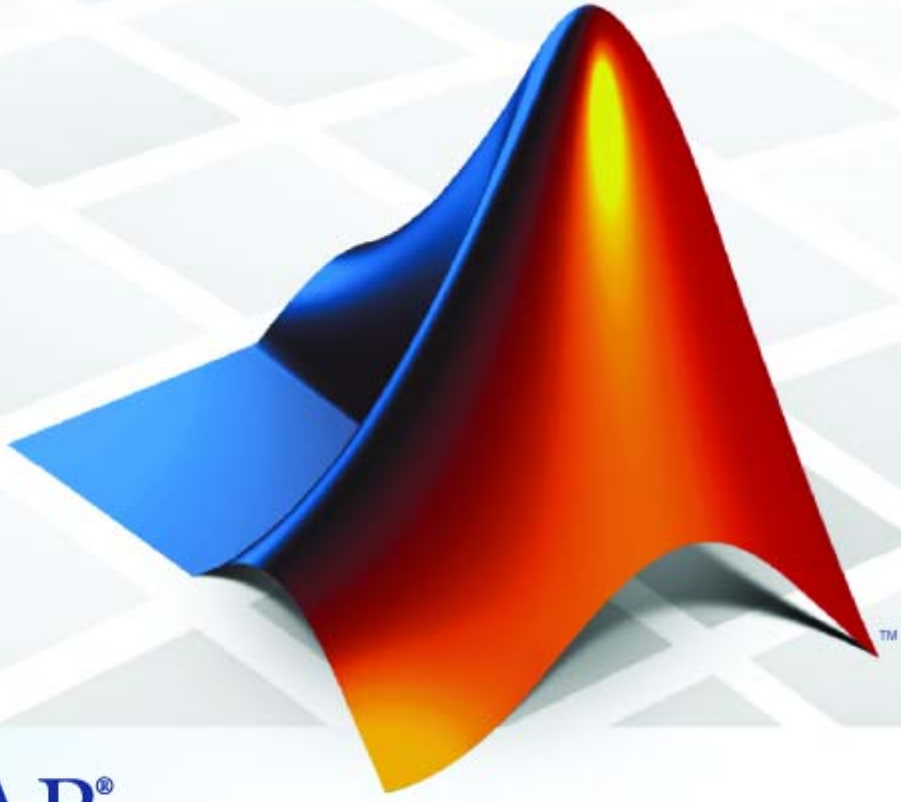
System Identification Toolbox™ 7
Getting Started Guide

Lennart Ljung



MATLAB® & SIMULINK®

The MathWorks™
Accelerating the pace of engineering and science

How to Contact The MathWorks

`www.mathworks.com`	Web
`comp.soft-sys.matlab`	Newsgroup
`www.mathworks.com/contact_TS.html`	Technical Support

`suggest@mathworks.com`	Product enhancement suggestions
`bugs@mathworks.com`	Bug reports
`doc@mathworks.com`	Documentation error reports
`service@mathworks.com`	Order status, license renewals, passcodes
`info@mathworks.com`	Sales, pricing, and general information

508-647-7000 (Phone)

508-647-7001 (Fax)

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

For contact information about worldwide offices, see the MathWorks Web site.

System Identification Toolbox™ Getting Started Guide

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See `www.mathworks.com/trademarks` for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Patents

The MathWorks products are protected by one or more U.S. patents. Please see `www.mathworks.com/patents` for more information.

Revision History

March 2007	First printing	New for Version 7.0 (Release 2007a)
September 2007	Second printing	Revised for Version 7.1 (Release 2007b)
March 2008	Third printing	Revised for Version 7.2 (Release 2008a)
October 2008	Online only	Revised for Version 7.2.1 (Release 2008b)
March 2009	Online only	Revised for Version 7.3 (Release 2009a)
September 2009	Online only	Revised for Version 7.3.1 (Release 2009b)
March 2010	Online only	Revised for Version 7.4 (Release 2010a)

About the Developers

System Identification Toolbox™ software is developed in association with the following leading researchers in the system identification field:

Lennart Ljung. Professor Lennart Ljung is with the Department of Electrical Engineering at Linköping University in Sweden. He is a recognized leader in system identification and has published numerous papers and books in this area.

Qinghua Zhang. Dr. Qinghua Zhang is a researcher at Institut National de Recherche en Informatique et en Automatique (INRIA) and at Institut de Recherche en Informatique et Systèmes Aléatoires (IRISA), both in Rennes, France. He conducts research in the areas of nonlinear system identification, fault diagnosis, and signal processing with applications in the fields of energy, automotive, and biomedical systems.

Peter Lindskog. Dr. Peter Lindskog is employed by NIRA Dynamics AB, Sweden. He conducts research in the areas of system identification, signal processing, and automatic control with a focus on vehicle industry applications.

Anatoli Juditsky. Professor Anatoli Juditsky is with the Laboratoire Jean Kuntzmann at the Université Joseph Fourier, Grenoble, France. He conducts research in the areas of nonparametric statistics, system identification, and stochastic optimization.

Contents

1 Product Overview

2 About System Identification

3 Using This Product

4 Tutorial – Identifying Linear Models Using the GUI

Tutorial – Identifying Linear Models Using the Command Line

6

Tutorial – Identifying Nonlinear Black-Box Models Using the GUI

7

Index

1

Product Overview

Why Use This Toolbox?

System Identification Toolbox software lets you estimate linear and nonlinear mathematical models of dynamic systems from measured data. Use the resulting models for analyzing system dynamics, simulating the output of a system for a given input, predicting future outputs based on previous observations of inputs and outputs, or for control design.

System identification is especially helpful for modeling systems that you cannot easily model from first principles or specifications, such as engine subsystems, thermofluid processes, and electromechanical systems. Such *black-box models* can simplify detailed first-principle models, such as finite-element models of structures and flight dynamics models, by fitting simpler models to their simulated responses.

You can also use System Identification Toolbox functions to compute the coefficients of ordinary differential and difference equations for systems modeled from first principles. Such models are called *grey-box models*.

For real-time applications in adaptive control, adaptive filtering, or adaptive prediction, you can use this product to perform recursive parameter estimation.

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.
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.
Neural Network 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.

Product	Description
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.

Documentation

System Identification Toolbox documentation includes:

- Getting Started Guide — Summarizes the capabilities of the System Identification Toolbox software and provides an overview of system identification. Step-by-step tutorials walk you through the most common System Identification Toolbox tasks.
- User's Guide — Describes the various tasks of using the System Identification Toolbox software.
- Reference — Describes System Identification Toolbox commands.
- Release Notes — Describes important changes in the current product version and compatibility considerations.

View the documentation online from the **Help** menu on the MATLAB® desktop.

2

About System Identification

What Is System Identification?

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

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 value for the adjustable parameters in the candidate model structure.
- Evaluate the estimated model to see if the model is adequate for your application needs.

About Dynamic Systems and Models

In this section...

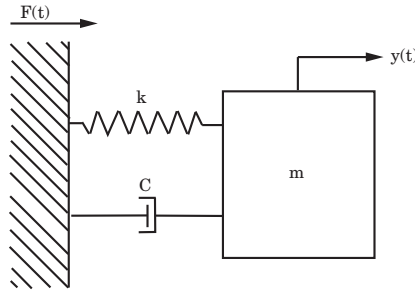
What Is a Dynamic Model?

In a dynamic system, the values of the output signals depend on both the instantaneous values of its 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 this passenger has been riding in the car (past behavior).

A *model* is a mathematical relationship between a system's input and output variables. 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 both in 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 shown in the next figure, 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.



Mass-Spring-Damper System Excited by Force F(t)

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:

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

where m is the mass, k the spring's stiffness constant, and c 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.

If you treat displacement $y(t)$ and velocity $v(t) = \frac{dy(t)}{dt}$ as state variables, you can express the previous equation of motion as a state-space model of the system:

$$\frac{dY}{dt} = AY(t) + BF(t)$$
$$y(t) = CY(t)$$

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

A = [0 1; $-k/m$ $-c/m$]

B = [0; 1/m]

C = [1 0]

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

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

where s is the Laplace variable.

Discrete-Time Dynamic Model Example

Suppose you can only observe 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, ... The variables are said to be *sampled* with sampling interval 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:

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

where a_1 and a_2 are the model parameters. The model parameters are related to the system constants m, c, and k, and the sampling interval 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 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 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 "Simulating and Predicting Model Output" in the User's Guide.

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 2-4.

System Identification Requires Measured Data

In this section...

Why Does System Identification Require Data?

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.

Using this toolbox, you build models using time-domain input-output signals, frequency response data, time series signals, and time-series spectra.

Obtaining a good model of your system depends on how well your measured data reflects the behavior of the system. See "Data Quality Requirements" on page 2-8.

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 $x(t)$ of the spring-mass-damper system at a uniform sampling frequency of 10 Hz, you obtain the following vectors of measured values:

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

where T_s = 0.1 seconds and NT_s is 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 sampling interval T_s provide sufficient information for creating such a model.

If you want to build a continuous-time model, you should also know the intersample behavior of the input signals during the experiment. For example, the input may 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 case of 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 data acquisition system to have 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 techniques available in the Signal Processing Toolbox software. For example, analyze the input spectra to determine if the input signals have sufficient power over the bandwidth of the system.

You can also analyze your data to determine peak frequencies, input delays, important time constants, and indication of nonlinearities using non-parametric analysis tools in this toolbox. You can use this information for configuring model structures for building models from data. See the following User's Guide topics:

- "Identifying Impulse-Response Models"
- "Identifying Frequency-Response Models"

Data Representation in This Toolbox

If you build models using the System Identification Tool GUI, you must import your data into the GUI.

If you use the command-line interface, specify your data using `iddata` and `idfrd` objects. These objects conveniently store data values and other information about the data, such as its sampling interval and intersample behavior.

For more information, see "Data Import and Processing" in the User's Guide.

Building Models from Data

In this section...

System Identification Requires a 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:

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

where 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 may have derived from first principles, but do not know numerical values of its parameters. You can then represent the model structure as a set of equations or state-space system in MATLAB and estimate the values of its parameters from data. This approach is known as *grey-box modeling*.

How the Toolbox Computes 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:

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

where 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:

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

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

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

Accordingly, the error $v(t)$ is called *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.

Configuring the Parameter Estimation Algorithm

You can configure the estimation algorithm by:

- Configuring the minimization criterion to focus the estimation in a desired frequency range, such as 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 the topics for estimating specific model structures in the *System Identification Toolbox User's Guide*.

Black-Box Modeling

In this section...

Selecting 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 models 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:

- 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 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. Higher model order increases the model flexibility for capturing complex phenomena. However, unnecessarily high orders can make the model less reliable.
- Explicitly modeling the noise:

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

  where 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" in the User's Guide.
- Using a nonlinear model structure.

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

Ultimately, you choose the simplest model structure that provides the best fit to your measured data. For more information, see Chapter 4, "Tutorial – Identifying Linear Models Using the GUI".

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 convert an estimated model into an linear time-invariant (LTI) object and linearize a nonlinear model about an operating point.

When to Use Nonlinear Model Structures?

A linear model is often sufficient to accurately describe the system dynamics and, in most cases, you should 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 via 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 current and voltage and then build 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" in User's Guide.

Black-Box Estimation Example

You can use the GUI 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 "About Dynamic Systems and Models" on page 2-3. 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:

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

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).

You can build a linear black-box model of a mass-spring-damper system using an Output Error structure using the following command:

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

where data is your measured input-output data, represented as an `iddata` object and the model order is [`nb nf nk`] = [`1 2 1`]. Usually, you do not know the model orders in advance. You should 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 `n4sid` command:

```
m = n4sid(data, 2)
```

where *`order`* = `2` represents the number of states in the model.

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

For more information about building models, see “Steps for Using the System Identification Tool GUI” and “Commands for Model Estimation” in the User’s Guide.

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 mass-spring-damper system is well known. In state-space form, the model is given by:

$$\frac{dY}{dt} = AY(t) + BF(t)$$
$$y(t) = CY(t)$$

where $Y(t) = [y(t);v(t)]$ 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 in the User's Guide
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 $k = -A(2,1)m$.	• "How to Estimate State-Space Models with Canonical Parameterization" • "How to 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`.	"ODE Parameter Estimation (Grey-Box Modeling)"

Evaluating Model Quality

In this section...

How to Evaluate and Improve Model Quality

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

- "Comparing Model Response to Measured Response" on page 2-20
- "Analyzing Residuals" on page 2-22
- "Analyzing Model Uncertainty" on page 2-22

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 in the User's Guide.

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. For more information about estimating each type of model structure, see the User's Guide. If these changes do not improve your results, you might need to revisit your experimental design and data gathering procedures.

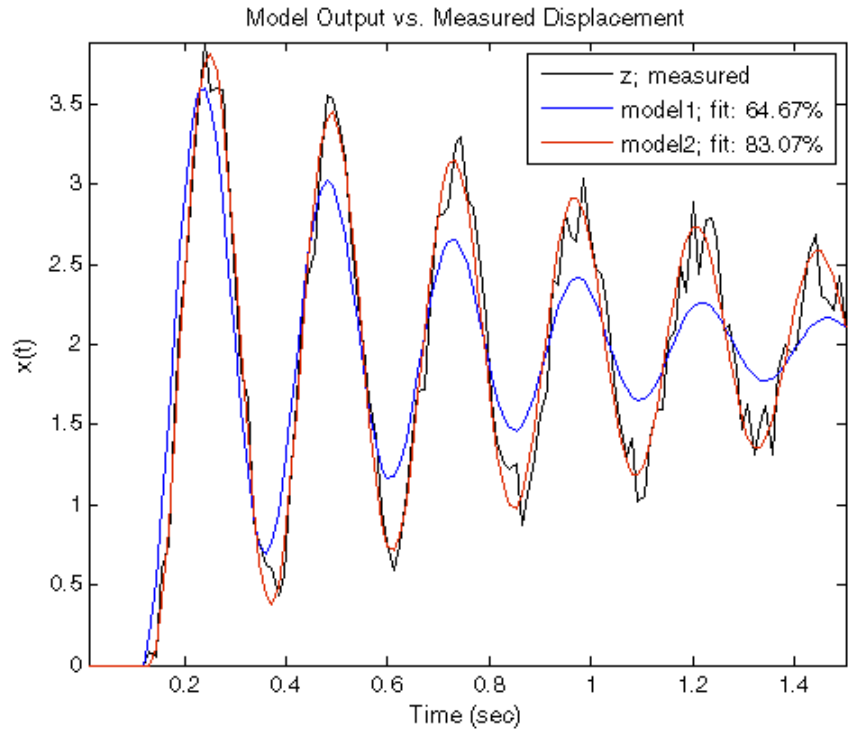
Comparing Model Response to Measured Response

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 previous figure indicates that `model2` is better than `model1` because `model2` better fits the data (65% vs. 83%).

The % fit 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 “Simulating and Predicting Model Output” in the User’s Guide.

Analyzing Residuals

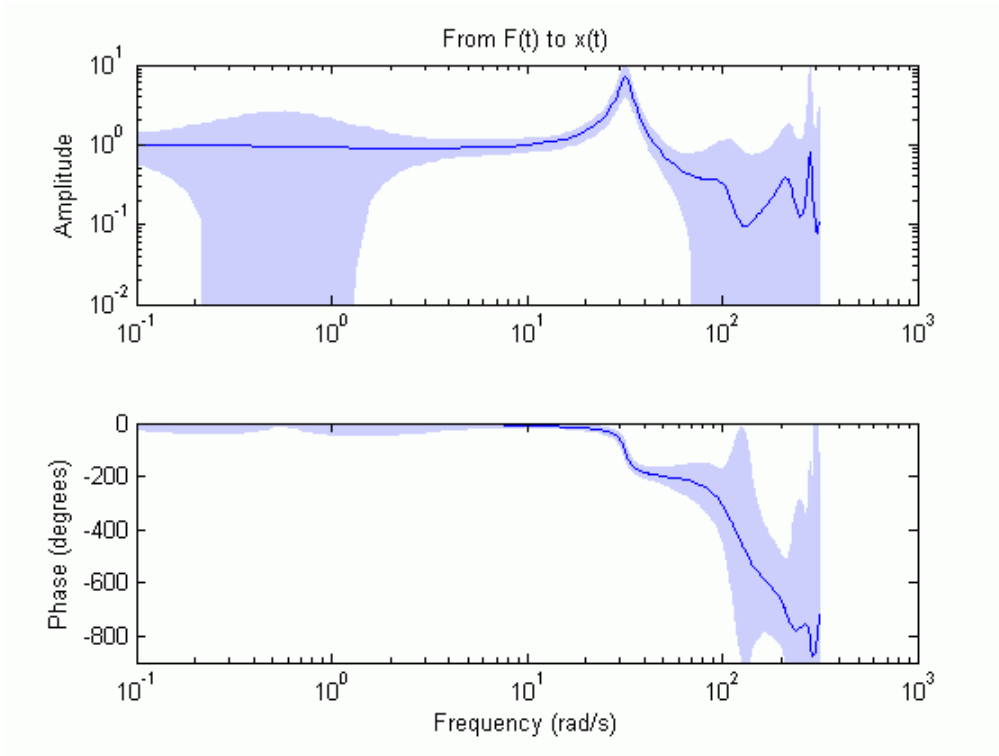
The System Identification Toolbox software lets you perform residual analysis to assess the model quality. Residuals represent the portion of the output data not explained by the estimated model. A good model has residuals uncorrelated with past inputs.

For more information, see “Residual Analysis” in the User’s Guide.

Analyzing 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 poor signal-to-noise ratio in measured data.

You can compute and visualize the effect of parameter uncertainties on the model response in time and frequency domains using pole-zero maps, Bode response, 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 model’s frequency response, 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 “Computing Model Uncertainty” in the User’s Guide.

Learn More

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, L., and T. Glad. *Modeling of Dynamic Systems*. PTR Prentice Hall, Upper Saddle River, NJ, 1994.

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

- Ljung, L. *System Identification: Theory for the User*. Second edition. PTR Prentice Hall, Upper Saddle River, NJ, 1999.
- Söderström, T., and P. Stoica. *System Identification*. Prentice Hall International, London, 1989.

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

Pintelon, R., and J. Schoukens. *System Identification. A Frequency Domain Approach*. Wiley-IEEE Press, New York, 2001.

For information on nonlinear identification, see the following references:

- Sjöberg, J., Q. Zhang, L. Ljung, A. Benveniste, B. Deylon, P. Glorennec, H. Hjalmarsson, and A. Juditsky, "Nonlinear Black-Box Modeling in System Identification: a Unified Overview." *Automatica*. Vol. 31, Issue 12, 1995, pp. 1691–1724.
- Juditsky, A., H. Hjalmarsson, A. Benveniste, B. Delyon, L. Ljung, J. Sjöberg, and Q. Zhang, "Nonlinear Black-Box Models in System Identification: Mathematical Foundations." *Automatica*. Vol. 31, Issue 12, 1995, pp. 1725–1750.
- Zhang, Q., and A. Benveniste, "Wavelet networks." *IEEE Transactions on Neural Networks*. Vol. 3, Issue 6, 1992, pp. 889–898.

- Zhang, Q., "Using Wavelet Network in Nonparametric Estimation." *IEEE Transactions on Neural Networks.* Vol. 8, Issue 2, 1997, pp. 227–236.

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

Oppenheim, J., and Willsky, A.S. *Signals and Systems*. PTR Prentice Hall, Upper Saddle River, NJ, 1985.

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

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

3

Using This Product

When to Use the GUI Versus the Command Line

New users should start by using the System Identification Tool GUI to become familiar with the product.

You can work either in the GUI 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. See topics about estimating linear models recursively in the *System Identification Toolbox User's Guide*.
- Converting between continuous-time and discrete-time models (see `c2d` and `d2c` reference pages).
- Converting models to Control System Toolbox LTI objects (see the `ss`, `tf`, and `zpk` reference pages).

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

Tip To learn more about estimating and validating models at the command line, see Chapter 6, "Tutorial – Identifying Linear Models Using the Command Line".

Starting This Toolbox

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

For information about whether to use the GUI or the command line, see "When to Use the GUI Versus the Command Line" on page 3-2.

To open the System Identification Tool GUI:

- Select **Start > Toolboxes > System Identification > System Identification Tool** from the MATLAB desktop.

Alternatively, you can open the System Identification Tool GUI by typing the following command in the MATLAB Command Window:

```
ident
```

To work at the command line, type the commands directly in the MATLAB Command Window. For more information about supported commands, see the reference pages.

Steps for Using This Toolbox

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 Tool GUI 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
   - 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 Simulate or predict model output.

5 Design a controller for the estimated plant using other MathWorks products.

   You can import an estimated linear model into the Control System Toolbox, Model Predictive Control Toolbox, Robust Control Toolbox, or Simulink products for control design. For more information about linearizing a nonlinear plant, see the `linapp` and `linearize` reference pages.

Commands for Model Estimation

The following table summarizes System Identification Toolbox estimation commands. For detailed information about using each command, see the corresponding reference page.

Commands for Constructing and Estimating Models

Model Type	Estimation Commands
Continuous-time low-order transfer functions (process models)	`pem`
Linear input-output polynomial models	`armax` (ARMAX only) `arx` (ARX only) `bj` (BJ only) `iv4` (ARX only) `oe` (OE only) `pem` (for all models)
State-space models	`n4sid` `pem`

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

Nonlinear ARX models	`nlarx`
Hammerstein-Wiener models	`nlhw`

Tutorials to Help You Get Started

You can use the following tutorials to help you quickly get started with the System Identification Toolbox software.

Tutorial	Description
Chapter 4, "Tutorial – Identifying Linear Models Using the GUI"	You learn how to identify and compare different linear black-box models from data using the System Identification Tool GUI.
Chapter 5, "Tutorial – Identifying Low-Order Transfer Functions (Process Models) Using the GUI"	You learn how to estimate the parameters of low-order, continuous-time transfer functions from data using the System Identification Tool GUI.
Chapter 6, "Tutorial – Identifying Linear Models Using the Command Line"	You learn how to identify different linear models from data using System Identification Toolbox commands.
Chapter 7, "Tutorial – Identifying Nonlinear Black-Box Models Using the GUI"	You learn how to identify nonlinear black-box models from data using the System Identification Tool GUI.

4

Tutorial – Identifying Linear Models Using the GUI

About This Tutorial

In this section...

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 Tool GUI:

- Import data arrays from the MATLAB workspace into the GUI.
- 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

In this section...

Loading Data into the MATLAB Workspace

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

```
load dryer2
```

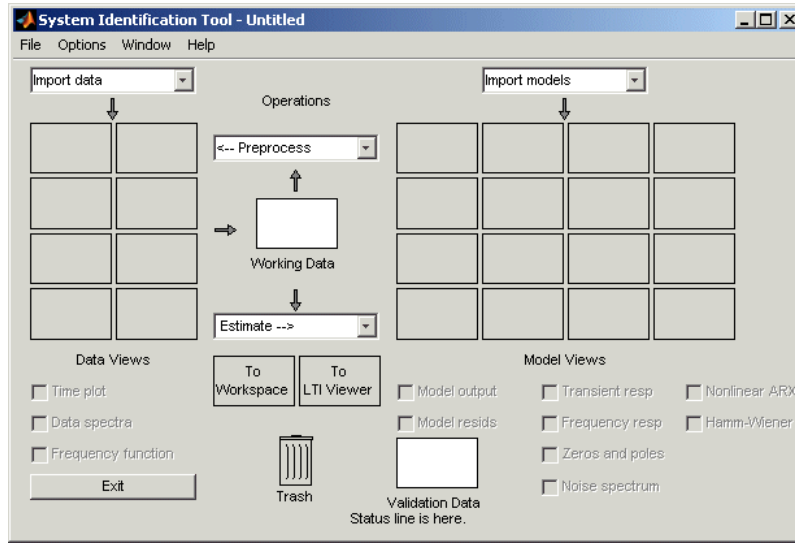
This command loads the data into the MATLAB workspace as two column vectors, `u2` and `y2`, respectively. The variable `u2` is the input data and `y2` is the output data.

Opening the System Identification Tool GUI

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

```
ident
```

The default session name, `Untitled`, appears in the title bar.



Importing Data Arrays into the System Identification Tool

You can import the single-input/single-output (SISO) data from a sample data file `dryer2.mat` into the GUI 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 5-5, and opened the System Identification Tool GUI, as described in “Opening the System Identification Tool GUI” on page 4-4.

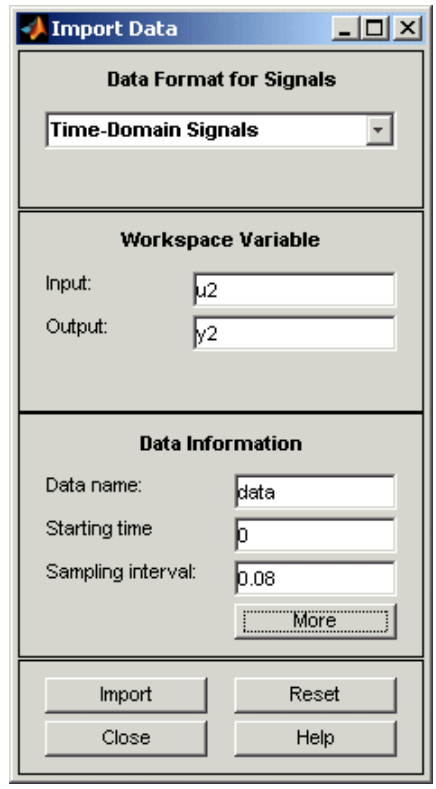
To import data arrays into the System Identification Tool GUI:

1. In the System Identification Tool GUI, select **Import data > Time domain data**. This action opens the Import Data dialog box.

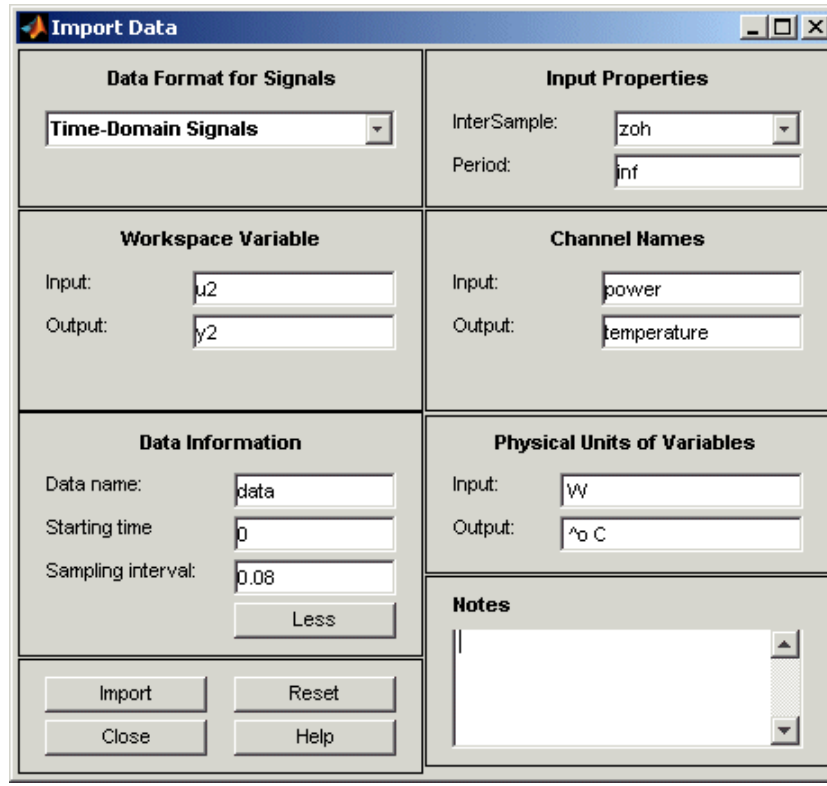
2. Specify the following options:

   - **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 Tool GUI 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.
   - **Sampling interval** — Enter `0.08` as the time between successive samples in seconds. This value is the actual sampling interval 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. Enter the settings shown in the following figure.



