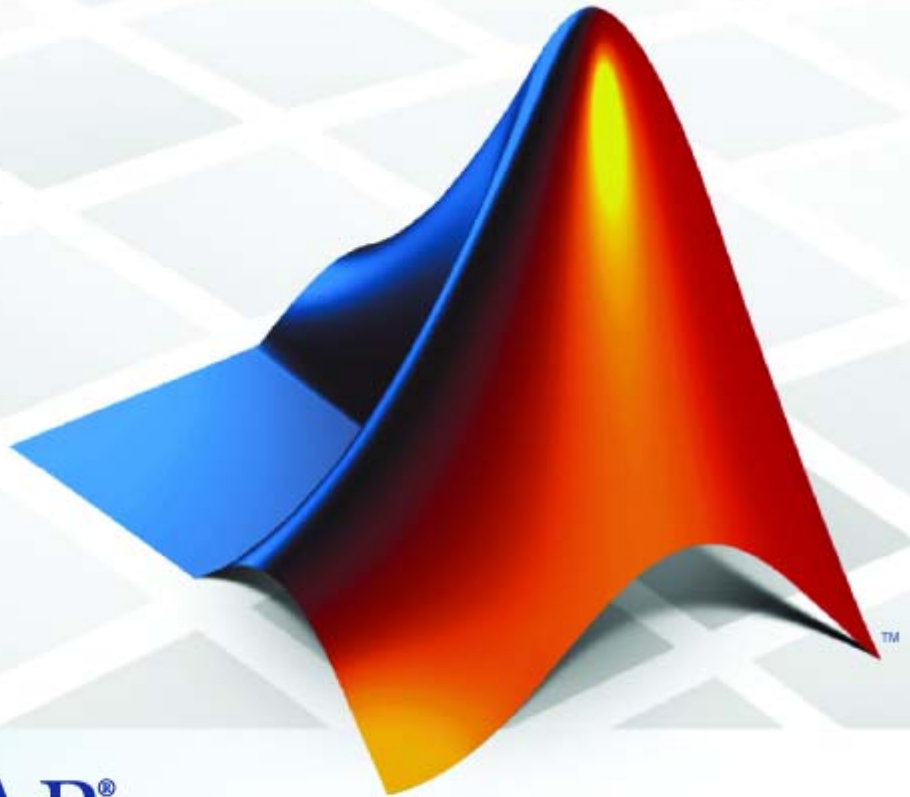


# System Identification Toolbox™ 7

## Getting Started Guide

*Lennart Ljung*



**MATLAB®**  
& **SIMULINK®**

## How to Contact The MathWorks



www.mathworks.com  
comp.soft-sys.matlab  
www.mathworks.com/contact\_TS.html

Web  
Newsgroup  
Technical Support



suggest@mathworks.com  
bugs@mathworks.com  
doc@mathworks.com  
service@mathworks.com  
info@mathworks.com

Product enhancement suggestions  
Bug reports  
Documentation error reports  
Order status, license renewals, passcodes  
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*

© COPYRIGHT 1988–2008 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

### Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See [www.mathworks.com/trademarks](http://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](http://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)

## 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.



## Product Overview

### 1

<b>What You Can Accomplish Using This Toolbox</b> .....	<b>1-2</b>
<b>Types of Data You Can Model</b> .....	<b>1-3</b>
<b>How This Toolbox Supports Identifying Dynamic Systems</b> .....	<b>1-4</b>
<b>Accessing the Documentation and Demos</b> .....	<b>1-5</b>
Accessing Documentation .....	<b>1-5</b>
Accessing Demos .....	<b>1-5</b>
<b>Related Products</b> .....	<b>1-7</b>
<b>Learn More</b> .....	<b>1-9</b>

## Using This Product

### 2

<b>When to Use the GUI Versus the Command Line</b> .....	<b>2-2</b>
<b>Starting This Toolbox</b> .....	<b>2-3</b>
<b>Steps for Using This Toolbox</b> .....	<b>2-4</b>
<b>Tutorials to Help You Get Started</b> .....	<b>2-6</b>

## Choosing Models to Estimate

### 3

<b>Data-Driven Modeling Using System Identification Toolbox™ Software</b> .....	<b>3-2</b>
<b>When to Identify Linear Versus Nonlinear Models</b> ....	<b>3-4</b>
<b>When to Identify Models from First Principles</b> .....	<b>3-6</b>
<b>When to Identify Black-Box Models</b> .....	<b>3-7</b>

## Tutorial – Identifying Linear Models Using the GUI

### 4

<b>About This Tutorial</b> .....	<b>4-3</b>
Objectives .....	<b>4-3</b>
Sample Data .....	<b>4-3</b>
<b>Preparing Data</b> .....	<b>4-5</b>
Loading Data into the MATLAB® Workspace .....	<b>4-5</b>
Opening the System Identification Tool GUI .....	<b>4-5</b>
Importing Data Arrays into the System Identification Tool .....	<b>4-6</b>
Plotting and Preprocessing Data .....	<b>4-11</b>
<b>Saving the GUI Session</b> .....	<b>4-20</b>
<b>Identifying Models Using Default Settings</b> .....	<b>4-22</b>
Why Identify Models Using Default Settings? .....	<b>4-22</b>
Using Quick Start to Identify Preliminary Models .....	<b>4-23</b>
Validating Preliminary Models .....	<b>4-23</b>
Types of Models Generated by Quick Start .....	<b>4-27</b>
<b>Refining Models</b> .....	<b>4-29</b>
Strategy for Refining Models .....	<b>4-29</b>

Estimating Possible Model Orders .....	4-29
Identifying State-Space and ARMAX Models .....	4-34
Choosing the Best Model .....	4-38
<b>Viewing Model Parameters .....</b>	<b>4-45</b>
Viewing Model Parameter Values .....	4-45
Viewing Parameter Uncertainties .....	4-47
<b>Exporting the Model to the MATLAB® Workspace .....</b>	<b>4-48</b>
<b>Exporting the Model to the LTI Viewer .....</b>	<b>4-50</b>

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

# 5

<b>About This Tutorial .....</b>	<b>5-3</b>
Objectives .....	5-3
Sample Data .....	5-3
<b>What Is a Continuous-Time Process Model? .....</b>	<b>5-5</b>
<b>Preparing Data .....</b>	<b>5-6</b>
Loading Data into the MATLAB® Workspace .....	5-6
Opening the System Identification Tool GUI .....	5-6
Importing Data Objects into the System Identification Tool .....	5-7
Plotting and Preprocessing Data .....	5-9
<b>Estimating Second-Order Transfer Functions (Process Models) with Complex Poles .....</b>	<b>5-13</b>
Estimating a Second-Order Transfer Function Using Default Settings .....	5-13
Tips for Specifying Known Parameters .....	5-18
Validating the Model .....	5-18
<b>Refining the Model by Including a Noise Model .....</b>	<b>5-22</b>

Estimating Models with Modified Settings .....	5-22
Comparing Models .....	5-23
<b>Viewing Model Parameters .....</b>	<b>5-26</b>
Viewing Model Parameter Values .....	5-26
Viewing Parameter Uncertainties .....	5-27
<b>Exporting the Model to the MATLAB® Workspace .....</b>	<b>5-29</b>
<b>Using the System Identification Toolbox™ Product</b>	
<b>with the Simulink® Software .....</b>	<b>5-30</b>
Preparing Input Data .....	5-30
Building the Simulink® Model .....	5-30
Configuring Blocks and Simulation Parameters .....	5-32
Running the Simulation .....	5-36

## Tutorial – Identifying Linear Models Using the Command Line

# 6

<b>About This Tutorial .....</b>	<b>6-3</b>
Objectives .....	6-3
Sample Data .....	6-3
<b>Preparing Data .....</b>	<b>6-5</b>
Loading Data into the MATLAB® Workspace .....	6-5
Plotting the Input/Output Data .....	6-6
Removing Equilibrium Values from the Data .....	6-7
Using Objects to Represent Data for System Identification .....	6-8
Creating iddata Objects .....	6-9
Plotting the Data .....	6-11
Selecting a Subset of the Data .....	6-15
<b>Estimating Step- and Frequency-Response Models .....</b>	<b>6-17</b>
Why Estimate Step- and Frequency-Response Models? ...	6-17
Estimating the Frequency Response .....	6-17
Estimating the Step Response .....	6-20



<b>Estimating Delays in the Multiple-Input System</b> .....	<b>6-22</b>
Why Estimate Delays? .....	<b>6-22</b>
Estimating Delays Using the ARX Model Structure .....	<b>6-22</b>
Estimating Delays Using Alternative Methods .....	<b>6-23</b>
<b>Estimating Model Orders Using an ARX Model</b>	
<b>Structure</b> .....	<b>6-25</b>
Why Estimate Model Order? .....	<b>6-25</b>
Commands for Estimating the Model Order .....	<b>6-25</b>
Model Order for the First Input-Output Combination ....	<b>6-27</b>
Model Order for the Second Input-Output Combination ..	<b>6-30</b>
<b>Estimating Continuous-Time Transfer Functions</b>	
<b>(Process Models)</b> .....	<b>6-33</b>
Specifying the Structure of the Process Model .....	<b>6-33</b>
Viewing the Model Structure and Parameter Values .....	<b>6-34</b>
Specifying Initial Guesses for Time Delays .....	<b>6-35</b>
Estimating Model Parameters Using pem .....	<b>6-36</b>
Validating the Process Model .....	<b>6-37</b>
Refining the Process Model by Including a Noise Model ..	<b>6-40</b>
<b>Estimating Black-Box Polynomial Models</b> .....	<b>6-44</b>
Model Orders for Estimating Polynomial Models .....	<b>6-44</b>
Estimating a Linear ARX Model .....	<b>6-45</b>
Estimating a State-Space Model .....	<b>6-48</b>
Estimating a Box-Jenkins Model .....	<b>6-51</b>
Comparing Models .....	<b>6-53</b>
<b>Simulating and Predicting Model Output</b> .....	<b>6-56</b>
Simulating the Model Output .....	<b>6-56</b>
Predicting the Future Output .....	<b>6-58</b>

## **Tutorial – Identifying Nonlinear Black-Box Models Using the GUI**

# 7

<b>About This Tutorial</b> .....	<b>7-2</b>
Objectives .....	<b>7-2</b>
Sample Data .....	<b>7-2</b>

<b>Preparing Data</b> .....	<b>7-4</b>
Loading Data into the MATLAB® Workspace .....	<b>7-4</b>
Creating iddata Objects .....	<b>7-4</b>
Starting the System Identification Tool .....	<b>7-6</b>
Importing Data Objects into the System Identification Tool .....	<b>7-7</b>
<b>Estimating Nonlinear ARX Models</b> .....	<b>7-9</b>
Estimating a Nonlinear ARX Model with Default Settings .....	<b>7-9</b>
Plotting Nonlinearity Cross-Sections for Nonlinear ARX Models .....	<b>7-13</b>
Changing the Nonlinear ARX Model Structure .....	<b>7-16</b>
Selecting a Subset of Regressors in the Nonlinear Block ..	<b>7-18</b>
Changing the Nonlinearity Estimator in a Nonlinear ARX Model .....	<b>7-20</b>
Selecting the Best Model .....	<b>7-21</b>
<b>Estimating Hammerstein-Wiener Models</b> .....	<b>7-23</b>
Estimating Hammerstein-Wiener Models with Default Settings .....	<b>7-23</b>
Plotting the Nonlinearities and Linear Transfer Function .....	<b>7-27</b>
Changing the Hammerstein-Wiener Model Structure ....	<b>7-31</b>
Changing the Nonlinearity Estimator in a Hammerstein-Wiener Model .....	<b>7-33</b>
Selecting the Best Model .....	<b>7-35</b>

## **Index**

---

# Product Overview

---

What You Can Accomplish Using This Toolbox (p. 1-2)	Summary of the capabilities of the System Identification Toolbox™ product
Types of Data You Can Model (p. 1-3)	Data supported by the System Identification Toolbox product
How This Toolbox Supports Identifying Dynamic Systems (p. 1-4)	How the System Identification Toolbox product fits into the stages of identifying dynamic systems
Accessing the Documentation and Demos (p. 1-5)	Information about installing the product, using the documentation, and accessing demos
Related Products (p. 1-7)	Summary of products that extend the capabilities of the System Identification Toolbox product
Learn More (p. 1-9)	References for learning more about modeling dynamic systems and system identification theory

## What You Can Accomplish Using This Toolbox

System Identification Toolbox™ software extends the MATLAB® computational environment for estimating linear and nonlinear mathematical models to fit measured data from dynamic systems. You might use the resulting model to simulate the output of a system for a given input and analyze the system response, predict future system outputs based on previous inputs and outputs, or for control design.

System identification is especially useful for modeling systems that you cannot easily represent in terms of first principles. In this case, you use the System Identification Toolbox software to perform *black-box modeling*, where the measured data determines the model structure. Examples of complex dynamic systems requiring black-box models include engine subsystems, flight dynamics systems, thermofluid processes, and electromechanical systems.

You can also use the 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.

You can validate models directly after each estimation to help you select the best dynamic model for your system.

For an overview of using the System Identification Toolbox software, see “Steps for Using This Toolbox” on page 2-4.

## Types of Data You Can Model

You can estimate linear models from both time- and frequency-domain data with single or multiple inputs and outputs. Time-domain data can be either real or complex. For nonlinear models, System Identification Toolbox™ software supports only time-domain data.

*Time-domain data* is one or more input variables  $u(t)$  and one or more output variables  $y(t)$ , sampled as a function of time. A special case of time-domain data is time-series data, which is one or more outputs  $y(t)$  and no measured input.

*Frequency-domain data* is the Fourier transform of the input and output time-domain signals. *Frequency-response data*, also called *frequency-function data*, represents complex frequency-response values for a linear system characterized by its transfer function  $G$ .

You can measure frequency-response data values directly using a spectrum analyzer, for example. Often, *frequency-domain* and *frequency-response* data are both referenced as frequency-domain data for the sake of brevity.

## How This Toolbox Supports Identifying Dynamic Systems

The general system identification process might include the following stages:

- 1** Experimental design and data acquisition
- 2** Data analysis and preprocessing, including plotting the data, removing offsets and linear trends, filtering, resampling, and selecting regions of interest
- 3** Estimation and validation of models
- 4** Model analysis and transformation, such as linear analysis, reducing model order, and converting between discrete-time and continuous-time representations
- 5** Model usage for intended applications, such as simulation or prediction of output values or control design

The System Identification Toolbox™ product supports all of these stages except data acquisition. This toolbox provides some support for experimental design by enabling you to generate input signals with different properties. You can also model data to validate and refine your experimental design.

## Accessing the Documentation and Demos

In this section...
“Accessing Documentation” on page 1-5
“Accessing Demos” on page 1-5

### Accessing Documentation

The MathWorks™ technical documentation is available online from the **Help** menu on the MATLAB® desktop.

The System Identification Toolbox™ documentation contains the following components:

- Getting Started Guide — Provides essential information for mapping your problem to the capabilities of the System Identification Toolbox product. Step-by-step tutorials walk you through the most common system identification tasks.
- User’s Guide — Describes tasks for using the System Identification Toolbox software.
- Reference — Describes the System Identification Toolbox commands.
- Release Notes — Describes important changes in the current product version and compatibility considerations.

**New Users.** The Getting Started Guide helps you begin using this toolbox quickly. After a brief introduction to the types of models you can estimate, follow the steps in the tutorials to estimate models in the System Identification Tool graphical user interface (GUI) or the MATLAB Command Window.

**Experienced Users.** Search or browse the documentation for information about specific tasks.

### Accessing Demos

The System Identification Toolbox product provides demo files that show you how to estimate models for dynamic systems from measured data. The available demos include both case studies and tutorials.

To access demos in the Help browser, type the following command in the MATLAB Command Window:

```
demo
```

In the **Demos** pane, select **Toolboxes > System Identification** to open the list of available demos.



## 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](http://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 <code>lsqnonlin</code> optimization algorithm for 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.

<b>Product</b>	<b>Description</b>
Signal Processing Toolbox™	<p>Provides additional options for:</p> <ul style="list-style-type: none"><li>• Filtering (The System Identification Toolbox software provides only the fifth-order Butterworth filter.)</li><li>• Spectral analysis</li></ul> <p>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.</p>
Simulink®	<p>Provides System Identification blocks for simulating the models you identified using the System Identification Toolbox software. Also provides blocks for model estimation.</p>

## Learn More

The goal of the System Identification Toolbox™ documentation is to provide 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 article:

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

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.



# Using This Product

---

When to Use the GUI Versus the  
Command Line (p. 2-2)

When to use the GUI versus the  
System Identification Toolbox™  
commands

Starting This Toolbox (p. 2-3)

Opening the System Identification  
Tool GUI and getting more  
information about command-line  
syntax

Steps for Using This Toolbox (p. 2-4)

Summary of typical tasks in the  
system identification workflow

Tutorials to Help You Get Started  
(p. 2-6)

Summary of tutorials to help new  
users get started quickly

## 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 2-2.

To open the System Identification Tool GUI:

- Select **Start > Toolboxes > System Identification** from the MATLAB® Command Window.

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** Prepare data for system identification by:

- Importing data into the MATLAB® workspace.
- Importing data into the System Identification Tool GUI or creating an `iddata` or `idfrd` object in the MATLAB Command Window.
- 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 controller design. For more information about linearizing a nonlinear plant, see the `linapp` and `linearize` reference pages.

## 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 single-input/single-output (SISO) 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 single-input/single-output (SISO) 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 multiple-input/single-output (MISO) 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 single-input/single-output (SISO) data using the System Identification Tool GUI.

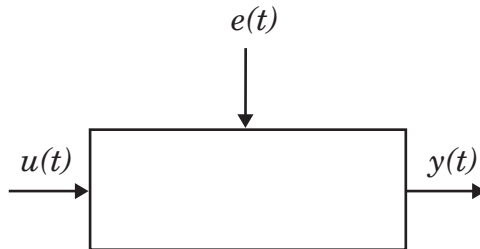
# Choosing Models to Estimate

---

Data-Driven Modeling Using System Identification Toolbox™ Software (p. 3-2)	Definition of data-driven modeling and the general structure of a dynamic model
When to Identify Linear Versus Nonlinear Models (p. 3-4)	Criteria for choosing linear or nonlinear models
When to Identify Models from First Principles (p. 3-6)	Criteria for choosing when to use an ODE to represent your system
When to Identify Black-Box Models (p. 3-7)	Criteria for choosing black-box models for systems that you cannot easily represent in terms of known physical laws

## Data-Driven Modeling Using System Identification Toolbox™ Software

*Data-driven modeling* means that you are fitting models to measured data.



Models describe the relationship between one or more measured input signals,  $u(t)$ , and one or more measured output signals,  $y(t)$ . The System Identification Toolbox™ software supports modeling input and output signals measured in either the time or frequency domain.

In addition, this toolbox supports modeling additional inputs that affect the system output and that you cannot measure or control. Such unmeasured inputs are called *disturbances* or *noise*,  $e(t)$ . For example, if the system is an airplane, its inputs might be the positions of various control surfaces, such as ailerons and elevators. The system outputs might be the airplane orientation, velocity, and position. The noise might be turbulence and wind gusts that affect the outputs.

You use System Identification Toolbox software to identify models that are special cases of the following general mathematical description of dynamic systems:

$$y(t) = g(u, \theta) + v(t)$$

The output  $y(t)$  of a system is determined by  $g$ , which might be a function of all previous inputs  $u(s)$  ( $s \leq t$ ) and system parameters  $\theta$ .  $v(t)$  is the output noise.

For nonlinear models,  $g$  can take a variety of forms.

For linear models, the general symbolic model description is given by:

$$y = Gu + He$$

$G$  is an operator that describes the system dynamics from the input to the output.  $G$  is often called a *transfer function* between  $u$  and  $y$ .  $H$  is an operator that describes the properties of the additive output disturbance and is called a *disturbance model*, or *noise model*. The actual disturbance contribution to the output,  $He$ , has real significance and contains all the known and unknown influences on the measured  $y$  not included in the input  $u$ . Therefore, if you repeat and experiment with the same input,  $He$  explains why the output signal is different.

The source of the noise,  $e$ , need not have a physical significance. In the case of an airplane, it is sufficient to estimate the noise in a black-box manner as arising from a white noise source via a transfer function  $H$ . Thus, you do not need to know how the wind gusts and turbulence are generated physically and all that matters are the characteristics of  $He$ , such as the frequency content of the spectrum of  $He$ .

If you know that your measured data includes noise, you can choose a model structure that computes  $H$  to produce a more accurate dynamic model. For more information about choosing to model noise in linear black-box models, see “When to Identify Black-Box Models” on page 3-7.

## When to Identify Linear Versus Nonlinear Models

You can identify both linear and nonlinear models using System Identification Toolbox™ software. In practice, all systems are nonlinear and the output is a nonlinear function of the input variables. However, a linear model is often sufficient to accurately describe the system dynamics.

Follow these guidelines to choose between using nonlinear and linear black-box models:

- When you have physical insight that the system is nonlinear, try transforming your input and output variables such that the relationship between the transformed variables is linear.

For example, you might be dealing with a system that has current and voltage as inputs to an immersion heater, and the temperature of the heated liquid as an output. In this case, the output depends on the inputs via the power of the heater, which is equal to the product of current and voltage. Instead of fitting a nonlinear model to two-input and one-output data, you can create a new input variable by taking the product of current and voltage and then fitting a linear model to the single-input/single-output data.

- Plot the response of the system to a specific input. If you notice that the responses are different depending on the input level or input sign, use a nonlinear model. For example, you might see that the output response to an input step up is much faster than the response to a step down.
- Try identifying several linear models of varying complexity. If the model output does not adequately reproduce the measured output, you might need to use a nonlinear model. Noisy data might also cause a model to fail reproducing measured output.

For a grey-box model, its linear or nonlinear structure is set by its differential or difference equations (ODEs). For more information about choosing this type of model, see “When to Identify Models from First Principles” on page 3-6.

For a black-box model, you can choose whether to estimate linear or nonlinear models. Linear approximations are very useful because they are simple and provide good results in many situations. Therefore, always estimate linear models first and see how well these models represent the dynamics. For

more information about choosing this type of model, see “When to Identify Black-Box Models” on page 3-7

---

**Note** For nonlinear black-box models, you can only estimate discrete-time models using time-domain data.

---

## When to Identify Models from First Principles

A *grey-box model* has a known mathematical structure and unknown parameters. Use this approach if you understand the physics of your system and can represent the system dynamics using ordinary differential or difference equations (ODEs).

You capture the ODE and the parameters you want to estimate in an M-file or MEX-file, and use the System Identification Toolbox™ product to estimate the model parameters. For example, to estimate the parameters of a transfer function that you defined, you must first represent the transfer function in state-space form.

Grey-box modeling has the following advantages over black-box modeling:

- You can impose known constraints on model characteristics, such as model parameters and noise variance.
- There are potentially fewer parameters to estimate.
- You can specify couplings between parameters when defining the model structure.
- In the nonlinear case, you can specify the dynamic equations explicitly.

Grey-box modeling is preferred. However, grey-box modeling requires that you know the relationship between the system variables and the parameters, which can be time consuming. For an alternative to grey-box modeling, see “When to Identify Black-Box Models” on page 3-7.

For information about estimating linear and nonlinear grey-box models, see the *System Identification Toolbox User’s Guide*.



## When to Identify Black-Box Models

System identification is especially useful for modeling systems that you cannot easily represent in terms of first principles or known physical laws. The parameters of a black-box model might not have a physical interpretation.

Black-box models can be linear or nonlinear. You can identify several black-box models that have different orders and choose the model that has the best performance. Black-box models can also be continuous-time or discrete-time models.

Black-box modeling has the following advantages:

- You do not need to know the structure and order of your model to get started quickly.
- You can estimate many model structures and compare them to choose the best one.

System Identification Toolbox™ software provides linear black-box model structure that let you model noise explicitly. For example, you can model noise for a low-order transfer function or a state-space model. Some input-output polynomial models, such as the ARMAX or Box-Jenkins (BJ) structure, provide additional parameters to model noise and decouple the dynamics from the noise.

---

**Note** Nonlinear black-box models do not support parametric noise modeling.

---

You might choose a linear model structure with a noise model in the following situations:

- You are specifically interested in a noise model, such as when developing noise-cancellation and noise-attenuation technologies, or for disturbance rejection in control design applications.
- You want to use the noise characteristics to improve the estimation of the dynamic model by emphasizing the frequencies that are least affected by noise during the estimation.



# Tutorial – Identifying Linear Models Using the GUI

---

About This Tutorial (p. 4-3)	Overview of the tutorial for estimating linear models from single-input/single-output (SISO) data
Preparing Data (p. 4-5)	How to load the sample MAT-file into the MATLAB® workspace, open the System Identification Tool GUI, import data into the GUI from the MATLAB workspace, and plot and preprocess the data
Saving the GUI Session (p. 4-20)	How to save a System Identification Tool session, including imported data sets and generated models
Identifying Models Using Default Settings (p. 4-22)	How to identify models using Quick Start to assess the complexity of the data and the performance of several polynomial and state-space models
Refining Models (p. 4-29)	How to get refine the model accuracy
Viewing Model Parameters (p. 4-45)	How to view estimated model parameters and the history of operations on the model and the corresponding data

Exporting the Model to the  
MATLAB® Workspace (p. 4-48)

How to make the model available  
to operations in the MATLAB  
Command Window for further  
processing with this toolbox or other  
MathWorks™ products

Exporting the Model to the LTI  
Viewer (p. 4-50)

How to export models to the LTI  
Viewer, which is available if you  
installed the Control System  
Toolbox™ product

## About This Tutorial

In this section...
“Objectives” on page 4-3
“Sample Data” on page 4-3

### Objectives

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

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

### Sample Data

The sample data is the MAT-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, much like a hair dryer. The input is the power supplied to the resistor wires, and the output is the air temperature at the outlet.

---

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

---

## Preparing Data

### In this section...

“Loading Data into the MATLAB® Workspace” on page 4-5

“Opening the System Identification Tool GUI” on page 4-5

“Importing Data Arrays into the System Identification Tool” on page 4-6

“Plotting and Preprocessing Data” on page 4-11

### Loading Data into the MATLAB® Workspace

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

```
load dryer2
```

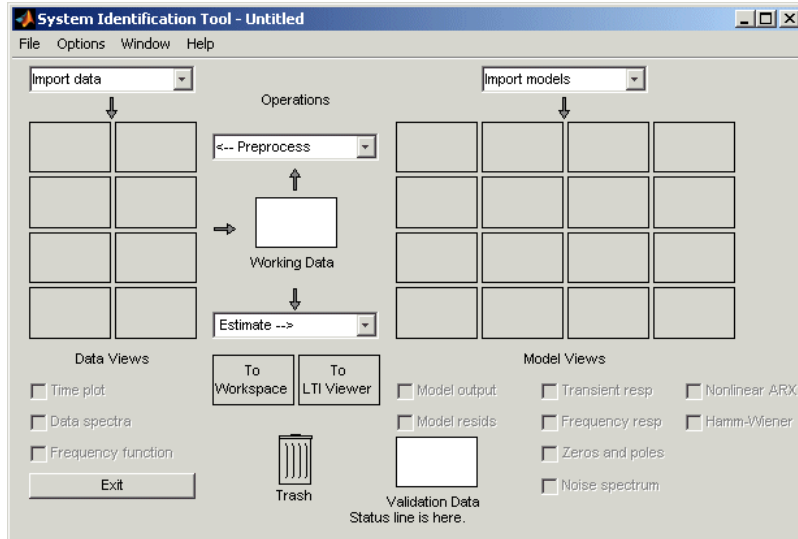
This command loads the data into the MATLAB workspace as two column vectors, `u2` and `y2`, respectively. The variable `u2` is the input data and `y2` is the output data.

