the ARX model.
  - Delay of 10, corresponding to n_k=10 samples (or 5 minutes) in the ARX model.

Note Because there is no relationship between the number of zeros estimated by the discrete-time ARX model and its continuous-time counterpart, you do not have an estimate for the number of zeros. In this tutorial, you can specify one zero for the first input/output combination, and no zero for the second-output combination.

Use the `idproc` command to create two model structures, one for each input/output combination:

```
midproc0 = idproc({'P2ZUD','P1D'}, 'TimeUnit', 'minutes');
```

The cell array `{'P2ZUD','P1D'}` specifies the model structure for each input/output combination:

- `'P2ZUD'` represents a transfer function with two poles (`P2`), one zero (`Z`), underdamped (complex-conjugate) poles (`U`) and a delay (`D`).
- `'P1D'` represents a transfer function with one pole (`P1`) and a delay (`D`).

The example also uses the `TimeUnit` parameter to specify the unit of time used.

Viewing the Model Structure and Parameter Values

View the two resulting models.

```
midproc0
```

```
midproc0 =
Process model with 2 inputs: y = G11(s)u1 + G12(s)u2
  From input 1 to output 1:
                      1+Tz*s
  G11(s) = Kp * --------------------- * exp(-Td*s)
                1+2*Zeta*Tw*s+(Tw*s)^2

           Kp = NaN
           Tw = NaN
         Zeta = NaN
           Td = NaN
           Tz = NaN

  From input 2 to output 1:
             Kp
  G12(s) = ---------- * exp(-Td*s)
            1+Tp1*s

           Kp = NaN
          Tp1 = NaN
           Td = NaN
```

```
Parameterization:
    {'P2DUZ'}    {'P1D'}
  Number of free coefficients: 8
  Use "getpvec", "getcov" for parameters and their uncertainties.

Status:
Created by direct construction or transformation. Not estimated.
```

The parameter values are set to NaN because they are not yet estimated.

Specifying Initial Guesses for Time Delays

Set the time delay property of the model object to 2.5 min and 5 min for each input/output pair as initial guesses. Also, set an upper limit on the delay because good initial guesses are available.

```
midproc0.Structure(1,1).Td.Value = 2.5;
midproc0.Structure(1,2).Td.Value = 5;
midproc0.Structure(1,1).Td.Maximum = 3;
midproc0.Structure(1,2).Td.Maximum = 7;
```

Note When setting the delay constraints, you must specify the delays in terms of actual time units (minutes, in this case) and not the number of samples.

Estimating Model Parameters Using procest

procest is an *iterative* estimator of process models, which means that it uses an iterative nonlinear least-squares algorithm to minimize a cost function. The *cost function* is the weighted sum of the squares of the errors.

Depending on its arguments, procest estimates different black-box polynomial models. You can use procest, for example, to estimate parameters for linear continuous-time transfer-function, state-space, or polynomial model structures. To use procest, you must provide a model structure with unknown parameters and the estimation data as input arguments.

For this portion of the tutorial, you must have already defined the model structure, as described in "Specifying the Structure of the Process Model" on page 3-55. Use midproc0 as the model structure and Ze1 as the estimation data:

```
midproc = procest(Ze1,midproc0);
present(midproc)


midproc =
Process model with 2 inputs: y = G11(s)u1 + G12(s)u2
  From input "ConsumptionRate" to output "Production":
                      1+Tz*s
  G11(s) = Kp * --------------------- * exp(-Td*s)
                1+2*Zeta*Tw*s+(Tw*s)^2

        Kp = -1.1807 +/- 0.29986
        Tw = 1.6437 +/- 714.6
      Zeta = 16.036 +/- 6958.9
        Td = 2.426 +/- 64.276
        Tz = -109.19 +/- 63.734
```

```
   From input "Current" to output "Production":
              Kp
  G12(s) = ---------- * exp(-Td*s)
            1+Tp1*s

        Kp = 10.264 +/- 0.048404
       Tp1 = 2.049 +/- 0.054901
        Td = 4.9175 +/- 0.034374

Parameterization:
   {'P2DUZ'}    {'P1D'}
  Number of free coefficients: 8
  Use "getpvec", "getcov" for parameters and their uncertainties.

Status:
Termination condition: Maximum number of iterations reached..
Number of iterations: 20, Number of function evaluations: 279

Estimated using PROCEST on time domain data "Ze1".
Fit to estimation data: 86.2%
FPE: 6.081, MSE: 5.984
More information in model's "Report" property.
```

Unlike discrete-time polynomial models, continuous-time models let you estimate the delays. In this case, the estimated delay values are different from the initial guesses you specified of 2.5 and 5 , respectively. The large uncertainties in the estimated values of the parameters of G_1(s) indicate that the dynamics from input 1 to the output are not captured well.
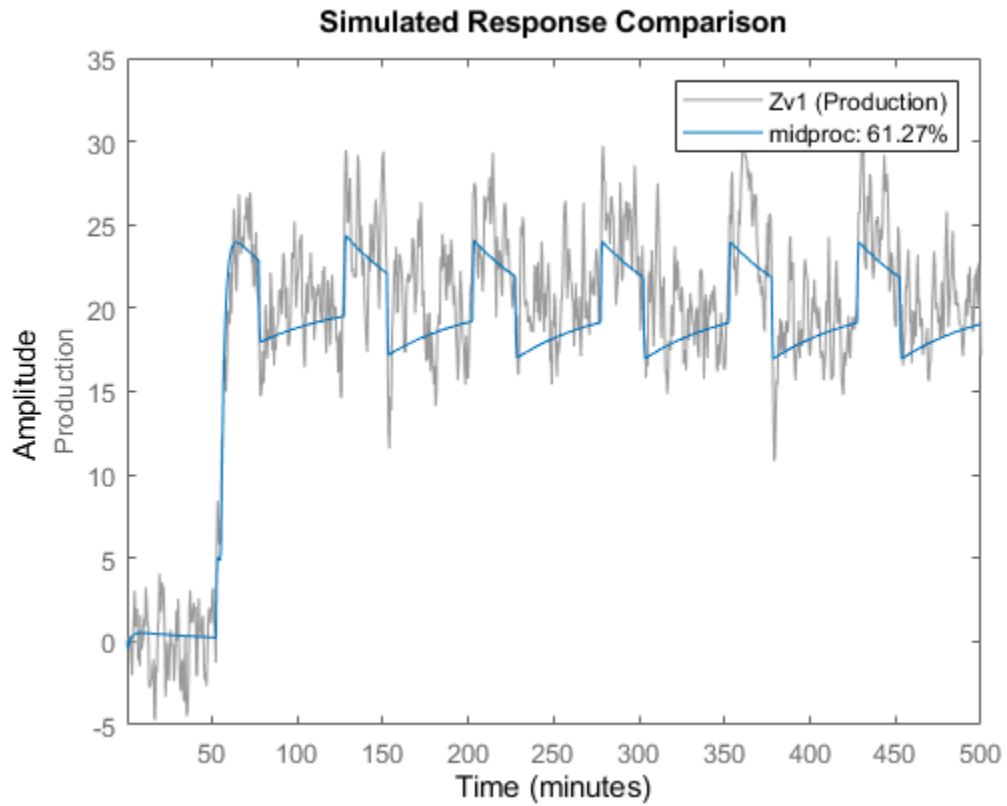
To learn more about estimating models, see "Process Models".

Validating the Model

In this portion of the tutorial, you create a plot that compares the actual output and the model output.

```
compare(Zv1,midproc)
```

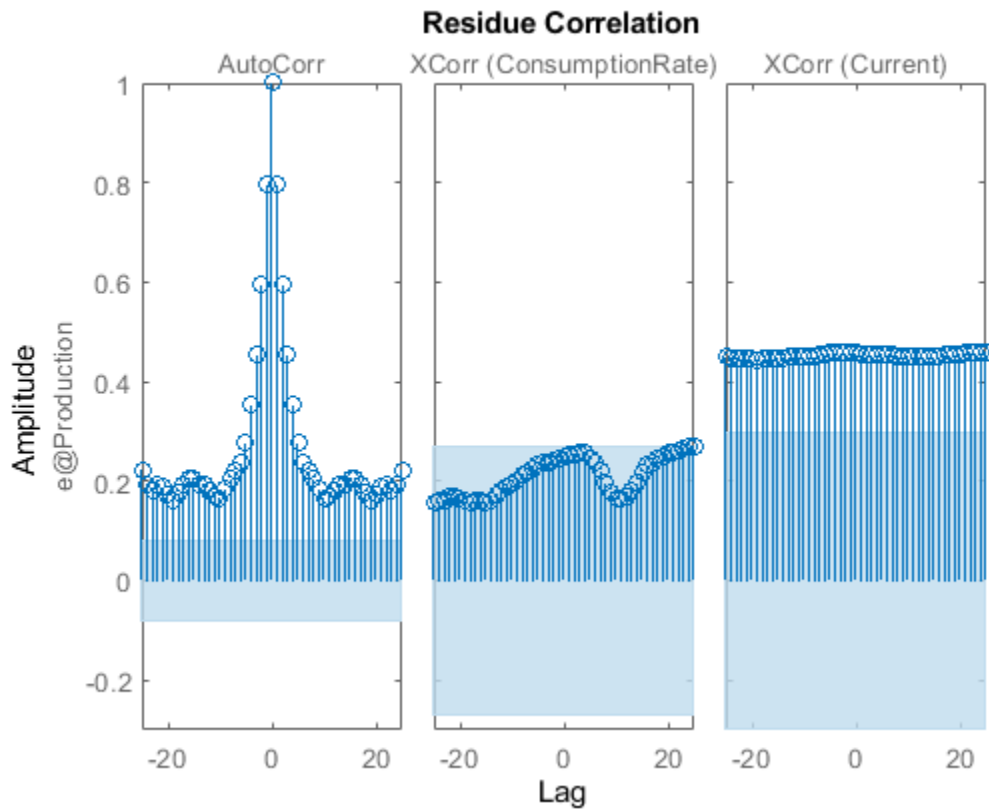## Simulated Response Comparison

The plot shows that the model output reasonably agrees with the measured output: there is an agreement of 60% between the model and the validation data.

Use `resid` to perform residual analysis.

```
resid(Zv1,midproc)
```

The cross-correlation between the second input and residual errors is significant. The autocorrelation plot shows values outside the confidence region and indicates that the residuals are correlated.

Change the algorithm for iterative parameter estimation to Levenberg-Marquardt.

```
Opt = procestOptions;
Opt.SearchMethod = 'lm';
midproc1 = procest(Ze1,midproc,Opt);
```

Tweaking the algorithm properties or specifying initial parameter guesses and rerunning the estimation may improve the simulation results. Adding a noise model may improve prediction results but not necessarily the simulation results.

Estimating a Process Model with Noise Model

This portion of the tutorial shows how to estimate a process model and include a noise model in the estimation. Including a noise model typically improves model prediction results but not necessarily the simulation results.

Use the following commands to specify a first-order ARMA noise:

```
Opt = procestOptions;
Opt.DisturbanceModel = 'ARMA1';
midproc2 = procest(Ze1,midproc0,Opt);
```

You can type `'dist'` instead of `'DisturbanceModel'`. Property names are not case sensitive, and you only need to include the portion of the name that uniquely identifies the property.

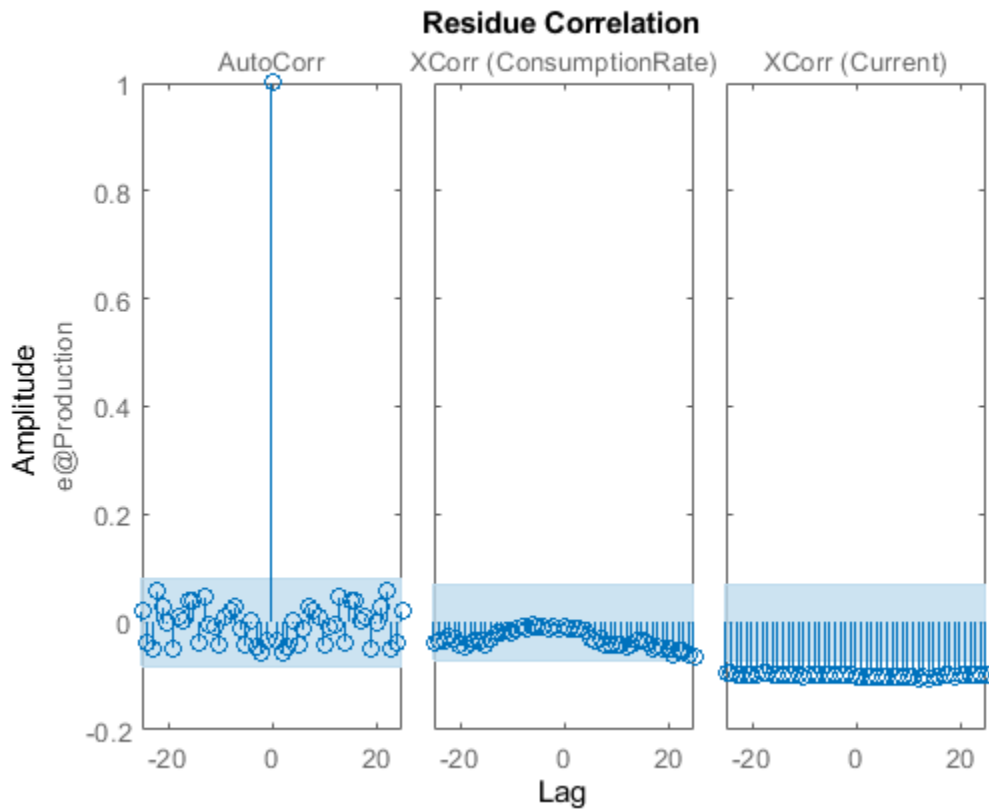Compare the resulting model to the measured data.

```
compare(Zv1,midproc2)
```
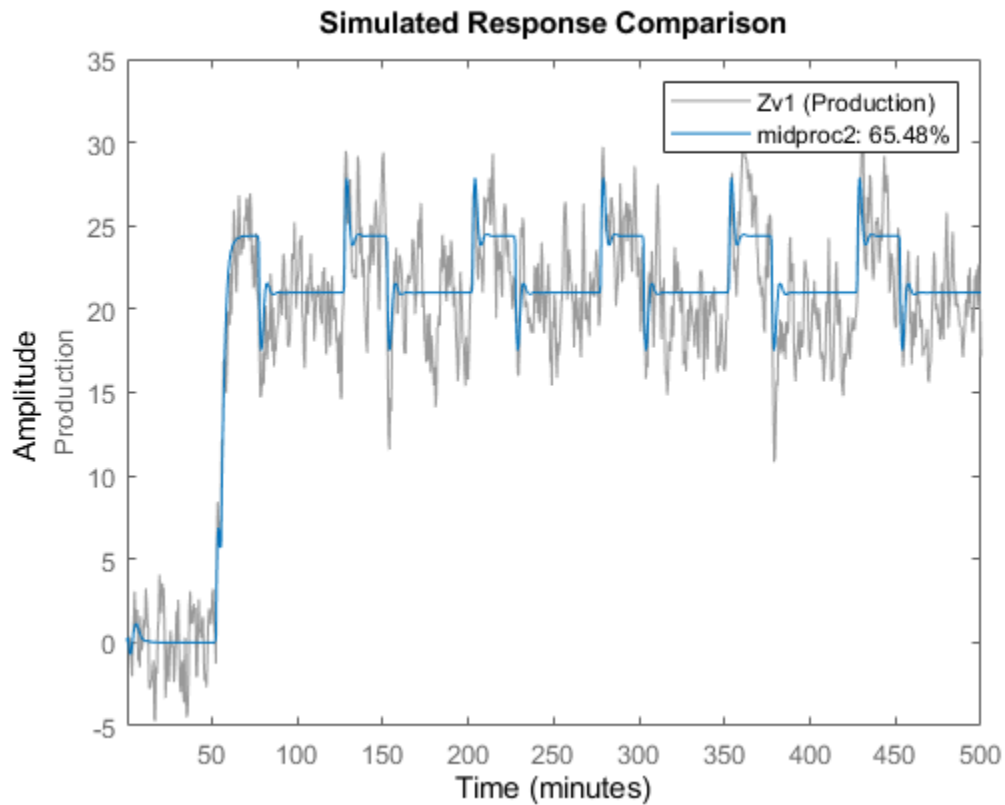

The plot shows that the model output maintains reasonable agreement with the validation-data output.

Perform residual analysis.

```
resid(Zv1,midproc2)
```

The residual plot shows that autocorrelation values are inside the confidence bounds. Thus adding a noise model produces uncorrelated residuals. This indicates a more accurate model.

Estimating Black-Box Polynomial Models

Model Orders for Estimating Polynomial Models

In this portion of the tutorial, you estimate several different types of black-box, input-output polynomial models.

You must have already prepared your data, as described in "Preparing Data" on page 3-37.

You can use the following previous results of estimated model orders to specify the orders of the polynomial model:

- For the first input/output combination, use:

  - Two poles, corresponding to n_a=2 in the ARX model.
  - One zero, corresponding to n_b=2 in the ARX model.
  - Delay of 5, corresponding to n_k=5 samples (or 2.5 minutes) in the ARX model.
- For the second input/output combination, use:

  - One pole, corresponding to n_a=1 in the ARX model.
  - No zeros, corresponding to n_b=1 in the ARX model.

- Delay of 10, corresponding to n_k=10 samples (or 5 minutes) in the ARX model.

Estimating a Linear ARX Model

About ARX Models

For a single-input/single-output system (SISO), the ARX model structure is:

$$y(t) + a_1 y(t-1) + ... + a_{na}y(t - n_a) =$$
$$b_1 u(t - n_k) + ... + b_{nb}u(t - n_k - n_b + 1) + e(t)$$

y(t) represents the output at time *t*, *u(t)* represents the input at time *t*, n_a is the number of poles, n_b is the number of zeros plus 1, n_k is the number of samples before the input affects output of the system (called the *delay* or *dead time* of the model), and *e(t)* is the white-noise disturbance.

The ARX model structure does not distinguish between the poles for individual input/output paths: dividing the ARX equation by *A*, which contains the poles, shows that *A* appears in the denominator for both inputs. Therefore, you can set n_a to the sum of the poles for each input/output pair, which is equal to 3 in this case.

The System Identification Toolbox product estimates the parameters $a_1...a_n$ and $b_1...b_n$ using the data and the model orders you specify.

Estimating ARX Models Using arx

Use `arx` to compute the polynomial coefficients using the fast, noniterative method `arx`:

```
marx = arx(Ze1,'na',3,'nb',[2 1],'nk',[5 10]);
present(marx) % Displays model parameters
              % with uncertainty information


marx =
Discrete-time ARX model: A(z)y(t) = B(z)u(t) + e(t)

  A(z) = 1 - 1.027 (+/- 0.02907) z^-1 + 0.1678 (+/- 0.042) z^-2 + 0.01289 (
                                                      +/- 0.02583) z^-3

  B1(z) = 1.86 (+/- 0.189) z^-5 - 1.608 (+/- 0.1888) z^-6

  B2(z) = 1.612 (+/- 0.07392) z^-10

Sample time: 0.5 minutes

Parameterization:
   Polynomial orders:   na=3   nb=[2 1]   nk=[5 10]
   Number of free coefficients: 6
   Use "polydata", "getpvec", "getcov" for parameters and their uncertainties.

Status:
Estimated using ARX on time domain data "Ze1".
Fit to estimation data: 90.7% (prediction focus)
FPE: 2.768, MSE: 2.719
More information in model's "Report" property.
```

MATLAB estimates the polynomials A , B1 , and B2. The uncertainty for each of the model parameters is computed to 1 standard deviation and appears in parentheses next to each parameter value.

Alternatively, you can use the following shorthand syntax and specify model orders as a single vector:

```
marx = arx(Ze1,[3 2 1 5 10]);
```

Accessing Model Data

The model you estimated, `marx`, is a discrete-time `idpoly` object. To get the properties of this model object, you can use the `get` function:

```
get(marx)
```

```
            A: [1 -1.0267 0.1678 0.0129]
            B: {[0 0 0 0 0 1.8599 -1.6084]  [0 0 0 0 0 0 0 0 0 0 1.6118]}
            C: 1
            D: 1
            F: {[1]  [1]}
IntegrateNoise: 0
      Variable: 'z^-1'
       IODelay: [0 0]
     Structure: [1x1 pmodel.polynomial]
 NoiseVariance: 2.7436
    InputDelay: [2x1 double]
   OutputDelay: 0
            Ts: 0.5000
      TimeUnit: 'minutes'
     InputName: {2x1 cell}
     InputUnit: {2x1 cell}
    InputGroup: [1x1 struct]
    OutputName: {'Production'}
    OutputUnit: {'mg/min'}
   OutputGroup: [1x1 struct]
         Notes: [0x1 string]
      UserData: []
          Name: ''
  SamplingGrid: [1x1 struct]
        Report: [1x1 idresults.arx]
```

You can access the information stored by these properties using dot notation. For example, you can compute the discrete poles of the model by finding the roots of the A polynomial.

```
marx_poles = roots(marx.a)
```

```
marx_poles =

    0.7953
    0.2877
   -0.0564
```

In this case, you access the A polynomial using `marx.a`.

The model `marx` describes system dynamics using three discrete poles.

Tip You can also use `pole` to compute the poles of a model directly.

Learn More

To learn more about estimating polynomial models, see "Input-Output Polynomial Models".

For more information about accessing model data, see "Data Extraction".

Estimating State-Space Models

About State-Space Models

The general state-space model structure is:

$$\frac{dx}{dt} = Ax(t) + Bu(t) + Ke(t)$$

$$y(t) = Cx(t) + Du(t) + e(t)$$

$y(t)$ represents the output at time t, $u(t)$ represents the input at time t, $x(t)$ is the state vector at time t, and $e(t)$ is the white-noise disturbance.

You must specify a single integer as the model order (dimension of the state vector) to estimate a state-space model. By default, the delay equals 1.

The System Identification Toolbox product estimates the state-space matrices A, B, C, D, and K using the model order and the data you specify.

The state-space model structure is a good choice for quick estimation because it contains only two parameters: n is the number of poles (the size of the A matrix) and nk is the delay.

Estimating State-Space Models Using n4sid

Use the `n4sid` command to specify a range of model orders and evaluate the performance of several state-space models (orders 2 to 8):

```
mn4sid = n4sid(Ze1,2:8,'InputDelay',[4 9]);
```

This command uses the fast, noniterative (subspace) method and opens the following plot. You use this plot to decide which states provide a significant relative contribution to the input/output behavior, and which states provide the smallest contribution.
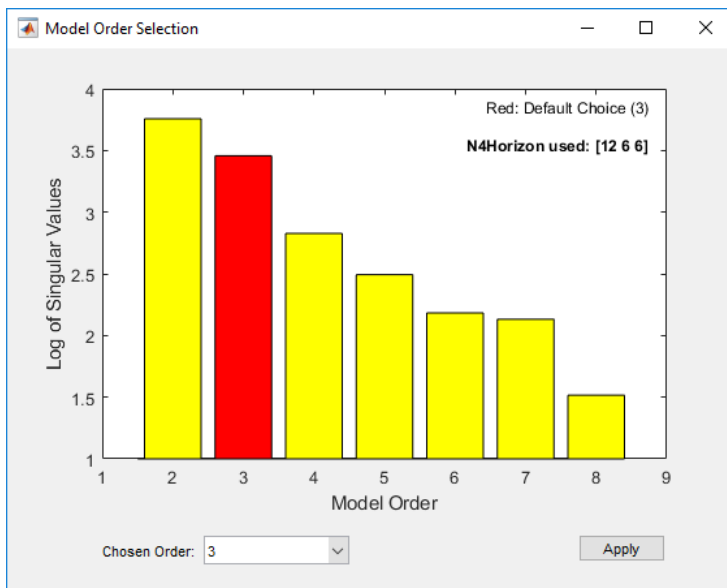
The vertical axis is a relative measure of how much each state contributes to the input/output behavior of the model (*log of singular values of the covariance matrix*). The horizontal axis corresponds to the model order n. This plot recommends n=3, indicated by a red rectangle.

The **Chosen Order** order field displays the recommended model order, 3 in this case, by default. You can change the order selection by using the **Chosen Order** drop-down list. Apply the value in the **Chosen Order** field and close the order-selection window by clicking **Apply**.

By default, n4sid uses a free parameterization of the state-space form. To estimate a canonical form instead, set the value of the SSParameterization property to 'Canonical' . You can also specify the input-to-output delay (in samples) using the 'InputDelay' property.

```
mCanonical = n4sid(Ze1,3,'SSParameterization','canonical','InputDelay',[4 9]);
present(mCanonical);  % Display model properties
```

```
mCanonical =
  Discrete-time identified state-space model:
    x(t+Ts) = A x(t) + B u(t) + K e(t)
       y(t) = C x(t) + D u(t) + e(t)

  A =
                     x1                   x2                   x3
    x1                0                    1                    0
    x2                0                    0                    1
    x3  0.0737 +/- 0.05919  -0.6093 +/- 0.1626    1.446 +/- 0.1287

  B =
            ConsumptionR            Current
    x1    1.844 +/-    0.175   0.5633 +/-   0.122
    x2    1.063 +/-   0.1673    2.308 +/- 0.1222
    x3  0.2779 +/- 0.09615     1.878 +/- 0.1058

  C =
               x1  x2  x3
    Production    1   0   0

  D =
              ConsumptionR        Current
    Production            0              0

  K =
               Production
    x1  0.8674 +/- 0.03169
    x2  0.6849 +/- 0.04145
    x3  0.5105 +/- 0.04352

  Input delays (sampling periods): 4  9

Sample time: 0.5 minutes

Parameterization:
   CANONICAL form with indices: 3.
   Feedthrough: none
   Disturbance component: estimate
   Number of free coefficients: 12
   Use "idssdata", "getpvec", "getcov" for parameters and their uncertainties.
```

```
Status:
Estimated using N4SID on time domain data "Ze1".
Fit to estimation data: 91.39% (prediction focus)
FPE: 2.402, MSE: 2.331
More information in model's "Report" property.
```

Note `mn4sid` and `mCanonical` are discrete-time models. To estimate a continuous-time model, set the `'Ts'` property to `0` in the estimation command, or use the `ssest` command:

```
mCT1 = n4sid(Ze1, 3, 'Ts', 0, 'InputDelay', [2.5 5])
mCT2 = ssest(Ze1, 3,'InputDelay', [2.5 5])
```

Learn More

To learn more about estimating state-space models, see "State-Space Models".

Estimating a Box-Jenkins Model

About Box-Jenkins Models

The general Box-Jenkins (BJ) structure is:

$$y(t) = \sum_{i=1}^{nu} \frac{B_i(q)}{F_i(q)} u_i(t - nk_i) + \frac{C(q)}{D(q)} e(t)$$

To estimate a BJ model, you need to specify the parameters n_b, n_f, n_c, n_d, and n_k.

Whereas the ARX model structure does not distinguish between the poles for individual input/output paths, the BJ model provides more flexibility in modeling the poles and zeros of the disturbance separately from the poles and zeros of the system dynamics.

Estimating a BJ Model Using polyest

You can use `polyest` to estimate the BJ model. `polyest` is an iterative method and has the following general syntax:

```
polyest(data,[na nb nc nd nf nk]);
```

To estimate the BJ model, type:

```
na = 0;
nb = [ 2 1 ];
nc = 1;
nd = 1;
nf = [ 2 1 ];
nk = [ 5 10];
mbj = polyest(Ze1,[na nb nc nd nf nk]);
```

This command specifies `nf=2` , `nb=2` , `nk=5` for the first input, and `nf=nb=1` and `nk=10` for the second input.

Display the model information.

```
present(mbj)
```

```
mbj =
Discrete-time BJ model: y(t) = [B(z)/F(z)]u(t) + [C(z)/D(z)]e(t)
  B1(z) = 1.823 (+/- 0.1792) z^-5 - 1.315 (+/- 0.2367) z^-6

  B2(z) = 1.791 (+/- 0.06431) z^-10

  C(z) = 1 + 0.1068 (+/- 0.04009) z^-1

  D(z) = 1 - 0.7452 (+/- 0.02694) z^-1

  F1(z) = 1 - 1.321 (+/- 0.06936) z^-1 + 0.5911 (+/- 0.05514) z^-2

  F2(z) = 1 - 0.8314 (+/- 0.006441) z^-1

Sample time: 0.5 minutes

Parameterization:
   Polynomial orders:   nb=[2 1]   nc=1   nd=1   nf=[2 1]
   nk=[5 10]
   Number of free coefficients: 8
   Use "polydata", "getpvec", "getcov" for parameters and their uncertainties.

Status:
Termination condition: Near (local) minimum, (norm(g) < tol)..
Number of iterations: 6, Number of function evaluations: 13

Estimated using POLYEST on time domain data "Ze1".
Fit to estimation data: 90.75% (prediction focus)
FPE: 2.733, MSE: 2.689
More information in model's "Report" property.
```

The uncertainty for each of the model parameters is computed to 1 standard deviation and appears in parentheses next to each parameter value.

The polynomials C and D give the numerator and the denominator of the noise model, respectively.