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.

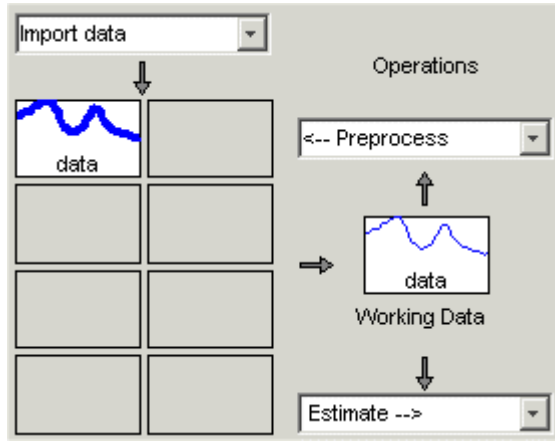
Tip When you have multiple inputs and outputs, enter a comma-separated list of **Input** and **Output** units corresponding to each channel.

- **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 data notes.

4 Click **Import** to add the icon named `data` to the System Identification Tool GUI.



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:

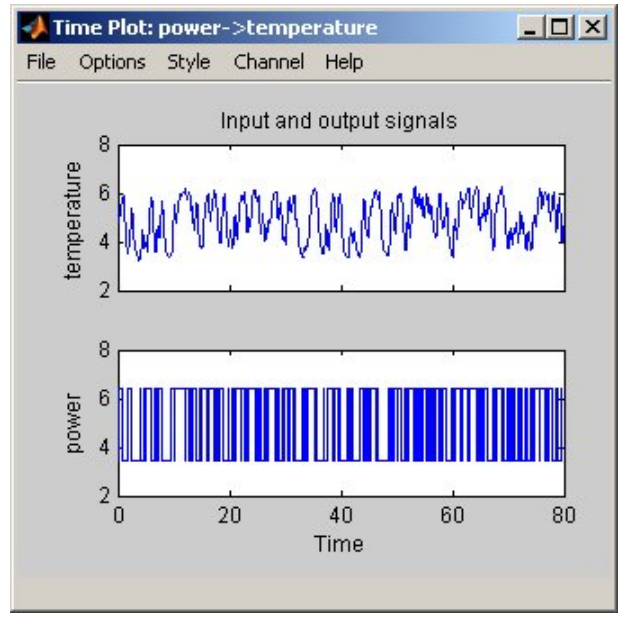
- Plot the data.
- Subtract the mean values of the input and the output to remove offsets.
- Split the data into two parts. You use one part of the data for model estimation, and the other part of the data for model validation.

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 Tool, as described in "Importing Data Arrays into the System Identification Tool" on page 4-5.

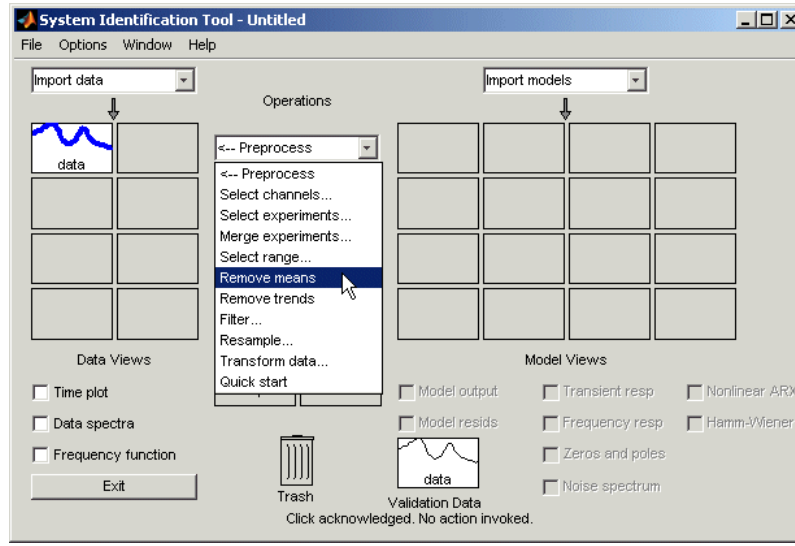
To plot and process the data:

1. In the System Identification Tool GUI, 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 Tool GUI.

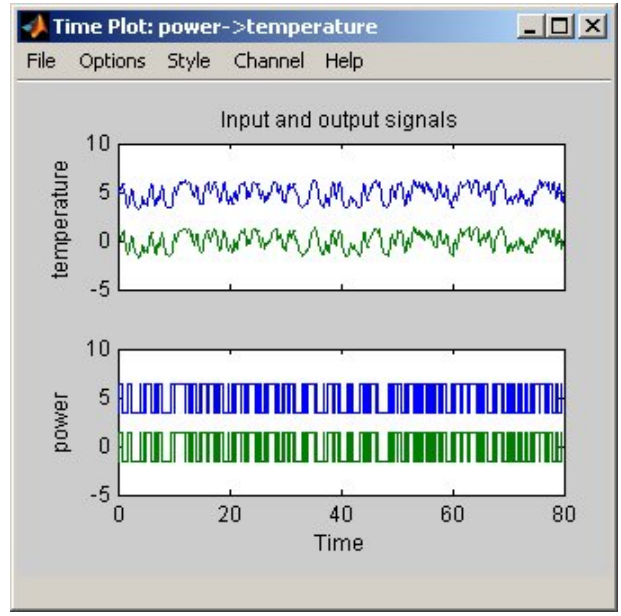


The top axes show the output data (temperature), and the bottom axes show the input data (power). Both the input and the output data have nonzero mean values.

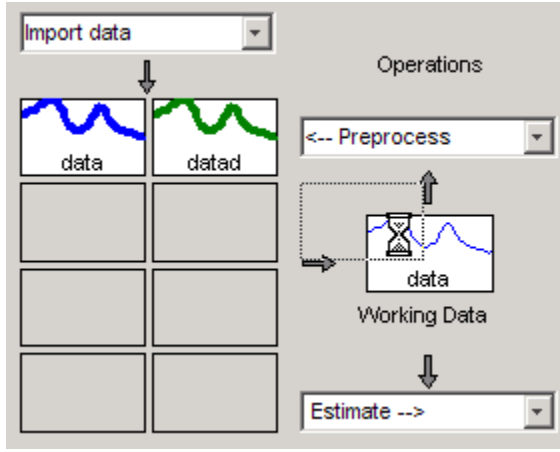
2 In the System Identification Tool GUI, select **<--Preprocess > Remove means** to subtract the mean input value from the input data and the mean output value from the output data.



This action adds a new data set to the System Identification Tool GUI 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 In the System Identification Tool GUI, drag the `datad` data set to the **Working Data** rectangle. This action specifies the detrended data to be used for estimating models.

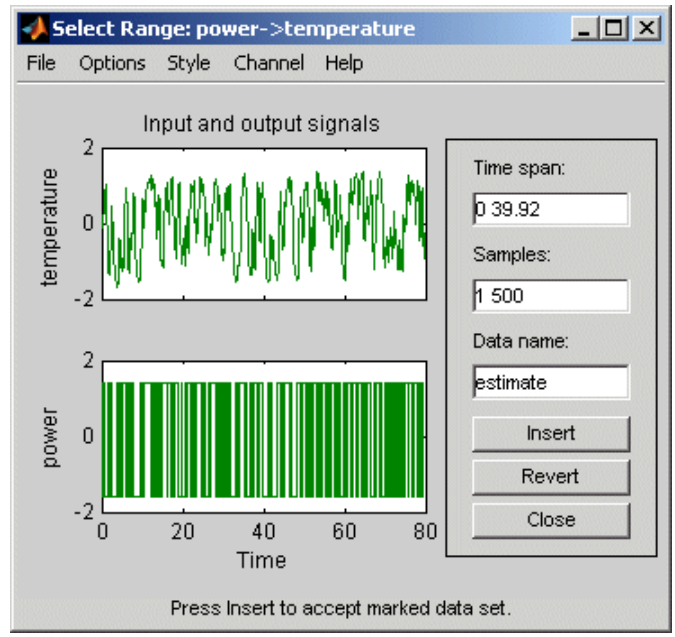


4 Select **<--Preprocess > Select range** to open the Select Range window.

In this window, you can split the data into two parts and specify the first part for model estimation, and the second part for model validation, as described in the following steps.

5 In the Select Range window, change the **Samples** field to select the first 500 samples, as follows:

```
1  500
```

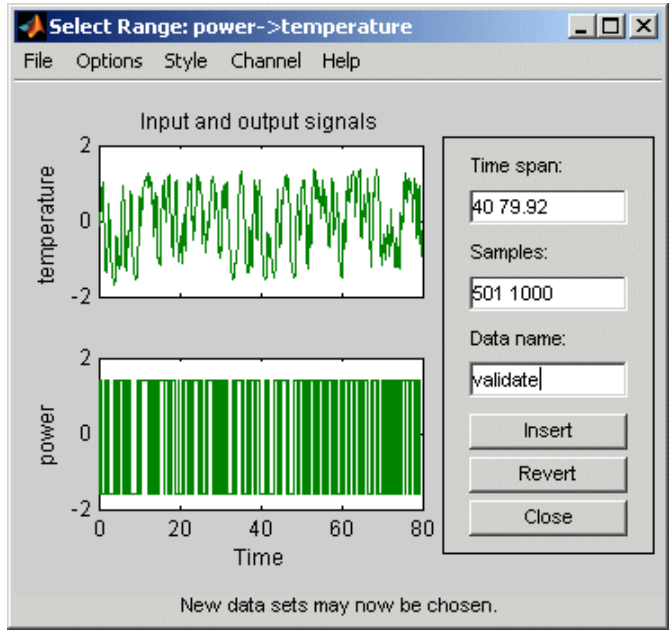


Tip You can also select data samples using the mouse by clicking and dragging a rectangular region on the plot. If you select samples on the input-channel axes, the corresponding region is also selected on the output-channel axes.

6 In the **Data name** field, type the name `estimate`, and click **Insert**. This action adds a new data set to the System Identification Tool GUI to be used for model estimation.

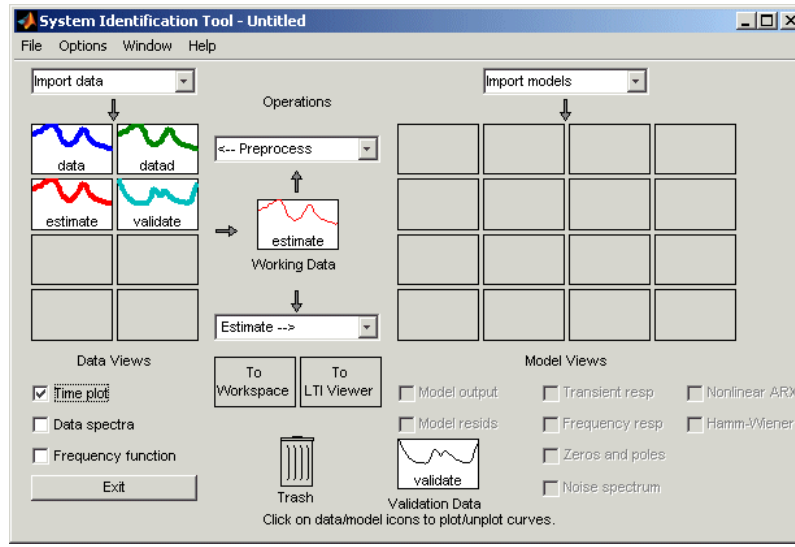
7 In the Select Range window, change the **Samples** field to select the last 500 samples, as follows:

```
501  1000
```

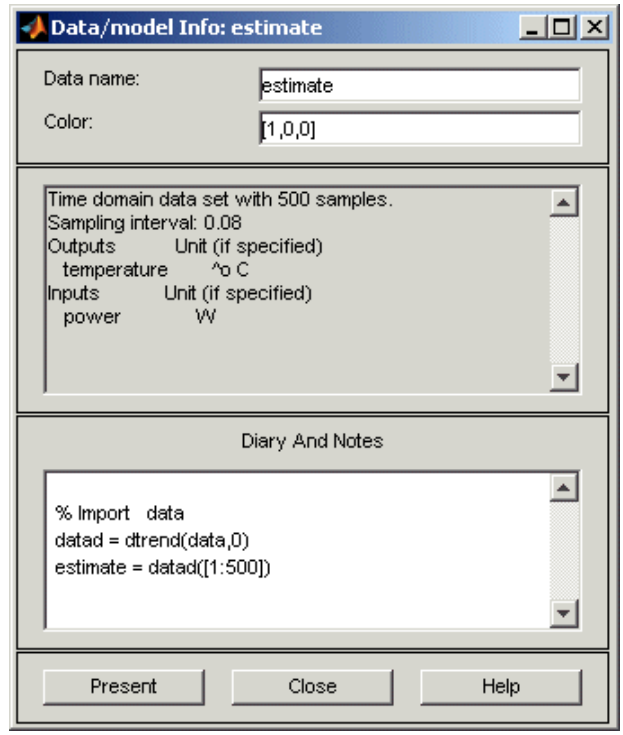


8 In the **Data name** field, type the name `validate`, and click **Insert**. This action adds a new data set to the System Identification Tool GUI to be used for model validation.

9 Drag and drop `estimate` to the **Working Data** rectangle, and drag and drop `validate` to the **Validation Data** rectangle so that the System Identification Tool GUI resembles the following figure.



10 To get information about a data set, right-click its icon. For example, right-click the `estimate` data set to open the Data/model Info dialog box.



In the Data/model Info dialog box, you can perform the following actions:

- Change the name of the data set in the **Data name** field.
- Change the color of the data icon by changing the 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.
- In the noneditable area, view the total number of samples, the sampling interval, and the output and input channel names and units.
- In the editable **Diary And Notes** area, view or edit the actions you performed on this data set. The actions are translated into commands equivalent to your GUI operations. For example, as shown in the Data/model Info: estimate window, the `estimate` 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)
  estimate = datad([1:500])
  ```

  For more information about these and other toolbox commands, see the reference page corresponding to each command.

Tip As an alternative shortcut, you can select **Preprocess > Quick start** from the System Identification Tool GUI 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 the *System Identification Toolbox User's Guide*.

Saving the GUI Session

After you process the data, as described in "Plotting and Processing Data" on page 4-10, 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.

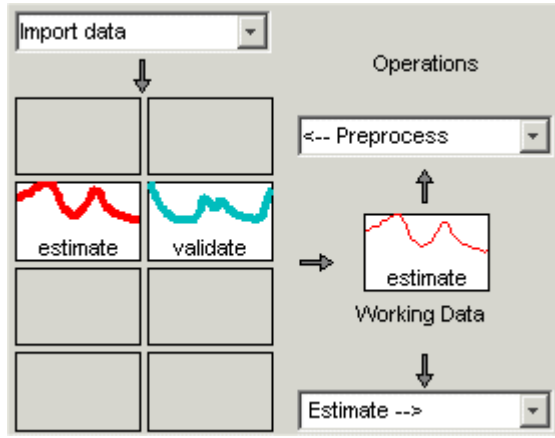
You must have already processed the data into the System Identification Tool, as described in "Plotting and Processing Data" on page 4-10.

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

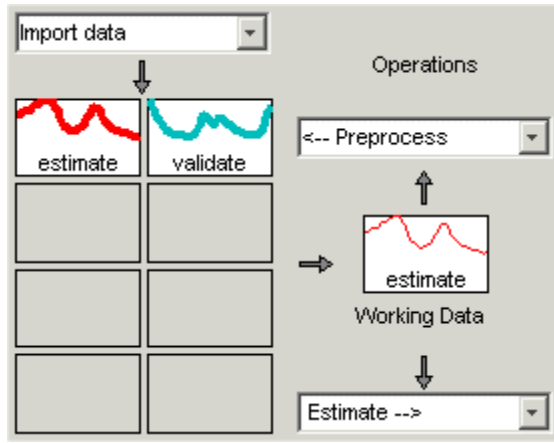
1. In the System Identification Tool GUI, drag and drop the `data` data set into the **Trash**.
2. Drag and drop the `datad` data set into the **Trash**.

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

The following figure shows the System Identification Tool GUI after moving the items to the **Trash**.



3 Drag and drop the `estimate` and `validate` data sets to fill the empty rectangles, as shown in the following figure.



4 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.

5 In the **File name** field, type the name of the session `dryer2_prep_data`, and click **Save**. The resulting file has a `.sid` extension.

Tip You can open a saved session when starting the System Identification Tool. For example, you can type the following command in the MATLAB Command Window:

```
ident('dryer2_prep_data')
```

For more information about managing sessions, see the topics on working with the System Identification Tool GUI in the *System Identification Toolbox User's Guide*.

Estimating Linear Models Using Quick Start

In this section...

How to Estimate Linear Models Using Quick Start

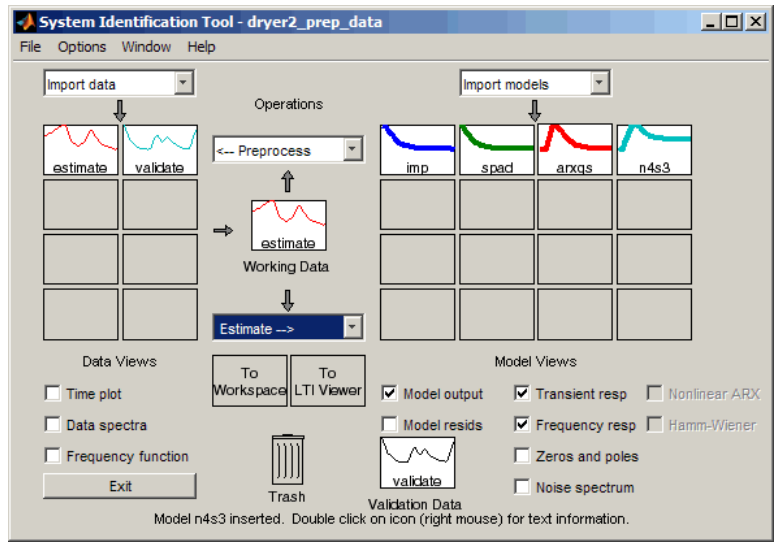
You can use the Quick Start feature in the System Identification Toolbox to estimate linear models. Quick Start might produce the final linear models you decide to use, or provide you with information required to configure the estimation of accurate parametric models, such as time constants, input delays,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

You must have already processed the data for estimation, as described in "Plotting and Processing Data" on page 4-10.

To identify linear models:

- In the System Identification Tool GUI, 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 4-25.



Types of Quick Start Linear Models

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

- `imp` — Step response over a period of time using the `impulse` 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:

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

  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. The System Identification Toolbox product estimates the parameters $a_1 \ldots a_n$ and $b_1 \ldots b_n$ 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:

  $$x(t+1) = Ax(t) + Bu(t) + Ke(t)$$
  $$y(t) = Cx(t) + Du(t) + e(t)$$

  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*.

Validating the Quick Start Models

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

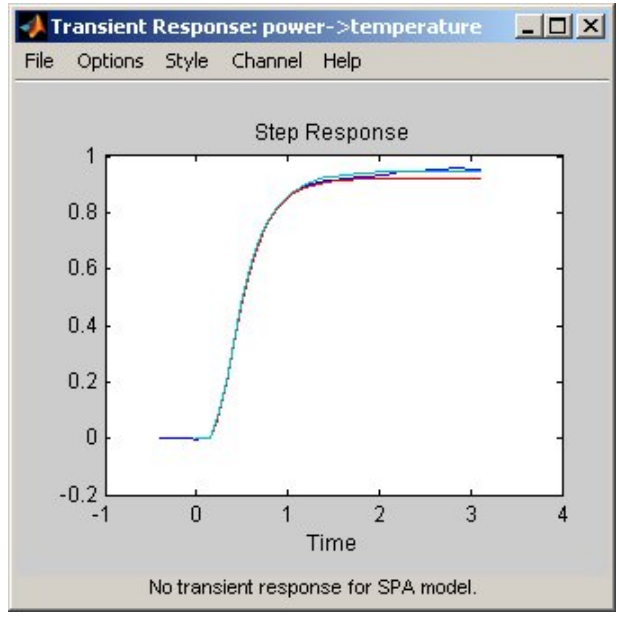
- Step-response plot
- Frequency-response plot
- Model-output plot

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 4-23.

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 Tool window to display the models on the plot.



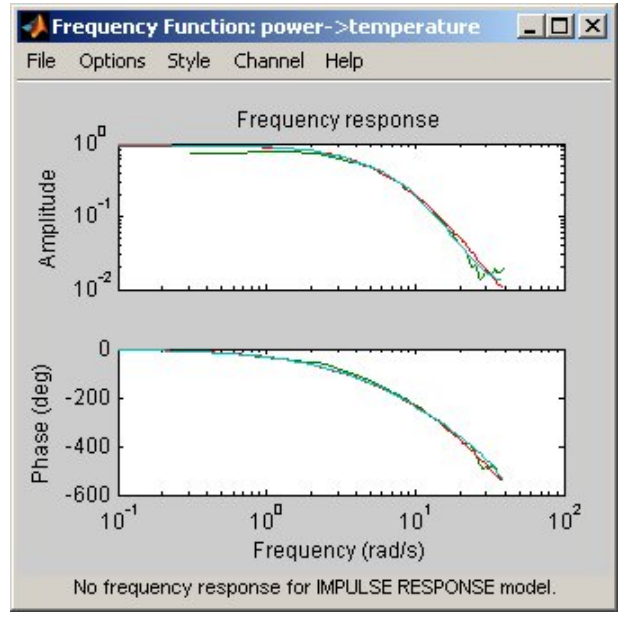
Step Response for imp, arxqs, and n4s3

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 sampling interval 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 Tool window to display the models on the plot.



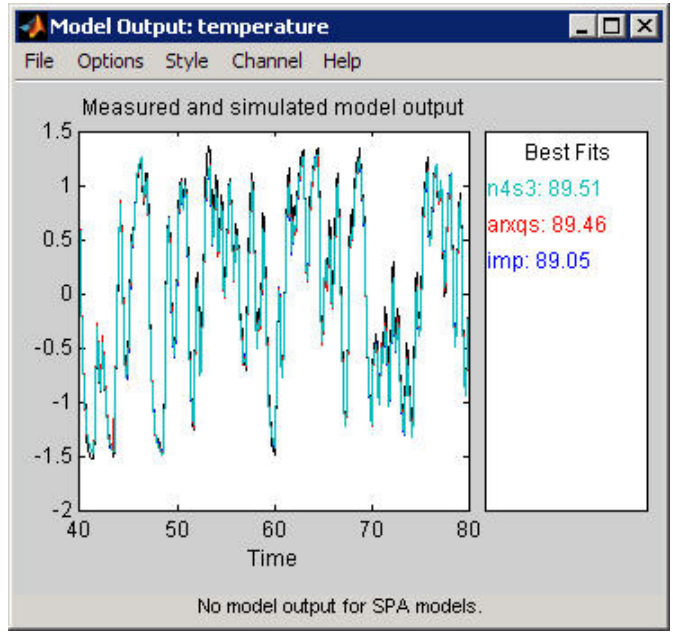
Frequency Response for Models spad, arxqs, and n4s3

Note The frequency-response plot does not include the impulse-response model, `imp`.

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 Tool 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 Accurate Linear Models

In this section...

Strategy for Estimating Accurate Models

The linear models you estimated in "Estimating Linear Models Using Quick Start" on page 4-23 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.

Tip You can convert a linear discrete-time model to a continuous-time model using the `d2c` command. For more information, see the corresponding reference page.

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.

About ARX Models

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

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

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 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 $a_1 \ldots a_n$ and $b_1 \ldots b_n$ from the data.

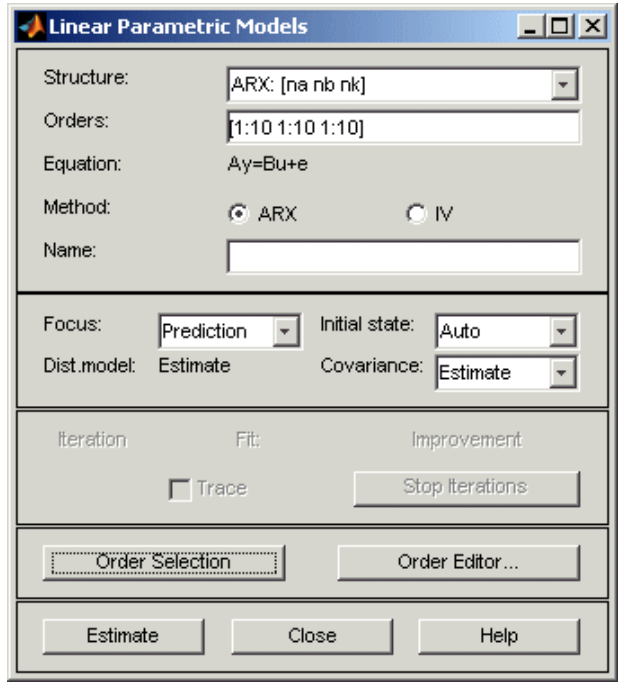
How to Estimate Model Orders

1. In the System Identification Tool GUI, select **Estimate > Linear parametric models** to open the Linear Parametric Models dialog box.

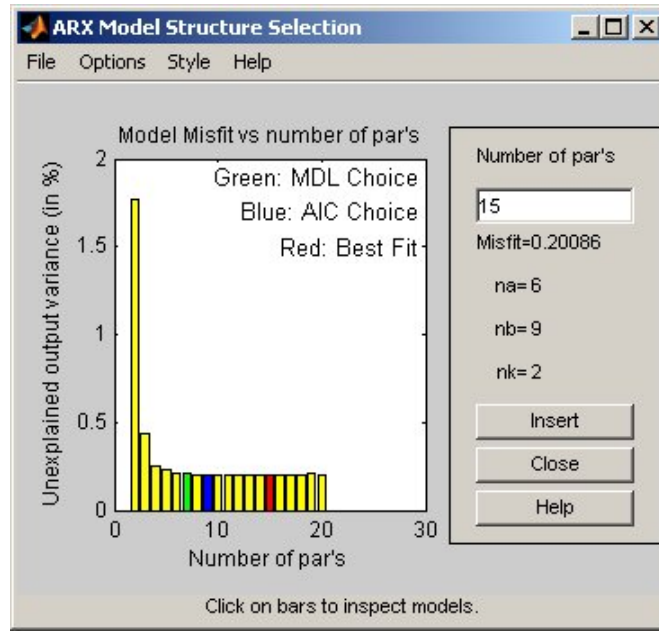
   The ARX model is already selected in the **Structure** list.

2 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]
```



3 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:

$$\text{Number of parameters} = n_a + n_b$$

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.

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.

---

4 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 Linear Parametric Models dialog box and performs the estimation.

This action adds the model `arx692` to the System Identification Tool GUI 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.