### 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 opened the System Identification Tool GUI, as described in “Opening the System Identification Tool GUI” on page 4-5.

---

**Note** The input and output signals need not have the same number of data samples.

---

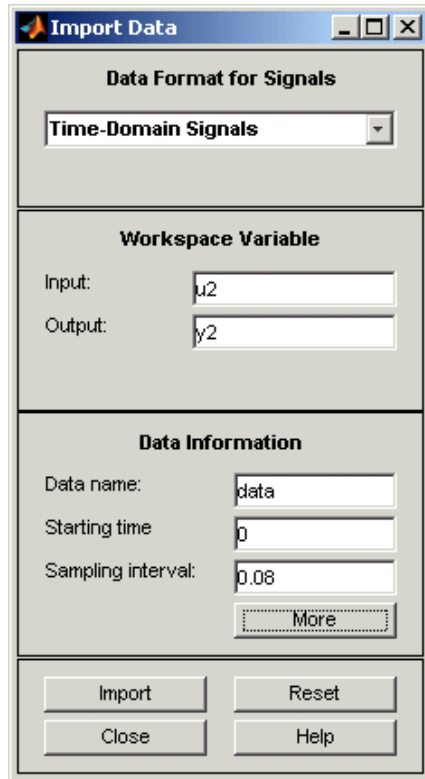


- 1 In the System Identification Tool GUI, select **Import data > Time domain data**. This action opens the Import Data dialog box.



- 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.

The screenshot shows the 'Import Data' dialog box with the following settings:

Section	Field	Value
Data Format for Signals	Format	Time-Domain Signals
	InterSample	zoh
Input Properties	Period	inf
	Input	u2
Workspace Variable	Output	y2
	Input	power
Channel Names	Output	temperature
	Data name	data
Data Information	Starting time	0
	Sampling interval	0.08
	Less	Less
Physical Units of Variables	Input	W
	Output	°C
Notes	Notes	

Buttons: Import, Reset, Close, Help

### Input Properties

- **InterSample** — Enter zoh (zero-order hold) to maintain a piecewise-constant input signal between samples. 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** — Inf specifies a nonperiodic input.

---

**Note** If your data is periodic, always include a whole number of periods for model estimation. For a periodic input, type the period 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.

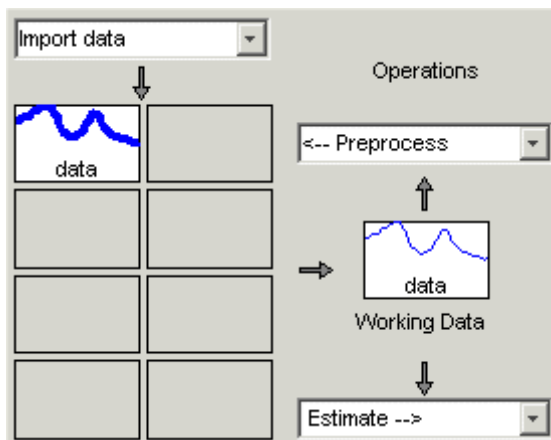
---

- **Output** — Enter °C 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 Preprocessing Data

In this portion of the tutorial, you examine the data and prepare 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 starting 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.

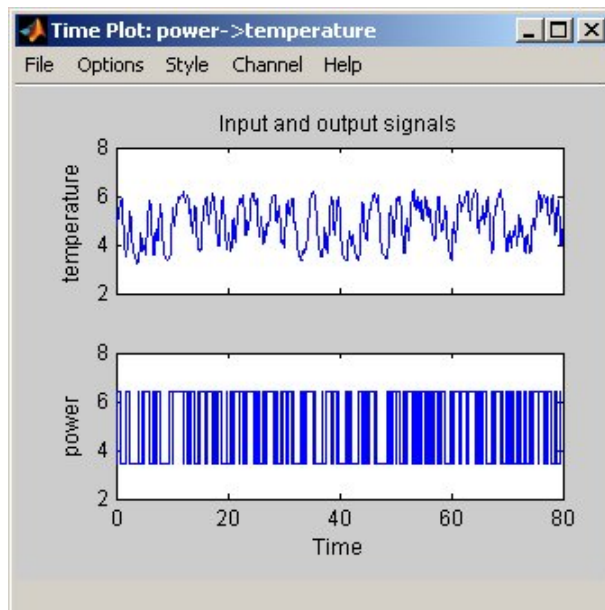
You must have already imported data into the System Identification Tool, as described in “Importing Data Arrays into the System Identification Tool” on page 4-6.

---

**Tip** For information about other types of preprocessing, such as resampling and filtering the data, see the topics about plotting and preprocessing data in the *System Identification Toolbox User’s Guide*.

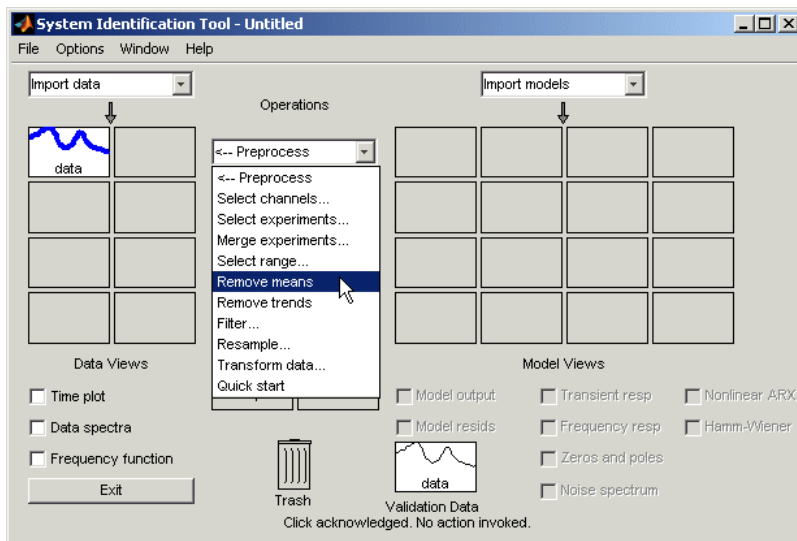
---

- 1 In the System Identification Tool GUI, select the **Time plot** check box to open the Time Plot.

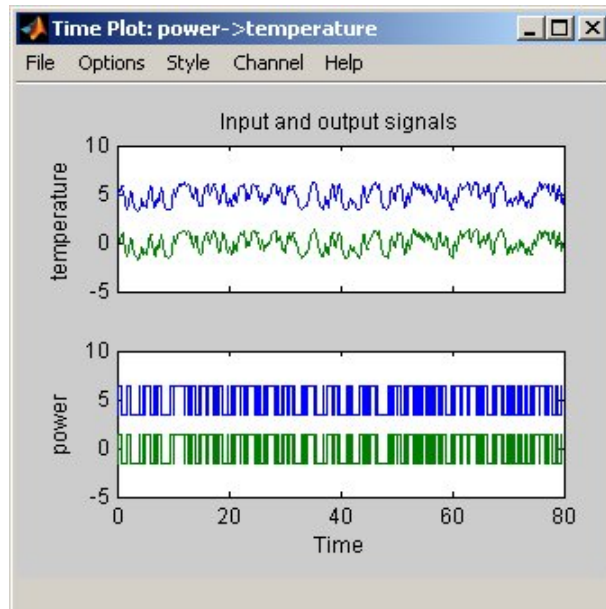


The top axes show the output data (temperature), and the bottom axes show the input data (power). Both the input and the output data have nonzero mean values.

- 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.

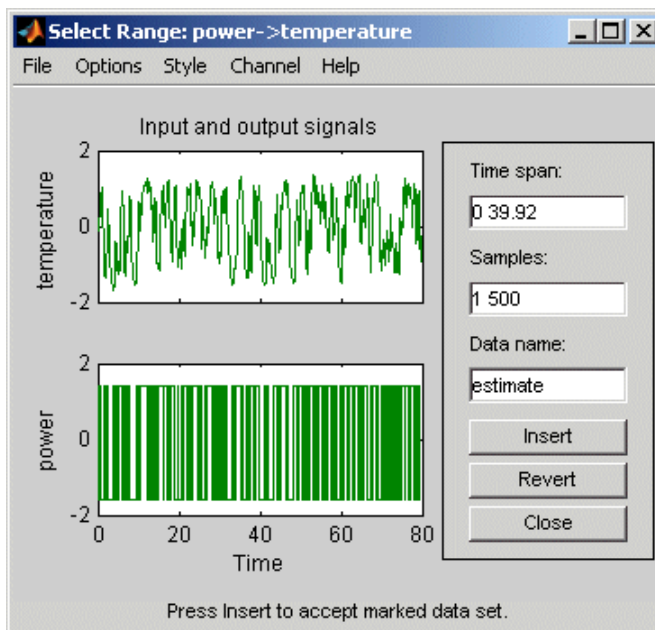
- 3 In the System Identification Tool GUI, drag the `datad` data set to the **Working Data** rectangle. This action specifies the detrended data to be used for estimating models.
- 4 Select **<-Preprocess > Select range** to open the Select Range window.

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



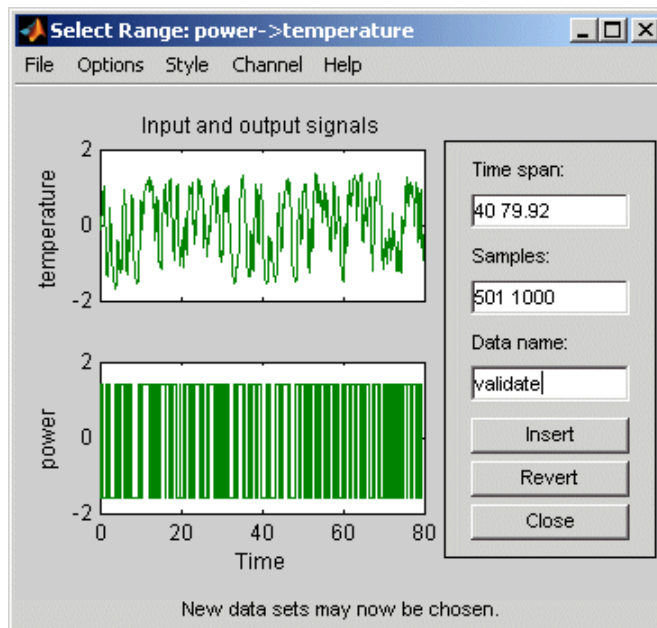
---

**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.

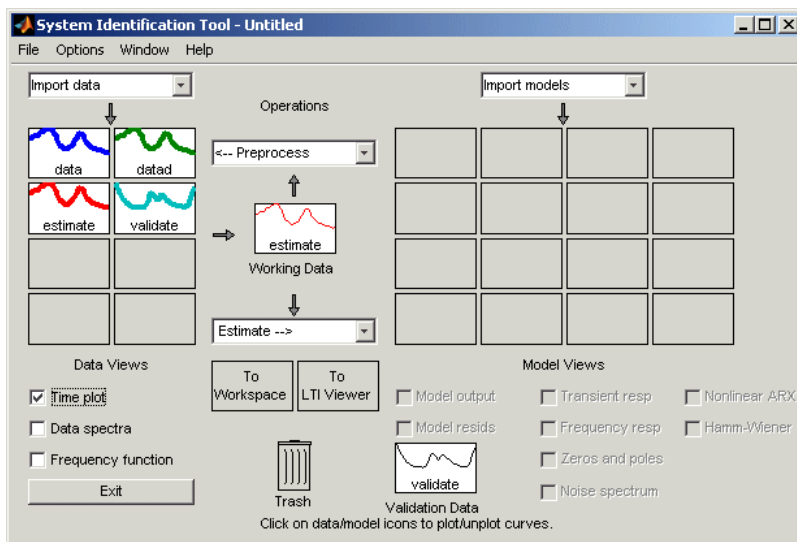
---

- 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.

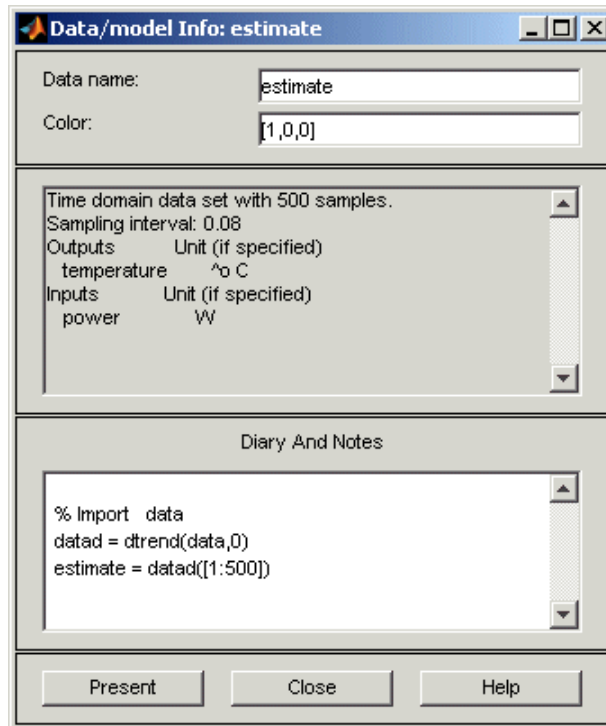



---

**Tip** If you have multiple data sets available from different experiments, you can use one data set for estimation and another data set for validation. Thus, you need not split the data set you originally imported.

---

- 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, the estimate data set is a result of importing the data, detrending the mean values, and selecting the first 500 samples of the data:

```
load dryer2
% Import data
datad = detrend(data,0)
estimate = datad([1:500])
```

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

---

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

---

For information about other types of preprocessing, such as resampling and filtering data, see the *System Identification Toolbox User's Guide*.

## Saving the GUI Session

After you preprocess the data, as described in “Plotting and Preprocessing Data” on page 4-11, you may 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.

In the following procedure, you delete the original data set `data` and the detrended data set `datad`, rearrange the data icons in the System Identification Tool GUI, and save the session.

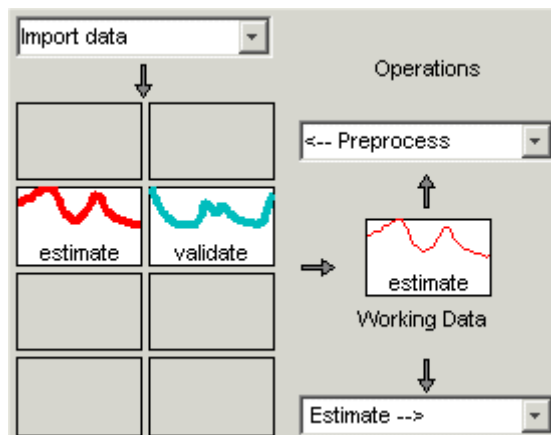
- 1 In the System Identification Tool GUI, drag and drop the `data` data set into the **Trash**.
- 2 Drag and drop the `datad` data set into the **Trash**.

---

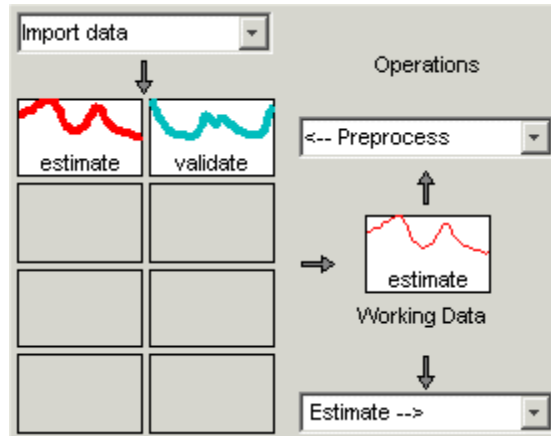
**Note** Moving items to the **Trash** does not delete them. To permanently delete items, select **Options > Empty trash** in the System Identification Tool GUI.

---

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 directory where you want to save the session file.
- 5** In the **File name** field, type the name of the session prep\_data, and click **Save**. The resulting file has a .sid extension.