Tip Alternatively, you can use the following shorthand syntax that specifies the orders as a single vector:

```
mbj = bj(Ze1,[2 1 1 1 2 1 5 10]);
```

bj is a version of `polyest` that specifically estimates the BJ model structure.

Learn More

To learn more about identifying input-output polynomial models such as BJ, see "Input-Output Polynomial Models".

Comparing Model Output to Measured Output

Compare the output of the ARX, state-space, and Box-Jenkins models to the measured output.

```
compare(Zv1,marx,mn4sid,mbj)
```

Simulated Response Comparison
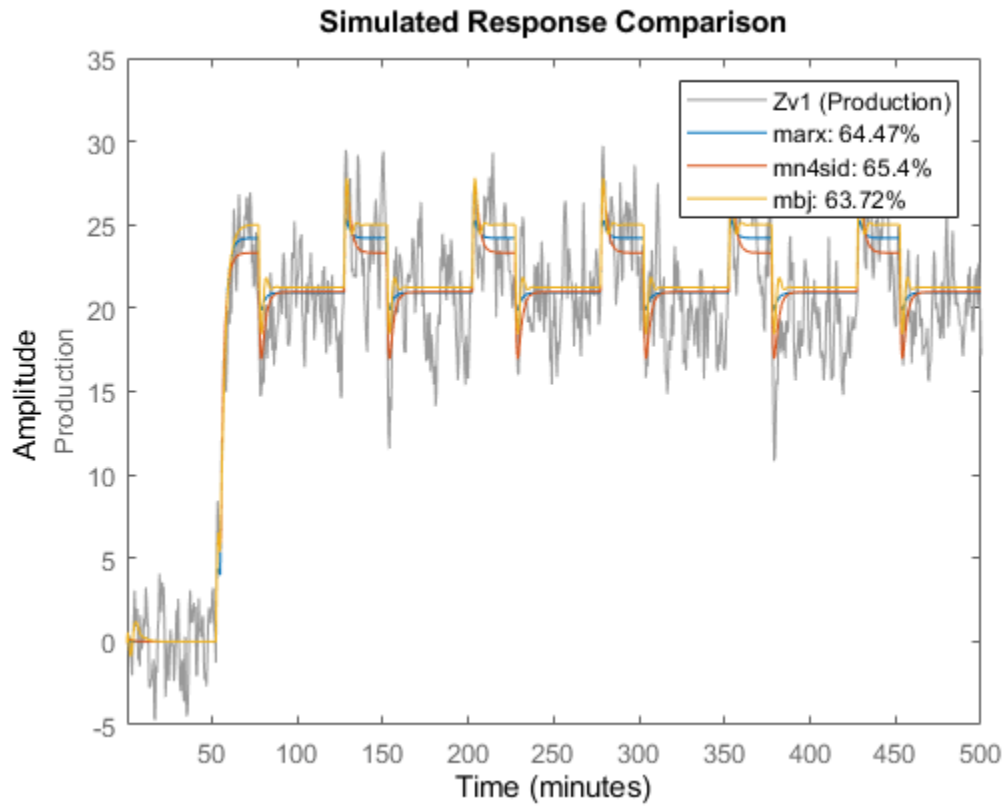
compare plots the measured output in the validation data set against the simulated output from the models. The input data from the validation data set serves as input to the models.

Perform residual analysis on the ARX, state-space, and Box-Jenkins models.

```
resid(Zv1,marx,mn4sid,mbj)
```
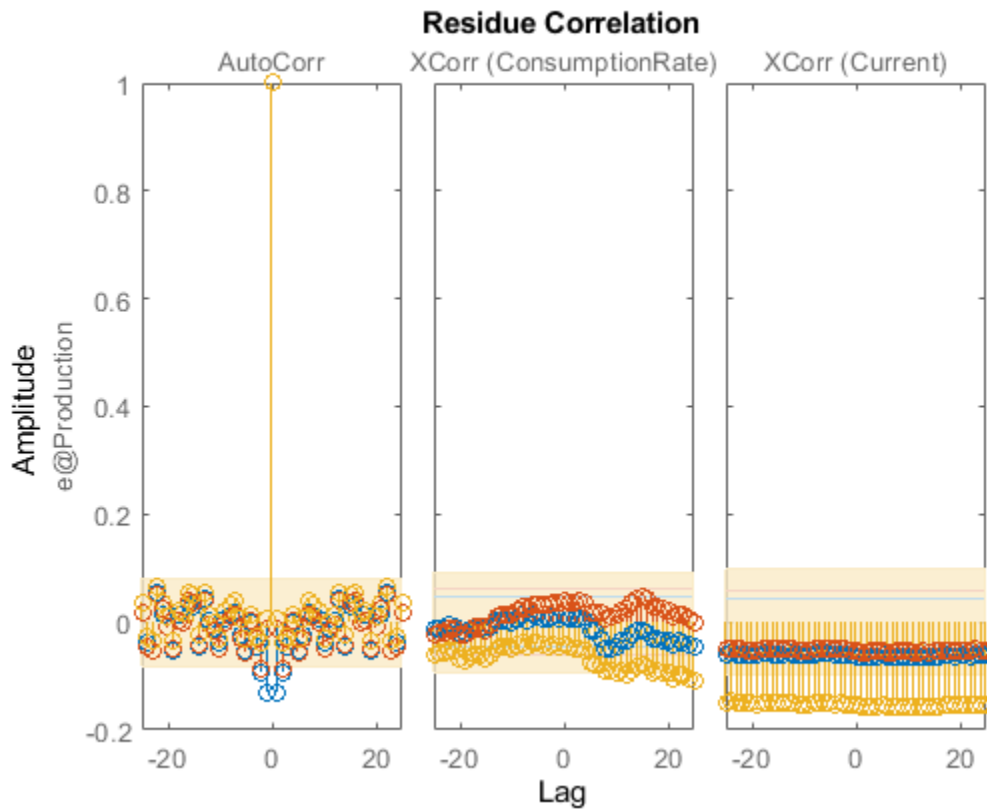
All three models simulate the output equally well and have uncorrelated residuals. Therefore, choose the ARX model because it is the simplest of the three input-output polynomial models and adequately captures the process dynamics.

Simulating and Predicting Model Output

Simulating the Model Output

In this portion of the tutorial, you simulate the model output. You must have already created the continuous-time model `midproc2`, as described in "Estimating Process Models" on page 3-55.

Simulating the model output requires the following information:

- Input values to the model
- Initial conditions for the simulation (also called *initial states*)

For example, the following commands use the `iddata` and `idinput` commands to construct an input data set, and use `sim` to simulate the model output:

```
% Create input for simulation
U = iddata([],idinput([200 2]),'Ts',0.5,'TimeUnit','min');
% Simulate the response setting initial conditions equal to zero
ysim_1 = sim(midproc2,U);
```

To maximize the fit between the simulated response of a model to the measured output for the same input, you can compute the initial conditions corresponding to the measured data. The best fitting

initial conditions can be obtained by using `findstates` on the state-space version of the estimated model. The following commands estimate the initial states `X0est` from the data set `Zv1`:

```
% State-space version of the model needed for computing initial states
midproc2_ss = idss(midproc2);
X0est = findstates(midproc2_ss,Zv1);
```

Next, simulate the model using the initial states estimated from the data.

```
% Simulation input
Usim = Zv1(:,[],:);
Opt = simOptions('InitialCondition',X0est);
ysim_2 = sim(midproc2_ss,Usim,Opt);
```

Compare the simulated and the measured output on a plot.

```
figure
plot([ysim_2.y, Zv1.y])
legend({'model output','measured'})
xlabel('time'), ylabel('Output')
```
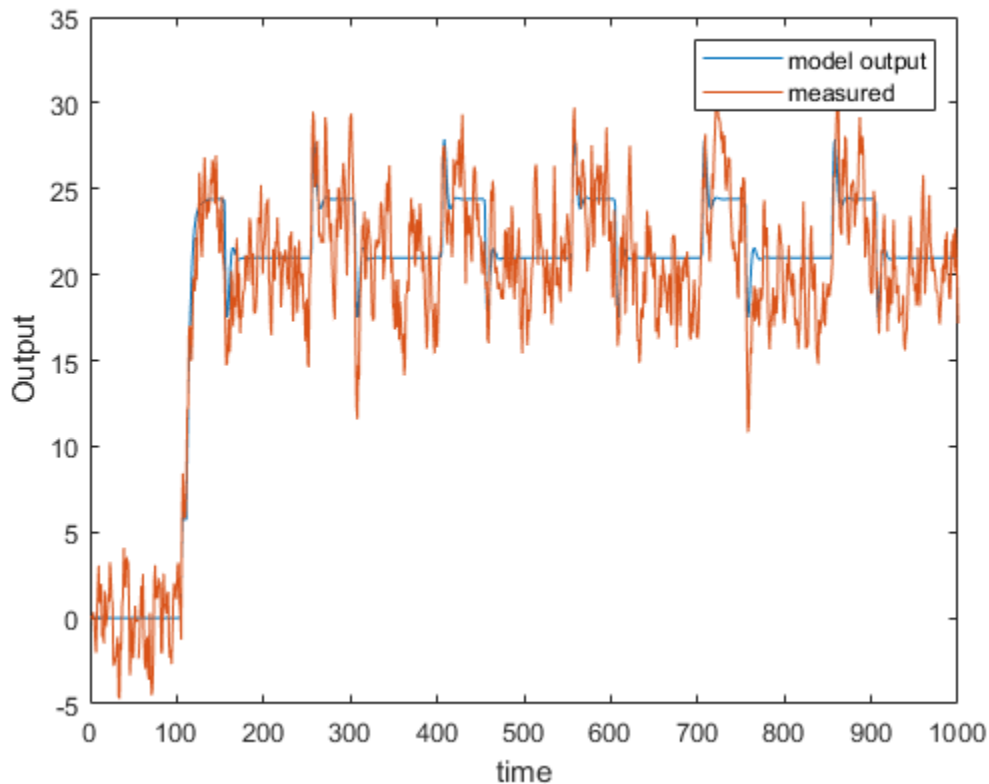

Predicting the Future Output

Many control-design applications require you to predict the future outputs of a dynamic system using the past input/output data.

For example, use `predict` to predict the model response five steps ahead:

```
predict(midproc2,Ze1,5)
```
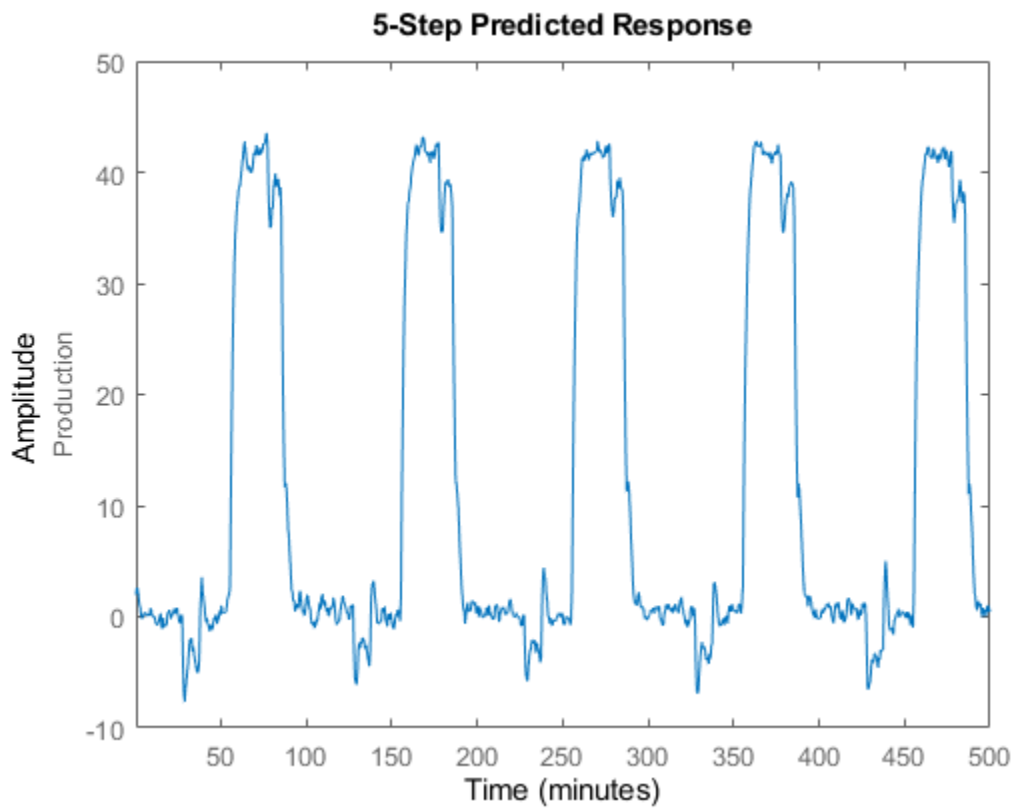

Compare the predicted output values with the measured output values. The third argument of `compare` specifies a five-step-ahead prediction. When you do not specify a third argument, `compare` assumes an infinite prediction horizon and simulates the model output instead.

```
compare(Ze1,midproc2,5)
```
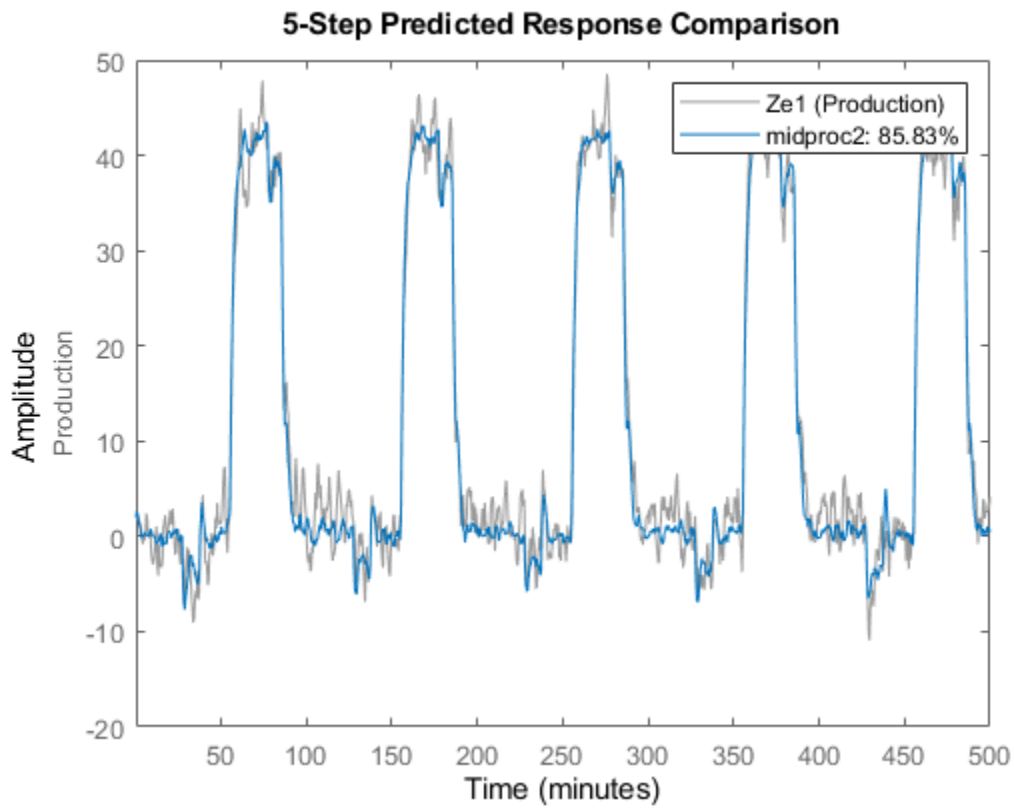
Use `pe` to compute the prediction error `Err` between the predicted output of `midproc2` and the measured output. Then, plot the error spectrum using the `spectrum` command.

```
[Err] = pe(midproc2,Zv1);
spectrum(spa(Err,[],logspace(-2,2,200)))
```

Power Spectrum

From: e@e@Production To: e@Production
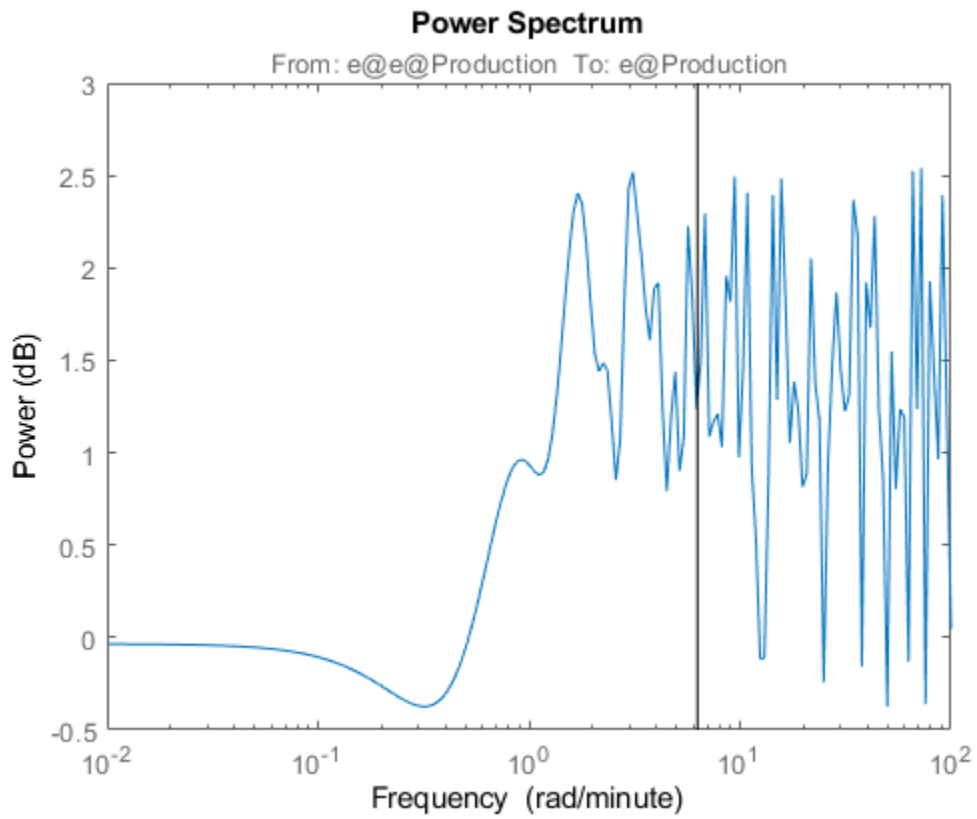
The prediction errors are plotted with a 1-standard-deviation confidence interval. The errors are greater at high frequencies because of the high-frequency nature of the disturbance.

Identify Low-Order Transfer Functions (Process Models) Using System Identification App

Introduction

Objectives

Estimate and validate simple, continuous-time transfer functions from single-input/single-output (SISO) data to find the one that best describes the system dynamics.

After completing this tutorial, you will be able to accomplish the following tasks using the System Identification app :

- Import data objects from the MATLAB workspace into the app.
- Plot and process the data.
- Estimate and validate low-order, continuous-time models from the data.
- Export models to the MATLAB workspace.
- Simulate the model using Simulink software.

Note This tutorial uses time-domain data to demonstrate how you can estimate linear models. The same workflow applies to fitting frequency-domain data.

Data Description

This tutorial uses the data file `proc_data.mat`, which contains 200 samples of simulated single-input/single-output (SISO) time-domain data. The input is a random binary signal that oscillates between -1 and 1. White noise (corresponding to a load disturbance) is added to the input with a standard deviation of 0.2, which results in a signal-to-noise ratio of about 20 dB. This data is simulated using a second-order system with underdamped modes (complex poles) and a peak response at 1 rad/s:

$$G(s) = \frac{1}{1 + 0.2s + s^2}e^{-2s}$$

The sample time of the simulation is 1 second.

What Is a Continuous-Time Process Model?

Continuous-time process models are low-order transfer functions that describe the system dynamics using static gain, a time delay before the system output responds to the input, and characteristic time constants associated with poles and zeros. Such models are popular in the industry and are often used for tuning PID controllers, for example. Process model parameters have physical significance.

You can specify different process model structures by varying the number of poles, adding an integrator, or including a time delay or a zero. The highest process model order you can specify in this toolbox is three, and the poles can be real or complex (underdamped modes).

In general, a linear system is characterized by a transfer function G, which is an operator that takes the input u to the output y:

$$y = Gu$$

For a continuous-time system, G relates the Laplace transforms of the input $U(s)$ and the output $Y(s)$, as follows:

$$Y(s) = G(s)U(s)$$

In this tutorial, you estimate G using different process-model structures.

For example, the following model structure is a first-order, continuous-time model, where K is the static gain, T_{p1} is a time constant, and T_d is the input-to-output delay:

$$G(s) = \frac{K}{1 + sT_{p1}}e^{-sT_d}$$

Preparing Data for System Identification

Loading Data into the MATLAB Workspace

Load the data in `proc_data.mat` by typing the following command in the MATLAB Command Window:

```
load proc_data
```

This command loads the data into the MATLAB workspace as the data object `z`. For more information about `iddata` objects, see the corresponding reference page.

Opening the System Identification App

To open the System Identification app , type the following command at the MATLAB Command Window:

```
systemIdentification
```

The default session name, `Untitled`, appears in the title bar.
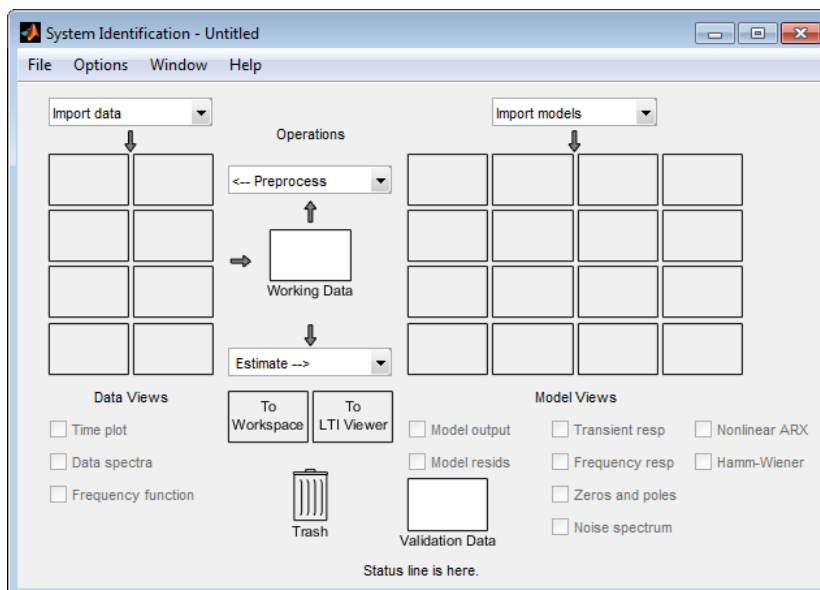
Importing Data Objects into the System Identification App

You can import data object into the app from the MATLAB workspace.

You must have already loaded the sample data into MATLAB, as described in "Loading Data into the MATLAB Workspace" on page 3-76, and opened the app, as described in "Opening the System Identification App" on page 3-76.

To import a data object into the System Identification app :

1   Select **Import data > Data object**.

This action opens the Import Data dialog box.
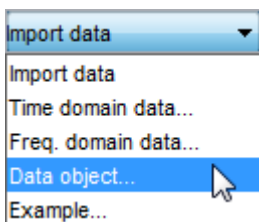
2   In the Import Data dialog box, specify the following options:

- **Object** — Enter z as the name of the MATLAB variable that is the time-domain data object. Press **Enter**.
- **Data name** — Use the default name z, which is the same as the name of the data object you are importing. This name labels the data in the System Identification app after the import operation is completed.
- **Starting time** — Enter 0 as the starting time. This value designates the starting value of the time axis on time plots.
- **Sample time** — Enter 1 as the time between successive samples in seconds. This value represents the actual sample time in the experiment.

The Import Data dialog box now resembles the following figure.
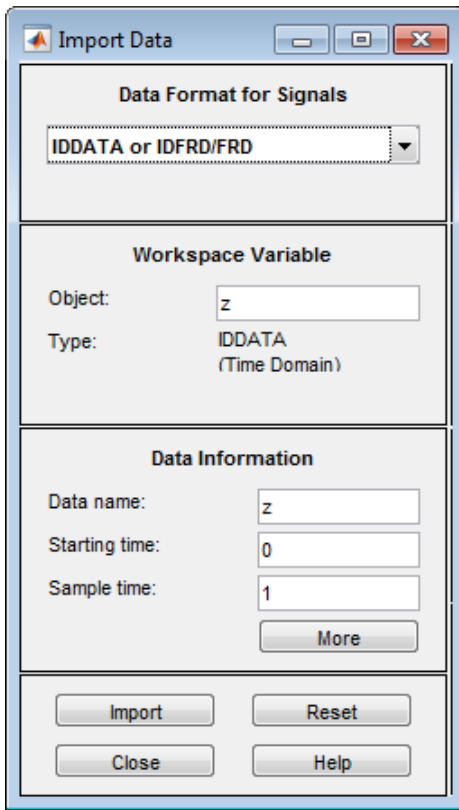
3 Click **Import** to add the data to the System Identification app. The app adds an icon to represent the data.
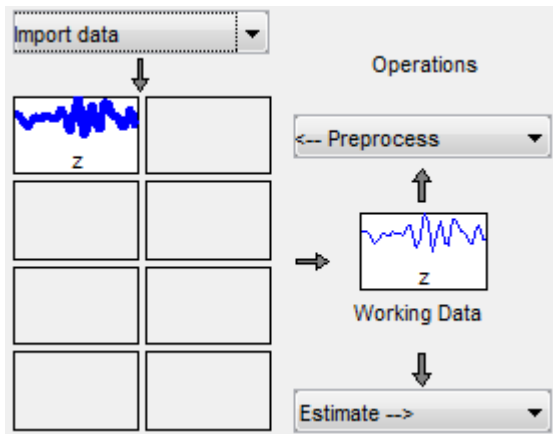
4 Click **Close** to close the Import Data dialog box.

Plotting and Processing Data

In this portion of the tutorial, you evaluate the data and process it for system identification. You learn how to:

- Plot the data.
- Remove offsets by subtracting the mean values of the input and the output.

- Split the data into two parts. You use one part of the data for model estimation, and the other part of the data for model validation.

The reason you subtract the mean values from each signal is because, typically, you build linear models that describe the responses for deviations from a physical equilibrium. With steady-state data, it is reasonable to assume that the mean levels of the signals correspond to such an equilibrium. Thus, you can seek models around zero without modeling the absolute equilibrium levels in physical units.

You must have already imported data into the System Identification app, as described in "Importing Data Objects into the System Identification App" on page 3-77.

To plot and process the data:

1   Select the **Time plot** check box to open the Time Plot window.
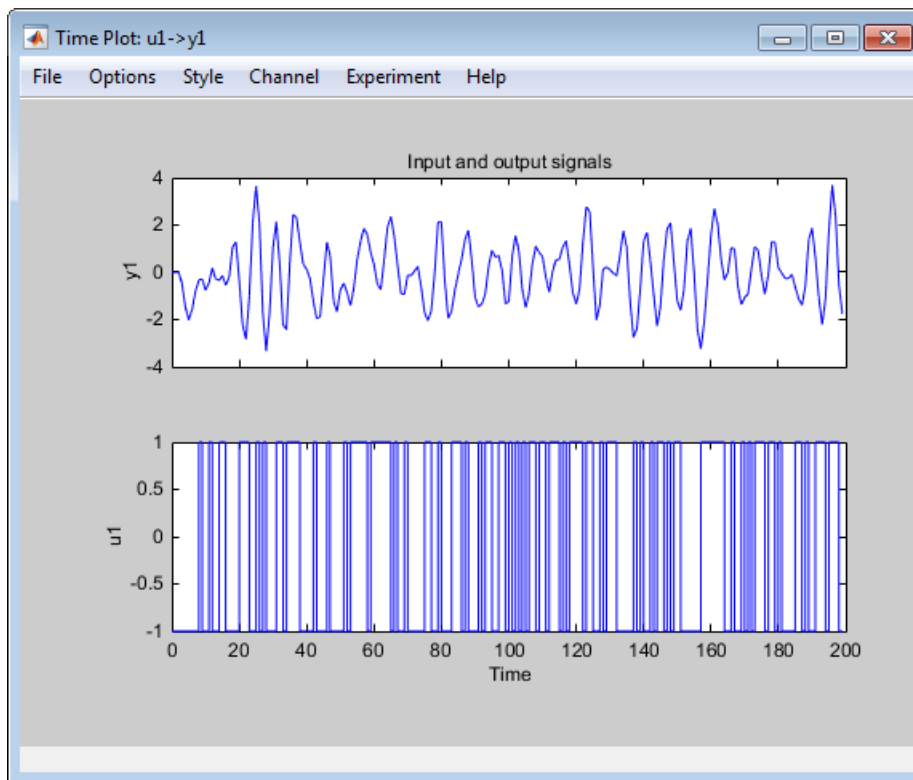
The bottom axes show the input data—a random binary sequence, and the top axes show the output data.

The next two steps demonstrate how to modify the axis limits in the plot.

2   To modify the vertical-axis limits for the input data, select **Options > Set axes limits** in the Time Plot figure window.

