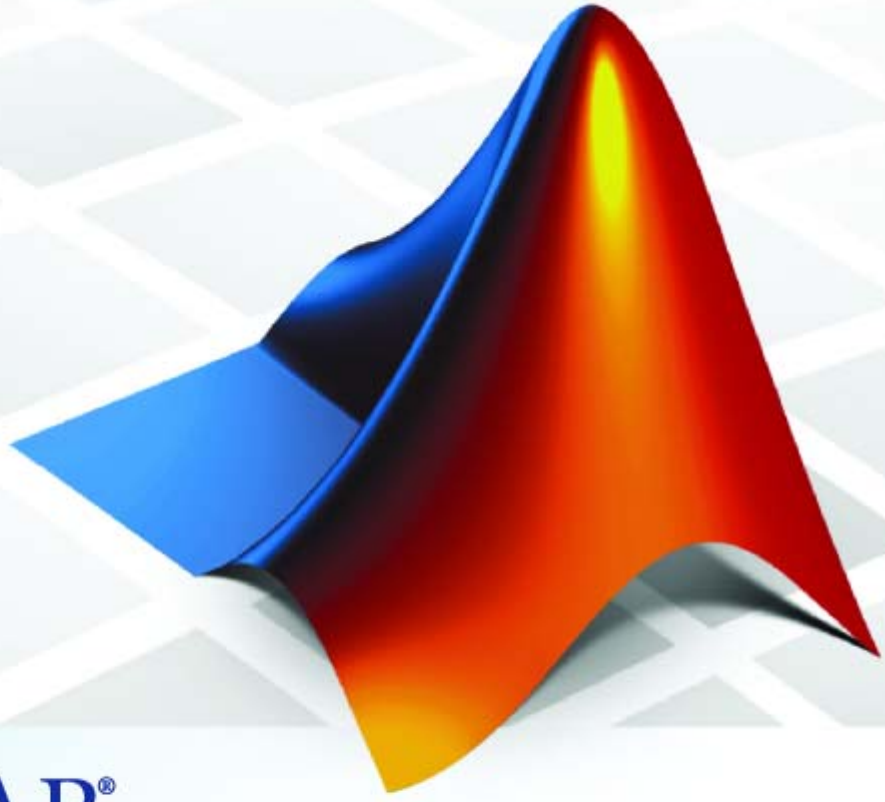
Getting Started with **System Identification Toolbox 7**

Lennart Ljung



MATLAB®
&SIMULINK®

The MathWorks
Accelerating the pace of engineering and science

How to Contact The MathWorks

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

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

508-647-7000 (Phone)

508-647-7001 (Fax)

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

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

Getting Started with System Identification Toolbox

Revision History

March 2007 First printing New for Version 7.0 (Release 2007a)

About the Developers

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

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

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

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

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

Contents

1 Introduction to System Identification Toolbox

What Is System Identification Toolbox? **1-2**
Stages of the Identification Process **1-3**
Installing System Identification Toolbox **1-3**
Accessing Demos **1-3**

Using the Documentation **1-5**

Related Products **1-6**

Working in the GUI Versus the MATLAB Command Window **1-8**

System Identification Workflow **1-9**
System Identification Stages **1-9**
Example of a Common Workflow **1-11**

Estimating Dynamic Models **1-15**
Choosing Grey-Box or Black-Box Models **1-17**
Do I Need Linear or Nonlinear Models? **1-18**
Modeling Multiple-Output Systems **1-19**
Choosing Continuous-Time or Discrete-Time Models **1-21**
Estimating a Noise Model **1-22**
Supported Black-Box Models **1-24**

Validating Models **1-36**
Comparing Model Output and Measured Output **1-36**
Analyzing Residuals **1-40**

Additional Resources **1-43**

Estimating Linear Models Using the System Identification Tool

2

About This Tutorial **2-3**

Before You Begin **2-4**
Loading Data into MATLAB Workspace **2-4**
Starting the System Identification Tool **2-4**

Importing Data into the System Identification Tool . . . **2-6**

Plotting and Preprocessing Data **2-12**

Removing Unused Data Sets and Saving the Session . . **2-21**

Estimating Preliminary Models **2-23**
Using Quick Start to Estimate Preliminary Models **2-23**
Analyzing Preliminary Models **2-25**