---

**Tip** To open a saved session when starting the System Identification Tool, type the session name as an argument. For example:

```
ident('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*.

## Identifying Models Using Default Settings

In this section...
“Why Identify Models Using Default Settings?” on page 4-22
“Using Quick Start to Identify Preliminary Models” on page 4-23
“Validating Preliminary Models” on page 4-23
“Types of Models Generated by Quick Start” on page 4-27

### Why Identify Models Using Default Settings?

After preparing the data for estimation, as described in “Plotting and Preprocessing Data” on page 4-11, you can use the Quick Start feature of the System Identification Toolbox™ product to identify and compare several model structures. You can use these models to assess whether linear modeling is sufficient. These models also help you gain insight into the possible order and delays in the system.

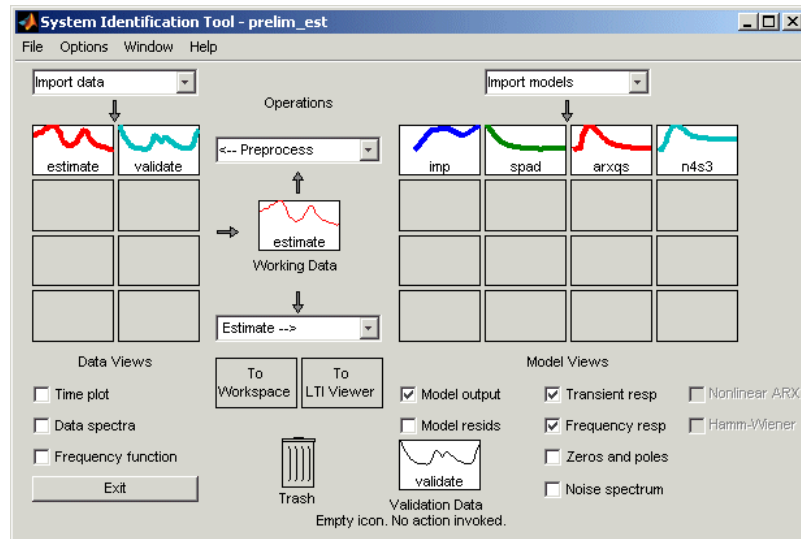


## Using Quick Start to Identify Preliminary Models

To identify preliminary models, select **Estimate > Quick start** in the System Identification Tool GUI.

This action generates plots of impulse response, frequency-response, and the output of state-space and polynomial models. For more information about these plots, see “Validating Preliminary Models” on page 4-23.

For a description of the generated models, see “Types of Models Generated by Quick Start” on page 4-27.



## Validating Preliminary Models

Estimating models using Quick Start generates the following three plots of the preliminary models you created in “Using Quick Start to Identify Preliminary Models” on page 4-23:

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

You can analyze these plots to determine the quality of the model. Close the model plots after you examine them.

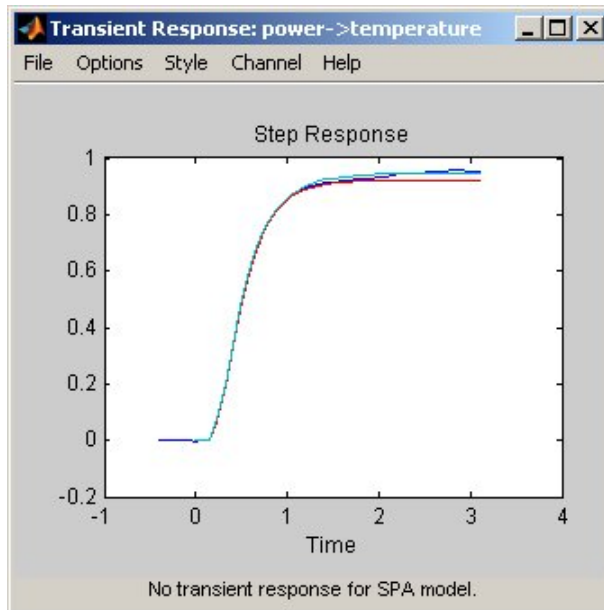
### Step-Response Plot

The following step-response plot shows agreement for the different model structures, 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.

---



### Step Response for imp, arxqs, and n4s3

---

**Note** The step-response plot does not include the frequency-response model, spad.

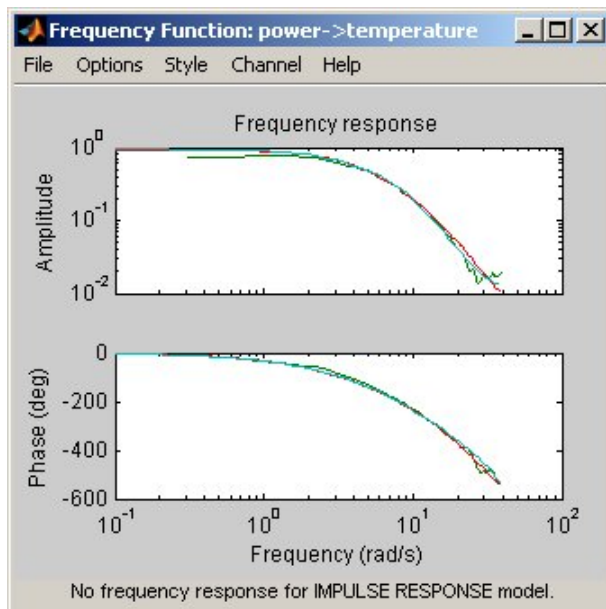
---

**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 for the different models, 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.



**Frequency Response for Models *spad*, *arxqs*, and *n4s3***

---

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

---

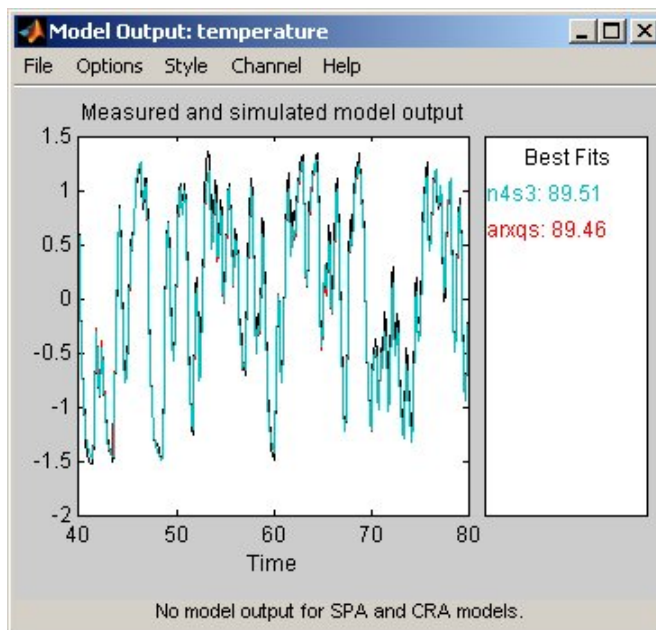
### Model-Output Plot

The Model Output window displays this model output together with 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.

---



**Measured Output and Model Output for Models `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.

---

## Types of Models Generated by Quick Start

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 using the impulse algorithm.

This model is nonparametric (not expressed in terms of parameters).

- `spad` — Frequency response using the spa algorithm. The frequency response is the Fourier transform of the impulse response of a linear system.

This model is nonparametric and computes the response for each frequency value. 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:

$$y(t) + a_1y(t-1) + \dots + a_{n_a}y(t-n_a) = b_1u(t-n_k) + \dots + b_{n_b}u(t-n_k-n_b+1) + e(t)$$

$y(t)$  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 *dead time*), and  $e(t)$  is the white-noise disturbance. The

System Identification Toolbox product estimates the parameters  $a_1 \dots a_n$  and  $b_1 \dots b_n$  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:

$$\begin{aligned} x(t+1) &= Ax(t) + Bu(t) + Ke(t) \\ y(t) &= Cx(t) + Du(t) + e(t) \end{aligned}$$

$y(t)$  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  $K$ .

## Refining Models

### In this section...

“Strategy for Refining Models” on page 4-29

“Estimating Possible Model Orders” on page 4-29

“Identifying State-Space and ARMAX Models” on page 4-34

“Choosing the Best Model” on page 4-38

### Strategy for Refining Models

Because the simple models in “Identifying Models Using Default Settings” on page 4-22 showed that a linear model sufficiently represents the dynamics of the system, it is worthwhile to refine these preliminary estimates and improve the model accuracy.

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 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, what model orders should you 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:

$$y(t) + a_1y(t-1) + \dots + a_{n_a}y(t-n_a) = b_1u(t-n_k) + \dots + b_{n_b}u(t-n_k-n_b+1) + e(t)$$

$y(t)$  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 the system output, and  $e(t)$  is the white-noise disturbance.

You must specify the model orders to estimate ARX models.

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

### 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 by default in the **Structure** list.

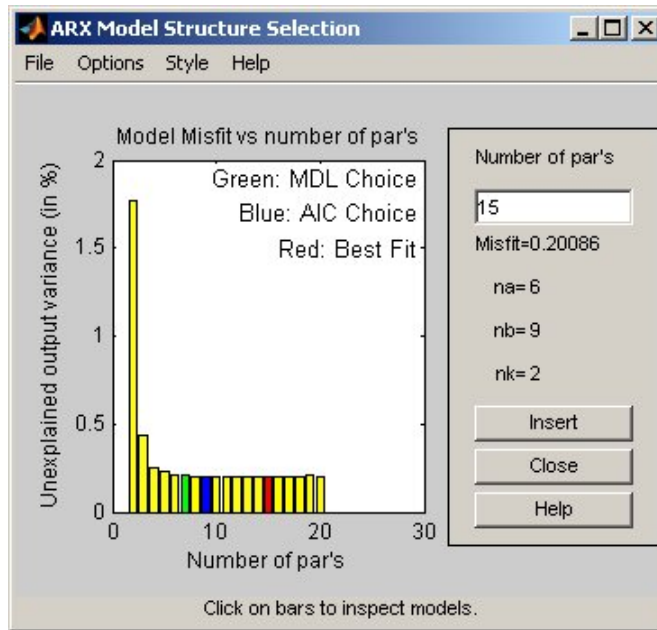


- 2** Edit the **Orders** field to try all combinations of poles, zeros, delays, where each value is from 1 to 10:

[1:10 1:10 1:10]

The screenshot shows the 'Linear Parametric Models' dialog box. The 'Orders' field is highlighted with a red box and contains the text '[1:10 1:10 1:10]'. Other fields include 'Structure' (ARX: [na nb nk]), 'Equation' (Ay=Bu+e), 'Method' (ARX selected), 'Name' (empty), 'Focus' (Prediction), 'Initial state' (Auto), 'Dist.model' (Estimate), and 'Covariance' (Estimate). Buttons for 'Trace', 'Stop iterations', 'Order Selection', 'Order Editor...', 'Estimate', 'Close', and 'Help' are also visible.

- 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 a specific number of parameters. The *prediction error* is the sum of the squares of the differences between the validation data output and the model 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.

---

- 4** In the ARX Model Structure Selection window, select 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, select the bar corresponding to 4 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 and ARMAX Models

By estimating ARX models for different order combinations, as described in “Estimating Possible Model Orders” on page 4-29, 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.

#### About State-Space Models

The general state-space model structure is:

$$\begin{aligned}x(t+1) &= Ax(t) + Bu(t) + Ke(t) \\ y(t) &= Cx(t) + Du(t) + e(t)\end{aligned}$$

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

You must specify a single integer as the model order 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  $n_k$  is the delay.

#### About ARMAX Models

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

$$y(t) + a_1y(t-1) + \dots + a_{n_a}y(t-n_a) = \\ b_1u(t-n_k) + \dots + b_{n_b}u(t-n_k-n_b+1) + \\ e(t) + c_1e(t-1) + \dots + c_{n_c}e(t-n_c)$$

$y(t)$  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 dead time (in terms of the number of samples) before the input affects output of the system, 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 parameters  $a_1 \dots a_n$ ,  $b_1 \dots b_n$ , and  $c_1 \dots c_n$  using the data and the model orders you specify.

### How to Estimate State-Space and ARMAX Models

To explore the state-space and ARMAX model structures:

- 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.

- 4 Click **Estimate** to add a state-space model called n4s6 to the System Identification Tool GUI.
- 5 From the **Structure** list, select ARMAX:  $[n_a \ n_b \ n_c \ n_k]$  to estimate an ARMAX model.

- 6** 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.

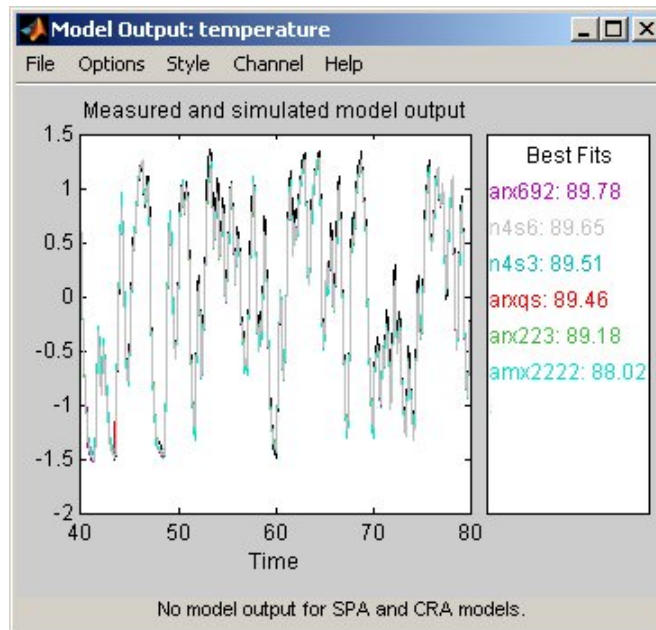
- 7** Click **Estimate** to add the ARMAX model to the System Identification Tool GUI.

---

**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.

---

The fit for amx2222 is about 1% lower than the other models.

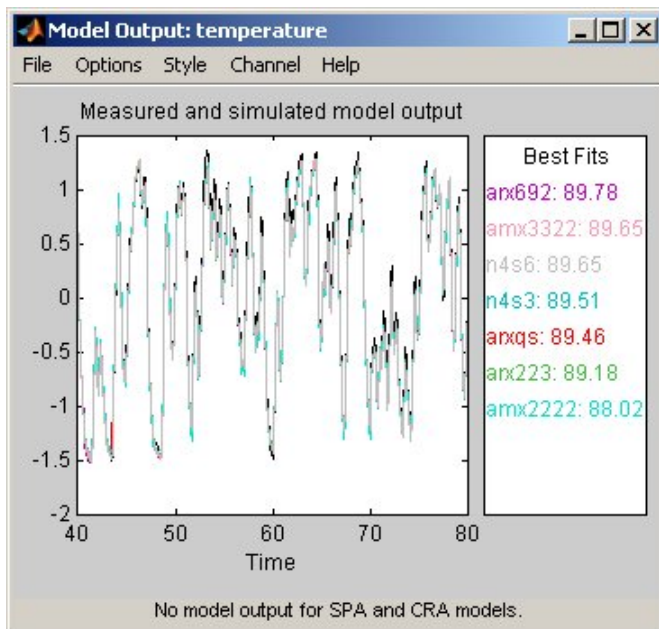


---

**Note** The **Best Fits** area in the Model Output window sorts models such that models with the best-fit model appear at the top.

---

- 8 Repeat steps 6 and 7 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 state-space models, see the corresponding section in the *System Identification Toolbox User's Guide*.

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

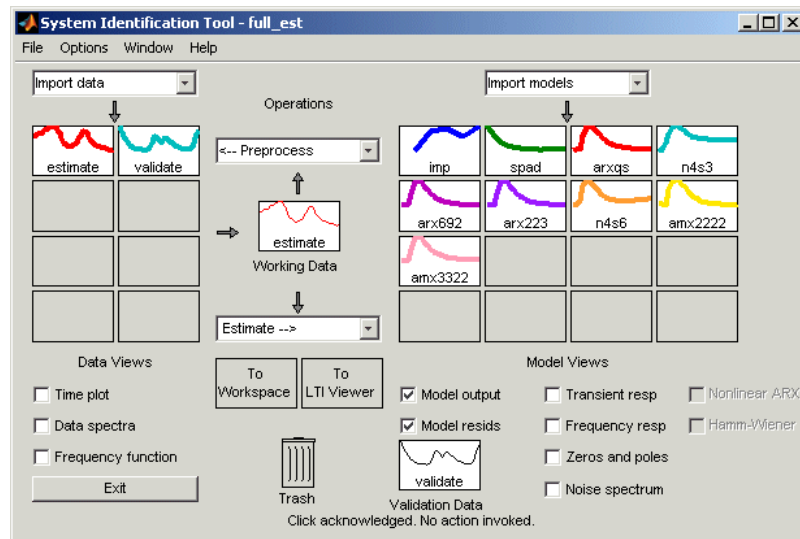
### Choosing the Best Model

- “Summary of Models” on page 4-39
- “Examining the Model Output” on page 4-39
- “Examining Model Residuals” on page 4-42



## Summary of Models

The following figure shows the System Identification Tool GUI, which includes all of the models you estimated so far.



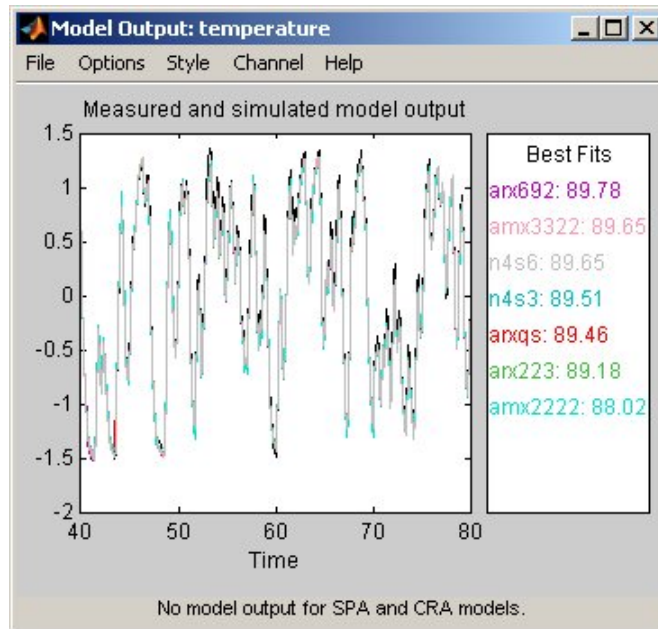
## Examining the Model Output

A good model is the simplest model that best explains the dynamics and successfully simulates or predicts the output for different inputs.

The Model Output window should be already open. It is automatically updated to include the new models you identified. Examine the model-output plot to see how well the model output matches the measured output in the validation data set.

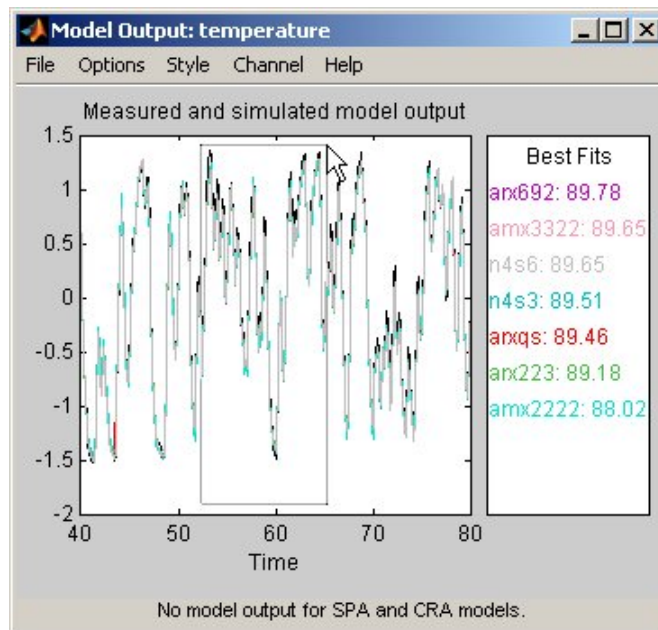
**Tip** If you closed the Model Output plot, you can regenerate it by selecting the **Model output** check box in the System Identification Tool GUI. If the new model does not appear on the plot, click the model icon in the System Identification Tool GUI to include this model on plots.

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.

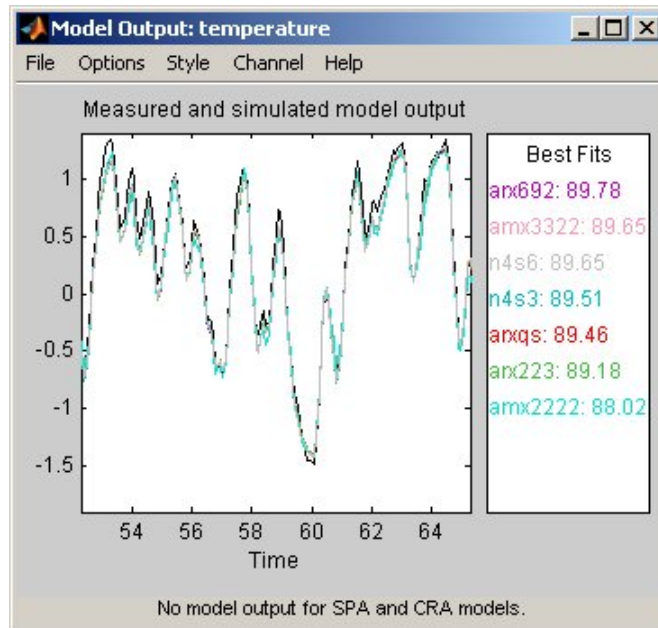


**Tip** To validate your models using a different data set, you can drag and drop this data set into the **Validation Data** rectangle in the System Identification Tool GUI. This action automatically updates any open model views. If you transform validation data into the frequency domain, the model-output 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.



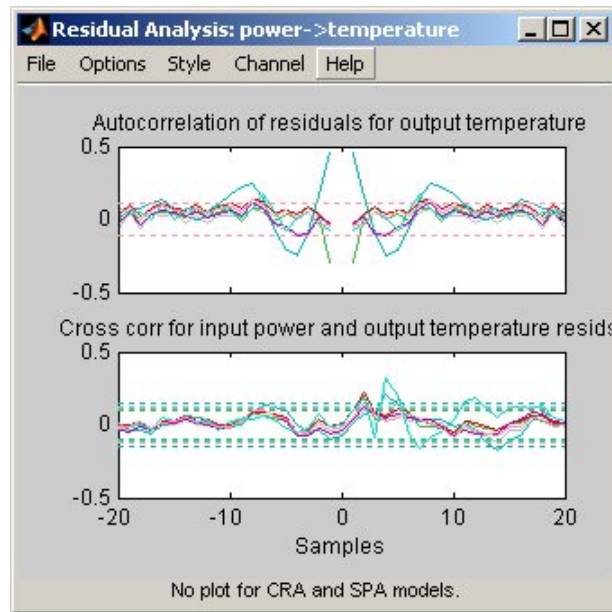
Releasing the mouse magnifies this region and shows that all models seem to agree with the validation data.



### Examining Model Residuals

In addition to comparing model output to measured output, you can validate a model by checking the behavior of its residuals.

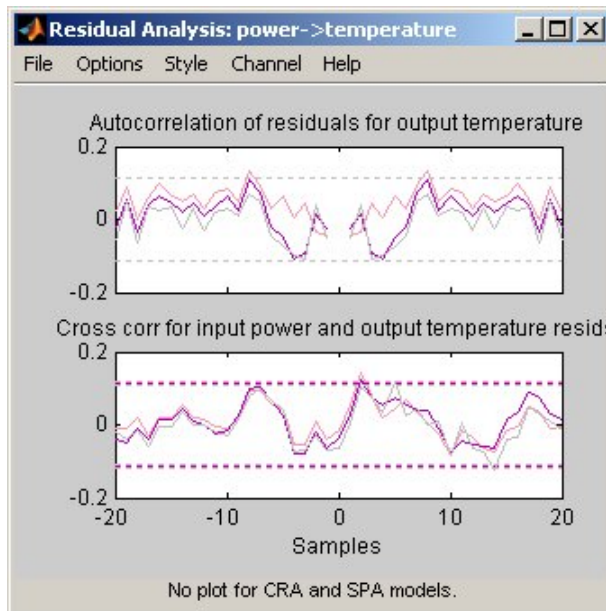
To generate a residual analysis plot, select the **Model resids** check box in the System Identification Tool GUI.



The top axes show the autocorrelation of residuals for the temperature output (whiteness test). The horizontal scale is the number of lags, or the difference between the time steps that are correlated. The horizontal dashed lines on the plot represent the model confidence interval. Any fluctuations within the confidence interval are considered to be insignificant. Two of the models,  $n4s3$  and  $arx223$ , produce residuals that enter outside the confidence interval. A good model should have a residual autocorrelation function within the confidence interval, indicating that the residuals are uncorrelated.

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. The models  $arxqs$  and  $amx222$  extend beyond the confidence interval and do not perform as well as the other models.

Click the model icons `n4s3`, `arx223`, `arxqs`, and `amx2222` in the System Identification Tool GUI to remove them from the Residual Analysis plot. The Residual Analysis plot now includes only the three models that pass the residual tests: `arx692`, `n4s6`, and `amx3322`.



The plots for these models fall within the confidence intervals. Therefore, it is reasonable to pick the simpler, low-order model `amx3322` as the final choice. The `amx3322` output agrees well with the measured output.

## Viewing Model Parameters

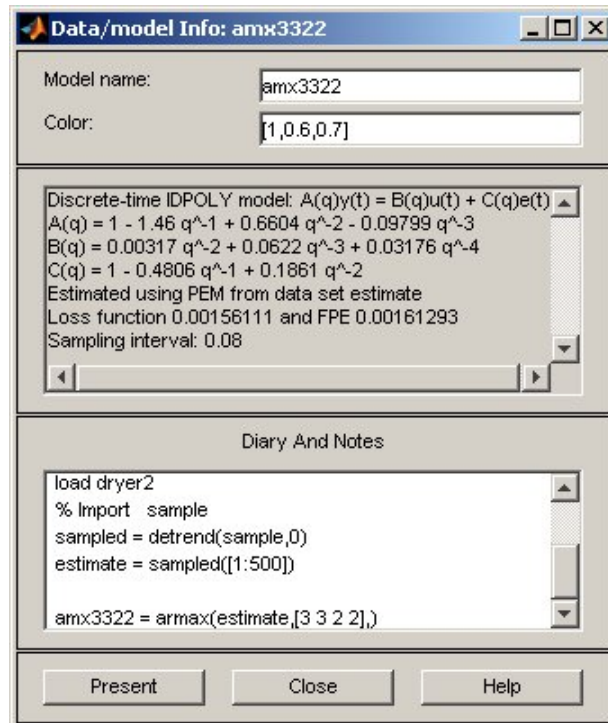
### In this section...

“Viewing Model Parameter Values” on page 4-45

“Viewing Parameter Uncertainties” on page 4-47

## Viewing Model Parameter Values

You can view the numerical parameter values of the model amx3322 by right-clicking 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:

$$A(q) = 1 - 1.46q^{-1} + 0.6604q^{-2} - 0.09799q^{-3}$$

$$B(q) = 0.00317q^{-2} + 0.0622q^{-3} + 0.03176q^{-4}$$

$$C(q) = 1 - 0.4806q^{-1} + 0.1861q^{-2}$$

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

$$\begin{aligned} y(t) - 1.46y(t-1) + 0.6604y(t-2) - 0.09799y(t-3) = \\ 0.00317u(t-2) + 0.622u(t-3) + 0.03176u(t-4) + \\ e(t) - 0.4806e(t-1) + 0.1861e(t-2) \end{aligned}$$

---

**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:

$$A(q) = 1 + a_1q^{-1} + \dots + a_naq^{-na}$$

$$B(q) = b_1q^{-1} + \dots + b_n bq^{-nb}$$

$$C(q) = 1 + c_1q^{-1} + \dots + c_n c q^{-nc}$$



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:

$$\begin{aligned} y(t) + a_1y(t-1) + \dots + a_{n_a}y(t-n_a) = \\ b_1u(t-n_k) + \dots + b_{n_b}u(t-n_k-n_b+1) + \\ e(t) + c_1e(t-1) + \dots + c_{n_c}e(t-n_c) \end{aligned}$$

$y(t)$  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 dead time (in terms of the number of samples) before the input affects output of the system, and  $e(t)$  is the white-noise disturbance.

## Viewing Parameter Uncertainties

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

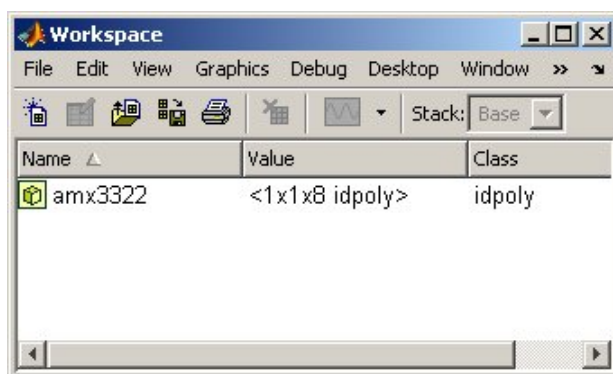
$$\begin{aligned} A(q) &= 1 - 1.46(+0.06003)q^{-1} \\ &\quad + 0.6604(+0.08906)q^{-2} \\ &\quad - 0.09799(+0.03519)q^{-3} \\ B(q) &= 0.00317(+0.001646)q^{-2} \\ &\quad + 0.0622(+0.002425)q^{-3} \\ &\quad + 0.03176(+0.005629)q^{-4} \\ C(q) &= 1 - 0.4806(+0.07558)q^{-1} \\ &\quad + 0.1861(+0.05597)q^{-2} \end{aligned}$$

The 1-standard-deviation uncertainty for each set of 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.

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 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.

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

---

About This Tutorial (p. 5-3)	Overview of the tutorial for estimating continuous-time transfer functions from single-input/single-output (SISO) data
What Is a Continuous-Time Process Model? (p. 5-5)	Description of the linear, continuous-time transfer function (process model)
Preparing Data (p. 5-6)	How to load the sample MAT-file into the MATLAB® workspace, open the System Identification Tool GUI, import a data object into the System Identification Tool from the MATLAB workspace, and plot and preprocess the data
Estimating Second-Order Transfer Functions (Process Models) with Complex Poles (p. 5-13)	How to estimate a second-order transfer function with complex poles (underdamped modes)
Refining the Model by Including a Noise Model (p. 5-22)	How to improve the model by modifying the estimation algorithm

Viewing Model Parameters (p. 5-26)	How to view estimated model parameters and the history of operations on the model and the corresponding data
Exporting the Model to the MATLAB® Workspace (p. 5-29)	How to make the model available to operations in the MATLAB Command Window for further processing with this toolbox or other MathWorks™ products
Using the System Identification Toolbox™ Product with the Simulink® Software (p. 5-30)	How to create and simulate a System Identification Toolbox™ model using the Simulink® software

## About This Tutorial

In this section...
“Objectives” on page 5-3
“Sample Data” on page 5-3

### Objectives

Estimate and validate simple, continuous-time transfer functions 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.
- Plot and preprocess data.
- Estimate and validate low-order, continuous-time models from the data.
- Export models to the MATLAB workspace.
- Export models to the LTI Viewer for linear analysis (requires the Control System Toolbox™ product).

### Sample Data

The sample data you use in this tutorial is in `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 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.

---

**Note** This tutorial uses time-domain data to demonstrate how you can estimate linear models. However, the same workflow applies to frequency-domain data.

---



## What Is a Continuous-Time Process Model?

*Continuous-time process models* are low-order transfer functions that describe the system dynamics using static gain, 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 adding or removing 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

In this section...
“Loading Data into the MATLAB® Workspace” on page 5-6
“Opening the System Identification Tool GUI” on page 5-6
“Importing Data Objects into the System Identification Tool” on page 5-7
“Plotting and Preprocessing Data” on page 5-9

### Loading Data into the MATLAB® Workspace

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