3   In the Limits for Time Plot dialog box, set the new vertical axis limit of the input data channel **u1** to [-1.5 1.5]. Click **Apply** and **Close**.
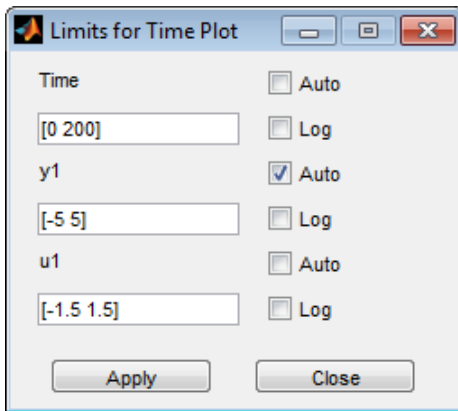
Note The other two fields in the Limits for Time Plot dialog box, **Time** and **y1**, let you set the axis limits for the time axis and the output channel axis, respectively. You can also specify each axis to be logarithmic or linear by selecting the corresponding option.

The following figure shows the updated time plot.
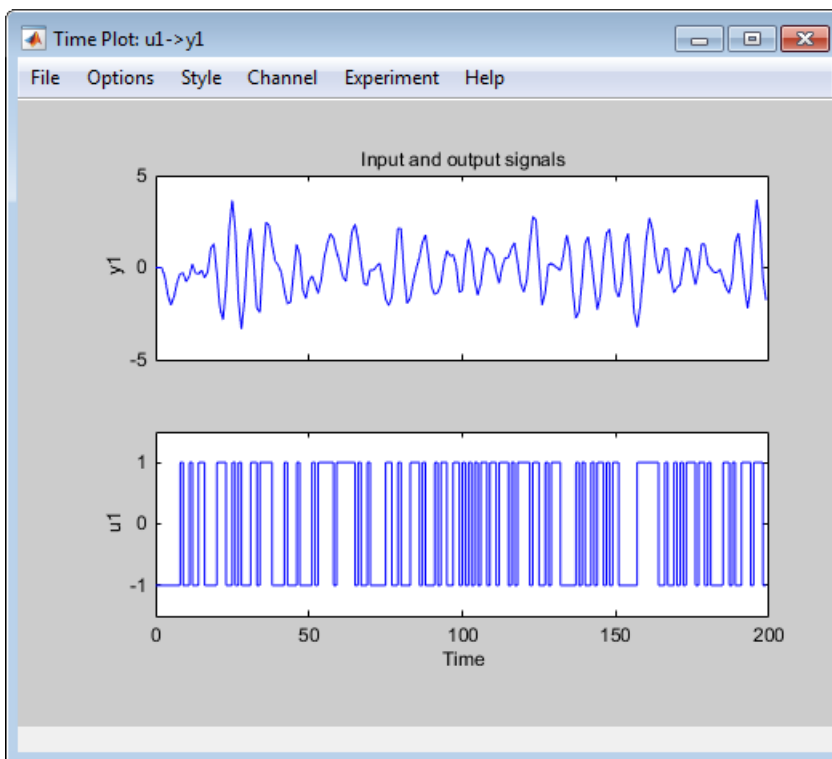
4   In the System Identification app , select **<--Preprocess > Quick start** to perform the following four actions:

- Subtract the mean value from each channel.
- Split the data into two parts.
- Specify the first part of the data as estimation data (or **Working Data**).

- Specify the second part of the data as **Validation Data**.

Learn More

For information about supported data processing operations, such as resampling and filtering the data, see "Preprocess Data".

Estimating a Second-Order Transfer Function (Process Model) with Complex Poles

Estimating a Second-Order Transfer Function Using Default Settings

In this portion of the tutorial, you estimate models with this structure:

$$G(s) = \frac{K}{\left(1 + 2\xi T_w s + T_w^2 s^2\right)} e^{-T_d s}$$

You must have already processed the data for estimation, as described in "Plotting and Processing Data" on page 3-78.

To identify a second-order transfer function:

1 In the System Identification app, select **Estimate > Process models** to open the Process Models dialog box.
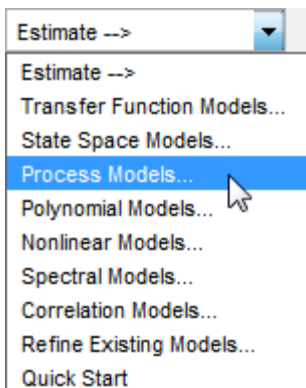
2 In the **Model Transfer Function** area of the Process Models dialog box, specify the following options:

- Under **Poles**, select 2 and `Underdamped`.

  This selection updates the Model Transfer Function to a second-order model structure that can contain complex poles.
- Make sure that the **Zero** and **Integrator** check boxes are cleared to exclude a zero and an integrator (self-regulating) from the model.
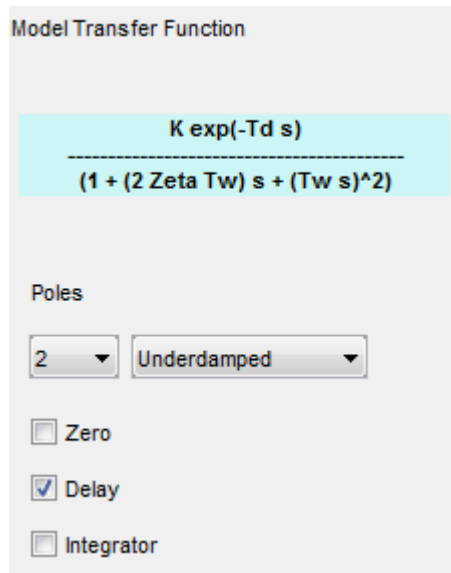
3 The **Parameter** area of the Process Models dialog box now shows four active parameters: K, Tw, Zeta, and Td. In the **Initial Guess** area, keep the default `Auto-selected` option to calculate the initial parameter values during the estimation. The **Initial Guess** column in the Parameter table displays `Auto`.

4 Keep the default **Bounds** values, which specify the minimum and maximum values of each parameter.

Tip If you know the range of possible values for a parameter, you can type these values into the corresponding **Bounds** fields to help the estimation algorithm. Press the **Enter** key after you specify the values.

---

5 Keep the default settings for the estimation algorithm:

- **Disturbance Model** — `None` means that the algorithm does not estimate the noise model. This option also sets the **Focus** to `Simulation`.

- **Focus** — `Simulation` means that the estimation algorithm does not use the noise model to weigh the relative importance of how closely to fit the data in various frequency ranges. Instead, the algorithm uses the input spectrum in a particular frequency range to weigh the relative importance of the fit in that frequency range.

Tip The `Simulation` setting is optimized for identifying models that you plan to use for output simulation. If you plan to use your model for output prediction or control applications, or to improve parameter estimates using a noise model, select `Prediction`.

- **Initial condition** — `Auto` means that the algorithm analyzes the data and chooses the optimum method for handling the initial state of the system. If you get poor results, you might try setting a specific method for handling initial states, rather than choosing it automatically.

- **Covariance** — `Estimate` means that the algorithm computes parameter uncertainties that display as model confidence regions on plots.

The app assigns a name to the model, shown in the **Name** field (located at the bottom of the dialog box). By default, the name is the acronym P2DU, which indicates two poles (P2), a delay (D), and underdamped modes (U).
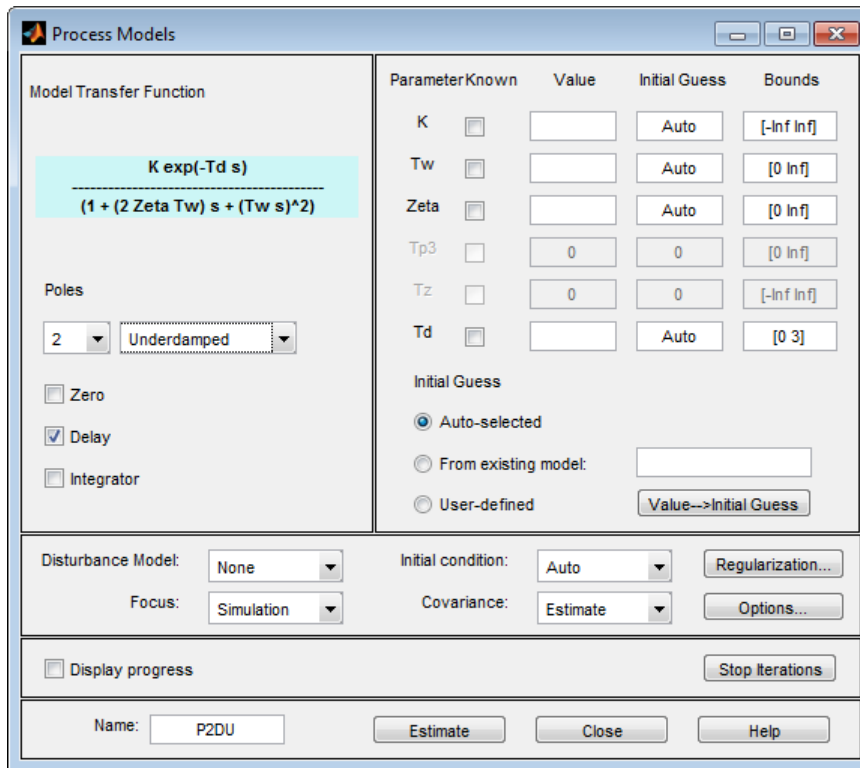
6 Click **Estimate** to add the model P2DU to the System Identification app.

Tips for Specifying Known Parameters

If you know a parameter value exactly, you can type this value in the **Value** column of the Process Models dialog box. Select the corresponding **Known** check box after you specify the value.

If you know the approximate value of a parameter, you can help the estimation algorithm by entering an initial value in the **Initial Guess** column. In this case, keep the **Known** check box cleared to allow the estimation to fine-tune this initial guess.

For example, to fix the time-delay value Td at 2s, type this value into **Value** field of the Parameter table in the Process Models dialog box. Then select the corresponding **Known** check box.

Validating the Model

You can analyze the following plots to evaluate the quality of the model:

• Comparison of the model output and the measured output on a time plot

• Autocorrelation of the output residuals, and cross-correlation of the input and the output residuals

You must have already estimated the model, as described in "Estimating a Second-Order Transfer Function Using Default Settings" on page 3-81.

Examining Model Output

You can use the model-output plot to check how well the model output matches the measured output in the validation data set. A good model is the simplest model that best describes the dynamics and successfully simulates or predicts the output for different inputs.

To generate the model-output plot, select the **Model output** check box in the System Identification app. If the plot is empty, click the model icon in the System Identification app window to display the model on the plot.
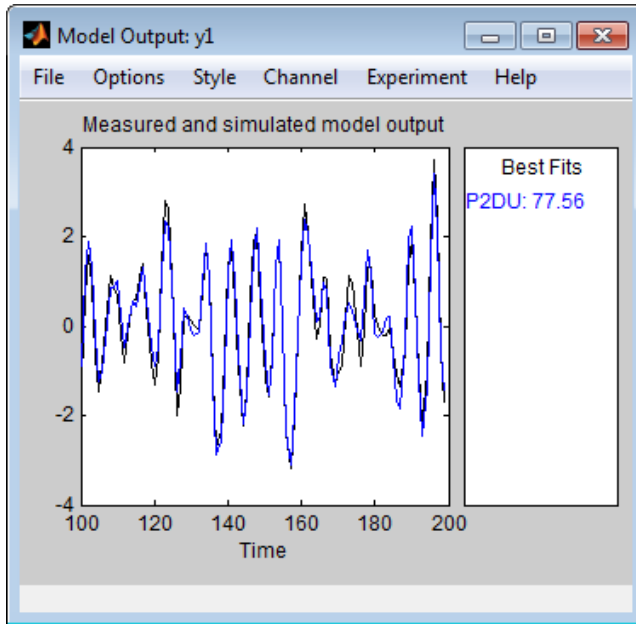
The System Identification Toolbox software uses input validation data as input to the model, and plots the simulated output on top of the output validation data. The preceding plot shows that the model output agrees well with the validation-data output.

The **Best Fits** area of the Model Output plot shows the agreement (in percent) between the model output and the validation-data output.

Recall that the data was simulated using the following second-order system with underdamped modes (complex poles), as described in "Data Description" on page 3-75, and has a peak response at 1 rad/s:

$$G(s) = \frac{1}{1 + 0.2s + s^2}e^{-2s}$$

Because the data includes noise at the input during the simulation, the estimated model cannot exactly reproduce the model used to simulate the data.

Examining Model Residuals

You can validate a model by checking the behavior of its residuals.

To generate a Residual Analysis plot, select the **Model resids** check box in the System Identification app.
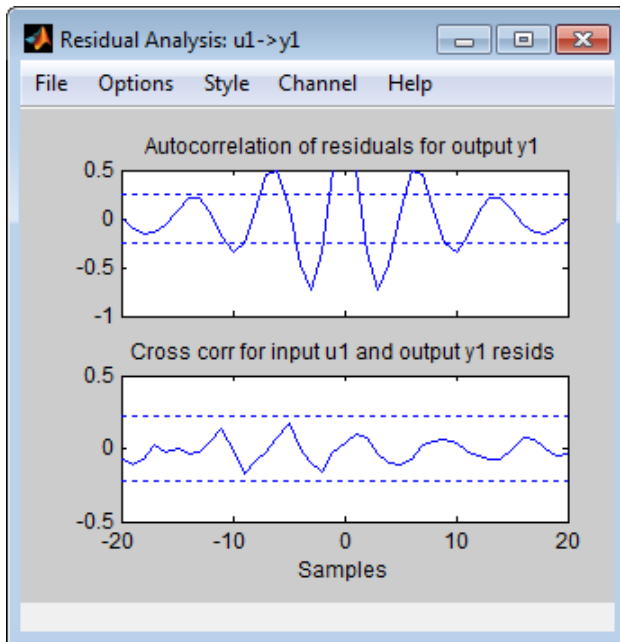
The top axes show the autocorrelation of residuals for the output (whiteness test). The horizontal scale is the number of lags, which is the time difference (in samples) between the signals at which the correlation is estimated. Any fluctuations within the confidence interval are considered to be insignificant. A good model should have a residual autocorrelation function within the confidence interval, indicating that the residuals are uncorrelated. However, in this example, the residuals appear to be correlated, which is natural because the noise model is used to make the residuals white.

The bottom axes show the cross-correlation of the residuals with the input. A good model should have residuals uncorrelated with past inputs (independence test). Evidence of correlation indicates that the model does not describe how a portion of the output relates to the corresponding input. For example, when there is a peak outside the confidence interval for lag k, this means that the contribution to the output $y(t)$ that originates from the input $u(t-k)$ is not properly described by the model. In this example, there is no correlation between the residuals and the inputs.

Thus, residual analysis indicates that this model is good, but that there might be a need for a noise model.

Estimating a Process Model with a Noise Component

Estimating a Second-Order Process Model with Complex Poles

In this portion of the tutorial, you estimate a second-order transfer function and include a noise model. By including a noise model, you optimize the estimation results for prediction application.

You must have already estimated the model, as described in "Estimating a Second-Order Transfer Function Using Default Settings" on page 3-81.

To estimate a second-order transfer function with noise:

1   If the Process Models dialog box is not open, select **Estimate > Process Models** in the System Identification app. This action opens the Process Models dialog box.
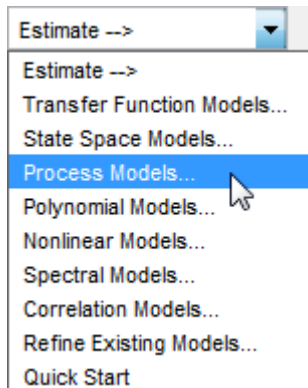
2   In the **Model Transfer Function** area, specify the following options:

- Under **Poles**, select 2 and `Underdamped`. This selection updates the Model Transfer Function to a second-order model structure that can contain complex poles. Make sure that the **Zero** and **Integrator** check boxes are cleared to exclude a zero and an integrator (self-regulating) from the model.
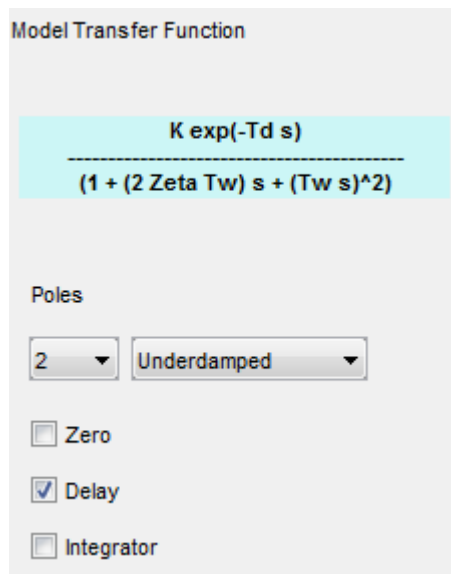
- **Disturbance Model** — Set to `Order 1` to estimate a noise model H as a continuous-time, first-order ARMA model:

$$H = \frac{C}{D}e$$

     where and D are first-order polynomials, and e is white noise.

     This action specifies the **Focus** as `Prediction`, which improves accuracy in the frequency range where the noise level is low. For example, if there is more noise at high frequencies, the algorithm assigns less importance to accurately fitting the high-frequency portions of the data.

- **Name** — Edit the model name to `P2DUe1` to generate a model with a unique name in the System Identification app.

3 Click **Estimate**.

4 In the Process Models dialog box, set the **Disturbance Model** to `Order 2` to estimate a second-order noise model.

5 Edit the **Name** field to `P2DUe2` to generate a model with a unique name in the System Identification app.

6 Click **Estimate**.

Validating the Models

In this portion of the tutorial, you evaluate model performance using the Model Output and the Residual Analysis plots.

You must have already estimated the models, as described in "Estimating a Second-Order Transfer Function Using Default Settings" on page 3-81 and "Estimating a Second-Order Process Model with Complex Poles" on page 3-85.

Comparing the Model Output Plots

To generate the Model Output plot, select the **Model output** check box in the System Identification app. If the plot is empty or a model output does not appear on the plot, click the model icons in the System Identification app window to display these models on the plot.

The following Model Output plot shows the simulated model output, by default. The simulated response of the models is approximately the same for models with and without noise. Thus, including the noise model does not affect the simulated output.
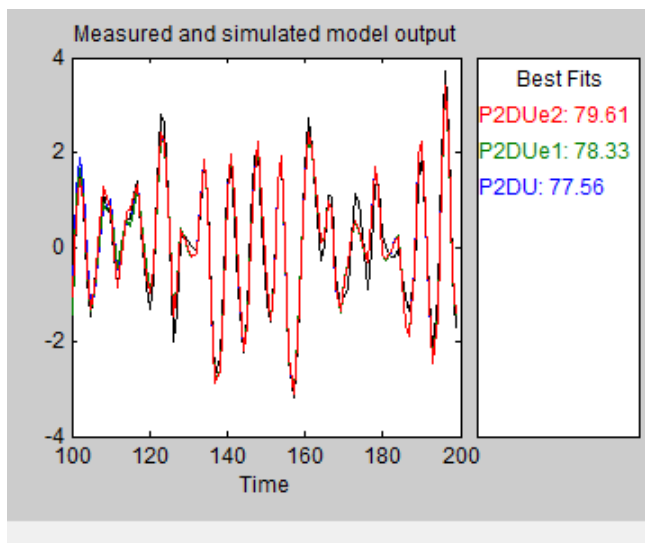
To view the predicted model output, select **Options > 5 step ahead predicted output** in the Model Output plot window.

The following Model Output plot shows that the predicted model output of `P2DUe2` (with a second-order noise model) is better than the predicted output of the other two models (without noise and with a first-order noise model, respectively).
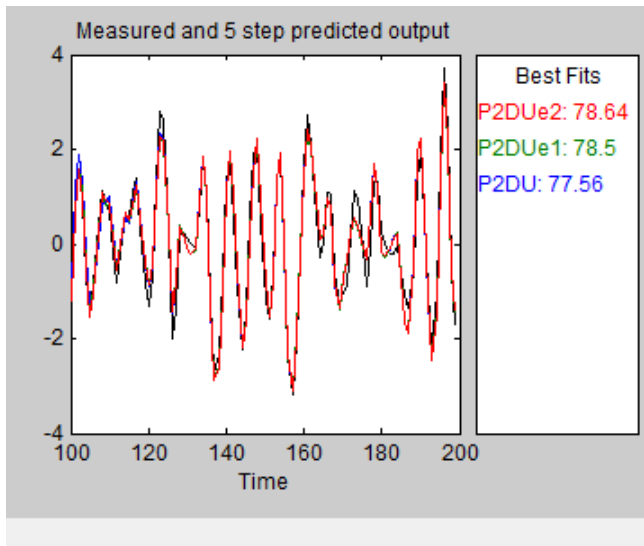
Comparing the Residual Analysis Plots

To generate the Residual Analysis plot, select the **Model resids** check box in the System Identification app. If the plot is empty, click the model icons in the System Identification app window to display these models on the plot.
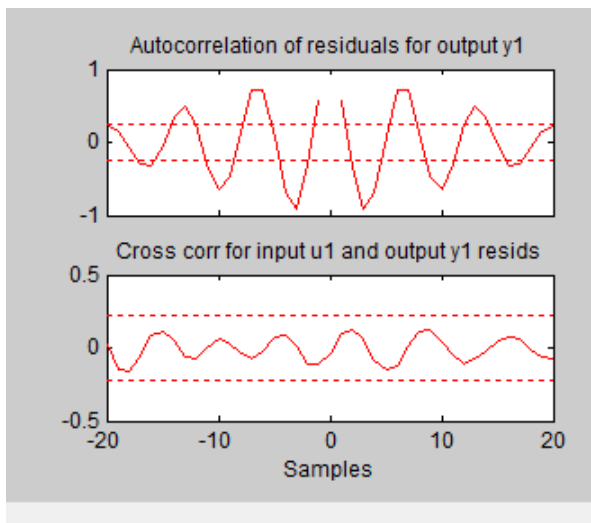
P2DUe2 falls well within the confidence bounds on the Residual Analysis plot.
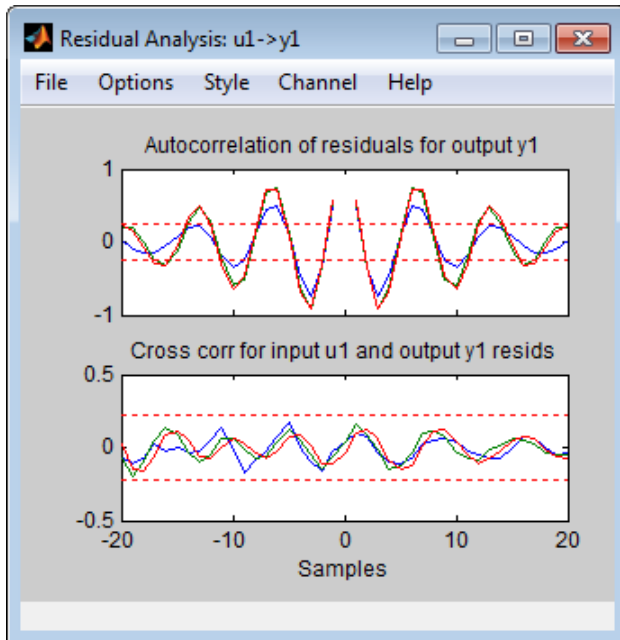
To view residuals for P2DUe2 only, remove models P2DU and P2DUe1 from the Residual Analysis plot by clicking the corresponding icons in the System Identification app.
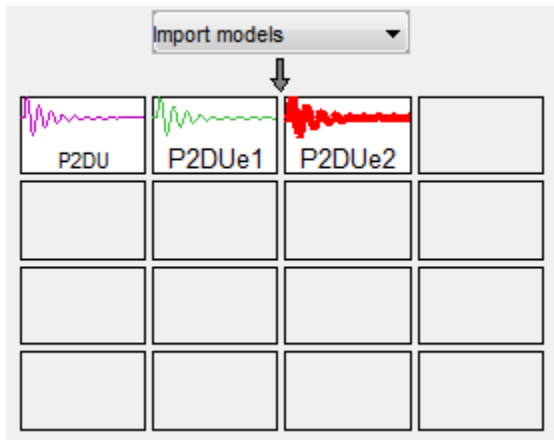
The Residual Analysis plot updates, as shown in the following figure.

The whiteness test for P2DUe2 shows that the residuals are uncorrelated, and the independence test shows no correlation between the residuals and the inputs. These tests indicate that P2DUe2 is a good model.

Viewing Model Parameters

Viewing Model Parameter Values

You can view the numerical parameter values and other information about the model P2DUe2 by right-clicking the model icon in the System Identification app . The Data/model Info dialog box opens.
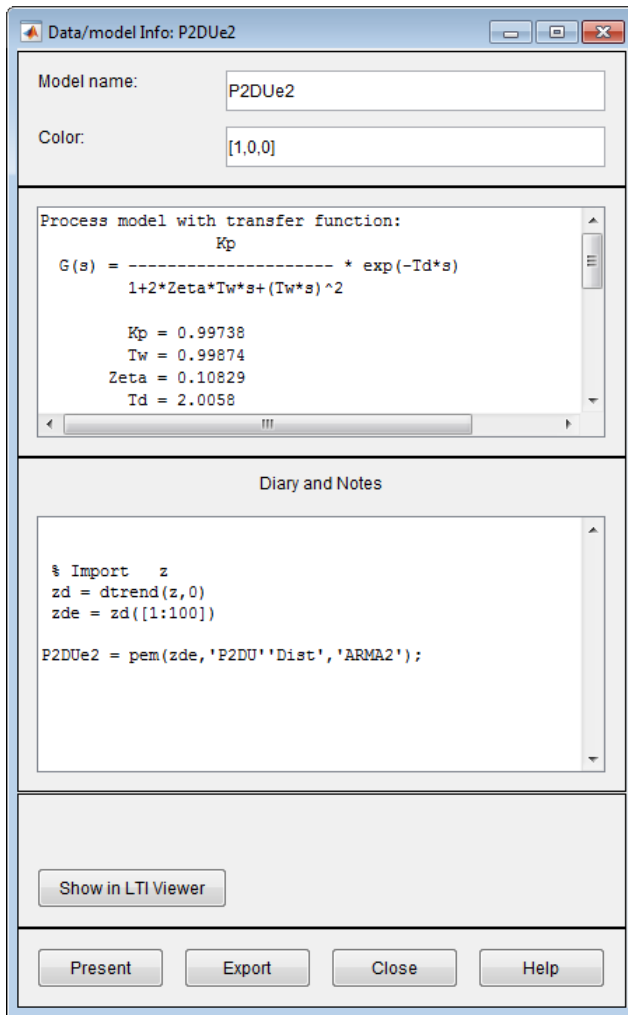
The noneditable area of the dialog box lists the model coefficients that correspond to the following model structure:

$$G(s) = \frac{K}{\left(1 + 2\xi T_w s + T_w{}^2 s^2\right)} e^{-T_d s}$$

The coefficients agree with the model used to simulate the data:

$$G(s) = \frac{1}{1 + 0.2s + s^2} e^{-2s}$$

Viewing Parameter Uncertainties

To view parameter uncertainties for the system transfer function, click **Present** in the Data/model Info dialog box, and view the information in the MATLAB Command Window.

```
Kp = 0.99821 +/- 0.019982
Tw = 0.99987 +/- 0.0037697
Zeta = 0.10828 +/- 0.0042304
Td = 2.004 +/- 0.0029717
```

The 1-standard-deviation uncertainty for each model parameter follows the `+/-` symbol.

P2DUe2 also includes an additive noise term, where H is a second-order ARMA model and e is white noise:

$$H = \frac{C}{D}e$$

The software displays the noise model H as a ratio of two polynomials, `C(s)/D(s)`, where:

```
C(s) = s^2 + 2.186 (+/- 0.08467) s + 1.089 (+/- 0.07951)
D(s) = s^2 + 0.2561 (+/- 0.09044) s + 0.5969 (+/- 0.3046)
```

The 1-standard deviation uncertainty for the model parameters is in parentheses next to each parameter value.

Exporting the Model to the MATLAB Workspace

You can perform further analysis on your estimated models from the MATLAB workspace. For example, if the model is a plant that requires a controller, you can import the model from the MATLAB workspace into the Control System Toolbox product. Furthermore, to simulate your model in the Simulink software (perhaps as part of a larger dynamic system), you can import this model as a Simulink block.

The models you create in the System Identification app are not automatically available in the MATLAB workspace. To make a model available to other toolboxes, Simulink, and the System Identification Toolbox commands, you must export your model from the System Identification app to the MATLAB workspace.