---

5 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 Linear Parametric Models dialog box and performs the estimation.
- The model `arx223` is added to the System Identification Tool GUI and the plots are updated to include its response and output.

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

Identifying State-Space Models

By estimating ARX models for different order combinations, as described in "Estimating Possible Model Orders" on page 4-30, 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:

$$x(t+1) = Ax(t) + Bu(t) + Ke(t)$$
$$y(t) = Cx(t) + Du(t) + e(t)$$

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* from the data.

The state-space model structure is a good choice for quick estimation because it requires that you specify only two parameters to get started: `n` is the number of poles (the size of the *A* matrix) and `nk` is the delay.

How to Estimate State-Space Models

1. In the System Identification Tool GUI, select **Estimate > Linear parametric models** to open the Linear Parametric Models dialog box.
2. From the **Structure** list, select `State Space:  n [nk]`.
3. In the **Orders** field, 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.

   ---

4. Click **Estimate** to add a state-space model called `n4s6` to the System Identification Tool GUI.

   Tip If you closed the Model Output window, you can regenerate it by selecting the **Model output** check box in the System Identification Tool GUI. If the new model does not appear on the plot, click the model icon in the System Identification Tool GUI to make the model active.

Learn More

To learn more about identifying state-space models, see the corresponding topic in the *System Identification Toolbox User's Guide*.

Identifying ARMAX Input-Output Polynomial Models

By estimating ARX models for different order combinations, as described in "Estimating Possible Model Orders" on page 4-30, 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:

$$\begin{aligned} y(t) + a_1 y(t-1) + \ldots + a_{na} y(t-n_a) = \\ \mathrm{b}_1 u(t-n_k) + \ldots + b_{nb} u(t-n_k-n_b+1) + \\ e(t) + c_1 e(t-1) + \ldots + c_{nc} e(t-n_c) \end{aligned}$$

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 $a_1 \ldots a_n$, $b_1 \ldots b_n$, and $c_1 \ldots c_n$ from the data.

How to Estimate ARMAX Models

1. In the System Identification Tool GUI, select **Estimate > Linear parametric models** to open the Linear Parametric Models dialog box.
2. From the **Structure** list, select `ARMAX: [na nb nc nk]` to estimate an ARMAX model.

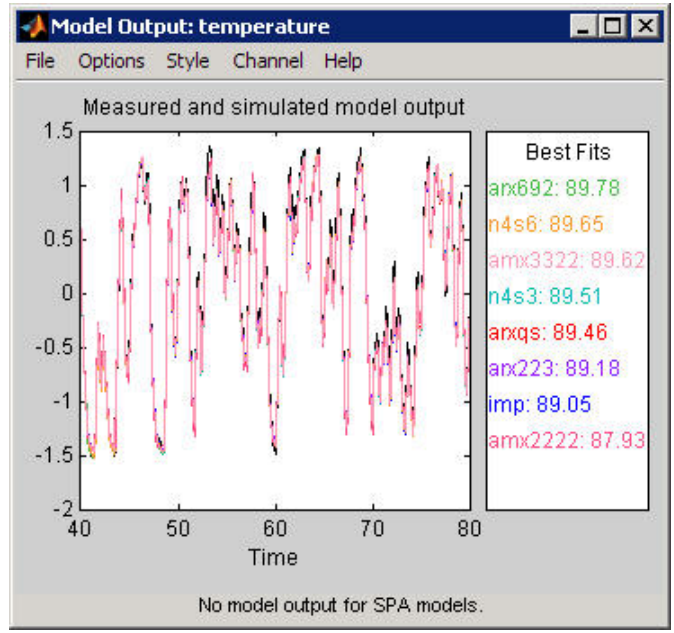
3 In the **Orders** field, set the orders *na*, *nb*, *nc*, and *nk* to the following values:

```
2 2 2 2
```

The model name in the **Name** field is `amx2222`, by default.

4 Click **Estimate** to add the ARMAX model to the System Identification Tool GUI.

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`.



Learn More

To learn more about identifying input-output polynomial models, such as ARMAX, see the corresponding topic in the *System Identification Toolbox User's Guide*.

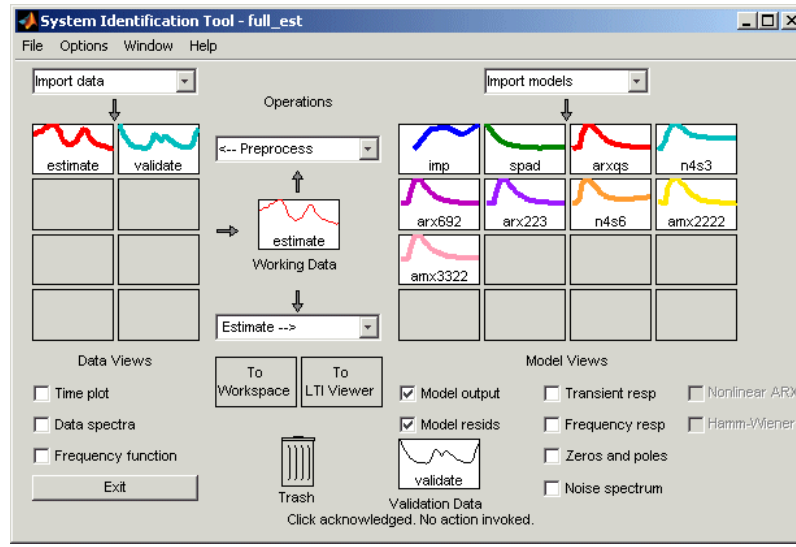
Choosing the Best Model

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

You must have already estimated the models, as described in “Estimating Accurate Linear Models” on page 4-30.

Summary of Models

The following figure shows the System Identification Tool GUI, which includes all of the estimated models in “Estimating Accurate Linear Models” on page 4-30.



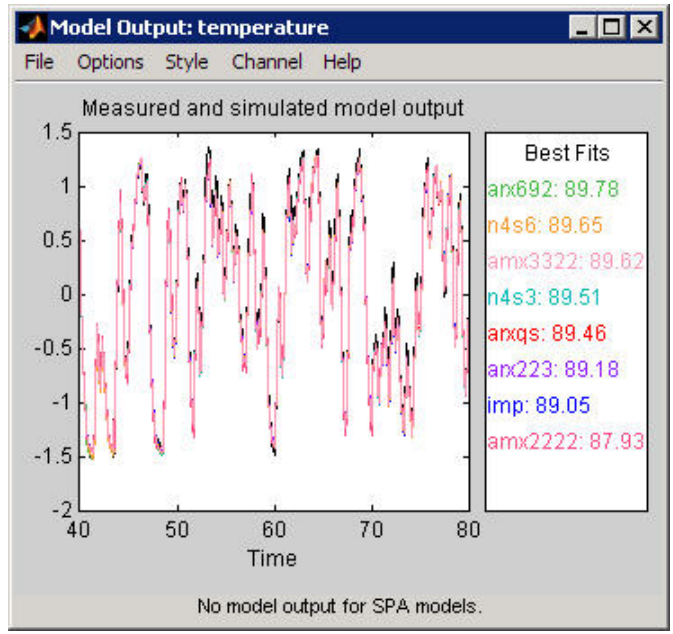
Examining the Model Output

If you closed the Model Output plot, you can regenerate it by selecting the **Model output** check box in the System Identification Tool GUI. If the model does not appear on the plot, click the model icon in the System Identification Tool GUI to display it on plots.

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

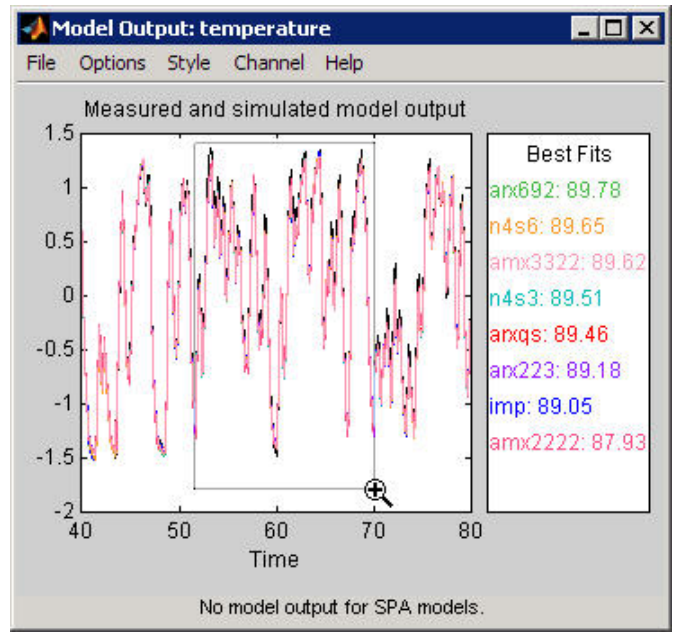
model that best describes the dynamics and successfully simulates or predicts the output for different inputs.

Models are listed by name in the **Best Fits** area of the Model Output plot. The highest-order model you created, `arx692`, fits the data as well as the simpler model `amx3322`.

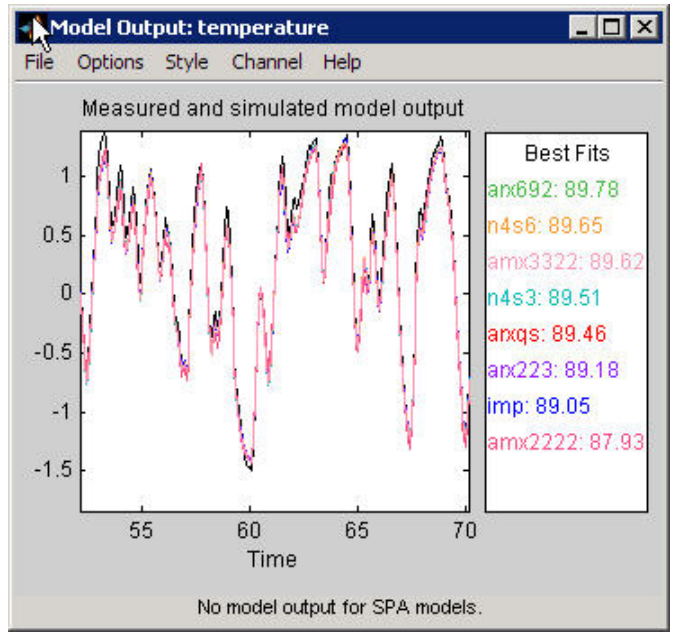


Tip To validate your models using a different data set, you can drag and drop this data set into the **Validation Data** rectangle in the System Identification Tool GUI. If you transform validation data into the frequency domain, the model-output plot updates to show the model comparison in the frequency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

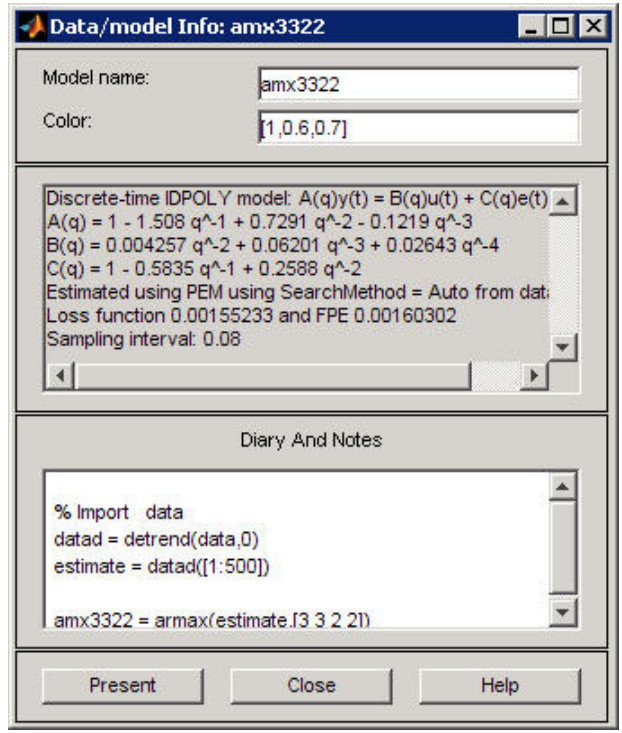
In this section...

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 Accurate Linear Models" on page 4-30.

To view the parameter values of the model `amx3322`, right-click the model icon in the System Identification Tool GUI. The Data/model Info dialog box opens.



The noneditable area of the Data/model Info dialog box lists the following parameter values:

```
A(q) = 1 - 1.508q^-1 + 0.7291q^-2 - 0.1219q^-3
B(q) = 0.004257q^-2 + 0.06201q^-3 + 0.02643q^-4
C(q) = 1 - 0.5835q^-1 + 0.2588q^-2
```

These parameter values correspond to the following difference equation for your system:

$$y(t) - 1.508y(t-1) + 0.7291y(t-2) - 0.1219y(t-3) = 0.004257u(t-2) + 0.06201u(t-3) + 0.02643u(t-4) + e(t) - 0.5835e(t-1) + 0.2588e(t-2)$$

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:

$$A(q) = 1 + a_1 q^{-1} + \ldots + a_{na} q^{-na}$$
$$B(q) = b_1 q^{-nk} + \ldots + b_{nb} q^{-nb-nk+1}$$
$$C(q) = 1 + c_1 q^{-1} + \ldots + c_{nc} q^{-nc}$$

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 Accurate Linear Models" on page 4-30.

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

```
A(q) = 1 - 1.508(+-0.05919)q^-1
       + 0.7291(+-0.08734)q^-2
       - 0.1219 (+-0.03424)q^-3
B(q) = 0.004257(+-0.001563)q^-2
       + 0.06201(+-0.002409)q^-3
       + 0.02643(+-0.005633)q^-4
C(q) = 1 - 0.5835(+-0.07189)q^-1
         + 0.2588(+-0.05253)q^-2
```

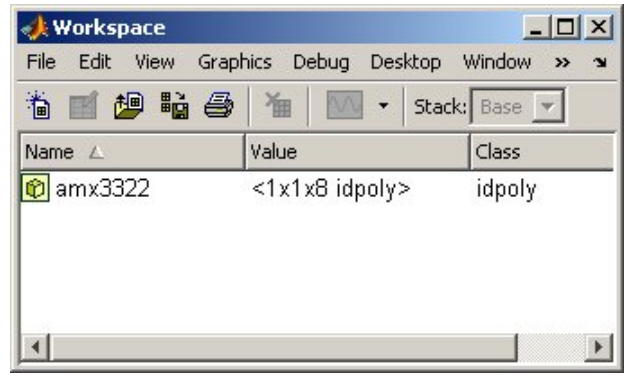
The 1-standard-deviation uncertainty for the model parameters is in parentheses next to each parameter value.

Exporting the Model to the MATLAB Workspace

The models you create in the System Identification Tool GUI are not automatically available in the MATLAB workspace. To make a model available to other toolboxes, the Simulink software, and the System Identification Toolbox commands, you must export your model from the System Identification Tool 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 Accurate Linear Models" on page 4-30.

To export the `amx3322` model, drag it to the **To Workspace** rectangle in the System Identification Tool GUI. The model appears in the MATLAB Workspace browser.



Note This model is an object that belongs to the `idpoly` model class. To learn more about this model object, see the corresponding reference page.

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 LTI object using:

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

You can extract the dynamic model and ignore the noise model using the following command:

```
ss_model=ss_model('m')
```

Exporting the Model to the LTI Viewer

If you have the Control System Toolbox product installed on your computer, the **To LTI Viewer** rectangle appears in the System Identification Tool GUI.

The LTI Viewer 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 LTI Viewer, drag and drop the model icon to the **To LTI Viewer** rectangle in the System Identification Tool GUI.

For more information about working with plots in the LTI Viewer, see the Control System Toolbox documentation.

5

Tutorial – Identifying Low-Order Transfer Functions (Process Models) Using the GUI

About This Tutorial

In this section...

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 Tool GUI:

- Import data objects from the MATLAB workspace into the GUI.
- 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 sampling interval 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

In this section...

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
```

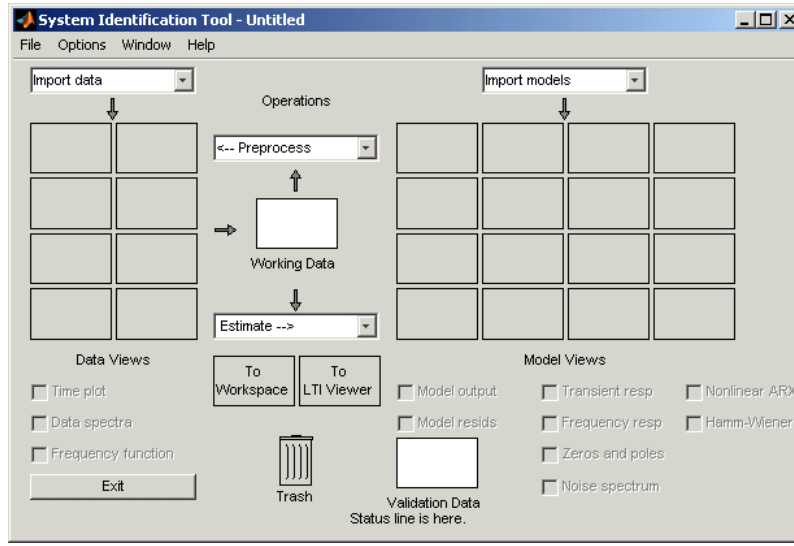
This command loads the data into the MATLAB workspace as the data object `z`. For more information about `iddata` objects, see the corresponding reference page.

Opening the System Identification Tool GUI

To open the System Identification Tool GUI, type the following command MATLAB Command Window:

```
ident
```

The default session name, `Untitled`, appears in the title bar.



Importing Data Objects into the System Identification Tool

You can import data object into the GUI 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 5-5, and opened the GUI, as described in “Opening the System Identification Tool GUI” on page 5-5.

To import a data object into the System Identification Tool GUI:

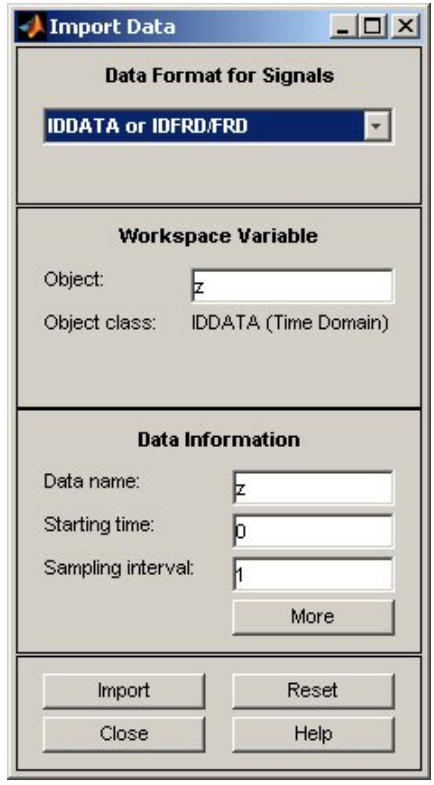
1 In the System Identification Tool GUI, select **Import data > Data object**.

This action opens the Import Data dialog box.

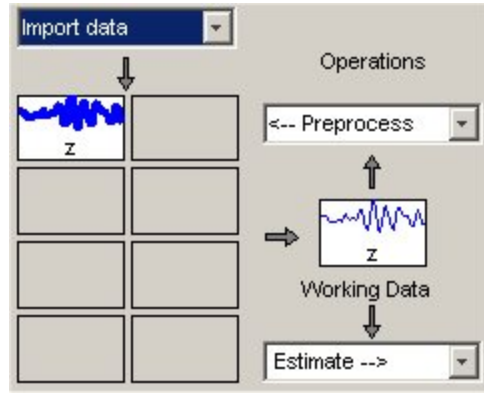
2 Specify the following options:

- **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 Tool GUI 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.
- **Sampling interval** — Enter `1` as the time between successive samples in seconds. This value represents the actual sampling interval in the experiment.

The Import Data dialog box now resembles the following figure.


Import Data
Data Format for Signals
IDDATA or IDFRD/FRD
Workspace Variable
Object:
z
Object class:
IDDATA (Time Domain)
Data Information
Data name:
z
Starting time:
0
Sampling interval:
1
More
Import
Reset
Close
Help

3 Click **Import** to add the icon named z to the System Identification Tool GUI.



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:

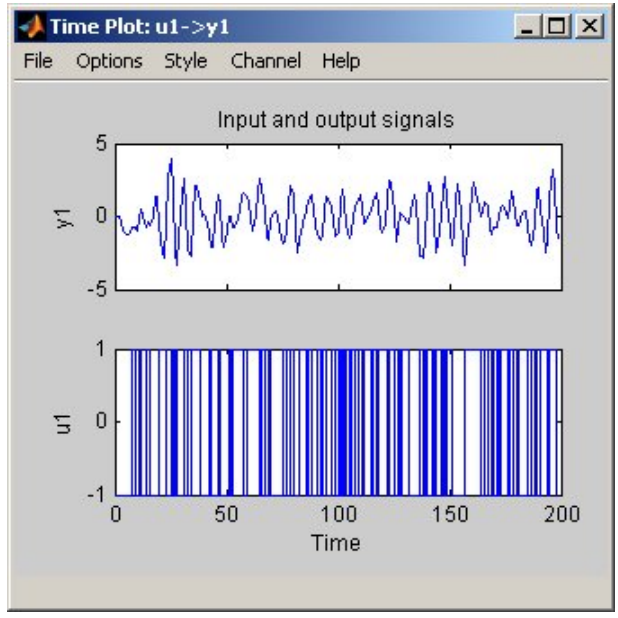
- Plot the data.
- Subtract the mean values of the input and the output to remove offsets.
- Split the data into two parts. You use one part of the data for model estimation, and the other part of the data for model validation.

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 Tool, as described in "Importing Data Objects into the System Identification Tool" on page 5-6.

To plot and process the data:

1 In the System Identification Tool GUI, 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.

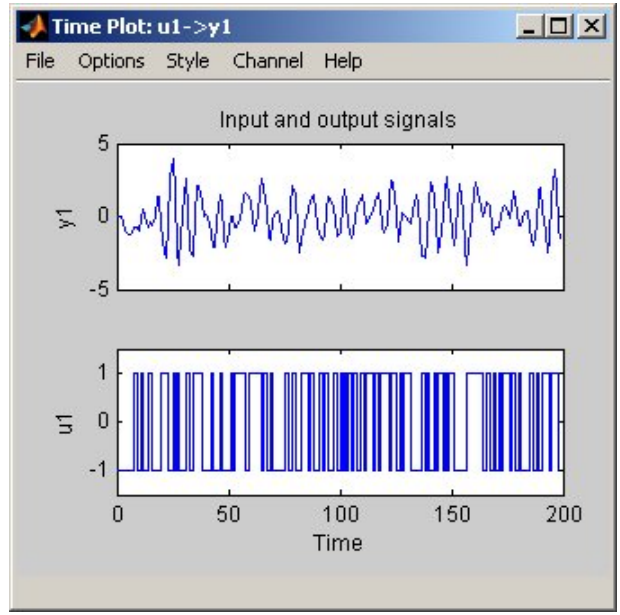
The next two steps demonstrate how to modify the axis limits in the plot.

2 To modify the vertical-axis limits for the input data, select **Options > Set axes limits**.

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 Tool GUI, 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 the *System Identification Toolbox User's Guide*.

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

In this section...

Estimating a Second-Order Transfer Function Using Default Settings

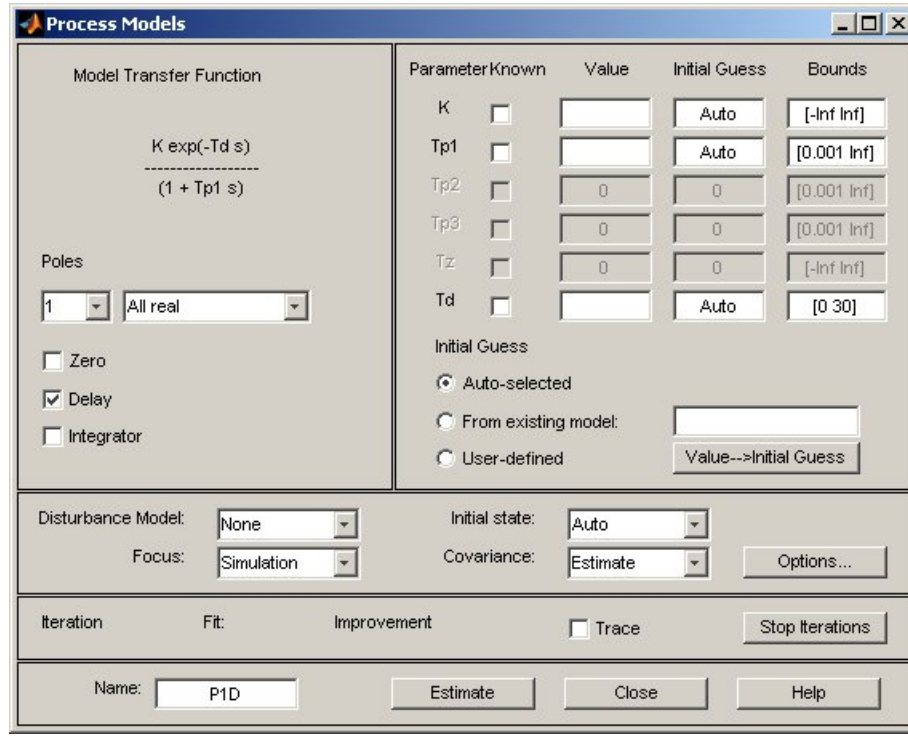
In this portion of the tutorial, you estimate models with this structure:

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

You must have already processed the data for estimation, as described in "Plotting and Processing Data" on page 5-9.

To identify a second-order transfer function:

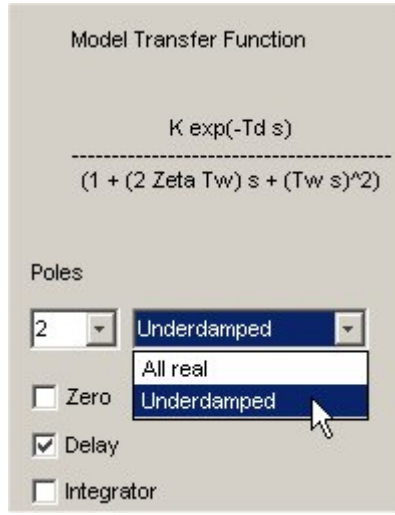
1 In the System Identification Tool GUI, select **Estimate > Process models** to open the Process Models dialog box.



2 In the **Model Transfer Function** area, specify the following options:

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

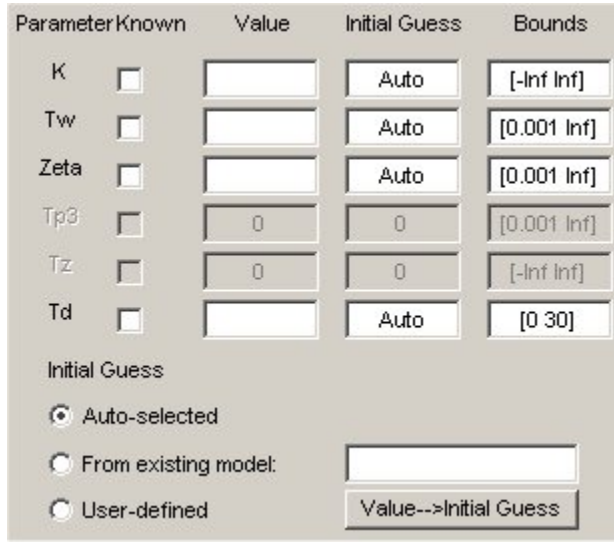
  This selection updates the Model Transfer Function to a second-order model structure that can contain complex poles.



  The **Parameter** area now shows four active parameters: `K`, `Tw`, `Zeta`, and `Td`. By default, the model **Name** (located at the bottom of the dialog box) is set to the acronym `P2DU`, which indicates two poles (`P2`), a delay (`D`), and underdamped modes (`U`).

- Make sure that the **Zero** and **Integrator** check boxes are cleared to exclude a zero and an integrator (self-regulating) from the model.

3 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.

---

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 state** — `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.

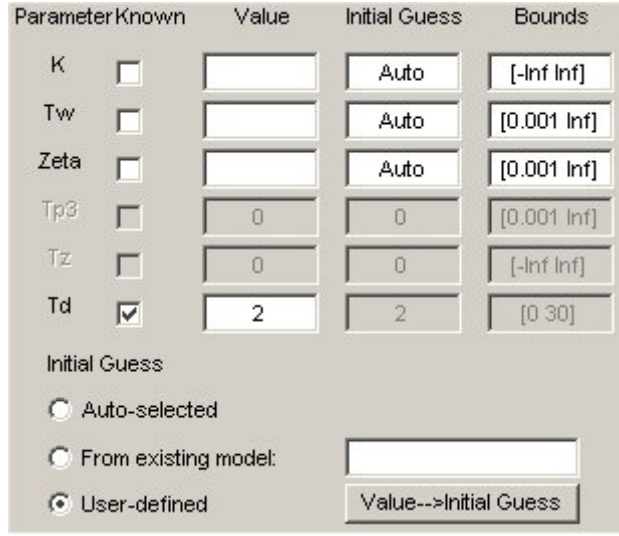
6 Click **Estimate**. This selection adds the model `P2DU` to the System Identification Tool GUI.

Tips for Specifying Known Parameters

If you know a parameter value exactly, you can type this value in the **Initial Guess** column of the Process Models dialog box.

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 `2`s, you can type this value into **Value** field of the Parameter table in the Process Models dialog box and 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 5-13.

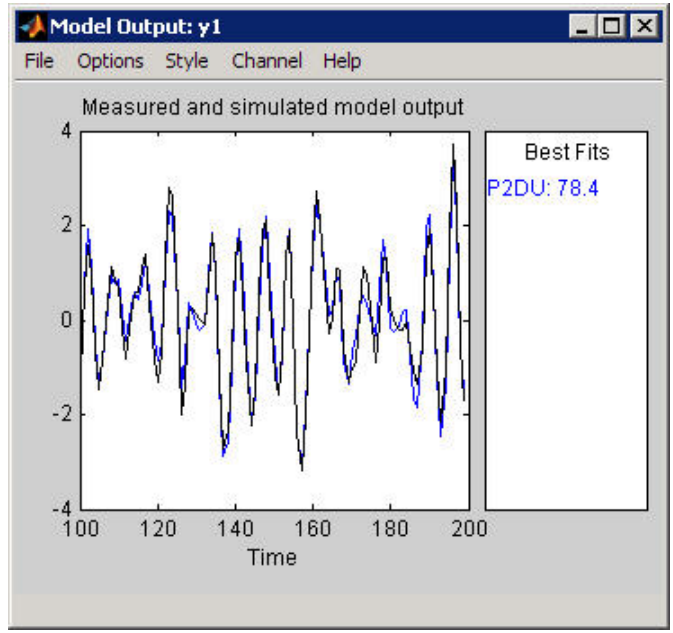
Examining Model Output

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 Tool GUI.

  If the plot is empty, click the model icon in the System Identification Tool window to display the model on the plot.



The System Identification Toolbox product 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 5-3, and has a peak response at 1 rad/s:

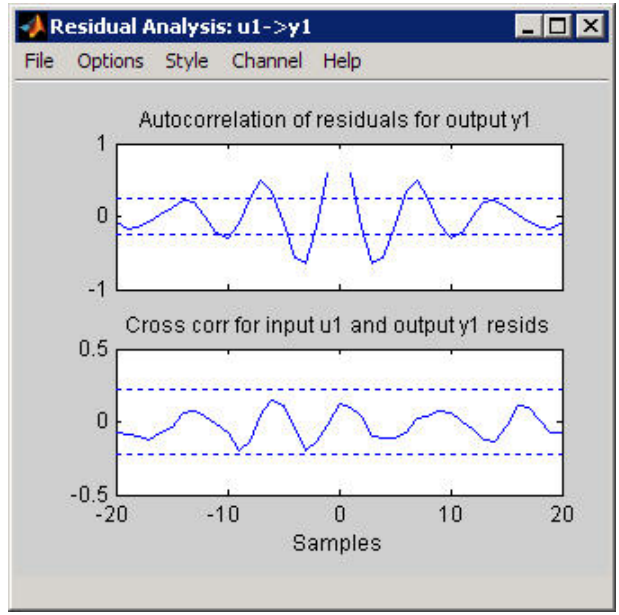
$$G(s) = \frac{1}{1 + 0.2s + s^2} e^{-2s}$$

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 Tool GUI.



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 Transfer Function with a Noise Model

In this section...

Estimating a Second-Order Transfer Function with Complex Poles and Noise

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 5-13.

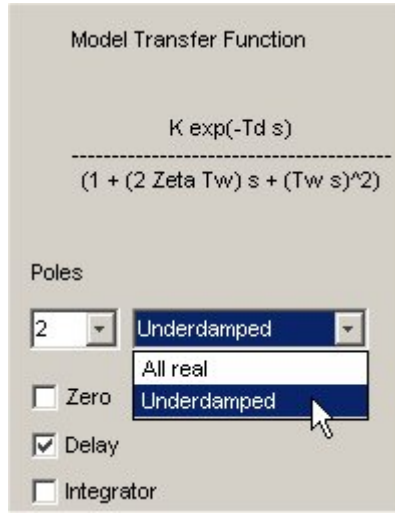
To estimate a second-order transfer function with noise:

1 If the Process Models dialog box is not open, select **Estimate > Process models** in the System Identification Tool GUI. 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 Tool GUI.

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 Tool GUI.

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 5-13 and "Estimating a Second-Order Transfer Function with Complex Poles and Noise" on page 5-22.

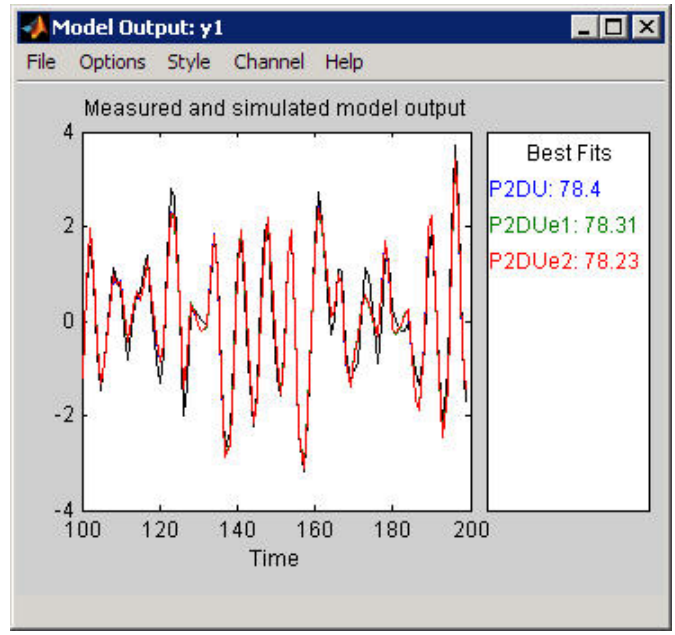
Comparing the Model Output Plots

To generate the Model Output plot:

- Select the **Model output** check box in the System Identification Tool GUI.

If the plot is empty or a model output does not appear on the plot, click the model icons in the System Identification Tool window to display these models on the plot.

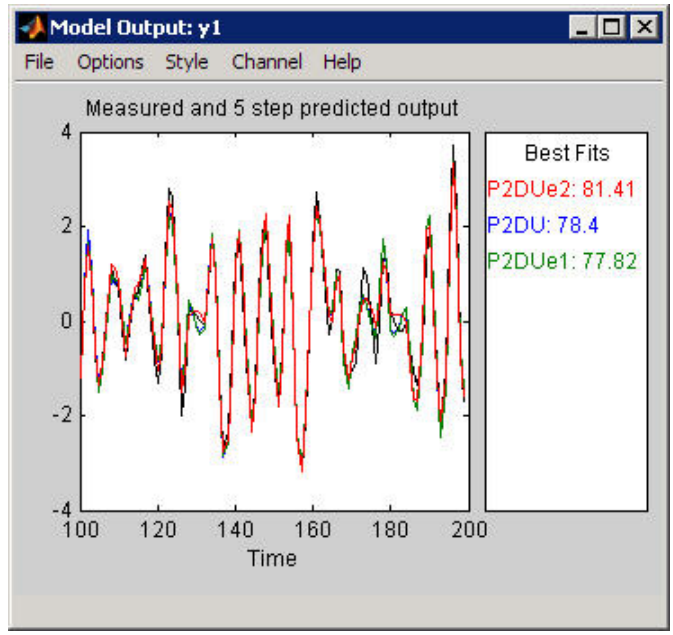
The following Model Output plot shows the simulated model output, by default. The simulated response of the models is approximately the same for models with and without noise. Thus, including the noise model does not affect the simulated output.



To view the predicted model output:

- In the Model Output plot window, select **Options > 5 step ahead predicted output**.

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).



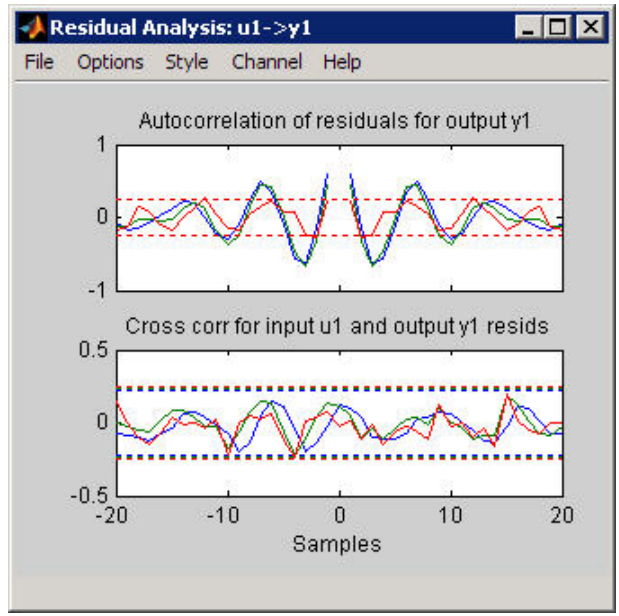
Comparing the Residual Analysis Plots

To generate the Residual Analysis plot:

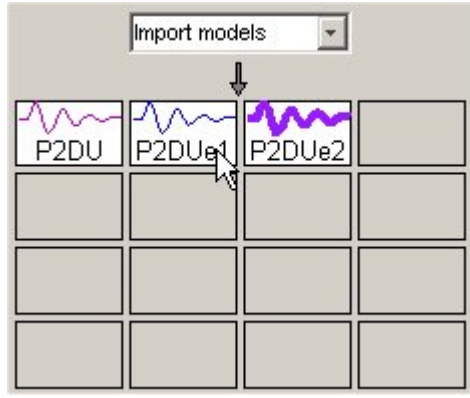
- Select the **Model resids** check box in the System Identification Tool GUI.

  If the plot is empty, click the model icons in the System Identification Tool window to display these models on the plot.

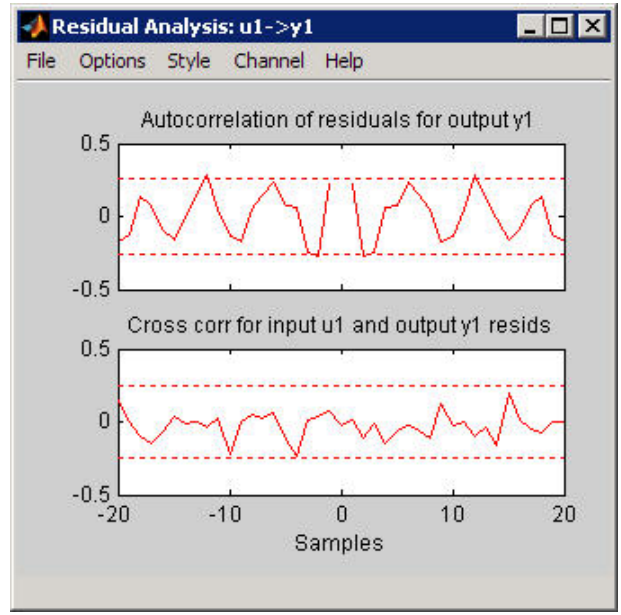
`P2DUe2` falls well within the confidence bounds on the Residual Analysis plot.



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 Tool GUI.



The Residual Analysis plot updates, as shown in the following figure.



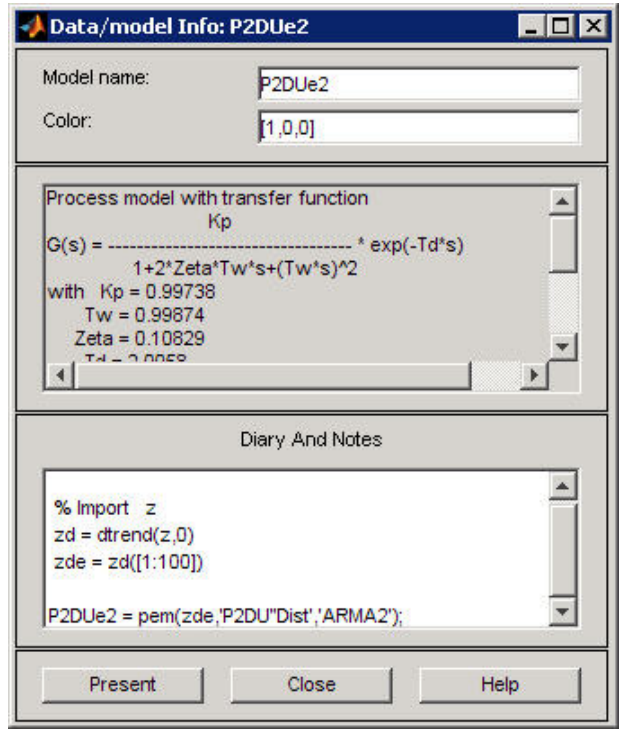
The whiteness test for `P2DUe2` shows that the residuals are uncorrelated, and the independence test shows no correlation between the residuals and the inputs. These tests indicate that `P2DUe2` is a good model.

Viewing Model Parameters

In this section...

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 Tool GUI. 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:

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

For the model `P2DUe2`:

- `Kp` is `0.99783`.
- `Tw` is `0.99874`.
- `Zeta` is `0.10829`.
- `Td` is `2.0058`.

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

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

`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 Data/model Info dialog box gives the noise model H as a ratio of two polynomials, `C(s)/D(s)`, where:

```
C(s) = s^2 + 2.14(+-0.08596)s + 1.175(+-0.3083)
D(s) = s^2 + 0.2427(+-0.08213)s + 0.61(+-0.07577)
```

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

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 = 0.99738+-0.01951
```

```
Tw = 0.99874+-0.0037075
Zeta = 0.10829+-0.0040998
Td = 2.0058+-0.002329
```

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

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 Tool GUI 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 Tool to the MATLAB workspace.

To export the `P2DUe2` model:

- In the System Identification Tool GUI, drag the model icon to the **To Workspace** rectangle. The model now appears in the MATLAB Workspace browser.

Note This model is an object of `idproc` class. Model objects encapsulate all properties of the model. To learn more about this model object, see the corresponding reference page.

Simulating a System Identification Toolbox Model in Simulink Software

In this section...

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 5-5.
- Estimate the second-order process model, as described in "Estimating a Second-Order Transfer Function with Complex Poles and Noise" on page 5-22.
- Export the model to the MATLAB workspace, as described in "Exporting the Model to the MATLAB Workspace" on page 5-33.

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 the *System Identification Toolbox User's Guide*.

Building the Simulink Model

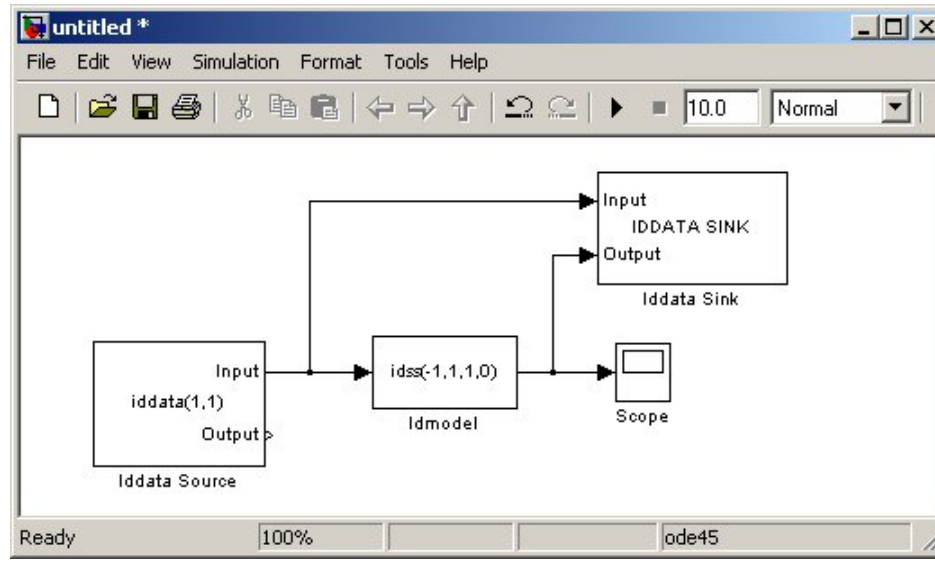
To add blocks to a Simulink model:

1. In the MATLAB Command Window, type `simulink`.
2. Select **File > New > Model** to open a new model window.
3. In the Simulink Library Browser, select the **System Identification Toolbox** library. The right side of the window displays blocks specific to the System Identification Toolbox product.

   Tip Another way to access the System Identification block library is to 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 until your model resembles the following figure.



In the next section, you configure these blocks to get data from the MATLAB workspace and set the simulation time interval and duration.

Learn More

For more information about working with Simulink models, see the Simulink documentation.

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 new model window, select **Simulation > Configuration Parameters**.

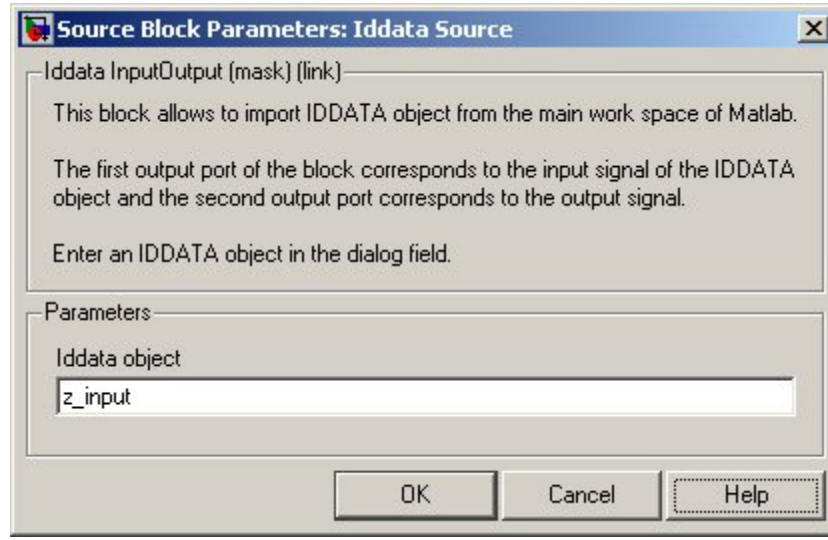
2 In the Configuration Parameters dialog box, type `200` in the **Stop time** field. Click **OK**.

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.

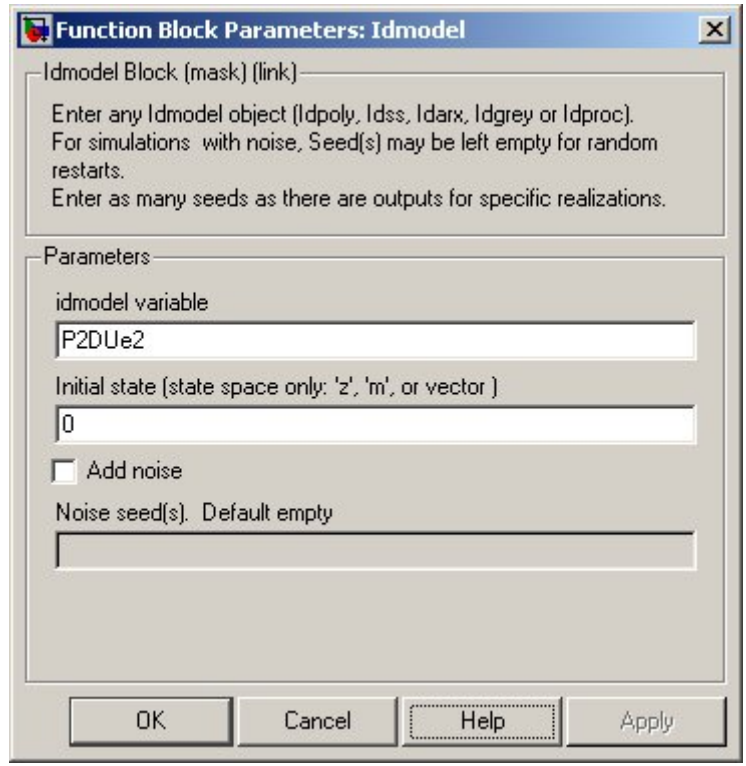
---

4 Click **OK**.

5 Double-click the Idmodel block to open the Function Block Parameters: Idmodel dialog box. Type the following variable name in the **idmodel variable** field:

```
P2DUe2
```

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



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

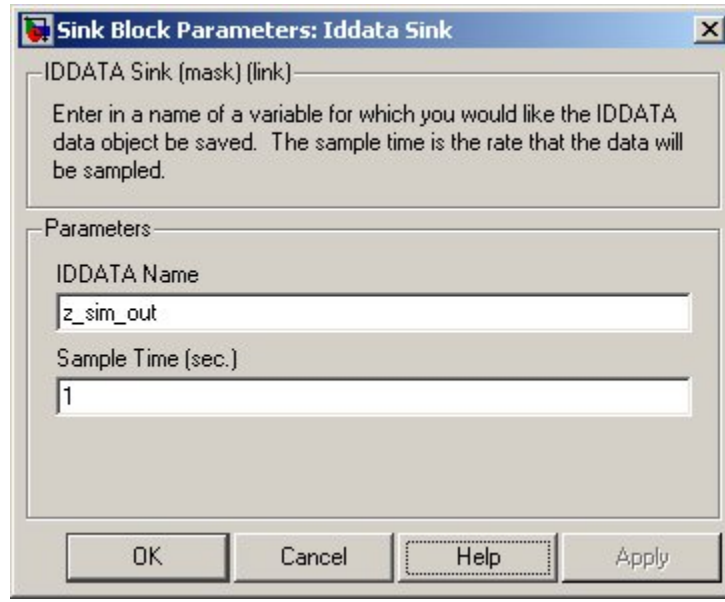
When **Add noise** is selected, Simulink derives the noise amplitude from the `NoiseVariance` property of the model and adds noise to the model accordingly. The simulation propagates this noise according to the noise model H that was estimated with the system dynamics:

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

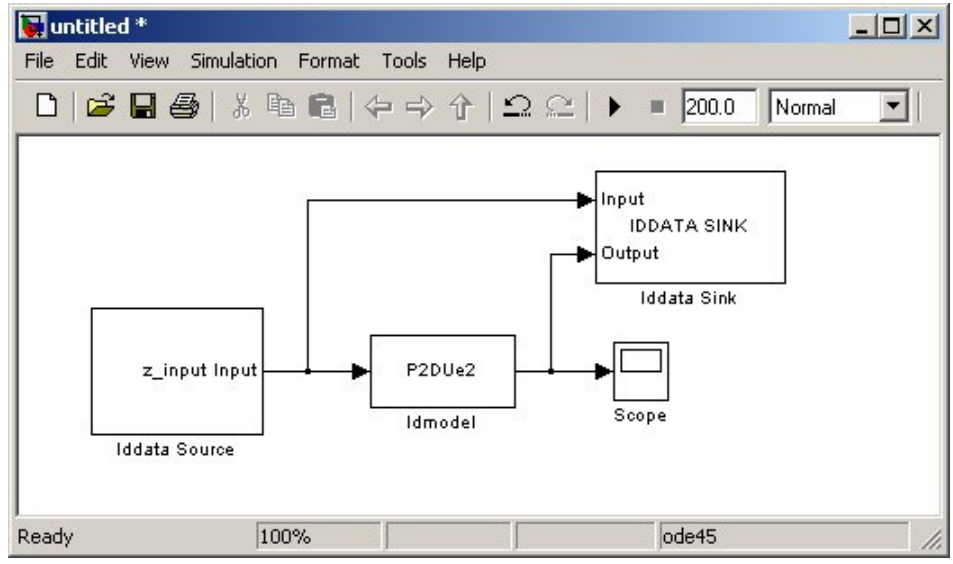
7 Double-click the Iddata Sink block to open the Sink Block Parameters: Iddata Sink dialog box. Type the following variable name in the **IDDATA Name** field:

```
z_sim_out
```

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

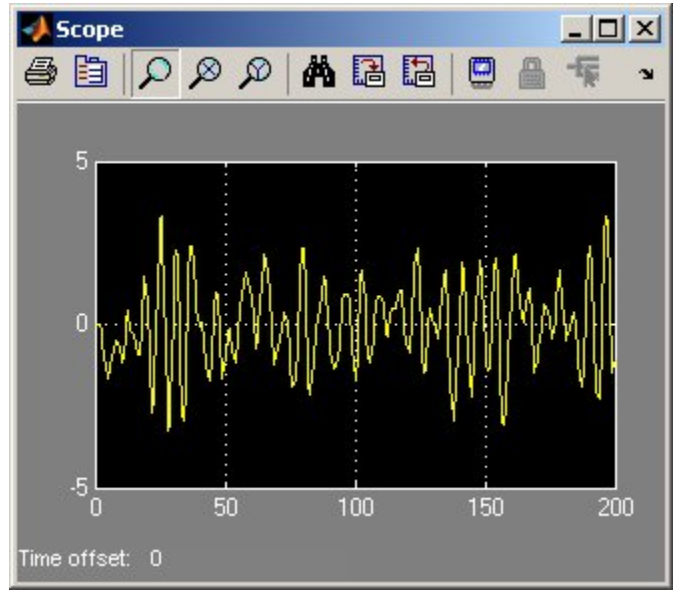


The resulting change to the Simulink model is shown in the following figure.



Running the Simulation

1. In the Simulink model window, select **Simulation > Start**.
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.

6

Tutorial – Identifying Linear Models Using the Command Line

About This Tutorial

In this section...