```
load proc_data
```

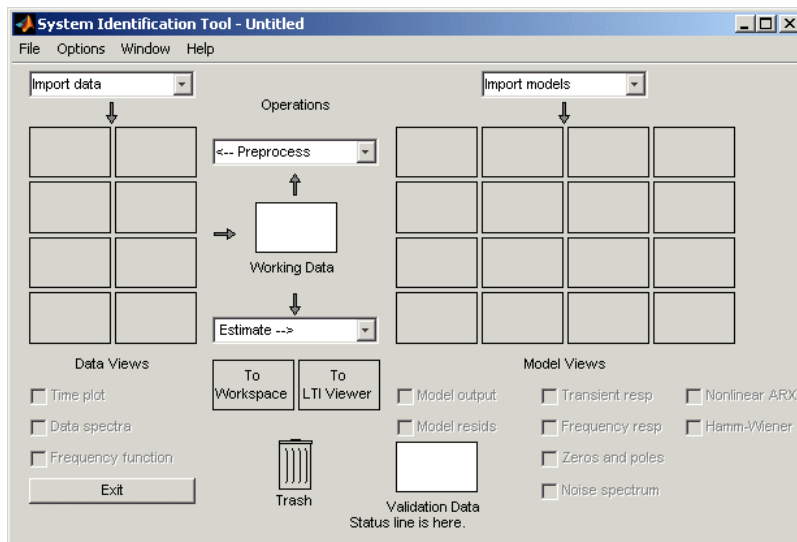
This command loads the data into the MATLAB workspace as the data object `z`. For more information about `iddata` objects, see the corresponding reference page.

### 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 the data objects into the GUI from the MATLAB workspace. You must have already opened the GUI, as described in “Opening the System Identification Tool GUI” on page 5-6.

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

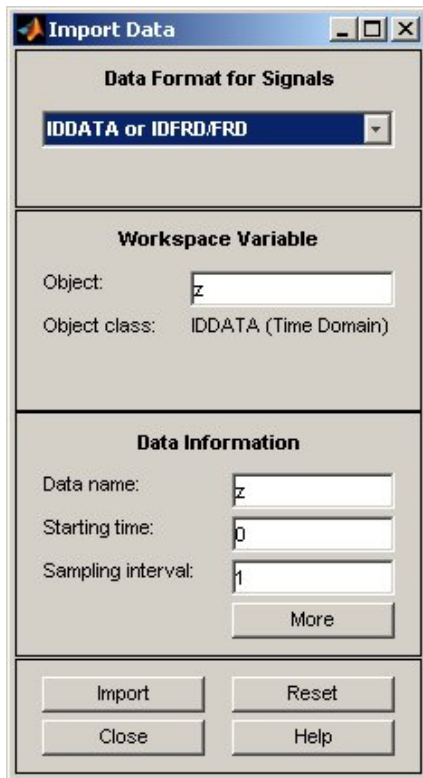
This action opens the Import Data dialog box.



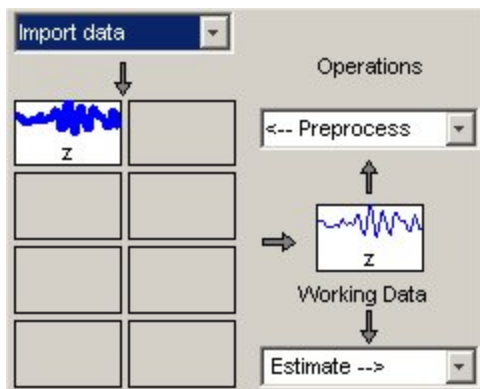
2 Specify the following options:

- **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.



- 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 Preprocessing Data

In this portion of the tutorial, you examine the data and prepare 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 starting 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.

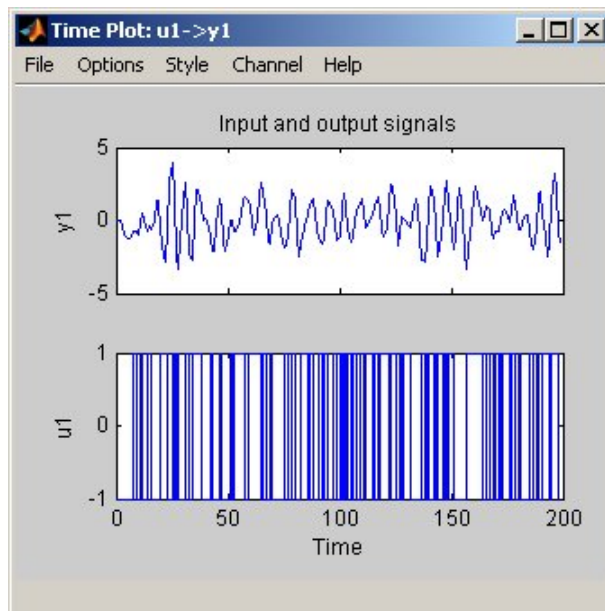
You must have already imported data into the System Identification Tool, as described in “Importing Data Objects into the System Identification Tool” on page 5-7.

---

**Tip** For information about other types of preprocessing, such as resampling and filtering the data, see the topics on plotting and preprocessing data in the *System Identification Toolbox User's Guide*.

---

- 1 In the System Identification Tool GUI, select the **Time plot** check box to open the Time Plot window.



The bottom axes show the simulated input data—a random binary sequence, and the top axes show the simulated 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**.

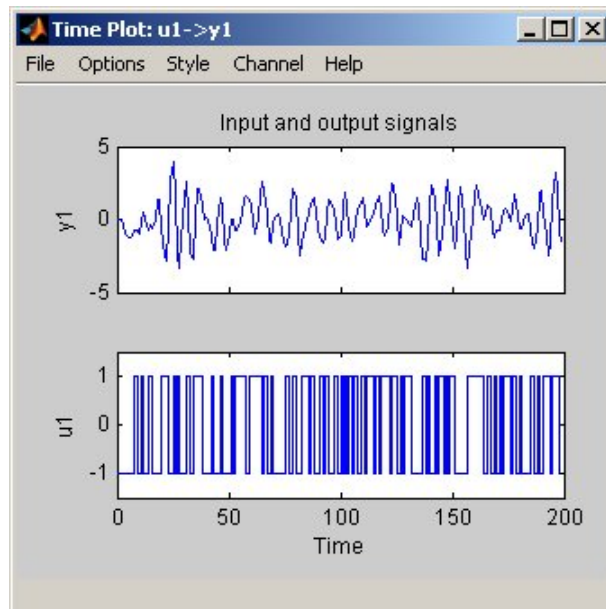


---

**Note** The other two fields, **Time** and **y1**, let you set the axis limits for the time axis and the output channel axis, respectively. In the Limits for Time Plot dialog box, you can also modify each axis to be logarithmic or linear.

---

The following figure shows the time plot.



- 4 In the System Identification Tool GUI, select **<-Preprocess > Quick start** to simultaneously 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**.



## Estimating Second-Order Transfer Functions (Process Models) with Complex Poles

### In this section...

“Estimating a Second-Order Transfer Function Using Default Settings” on page 5-13

“Tips for Specifying Known Parameters” on page 5-18

“Validating the Model” on page 5-18

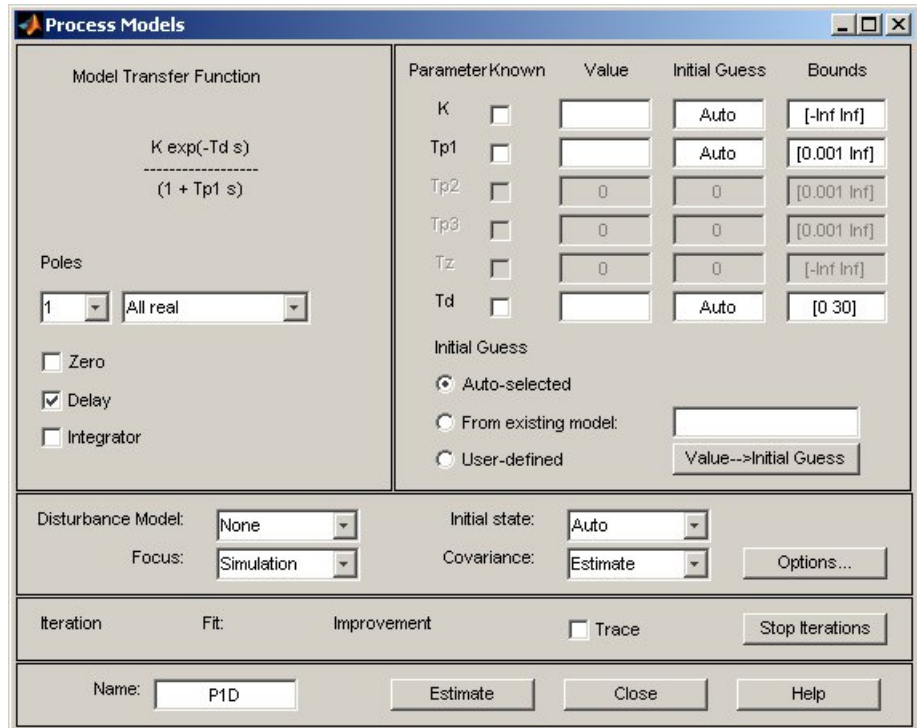
### Estimating a Second-Order Transfer Function Using Default Settings

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

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

You must have already prepared the data for estimation, as described in “Plotting and Preprocessing Data” on page 5-9.

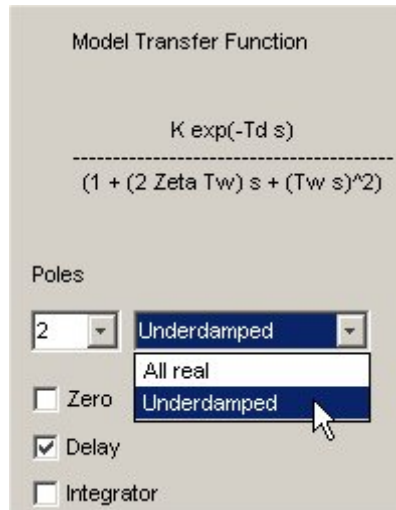
- 1 In the System Identification Tool GUI, select **Estimate > Process models** to open the Process Models dialog box.



**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,  $T_w$ , Zeta, and  $T_d$ . By default, the model **Name** is set to the acronym P2DU, which indicates two poles (P2), the presence of a delay (D), and underdamped modes (U).

---

**Note** You can edit the model name. Choose a model name that is unique in the System Identification Tool GUI.

---

- 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, select Auto-selected to calculate the initial parameter values during the estimation. The **Initial Guess** column in the Parameter table displays Auto.

Parameter	Known	Value	Initial Guess	Bounds
K	<input type="checkbox"/>	<input type="text"/>	Auto	[-Inf Inf]
Tw	<input type="checkbox"/>	<input type="text"/>	Auto	[0.001 Inf]
Zeta	<input type="checkbox"/>	<input type="text"/>	Auto	[0.001 Inf]
Tp3	<input type="checkbox"/>	0	0	[0.001 Inf]
Tz	<input type="checkbox"/>	0	0	[-Inf Inf]
Td	<input type="checkbox"/>	<input type="text"/>	Auto	[0 30]

Initial Guess

Auto-selected  
 From existing model:   
 User-defined:  Value-->Initial Guess

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

When you know the range of possible values for a parameter, type these values into the corresponding **Bounds** fields to help the estimation algorithm.

- 5 Keep the defaults for the estimation algorithm settings:
- **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 data that has a high signal-to-noise ratio and for when you plan to use your model for simulation applications. If your system contains significant noise and you want to either model the noise or improve parameter estimates using the noise model, then 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 on plots as model confidence regions.
- 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, type this value in the **Initial Guess** column.

When you know the value of a parameter approximately, you can help the estimation algorithms 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  $T_d$  at 2s, you can type the value into **Value** field of the Parameter table in the Process Models dialog box and select the corresponding **Known** check box.

Parameter	Known	Value	Initial Guess	Bounds
K	<input type="checkbox"/>		Auto	[-Inf Inf]
$T_w$	<input type="checkbox"/>		Auto	[0.001 Inf]
Zeta	<input type="checkbox"/>		Auto	[0.001 Inf]
$T_{p3}$	<input type="checkbox"/>	0	0	[0.001 Inf]
$T_z$	<input type="checkbox"/>	0	0	[-Inf Inf]
$T_d$	<input checked="" type="checkbox"/>	2	2	[0 30]

Initial Guess

Auto-selected  
 From existing model:   
 User-defined

## Validating the Model

In this portion of the tutorial, you generate the following two plots to examine the model you created in “Estimating a Second-Order Transfer Function Using Default Settings” on page 5-13:

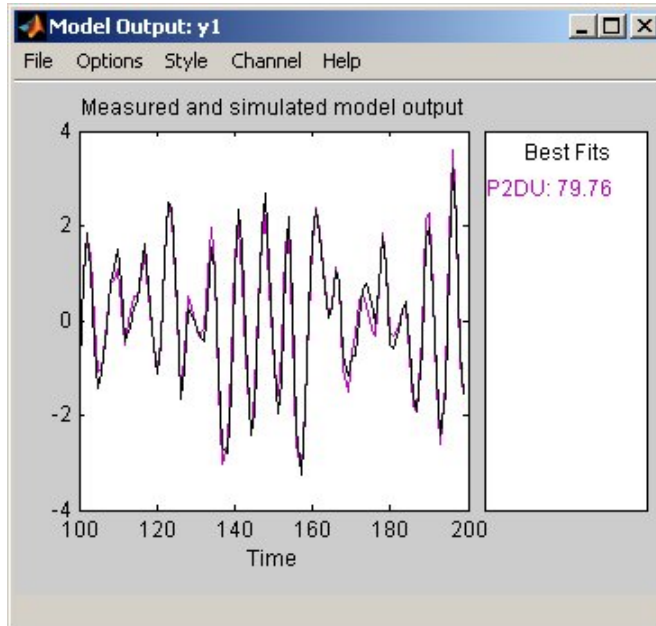
- Comparison of the model output and the measured output on a time plot

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

### Examining Model Output

A good model is the simplest model that best explains the dynamics and successfully simulates or predicts the output for different inputs. Use the model-output plot to check how well the model's output matches the measured output in the validation data set.

To generate the model-output plot, select the **Model output** check box in the System Identification Tool GUI.



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 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 sample data is simulated using the following second-order system with underdamped modes (complex poles), and has a peak response at 1 rad/s:

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

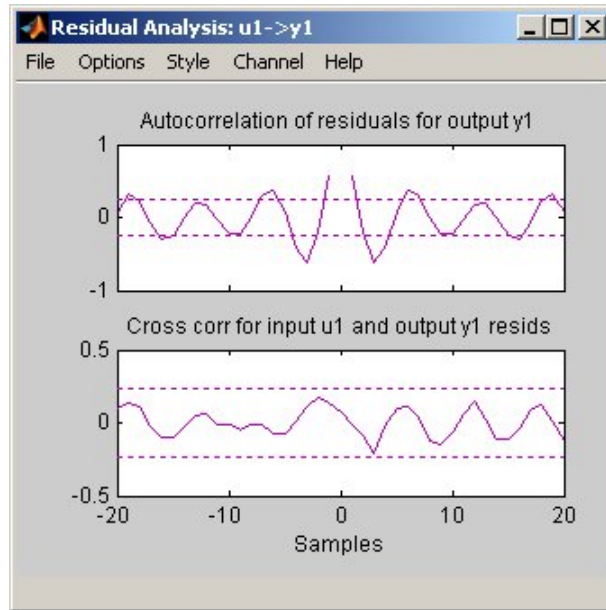
Because the data includes noise at the input during the simulation, the estimated model cannot exactly reproduce the model used to simulate the data.

### Examining Model Residuals

In addition to comparing model output to measured output, 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, or the difference between the time steps that are correlated. 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.

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.

## Refining the Model by Including a Noise Model

### In this section...

“Estimating Models with Modified Settings” on page 5-22

“Comparing Models” on page 5-23

### Estimating Models with Modified Settings

In this portion of the tutorial, you modify the estimation algorithm and include a noise model to improve the model results.

---

**Note** The Process Models dialog box should still be open. If you closed it, repeat the procedure in “Estimating a Second-Order Transfer Function Using Default Settings” on page 5-13.

---

- 1 In the Process Models dialog box, modify the following settings:
  - **Focus** — Set to **Prediction** to specify that the estimation algorithm use the noise model to weigh the relative importance of how closely to fit the data in various frequency ranges. The presence of (high-frequency) noise results in the algorithm assigning less importance to fitting the high-frequency portions of the data.
  - **Disturbance Model** — Set to **Order 1** to estimate a noise model  $H$  as a continuous-time, first-order ARMA model:

$$y = Gu + He$$

where  $e$  is white noise.

- **Name** — Edit the model name to **P2DUe1** to generate a model with a unique name in the System Identification Tool GUI.
- 2 Click **Estimate**.
  - 3 In the Process Models dialog box, set the **Disturbance Model** to **Order 2** to estimate a second-order noise model.

**4** Edit the **Name** field to P2DUe2 to generate a model with a unique name in the System Identification Tool GUI.

**5** Click **Estimate**.

## Comparing Models

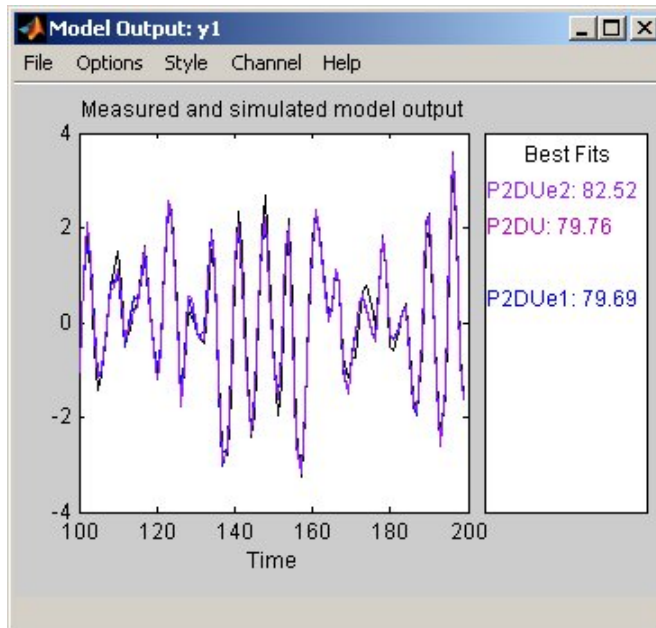
The Model Output and the Residual Analysis windows dynamically update to include the two new models. In this portion of the tutorial, you use these plots to compare the estimated models.

---

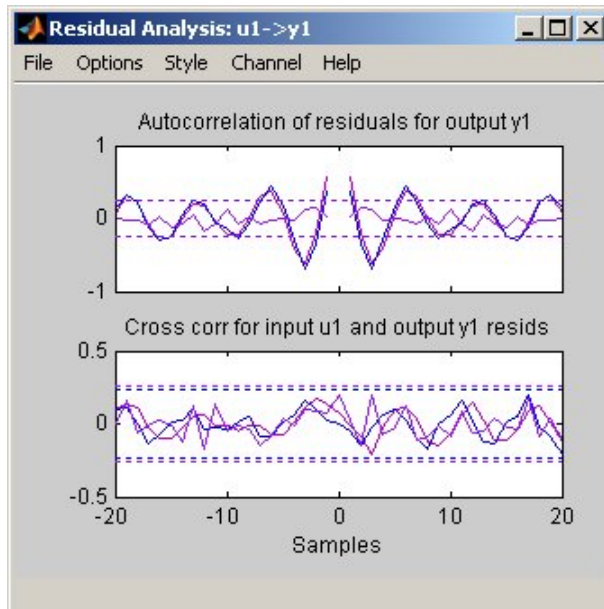
**Note** If you closed these plots, you can reopen them by selecting the **Model output** and the **Model resids** check boxes in the System Identification Tool GUI.

---

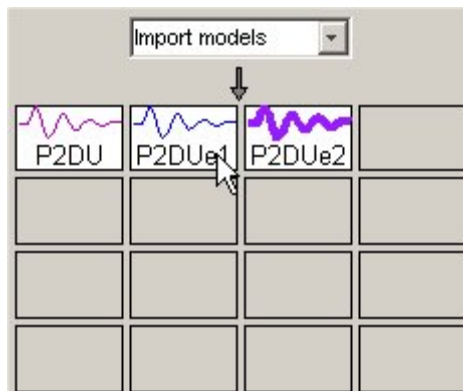
The following Model Output plot shows that the P2DUe2 model has a better performance than the other two models. All three models show agreement with the validation-data output.



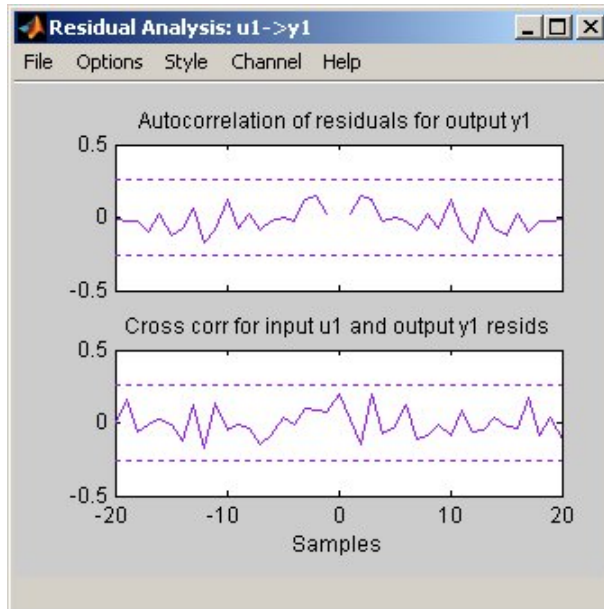
Furthermore, P2DUe2 falls well within the confidence bounds on the Residual Analysis plot.



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

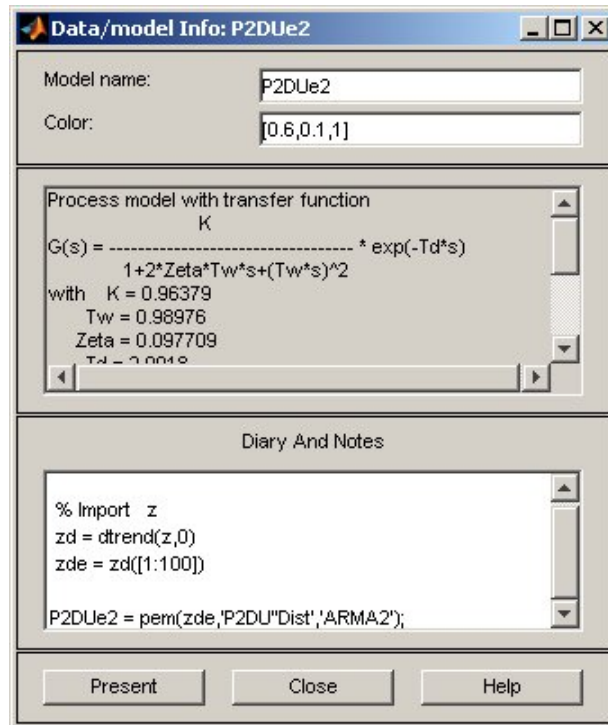
### In this section...

“Viewing Model Parameter Values” on page 5-26

“Viewing Parameter Uncertainties” on page 5-27

### 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:

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

For the model P2DUe2:

- K is 0.96379.
- Tw is 0.98976.
- Zeta is 0.097709.
- Td is 2.0018.

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:

$$y = Gu + He$$

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

$$\begin{aligned} C(s) &= s^2 + 2.03(+0.06772)s + 2.621(+0.3984) \\ D(s) &= s^2 + 0.2123(+0.07437)s + 1.113(+0.07804) \end{aligned}$$

The 1-standard-deviation uncertainty for each set of 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.

$$\begin{aligned} K &= 0.96379+0.018245 \\ T_w &= 0.98976+0.0055579 \end{aligned}$$

Zeta = 0.097709+-0.0064056  
Td = 2.0018+-0.0025342

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, drag it to the **To Workspace** rectangle in the System Identification Tool GUI. 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.

---

## Using the System Identification Toolbox™ Product with the Simulink® Software

In this section...
“Preparing Input Data” on page 5-30
“Building the Simulink® Model” on page 5-30
“Configuring Blocks and Simulation Parameters” on page 5-32
“Running the Simulation” on page 5-36

### Preparing Input Data

You can 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.

You must have completed the previous steps in this tutorial to make these variables available in the MATLAB® workspace.

---

**Note** Simulink must be installed to build a Simulink model.

---

Because you only need the input channel of  $z$  for providing input to the model, type the following in the MATLAB Command Window:

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

### Building the Simulink® Model

The following steps guide you through the process of adding blocks to a Simulink model. For more information about working with Simulink models, see the Simulink documentation.

- 1 In the MATLAB Command Window, type `simulink` in the MATLAB Command Window.

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

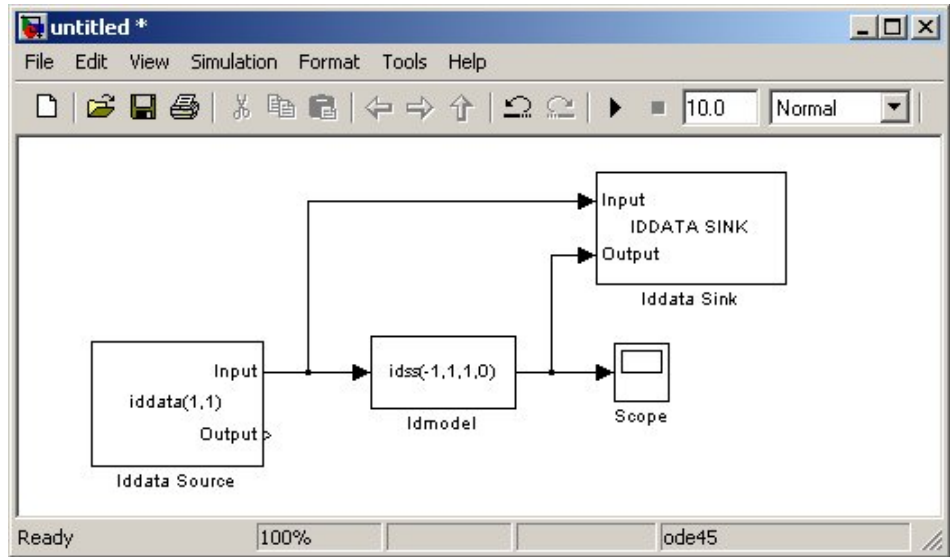
---

**Tip** An alternative 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 window, 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.

## Configuring Blocks and Simulation Parameters

The following procedure guides you through the following tasks that configure the model blocks:

- Get data from the MATLAB workspace.
- Set the simulation time interval and duration.

- 1 In the new model window, select **Simulation > Configuration Parameters**.

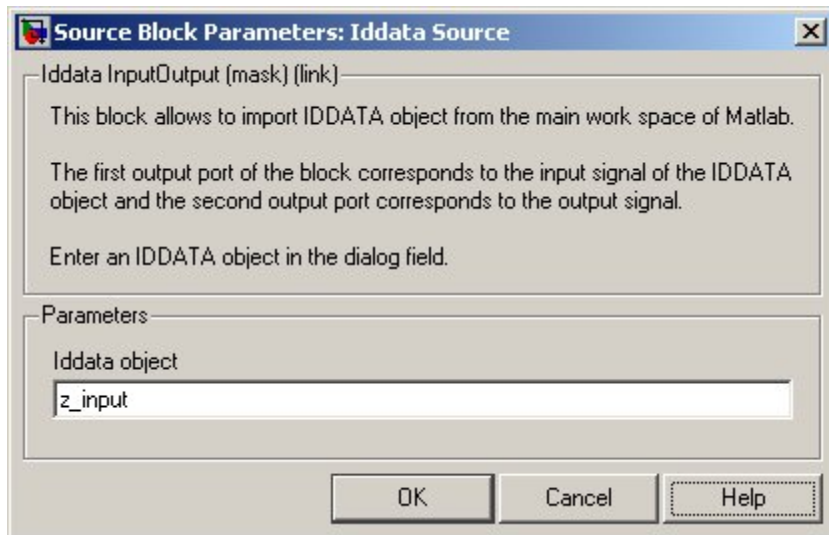
- 2 In the Configuration Parameters dialog box, type 200 in the **Stop time** field. Click **OK**.

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. Next, 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.

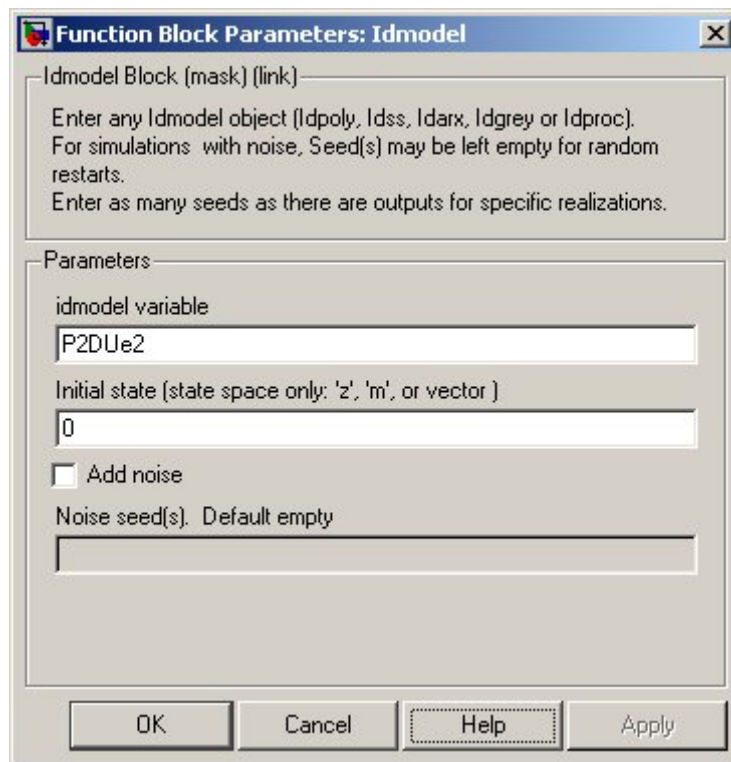
---

- 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 system. Click **OK**.

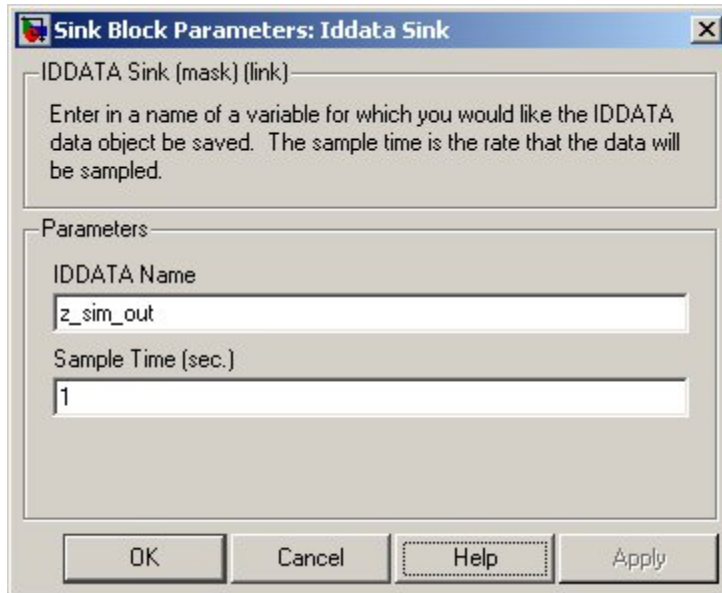
When **Add noise** is selected, Simulink derives the noise amplitude from the `NoiseVariance` property of the model and adds noise to the model accordingly. The simulation propagates this noise according to the noise model  $H$  that was estimated with the system dynamics:

$$y = Gu + He$$

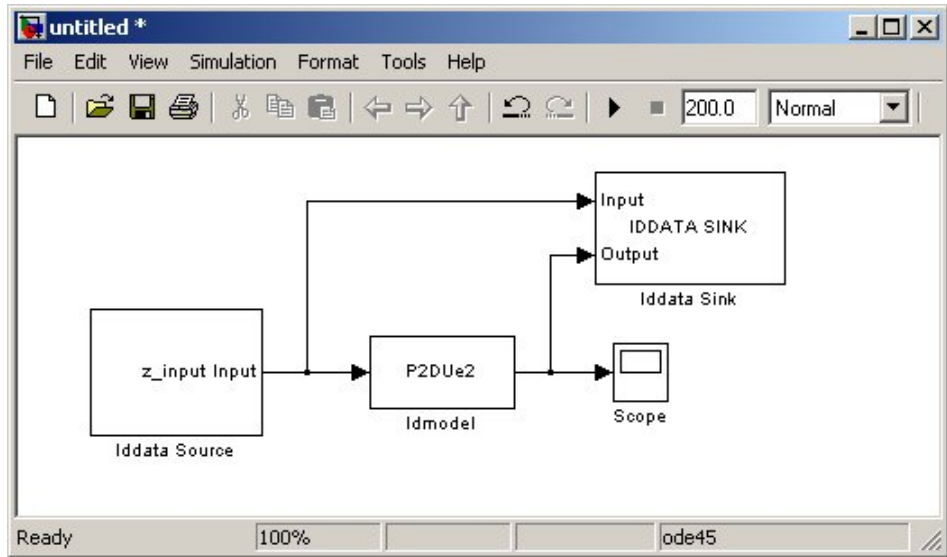
- 7 Double-click the Iddata Sink block to open the Sink Block Parameters: Iddata Sink dialog box. Type the following variable name in the **IDDATA Name** field:

`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**.



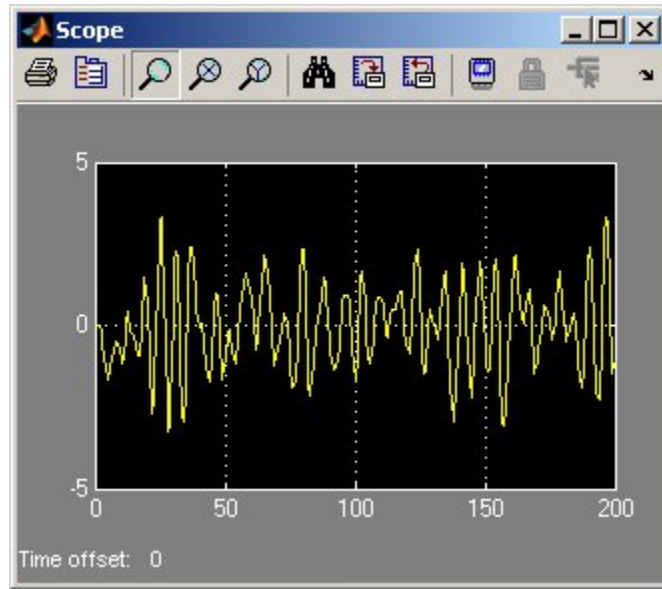
The resulting change to the Simulink model is shown in the following figure.



### 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.



# Tutorial – Identifying Linear Models Using the Command Line

---

About This Tutorial (p. 6-3)	Overview of the tutorial for estimating linear models from multiple-input/single-output (MISO) data using the System Identification Toolbox™ objects and methods
Preparing Data (p. 6-5)	How to load the sample MAT-file into the MATLAB® workspace, plot the data, and subtract equilibrium values from the data
Estimating Step- and Frequency-Response Models (p. 6-17)	How to estimate impulse-response and frequency-response models to analyze the dynamic characteristics of the system
Estimating Delays in the Multiple-Input System (p. 6-22)	How to estimate input/output delays in the system using an ARX polynomial model
Estimating Model Orders Using an ARX Model Structure (p. 6-25)	How to estimate model orders using an ARX polynomial model
Estimating Continuous-Time Transfer Functions (Process Models) (p. 6-33)	How to estimate and validate a linear, continuous-time transfer function (process model) both with and without noise

Estimating Black-Box Polynomial  
Models (p. 6-44)

How to estimate and validate ARX,  
state-space, and Box-Jenkins models

Simulating and Predicting Model  
Output (p. 6-56)

How to simulate and predict model  
output using `sim` and `predict`,  
respectively

## About This Tutorial

In this section...
“Objectives” on page 6-3
“Sample Data” on page 6-3

### Objectives

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

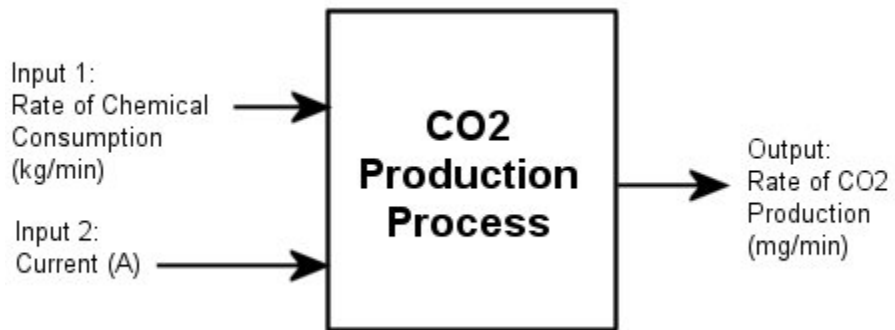
After completing this tutorial, you will be able to accomplish the following tasks using the command line:

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

### Sample Data

The sample data is the MAT-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.



---

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

---

## Preparing Data

### In this section...

- “Loading Data into the MATLAB® Workspace” on page 6-5
- “Plotting the Input/Output Data” on page 6-6
- “Removing Equilibrium Values from the Data” on page 6-7
- “Using Objects to Represent Data for System Identification” on page 6-8
- “Creating iddata Objects” on page 6-9
- “Plotting the Data” on page 6-11
- “Selecting a Subset of the Data” on page 6-15

### Loading Data into the MATLAB® Workspace

Load the sample 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 browser:

- `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.

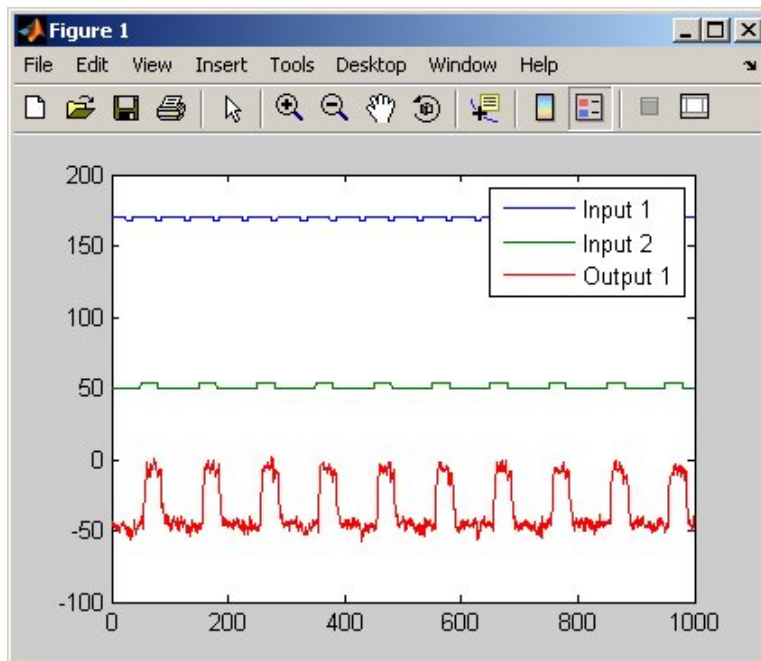
For both experiments, the input data consists of two columns of values. The first column of values is the rate of chemical consumption (in kilograms per minute), and the second column of values is the current (in amperes). The output data is a single column of the rate of carbon-dioxide production (in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

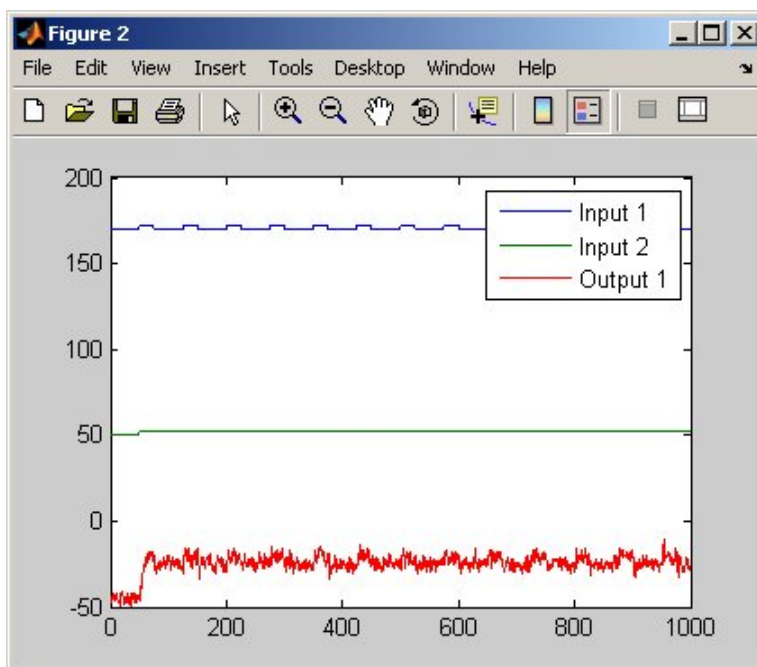
The following plot shows the first experiment, where the operator applies a pulse wave to each input to perturb it from its steady-state equilibrium.



**Input and Output Data from Experiment 1**



The following 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 6-6, 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 starting 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 7-4.

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)
```