Fine-Tuning Model Orders and Delays **2-29**
Estimating a Range of Best-Fit Orders **2-29**
Trying State-Space and ARMAX Structures **2-34**
Choosing the Best Model **2-39**

Exporting the Model to the MATLAB Workspace **2-48**

Exporting the Model to the LTI Viewer **2-50**

Estimating Continuous-Time Process Models Using the System Identification Tool

3

About This Tutorial **3-3**

Before You Begin **3-5**
Loading Data into the MATLAB Workspace **3-5**
Starting the System Identification Tool **3-5**

Importing Data into the System Identification Tool . . . **3-7**

Plotting and Preprocessing Data **3-10**

Estimating Second-Order Process Models with Complex Poles **3-14**
Estimating an Initial Model **3-14**
Validating the Initial Model **3-19**

Fine-Tuning the Process Model **3-22**
Estimating Models with Modified Settings **3-22**
Comparing Models **3-23**

Getting the Model Parameters **3-26**

Exporting the Model to the MATLAB Workspace **3-28**

Using Simulink with System Identification Toolbox . . **3-29**
Preparing Input Data in the MATLAB Workspace **3-29**
Building the Simulink Model **3-30**
Configuring Blocks and Simulation Parameters **3-31**
Running the Simulation **3-35**

4 Estimating Linear Models in the MATLAB Command Window

About This Tutorial **4-3**

Before You Begin **4-5**
Loading Data into the MATLAB Workspace **4-5**
Plotting the Input and Output Data **4-5**
Removing Equilibrium Values from the Data **4-7**

Representing Data for System Identification **4-9**
Creating iddata Objects **4-9**
Plotting the Data **4-11**

Estimating Nonparametric Models **4-16**
Estimating the Frequency Response **4-16**
Estimating the Step Response **4-19**

Estimating Model Orders and Delays **4-21**
Estimating the Delay **4-21**
Estimating Model Orders Using a Simple ARX Structure **4-23**

Estimating Continuous-Time Process Models **4-31**
Setting Up the Structure of the Process Model **4-31**
Estimating Model Parameters Using pem **4-35**
Validating the Process Models **4-37**
Estimating a Noise Model to Improve Results **4-39**

Estimating Black-Box Polynomial Models **4-43**
Linear ARX Model **4-43**
State-Space Model **4-46**
Box-Jenkins Model **4-49**

Comparing Models **4-52**

Using Models for Simulation and Prediction **4-54**
Simulating the Model Response **4-54**
Predicting the Model Response **4-56**

Index

1

Introduction to System Identification Toolbox

What Is System Identification Toolbox? (p. 1-2)	Description of System Identification Toolbox capabilities.
Using the Documentation (p. 1-5)	Summary of available documentation for this Toolbox.
Related Products (p. 1-6)	Products that extend and complement System Identification Toolbox.
Working in the GUI Versus the MATLAB Command Window (p. 1-8)	When to use the System Identification Tool GUI versus System Identification Toolbox objects, methods, and functions in the MATLAB Command Window.
System Identification Workflow (p. 1-9)	Summary of typical tasks in the system identification workflow.
Estimating Dynamic Models (p. 1-15)	Types of models you can estimate in System Identification Toolbox.
Validating Models (p. 1-36)	Description of two approaches for validating models.
Additional Resources (p. 1-43)	References for learning more about modeling dynamic systems and system identification theory.

What Is System Identification Toolbox?

System Identification Toolbox extends the MATLAB® computation environment and lets you fit linear and nonlinear mathematical models to input and output data from dynamic systems.

System identification is especially useful for modeling systems that you cannot easily represent in terms of first principles. Examples of such complex dynamic systems include engine subsystems, flight dynamics systems, thermofluid processes, and electromechanical systems. For real-time applications in adaptive control, adaptive filtering, or adaptive prediction, you can use System Identification Toolbox to perform recursive parameter estimation.

System Identification Toolbox supports both time- and frequency-domain data with single or multiple inputs and single or multiple outputs. Time-domain data can be real or complex. You can also use this Toolbox to estimate models for time-series data.