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:

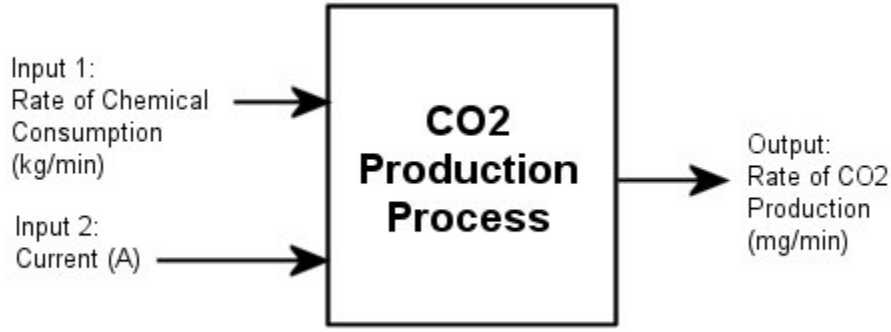
- Create data objects to represent data.
- Plot the data.
- Process data by removing offsets from the input and output signals.
- Estimate and validate linear models from the data.
- Simulate and predict model output.

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.


Input 1:
Rate of Chemical
Consumption
(kg/min)
Input 2:
Current (A)
CO2
Production
Process
Output:
Rate of CO2
Production
(mg/min)

Preparing Data

In this section...

Loading Data into the MATLAB Workspace

Load the data in `co2data.mat` by typing the following in the MATLAB Command Window:

```
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.

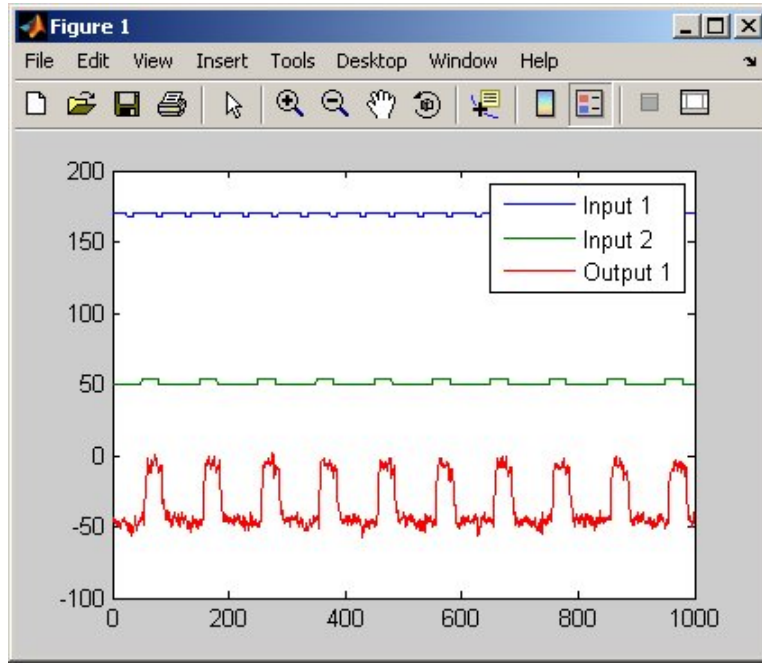
For both experiments, the input data consists of two columns of values. The first column of values is the rate of chemical consumption (in kilograms per minute), and the second column of values is the current (in amperes). The output data is a single column of the rate of carbon-dioxide production (in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

Plotting the Input/Output Data

You can plot the input and output data from both experiments using the following commands:

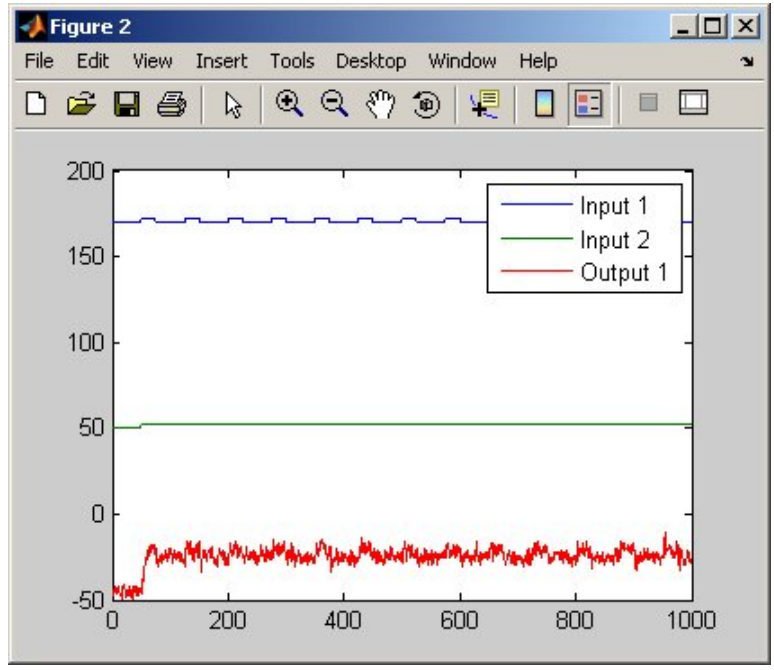
```
% Plot the input and output data from both experiments
plot(Time,Input_exp1,Time,Output_exp1)
legend('Input 1','Input 2','Output 1')
figure
plot(Time,Input_exp2,Time,Output_exp2)
legend('Input 1','Input 2','Output 1')
```

The resulting plot shows the first experiment, where the operator applies a pulse wave to each input to perturb it from its steady-state equilibrium.



Input and Output Data from Experiment 1

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



Input and Output Data from Experiment 2

Removing Equilibrium Values from the Data

Plotting the data, as described in "Plotting the Input/Output Data" on page 6-5, 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 the inputs and the outputs in both experiments:

```
% 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,:));
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,:));
```

Note The `ones` command replicates the two mean values, one for each input, in a two-dimensional array.

Using Objects to Represent Data for System Identification

The System Identification Toolbox data objects, `iddata` and `idfrd`, 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.

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 7-9.

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

```
% Create two data objects, ze and zv.
Ts = 0.5; % Sampling interval 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 sampling interval.

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)
```

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:

```
% 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, by typing the following syntax:

```
ze.inputname
```

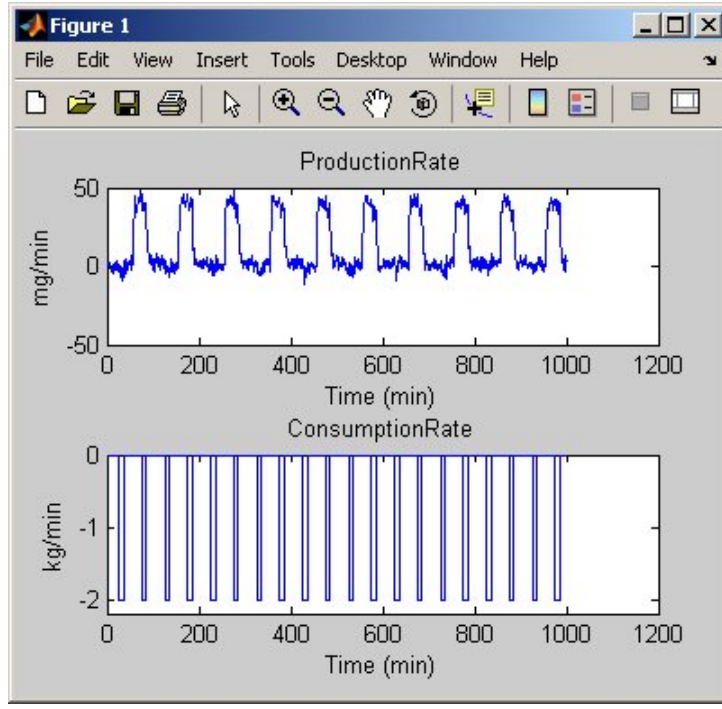
Tip Property names, such as `InputUnit`, are not case sensitive. You can also abbreviate property names that start with `Input` or `Output` by substituting `u` for `Input` and `y` for `Output` in the property name. For example, `OutputUnit` is equivalent to `yunit`.

Plotting the Data in a Data Object

You can plot `iddata` objects using the MATLAB `plot` command:

```
plot(ze)     % Plot the estimation data
```

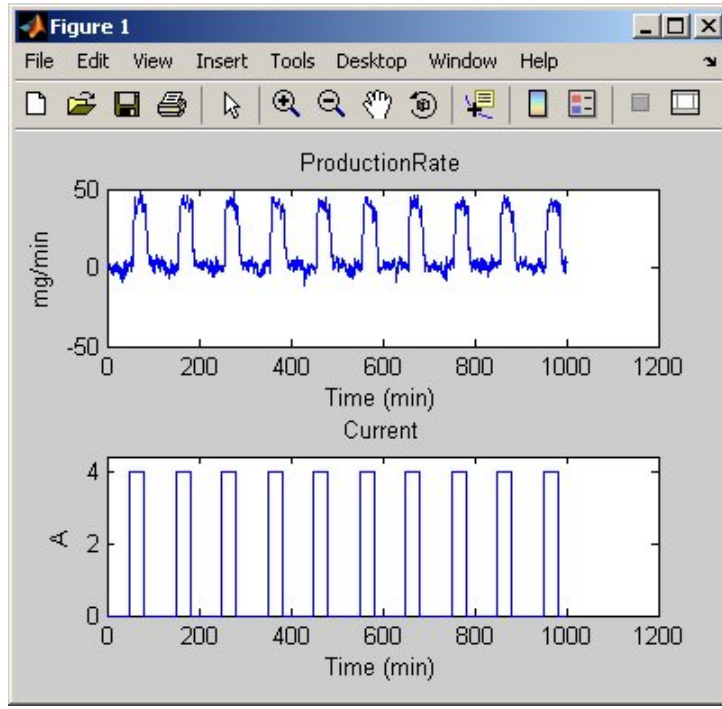
This opens the following plot. The bottom axes show the first input `ConsumptionRate`, and the top axes show the output `ProductionRate`.



Input 1 and Output for ze

For multivariable data, only one input/output pair appears on the plot at a time. To view the second input `Current`, select the MATLAB Figure window, and press **Enter** to update the plot.

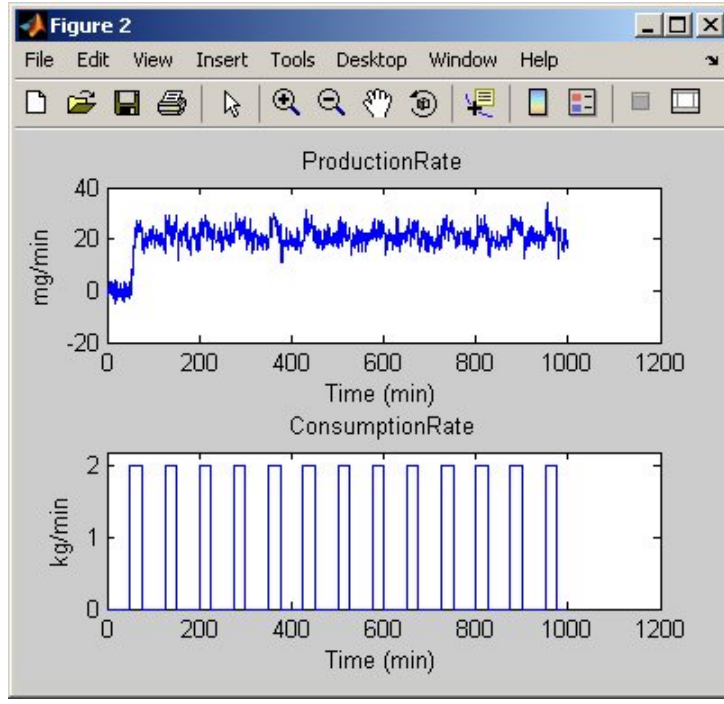
Tip For plots of data with multiple inputs and outputs, press **Enter** to view the next input/output pair.



Input 2 and Output for ze

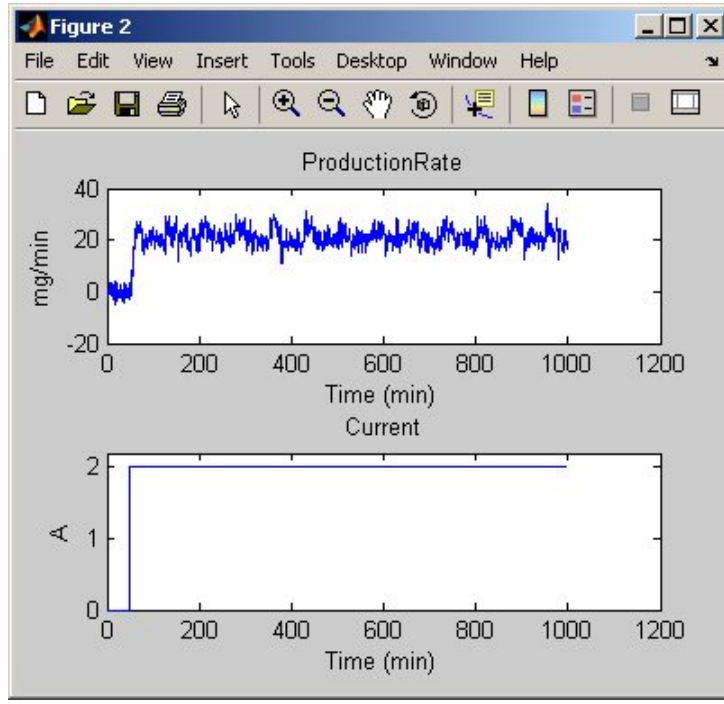
To plot the validation data in a new MATLAB Figure window, type the following commands in the MATLAB Command Window:

```
figure   % Open a new MATLAB Figure window
plot(zv) % Plot the validation data
```



Input 1 and Output for zv

Select the MATLAB Figure window, and press **Enter** to view the second input on the plot.



Input 2 and Output for zv

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 Step- and Frequency-Response Models

In this section...

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 6-8.

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.

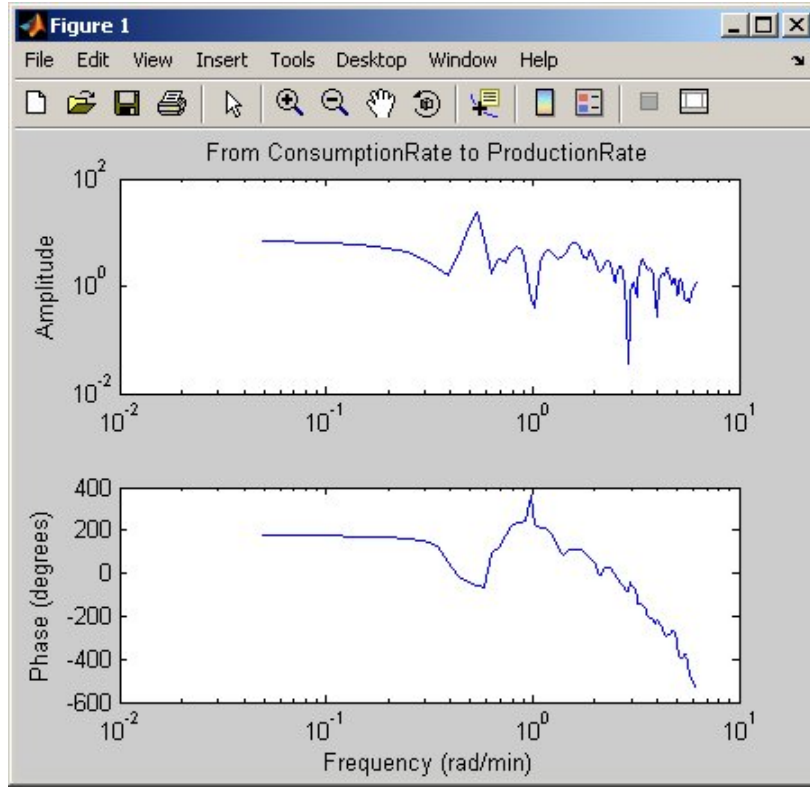
Use the spa command to estimate the frequency response:

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

To plot the frequency response as a Bode plot, type the following command in the MATLAB Command Window:

```
bode(Ge)
```

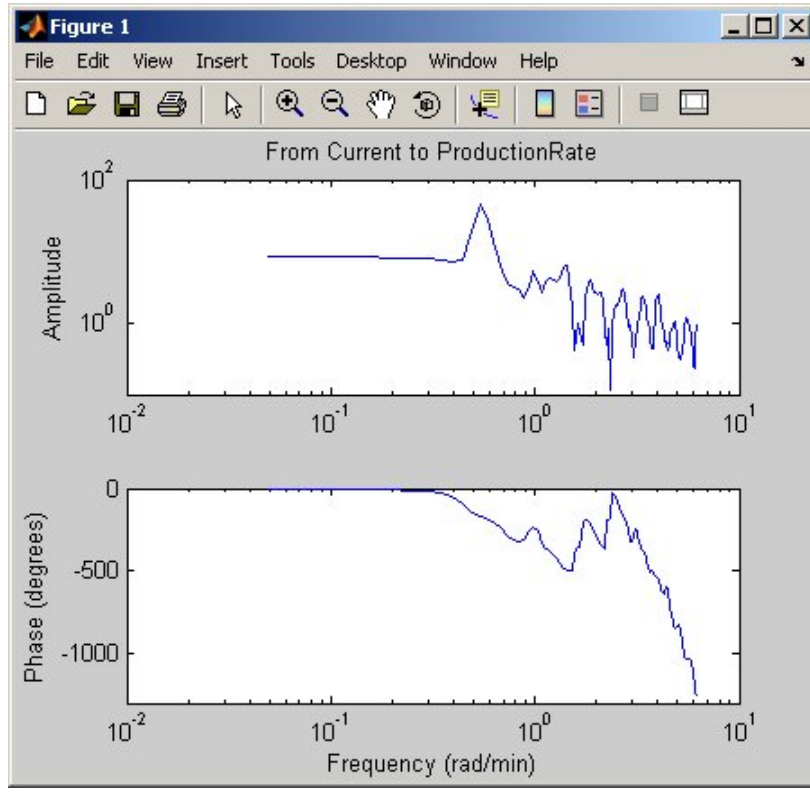
This command produces the following plot.



Frequency Response for the First Input-Output Path

The amplitude peaks at the frequency of about 0.7 rad/s, which suggests a possible resonant behavior (complex poles) for the first input-to-output combination—`ConsumptionRate` to `ProductionRate`.

To view the second input `Current`, select the MATLAB Figure window, and press **Enter**. The input/output pair is displayed, as shown in the following figure.



Frequency Response for the Second Input-Output Path

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

Tip When your data contains multiple inputs and outputs, press **Enter** to view the next input/output pair.

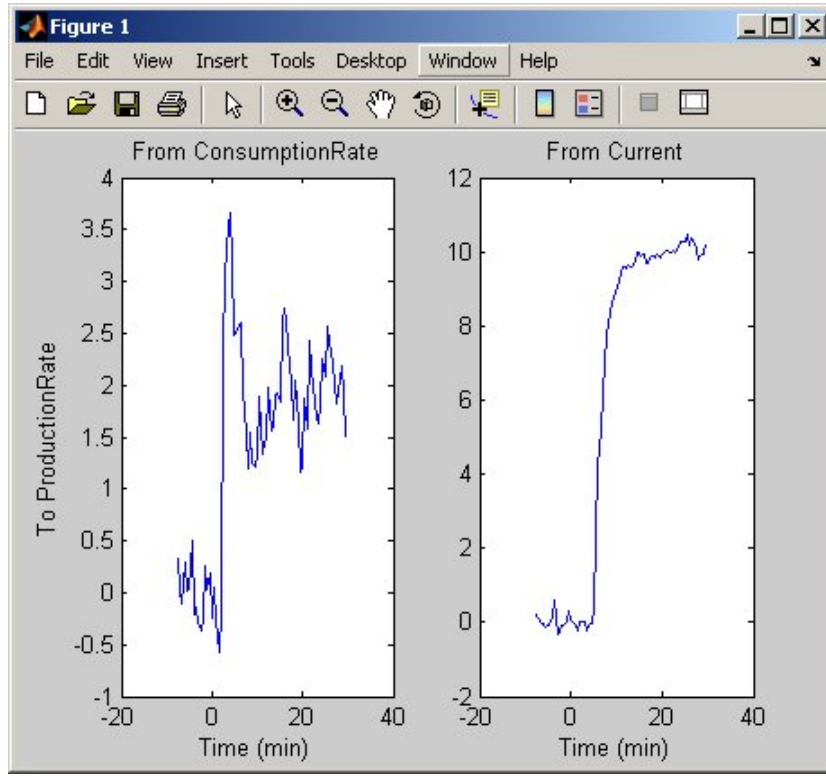
Estimating the Step Response

To estimate the step response from the data, use the `step` command with the following arguments:

```
step(ze,30)
```

The first `step` argument is the name of the data object. The second argument is the duration of the step input in the time units you specified (minutes).

This calculation produces the following plot.



Step Response from Both Inputs to the Output

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 Step Response” on page 6-18.

Estimating Delays in the Multiple-Input System

In this section...

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 `delayest` 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 the following command in the MATLAB Command Window:

```
delayest(ze)
```

System Identification Toolbox software responds with the following:

```
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 sampling interval 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

In this section...

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.

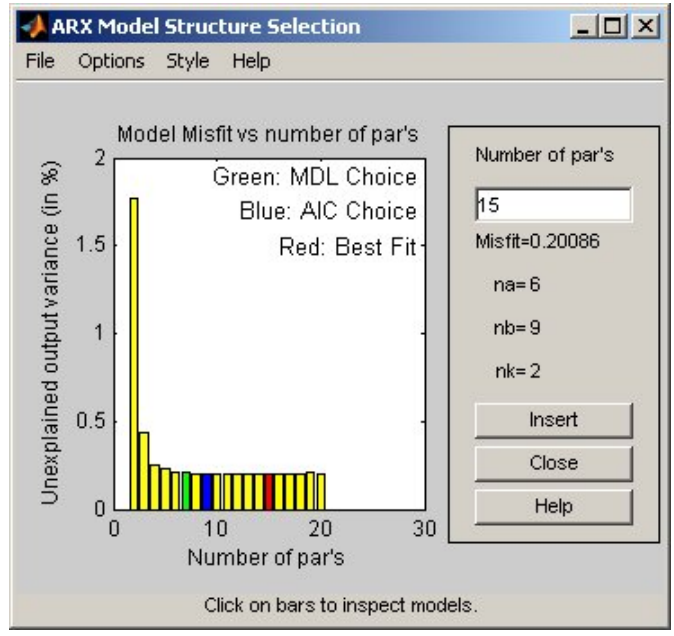
Commands for Estimating the Model Order

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 the preceding plot to select the best-fit model. The horizontal axis is the total number of parameters:

$$\text{Number of parameters} = n_a + n_b$$

For the ARX model, n_a is the number of poles, n_b is the number of b parameters (equal to the number of zeros plus 1), and n_k is the delay.

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.

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 Step- and Frequency-Response Models" on page 6-15 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 6-20, 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.

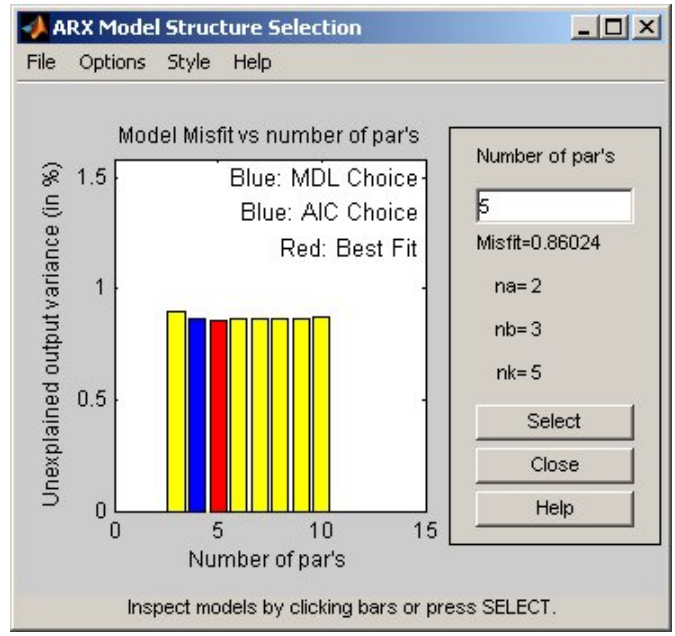
   ```
   NN1 = struc(2:5,1:5,5);
   ```

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 Step- and Frequency-Response Models" on page 6-15 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 6-20, the delay for this input/output combination was estimated to be 10.

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

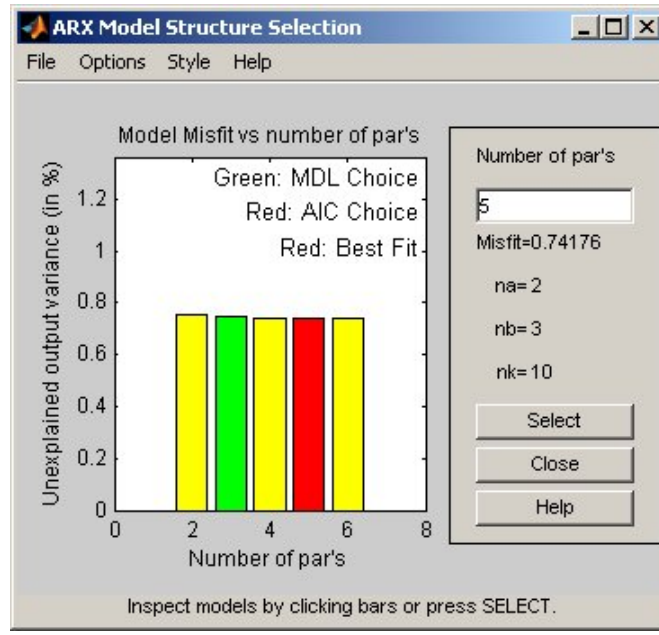
1 Use `struc` to create a matrix of possible model orders.

```
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 Continuous-Time Transfer Functions (Process Models)

In this section...

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 6-4.