MATLAB returns the following data properties and values:

```
Domain: 'Time'  
Name: []  
OutputData: [2001x1 double]  
y: 'Same as OutputData'  
OutputName: {'y1'}  
OutputUnit: {''}  
InputData: [2001x2 double]  
u: 'Same as InputData'  
InputName: {2x1 cell}  
InputUnit: {2x1 cell}  
Period: [2x1 double]  
InterSample: {2x1 cell}  
Ts: 0.5000  
Tstart: []  
SamplingInstants: [2001x0 double]  
TimeUnit: ''  
ExperimentName: 'Exp1'  
Notes: []  
UserData: []
```

To learn more about these properties, 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`.

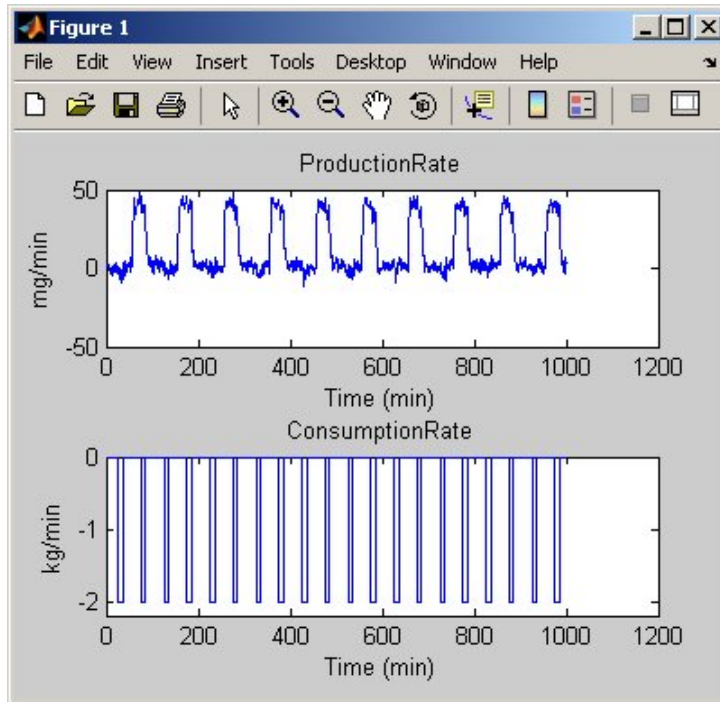
---

## Plotting the Data

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

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

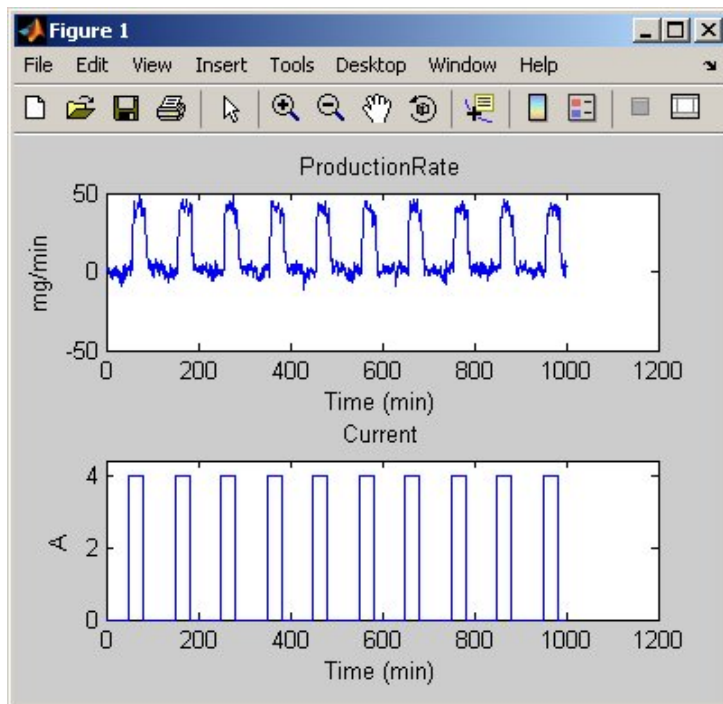
This opens the following plot. The bottom axes show the first input ConsumptionRate, and the top axes show the output ProductionRate.



### 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.

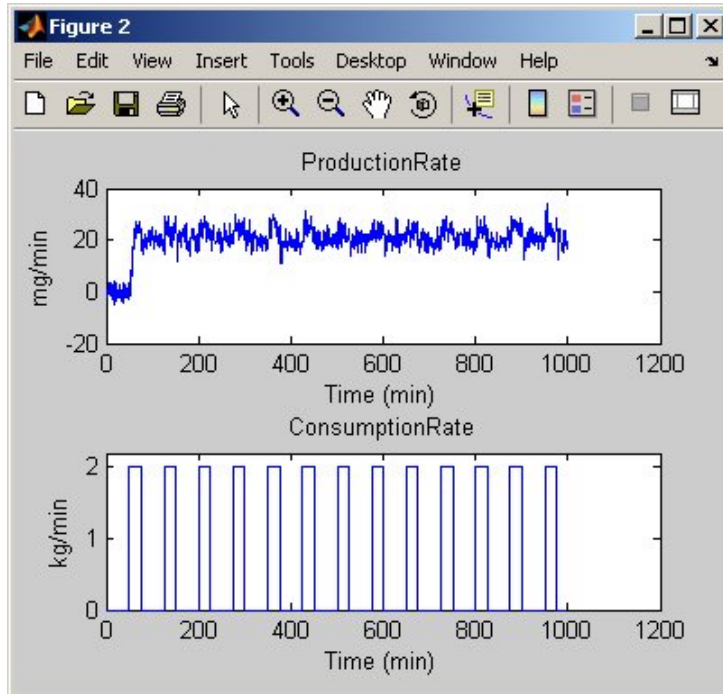
**Tip** For plots of data with multiple inputs and outputs, press **Enter** to view the next input/output pair.



**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:

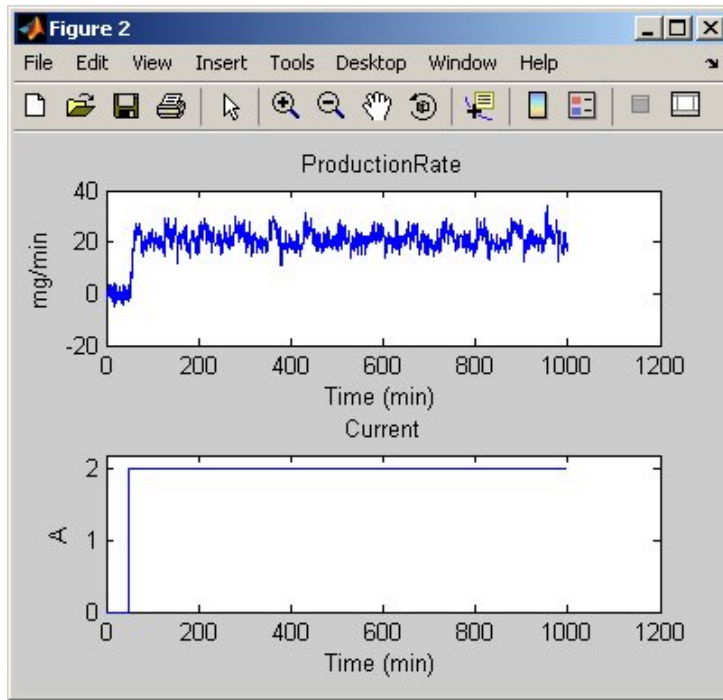
```
figure % Open a new MATLAB Figure window  
plot(zv) % Plot the validation data
```



**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?” on page 6-17

“Estimating the Frequency Response” on page 6-17

“Estimating the Step Response” on page 6-20

### 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 6-9.

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.

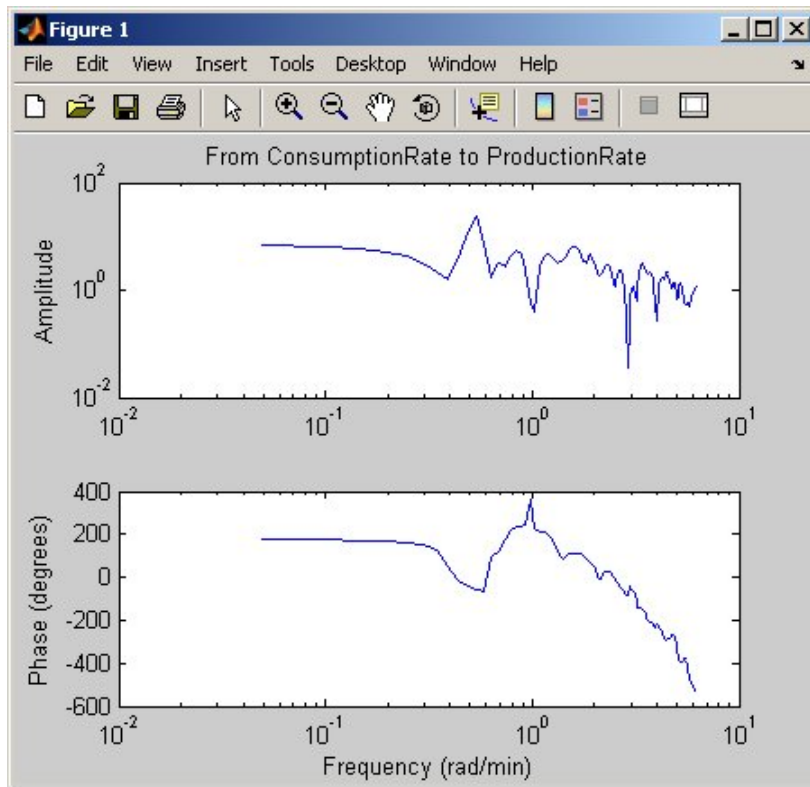
Use the `spa` command to estimate the frequency response:

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

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

```
bode(Ge)
```

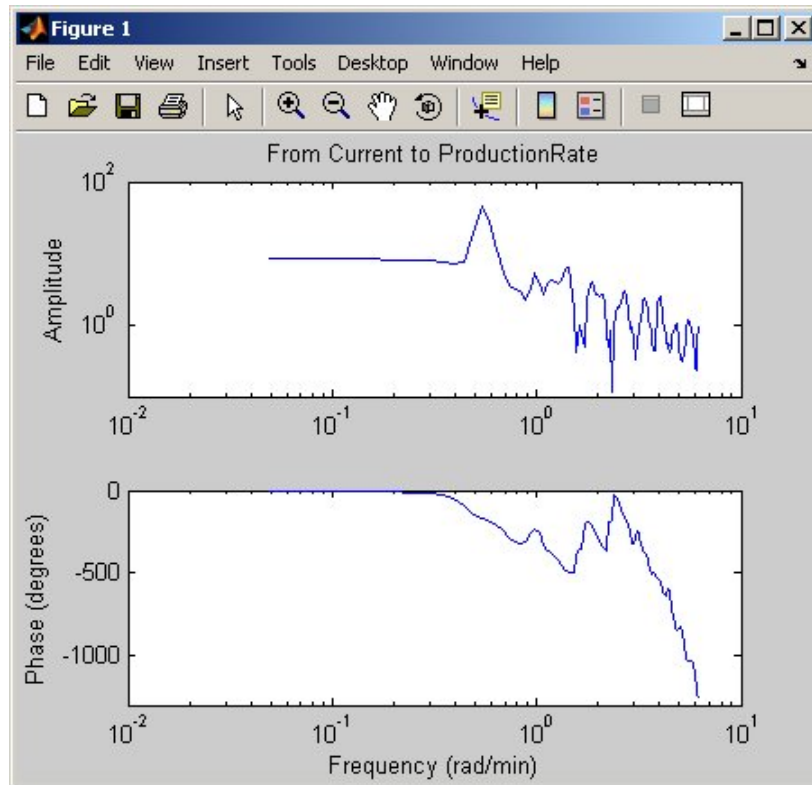
This command produces the following plot.



### 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.

---

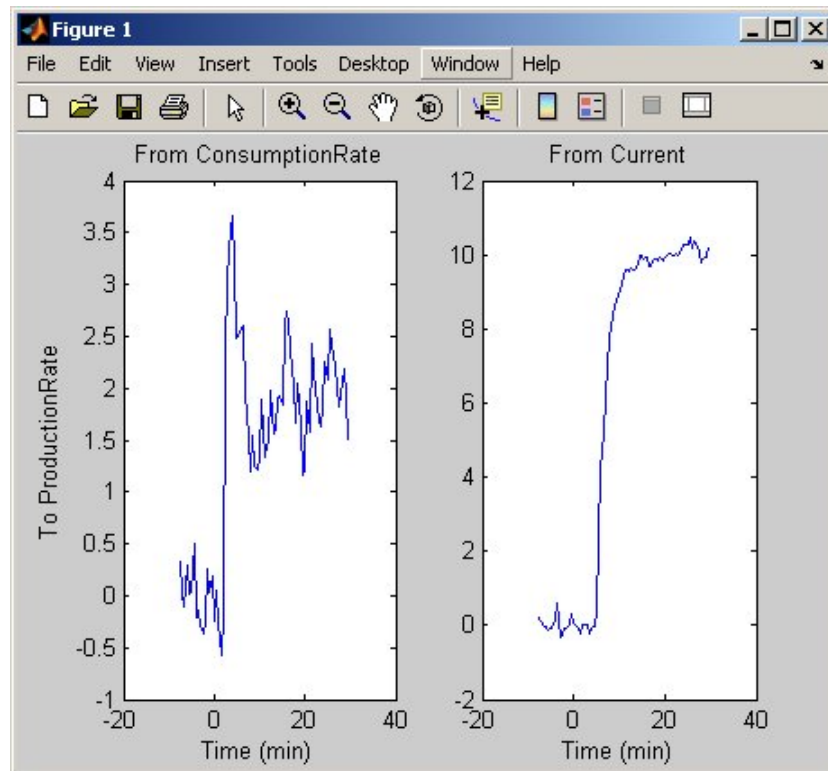
## Estimating the Step Response

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 6-20.

## Estimating Delays in the Multiple-Input System

In this section...
“Why Estimate Delays?” on page 6-22
“Estimating Delays Using the ARX Model Structure” on page 6-22
“Estimating Delays Using Alternative Methods” on page 6-23

### 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:

$$y(t) + a_1y(t-1) + \dots + a_{n_a}y(t-n_a) = b_1u(t-n_k) + \dots + b_{n_b}u(t-n_k-n_b+1) + e(t)$$

$y(t)$  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 delay (the number of samples before the input affects output of the system), 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?” on page 6-25

“Commands for Estimating the Model Order” on page 6-25

“Model Order for the First Input-Output Combination” on page 6-27

“Model Order for the Second Input-Output Combination” on page 6-30

### 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.

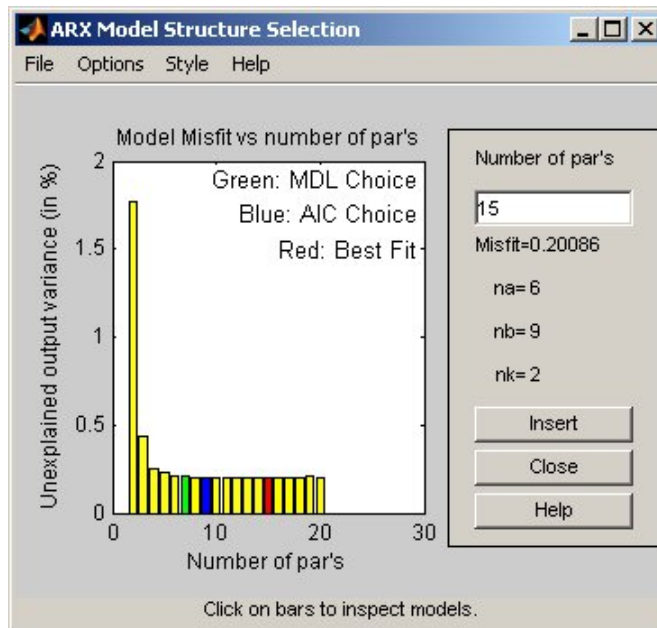
### Commands for Estimating the Model Order

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 a specific number of parameters. The *prediction error* is the sum of the squares of the differences between the validation data output and the model 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.

---

## 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 6-17 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 6-22, 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.

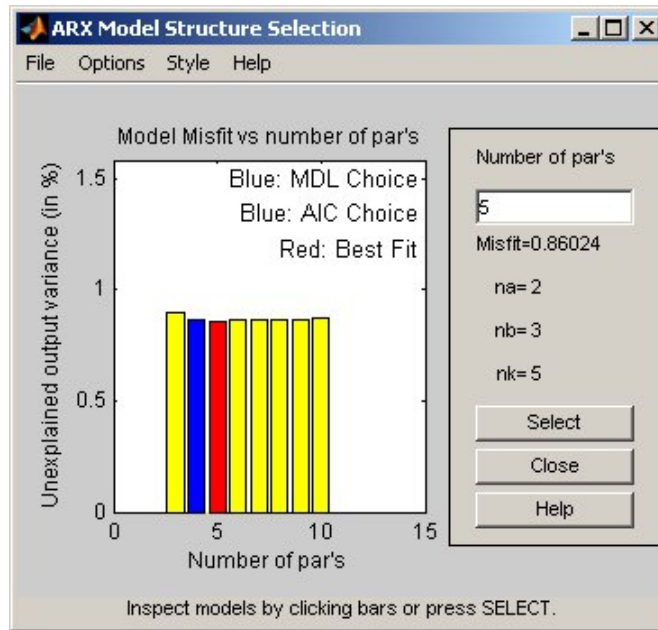
```
selstruc(arxstruc(ze(:, :, 1), zv(:, :, 1), NN1))
```

---

**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 6-17 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 6-22, 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.

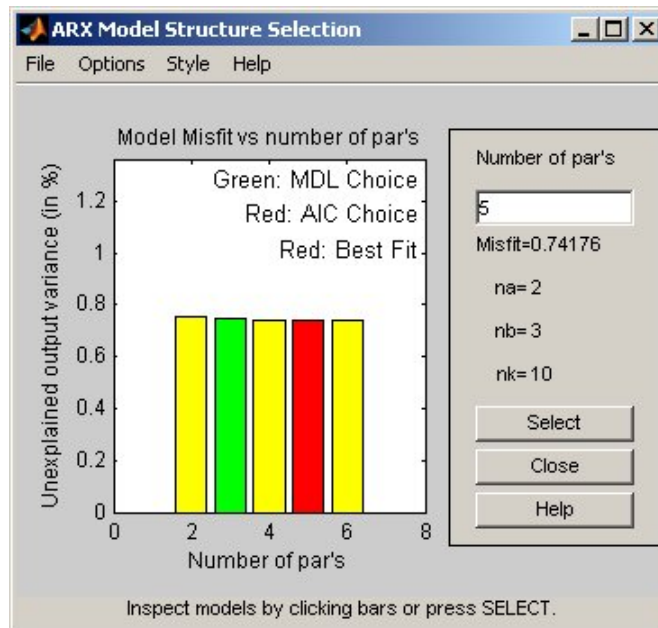
```
NN2 = struc(1:3,1:3,10);
```



- 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” on page 6-33

“Viewing the Model Structure and Parameter Values” on page 6-34

“Specifying Initial Guesses for Time Delays” on page 6-35

“Estimating Model Parameters Using pem” on page 6-36

“Validating the Process Model” on page 6-37

“Refining the Process Model by Including a Noise Model” on page 6-40

### 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 6-5.

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 computes the following output:

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

where

$$G_1(s) = K * \frac{1+Tz*s}{1+2*Zeta*Tw*s+(Tw*s)^2} * \exp(-Td*s)$$

with    K = NaN  
           Tw = NaN  
         Zeta = NaN  
           Td = NaN  
           Tz = NaN

$$G_2(s) = \frac{K}{1+Tp1*s} * \exp(-Td*s)$$

with    K = 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:

```
midproc0.Td = [2.5 5];
```

---

**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 6-33. Use `midproc0` as the model structure and `Ze1` as the estimation data:

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

MATLAB estimates the following parameters:

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

where

$$G_1(s) = K * \frac{1+Tz*s}{1+2*Zeta*Tw*s+(Tw*s)^2} * \exp(-Td*s)$$

with    K = 0.12845  
           Tw = 0.70079  
           Zeta = 17.876  
           Td = 2.4739  
           Tz = 477.14

$$G_2(s) = \frac{K}{1+Tp1*s} * \exp(-Td*s)$$

with    K = 10.418  
           Tp1 = 2.1116  
           Td = 4.8864

Estimated using PEM from data set Ze1  
 Loss function 6.2021 and FPE 6.30214

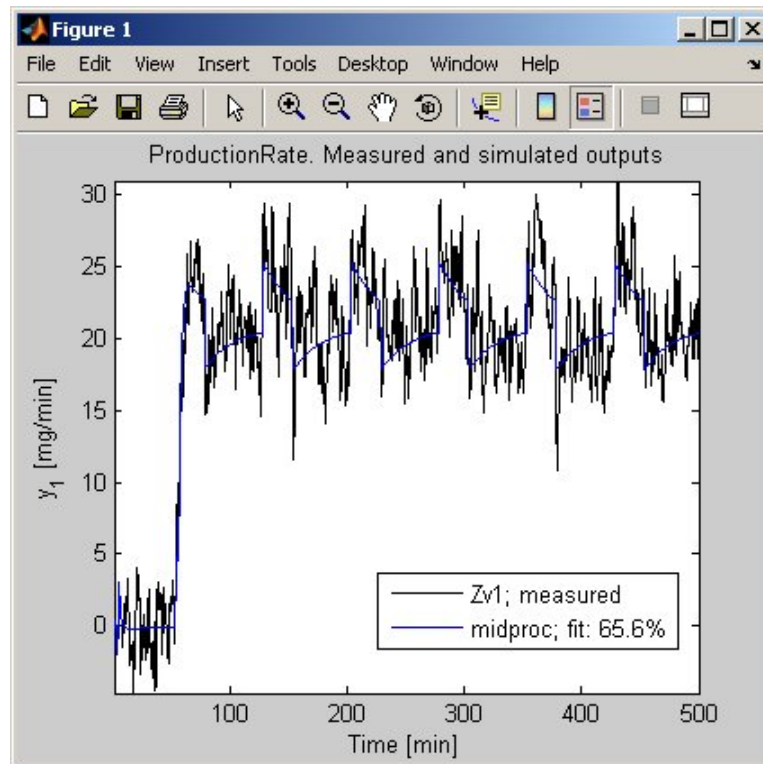
Unlike discrete-time polynomial models, continuous-time models let you estimate the delays. In this case, the estimated delay values 2.4739 and 4.8864 are different from the initial guesses you specified of 2.5 and 5, respectively.