To export the P2DUe2 model, drag the model icon to the **To Workspace** rectangle in the System Identification app. Alternatively, click **Export** in the Data/model Info dialog box. The model now appears in the MATLAB Workspace browser.
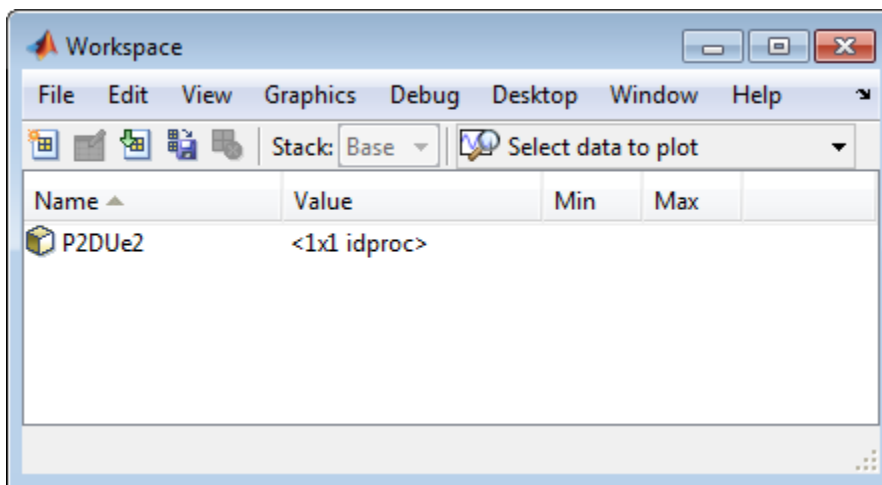
Note This model is an `idproc` model object.

Simulating a System Identification Toolbox Model in Simulink Software

Prerequisites for This Tutorial

In this tutorial, you create a simple Simulink model that uses blocks from the System Identification Toolbox library to bring the data z and the model P2DUe2 into Simulink.

To perform the steps in this tutorial, Simulink must be installed on your computer.

Furthermore, you must have already performed the following steps:

- Load the data set, as described in "Loading Data into the MATLAB Workspace" on page 3-76.
- Estimate the second-order process model, as described in "Estimating a Second-Order Process Model with Complex Poles" on page 3-85.
- Export the model to the MATLAB workspace, as described in "Exporting the Model to the MATLAB Workspace" on page 3-91.

Preparing Input Data

Use the input channel of the data set z as input for simulating the model output by typing the following in the MATLAB Command Window:

```
z_input = z;     % Creates a new iddata object.
z_input.y = []; % Sets the output channel
                % to empty.
```

Alternatively, you can specify any input signal.

Learn More

For more information about representing data signals for system identification, see "Representing Data in MATLAB Workspace".

Building the Simulink Model

To add blocks to a Simulink model:

1
    On the MATLAB **Home** tab, click [icon] **Simulink**.

2
    In the Simulink start page, click **Blank Model**. Then click **Create Model** to open a new model window.

3
    In the Simulink model window, click [icon] to open the Library Browser. In the Library Browser, select the **System Identification Toolbox** library. The right side of the window displays blocks specific to the System Identification Toolbox product.


    Tip Alternatively, to access the System Identification block library, type slident in the MATLAB Command Window.

4
    Drag the following System Identification Toolbox blocks to the new model window:

- IDDATA Sink block
- IDDATA Source block

- IDMODEL model block

5 In the Simulink Library Browser, select the **Simulink > Sinks** library, and drag the Scope block to the new model window.

6 In the Simulink model window, connect the blocks to resembles the following figure.
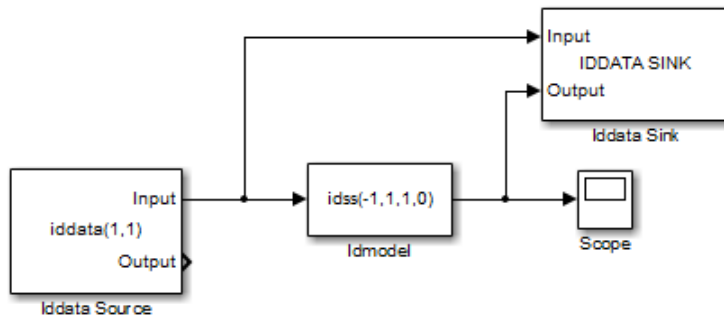
Next, you configure these blocks to get data from the MATLAB workspace and set the simulation time interval and duration.

Configuring Blocks and Simulation Parameters

This procedure guides you through the following tasks to configure the model blocks:

- Getting data from the MATLAB workspace.
- Setting the simulation time interval and duration.

1 In the Simulink Editor, select **Modeling > Model Settings > Model Settings Ctrl+E**.

2 In the Configuration Parameters dialog box, in the **Solver** subpane, in the **Stop time** field, type 200. Click **OK**.

This value sets the duration of the simulation to 200 seconds.

3 Double-click the Iddata Source block to open the Source Block Parameters: Iddata Source dialog box. Then, type the following variable name in the **IDDATA object** field:

```
z_input
```

This variable is the data object in the MATLAB workspace that contains the input data.
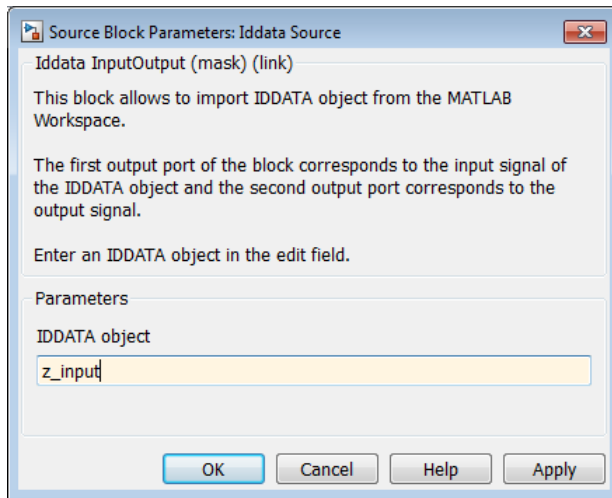
Tip As a shortcut, you can drag and drop the variable name from the MATLAB Workspace browser to the **IDDATA object** field.

Click **OK**.

4 Double-click the Idmodel block to open the Function Block Parameters: Idmodel dialog box.

a Type the following variable name in the **Model variable** field:

P2DUe2

This variable represents the name of the model in the MATLAB workspace.

b Clear the **Add noise** check box to exclude noise from the simulation. Click **OK**.

When **Add noise** is selected, Simulink derives the noise amplitude from the `NoiseVariance` property of the model and adds noise to the model accordingly. The simulation propagates this noise according to the noise model H that was estimated with the system dynamics:

$$H = \frac{C}{D}e$$
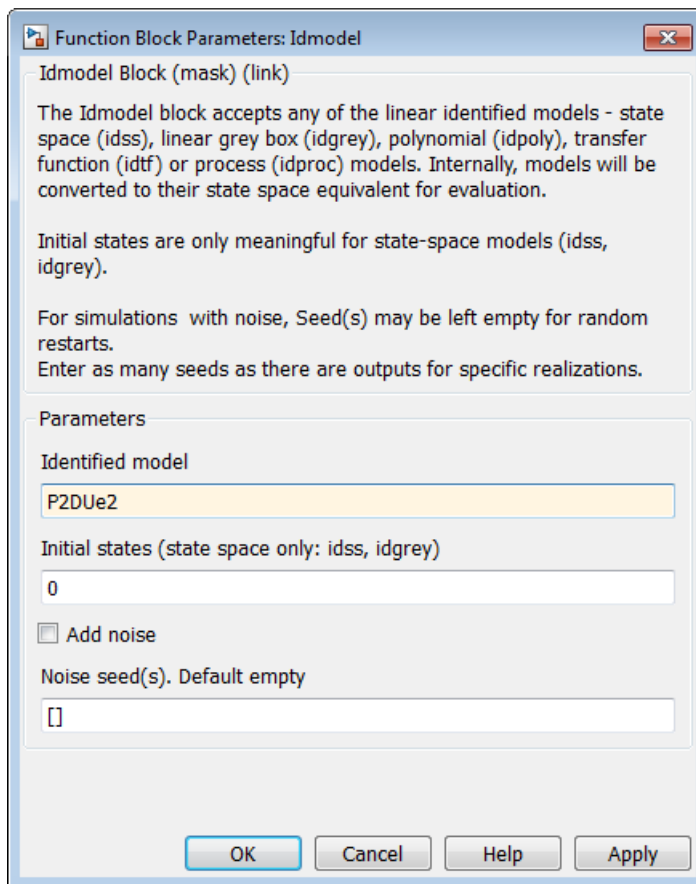
Click **OK**.

5 Double-click the Iddata Sink block to open the Sink Block Parameters: Iddata Sink dialog box. Type the following variable name in the **IDDATA Name** field:

```
z_sim_out
```

6 Type 1 in the **Sample Time (sec.)** field to set the sample time of the output data to match the sample time of the input data.
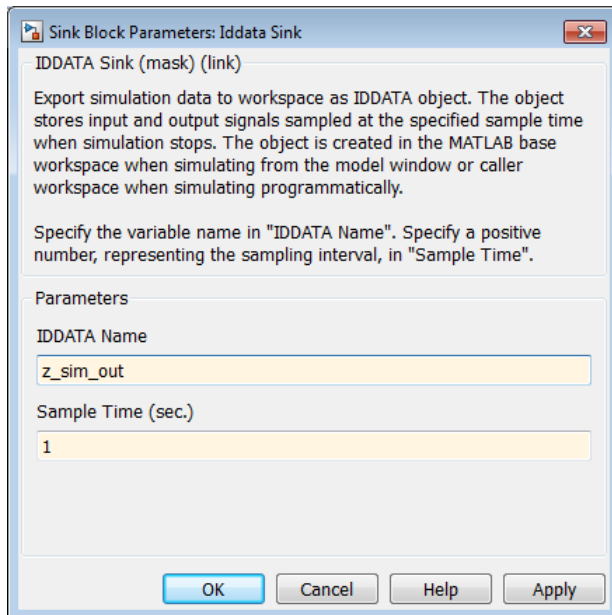
Click **OK**.

The resulting change to the Simulink model is shown in the following figure.
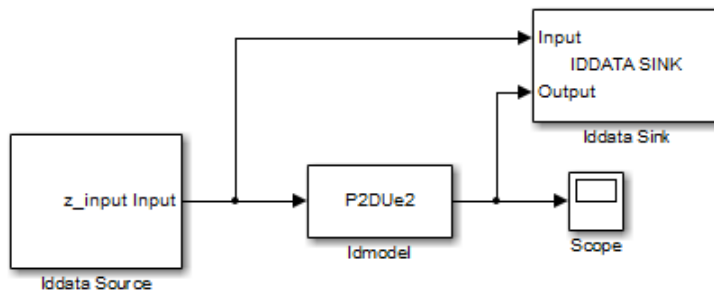
Running the Simulation

1   In the Simulink Editor, select **Simulation** > **Run**.

2   Double-click the Scope block to display the time plot of the model output.
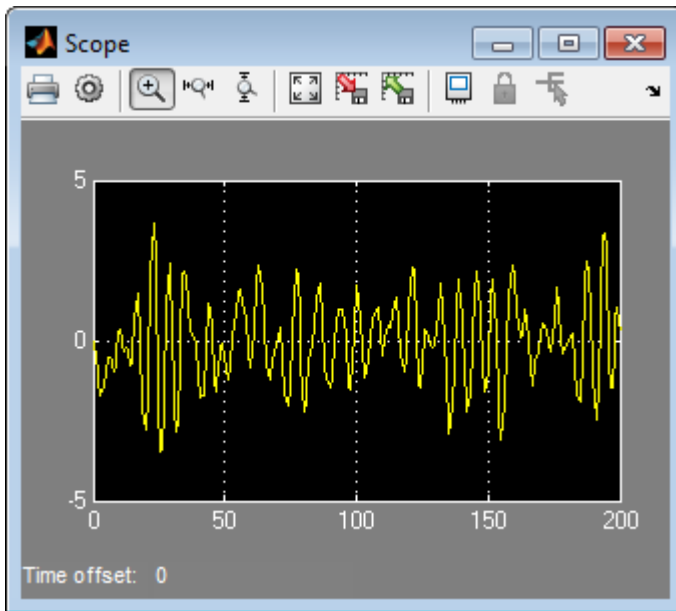
3   In the MATLAB Workspace browser, notice the variable `z_sim_out` that stores the model output as an `iddata` object. You specified this variable name when you configured the Iddata Sink block.

This variable stores the simulated output of the model, and it is now available for further processing and exploration.

See Also

More About

*   "What Is a Process Model?"
*   "Estimate Process Models Using the App"

Estimating Models Using Frequency-Domain Data

The System Identification Toolbox software lets you use frequency-domain data to identify linear models at the command line and in the **System Identification** app. You can estimate both continuous-time and discrete-time linear models using frequency-domain data. This topic presents an overview of model estimation in the toolbox using frequency-domain data. For an example of model estimation using frequency-domain data, see "Frequency Domain Identification: Estimating Models Using Frequency Domain Data".

Frequency-domain data can be of two types:

- *Frequency domain input-output data* — You obtain the data by computing Fourier transforms of time-domain input, $u(t)$, and output, $y(t)$, signals. The data is the set of input, $U(\omega)$, and output, $Y(\omega)$, signals in frequency domain. In the toolbox, frequency-domain input-output data is represented using `iddata` objects. For more information, see "Representing Frequency-Domain Data in the Toolbox" on page 3-99.

- *Frequency-response data* — Also called frequency function or frequency-response function (FRF), the data consists of transfer function measurements, $G(i\omega)$, of a system at a discrete set of frequencies ω. Frequency-response data at a frequency ω tells you how a linear system responds to a sinusoidal input of the same frequency. In the toolbox, frequency-response data is represented using `idfrd` objects. For more information, see "Representing Frequency-Domain Data in the Toolbox" on page 3-99. You can obtain frequency-response data in the following ways:

  - Measure the frequency-response data values directly, such as by using a spectrum analyzer.

  - Perform spectral analysis of time-domain or frequency-domain input-output data (`iddata` objects) using commands such as `spa` and `spafdr`.

  - Compute the frequency-response of an identified linear model using commands such as `freqresp`, `bode`, and `idfrd`.

The workflow for model estimation on page 2-3 using frequency-domain data is the same as that for estimation using time-domain data. If needed, you first prepare the data for model identification by removing outliers and filtering the data. You then estimate a linear parametric model from the data, and validate the estimation.

Advantages of Using Frequency-Domain Data

Using frequency-domain data has the following advantages:

- Data compression — You can compress long records of data when you convert time-domain data to frequency domain. For example, you can use logarithmically spaced frequencies.

- Non uniformity — Frequency-domain data does not have to be uniformly spaced. Your data can have frequency-dependent resolution so that more data points are used in the frequency regions of interest. For example, the frequencies of interest could be the bandwidth range of a system, or near the resonances of a system.

- Prefiltering — Prefiltering of data in the frequency-domain becomes simple. It corresponds to assigning different weights to different frequencies of the data.

- Continuous-time signal - You can represent continuous-time signals using frequency-domain data and use the data for estimation.

Representing Frequency-Domain Data in the Toolbox

Before performing model estimation, you specify the frequency-domain data as objects in the toolbox. You can specify both continuous-time and discrete-time frequency-domain data on page 3-102.

- **Frequency domain input-output data** — Specify as an `iddata` object. In the object, you store $U(\omega)$, $Y(\omega)$, and frequency vector ω. The `Domain` property of the object is `'Frequency'`, to specify that the object contains frequency-domain signals. If $U(\omega)$, $Y(\omega)$ are discrete-time Fourier transforms of discrete-time signals, sampled with sampling interval `Ts`, denote the sampling interval in the `iddata` object. If $U(\omega)$, $Y(\omega)$ are Fourier transforms of continuous-time signals, specify `Ts` as `0` in the `iddata` object.

  To plot the data at the command line, use the `plot` command.

  For example, you can plot the phase and magnitude of frequency-domain input-output data.

  Load time-domain input-output data.

  ```
  load iddata1 z1
  ```

  The time-domain inputs `u` and outputs `y` are stored in `z1`, an `iddata` object whose `Domain` property is set to `'Time'`.

  Fourier-transform the data to obtain frequency-domain input-output data.

  ```
  zf = fft(z1);
  ```

  The `Domain` property of `zf` is set to `'Frequency'`, indicating that it is frequency-domain data.

  Plot the magnitude and phase of the frequency-domain input-output data.

  ```
  plot(zf)
  ```

Input-Output Data
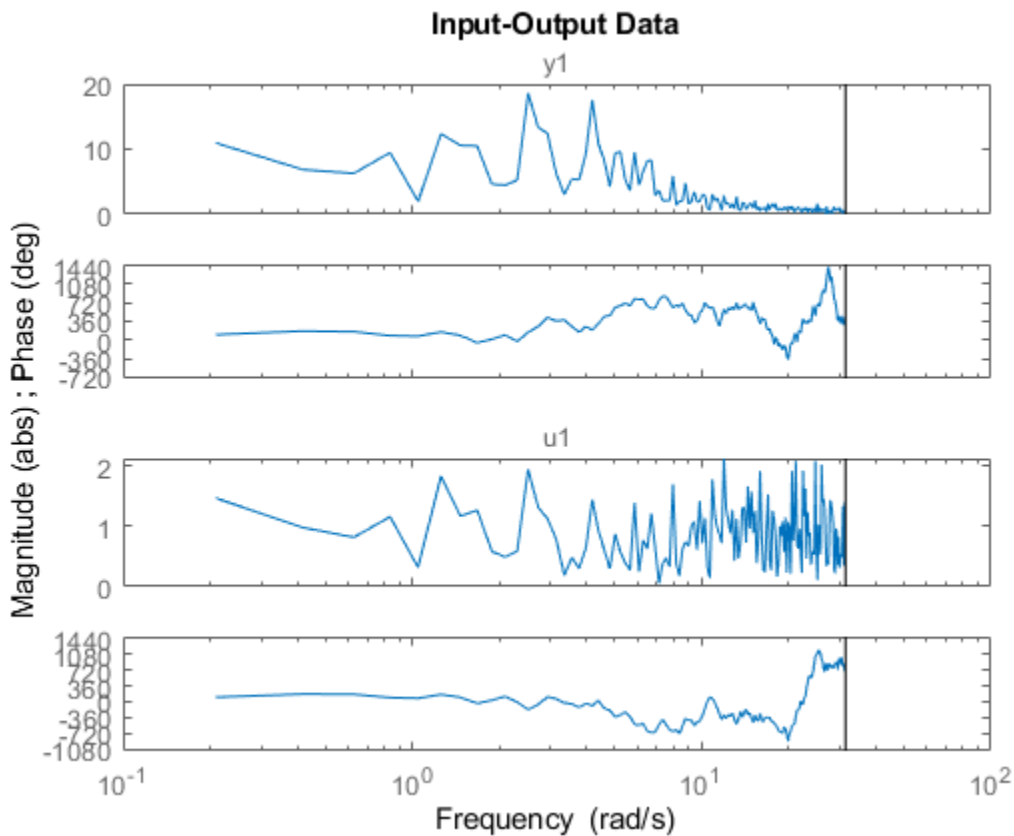
- **Frequency-response data** — Specify as an `idfrd` object. If you have Control System Toolbox software, you can also specify the data as an `frd` object.

  To plot the data at the command line, use the `bode` command.

  For example, you can plot the frequency-response of a transfer function model.

  Create a transfer function model of your system.

  ```
  sys = tf([1 0.2],[1 2 1 1]);
  ```

  Calculate the frequency-response of the transfer function model, `sys`, at 100 frequency points. Specify the range of the frequencies as 0.1 rad/s to 10 rad/s.

  ```
  freq = logspace(-1,1,100);
  frdModel = idfrd(sys,freq);
  ```

  Plot the frequency-response of the model.

  ```
  bode(frdModel)
  ```
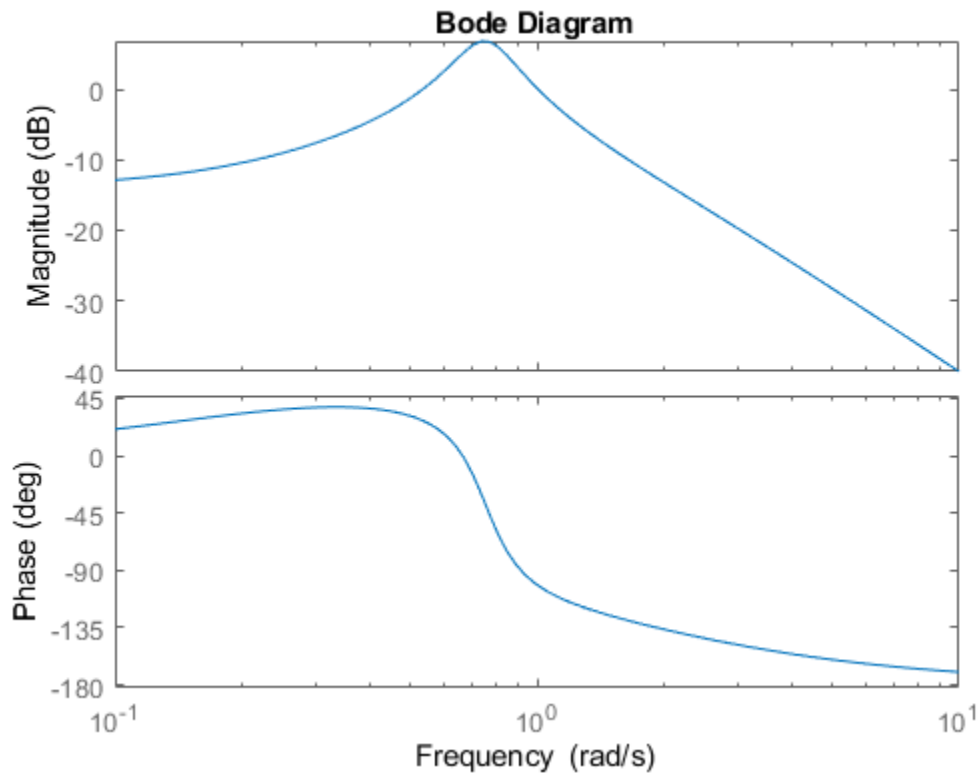
For more information about the frequency-domain data types and how to specify them, see "Frequency-Domain Data Representation".

You can also transform between frequency-domain and time-domain data types using the following commands.

Original Data Format	To Time-Domain Data (iddata object)	To Frequency-Domain Data (iddata object)	To Frequency-Response Data (idfrd object)
Time-Domain Data (iddata object)	N/A	Use fft	- Use etfe, spa, or spafdr. - Estimate a linear parametric model from the iddata object, and use idfrd to compute frequency-response data.
Frequency-Domain Data (iddata object)	Use ifft (works only for evenly spaced frequency-domain data).	N/A	- Use etfe, spa, or spafdr. - Estimate a linear parametric model from the iddata object, and use idfrd to compute frequency-response data.

Original Data Format	To Time-Domain Data (iddata object)	To Frequency-Domain Data (iddata object)	To Frequency-Response Data (idfrd object)
Frequency-Response Data (idfrd object)	Not supported	Use iddata. The software creates a frequency-domain iddata object that has the same ratio between output and input as the original idfrd object frequency-response data.	• Use spafdr. The software calculates frequency-response data with a different resolution (number and spacing of frequencies) than the original data.

For more information about transforming between data types in the app or at the command line, see the "Transform Data" category page.

Continuous-Time and Discrete-Time Frequency-Domain Data

Unlike time-domain data, the sample time Ts of frequency-domain data can be zero. Frequency-domain data with zero Ts is called continuous-time data. Frequency-domain data with Ts greater than zero is called discrete-time data.

You can obtain continuous-time frequency-domain data (Ts = 0) in the following ways:

• Generate the data from known continuous-time analytical expressions.

  For example, suppose that you know the frequency-response of your system is $G(\omega) = 1/(b + j\omega)$, where b is a constant. Also assume that the time-domain inputs to your system are, $u(t) = e^{-at}\sin w_0 t$, where a is a constant greater than zero, and $u(t)$ is zero for all times t less than zero. You can compute the Fourier transform of $u(t)$ to obtain

  $$U(\omega) = \omega_0/[(a + j\omega)^2 + w_0{}^2]$$

  Using $U(\omega)$ and $G(\omega)$ you can then get the frequency-domain expression for the outputs:

  $$Y(\omega) = G(\omega)U(\omega)$$

  You can now evaluate the analytical expressions for $Y(\omega)$ and $U(\omega)$ over a grid of frequency values ($\omega_{grid} = \omega_1, \omega_2, ..., \omega_n$), and get a vector of frequency-domain input-output data values (Y_{grid}, U_{grid}). You can package the input-output data as a continuous-time iddata object by specifying a zero sample time, Ts.

  ```
  Ts = 0;
  zf = iddata(Ygrid,Ugrid,Ts,'Frequency',wgrid)
  ```
• Compute the frequency response of a continuous-time linear system at a grid of frequencies.

  For example, in the following code, you generate continuous-time frequency-response data, FRDc, from a continuous-time transfer function model, sys for a grid of frequencies, freq.

  ```
  sys = idtf(1,[1 2 2]);
  freq = logspace(-2,2,100);
  FRDc = idfrd(sys,freq);
  ```

- Measure amplitudes and phases from a sinusoidal experiment, where the measurement system uses anti-aliasing filters. You measure the response of the system to sinusoidal inputs at different frequencies, and package the data as an `idfrd` object. For example, the frequency-response data measured with a spectrum analyzer is continuous-time.

  You can also conduct an experiment by using periodic, continuous-time signals (multiple sine waves) as inputs to your system and measuring the response of your system. Then you can package the input and output data as an `iddata` object.

You can obtain discrete-time frequency-domain data (Ts >0) in the following ways:

- Transform the measured time-domain values using a discrete Fourier transform.

  For example, in the following code, you compute the discrete Fourier transform of time-domain data, `y`, that is measured at discrete time-points with sample time 0.01 seconds.

  ```
  t = 0:0.01:10;
  y = iddata(sin(2*pi*10*t),[],0.01);
  Y = fft(y);
  ```

- Compute the frequency response of a discrete-time linear system.

  For example, in the following code, you generate discrete-time frequency-response data, `FRDd`, from a discrete-time transfer function model, `sys`. You specify a non-zero sample time for creating the discrete-time model.

  ```
  Ts = 1;
  sys = idtf(1,[1 0.2 2.1],Ts);
  FRDd = idfrd(sys,logspace(-2,2,100));
  ```

You can use continuous-time frequency-domain data to identify only continuous-time models. You can use discrete-time frequency-domain data to identify both discrete-time and continuous-time models. However, identifying continuous-time models from discrete-time data requires knowledge of the intersample behavior of the data. For more information, see "Estimating Continuous-Time and Discrete-Time Models" on page 3-105.

Note For discrete-time data, the software ignores frequency-domain data above the Nyquist frequency during estimation.

Preprocessing Frequency-Domain Data for Model Estimation

After you have represented your frequency-domain data using `iddata` or `idfrd` objects, you can prepare the data for estimation by removing spurious data and by filtering the data.

To view the spurious data, plot the data in the app, or use the `plot` (for `iddata` objects) or `bode` (for `idfrd` objects) commands. After identifying the spurious data in the plot, you can remove them. For example, if you want to remove data points 20–30 from `zf`, a frequency-domain `iddata` object, use the following syntax:

```
zf(20:30) = [];
```

Since frequency-domain data does not have to be specified with a uniform spacing, you do not need to replace the outliers.

You can also prefilter high-frequency noise in your data. You can prefilter frequency-domain data in the app, or use `idfilt` at the command line. Prefiltering data can also help remove drifts that are low-frequency disturbances. In addition to minimizing noise, prefiltering lets you focus your model on specific frequency bands. The frequency range of interest often corresponds to a passband over the breakpoints on a Bode plot. For example, if you are modeling a plant for control-design applications, you can prefilter the data to enhance frequencies around the desired closed-loop bandwidth.

For more information, see "Filtering Data".

Estimating Linear Parametric Models

After you have preprocessed the frequency-domain data, you can use it to estimate continuous-time and discrete-time models on page 3-105.

Supported Model Types

You can estimate the following linear parametric models using frequency-domain data. The noise component of the models is not estimated, except for ARX models.

Model Type	Additional Information	Estimation Commands	Estimation in the App
"Transfer Function Models"		• `tfest`	See "Estimate Transfer Function Models in the System Identification App".
"State-Space Models"	Estimated K matrix of the state-space model is zero.	• `ssest` • `n4sid`	See "Estimate State-Space Models in System Identification App".
"Process Models"	Disturbance model is not estimated.	• `procest`	See "Estimate Process Models Using the App".
"Input-Output Polynomial Models"	You can estimate only output-error and ARX models.	• `oe` • `arx` • `iv4` • `ivx` • `polyest` with `na`, `nc`, and `nd` orders of the polynomial specified as zero	See "Estimate Polynomial Models in the App".
"Linear Grey-Box Models"	Model parameters that are only related to the noise matrix K are not estimated.	• `greyest`	Grey-box model estimation is not available in the app.
"Correlation Models" (Impulse-response models)		• `impulseest`	See "Estimate Impulse-Response Models Using System Identification App".

Model Type	Additional Information	Estimation Commands	Estimation in the App
"Frequency-Response Models" (Estimated as `idfrd` objects)		• `etfe` • `spa` • `spafdr`	See "Estimate Frequency-Response Models in the App".

Before performing the estimation, you can specify estimation options, such as how the software treats initial conditions of the estimation data. To do so at the command line, use the estimation option set corresponding to the estimation command. For example, suppose that you want to estimate a transfer function model from frequency-domain data, `zf`, and you also want to estimate the initial conditions of the data. Use the `tfestOptions` option set to specify the estimation options, and then estimate the model.

```
opt = tfestOptions('InitialCondition','estimate');
sys = tfest(zf,opt);
```

`sys` is the estimated transfer function model. For information about extracting estimated parameter values from the model, see "Extracting Numerical Model Data". After performing the estimation, you can validate the estimated model on page 3-108.