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'});
```

The argument of `idproc` is a cell array that contains two strings, where each string 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`).

Viewing the Model Structure and Parameter Values

To view the two resulting models, type the following command in the MATLAB Command Window:

```
midproc0
```

MATLAB displays the following model structure:

```
Process model with 2 inputs: y = G_1(s)u_1 + G_2(s)u_2

where
                     1+Tz*s
G_1(s) = Kp * ---------------------- * exp(-Td*s)
              1+2*Zeta*Tw*s+(Tw*s)^2


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


            Kp
G_2(s) = ---------- * exp(-Td*s)
          1+Tp1*s

with   Kp = NaN
      Tp1 = NaN
       Td = NaN

This model was not estimated from data.
```

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.Td.value = [2.5 5];
midproc0.Td.max = [3 7];
```

Note When setting the `Td` model property, 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 pem

`pem` is an *iterative* estimation command, 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, `pem` estimates different black-box polynomial models. You can use `pem`, for example, to estimate parameters for linear continuous-time transfer-function, state-space, or polynomial model structures. To use `pem`, 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 6-31. Use `midproc0` as the model structure and `Ze1` as the estimation data:

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

MATLAB displays the following model structure with estimated parameter values:

```
Process model with 2 inputs: y = G_1(s)u_1 + G_2(s)u_2

where
                     1+Tz*s
G_1(s) = Kp * ---------------------- * exp(-Td*s)
              1+2*Zeta*Tw*s+(Tw*s)^2

with    Kp = 0.11276+-7.3308
        Tw = 108.08+-196.4
      Zeta = 1.8735+-3.5244
        Td = 0+-1.5255
        Tz = 9031.6+-5.5446e+005


           Kp
G_2(s) = ---------- * exp(-Td*s)
          1+Tp1*s

with  Kp = 10.068+-0.062327
     Tp1 = 2.1189+-0.072149
      Td = 4.8392+-0.046529

Estimated using PEM using SearchMethod = Auto from
data set Ze1
Loss function 8.89822 and FPE 9.04059
```

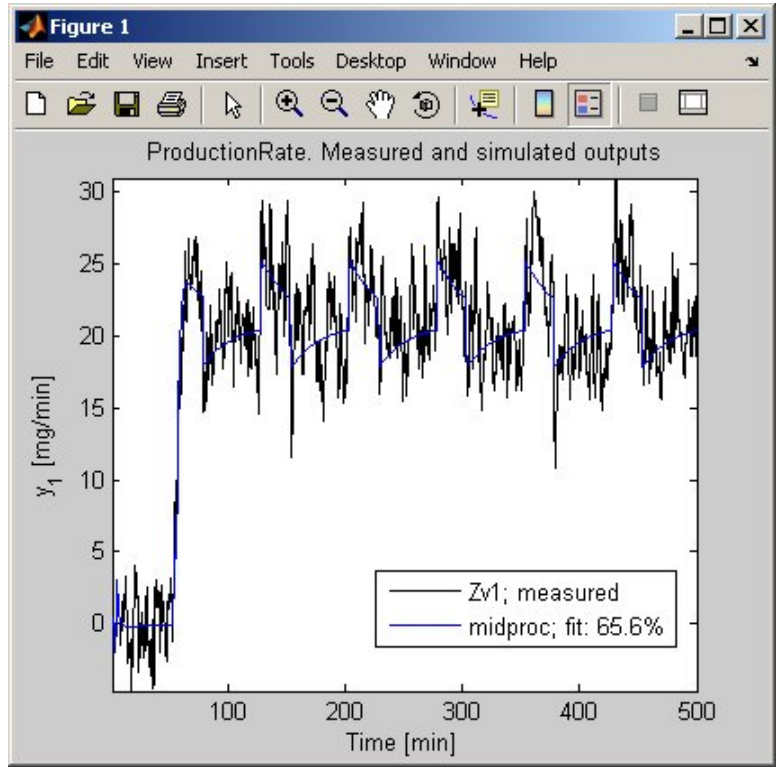
Unlike discrete-time polynomial models, continuous-time models let you estimate the delays. In this case, the estimated delay values `0` and `4.8392` are different from the initial guesses you specified of `2.5` and `5`, respectively. The large uncertainties in the estimated values of the parameters 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 the corresponding topic in the System Identification Toolbox documentation.

Validating the Process 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,midproc)
```

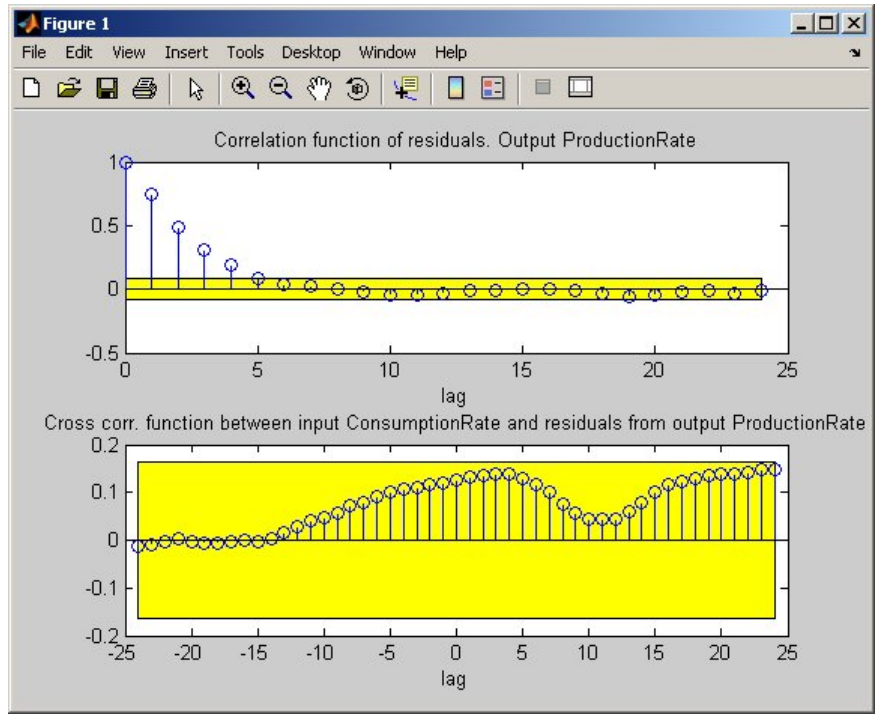


The preceding plot shows that the model output reasonably agrees with the measured output: there is an agreement of 65.6% between the model and the validation data.

Use `resid` to perform residual analysis:

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

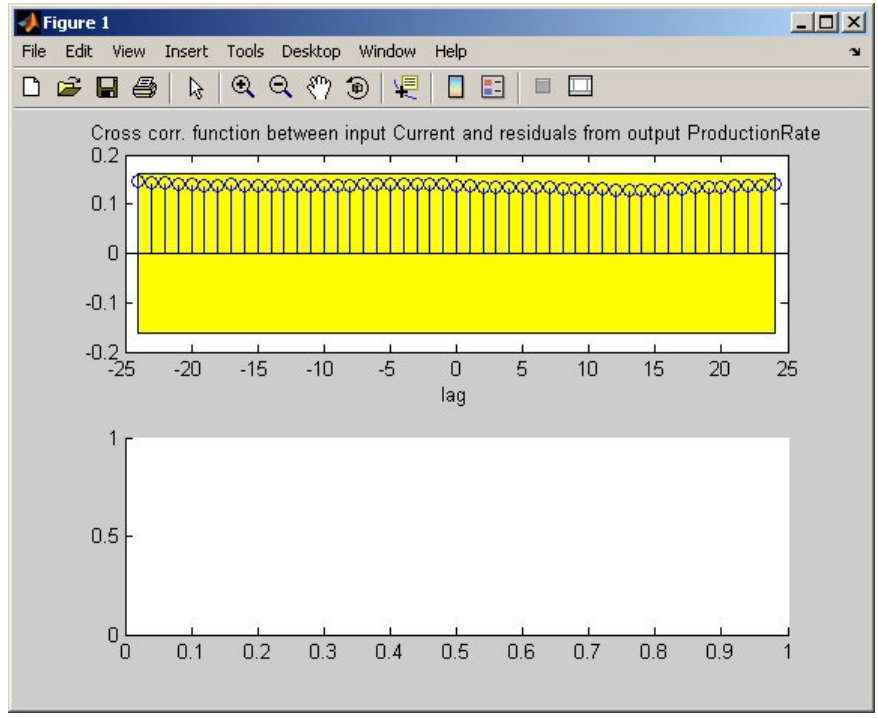
Because the sample system has two inputs, there are two cross-correlation plots of the residuals with each input, as shown in the following figure.



Autocorrelation and Cross-Correlations of Residuals with the First Input

The cross-correlation between the first input and residual errors is significant.

After MATLAB displays the first plot, press **Enter** to view the cross-correlation with the second input, as shown in the following figure.



Cross-Correlations of Residuals with the Second Input

In the preceding figure, the autocorrelation plot shows values outside the confidence region and indicates that the residuals are correlated. However, the cross-correlation with each of the two inputs shows no significant correlation.

You can try to improve the estimation results by estimating the model again with more iterations and trying different search methods. For example, you can use the model `midproc` as an initial guess for the next iterations:

```
midproc1 = pem(Ze1,midproc,'SearchMethod','lm')
compare(Zv1,midproc,midproc1)
```

```
resid(Zv1, midproc1)
```

The resulting plots show an improved fit of the simulated model output to the validation data and a reduced correlation in the residuals. The `SearchMethod='lm'` setting uses the Levenberg-Marquardt method for iterative parameter estimation.

For more information about the `SearchMethod` field of the `Algorithm` model property of linear models, see the corresponding reference page.

Estimating a Transfer Function with a 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 command to specify a first-order ARMA noise:

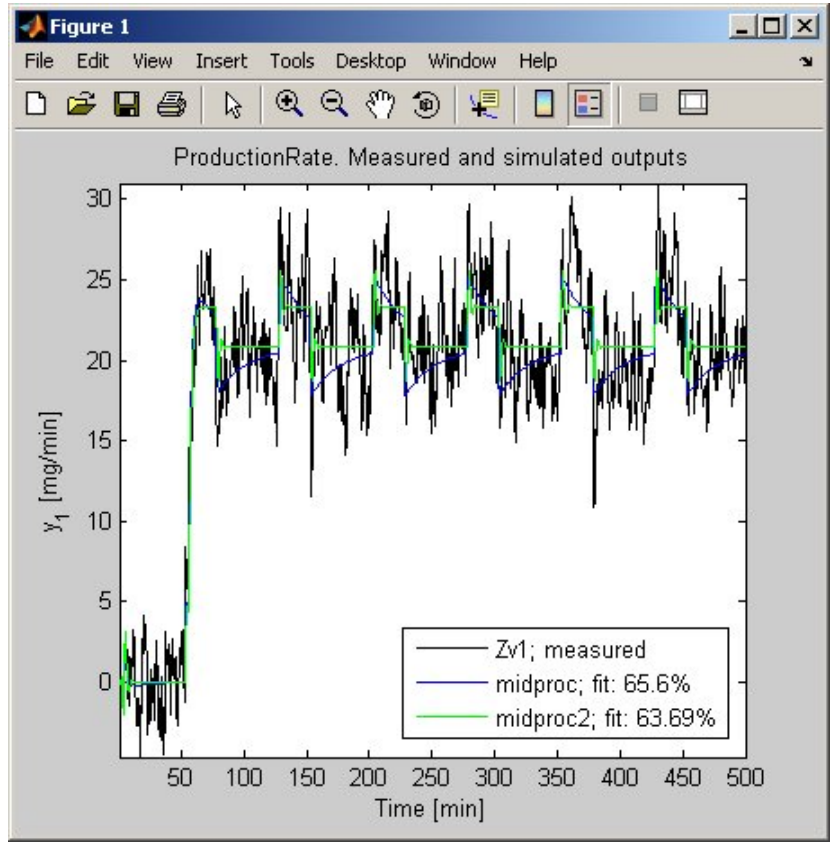
```
midproc2 = pem(Ze1,midproc0,'DisturbanceModel','arma1')
```

Note 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.

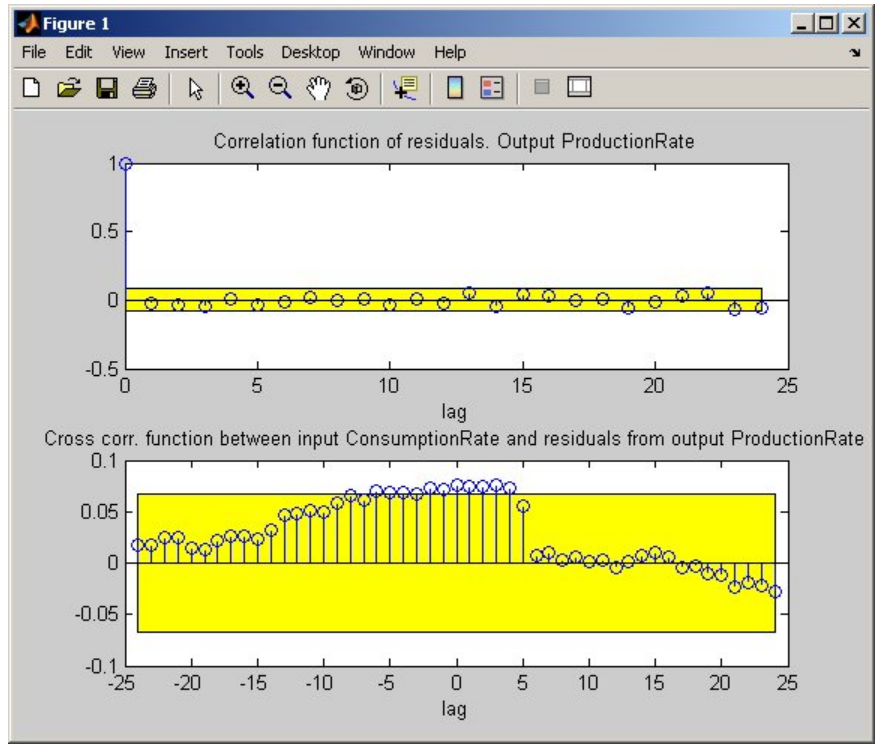
Compare the resulting model to the measured data and perform residual analysis, as follows:

```
compare(Zv1,midproc2)
figure
resid(Zv1,midproc2)
```

The following plot shows that the model output maintains reasonable agreement with the validation-data output. Press **Enter** to view the cross-correlation of the residuals with the second input.



The next plot shows that adding a noise model produces uncorrelated residuals: the top set of axes show that the autocorrelation values are inside the confidence bounds. This indicates a more accurate model.



Estimating Black-Box Polynomial Models

In this section...

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 6-4.

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:

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

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 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 $a_1 \ldots a_n$ and $b_1 \ldots b_n$ 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
```

MATLAB estimates the polynomials `A`, `B1`, and `B2`:

```
Discrete-time IDPOLY model: A(q)y(t) = B(q)u(t) + e(t)
A(q) = 1 - 1.027 (+-0.02917) q^-1
         + 0.1675 (+-0.04214) q^-2
         + 0.01307 (+-0.02591) q^-3
B1(q) = 1.86 (+-0.1896) q^-5 - 1.608 (+-0.1894) q^-6
B2(q) = 1.612 (+-0.07417) q^-10
```

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

Tip 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 -1.0266 0.1675 0.0131]
                 b: [2x11 double]
                 c: 1
                 d: 1
                 f: [2x1 double]
                da: [0 0.0292 0.0421 0.0259]
                db: [2x11 double]
                dc: 0
                dd: 0
                df: [2x1 double]
                na: 3
                nb: [2 1]
                nc: 0
                nd: 0
                nf: [0 0]
                nk: [5 10]
       InitialState: 'Auto'
               Name: ''
                 Ts: 0.5000
          InputName: {2x1 cell}
          InputUnit: {2x1 cell}
         OutputName: {'ProductionRate'}
         OutputUnit: {'mg/min'}
           TimeUnit: 'min'
    ParameterVector: [6x1 double]
              PName: {}
  CovarianceMatrix: [6x6 double]
     NoiseVariance: 2.7611
        InputDelay: [2x1 double]
         Algorithm: [1x1 struct]
    EstimationInfo: [1x1 struct]
             Notes: {}
          UserData: []
```

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
    0.2883
   -0.0570
```

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 the `zpkdata` command to compute the poles of a model directly.

Learn More

To learn more about estimating polynomial models, see the corresponding sections in the *System Identification Toolbox User's Guide*.

For more information about accessing model data, see the topic on extracting numerical data from linear models in the *System Identification Toolbox User's Guide*.

Estimating a State-Space Model

About State-Space Models

The general state-space model structure is:

$$x(t+1) = Ax(t) + Bu(t) + Ke(t)$$
$$y(t) = Cx(t) + Du(t) + e(t)$$

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.

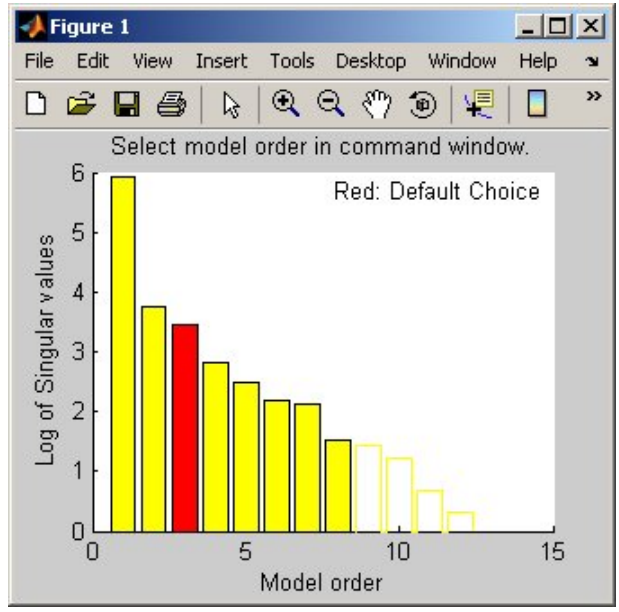
The state-space model structure is a good choice for quick estimation because it contains only two parameters: `n` is the number of poles (the size of the *A* matrix) and `nk` is the delay.

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,'nk',[5 10]);
```

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.

To select this model order, type `3` in the MATLAB Command Window, and press **Enter**.

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'`:

```
mCanonical = n4sid(Ze1,3,'nk',[5 10],...
             'ssparameterization','canonical');
present(mCanonical)    % Displays model properties
```

Note When you examine the displayed properties, notice that the model order is high. This high order occurs because the model uses additional states to incorporate input delays.

Learn More

To learn more about estimating state-space models, see the corresponding topic in the *System Identification Toolbox User's Guide*.

Estimating a Box-Jenkins Model

About Box-Jenkins Models

The general Box-Jenkins (BJ) structure is:

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

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 pem

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

```
pem(data,'na',na,'nb',nb,'nc',nc,'nd',nd,'nf',nf,'nk',nk)
```

In this case, `data` is an `iddata` or `idfrd` object, and `na`, `nb`, `nc`, `nd`, `nf`, and `nk` specify the model order.

To estimate the BJ model, type the following command in the MATLAB Command Window:

```
mbj = pem(Ze1,'nf',[2 1],'nb',[2 1],'nc',1,'nd',1,'nk',[5 10]);
present(mbj)
```

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

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 `pem` that specifically estimates the BJ model structure.

MATLAB estimates the following polynomial coefficients:

```
Discrete-time IDPOLY model:
y(t) = [B(q)/F(q)]u(t) + [C(q)/D(q)]e(t)
B1(q) = 1.823(+-0.1792)q^-5 - 1.315(+-0.2368)q^-6
B2(q) = 1.791(+-0.06435)q^-10
C(q) = 1 + 0.1066(+-0.04015)q^-1
D(q) = 1 - 0.7453(+-0.027)q^-1
F1(q) = 1 - 1.321(+-0.06939)q^-1 + 0.5911(+-0.05516)q^-2
F2(q) = 1 - 0.8314(+-0.006445)q^-1
```

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.

Learn More

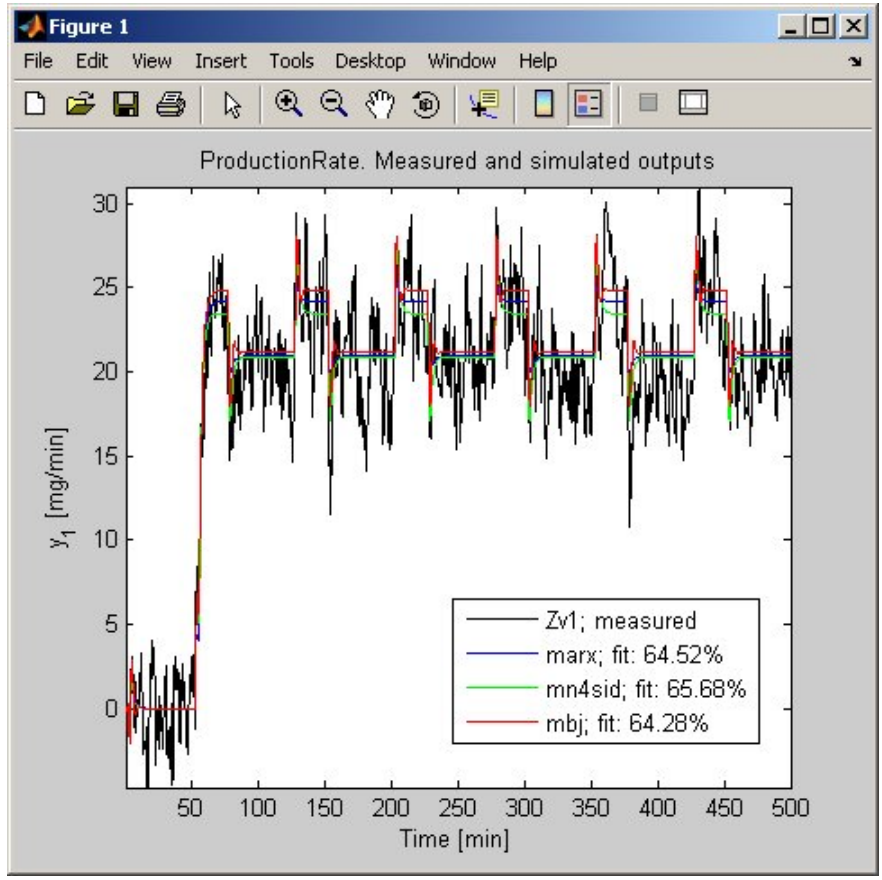
To learn more about estimating BJ models, see the corresponding topic in the *System Identification Toolbox User's Guide*.

Comparing Model Output to Measured Output

To compare the output of the ARX, state-space, and Box-Jenkins models to the measured output, use the `compare` function:

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

`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.



Measured Output and Simulated Outputs

To perform residual analysis on the ARX model, type the following command:

```
resid(Zv1,marx)
```

Because the sample system has two inputs, there are two plots showing the cross-correlation of the residuals with each input. Press **Enter** to view the cross-correlation with the second input.

To perform residual analysis on the state-space model, type the following command:

```
resid(Zv1,mn4sid)
```

Finally, to perform residual analysis on the BJ model, type the following command:

```
resid(Zv1,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

In this section...

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 Continuous-Time Transfer Functions (Process Models)" on page 6-31.

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);
% Simulate the response setting initial conditions
% equal to zero
ysim_1 = sim(midproc2,U,'InitialState','zero')
```

To match the simulated response of a model to the measured output for the same input, compute the initial states corresponding to the measured data. You can ensure that the simulated response of the model `midproc2` provides a good fit to the measured data `Zv1` by using `findstates(idmodel)`. The following command estimates the initial states `X0est` from the data set `Zv1`:

```
X0est = findstates(midproc2,Zv1);
```

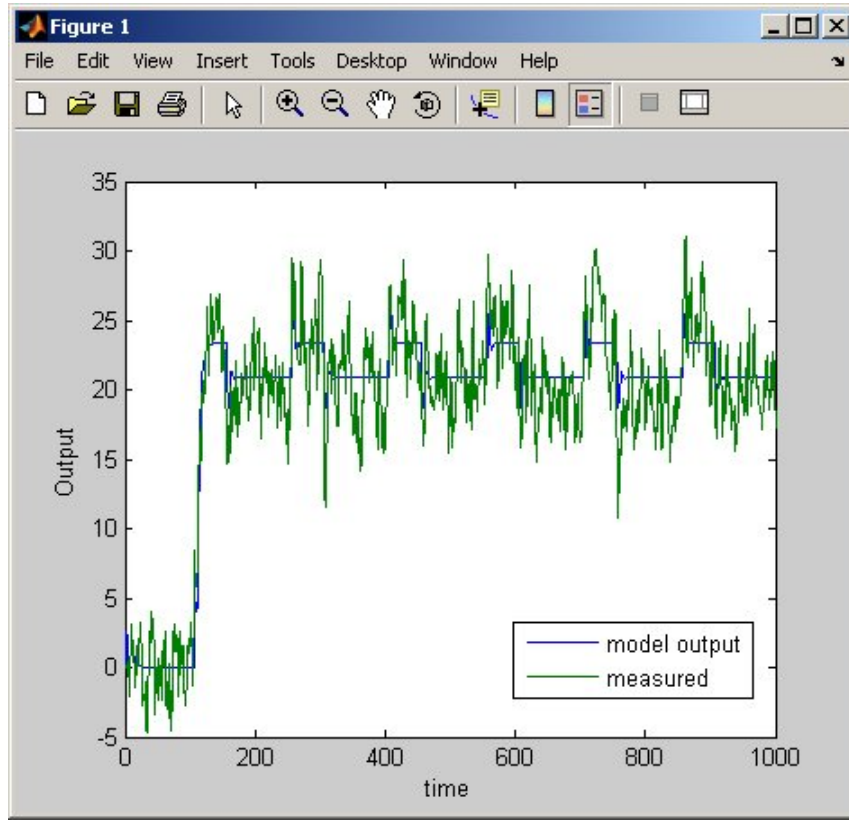
Next, simulate the model using the initial states estimated from the data:

```
Usim = Zv1.u; % Simulation input
ysim_2 = sim(midproc2,Usim,'InitialState',X0est);
```

Compare the simulated and the measured output on a plot:

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

The comparison of simulated and measured output is displayed in the following figure.



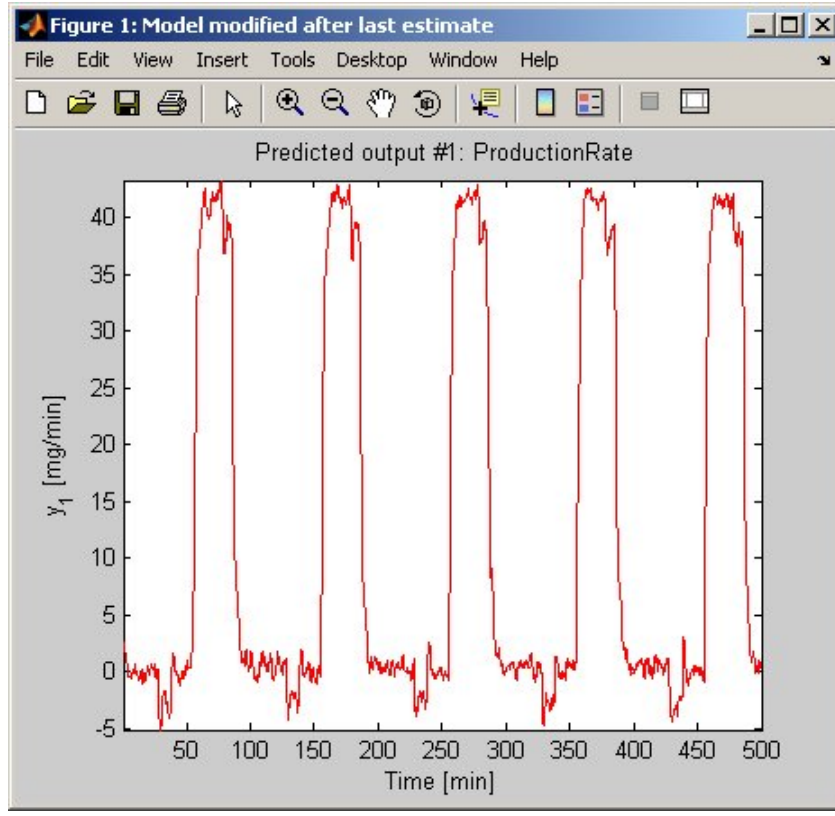
Predicting the Future Output

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)
```

The predicted output is displayed in the following figure.