To learn more about estimating models, see the corresponding section 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)
```



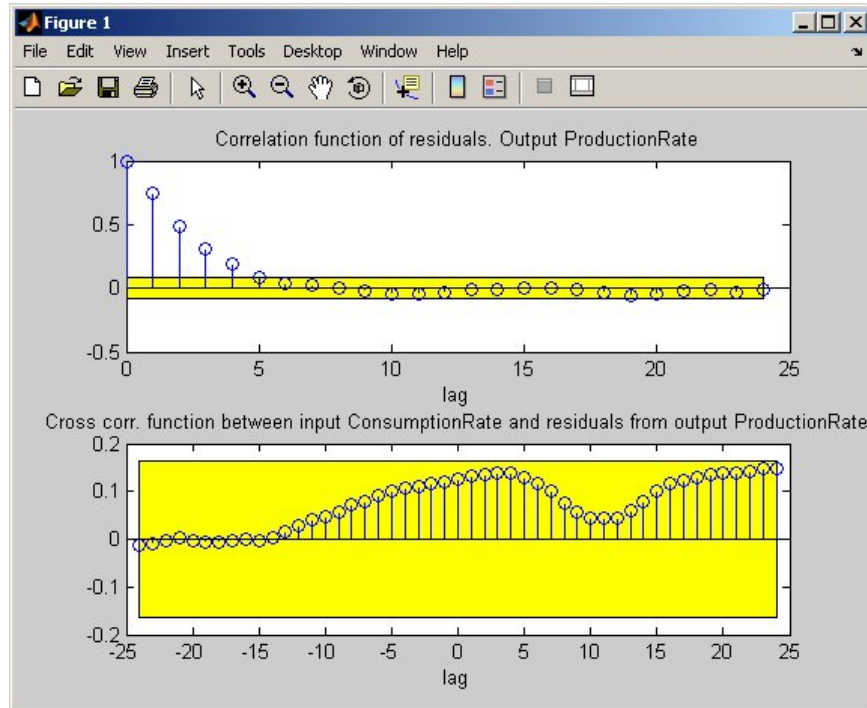
The preceding plot shows that the model output reasonably agrees with the measured output: there is an agreement of 65.6% between the model and the validation data.

Use `resid` to perform residual analysis:

```
resid(Zv1,midproc0)
```

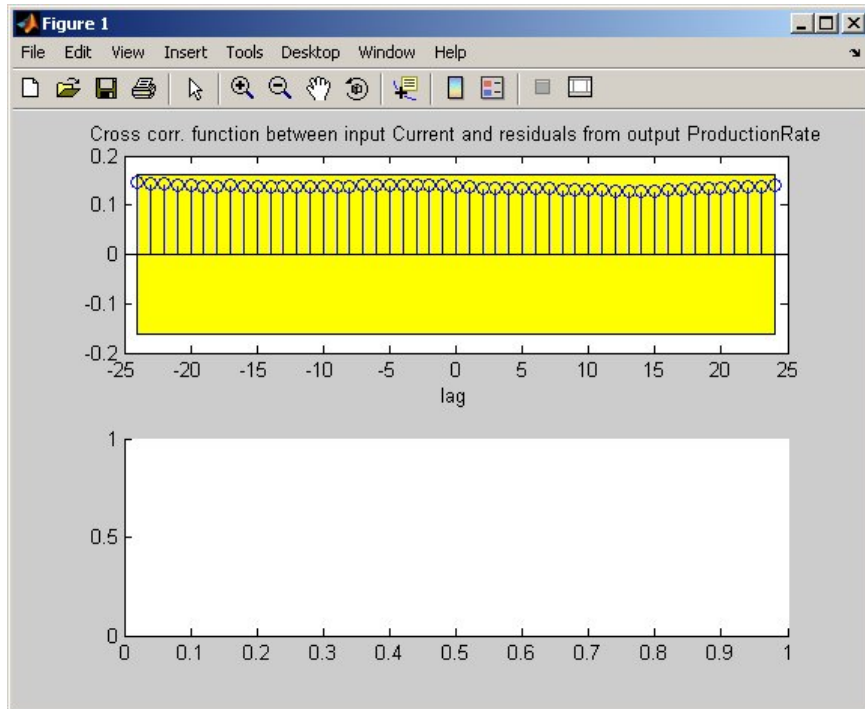


Because the sample system has two inputs, there are two cross-correlation plots of the residuals with each input, as shown in the following figure.



**Autocorrelation and Cross-Correlations of Residuals with the First Input**

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. This lack of correlation indicates that this model is accurate, but that there might be a need for a noise model.

## Refining the Process Model by Including a Noise Model

In “Validating the Process Model” on page 6-37, you noticed that the model performed well except that it produced correlated residuals. This correlation

of residuals indicates evidence of unmodeled dynamics, which might be entering the system as an external disturbance.

This portion of the tutorial shows how you can improve the process model by including a noise model in the estimation.

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

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

---

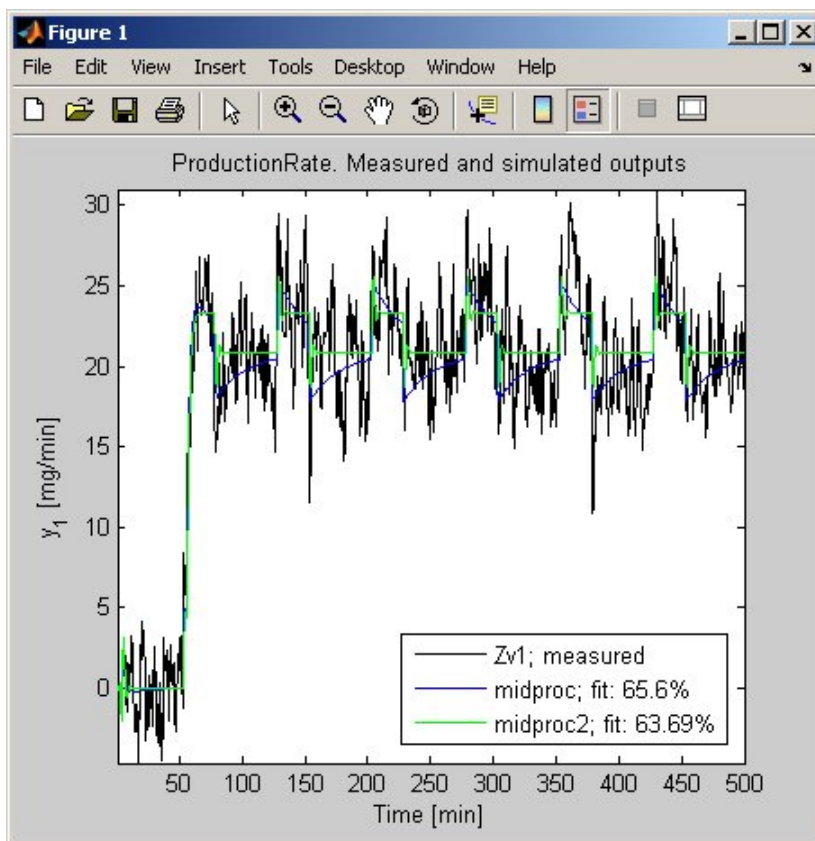
**Note** You can type 'dist' instead of 'DisturbanceModel'. Property names are not case sensitive, and you only need to include the portion of the name that uniquely identifies the property.

---

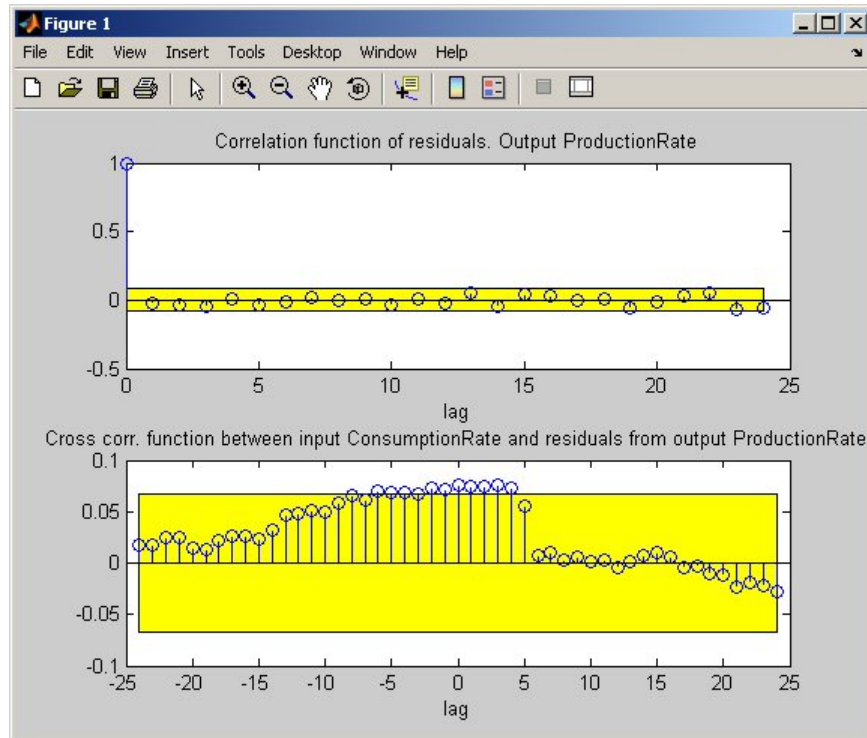
Compare the new model to the old model and to the measured data, and perform residual analysis, as follows:

```
compare(Zv1,midproc,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” on page 6-44
“Estimating a Linear ARX Model” on page 6-45
“Estimating a State-Space Model” on page 6-48
“Estimating a Box-Jenkins Model” on page 6-51
“Comparing Models” on page 6-53

### 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 6-5.

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” on page 6-45
- “Estimating ARX Models Using arx” on page 6-45
- “Accessing Model Data” on page 6-47
- “Learn More” on page 6-48

### About ARX Models

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

$$y(t) + a_1y(t-1) + \dots + a_{n_a}y(t-n_a) = b_1u(t-n_k) + \dots + b_{n_b}u(t-n_k-n_b+1) + e(t)$$

$y(t)$  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 the system output, 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  $a_1 \dots a_n$  and  $b_1 \dots b_n$  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
```

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 -0.9861 0.1512 0.0095]
        b: [2x11 double]
        c: 1
        d: 1
        f: [2x1 double]
    da: [0 0.0301 0.0424 0.0261]
    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: 2.7732
        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 =

    0.7751
    0.2585
   -0.0475
```

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” on page 6-49
- “Estimating State-Space Models Using `n4sid`” on page 6-49
- “Learn More” on page 6-51

## 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 values at time  $t$ , and  $e(t)$  is the white-noise disturbance.

You must specify a single integer as the model order 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.

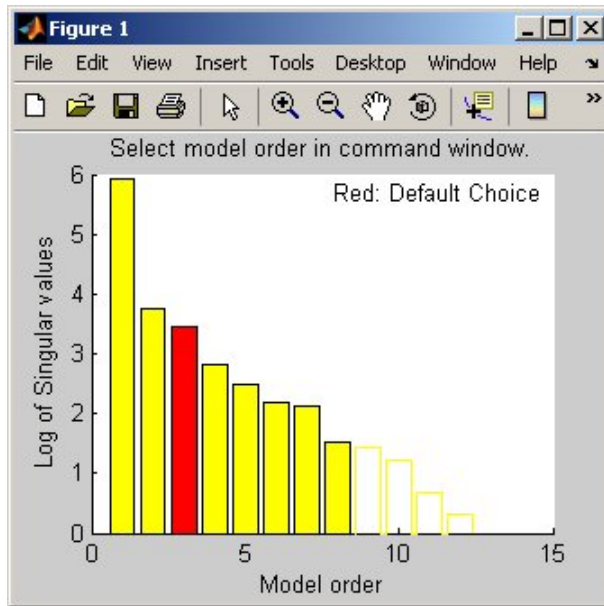
The state-space model structure is a good choice for quick estimation because it contains only two parameters:  $n$  is the number of poles (the size of the  $A$  matrix) and  $nk$  is the delay.

## 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 section in the *System Identification Toolbox User's Guide*.

## Estimating a Box-Jenkins Model

- “About Box-Jenkins Models” on page 6-51
- “Estimating a BJ Model Using pem” on page 6-51
- “Learn More” on page 6-53

### About Box-Jenkins Models

The general Box-Jenkins (BJ) structure is:

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

To estimate a BJ model, you need to specify the parameters  $n_b$ ,  $n_p$ ,  $n_c$ ,  $n_d$ , 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) = 1.903 (+-0.1888) q^-5 - 1.469 (+-0.2304) q^-6  
B2(q) = 2.086 (+-0.09506) q^-10  
C(q) = 1 + 0.1149 (+-0.04131) q^-1  
D(q) = 1 - 0.7364 (+-0.02856) q^-1  
F1(q) = 1 - 1.361 (+-0.06205) q^-1 + 0.5982 (+-0.05408) q^-2  
F2(q) = 1 - 0.8031 (+-0.009455) 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**

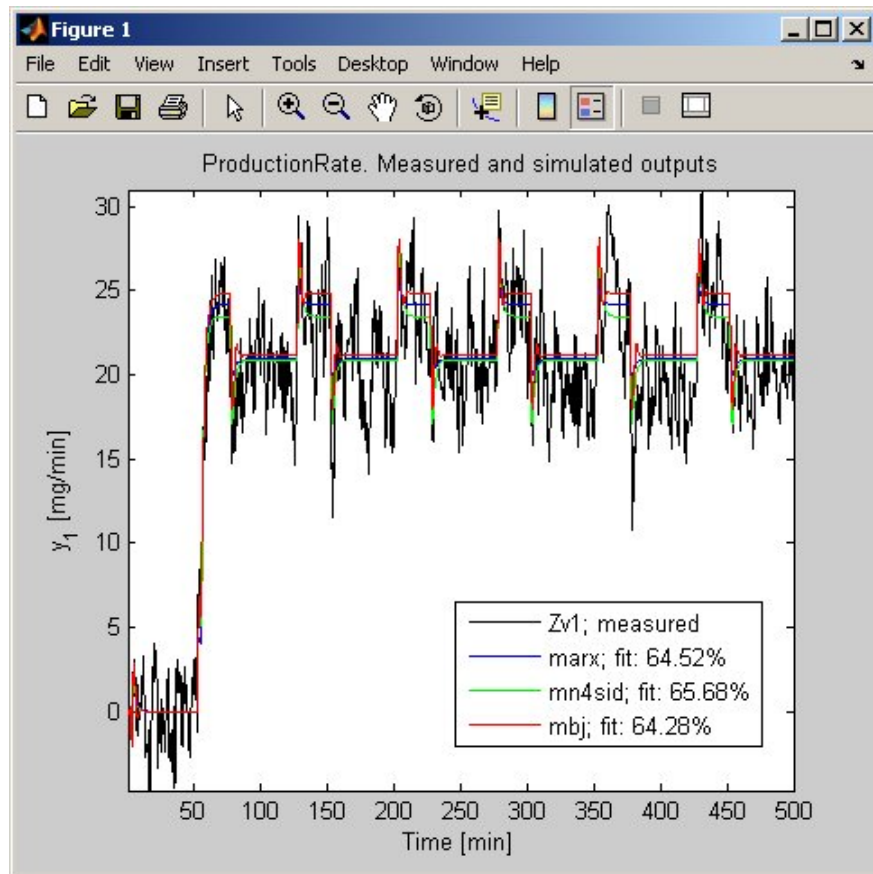
To learn more about estimating BJ models, see the corresponding section in the *System Identification Toolbox User's Guide*.

**Comparing Models**

To compare the output of the ARX, state-space, and Box-Jenkins models with 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” on page 6-56

“Predicting the Future Output” on page 6-58

### 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 6-33.

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')
```

Carefully consider which initial conditions you use in the simulation. A system produces different responses for different initial conditions, even when the input data to the model is the same. If you do not specify the correct initial states, your model response does not match the measured output for your data. Therefore, when you use simulation to validate a model by matching simulated response to measured response, you must estimate the initial conditions from the measured data and use these as the initial states of the simulation.

Use `findstates(idmodel)` to estimate the initial conditions `X0est` from the data set `Zv1`:

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

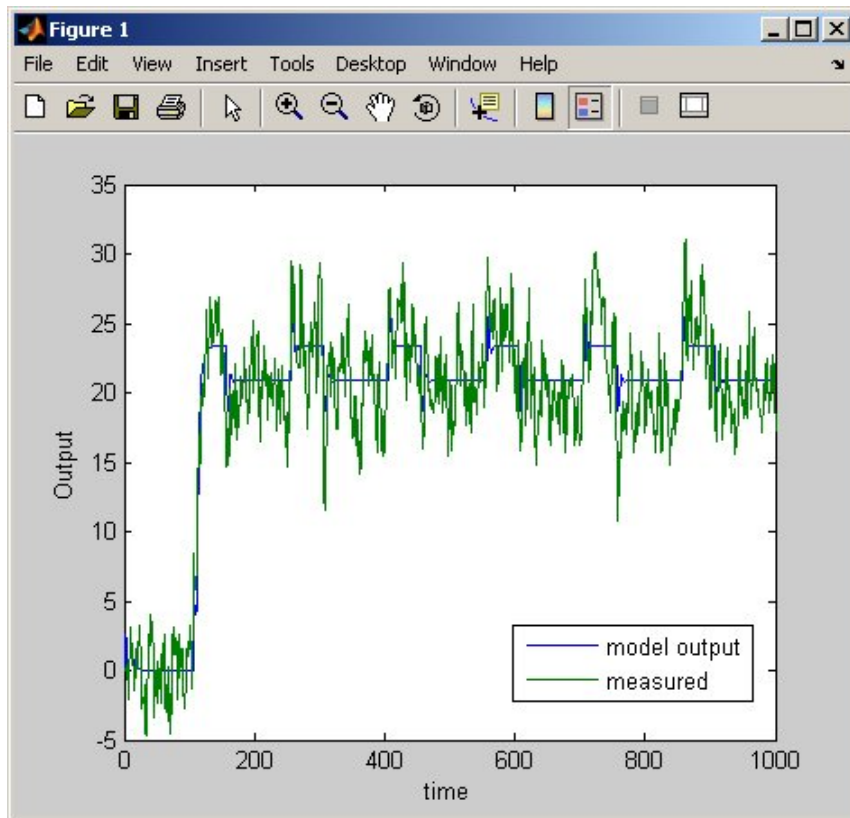
Next, simulate the model using the initial states estimated from the data:

```
ysim_2 = sim(midproc2,U,'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.



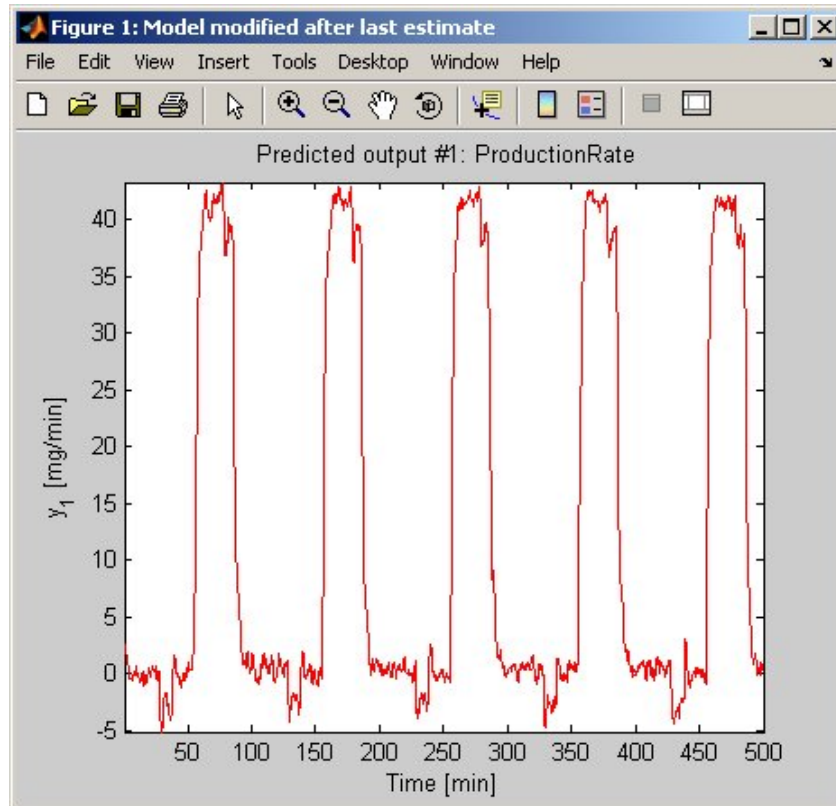
### Predicting the Future Output

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:

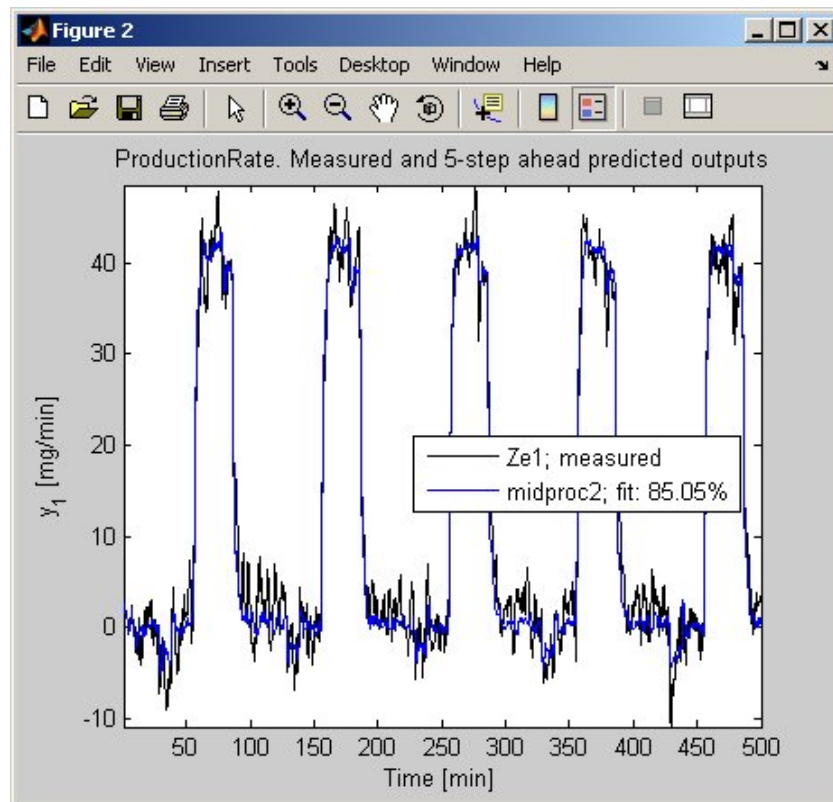
```
compare(Ze1,midproc2,5)
```

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 6-56, `compare` assumes an infinite prediction horizon and simulates the model output instead.

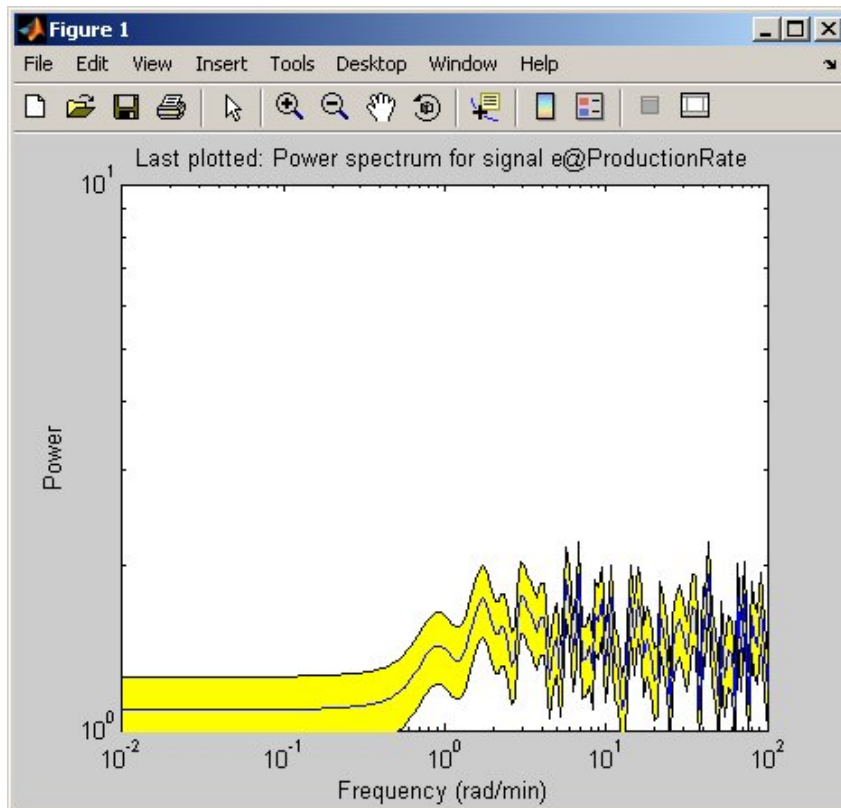
---



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.