Note A zero initial condition for time-domain data does not imply a zero initial condition for the corresponding frequency-domain data. For time-domain data, zero initial conditions mean that the system is assumed to be in a state of rest before the start of data collection. In the frequency-domain, initial conditions can be ignored only if the data collected is periodic in nature. Thus, if you have time-domain data collected with zero initial conditions, and you convert it to frequency-domain data to estimate a model, you have to estimate the initial conditions as well. You cannot specify them as zero.

You cannot perform the following estimations using frequency-domain data:

- Estimation of the noise component of a linear model, except for ARX models.
- Estimation of nonlinear models.
- Estimation of time series models using spectrum data only. Spectrum data is the power spectrum of a signal, commonly stored in the `SpectrumData` property of an `idfrd` object.
- Online estimation using recursive algorithms.

Estimating Continuous-Time and Discrete-Time Models

You can estimate all the supported linear models on page 3-104 as discrete-time models, except for process models. Process models are defined in continuous-time only. For the estimation of discrete-time models, you must use discrete-time data.

You can estimate all the supported linear models as continuous-time models, except for correlation models (see `impulseest`). You can estimate continuous-time models using both continuous-time and discrete-time data. For information about continuous-time and discrete-time data, see "Continuous-Time and Discrete-Time Frequency-Domain Data" on page 3-102.

If you are estimating a continuous-time model using discrete-time data, you must specify the *intersample behavior* of the data. The specification of intersample behavior depends on the type of frequency-domain data.

- Discrete-time frequency-domain input-output data (`iddata` object) — Specify the intersample behavior of the time-domain input signal $u(t)$ that you Fourier transformed to obtain the frequency-domain input signal $U(\omega)$.

- Discrete-time frequency-response data (`idfrd` object) — The data is generated by computing the frequency-response of a discrete-time model. Specify the intersample behavior as the discretization method assumed to compute the discrete-time model from an underlying continuous-time model. For an example, see "Specify Intersample Behavior for Discrete-Time Frequency-Response Data" on page 3-106.

You can specify the intersample behavior to be piecewise constant (zero-order hold), linearly interpolated between the samples (first-order hold), or band-limited. If you specify the discrete-time data from your system as band-limited (that is no power above the Nyquist frequency), the software treats the data as continuous-time by setting the sample time to zero. The software then estimates a continuous-time model from the data. For more information, see "Effect of Input Intersample Behavior on Continuous-Time Models".

Specify Intersample Behavior for Discrete-Time Frequency-Response Data

This example shows the effect of intersample behavior on the estimation of continuous-time models using discrete-time frequency-response data.

Generate discrete-time frequency-response data. To do so, first construct a continuous-time transfer function model, `sys`. Then convert it to a discrete-time model, `sysd`, using the `c2d` command and first-order hold (FOH) method. Use the discrete-time model `sysd` to generate frequency-response data at specified frequencies, `freq`.

```
sys = idtf([1 0.2],[1 2 1 1]);
sysd = c2d(sys,1,c2dOptions('Method','foh'));
freq = logspace(-1,0,10);
FRdata = idfrd(sysd,freq);
```

`FRdata` is discrete-time data. The software sets the `InterSample` property of `FRdata` to `'foh'`, which is the discretization method that was used to obtain `sysd` from `sys`.

Estimate a third-order continuous-time transfer function from the discrete-time data.

```
model1 = tfest(FRdata,3,1)

model1 =

        s + 0.2
  -------------------
  s^3 + 2 s^2 + s + 1

Continuous-time identified transfer function.

Parameterization:
   Number of poles: 3   Number of zeros: 1
   Number of free coefficients: 5
   Use "tfdata", "getpvec", "getcov" for parameters and their uncertainties.

Status:
Estimated using TFEST on frequency response data "FRdata".
Fit to estimation data: 100%
FPE: 5.173e-31, MSE: 1.748e-31
```

`model1` is a continuous-time model, estimated using discrete-time frequency-response data. The underlying continuous-time dynamics of the original third-order model `sys` are retrieved in `model1` because the correct intersample behavior is specified in `FRdata`.

Now, specify the intersample behavior as zero-order hold (ZOH), and estimate a third-order transfer function model.

```
FRdata.InterSample = 'zoh';
model2 = tfest(FRdata,3,1)

model2 =

         -15.57 s - 3.305
  ------------------------------
   s^3 - 30.01 s^2 - 6.957 s - 17

Continuous-time identified transfer function.

Parameterization:
   Number of poles: 3    Number of zeros: 1
   Number of free coefficients: 5
   Use "tfdata", "getpvec", "getcov" for parameters and their uncertainties.

Status:
Estimated using TFEST on frequency response data "FRdata".
Fit to estimation data: 94.91%
FPE: 0.004499, MSE: 0.001501
```

`model2` does not capture the dynamics of the original model `sys`. Thus, sampling related errors are introduced in the model estimation when the intersample behavior is not correctly specified.

Convert Frequency-Response Data Model to a Transfer Function

This example shows how to convert a frequency-response data (FRD) model to a transfer function model. You treat the FRD model as estimation data and then estimate the transfer function.

Obtain an FRD model.

For example, use `bode` to obtain the magnitude and phase response data for the following fifth-order system:

$$G(s) = \frac{s + 0.2}{s^5 + s^4 + 0.8s^3 + 0.4s^2 + 0.12s + 0.04}$$

Use 100 frequency points between 0.1 rad/s to 10 rad/s to obtain the FRD model. Use `frd` to create a frequency-response model object.

```
freq = logspace(-1,1,100);
sys0 = tf([1 0.2],[1 1  0.8  0.4  0.12  0.04]);
[mag,phase] = bode(sys0,freq);
frdModel = frd(mag.*exp(1j*phase*pi/180),freq);
```

Obtain the best third-order approximation to the system dynamics by estimating a transfer function with 3 zeros and 3 poles.

```
np = 3;
nz = 3;
sys = tfest(frdModel,np,nz);
```

`sys` is the estimated transfer function.

Compare the response of the FRD Model and the estimated transfer function model.

```
bode(frdModel,sys,freq(1:50));
```


Bode Diagram
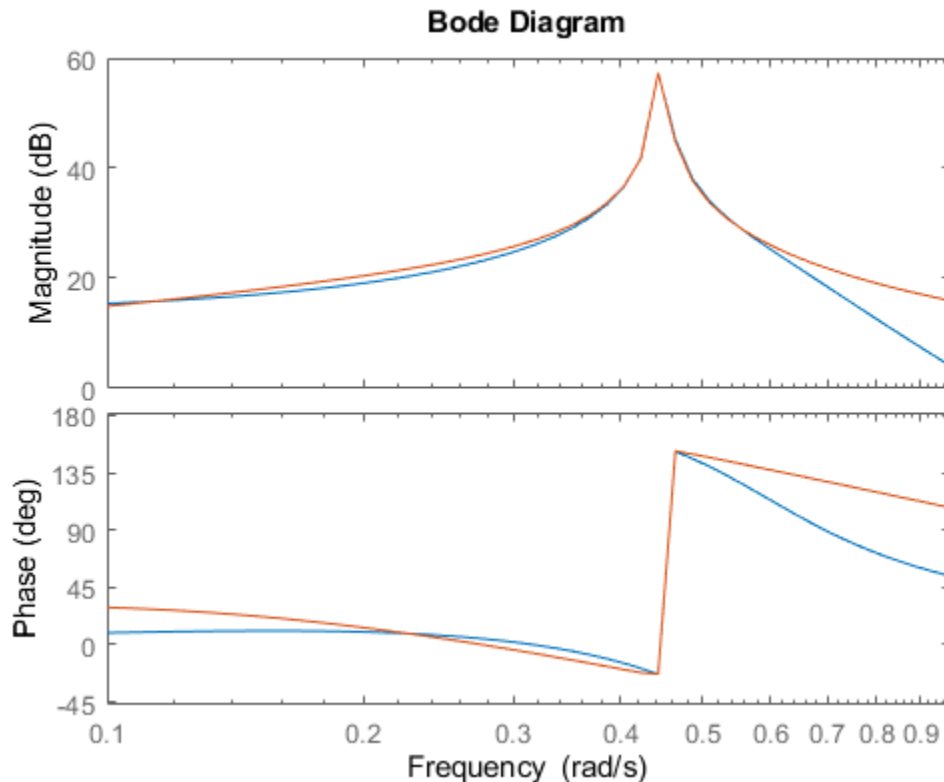
The FRD model is generated from the fifth-order system `sys0`. While `sys`, a third-order approximation, does not capture the entire response of `sys0`, it captures the response well until approximately 0.6 rad/s.

Validating Estimated Model

After estimating a model for your system, you can validate whether it reproduces the system behavior within acceptable bounds. It is recommended that you use separate data sets for estimating and validating your model. You can use time-domain or frequency-domain data to validate a model estimated using frequency-domain data. If you are using input-output validation data to validate the estimated model, you can compare the simulated model response to the measured validation data output. If your validation data is frequency-response data, you can compare it to the frequency response of the model. For example, to compare the output of an estimated model `sys` to measured validation data `zv`, use the following syntax:

```
compare(zv,sys);
```

You can also perform a residual analysis. For more information see, "Validating Models After Estimation".

Troubleshooting Frequency-Domain Identification

When you estimate a model using frequency-domain data, the estimation algorithm minimizes a loss (cost) function. For example, if you estimate a SISO linear model from frequency-response data `f`, the estimation algorithm minimizes the following least-squares loss function:

$$\underset{G(\omega)}{\text{minimize}} \sum_{k=1}^{N_f} |W(\omega_k)(G(\omega_k) - f(\omega_k))|^2$$

Here `W` is a frequency-dependent weight that you specify, `G` is the linear model that is to be estimated, ω is the frequency, and N_f is the number of frequencies at which the data is available. The quantity $(G(\omega_k) - f(\omega_k))$ is the frequency-response error. For frequency-domain input-output data, the algorithm minimizes the weighted norm of the output error instead of the frequency-response error. For more information, see "Loss Function and Model Quality Metrics". During estimation, spurious or uncaptured dynamics in your data can effect the loss function and result in unsatisfactory model estimation.

- *Unexpected, spurious dynamics* — Typically observed when the high magnitude regions of data have low signal-to-noise ratio. The fitting error around these portions of data has a large contribution to the loss function. As a result the estimation algorithm may overfit and assign unexpected dynamics to noise in these regions. To troubleshoot this issue:

  - Improve signal-to-noise ratio — You can gather more than one set of data, and average them. If you have frequency-domain input-output data, you can combine multiple data sets by using the `merge` command. Use this data for estimation to obtain an improved result. Alternatively, you can filter the dataset, and use it for estimation. For example, use a moving-average filter over the data to smooth the measured response. Apply the smoothing filter only in regions of data where you are confident that the unsmoothness is due to noise, and not due to system dynamics.

  - Reduce the impact of certain portions of data on the loss function — You can specify a frequency-dependent weight. For example, if you are estimating a transfer function model, specify the weight in the `WeightingFilter` option of the estimation option set `tfestOptions`. Specify a small weight in frequency regions where the spurious dynamics exist. Alternatively, use fewer data points around this frequency region.

- *Uncaptured dynamics* — Typically observed when the dynamics you want to capture have a low magnitude relative to the rest of data. Since a poor fit to low magnitude data contributes less to the loss function, the algorithm may ignore these dynamics to reduce errors at other frequencies. To troubleshoot this issue:

  - Specify a frequency-dependent weight — Specify a large weight for the frequency region where you would like to capture dynamics.

  - Use more data points around this region.

For an example of these troubleshooting techniques, see "Troubleshoot Frequency-Domain Identification of Transfer Function Models".

If you do not achieve a satisfactory model using these troubleshooting techniques, try a different model structure or estimation algorithm.

Next Steps After Identifying a Model

After estimating a model, you can perform model transformations, extract model parameters, and simulate and predict the model response. Some of the tasks you can perform are:

- "Transforming Between Discrete-Time and Continuous-Time Representations"
- "Transforming Between Linear Model Representations" — Transform between linear parametric model representations, such as between polynomial, state-space, and zero-pole representations.
- "Extracting Numerical Model Data" — For example, extract the poles and zeros of the model using `pole` and `zero` commands, respectively. Compute the model frequency response for a specified set of frequencies using `freqresp`.
- Simulating and predicting model response
- Using the model for control design

See Also

More About

- "System Identification Workflow" on page 2-3
- "Frequency Domain Identification: Estimating Models Using Frequency Domain Data"
- "Effect of Input Intersample Behavior on Continuous-Time Models"
- "Troubleshoot Frequency-Domain Identification of Transfer Function Models"

Nonlinear Model Identification

Identify Nonlinear Black-Box Models Using System Identification App

Introduction

Objectives

Estimate and validate nonlinear models from single-input/single-output (SISO) data to find the one that best represents your system dynamics.

After completing this tutorial, you will be able to accomplish the following tasks using the **System Identification** app:

- Import data objects from the MATLAB workspace into the app.
- Estimate and validate nonlinear models from the data.
- Plot and analyze the behavior of the nonlinearities.

Data Description

This tutorial uses the data file `twotankdata.mat`, which contains SISO time-domain data for a two-tank system, shown in the following figure.

Two-Tank System

In the two-tank system, water pours through a pipe into Tank 1, drains into Tank 2, and leaves the system through a small hole at the bottom of Tank 2. The measured input $u(t)$ to the system is the voltage applied to the pump that feeds the water into Tank 1 (in volts). The measured output $y(t)$ is the height of the water in the lower tank (in meters).

Based on Bernoulli's law, which states that water flowing through a small hole at the bottom of a tank depends nonlinearly on the level of the water in the tank, you expect the relationship between the input and the output data to be nonlinear.

`twotankdata.mat` includes 3000 samples with a sample time of 0.2 s.

What Are Nonlinear Black-Box Models?

Types of Nonlinear Black-Box Models

You can estimate nonlinear discrete-time black-box models for both single-output and multiple-output time-domain data. You can choose from two types of nonlinear, black-box model structures:

- Nonlinear ARX models
- Hammerstein-Wiener models

Note You can estimate Hammerstein-Wiener black-box models from input/output data only. These models do not support time-series data, where there is no input.

For more information on estimating nonlinear black-box models, see "Nonlinear Model Identification".

What Is a Nonlinear ARX Model?

A nonlinear ARX model consists of model regressors and an output function. The output function includes linear and nonlinear functions that act on the model regressors to give the model output and a fixed offset for that output. This block diagram represents the structure of a nonlinear ARX model in a simulation scenario.
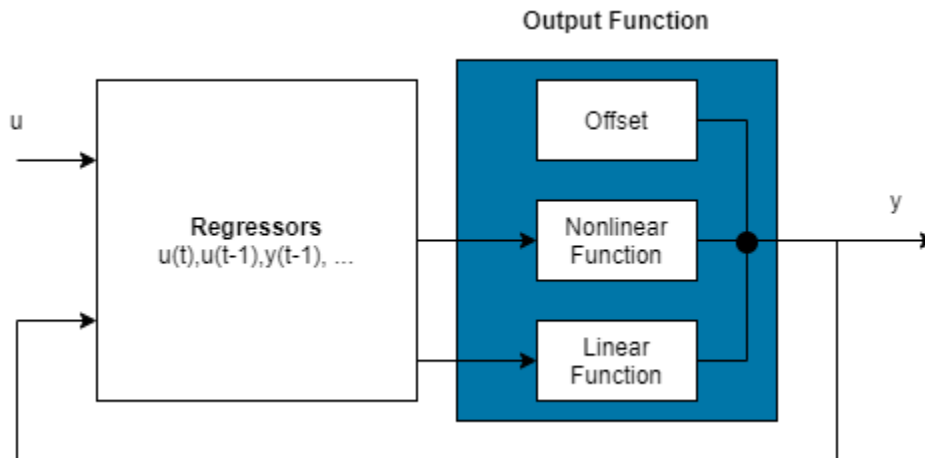
The software computes the nonlinear ARX model output y in two stages:

1   It computes regressor values from the current and past input values and the past output data.

In the simplest case, regressors are delayed inputs and outputs, such as $u(t–1)$ and $y(t–3)$. These kind of regressors are called linear regressors. You specify linear regressors using the `linearRegressor` object. You can also specify linear regressors by using linear ARX model orders as an input argument. For more information, see "Nonlinear ARX Model Orders and Delay". However, this second approach constrains your regressor set to linear regressors with consecutive delays. To create polynomial regressors, use the `polynomialRegressor` object. You can also specify custom regressors, which are nonlinear functions of delayed inputs and outputs. For example, $u(t–1)y(t–3)$ is a custom regressor that multiplies instances of input and output together. Specify custom regressors using the `customRegressor` object.

You can assign any of the regressors as inputs to the linear function block of the output function, the nonlinear function block, or both.

2   It maps the regressors to the model output using an output function block. The output function block can include linear and nonlinear blocks in parallel. For example, consider the following equation:

$$F(x) = L^T(x - r) + g(Q(x - r)) + d$$

Here, x is a vector of the regressors, and r is the mean of x. $F(x) = L^T(x - r) + y_0$ is the output of the linear function block. $g(Q(x - r)) + y_0$ represents the output of the nonlinear function block. Q is a projection matrix that makes the calculations well-conditioned. d is a scalar offset that is added to the combined outputs of the linear and nonlinear blocks. The exact form of $F(x)$ depends on your choice of output function. You can select from the available mapping objects, such as

tree-partition networks, wavelet networks, and multilayer neural networks. You can also exclude either the linear or the nonlinear function block from the output function.

When estimating a nonlinear ARX model, the software computes the model parameter values, such as *L*, *r*, *d*, *Q*, and other parameters specifying *g*.

The resulting nonlinear ARX models are `idnlarx` objects that store all model data, including model regressors and parameters of the output function. For more information about these objects, see "Nonlinear Model Structures".

What Is a Hammerstein-Wiener Model?

This block diagram represents the structure of a Hammerstein-Wiener model:

Where,

- *f* is a nonlinear function that transforms input data *u*(*t*) as *w*(*t*) = *f*(*u*(*t*)).

  w(*t*), an internal variable, is the output of the Input Nonlinearity block and has the same dimension as *u*(*t*).
- *B/F* is a linear transfer function that transforms *w*(*t*) as *x*(*t*) = (*B/F*)*w*(*t*).

  x(*t*), an internal variable, is the output of the Linear block and has the same dimension as *y*(*t*).

  B and *F* are similar to polynomials in a linear Output-Error model. For more information about Output-Error models, see "What Are Polynomial Models?".

  For *ny* outputs and *nu* inputs, the linear block is a transfer function matrix containing entries:

  $$\frac{B_{j,\,i}(q)}{F_{j,\,i}(q)}$$

  where $j = 1,2,\ldots,\text{ny}$ and $i = 1,2,\ldots,\text{nu}$.
- *h* is a nonlinear function that maps the output of the linear block *x*(*t*) to the system output *y*(*t*) as *y*(*t*) = *h*(*x*(*t*)).

Because *f* acts on the input port of the linear block, this function is called the *input nonlinearity*. Similarly, because *h* acts on the output port of the linear block, this function is called the *output nonlinearity*. If your system contains several inputs and outputs, you must define the functions *f* and *h* for each input and output signal. You do not have to include both the input and the output nonlinearity in the model structure. When a model contains only the input nonlinearity *f*, it is called a Hammerstein model. Similarly, when the model contains only the output nonlinearity *h*, it is called a *Wiener* model.

The software computes the Hammerstein-Wiener model output *y* in three stages:

1   Compute *w*(*t*) = *f*(*u*(*t*)) from the input data.

   w(*t*) is an input to the linear transfer function *B/F*.

   The input nonlinearity is a static (*memoryless*) function, where the value of the output a given time *t* depends only on the input value at time *t*.

You can configure the input nonlinearity as a sigmoid network, wavelet network, saturation, dead zone, piecewise linear function, one-dimensional polynomial, or a custom network. You can also remove the input nonlinearity.

2 Compute the output of the linear block using $w(t)$ and initial conditions: $x(t) = (B/F)w(t)$.

You can configure the linear block by specifying the orders of numerator B and denominator F.

3 Compute the model output by transforming the output of the linear block $x(t)$ using the nonlinear function h as $y(t) = h(x(t))$.

Similar to the input nonlinearity, the output nonlinearity is a static function. You can configure the output nonlinearity in the same way as the input nonlinearity. You can also remove the output nonlinearity, such that $y(t) = x(t)$.

Resulting models are `idnlhw` objects that store all model data, including model parameters and nonlinearity estimators. For more information about these objects, see "Nonlinear Model Structures".

Prepare Data

Load Data into the MATLAB Workspace

Load sample data in `twotankdata.mat` by typing the following command in the MATLAB Command Window:

```
load twotankdata
```

This command loads the following two variables into the MATLAB Workspace browser:

- `u` is the input data, which is the voltage applied to the pump that feeds the water into Tank 1 (in volts).
- `y` is the output data, which is the water height in Tank 2 (in meters).

Creating iddata Objects

System Identification Toolbox data objects encapsulate both data values and data properties into a single entity. You can use the System Identification Toolbox commands to conveniently manipulate these data objects as single entities.

You must have already loaded the sample data into the MATLAB workspace, as described in "Load Data into the MATLAB Workspace" on page 4-5.

Use the following commands to create two `iddata` data objects, `ze` and `zv`, where `ze` contains data for model estimation and `zv` contains data for model validation. `Ts` is the sample time.

```
Ts = 0.2; % Sample time is 0.2 sec
z = iddata(y,u,Ts);
% First 1000 samples used for estimation
ze = z(1:1000);
% Remaining samples used for validation
zv = z(1001:3000);
```

To view the properties of the `iddata` object, use the `get` command. For example:

```
get(ze)
```

MATLAB software returns the following data properties and values:

```
             Domain: 'Time'
               Name: ''
         OutputData: [1000x1 double]
                  y: 'Same as OutputData'
         OutputName: {'y1'}
         OutputUnit: {''}
          InputData: [1000x1 double]
                  u: 'Same as InputData'
          InputName: {'u1'}
          InputUnit: {''}
             Period: Inf
        InterSample: 'zoh'
                 Ts: 0.2000
             Tstart: 0.2000
   SamplingInstants: [1000x0 double]
           TimeUnit: 'seconds'
     ExperimentName: 'Exp1'
              Notes: {}
           UserData: []
```

To modify data properties, use dot notation. For example, to assign channel names and units that label plot axes, type the following syntax in the MATLAB Command Window:

```
% Set time units to minutes
ze.TimeUnit = 'sec';
% Set names of input channels
ze.InputName = 'Voltage';
% Set units for input variables
ze.InputUnit = 'V';
% Set name of output channel
ze.OutputName = 'Height';
% Set unit of output channel
ze.OutputUnit = 'm';

% Set validation data properties
zv.TimeUnit = 'sec';
zv.InputName = 'Voltage';
zv.InputUnit = 'V';
zv.OutputName = 'Height';
zv.OutputUnit = 'm';
```

To verify that the `InputName` property of `ze` is changed, type the following command:

```
ze.inputname
```

Tip Property names, such as `InputName`, are not case sensitive. You can also abbreviate property names that start with `Input` or `Output` by substituting u for `Input` and y for `Output` in the property name. For example, `OutputUnit` is equivalent to `yunit`.

Start the System Identification App

To open the **System Identification** app, type the following command in the MATLAB Command Window:

```
systemIdentification
```

The default session name, `Untitled`, appears in the title bar.
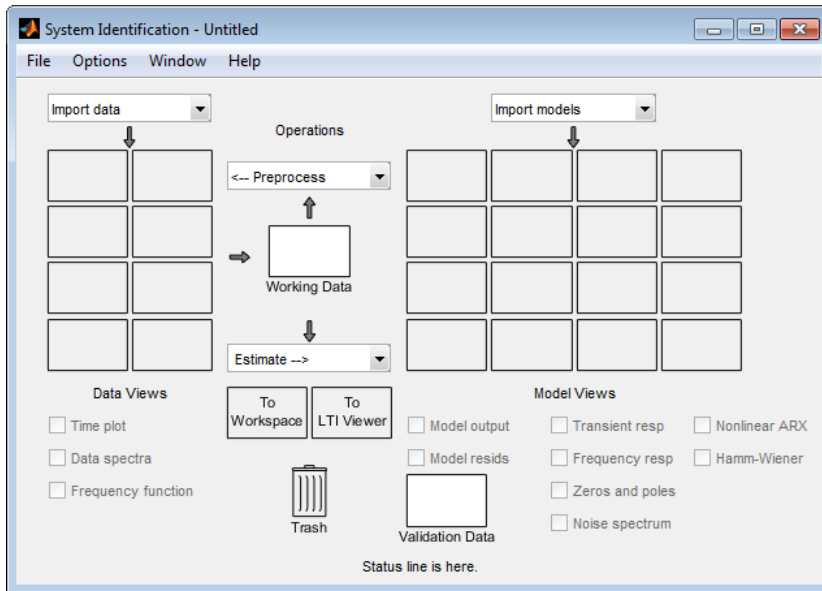
Import Data Objects into the System Identification App

You can import the data objects into the app from the MATLAB workspace.

You must have already created the data objects, as described in "Creating iddata Objects" on page 4-5, and opened the app, as described in "Start the System Identification App" on page 4-6.

To import data objects:

1   In the **System Identification** app, select **Import data > Data object**.

This action opens the Import Data dialog box.
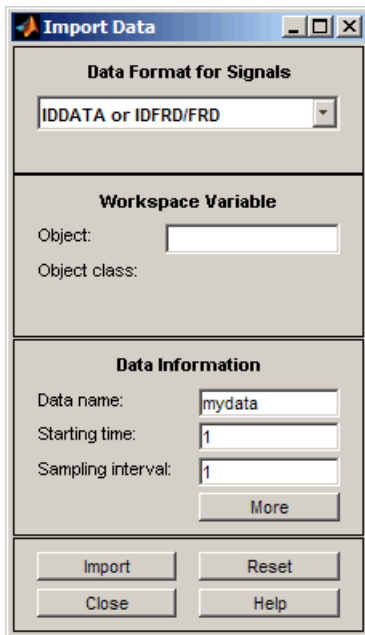
2 Enter ze in the **Object** field to import the estimation data. Press **Enter**.

This action enters the object information into the Import Data fields.

Click **More** to view additional information about this data, including channel names and units.

3 Click **Import** to add the icon named ze to the **System Identification** app.

4 In the Import Data dialog box, type zv in the **Object** field to import the validation data. Press **Enter**.

5 Click **Import** to add the icon named zv to the **System Identification** app.

6 In the Import Data dialog box, click **Close**.

7 In the **System Identification** app, drag the validation data **zv** icon to the **Validation Data** rectangle. The estimation data **ze** icon is already designated in the **Working Data** rectangle.

Alternatively, right-click the zv icon to open the Data/model Info dialog box. Select the **Use as Validation Data** check-box. Click **Apply** and then **Close** to add zv to the **Validation Data** rectangle.

The **System Identification** app now resembles the following figure.
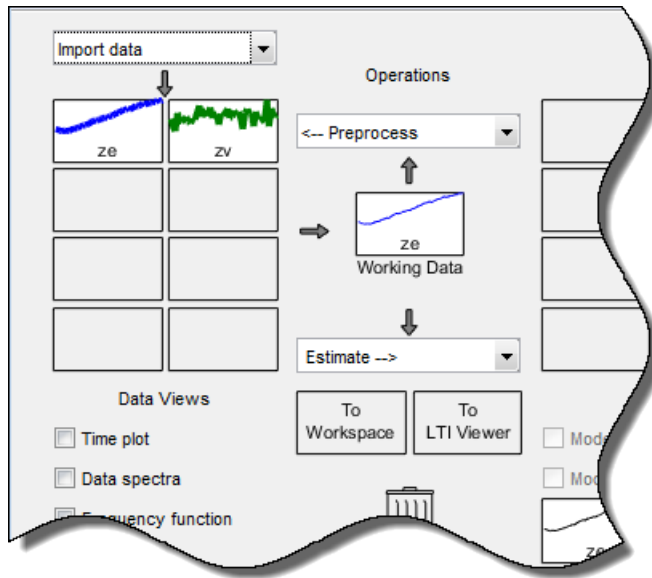
Estimate Nonlinear ARX Models

Estimate Nonlinear ARX Model with Default Settings

In this portion of the tutorial, you estimate a nonlinear ARX model using default model structure and estimation options.

You must have already prepared the data, as described in "Prepare Data" on page 4-5. For more information about nonlinear ARX models, see "What Is a Nonlinear ARX Model?" on page 4-3.

Note The illustrations in this tutorial represent a typical **System Identification** app session. Your results may not precisely match the results in the images.

---

1   In the **System Identification** app, select **Estimate > Nonlinear models**.
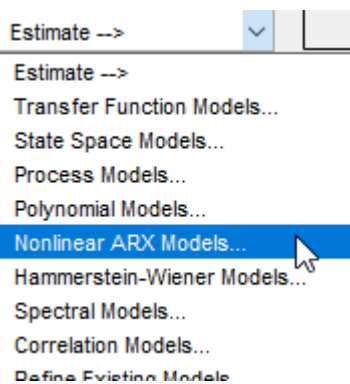
This action opens the **Estimate Nonlinear ARX Models** dialog box.