System Identification Toolbox provides an interactive graphical user interface (GUI) called the System Identification Tool, which is ideal for getting started with this software. Alternatively, you can use System Identification Toolbox objects, methods, and functions in the MATLAB Command Window. To learn more about the difference between these two environments, see “Working in the GUI Versus the MATLAB Command Window” on page 1-8.

You can use System Identification Toolbox commands to simulate or predict output from estimated linear and nonlinear models. However, if your model is one aspect of a larger problem, you can import linear models into other MathWorks products, such as Control System Toolbox or Simulink®. In case of control design applications, you might use System Identification Toolbox to create a plant model for the purpose of designing a plant controller.

This section discusses the following topics:

- “Stages of the Identification Process” on page 1-3
- “Installing System Identification Toolbox” on page 1-3
- “Accessing Demos” on page 1-3

Stages of the Identification Process

The system identification process includes 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 reducing model order and converting between discrete-time and continuous-time representations.
5. Simulation or prediction of future output using the model.

System Identification Toolbox supports all stages except data acquisition. This Toolbox provides limited support for experimental design, enabling you to generate input signals with different properties. You can also use this Toolbox to validate and fine-tune your experimental design by modeling preliminary data.

Installing System Identification Toolbox

System Identification Toolbox builds on the foundation of MATLAB to provide functions for estimating linear models of dynamic systems. For more information about installing MATLAB and Toolboxes, see the MATLAB Installation documentation.

Accessing Demos

System Identification Toolbox provides demonstration files that show you how to estimate models for dynamic systems from input and output data. The available demonstrations include both case studies and tutorials.

To open the demos in the Help browser, type the following at the MATLAB prompt:

```
demo
```

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

Using the Documentation

All MathWorks technical documentation is available online in the MATLAB Help browser, which you open by selecting **Help > Full Product Family Help** in the MATLAB Command Window.

System Identification Toolbox documentation contains the following components:

- *Getting Started with System Identification Toolbox* — Provides several tutorials that walk you through the most common tasks in system identification. These tutorials use sample data files that are installed with the latest version of this Toolbox.
- *System Identification Toolbox User's Guide* — Provides a complete description of System Identification Toolbox features.
- Function Reference — Contains a complete description of System Identification Toolbox objects, methods, and functions. The reference pages summarize the essential syntax and usage.
- Release Notes — Describes important changes in the most recent version of System Identification Toolbox and compatibility considerations.

New Users. If you are new to System Identification Toolbox, this Getting Started guide is designed to help you start using System Identification Toolbox quickly. Using this guide, you learn how to estimate linear models in the System Identification Tool graphical user interface (GUI) and the MATLAB Command Window. Follow along with the steps in the tutorials to learn how to perform the most common tasks in a system identification workflow.

Experienced Users. If you are already familiar with System Identification Toolbox, see *System Identification Toolbox User's Guide* for information about performing specific tasks and descriptive information about supported models.

Related Products

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

```
www.mathworks.com
```

Products That Extend System Identification Toolbox

Product	Description
Control System Toolbox	Provides extensive linear model analysis capabilities through the LTI Viewer. Uses the plant models created in System Identification Toolbox for controller design.
Optimization Toolbox	When this Toolbox is installed, you have the option of using the `lsqnonlin` optimization algorithm for nonlinear identification.
Neural Network Toolbox	Provides flexible neural-network structures for estimating nonlinear models using System Identification Toolbox.

Products That Extend System Identification Toolbox (Continued)

Product	Description
Signal Processing Toolbox	Provides extensive data analysis capabilities, including: • A range of filtering options. (System Identification Toolbox only provides the fifth-order Butterworth filter.) • Spectral analysis options. After using the advanced data processing capabilities of Signal Processing Toolbox, you can import the data into System Identification Toolbox for modeling.
Simulink	Provides System Identification blocks for simulating the models you identified using System Identification Toolbox. Also provides blocks for model estimation.

Working in the GUI Versus the MATLAB Command Window

Both the System Identification Tool GUI and System Identification Toolbox functions let you preprocess data, and estimate, validate, and compare models. Which environment you choose for performing these system identification tasks is largely a matter of preference.