# Tutorial – Identifying Nonlinear Black-Box Models Using the GUI

---

About This Tutorial (p. 7-2)

Overview of the tutorial for estimating nonlinear black-box models from single-input/single-output (SISO) data

Preparing Data (p. 7-4)

How to load the sample MAT-file into the MATLAB® workspace, create a data object, open the System Identification Tool GUI, and import data into the System Identification Tool from the MATLAB workspace

Estimating Nonlinear ARX Models (p. 7-9)

How to estimate and validate nonlinear ARX models for single-input/single-output (SISO) data using the System Identification Tool GUI

Estimating Hammerstein-Wiener Models (p. 7-23)

How to estimate and validate Hammerstein-Wiener models for single-input/single-output (SISO) data using the System Identification Tool GUI

# About This Tutorial

In this section...
“Objectives” on page 7-2
“Sample Data” on page 7-2

## Objectives

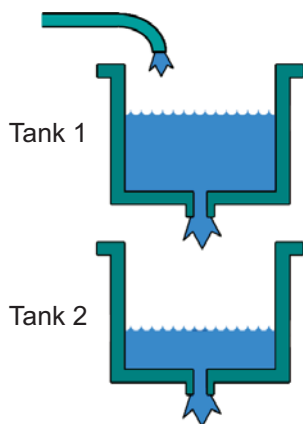
Estimate and validate nonlinear models from single-input/single-output (SISO) data to find the one that best represents your system dynamics.

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.

## Sample Data

The sample data you use in this tutorial is in `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.

## Preparing Data

In this section...
“Loading Data into the MATLAB® Workspace” on page 7-4
“Creating iddata Objects” on page 7-4
“Starting the System Identification Tool” on page 7-6
“Importing Data Objects into the System Identification Tool” on page 7-7

### 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:

- `y` is the output data, which is the water height in Tank 2 (in meters).
- `u` is the input data, which is the voltage applied to the pump that feeds the water into Tank 1 (in volts).

### 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 7-4.

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 0.5 min
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.

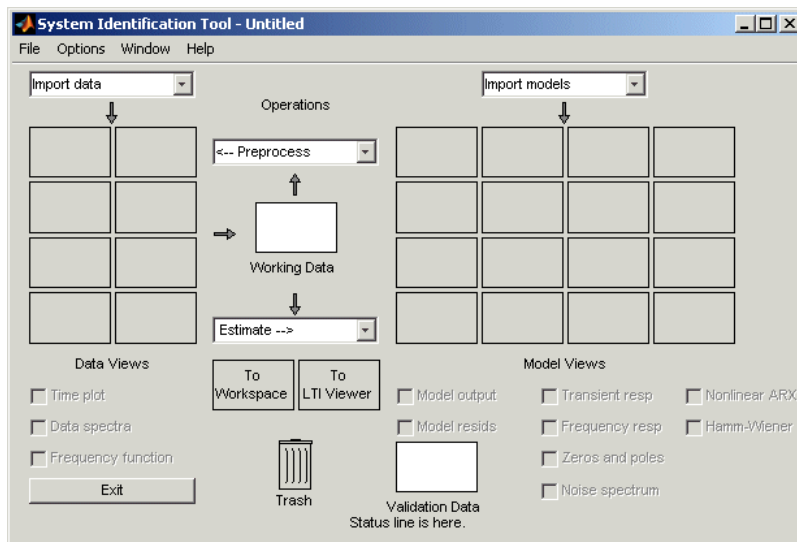
---

### 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, displays 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 7-4, and opened the GUI, as described in “Starting the System Identification Tool” on page 7-6.

- 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 fields.

Click **More** to view the following 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 **ze** icon to the **Working Data** rectangle, and drag the **zv** icon to the **Validation Data** rectangle.



## Estimating Nonlinear ARX Models

### In this section...

“Estimating a Nonlinear ARX Model with Default Settings” on page 7-9

“Plotting Nonlinearity Cross-Sections for Nonlinear ARX Models” on page 7-13

“Changing the Nonlinear ARX Model Structure” on page 7-16

“Selecting a Subset of Regressors in the Nonlinear Block” on page 7-18

“Changing the Nonlinearity Estimator in a Nonlinear ARX Model” on page 7-20

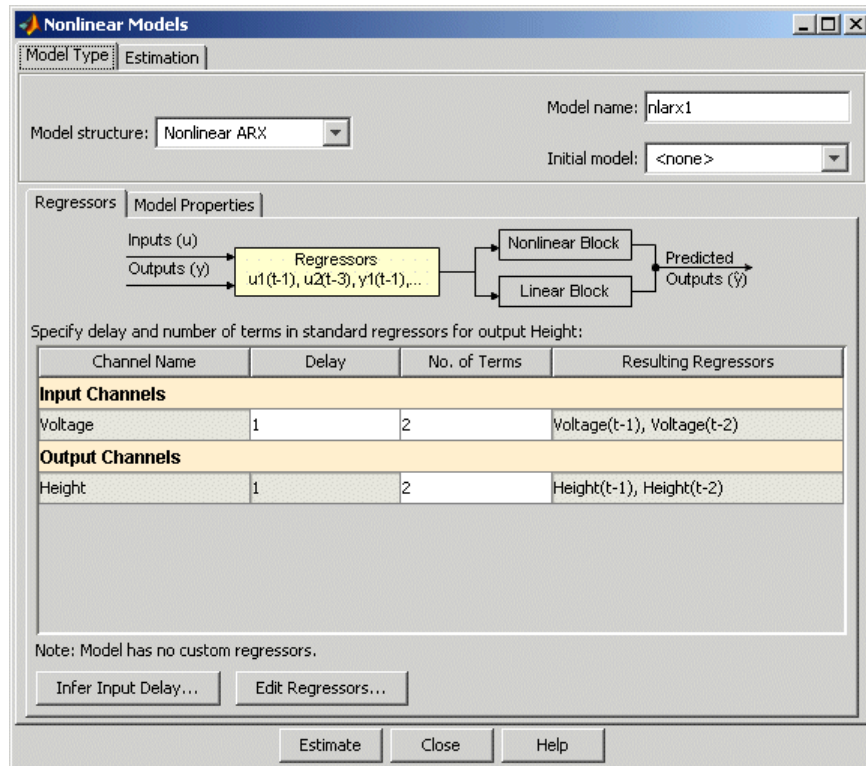
“Selecting the Best Model” on page 7-21

### Estimating a Nonlinear ARX Model with Default Settings

In this portion of the tutorial, you estimate a nonlinear ARX model using default estimation options.

You must have already prepared the data, as described in “Preparing Data” on page 7-4.

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



The **Model Type** tab is already open and the default **Model Structure** is Nonlinear ARX.

In the **Regressors** tab, the model orders for both **Input Channels** and **Output Channels** are specified by the **Delay** of 1 and **No. of Terms** equal to 2. Thus, the model output  $y(t)$  is related to the input  $u(t)$  via the following nonlinear autoregressive equation:

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

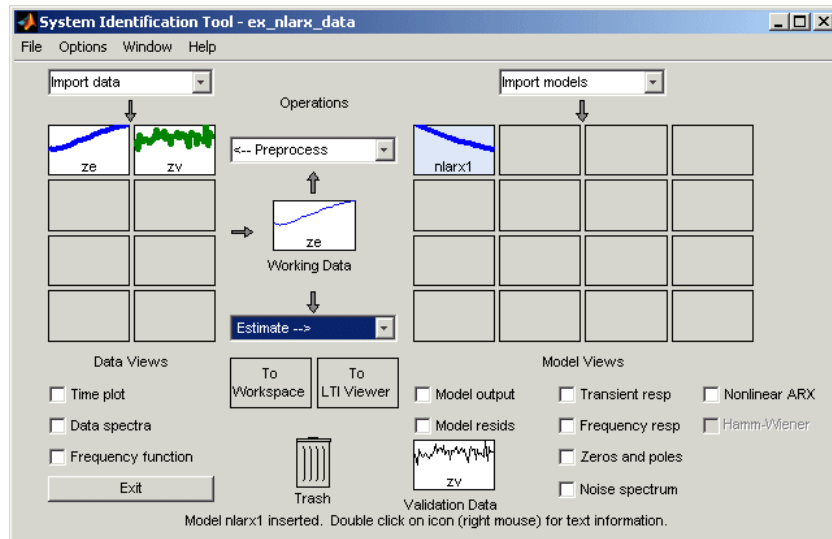
$f$  is the nonlinearity estimator you select in the **Model Properties** tab.

## 2 Click the **Model Properties** tab.

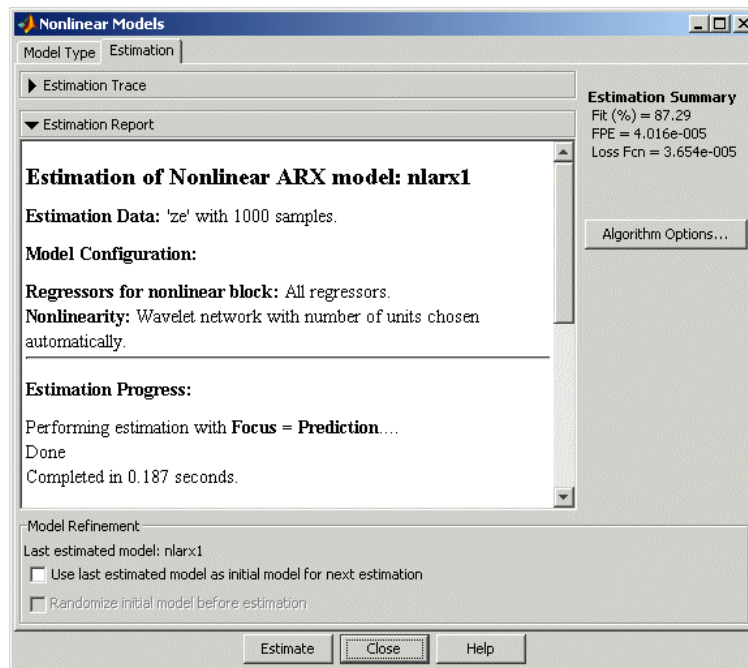
The **Nonlinearity** represents the nonlinear function  $f$  and is already set to Wavelet Network, by default. The number of units for the nonlinearity estimator is set to **Select automatically**, which lets the algorithm search for the best number of nonlinearity units during estimation.

## 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 the following estimation information in the **Estimation** tab.

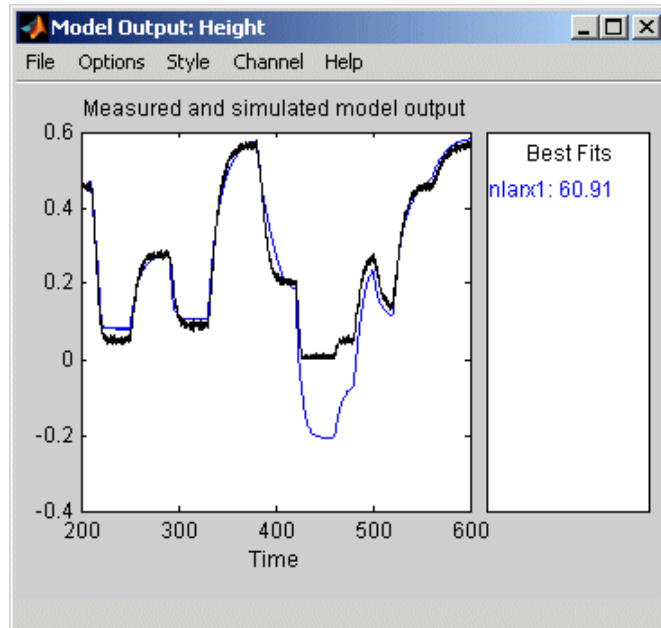


---

**Note** **Fit (%)** is computed using the estimation data set, and not the validation data set. However, the model output plot shows 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.



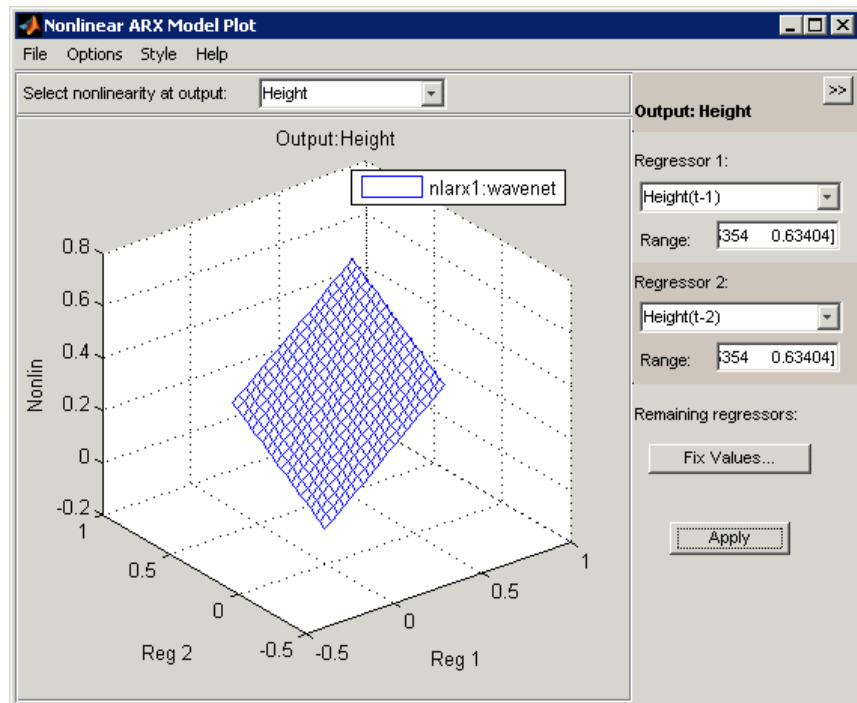
The **Best Fits** area of the Model Output plot shows that the agreement between the model output and the validation-data output is 60.91%.

## 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.

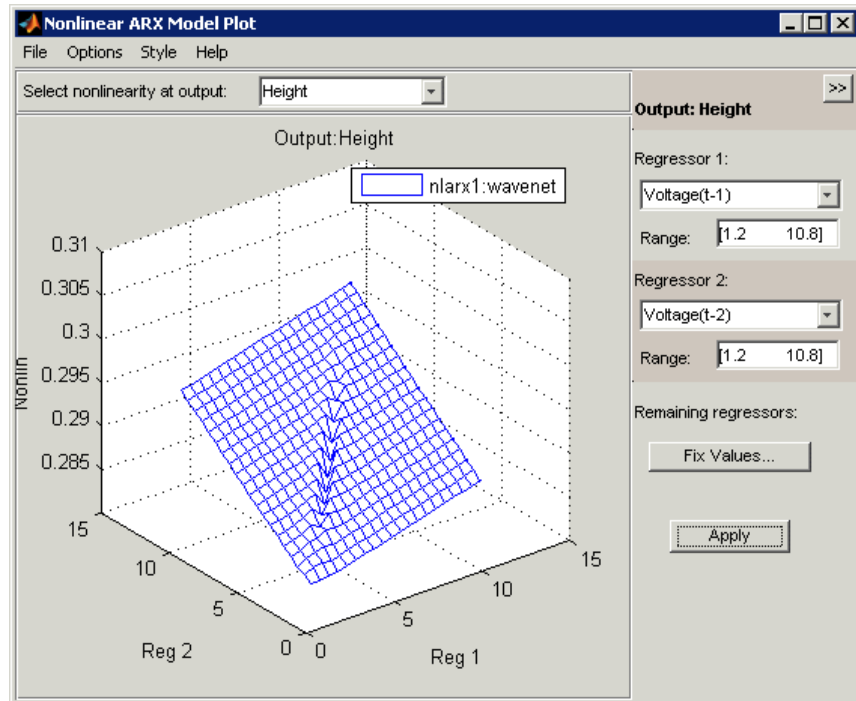
- 1 In the System Identification Tool GUI, select the **Nonlinear ARX** check box to view the nonlinearity cross-sections.

By default, the plot shows the relationship between the output regressors  $\text{Height}(t-1)$  and  $\text{Height}(t-2)$ . This plot shows a regular plane in the following figure. Thus, the relationship between the regressors is approximately a linear plane.



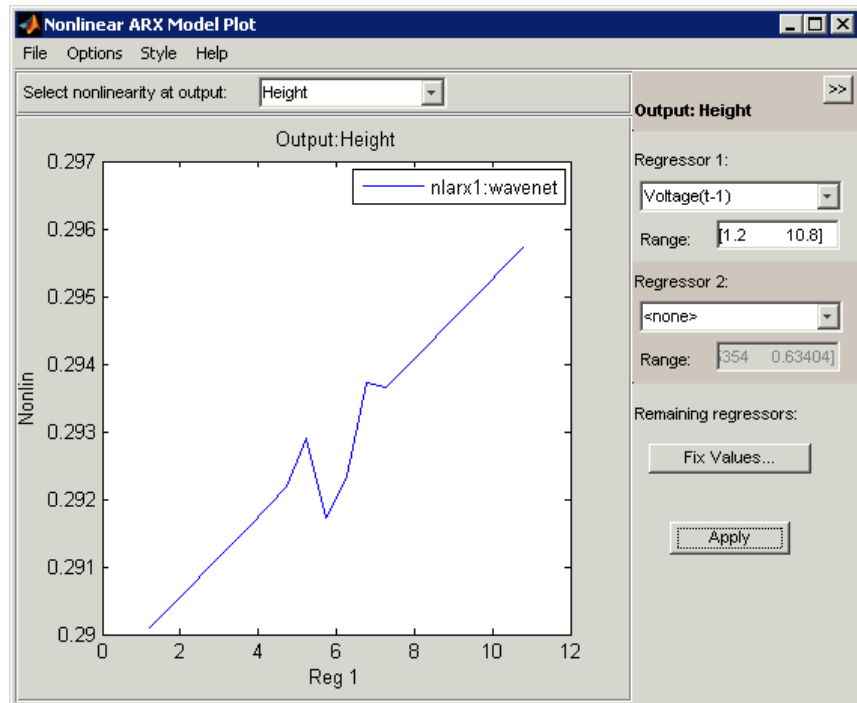
- 2** In the Nonlinear ARX Model Plot window, keep the default value for **Regressor 1** at Voltage(t-1). Set **Regressor 2** to Voltage(t-2). Click **Apply**.

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



- 3** To rotate the nonlinearity surface, select **Style > 3D Rotate** 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 after modifying the default input delay and the nonlinearity settings. Typically, you select model orders and delays 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 7-9.

- 1 In the Nonlinear Models dialog box, click the **Model Type** tab, and click the **Regressors** tab.



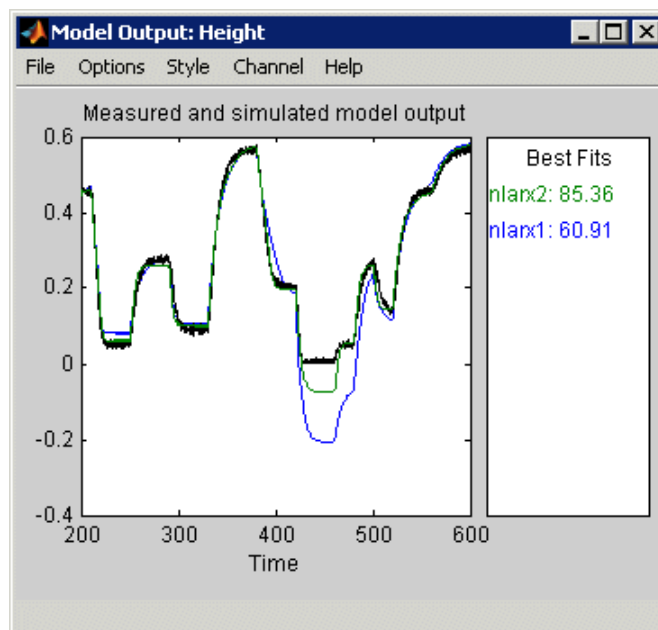
- 2** For the Voltage input channel, double-click the corresponding **Delay** cell, enter 3, and press **Enter**.

This action updates the **Resulting Regressors** list. The list now includes Voltage(t-3) and Voltage(t-4), which are two 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 **Estimation** tab.

The **Best Fits** area of the Model Output window shows that the nlarx2 fit is 85.36%.

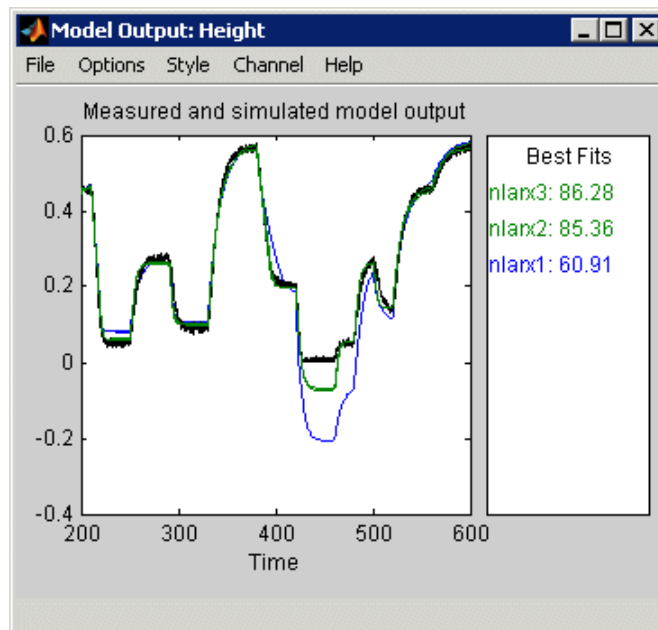


- 4** In the Nonlinear Models dialog box, click the **Model Properties** tab.
- 5** In the **Number of units in nonlinear block**, select **Enter**, and type 6.

### 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.

The **Best Fits** area of the Model Output window shows that the `nlarx3` fit is 86.28%.

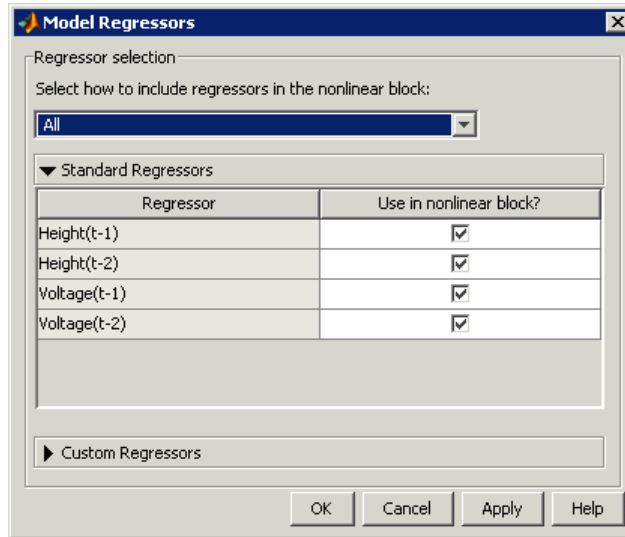


## Selecting a Subset of Regressors in the Nonlinear Block

In this portion of the tutorial, you try to improve the fit by selecting 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 7-16.

- 1 In the Nonlinear Models dialog box, click the **Model Type** tab, and click the **Regressors** tab.
- 2 Click **Edit Regressors** to open the Model Regressors dialog box.



- 3 Clear the following check boxes:

- **Height(t-2)**
- **Voltage(t-1)**

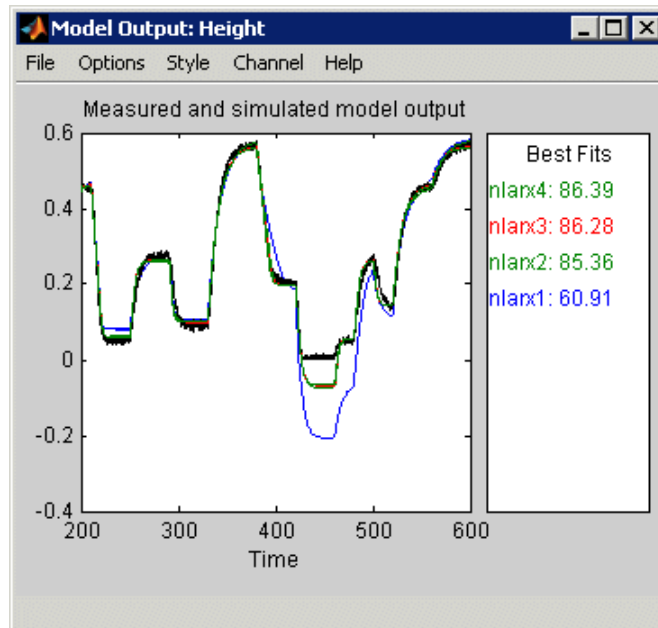
Click **OK**.

This action excludes the time-shifted Height (t-2) and Voltage (t-1) from the list of inputs to the nonlinear block.

### 4 Click **Estimate**.

This action adds the model `n1arx4` to the System Identification Tool GUI. It also updates the Model Output window, as shown in the following figure.

The **Best Fits** area of the Model Output window shows that the `n1arx4` fit is 86.39%, which is only a fraction of a percent improvement from the previous fit.



## Changing the Nonlinearity Estimator in a Nonlinear ARX Model

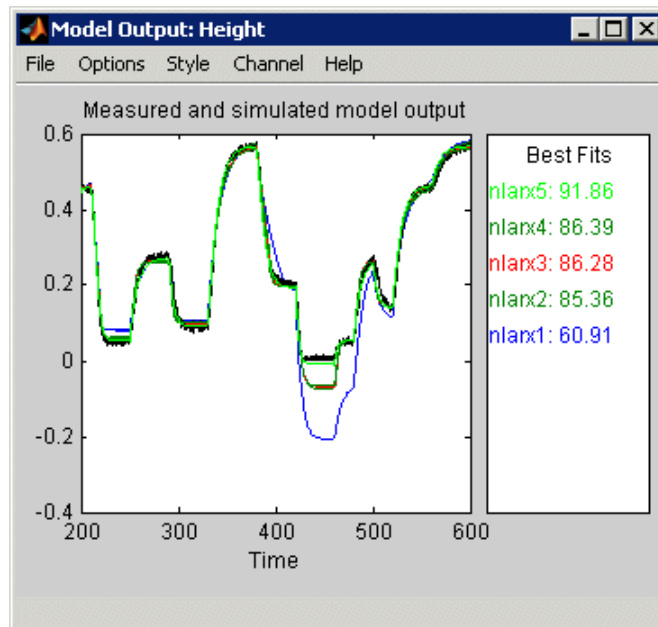
In this portion of the example, you improve the fit of the model estimated with default settings, `n1arx1`, by changing the nonlinearity.

- 1 In the Nonlinear Models dialog box, click the **Model Type** tab.
- 2 In the **Initial model** list, select `n1arx1`.
- 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**.

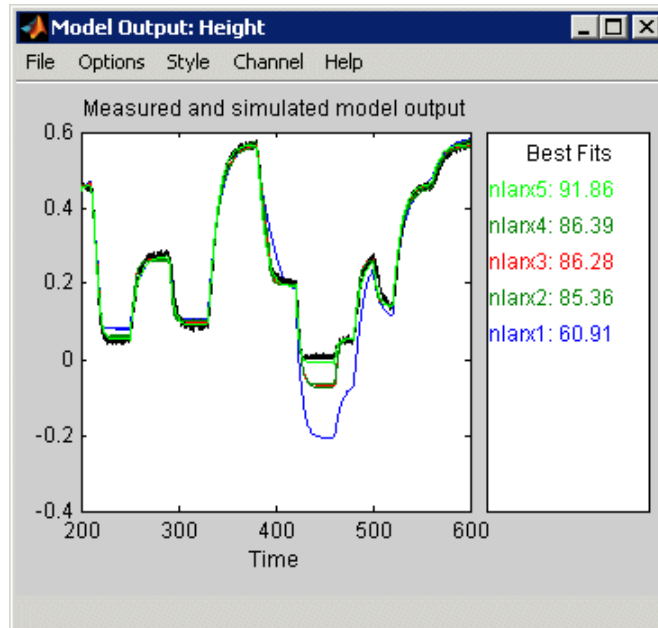
This action adds the model `nlarx5` to the System Identification Tool GUI. It also updates the Model Output plot, as shown in the following figure.