The dialog box contains two tabs — **Model Structure** and **Estimation Options**.

The **Model Structure** tab contains parameters that configure the structure of the model, such as the choice of the regressors and the type of the output function. Specify the name of the model to be estimated in **Model name**. The default model name is `nlarx1`.

Use the **Regressors** tab to create the regressor sets to use. By default, the app creates a linear regressor set with lags `[1 2]` in each variable and stores the regressor as `Linear regressors #1` in the **Regressor Sets** area. This panel contains a table that allows you to modify the lags to use for each variable.
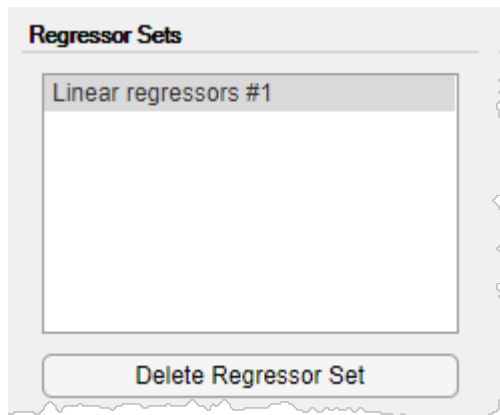
The configuration panel **Configure: Linear regressors #1** contains a table that shows the lag values that the regressor set uses. You can modify these lags.
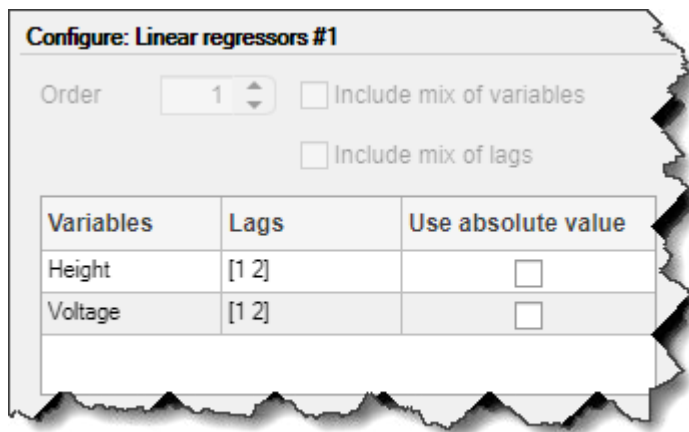
The regressor assignment table **Assign Regressors to Output Function** lists the regressor names and the regressor assignments to the output function components.

The regressor assignments in the table produce a model for the output `Height` of:

Height(t) = *f*(*Height*(t-1), *Height*(t-2),*Voltage*(t-1),*Voltage*(t-2))

Here, *f*(.) is a static nonlinear function that maps the regressors to the output. To choose *f*(.) and its associated parameters, use the **Output Function** tab.

You can add more regressor sets, such as polynomials and custom formulas, by selecting a regressor type in **Add regressors of type**.
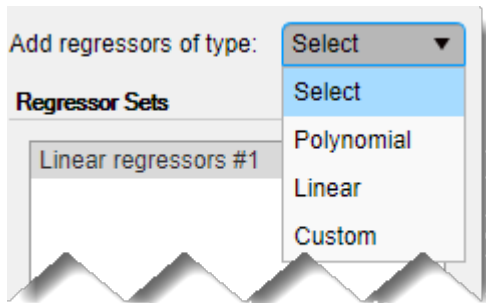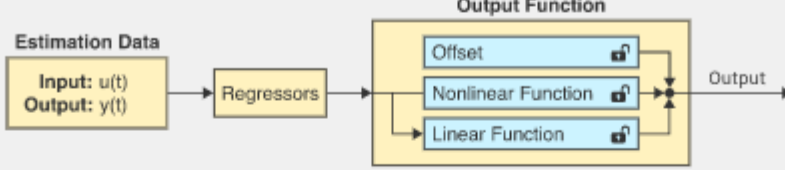
This example uses only `Linear regressors #1`.

2   Select the **Output Function** tab.

The tab shows that the app selects **Wavelet Network** as the default type of the output function *f*(.). This function uses a sum of a linear function, a nonlinear function (which is a sum of scaled wavelets), and an offset term to compute the output. The tab also shows a set of configuration parameters. The number of wavelets used by the function is configured to be chosen automatically during estimation. You can modify this selection. You can also modify the initial values of the linear coefficients and the offset. For the first estimate, keep the default choices.
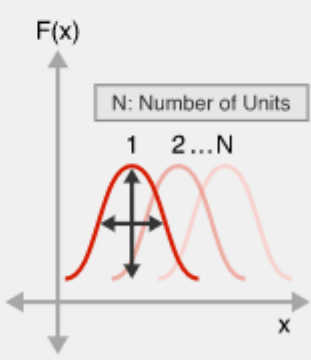
3   Click **Estimate**.

This action adds the model `nlarx1` to the model board of the **System Identification** app, as shown in the following figure.
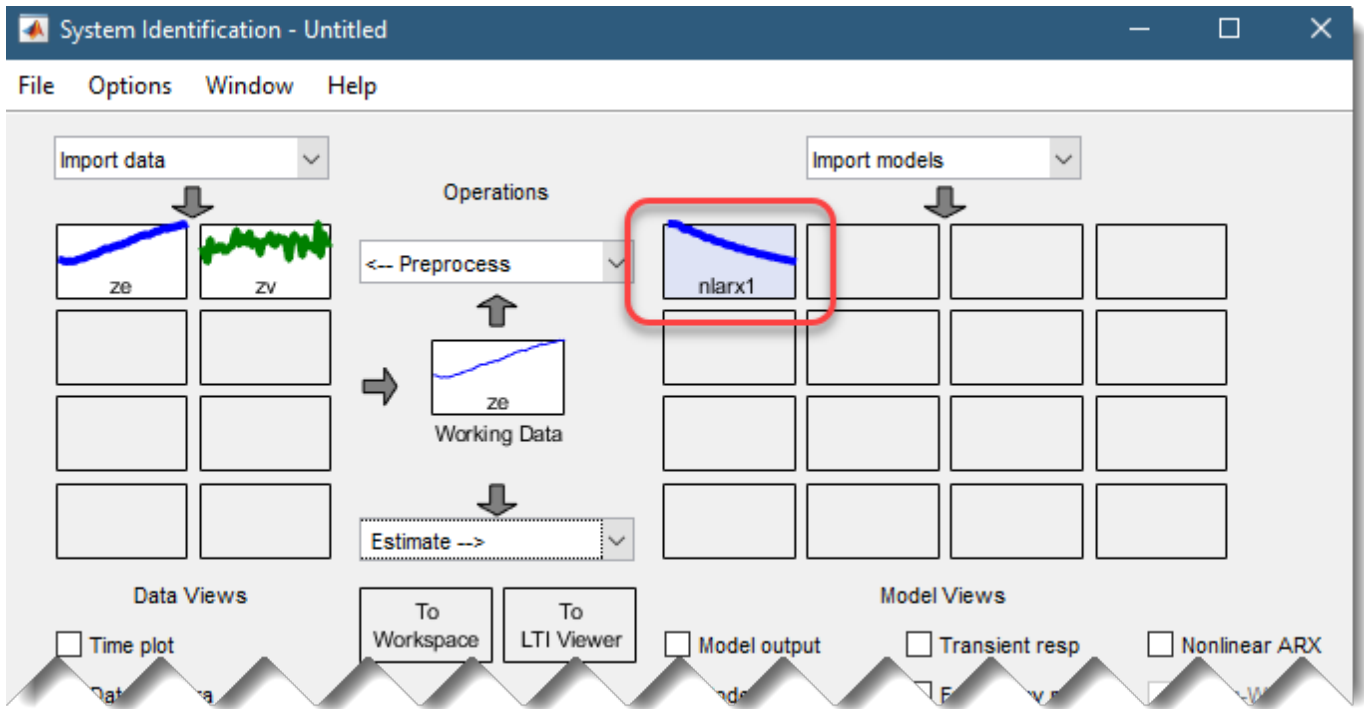
4   Double click or right-click the model icon `nlarx1` The model board shows information about the estimated model. The top area describes the model structure and estimation results (data used, percent fit and other quality metrics). The bottom area (**Diary and Notes**) shows the MATLAB script that you can use to reproduce the estimation results at the command line.

Data/model Info: nlarx1

Model name: | nlarx1

Color: | [0,0,1]

```
Output function: Wavelet Network with 7 units

Name: nlarx1
Sample time: 0.2 seconds

Status:
Estimated using NLARX on time domain data "ze".
Fit to estimation data: 96.62% (prediction focus)
FPE: 3.743e-05, MSE: 3.647e-05
```

Diary and Notes

```
% Create Output Function
NL = wavenet;
% Create a template IDNLARX model
initialModel = idnlarx(ze.OutputName, ze.InputName, Regressors, NL);

Options = nlarxOptions;

% Estimate a Nonlinear ARX Model
nlarx1 = nlarx(ze, initialModel, Options);
```

Show in LTI Viewer

Present | Export | Close | Help

Note **Fit (%)** is computed using the estimation data set with prediction focus. and not the validation data set. However, the model output plot in the next step compares the fit to the validation data set. To see how well the model can simulate the response of an independent validation dataset, use the **Model Output** plot, as discussed in the following step.

---

5   In the **Model Views** area of the **System Identification** app, select **Model output**. This action simulates the model using the input validation data as the input to the model and plots the simulated output on top of the output validation data.
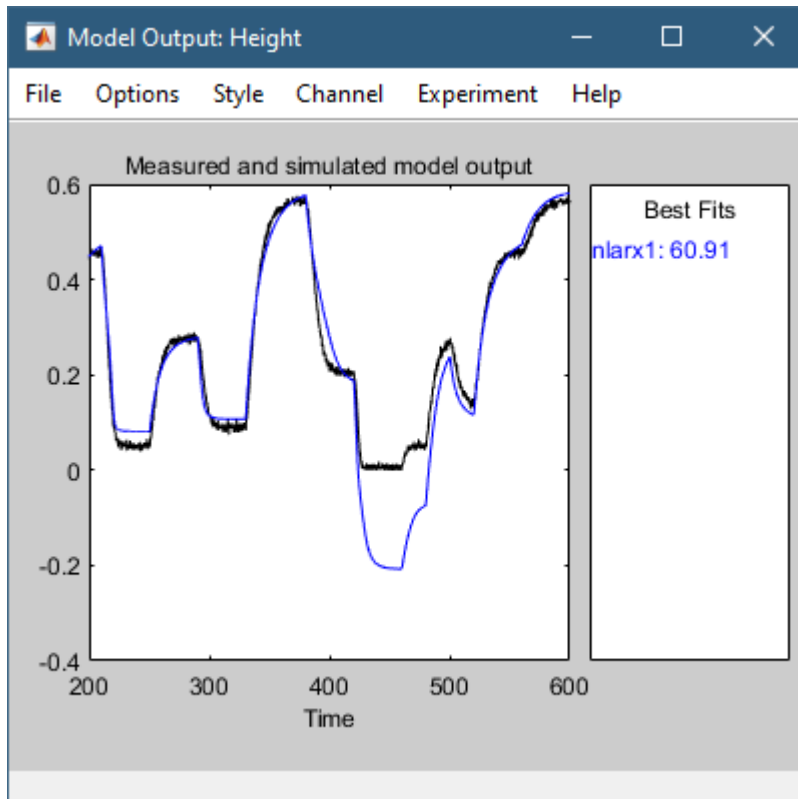
The **Best Fits** area of the model output plot shows the agreement between the model output and the validation-data output using a normalized root mean squared (NRMSE) metric of goodness of fit. The agreement is poor in the 400-500 time span.

Plot Nonlinearity Cross-Sections for Nonlinear ARX Models

Perform the following procedure to view the shape of the nonlinearity as a function of regressors on a Nonlinear ARX Model plot.

1 In the **System Identification** app, select the **Nonlinear ARX** check box to view the nonlinearity cross-sections.

By default, the plot shows the relationship between the output regressors `Height(t-1)` and `Height(t-2)`. This plot shows a regular plane in the following figure. Thus, the relationship between the regressors and the output is approximately a linear plane.
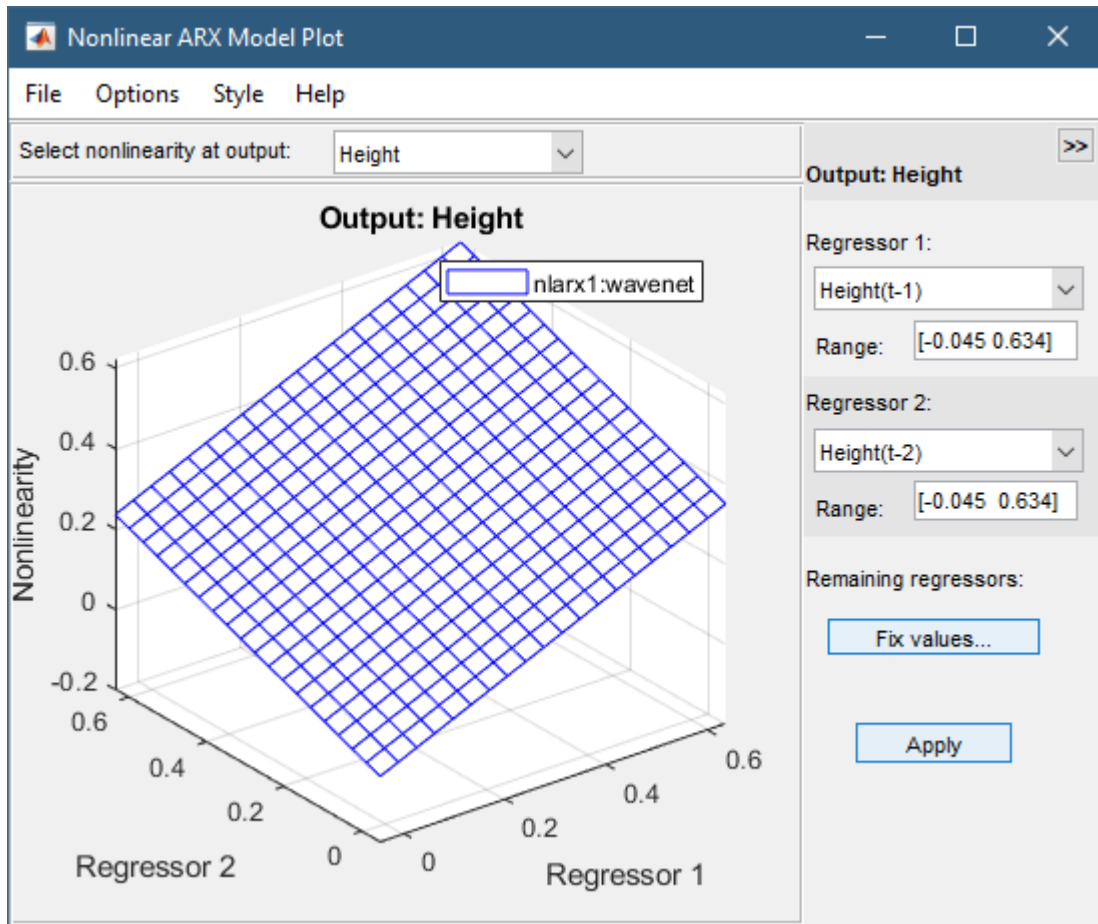
2 In the Nonlinear ARX Model Plot window, set **Regressor 1** to `Voltage(t-1)`. Set **Regressor 2** to `Voltage(t-2)`. Click **Apply**.

The relationship between these regressors and the output is nonlinear, as shown in the following plot.
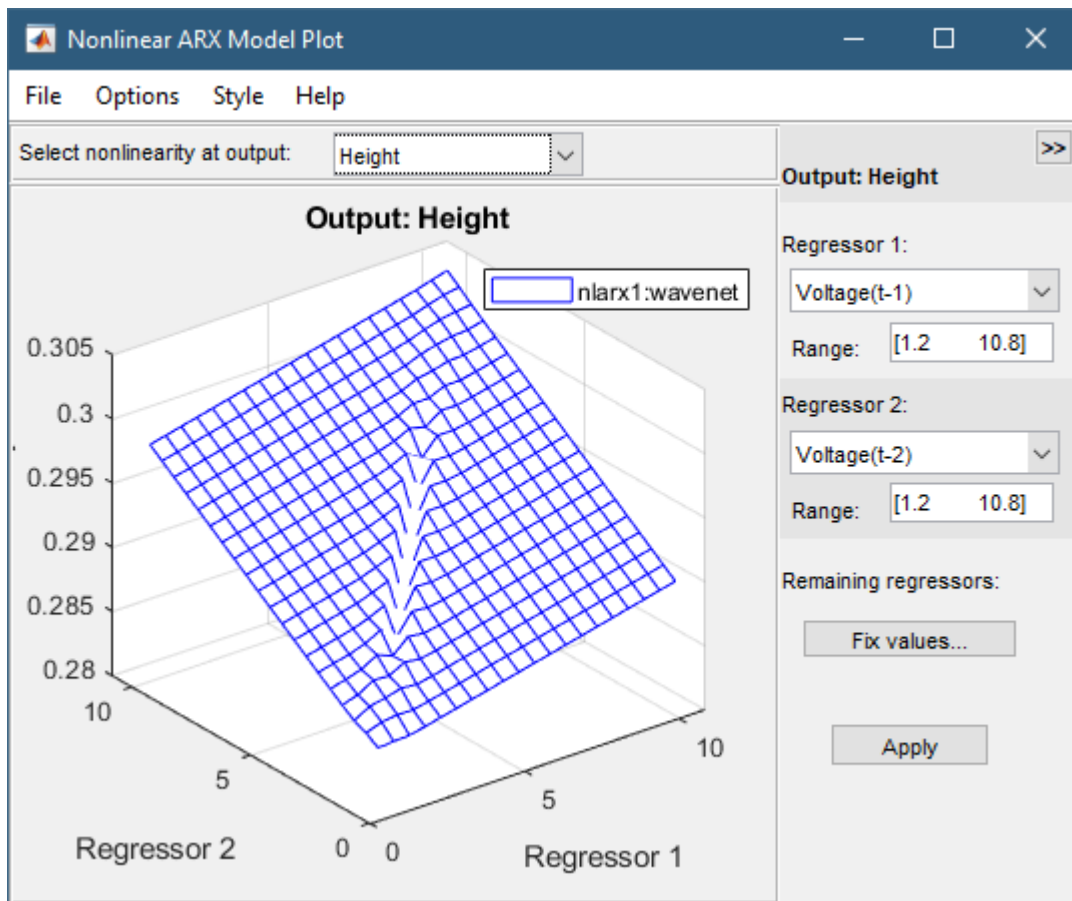
3   To rotate the nonlinearity surface, select **Style** > **Rotate 3D** and drag the plot to a new orientation.

4   To display a 1-D cross-section for Regressor 1, set Regressor 2 to `none`, and click **Apply**. The following figure shows the resulting nonlinearity magnitude for Regressor 1, which represents the time-shifted voltage signal, `Voltage(t-1)`.
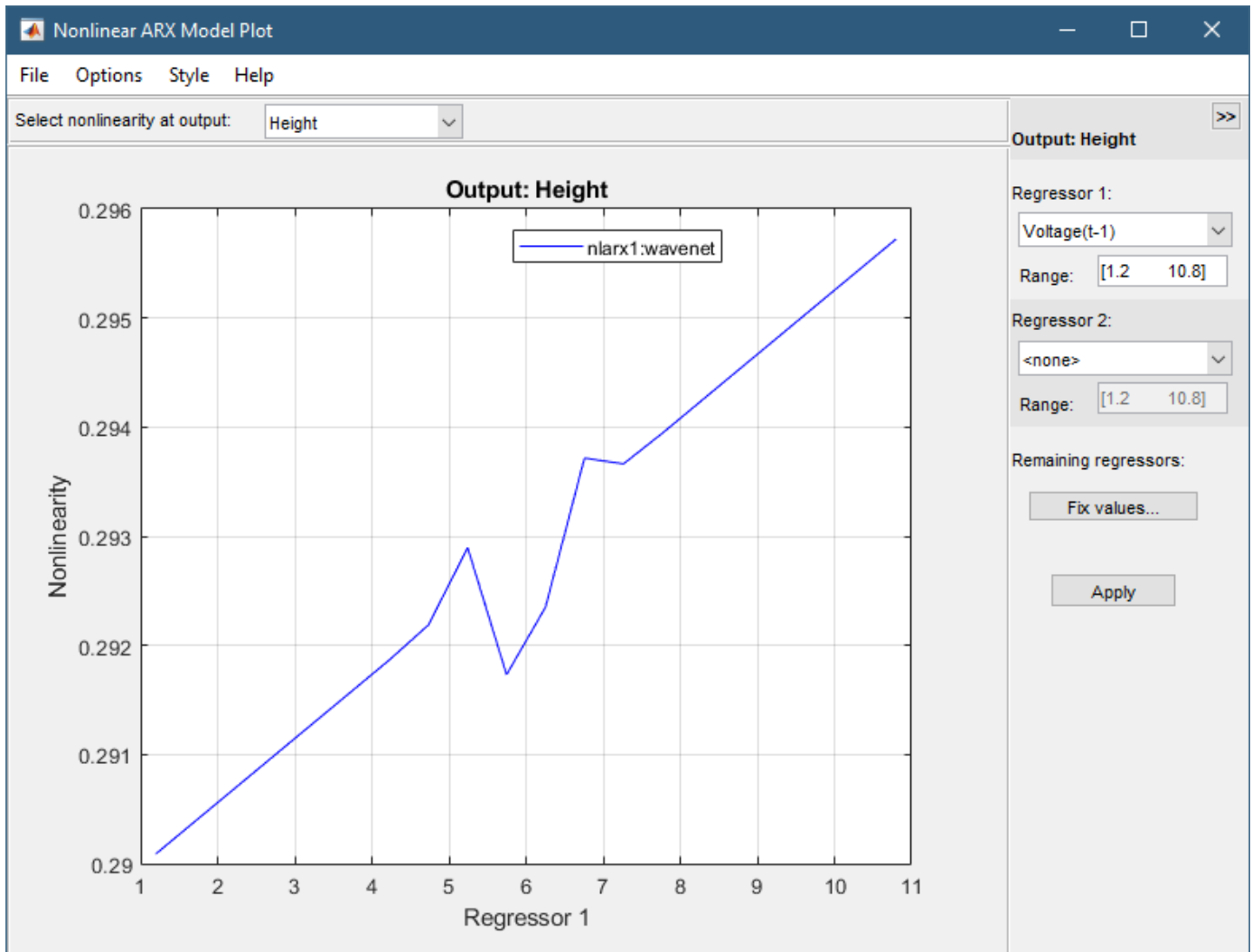
Change Nonlinear ARX Model Structure

In this portion of the tutorial, you estimate a nonlinear ARX model with specific input delay and nonlinearity settings. Typically, you select model orders by trial and error until you get a model that produces an accurate fit to the data.

You must have already estimated the nonlinear ARX model with default settings, as described in "Estimate Nonlinear ARX Model with Default Settings" on page 4-9.

1 In the **Estimate Nonlinear ARX Models** dialog box, select the **Model Structure** tab, and then select the **Regressors** tab.

2 In the **Configure: Linear regressors #1** panel, change the `Voltage` variable lags to `[3 4]`. This is because the data suggests that there is a minimum of 3 sample lags between the input and the output.

   This action updates the regressors list to show `Voltage(t-3)` and `Voltage(t-4)`—terms with a minimum delay of three samples.
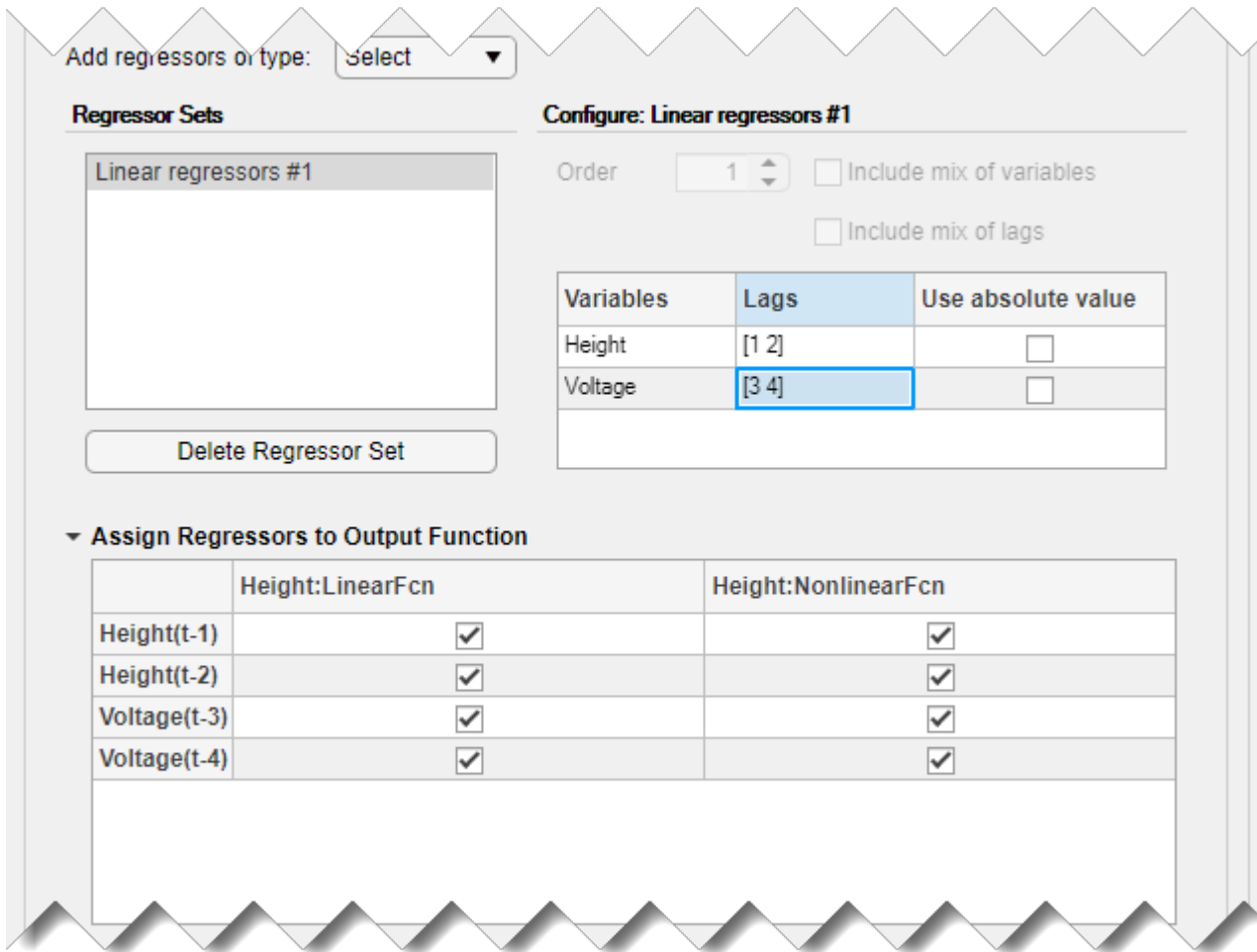
3 Click **Estimate**.

This action adds the model `nlarx2` to the **System Identification** app and updates the **Model Output** window to include this model. Double-clicking on the `nlarx2` icon in the model board of the main app window shows the estimation results.

Data/model Info: nlarx2   — □ ✕

Model name: nlarx2

Color: [0,0.5,0]

```
Nonlinear ARX model with 1 output and 1 input
  Inputs: Voltage
  Outputs: Height

Regressors:
  Linear regressors in variables Height, Voltage

Output function: Wavelet Network with 5 units
```

Diary and Notes

```
% Create Output Function
NL = wavenet;
% Create a template IDNLARX model
initialModel = idnlarx(ze.OutputName, ze.InputName, Regressors, NL);

Options = nlarxOptions;

% Estimate a Nonlinear ARX Model
nlarx2 = nlarx(ze, initialModel, Options);
```

Show in LTI Viewer

Present   Export   Close   Help

The app also updates the model output plot to display the fit of the model `nlarx2` to the validation dataset `zv`. The plot shows that the choice of correct input variable lags improves the fit.
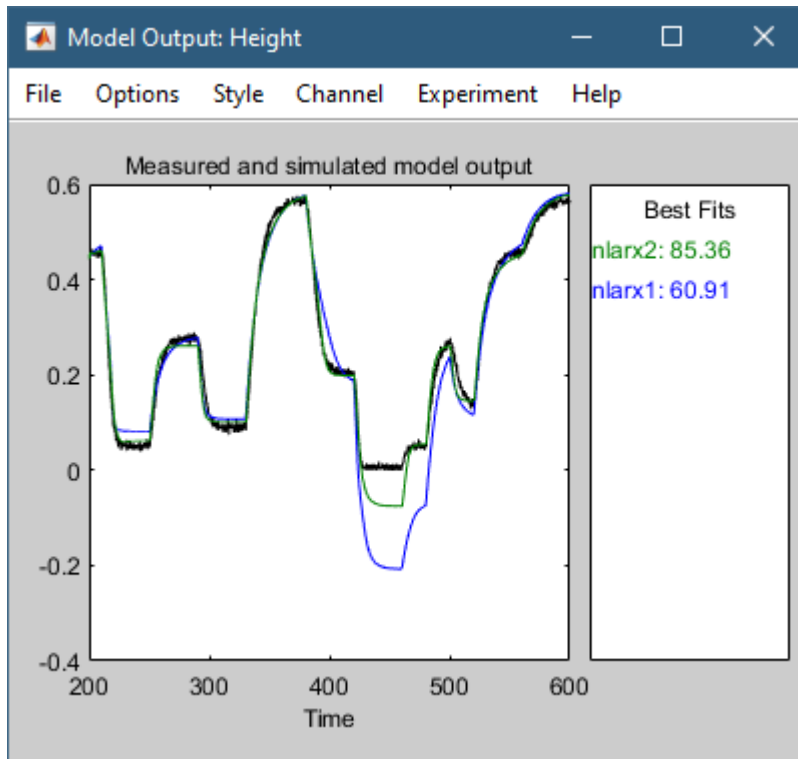
4.   In the **Estimate Nonlinear ARX Models** dialog box, select the **Model Structure** tab, and then select the **Regressors** tab.