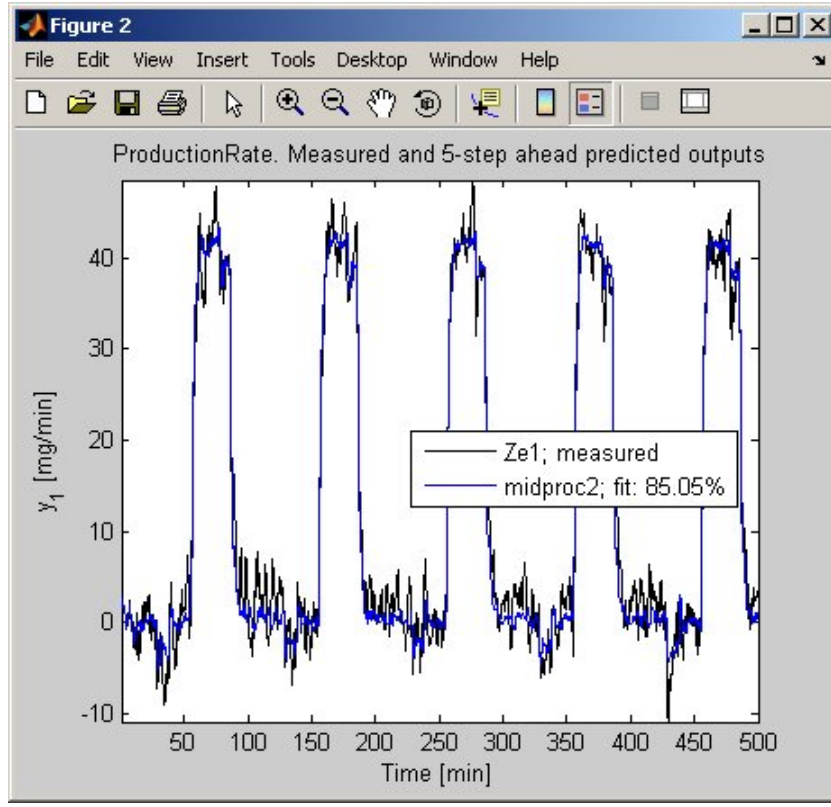


To compare the predicted output values with the measured output values, use the following command:

```
compare(Ze1,midproc2,5)
```

The third argument of `compare` specifies a five-step-ahead prediction, as shown in the following figure.

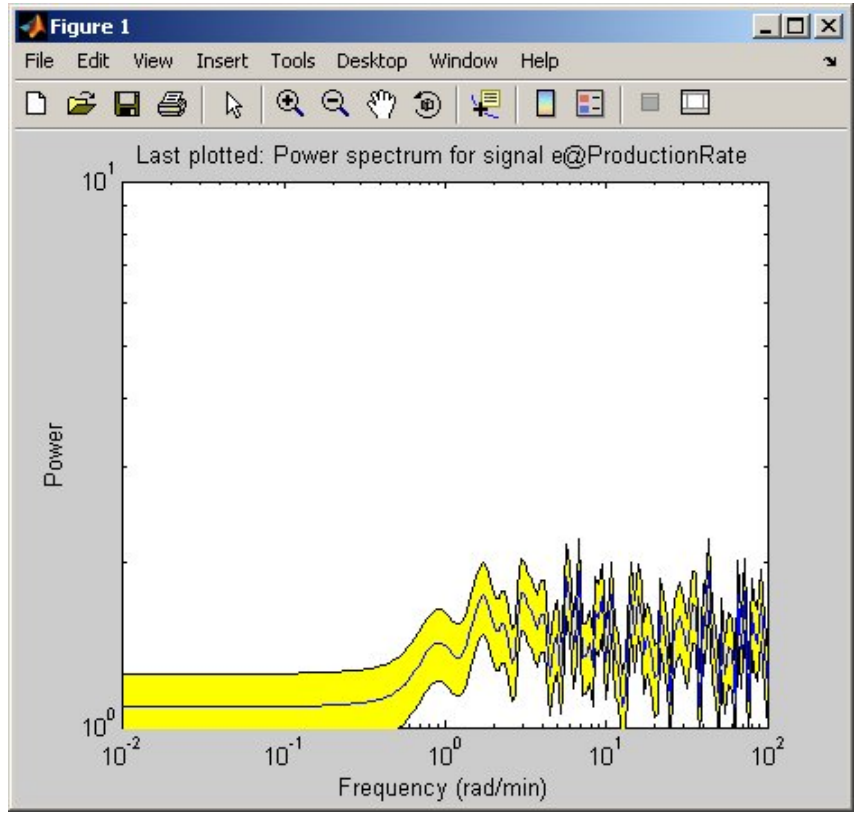
Note When you do not specify a third argument, as in "Simulating the Model Output" on page 6-54, `compare` assumes an infinite prediction horizon and simulates the model output instead.



Use `pe` to compute the prediction error `Err` between the predicted output of `midproc2` and the measured output. Then, plot the error spectrum on a Bode plot.

```
[Err] = pe(midproc2,Zv1);
bode(spa(Err,[],logspace(-2,2,200)),...
         'mode','same','sd',1,'fill')
```

As shown in the following figure, 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.



7

Tutorial – Identifying Nonlinear Black-Box Models Using the GUI

About This Tutorial

In this section...

Objectives

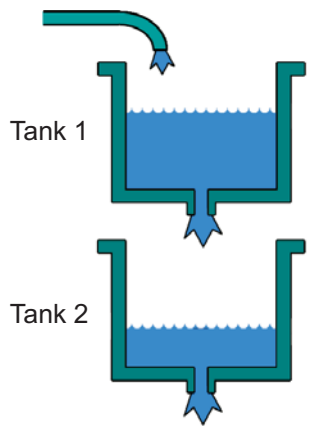
Estimate and validate nonlinear models from single-input/single-output (SISO) data to find the one that best represents your system dynamics.

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

- Import data objects from the MATLAB workspace into the GUI.
- 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 sampling interval of 0.2 s.

What Are Nonlinear Black-Box Models?

In this section...

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:

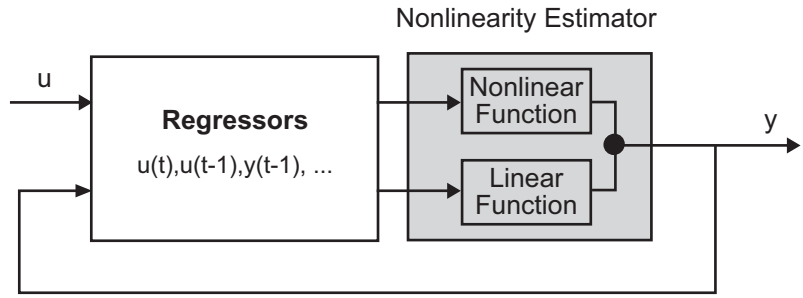
- Nonlinear ARX models
- Hammerstein-Wiener models

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.

To learn how to estimate nonlinear black-box models at the command line, see the topics on estimating nonlinear models in the *System Identification Toolbox User's Guide*.

What Is a Nonlinear ARX Model?

This block diagram represents the structure of a nonlinear ARX model:



The nonlinear ARX model computes the output *y* in two stages:

1. Computes regressors from the current and past input values and past output data.

   In the simplest case, regressors are delayed inputs and outputs, such as *u(t-1)* and *y(t-3)*—called *standard* regressors. You can also specify *custom* regressors, which are nonlinear functions of delayed inputs and outputs. For example, tan(*u*(*t*-1)) or *u*(*t*-1)**y*(*t*-3).

   By default, all regressors are inputs to both the linear and the nonlinear function blocks of the nonlinearity estimator. You can choose a subset of regressors as inputs to the nonlinear function block.

2. The nonlinearity estimator block maps the regressors to the model output using a combination of nonlinear and linear functions. You can select from available nonlinearity estimators, such as tree-partition networks, wavelet networks, and multi-layer neural networks. You can also exclude either the linear or the nonlinear function block from the nonlinearity estimator.

The nonlinearity estimator block can include linear and nonlinear blocks in parallel. For example:

$$F(x) = L^T(x - r) + d + g\left(Q(x - r)\right)$$

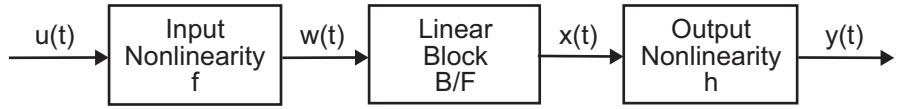
x is a vector of the regressors. $L^T(x) + d$ is the output of the linear function block and is affine when d≠0. d is a scalar offset. $g\left(Q(x - r)\right)$ represents the output of the nonlinear function block. r is the mean of the regressors x. Q is

a projection matrix that makes the calculations well conditioned. The exact form of $F(x)$ depends on your choice of the nonlinearity estimator.

Estimating a nonlinear ARX model computes the model parameter values, such as L, r, d, Q, and other parameters specifying g. Resulting models are `idnlarx` objects that store all model data, including model regressors and parameters of the nonlinearity estimator. See the `idnlarx` reference page for more information.

What Is a Hammerstein-Wiener Model?

This block diagram represents the structure of a Hammerstein-Wiener model:



where:

- $w(t) = f(u(t))$ is a nonlinear function transforming input data $u(t)$. *w(t)* has the same dimension as *u(t)*.
- $x(t) = (B/F)w(t)$ is a linear transfer function. *x(t)* has the same dimension as *y(t)*.

  where B and F are similar to polynomials in the linear Output-Error model, as described in "What Are Black-Box 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 $j = 1,2,\ldots,ny$ and $i = 1,2,\ldots,nu$.
- $y(t) = h(x(t))$ is a nonlinear function that maps the output of the linear block to the system output.

w(t) and *x(t)* are internal variables that define the input and output of the linear block, respectively.

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 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 nonlinearities f and h are scalar functions, one nonlinear function for each input and output channel.

The Hammerstein-Wiener model computes the output y in three stages:

1 Computes $w(t) = f(u(t))$ 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 Computes 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 numerator B and denominator F orders.

3 Compute the model output by transforming the output of the linear block $x(t)$ using the nonlinear function h: $y(t) = h(x(t))$.

Similar to the input nonlinearity, the output nonlinearity is a static function. 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 estimator. See the `idnlhw` reference page for more information.

Preparing Data

In this section...

Loading 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 "Loading Data into the MATLAB Workspace" on page 7-9.

Use the following commands to create two data objects, `ze` and `zv`, where `ze` contains data for model estimation and `zv` contains data for model validation. `Ts` is the sampling interval.

```
Ts = 0.2; % Sampling interval 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 an `iddata` object, use the `get` command. For example, type this command to get the properties of the estimation data:

```
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: ''
      ExperimentName: 'Exp1'
               Notes: []
            UserData: []
```

To learn more about these properties, see the `iddata` reference pages.

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:

```
% 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
```

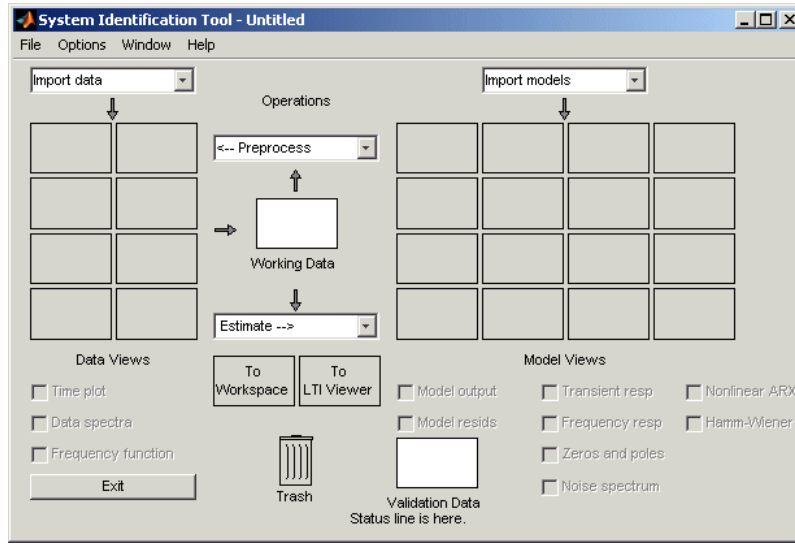
Tip Property names, such as `InputName`, are not case sensitive. You can also abbreviate property names that start with `Input` or `Output` by substituting `u` for `Input` and `y` for `Output` in the property name. For example, `OutputUnit` is equivalent to `yunit`.

Starting the System Identification Tool

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