The **Best Fits** area of the Model Output window shows that the `nlarx5` fit is 91.86%.



## 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 “Changing the Nonlinearity Estimator in a Nonlinear ARX Model” on page 7-20, as shown in the following figure.



## Estimating Hammerstein-Wiener Models

### In this section...

“Estimating Hammerstein-Wiener Models with Default Settings” on page 7-23

“Plotting the Nonlinearities and Linear Transfer Function” on page 7-27

“Changing the Hammerstein-Wiener Model Structure” on page 7-31

“Changing the Nonlinearity Estimator in a Hammerstein-Wiener Model” on page 7-33

“Selecting the Best Model” on page 7-35

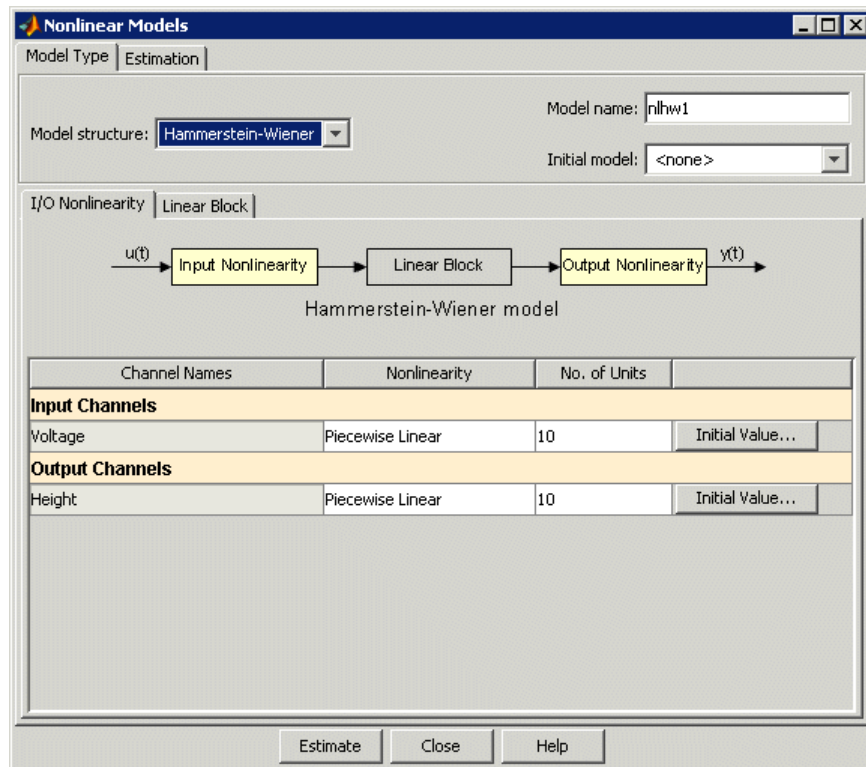
### Estimating Hammerstein-Wiener Models with Default Settings

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

You must have already prepared the data, as described in “Preparing Data” on page 7-4.

- 1** In the System Identification Tool GUI, select **Estimate > Nonlinear models** to open the Nonlinear Models dialog box.
- 2** In the **Model Type** tab, select Hammerstein-Wiener in the **Model Structure** list.

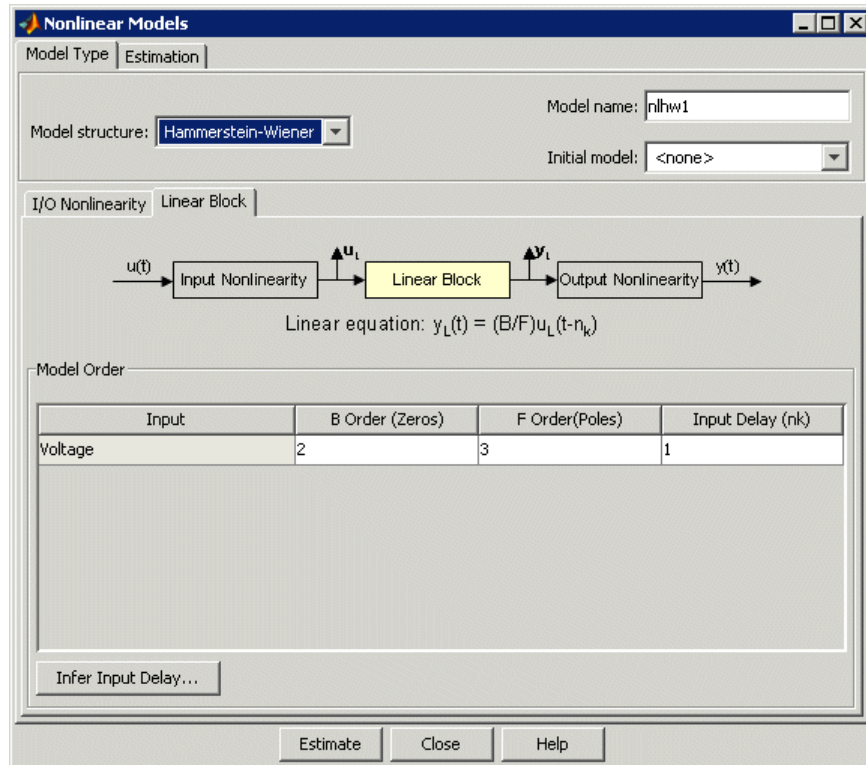
3 Keep the defaults in the **I/O Nonlinearity** tab.



By default, the nonlinearity estimator is Piecewise Linear with 10 units for **Input Channels** and **Output Channels**.



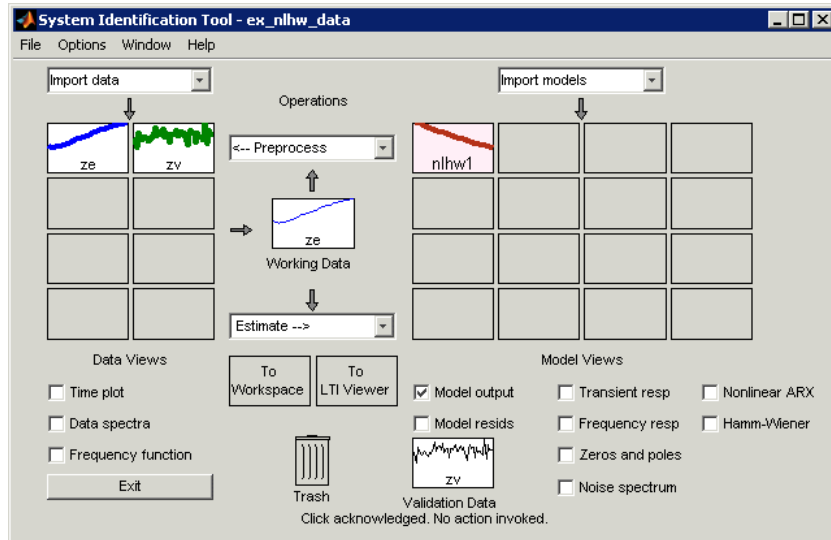
**4** Keep the defaults in the **Linear Block** tab.



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**.

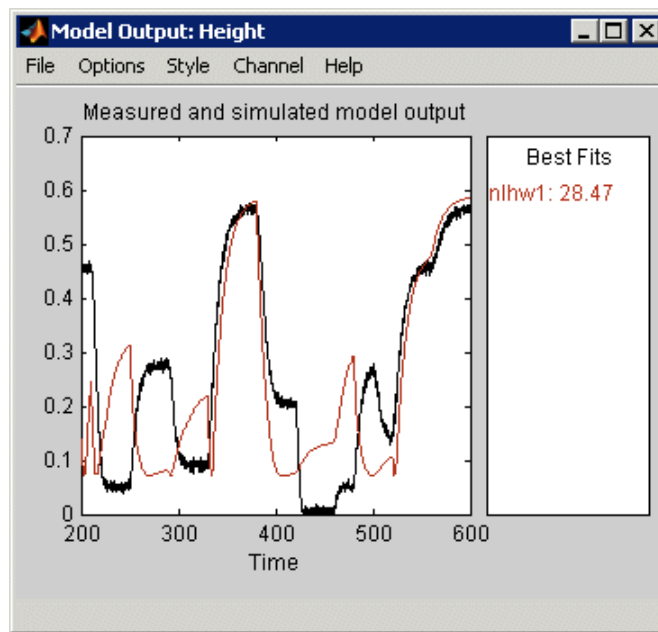
This action adds the model n1hw1 to the System Identification Tool GUI, as shown in the following figure.



- 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 that the agreement between the model output and the validation-data output is 28.47%. Thus, the default settings do not produce an accurate fit.

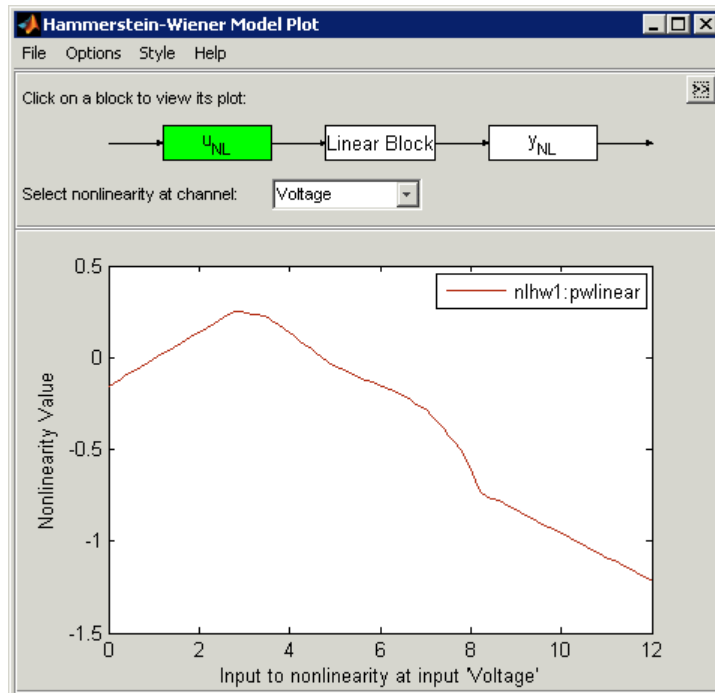


## Plotting the Nonlinearities and Linear Transfer Function

You can plot the input/output nonlinearities and the linear transfer function of the model on a Hammerstein-Wiener plot.

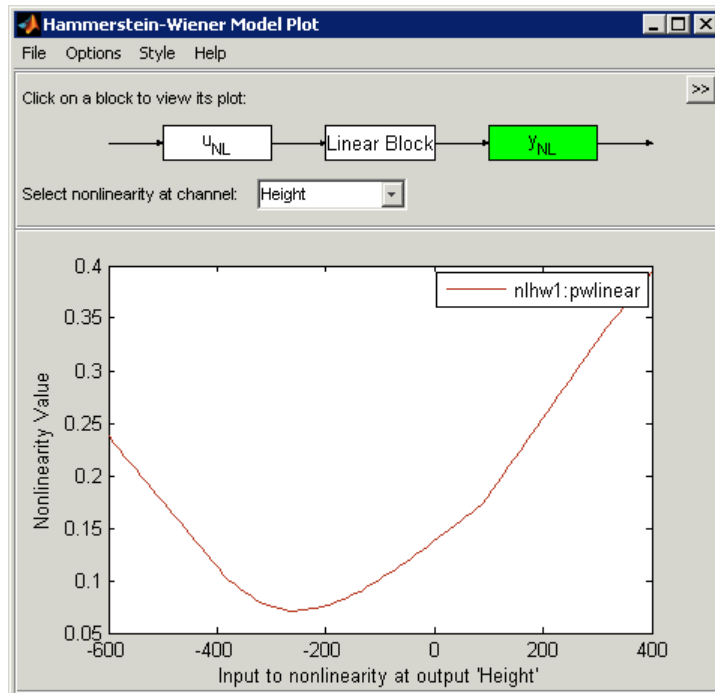
- 1 In the System Identification Tool GUI, select the **Hamm-Wiener** check box to view the Hammerstein-Wiener model plot.

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



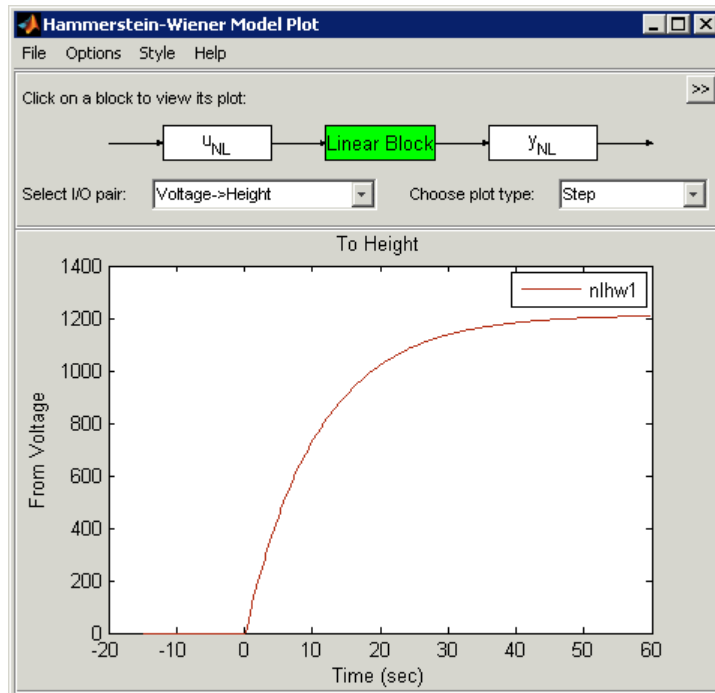
- Click the  $y_{NL}$  rectangle in the top portion of the Hammerstein-Wiener Model Plot window.

The plot updates to display the output nonlinearity.

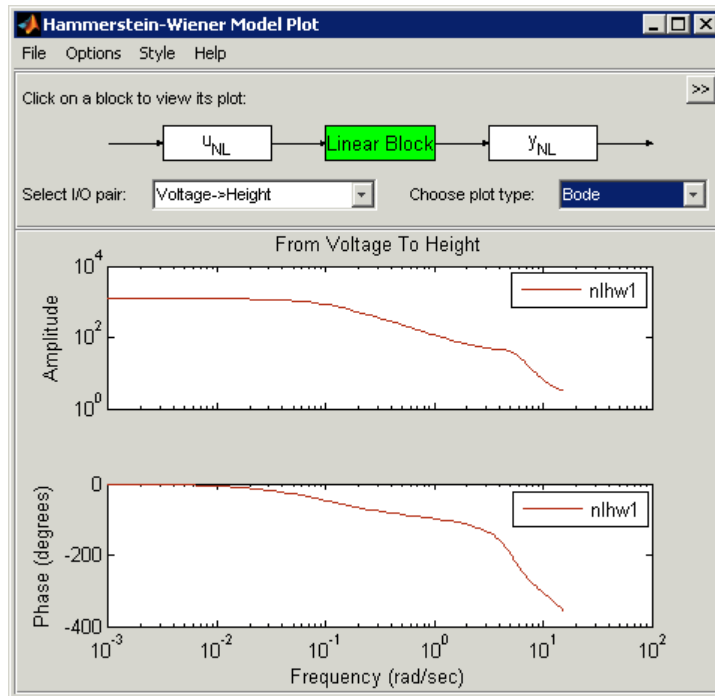


- 3 Click the **Linear Block** rectangle in the top portion of the Hammerstein-Wiener Model Plot window.

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, as shown in the following figure.



## Changing the Hammerstein-Wiener Model Structure

In this portion of the tutorial, you estimate a Hammerstein-Wiener model after modifying the default model order and the nonlinearity settings. Typically, you select model orders and delays by trial and error until you get a model that produces an accurate 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 7-23.

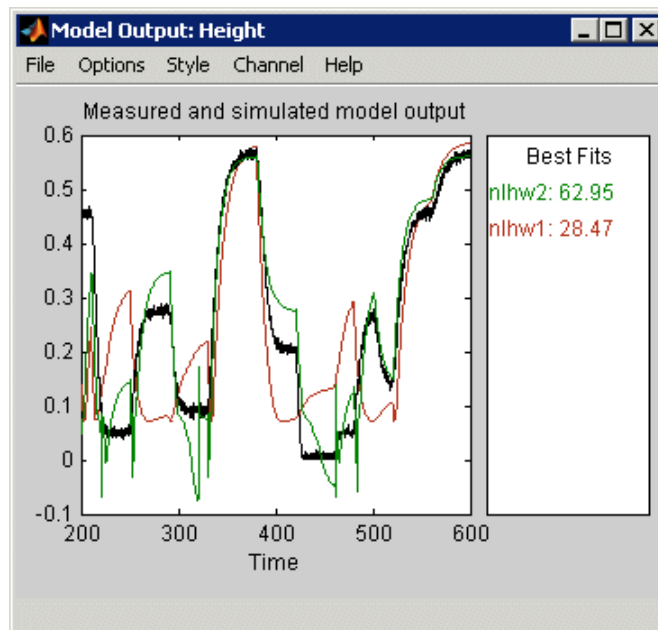
- 1** In the Nonlinear Models dialog box, click the **Model Type** tab, and click the **Linear Block** tab.

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

**3** Click **Estimate**.

This action adds the model n1hw2 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 that the n1hw2 fit is 62.95%.



**4** In the Nonlinear Models dialog box, select the **I/O Nonlinearity** tab.

**5** For the Voltage input channel, double-click the corresponding **No. of Units** cell, and type 20 as the number of units. Press **Enter**.

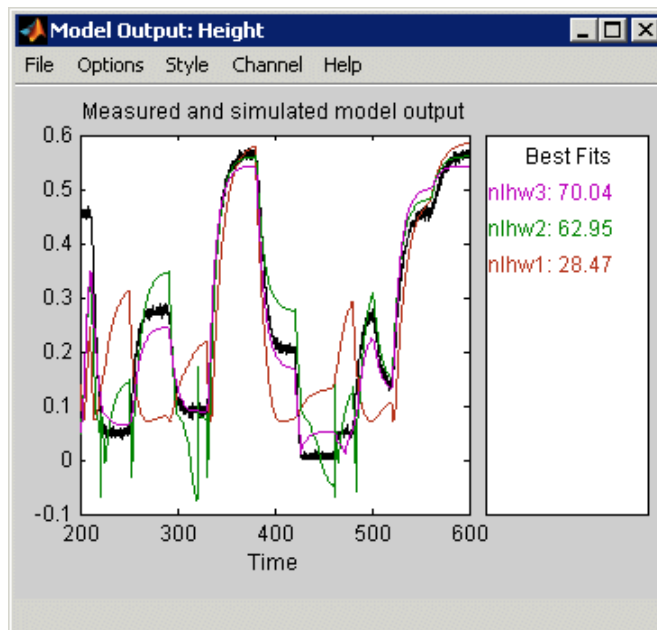
This action changes the number of nonlinearity units for the Piecewise Linear nonlinearity estimator corresponding to the input channel.



## 6 Click **Estimate**.

This action adds the `n1hw3` model to the System Identification Tool GUI. It also updates the Model Output window, as shown in the following figure.

The **Best Fits** area of the Model Output window shows that the `n1hw3` fit is 70.04%.



## Changing the Nonlinearity Estimator in a Hammerstein-Wiener Model

In this portion of the example, you improve the fit by changing the nonlinearity estimator.

- 1 In the Nonlinear Models dialog box, click the **Model Type** 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** In the Nonlinear Models dialog box, click the **Model Type** tab, and click the **I/O Nonlinearity** tab.
- 4** For the Voltage input, click the **Nonlinearity** cell, and select Sigmoid Network from the list, as shown in the following figure.

Channel Names	Nonlinearity	No. of Units	
<b>Input Channels</b>			
Voltage	Piecewise Linear	20	Initial Value...
<b>Output Channels</b>			
Height	Sigmoid Network	10	Initial Value...

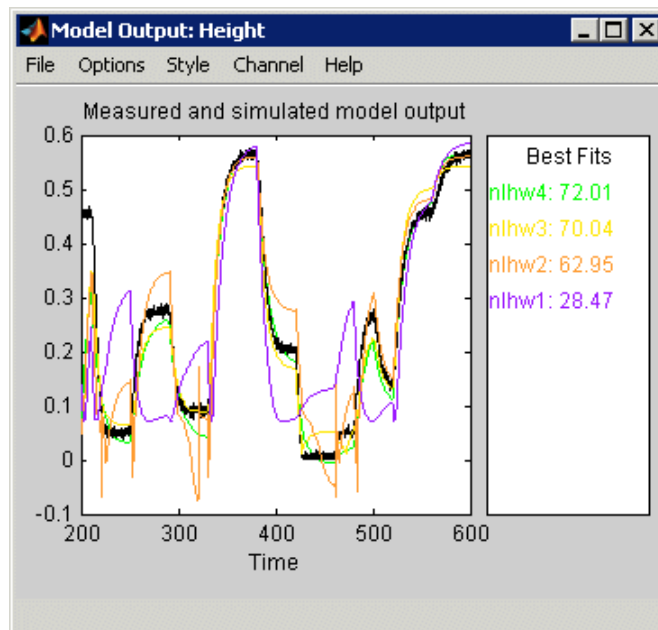
This action updates the corresponding **No. of Units** cell to 10 sigmoid units, as shown in the following figure.

Channel Names	Nonlinearity	No. of Units	
<b>Input Channels</b>			
Voltage	Sigmoid Network	10	
<b>Output Channels</b>			
Height	Piecewise Linear	10	Initial Value...

## 5 Click **Estimate**.

This action adds the model n1hw4 to the System Identification Tool GUI. It also updates the Model Output window, as shown in the following figure.

The **Best Fits** area of the Model Output window shows that the n1hw4 fit is 72.01%.



---

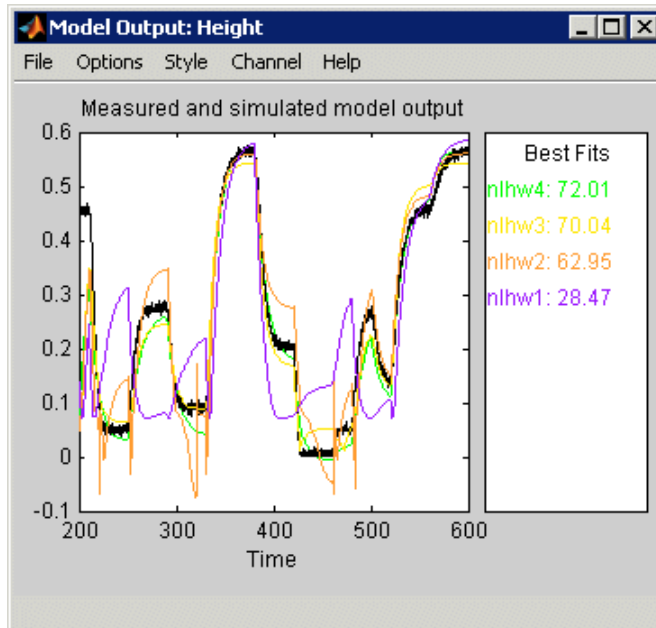
**Tip** If you know that your system includes saturation or dead-zone nonlinearities, you can specify these specialized nonlinearity estimators in your model. Piecewise Linear and Sigmoid Network are nonlinearity estimators for general nonlinearity approximation.

---

## 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 “Changing the Nonlinearity Estimator in a Hammerstein-Wiener Model” on page 7-33, as shown in the following figure.



## A

### ARMAX

- estimating using the System Identification Tool 4-34

### ARX

- estimating at the command line 6-44
- estimating using Quick Start 4-23

## B

### black-box models

- advantages 3-7
- estimating using Quick Start 4-23

### Box-Jenkins models

- estimating at the command line 6-44

## C

### comparing models

- at the command line 6-53

## D

### data

- creating iddata object 6-9
- estimation versus validation 3-2
- importing MAT-file into the System Identification Tool 4-6
- importing object into the System Identification Tool 5-7
- loading into the MATLAB workspace 4-5
- plotting at the command line 6-6
- plotting iddata object 6-6
- plotting in the System Identification Tool 4-11
- preprocessing in the System Identification Tool 4-11

### delay

- estimating at the command line 6-22

- estimating using the System Identification Tool 4-29

- disturbance model. *See* noise model

- dynamic system 3-2

## E

- estimation data 3-2

- exporting models

  - to MATLAB 4-48

  - to the LTI Viewer 4-50

## F

- frequency-response models

  - estimating at the command line 6-17

  - estimating using Quick Start 4-23

## G

- grey-box models

  - advantages 3-6

## I

- iddata object

  - creating 6-9

  - plotting 6-6

- importing data

  - iddata object into the System Identification Tool 5-7

  - MAT-file into the System Identification Tool 4-6

- impulse-response models

  - estimating at the command line 6-17

  - estimating using Quick Start 4-23

## L

- linear models

  - estimating at the command line 6-3

- estimating using the System Identification Tool 4-3
- versus nonlinear 3-4
- loading data into the MATLAB Workspace
  - browser 4-5
- LTI Viewer 4-50

## M

- model order
  - estimating at the command line 6-25
  - estimating using the System Identification Tool 4-29
- model parameters
  - viewing in the System Identification Tool 4-45
- models
  - choosing linear or nonlinear 3-4
  - defining structure using `idproc` 6-33
  - definition 3-2
  - estimating at the command line 6-3 6-33
  - estimating low-order continuous-time transfer functions 5-3
  - estimating noise models 5-22
  - estimating using Quick Start 4-23
  - estimating using the System Identification Tool 4-3 5-3
  - using 6-56

## N

- noise model 3-3
  - estimating for model 5-22
- nonlinear models
  - when to estimate 3-4
- nonparametric model
  - analyzing plots 4-23

## P

- plotting data

- at the command line 6-6
- in the System Identification Tool 4-11
- plotting models
  - in the LTI Viewer 4-50
- prediction
  - at the command line 6-56
- preprocessing data
  - in the System Identification Tool 4-11

## Q

- Quick Start
  - for estimating models 4-23
  - for preprocessing data 4-19

## R

- removing data sets from the System Identification Tool 4-20
- residuals
  - plotting using the System Identification Tool 4-42

## S

- simulation
  - at the command line 6-56
  - using the Simulink software 5-30
- Simulink software 5-30
- state-space models
  - estimating at the command line 6-44
  - estimating using Quick Start 4-23
  - estimating using the System Identification Tool 4-34
- step-response models
  - estimating at the command line 6-17
  - estimating using Quick Start 4-23
- System Identification Tool
  - estimating continuous-time models 5-3
  - estimating linear models 4-3
  - estimating models using Quick Start 4-23

- exporting models to MATLAB 4-48
- exporting models to the LTI Viewer 4-50
- removing data sets 4-20
- saving sessions 4-20
- starting 2-3 4-5
- versus command line 2-2

System Identification Toolbox product

- about 1-1
- demos 1-5
- documentation 1-5
- related products 1-7
- resources 1-9
- steps for using 2-4
- using with Simulink software 5-30

## **T**

transient-response models

- estimating at the command line 6-17
- estimating using Quick Start 4-23

trash 4-20

## **U**

using the System Identification Toolbox product 2-4

## **V**

validating models

- at the command line 6-37
- using the System Identification Tool 4-38

validation data 3-2