5.   Select the **Output Function** tab.

6.   For the **Number of Units**, select **Enter** and enter 6. Setting a specific number controls the flexibility of the nonlinear function by directly specifying the number of wavelets that **Wavelet Network** uses.
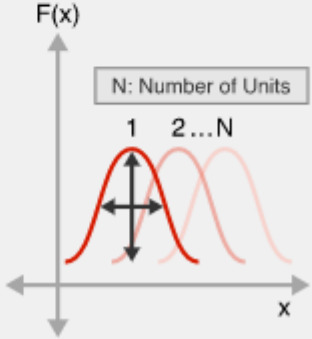
7 Click **Estimate**.

This action adds the model `nlarx3` to the **System Identification** app. It also updates the **Model Output** window, as shown in the following figure.

Measured and simulated model output

Best Fits

nlarx3: 86.28
nlarx2: 85.36
nlarx1: 60.91
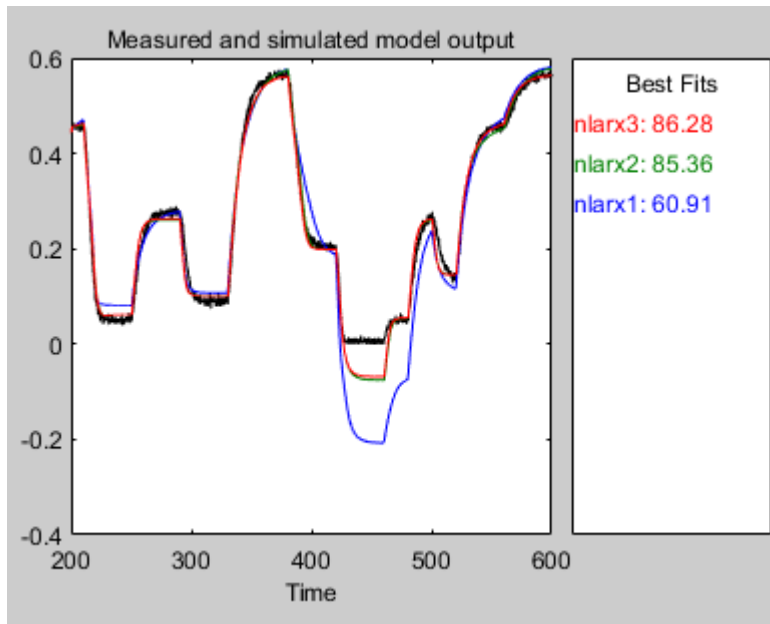
Use Polynomial Regressors

You can estimate a nonlinear ARX model that use higher powers of lagged variables as regressors. In this example, you will add second-order polynomials of the lagged variables, including cross terms.

1   In the **Estimate Nonlinear ARX Models** dialog, select the **Model Structure** tab and then select the **Regressors** tab.

2   Expand **Add regressors of type** and select **Polynomial**.
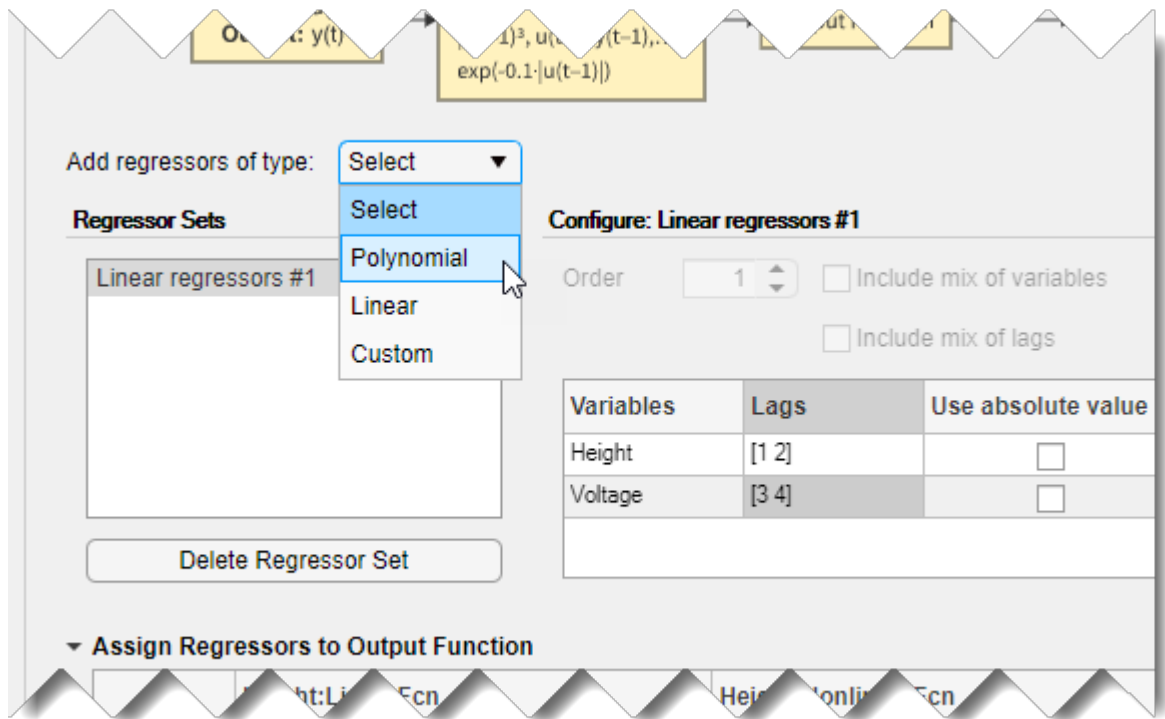
This action adds a polynomial regressor set called `Polynomial regressors #1` to the list of regressor sets. By default, this set is configured to generate regressors of order 2, as shown in the configuration panel.
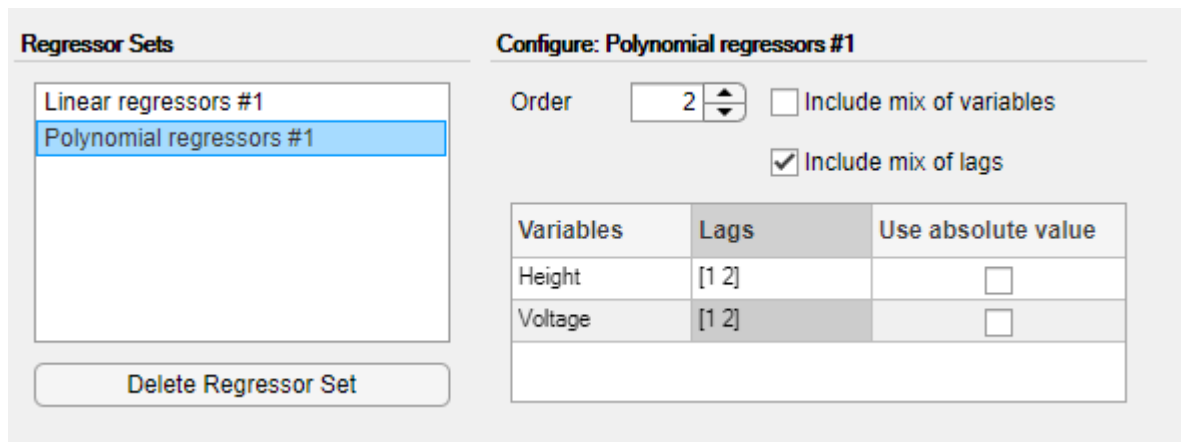
3   Change the `Voltage` lags to `[3 4]`.

4   Select **Include mix of variables** to include all cross terms. This action adds ten second-order regressors to the model, in addition to the four contributed by **Linear regressor #1**.

5   Click **Estimate**. This action adds the model `nlarx4` to the app. The app updates the **Model Output** plot, as shown in the following figure.
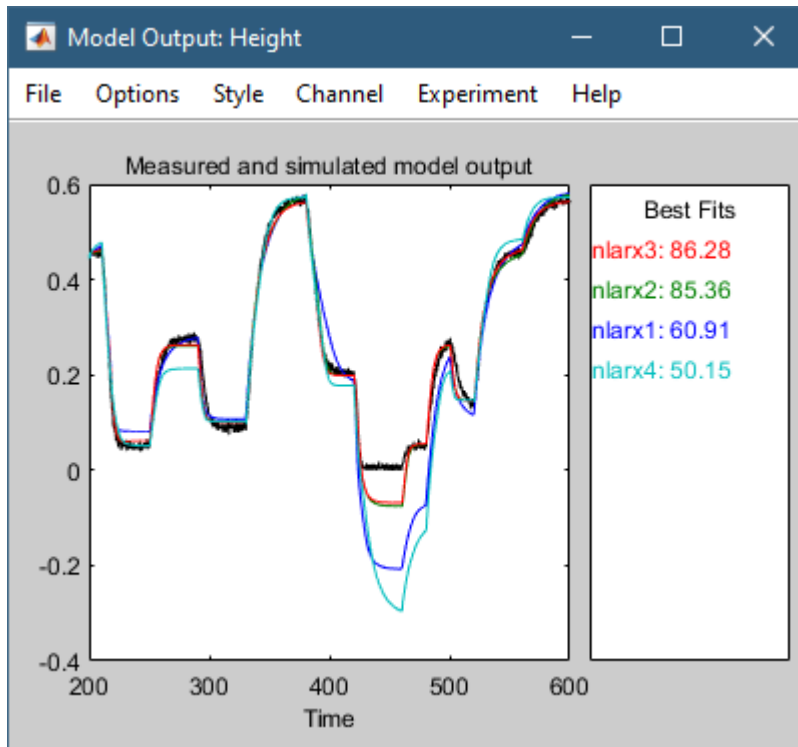
The plot shows that the addition of polynomial regressors has not improved the generalization ability of the model. That is, the model does not provide a better fit between the model output and the measured output for independent validation data. This result is typically the scenario with overfits, that is, when the model contains more regressors than absolutely required.

Select Subset of Regressors in Nonlinear Block

You can estimate a nonlinear ARX model that includes only a subset of regressors that enter as inputs to the nonlinear block. By default, all regressors are used in the nonlinear block. In this portion of the tutorial, you assign only a subset of the regressors to the nonlinear block.

You must have already specified the model structure, as described in "Change Nonlinear ARX Model Structure" on page 4-19.

1 In the **Estimate Nonlinear ARX Models** dialog box, select the **Model Structure** tab, and then select the **Regressors** tab.

2 If collapsed, expand the **Assign Regressors to Output Function** panel.

3 Clear the selections for all the second order regressors from the **Height:NonlinearFcn** column of the table. Also, clear the selections for all the regressors with cross terms (that is, regressors composed of products of 2 terms) from **Height:LinearFcn**
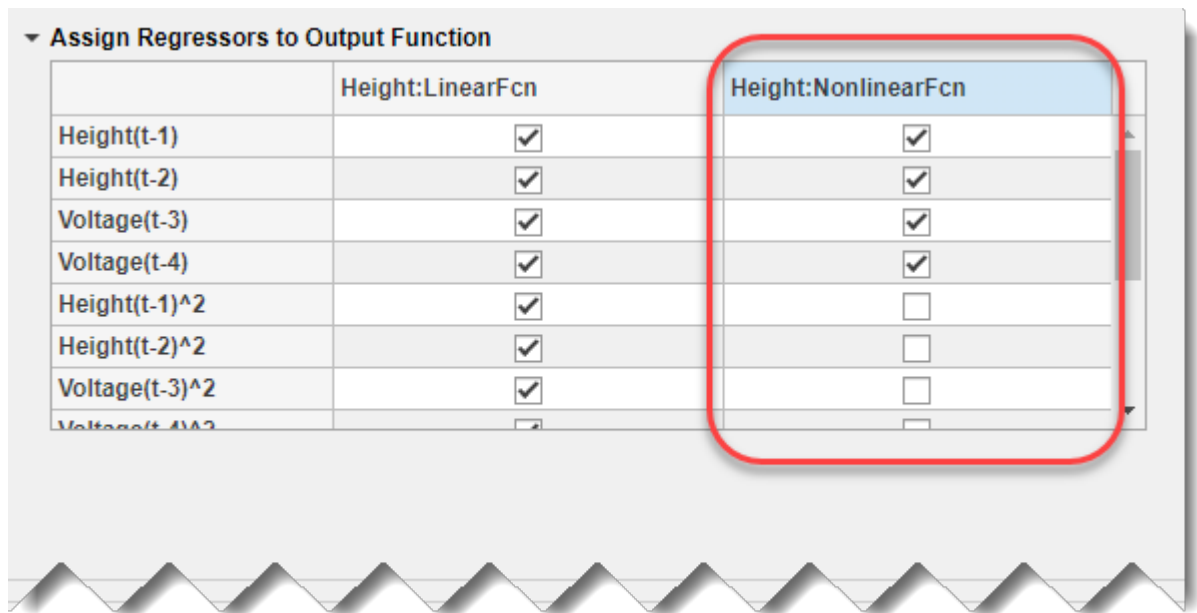
4 Click **Estimate**.

This action adds the model `nlarx5` to the **System Identification** app. It also updates the **Model Output** window.
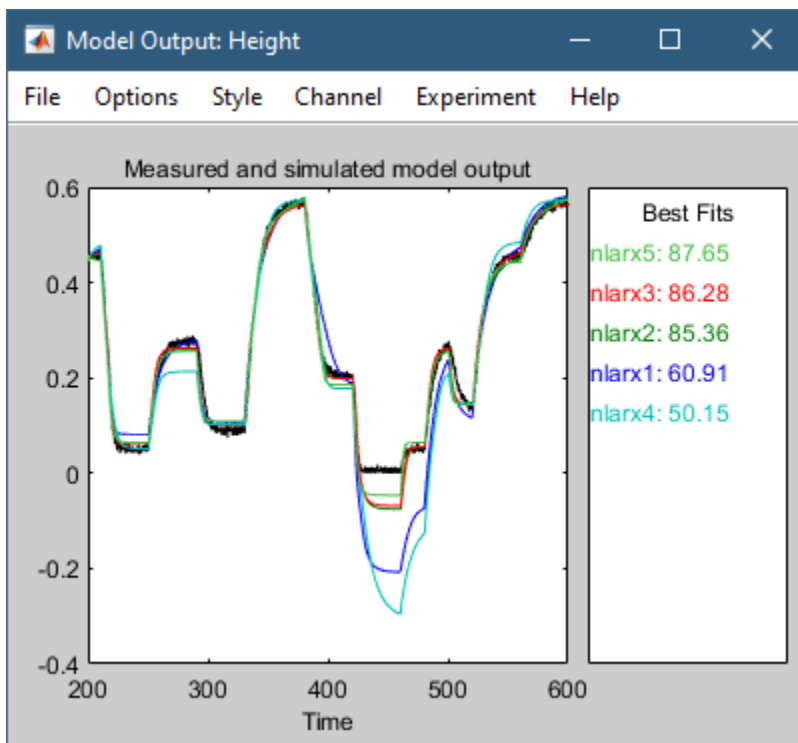
This model shows improved generalization ability.

Specify Previously Estimated Model with Different Output Function Nonlinearity

You can estimate a series of nonlinear ARX models by making systematic variations to the model structure and base each new model on the configuration of a previously estimated model. In this portion of the tutorial, you estimate a nonlinear ARX model that is similar to an existing model (`nlarx3`), but which has a different nonlinearity in the output function.

1 In the **Estimate Nonlinear ARX Models** dialog box, select the **Model Structure** tab.

2 Select `nlarx3` from **Initial Model**. This sets the properties that are displayed in the **Regressors** and **Output Function** tabs to the values corresponding to `nlarx3`.
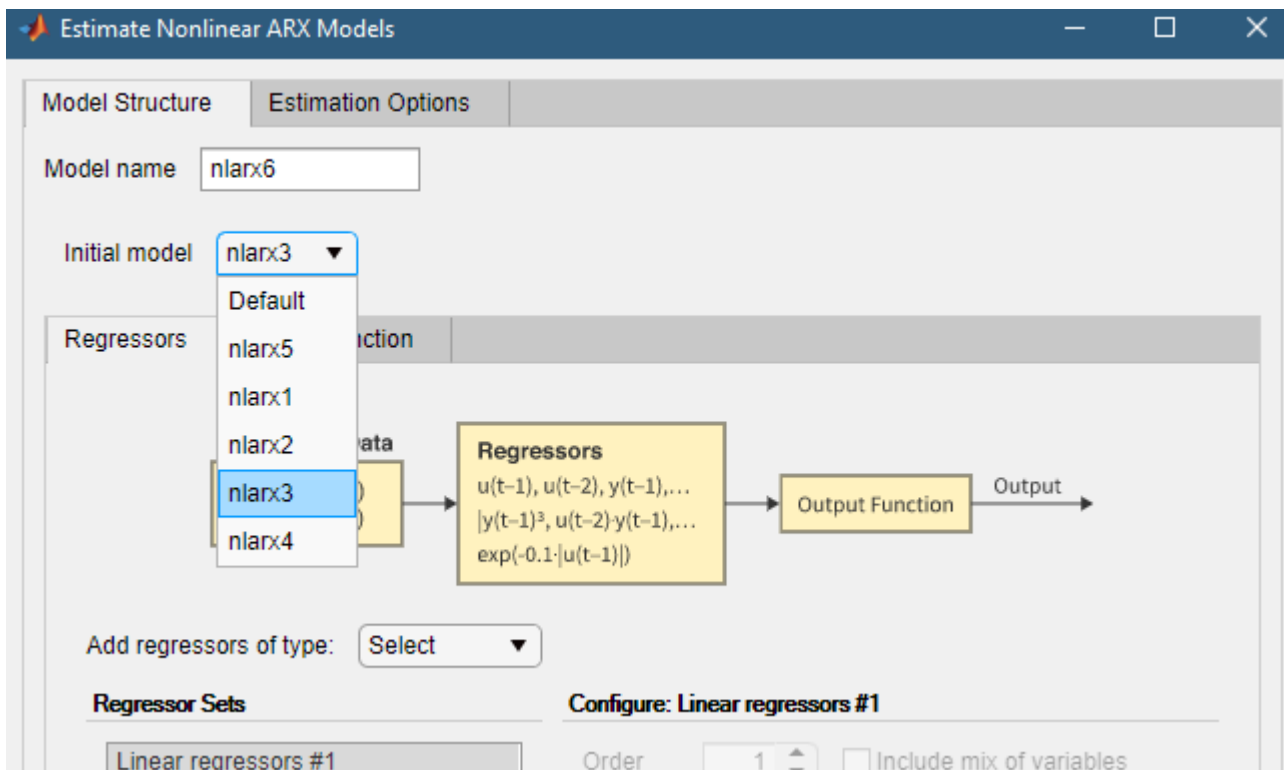
3 Select the **Output Function** tab.

4 In the table, change the choice of **Nonlinear Function** to `Sigmoid Network`. This sets the output function to a sigmoid network.

5

6 In **Enter the number of units**, enter 6.

7 Click **Estimate**.

This action adds the model `nlarx6` to the **System Identification** app. It also updates the **Model Output** plot, as shown in the following figure.
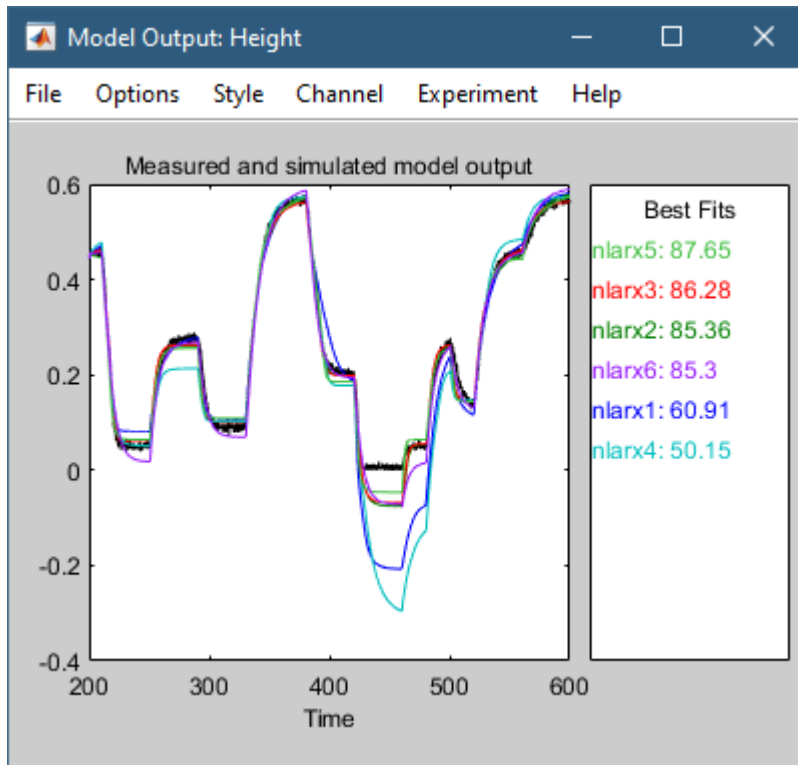
Select the Best Model

The best model is the simplest model that accurately describes the dynamics. The results indicate that model `nlarx3` is a candidate for being the best idnlarx model for this data, based on the various model structure choices that this example explored.

Export Best Model to Base Workspace

Drag the `nlarx3` icon to the **To Workspace** box in the main **System Identification** app window. Alternatively, double click the `nlarx3` icon to open its information board. From there, click **Export** to export the model to the base workspace.
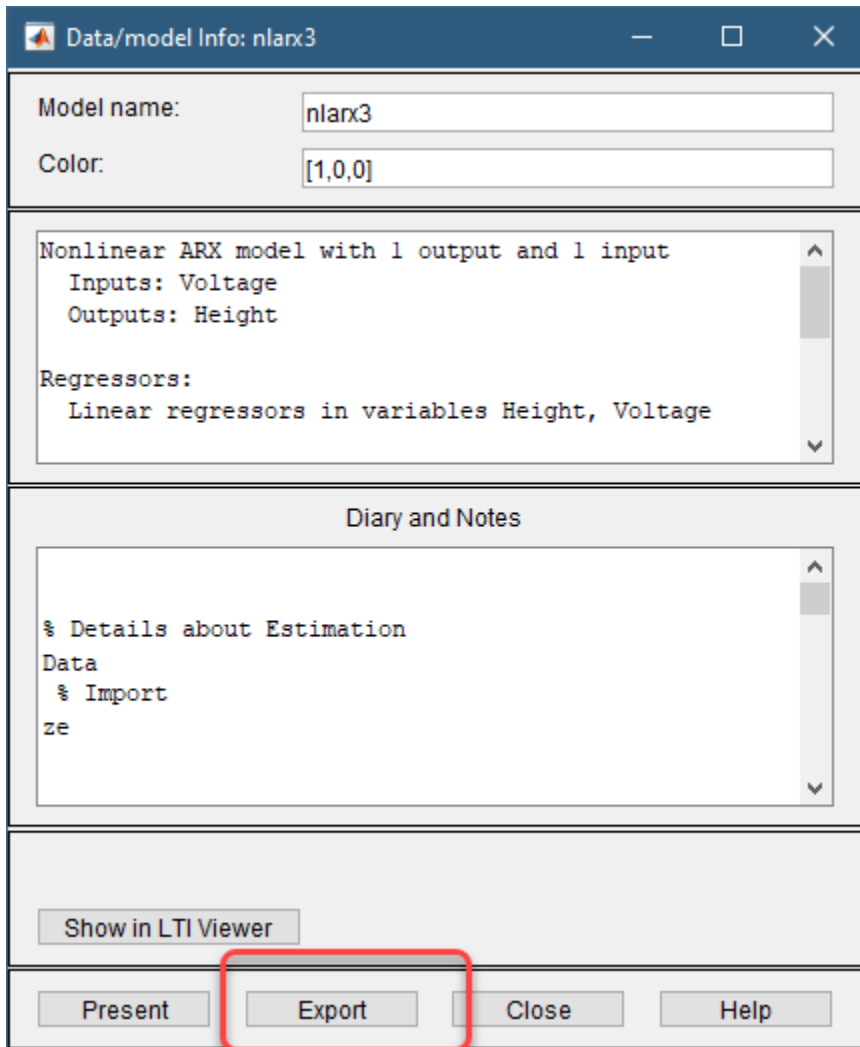
Estimate Hammerstein-Wiener Models

Estimate Hammerstein-Wiener Models with Default Settings

In this portion of the tutorial, you estimate nonlinear Hammerstein-Wiener models using default model structure and estimation options.

You must have already prepared the data, as described in "Prepare Data" on page 4-5. For more information about nonlinear ARX models, see "What Is a Hammerstein-Wiener Model?" on page 4-4

Note The illustrations in this tutorial represent a typical **System Identification** app session. Your results may not precisely match the results in the images.

1 In the **System Identification** app, select **Estimate > Hammerstein-Wiener Models** to open the **Estimate Hammerstein-Wiener Models** dialog box. The dialog box contains two tabs — **Model Structure** and **Estimation Options**.

2 Inspect the default model structure. The **Model Structure** tab contains three tabs, one tab for each component of the model. These components are the input nonlinearity, the linear dynamic block, and the output nonlinearity.

    a Select the **Input Nonlinearity** tab. The default nonlinearity is a **Piecewise Linear** function with 10 breakpoints.

    b Select the **Linear Block** tab. The linear transfer function has a numerator order of 2, which implies a second-order polynomial in z^{-1}, a denominator order of 3, and an input delay of 1 sample.

c Select the **Output Nonlinearity** tab. As with the **Input Nonlinearity** tab, the default nonlinearity is a **Piecewise Linear** function with 10 breakpoints.

3 Click **Estimate**.

This action estimates the parameters of the Hammerstein-Wiener model to minimize the difference between the model output and the measured output (`ze.OutputData`). The estimated parameters are:

a The breakpoint locations (x- and y-coordinates) of the input and output piecewise-linear functions.

b The numerator and the denominator polynomial coefficients that compose the linear block transfer function.

The software adds the estimated model `nlhw1` to the model board of the **System Identification** app.

4-33

4 In the **System Identification** app, select the **Model output** check box.

This action simulates the model using the input validation data as input to the model and plots the simulated output on top of the output validation data. Note that the validation data is set to zv.
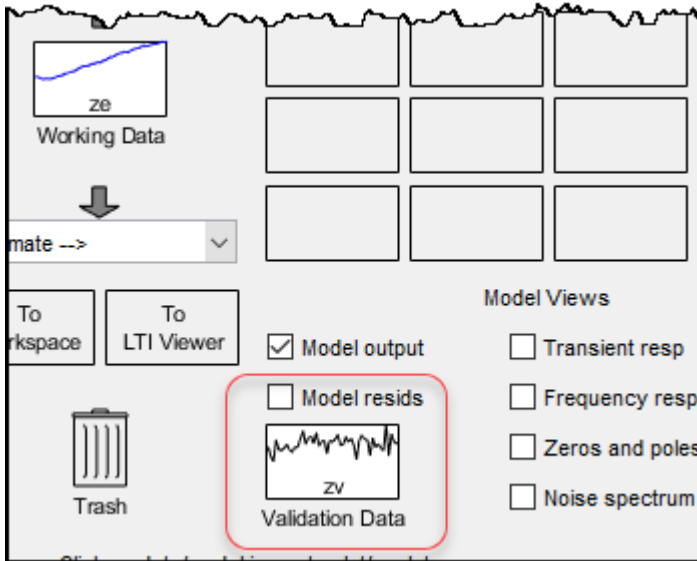
The **Best Fits** area of the Model Output window shows the agreement between the model output and the validation-data output.
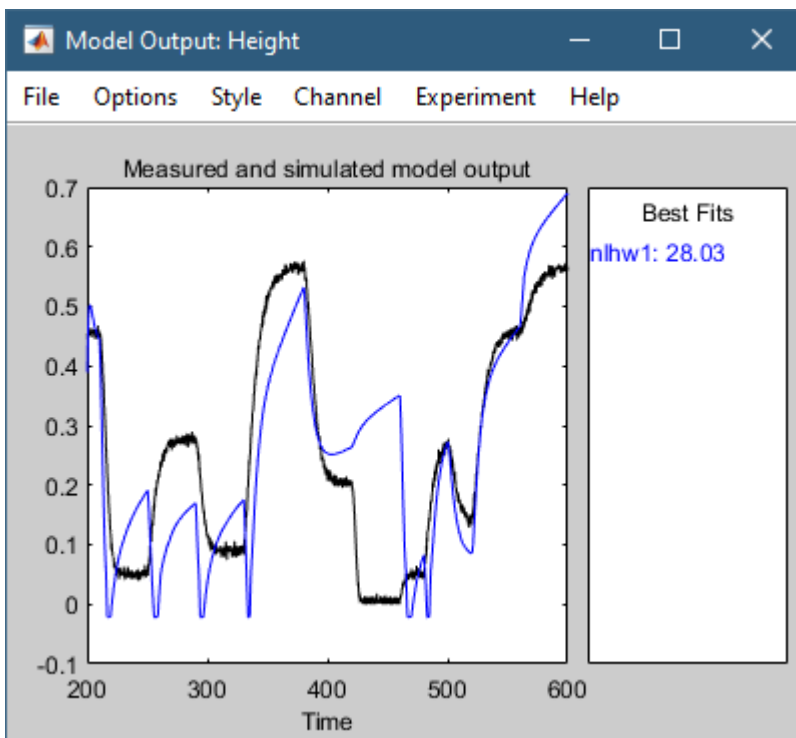
The model `nlhw1` does not validate well. The fit to `zv.OutputData` is poor.