```
ident
```

The default session name, `Untitled`, appears in the title bar.



Importing Data Objects into the System Identification Tool

You can import the data objects into the GUI from the MATLAB workspace.

You must have already created the data objects, as described in "Creating iddata Objects" on page 7-9, and opened the GUI, as described in "Starting the System Identification Tool" on page 7-11.

1 In the System Identification Tool GUI, 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 Tool GUI.

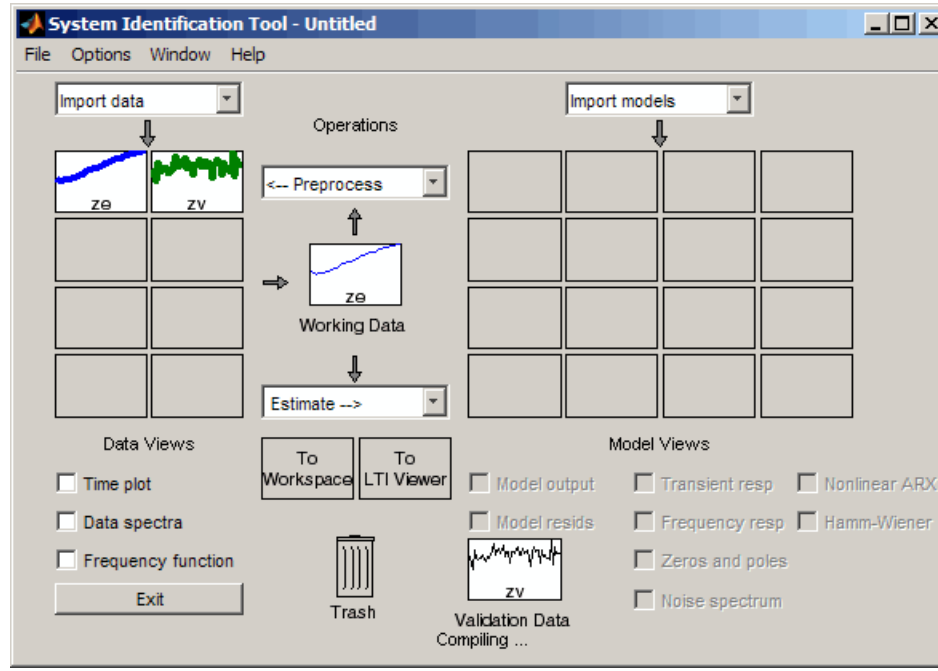
4. In the Import Data dialog box, type `zv` in the **Object** field to import the validation data. Press **Enter**.

5. Click **Import** to add the icon named `zv` to the System Identification Tool GUI.

6. In the Import Data dialog box, click **Close**.

7. In the System Identification Tool GUI, drag the validation data **zv** icon to the **Validation Data** rectangle. The estimation data **ze** icon is already designated in the **Working Data** rectangle.

   The System Identification Tool GUI resembles the following figure.


System Identification Tool - Untitled
File
Options
Window
Help
Import data
Operations
Import models
ze
zv
<-- Preprocess
ze
Working Data
Estimate -->
Data Views
Time plot
Data spectra
Frequency function
Exit
To Workspace
To LTI Viewer
Trash
Model Views
Model output
Model resids
Transient resp
Frequency resp
Nonlinear ARX
Hamm-Wiener
Zeros and poles
Noise spectrum
zv
Validation Data
Compiling ...

Estimating Nonlinear ARX Models

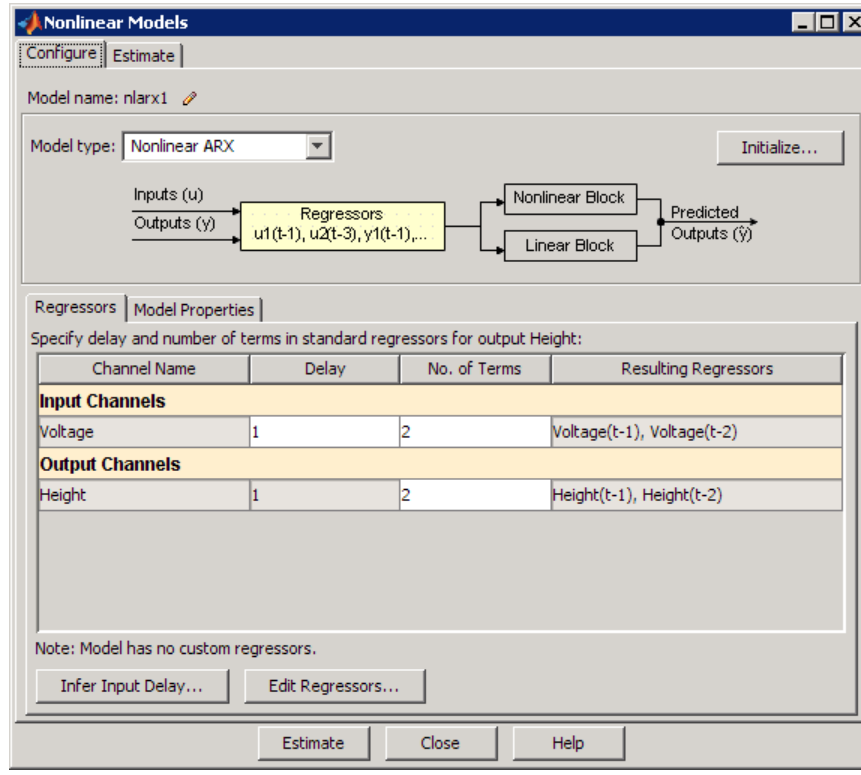
In this section...

Estimating a 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 "Preparing Data" on page 7-9. For more information about nonlinear ARX models, see "What Is a Nonlinear ARX Model?" on page 7-4

1 In the System Identification Tool GUI, select **Estimate > Nonlinear models** to open the Nonlinear Models dialog box.



The **Configure** tab is already open and the default **Model structure** is `Nonlinear ARX`.

In the **Regressors** tab, the **Input Channels** and **Output Channels** have **Delay** set to `1` and **No. of Terms** set to `2`. The model output *y(t)* is related to the input *u(t)* via the following nonlinear autoregressive equation:

$$y(t) = f\left(y(t-1), y(t-2), u(t-1), u(t-2)\right)$$

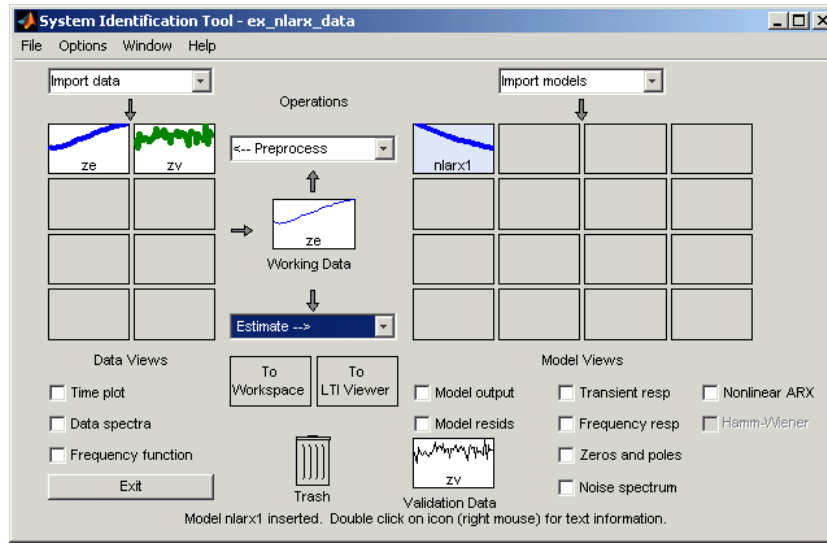
f is the nonlinearity estimator you select in the **Model Properties** tab.

2 Click the **Model Properties** tab.

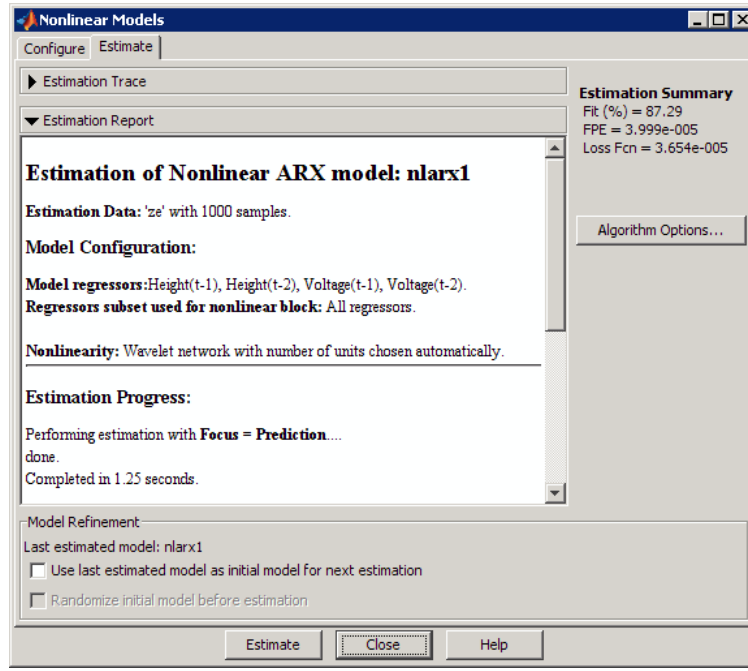
The **Nonlinearity** represents the nonlinear function *f* and is set to `Wavelet Network` by default. The number of units for the nonlinearity estimator is set to **Select automatically** and controls the flexibility of the nonlinearity—more units correspond to a more flexible nonlinearity.

3 Click **Estimate**.

This action adds the model `nlarx1` to the System Identification Tool GUI, as shown in the following figure.

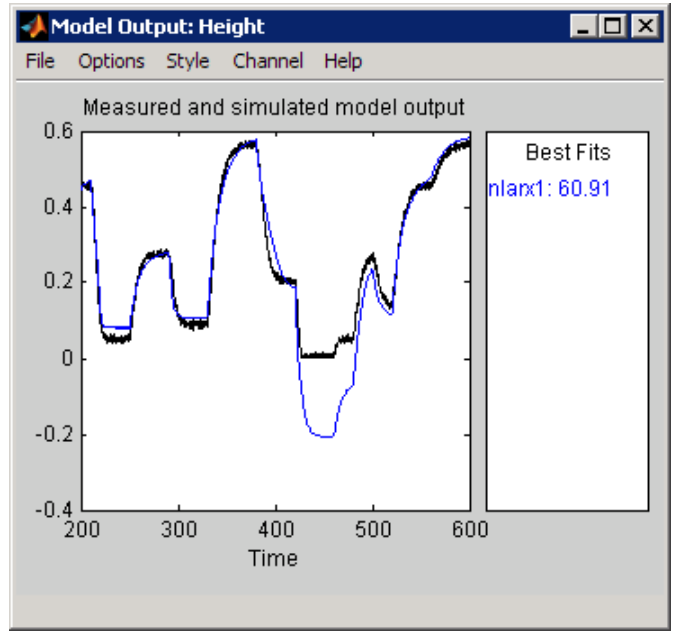


The Nonlinear Models dialog box displays a summary of the estimation information in the **Estimate** tab. The **Fit (%)** is the mean square error between the measured data and the simulated output of the model: 100% corresponds to a perfect fit (no error) and 0% to a model that is not capable of explaining any of the variation of the output and only the mean level.



Note **Fit (%)** is computed using the estimation data set, and not the validation data set. However, the model output plot in the next step compares the fit to the validation data set.

4 In the System Identification Tool GUI, select the **Model output** check box. Simulation of the model output uses the input validation data as input to the model. It plots the simulated output on top of the output validation data.



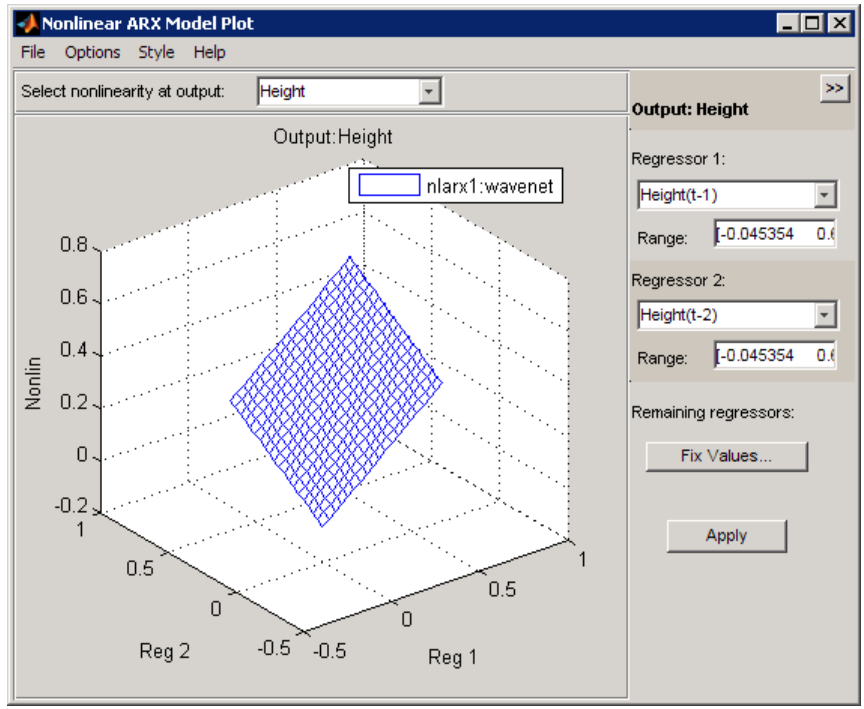
The **Best Fits** area of the Model Output plot shows that the agreement between the model output and the validation-data output.

Plotting 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.

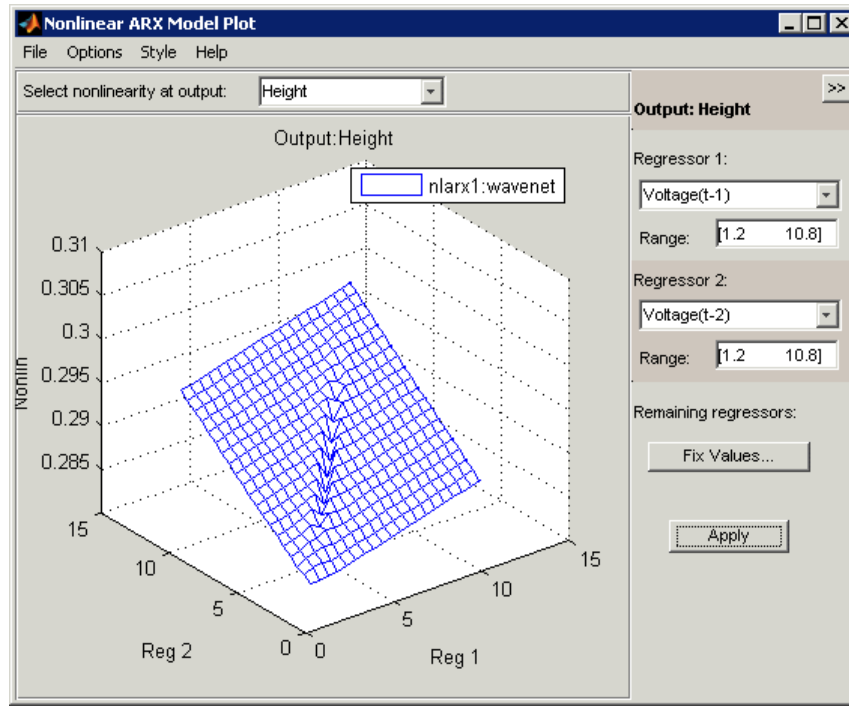
1 In the System Identification Tool GUI, select the **Nonlinear ARX** check box to view the nonlinearity cross-sections.

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**.

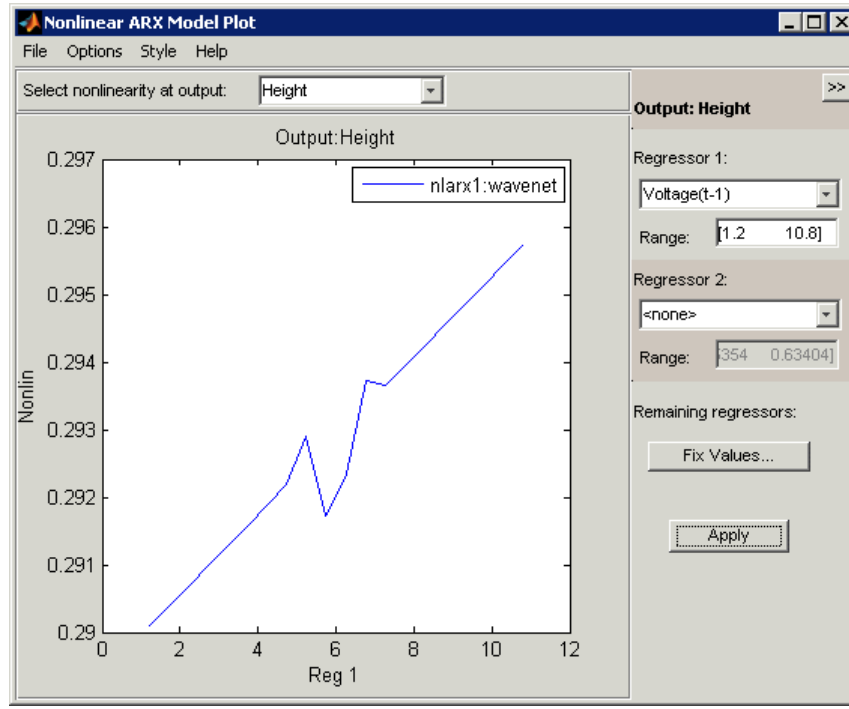
The relationship between these regressors and the output is nonlinear, as shown in the following plot.



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)`.



Changing the Nonlinear ARX Model Structure

In this portion of the tutorial, you estimate a nonlinear ARX model with specific input delay and the 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 "Estimating a Nonlinear ARX Model with Default Settings" on page 7-15.

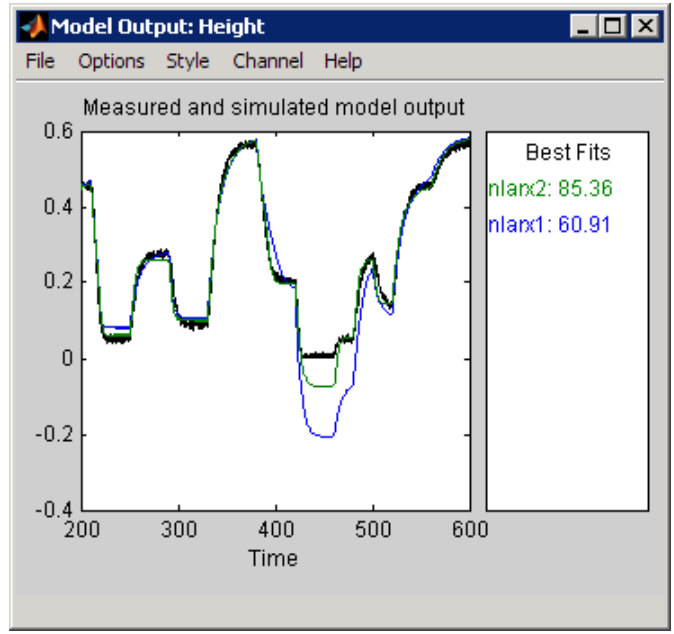
1 In the Nonlinear Models dialog box, click the **Configure** tab, and click the **Regressors** tab.

2 For the `Voltage` input channel, double-click the corresponding **Delay** cell, enter `3`, and press **Enter**.

   This action updates the **Resulting Regressors** list. The list now includes `Voltage(t-3)` and `Voltage(t-4)`—terms with a minimum input delay of three samples.

3 Click **Estimate**.

   This action adds the model `nlarx2` to the System Identification Tool GUI and updates the Model Output window to include this model. The Nonlinear Models dialog box displays the new estimation information in the **Estimate** tab.

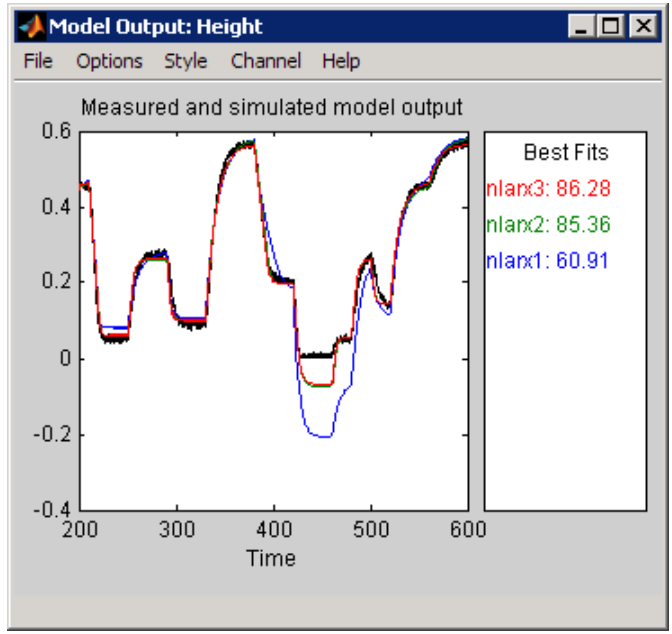


4 In the Nonlinear Models dialog box, click the **Configure** tab, and select the **Model Properties** tab.

5 In the **Number of units in nonlinear block** area, select **Enter**, and type `6`. This number controls the flexibility of the nonlinearity.

6 Click **Estimate**.

This action adds the model `nlarx3` to the System Identification Tool GUI. It also updates the Model Output window, as shown in the following figure.



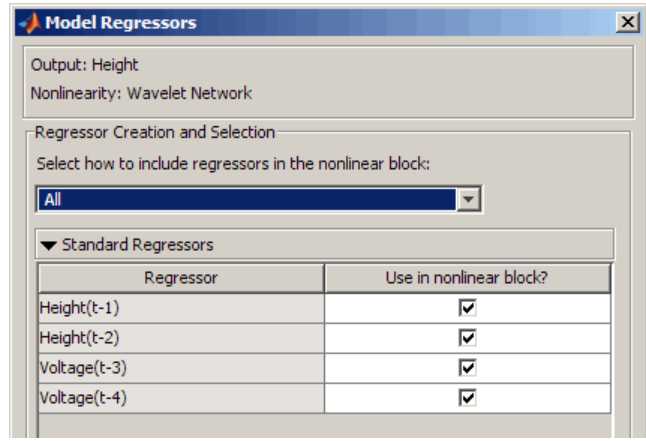
Selecting a Subset of Regressors in the Nonlinear Block

You can estimate a nonlinear ARX model that includes only a subset of standard regressors that enter as inputs to the nonlinear block. By default, all standard and custom regressors are used in the nonlinear block. In this example, you only have standard regressors.

You must have already specified the model structure, as described in "Changing the Nonlinear ARX Model Structure" on page 7-22.

1 In the Nonlinear Models dialog box, click the **Configure** tab, and select the **Regressors** tab.

2 Click the **Edit Regressors** button to open the Model Regressors dialog box.



3 Clear the following check boxes:

- **Height(t-2)**
- **Voltage(t-3)**

Click **OK**.

This action excludes the time-shifted `Height(t-2)` and `Voltage(t-3)` from the list of inputs to the nonlinear block.

4 Click **Estimate**.

This action adds the model `nlarx4` to the System Identification Tool GUI. It also updates the Model Output window.

Efficiently Modifying Model Structure for Estimating Nonlinear ARX Models

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 example, you estimate a nonlinear ARX model based that is similar to an existing model (`nlarx3`), but with a different nonlinearity.

1 In the Nonlinear Models dialog box, select the **Configure** tab. Click **Initialize**. This action opens the Initial Model Specification dialog box.

2 In the **Initial Model** list, select `nlarx3`. Click **OK**.

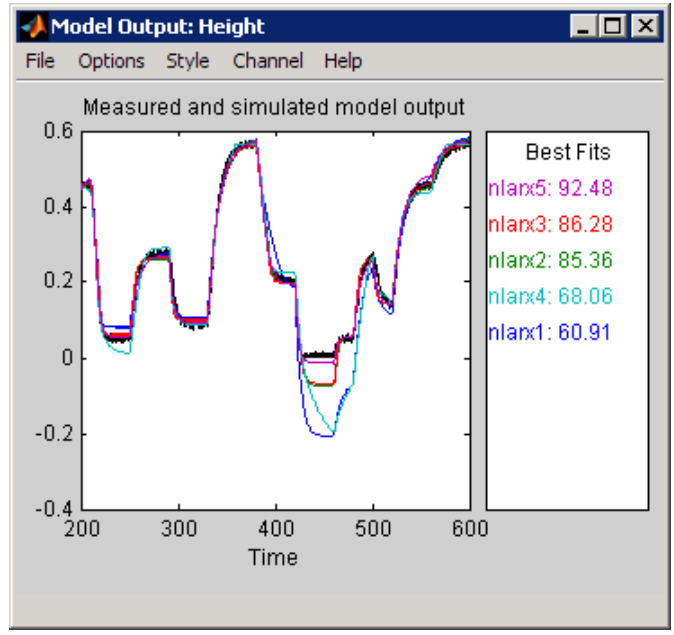
3 Click the **Model Properties** tab.

4 In the **Nonlinearity** list, select `Sigmoid Network`.

5 In the **Number of units in nonlinear block** field, type `6`.

6 Click **Estimate**.

This action adds the model `nlarx5` to the System Identification Tool GUI. It also updates the Model Output plot, as shown in the following figure.



Selecting the Best Model

The best model is the simplest model that accurately describes the dynamics. In this tutorial, the best model fit was produced in “Efficiently Modifying Model Structure for Estimating Nonlinear ARX Models” on page 7-25.

To view information about this model, including the model order, nonlinearity, and list of regressors, right-click the model icon in the System Identification Tool GUI.

Estimating Hammerstein-Wiener Models

In this section...

Estimating Hammerstein-Wiener Models with Default Settings

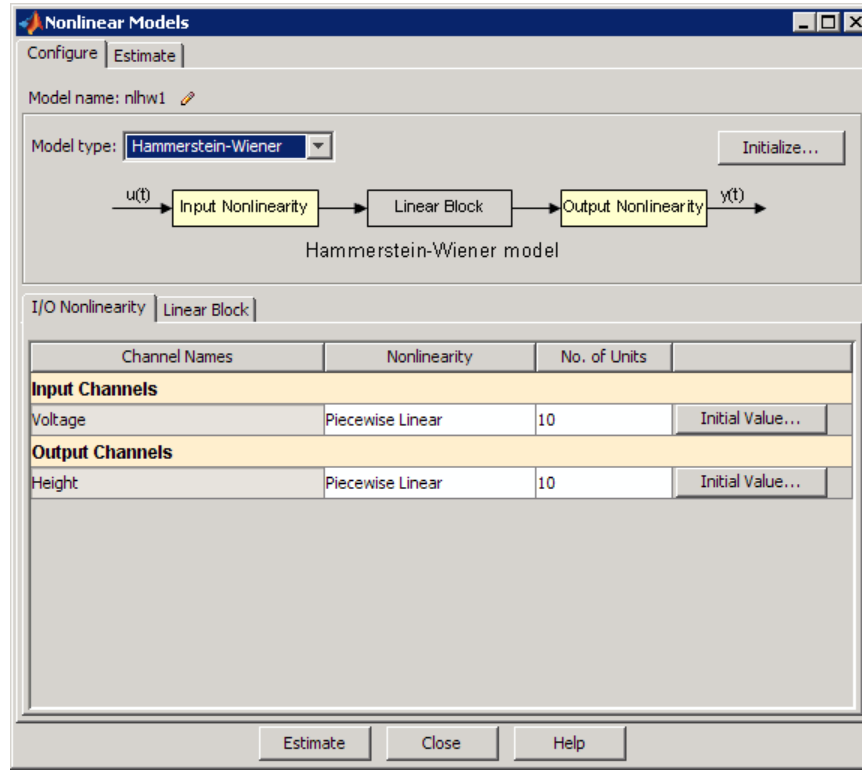
In this portion of the tutorial, you estimate nonlinear Hammerstein-Wiener models using default model structure and estimation options.

You must have already prepared the data, as described in "Preparing Data" on page 7-9. For more information about nonlinear ARX models, see "What Is a Hammerstein-Wiener Model?" on page 7-6

1. In the System Identification Tool GUI, select **Estimate > Nonlinear models** to open the Nonlinear Models dialog box.

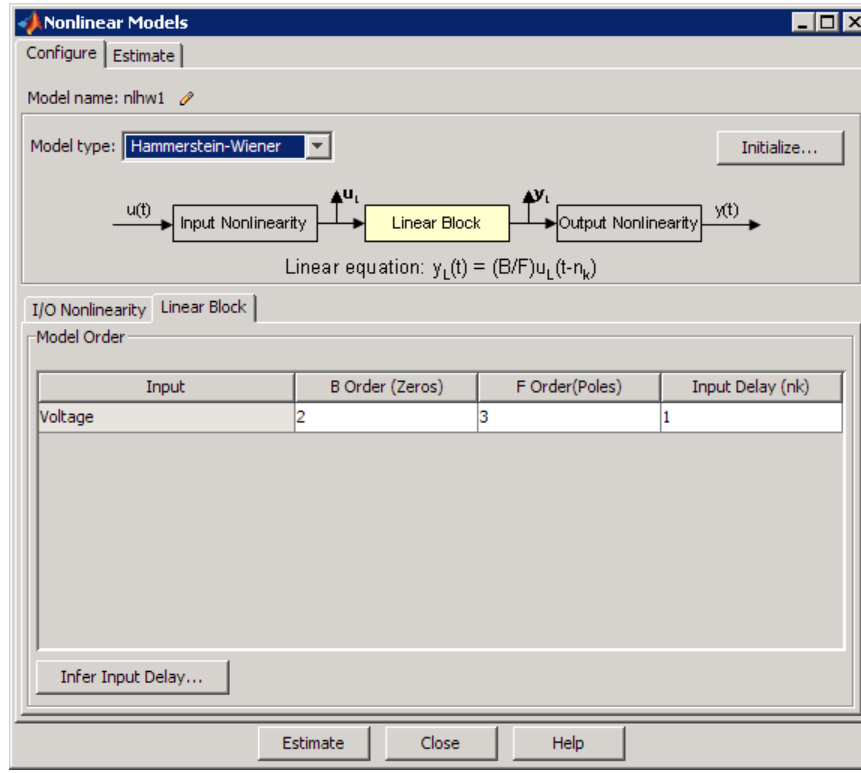
2. In the **Configure** tab, select `Hammerstein-Wiener` in the **Model type** list.

3 Keep the defaults in the **I/O Nonlinearity** tab.



By default, the nonlinearity estimator is `Piecewise Linear` with `10` units for **Input Channels** and **Output Channels**, which corresponds to `10` breakpoints for the piecewise linear function.

4 In the **Linear Block** tab, keep the defaults.



By default, the model orders and delays of the linear output-error (OE) model are n_b=2, n_f=3, and n_k=1.

5 Click **Estimate**.

This action adds the model `nlhw1` to the System Identification Tool GUI.

6 In the System Identification Tool GUI, select the **Model output** check box.

Simulation of the model output uses the input validation data as input to the model. It plots the simulated output on top of the output validation data.

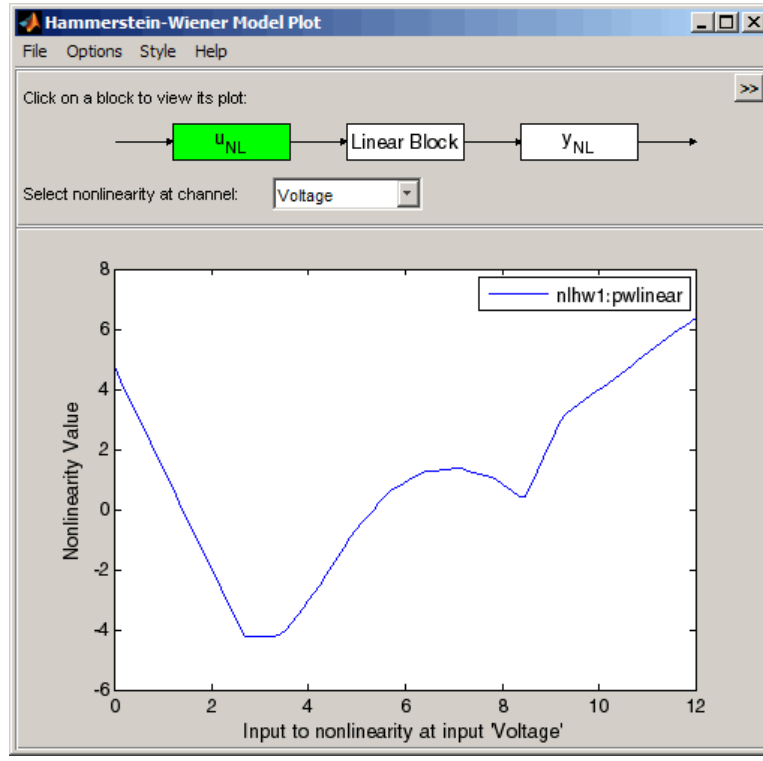
The **Best Fits** area of the Model Output window shows the agreement between the model output and the validation-data output.

Plotting the 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.

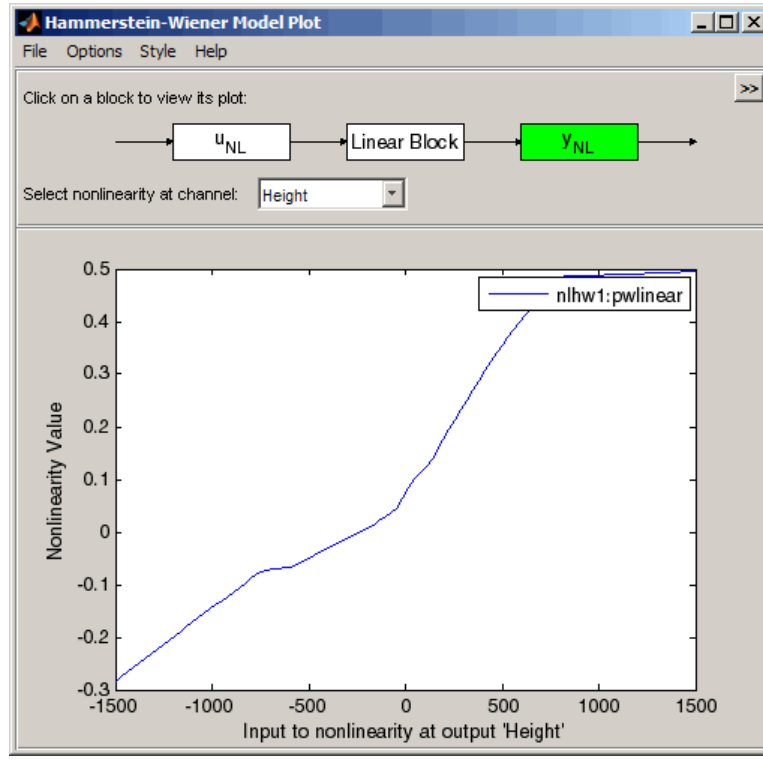
1 In the System Identification Tool GUI, select the **Hamm-Wiener** check box to view the Hammerstein-Wiener model plot.

The plot displays the input nonlinearity, as shown in the following figure.



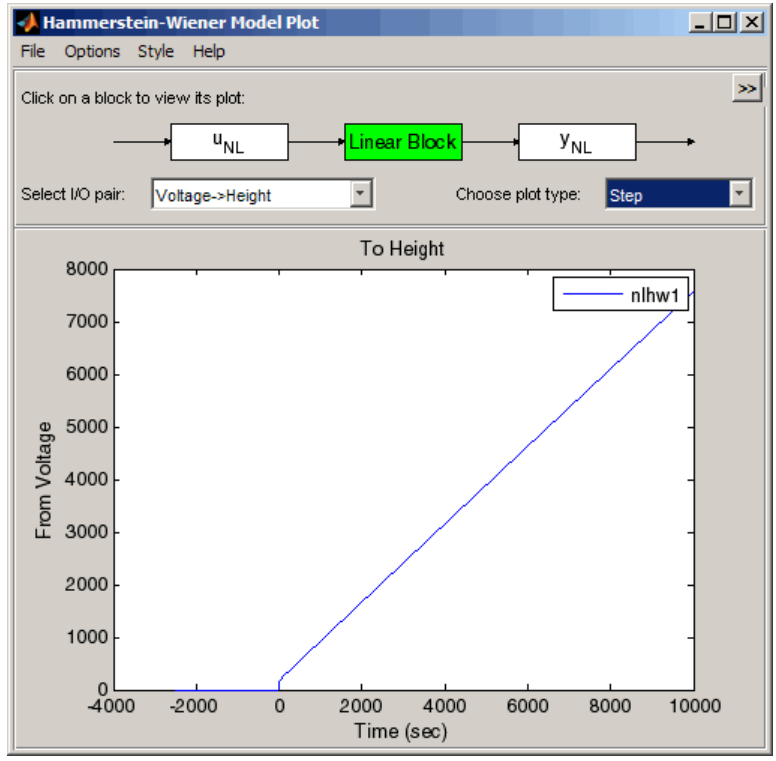
2 Click the y_{NL} rectangle in the top portion of the Hammerstein-Wiener Model Plot window.

The plot updates to display the output nonlinearity.

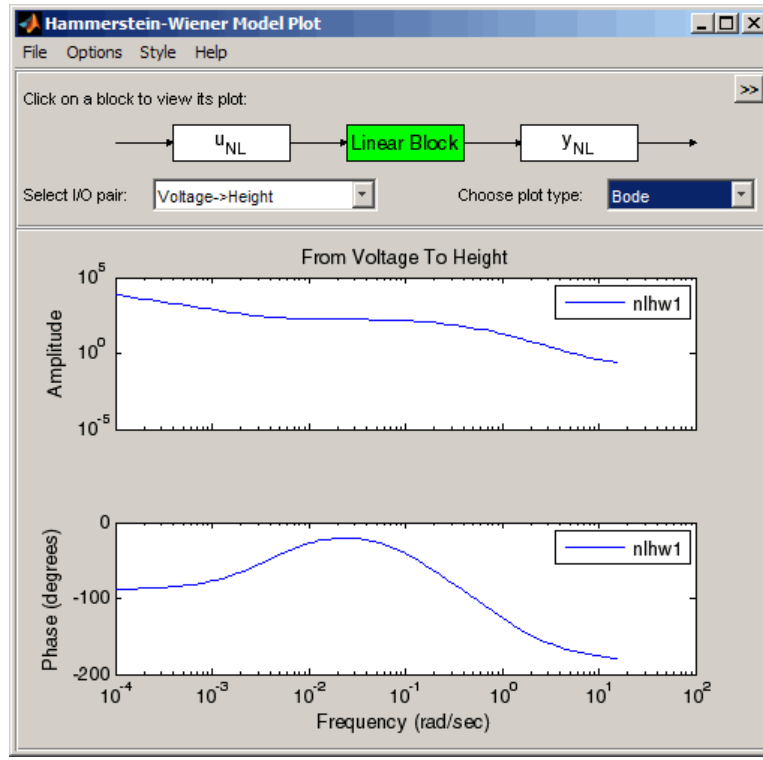


3 Click the **Linear Block** rectangle in the top portion of the Hammerstein-Wiener Model Plot window.

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.



Changing the Hammerstein-Wiener Model Structure

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 “Estimating Hammerstein-Wiener Models with Default Settings” on page 7-28.

1 In the Nonlinear Models dialog box, click the **Configure** tab, and select the **Linear Block** tab.

2 For the `Voltage` input channel, double-click the corresponding **Input Delay (nk)** cell, change the value to `3`, and press **Enter**.

3 Click **Estimate**.

This action adds the model `nlhw2` to the System Identification Tool GUI and the Model Output window is updated to include this model, as shown in the following figure.

The **Best Fits** area of the Model Output window shows the quality of the `nlhw2` fit.

Changing the Nonlinearity Estimator in a 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` and `Sigmoid Network` are nonlinearity estimators for general nonlinearity approximation.

---

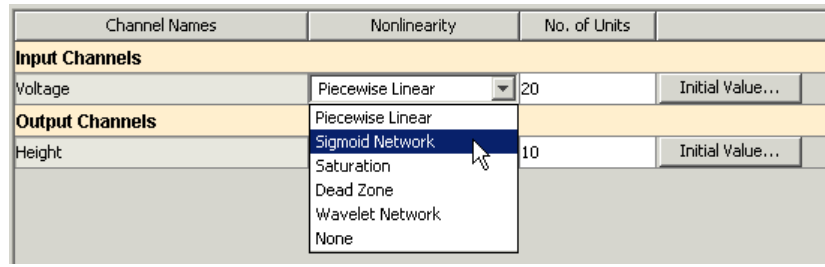
1 In the Nonlinear Models dialog box, click the **Configure** tab, and click the **Linear Block** tab.

2 For the `Voltage` input channel, double-click the corresponding **Input Delay (nk)** cell, type `1`, and press **Enter**.

This action restores the input delay to the default value.

3 Select the **I/O Nonlinearity** tab.

4 For the `Voltage` input, click the **Nonlinearity** cell, and select `Sigmoid Network` from the list. Click the corresponding **No. of Units** cell and set the value to `20`.



5 Click **Estimate**.

This action adds the model `nlhw3` to the System Identification Tool GUI. It also updates the Model Output window.

6 In the Nonlinear Models dialog box, click the **Configure** tab, and click the **I/O Nonlinearity** tab.

7 Set the `Voltage` input **Nonlinearity** to `Wavelet Network`. This action sets the **No. of Units** to be determined automatically, by default.

8 Set the `Height` output **Nonlinearity** to `One-dimensional Polynomial`.

9 Click **Estimate**.

This action adds the model `nlhw4` to the System Identification Tool GUI. It also updates the Model Output window, as shown in the following figure.

Selecting the Best Model

The best model is the simplest model that accurately describes the dynamics.

In this example, the best model fit was produced in “Estimating Hammerstein-Wiener Models with Default Settings” on page 7-28.

Index

L

M

N

P

Q

R

S

T

U

V