Plot Nonlinearities and Linear Transfer Function

You can plot the input/output nonlinearities and the linear transfer function of the model on a Hammerstein-Wiener plot.

1   In the **System Identification** app, select **Hamm-Wiener** to view the Hammerstein-Wiener model plot.

The plot displays the input nonlinearity, as shown in the following figure.
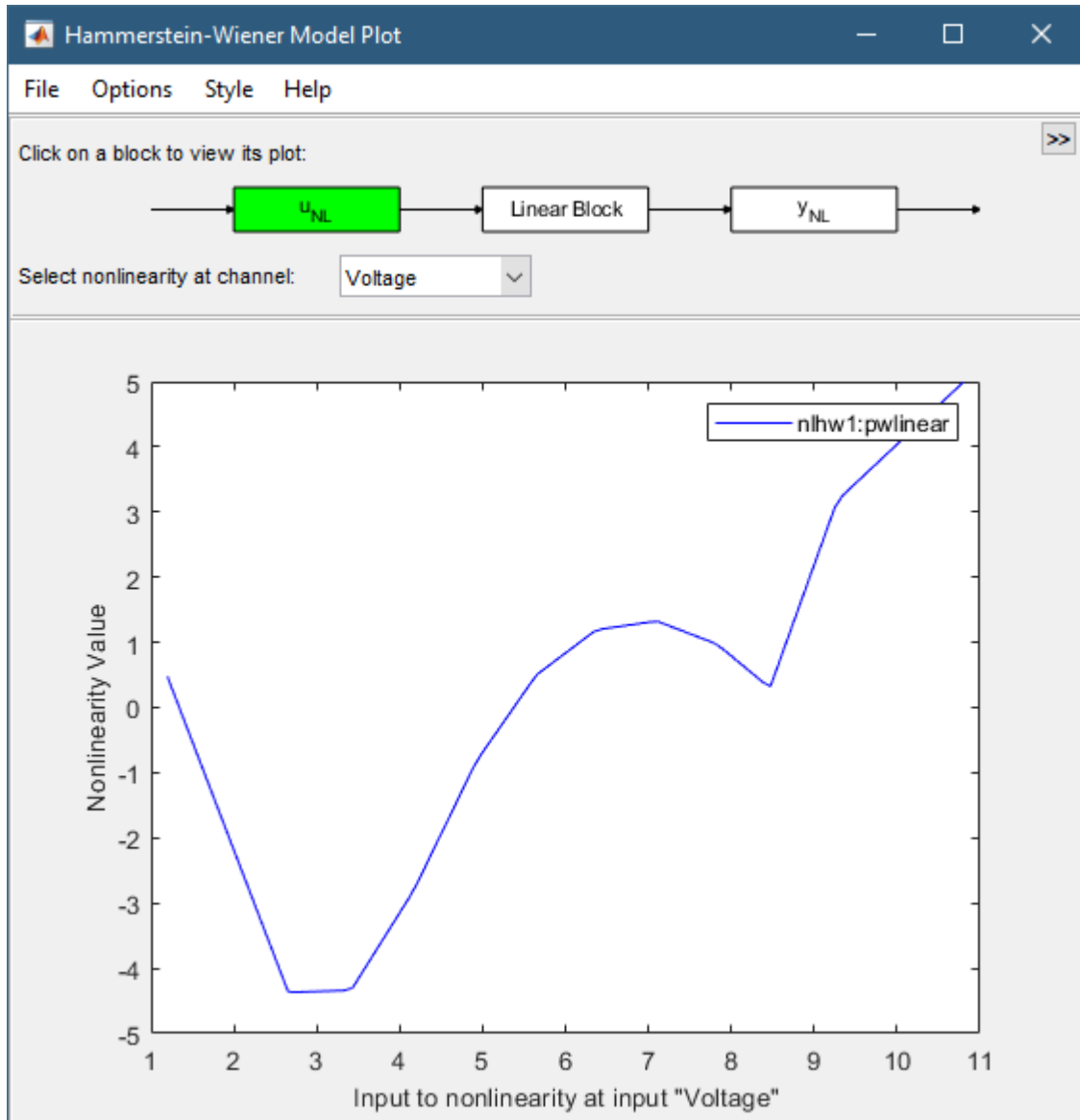
2   Click the $\mathbf{y_{NL}}$ rectangle in the top portion of the Hammerstein-Wiener Model Plot window.

The plot updates to display the output nonlinearity.
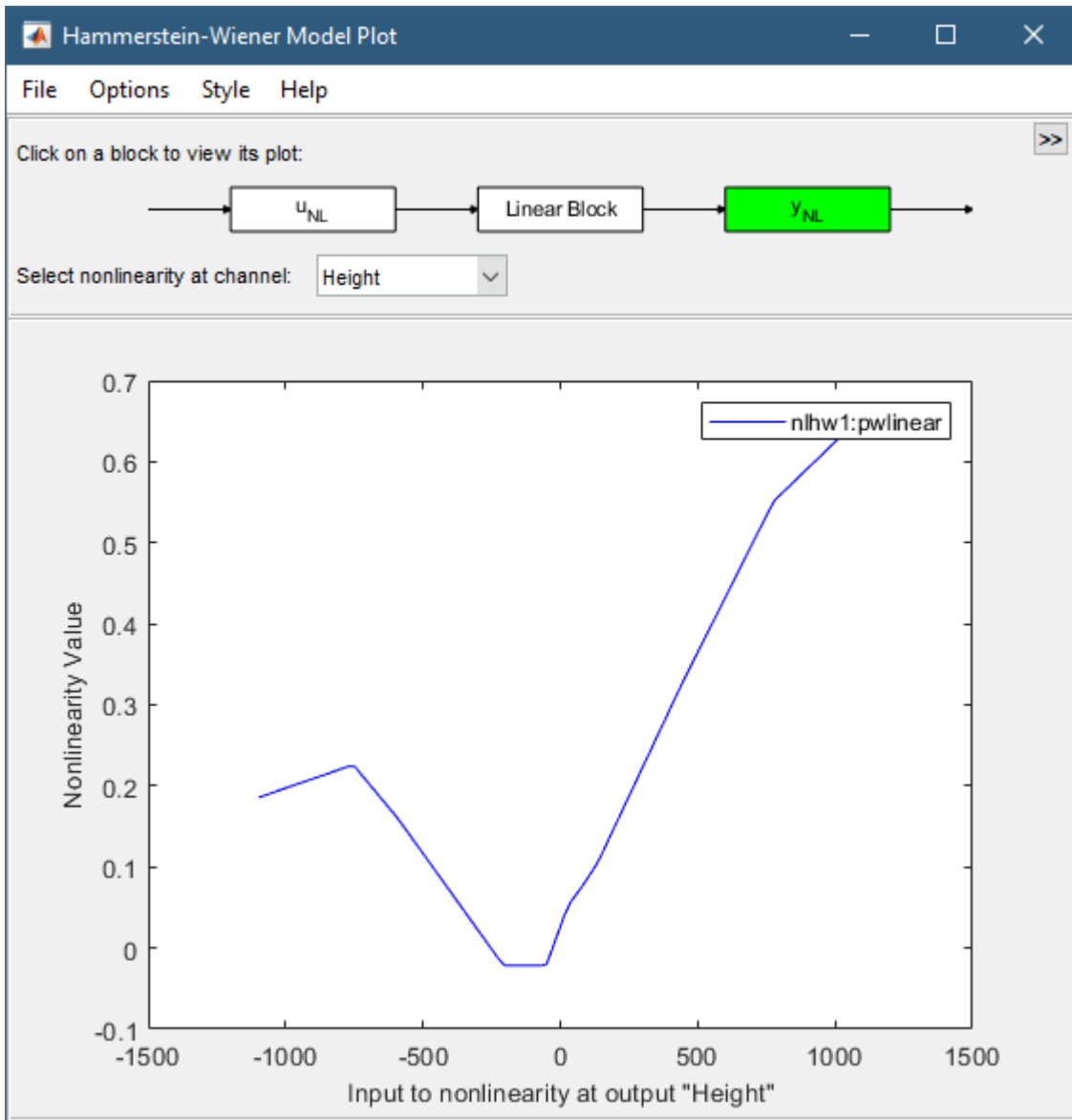
3 Click the **Linear Block** rectangle in the top portion of the Hammerstein-Wiener Model Plot window.

The plot updates to display the step response of the linear transfer function.
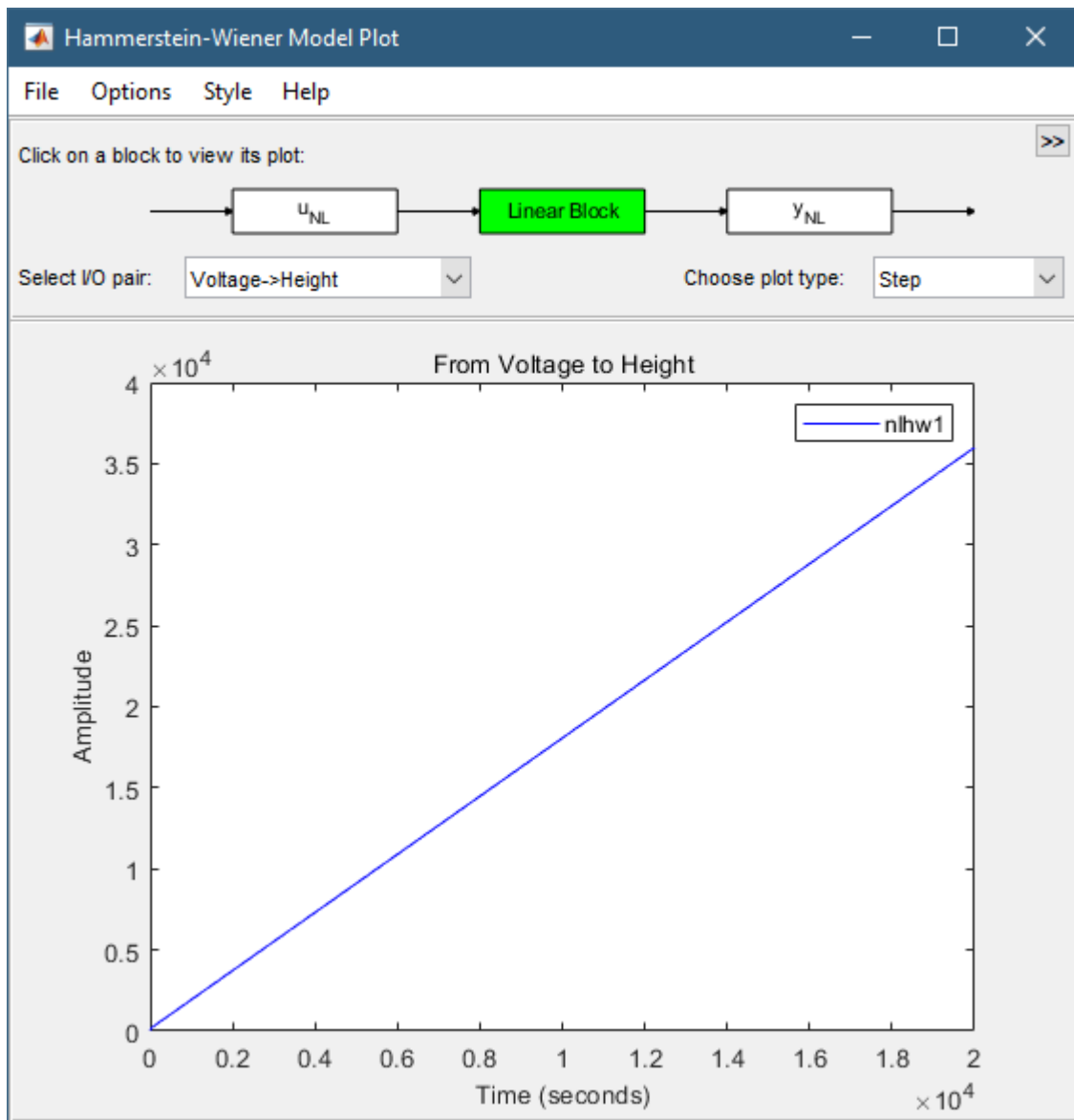
4 In the **Choose plot type** list, select Bode. This action displays a Bode plot of the linear transfer function.
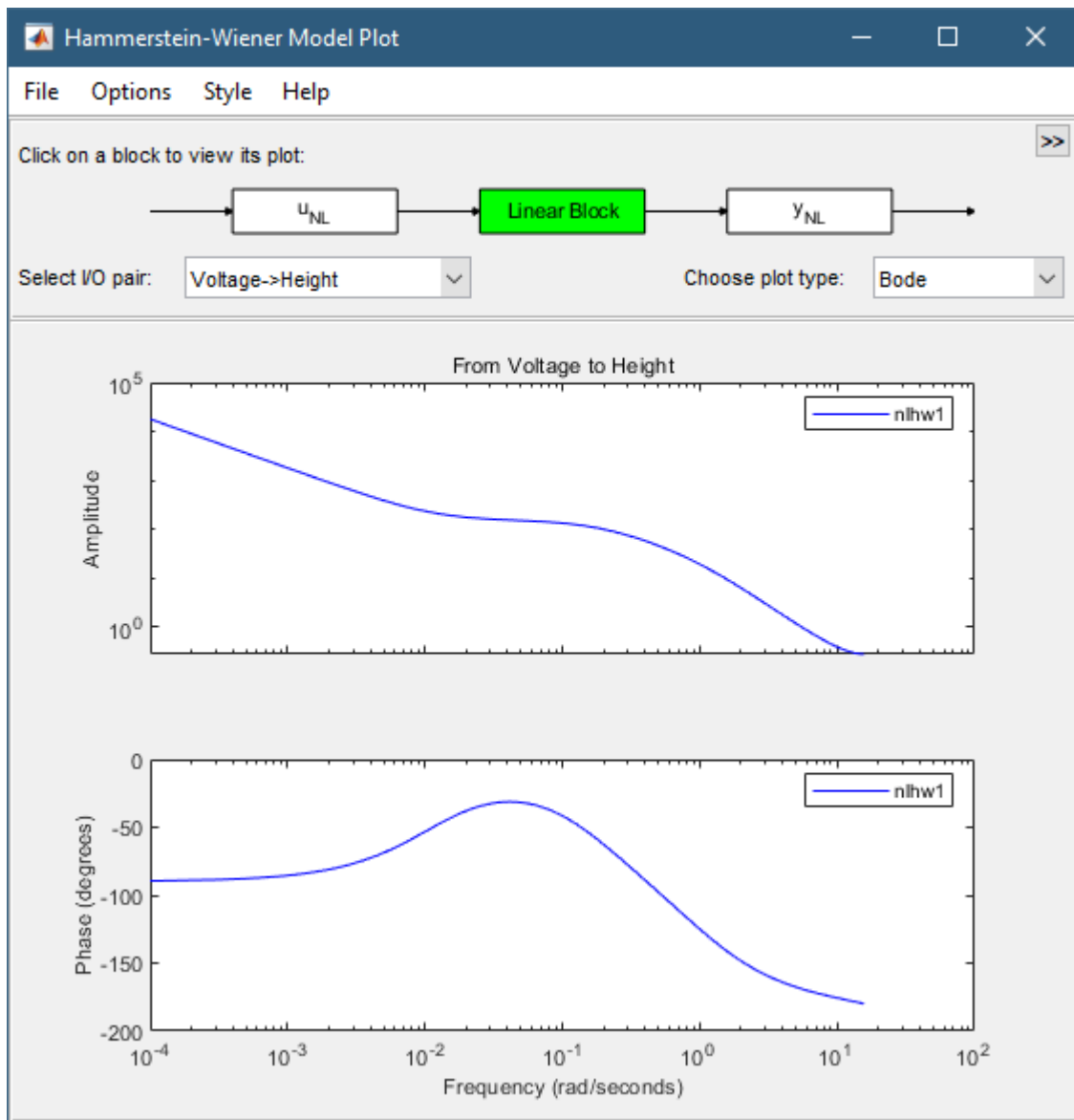
Change Hammerstein-Wiener Model Input Delay

In this portion of the tutorial, you estimate a Hammerstein-Wiener model with a specific model order and nonlinearity settings. Typically, you select model orders and delays by trial and error until you get a model that produces a satisfactory fit to the data.

You must have already estimated the Hammerstein-Wiener model with default settings, as described in "Estimate Hammerstein-Wiener Models with Default Settings" on page 4-31.

1   In the **Estimate Hammerstein-Wiener** dialog box, select the **Model Structure** tab and then the **Linear Block** tab.

2   For the `Voltage` input channel, double-click the corresponding **Input Delay (nk)** cell and enter the value 3.

3   Click **Estimate**.

This action adds the model `nlhw2` to the **System Identification** app and updates the **Model Output** plot, as shown in the following figure.
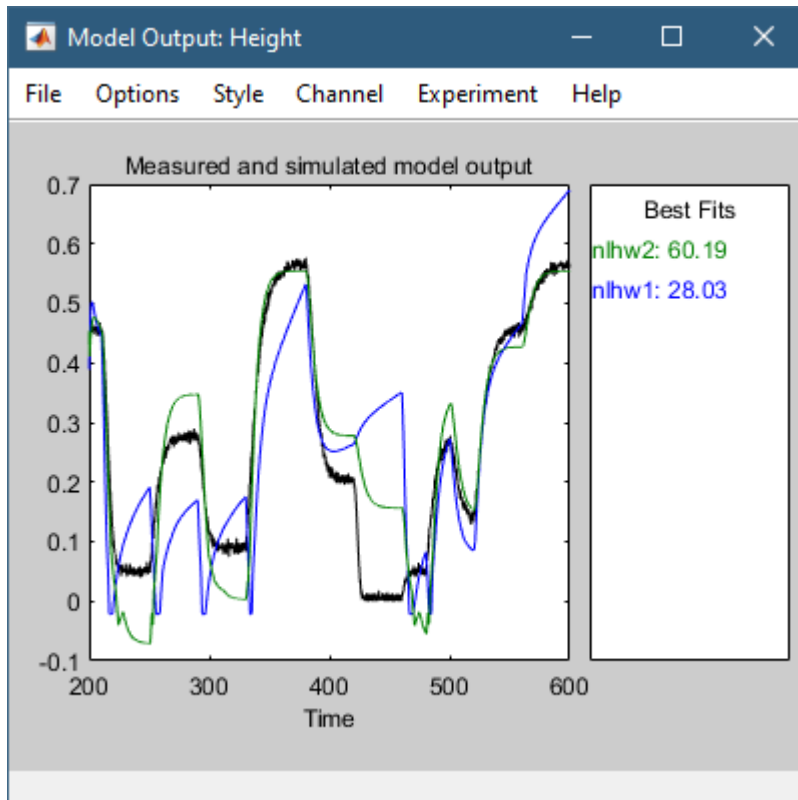
The **Best Fits** panel of the Model Output window shows the quality of the `nlhw2` fit, which is an improvement over that of the model `nlhw1`.

Change Nonlinearity Estimator in Hammerstein-Wiener Model

In this portion of the example, you modify the default Hammerstein-Wiener model structure by changing its nonlinearity estimator.

Tip If you know that your system includes saturation or dead-zone nonlinearities, you can specify these specialized nonlinearity estimators in your model. `Piecewise Linear`, `Wavelet Network`, and `Sigmoid Network` are nonlinearity estimators for general nonlinearity approximation.

1 In the **Estimate Hammerstein-Wiener Models** dialog box, select the **Model Structure** tab and then the **Input Nonlinearity** tab.

2 Click the **Nonlinearity** cell for the `Voltage` variable and select `Sigmoid Network` from the list.

Input Nonlinearity	Linear Block	Output Nonlinearity

u → Input → Input Nonlinearity → x → Linear Block (dynamic) → w

Input

Input	Nonlinearity
Voltage	Sigmoid Network ▾
	Piecewise Linear

Properties for "Sigmoid Network" non [Sigmoid Network]

Saturation

Deadzone

Wavelet Network

Enter the number of units   10   One-dimensional Polynomial

Custom Network

None
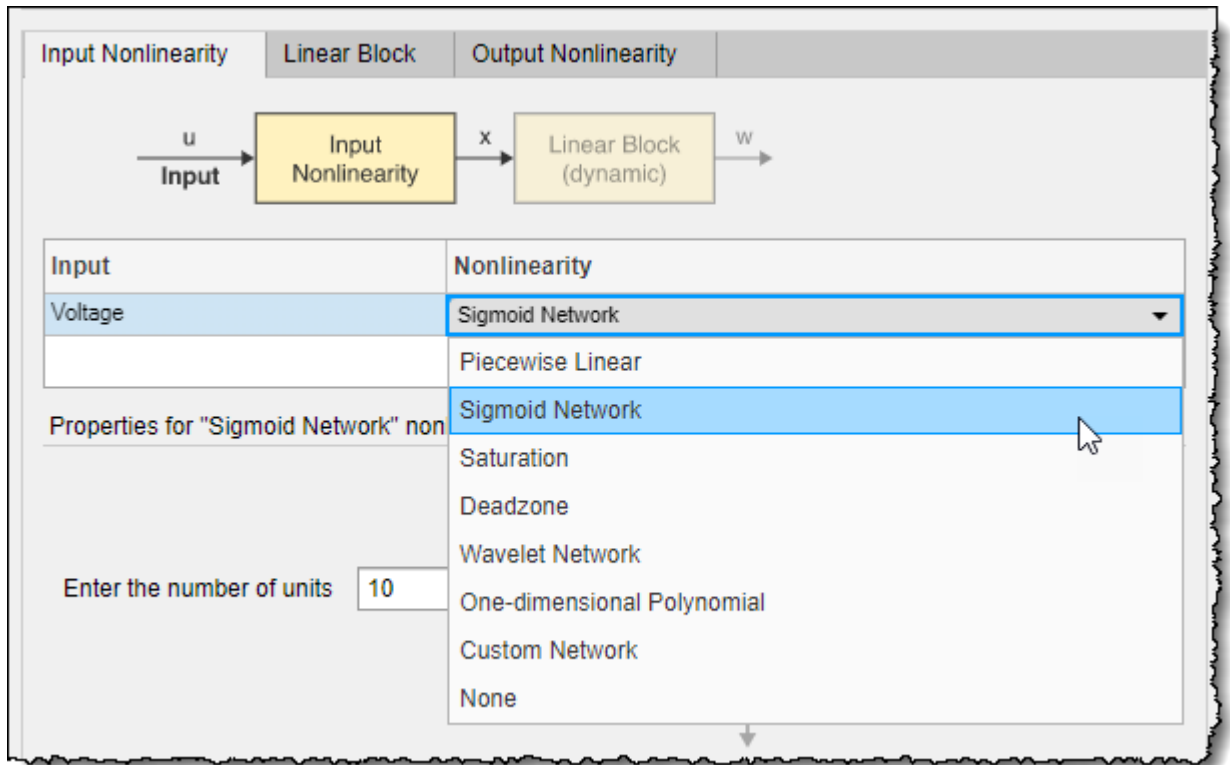
3   In **Enter the number of units**, set the value to 20.

Input	Nonlinearity
Voltage	Sigmoid Network

Properties for "Sigmoid Network" nonlinearity for input "Voltage"

Enter the number of units   20

F(x)

Sigmoid

x
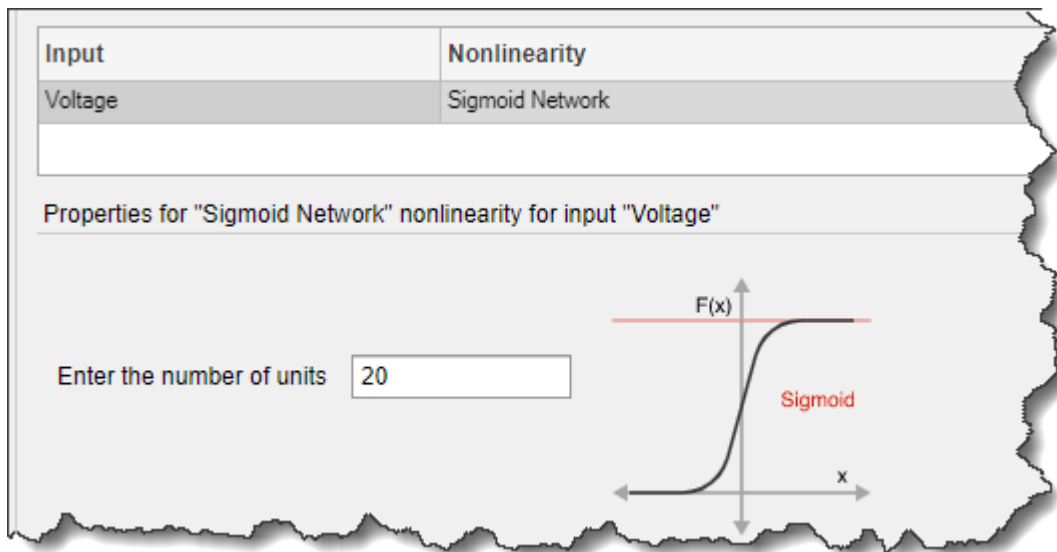
4   Click **Estimate**.

This action adds the model nlhw3 to the **System Identification** app. It also updates the **Model Output** window, as shown in the following figure.
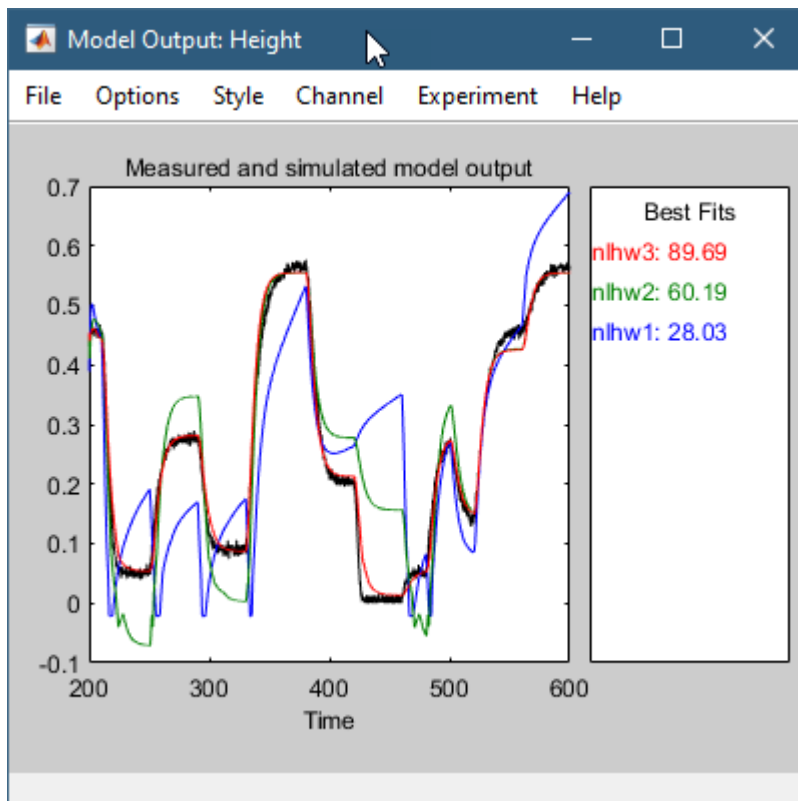
5 In the **Estimate Hammerstein-Wiener Models** dialog box, select the **Input Nonlinearity** tab.

6 For the `Voltage` input, set **Nonlinearity** to `Wavelet Network`. The default value for **Number of Units** is **Select Automatically**.

7   Select the **Output Nonlinearity** tab.

8   Set the `Height` output **Nonlinearity** to `One-dimensional Polynomial`. Set **Degree of polynomial** to 2.

9 Click **Estimate**.

This action adds the model `nlhw4` to the **System Identification** app. It also updates the **Model Output** window, as shown in the following figure.

Select the Best Model

The best model is the simplest model that accurately describes the dynamics.

In this example, the models `nlhw3` and `nlhw4` are good candidates.