New users should start by using the System Identification Tool GUI to become familiar with the product. To open the System Identification Tool window, type the following at the MATLAB prompt:

```
ident
```

Two tutorials in this Getting Started guide walk you through some of the most common tasks in the System Identification Tool. For more information about using this graphical user interface, see the sections on using the System Identification Tool GUI in the *System Identification Toolbox User's Guide*.

The following operations are only available from the MATLAB Command Window:

- Generating input and output data (see `idinput`)
- Creating linear and nonlinear grey-box models, described by an ordinary differential equation.
- Using recursive online estimation methods. See the sections on recursive parameter estimation in the *System Identification Toolbox User's Guide*.
- Converting between continuous-time and discrete-time (see `c2d` and `d2c`).
- Converting models to LTI objects (see `ss`, `tf`, and `zpk`).

Note Conversions to LTI objects require Control System Toolbox.

For a tutorial that walks you through the most common tasks in the MATLAB Command Window, see Chapter 4, "Estimating Linear Models in the MATLAB Command Window".

System Identification Workflow

System identification is an iterative process, where you estimate different models for your data and compare the model performance. Ultimately, you choose the simplest model that adequately describes the dynamics of your system.

This section discusses the following topics:

- "System Identification Stages" on page 1-9
- "Example of a Common Workflow" on page 1-11

System Identification Stages

The typical identification workflow includes four stages:

1. Prepare data for system identification by doing the following:
   - Importing data into MATLAB workspace and representing the data using System Identification Toolbox format.
   - Plotting data on a time plot or an estimated frequency response plot to examine the data features. Use the `advice` command to analyze the data for the presence of constant offsets and trends, delay, feedback, and signal excitation levels.
   - Preprocessing data by removing offsets and linear trends, interpolating missing values, filtering to emphasize a specific frequency range, or resampling using a different time interval.

2 Estimate and validate models.

System Identification Toolbox supports a wide range of linear and nonlinear models, including linear nonparametric models that estimate the impulse and frequency response at each time or frequency value, linear polynomial and state-space models, and nonlinear ARX and Hammerstein-Wiener structures. You validate each model during estimation to help you fine-tune your modeling strategy by doing the following:

- Computing and plotting model frequency and transient response.
- Comparing simulated or predicted model output to measured output.
- Computing and performing correlation tests on the residuals.

When you do not achieve a satisfactory model, you can try a different model structure and order or try another identification algorithm. In some cases, 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 further modeling.

3 Postprocess and transform models by doing the following:

- Transforming the model between discrete-time and continuous-time descriptions.

  System Identification Toolbox lets you estimate linear continuous-time models directly. For example, you can estimate low-order transfer functions, called *continuous-time process models* for time- or frequency-domain data. You can also estimate a continuous-time model of any structure for frequency-domain data.

  To get a linear continuous-time model of arbitrary order for time-domain data, estimate a discrete-time model, and then use `d2c` to transform it to a continuous-time model.

- Reducing model order based on pole-zero cancellations on a pole-zero plot or using `balred` (requires Control System Toolbox).
- Converting model to LTI Object used by Control System Toolbox (requires Control System Toolbox).
- Importing model into Simulink using the System Identification block library.

4 Use models in simulation or prediction.

   You can use System Identification Toolbox functions to simulate or predict linear and nonlinear model output. You can also import linear models and nonlinear grey-box models into Simulink.

   To design a controller for a nonlinear plant, use `lintan` or `linapp` to linearize the model, and then import the model into Control System Toolbox. For more information about these functions, see the reference pages.

Example of a Common Workflow

This section discusses how System Identification Toolbox can help you model a dynamic system to fit measured input and output data. It also explains the general approach for choosing model structure and order.

This section discusses the following topics:

- "Preparing Data for System Identification" on page 1-11
- "Estimating a Simple ARX Model" on page 1-12
- "How System Identification Toolbox Estimates Parameters" on page 1-13
- "Validating the Model" on page 1-14
- "Using Your Model" on page 1-14

See the tutorials in this Getting Started guide for detailed examples of estimating different types of models.

Preparing Data for System Identification

Suppose you measure single-input and single-output data, $u(t)$ and $y(t)$, respectively. You set up the experiment to collect the data at a sampling interval T=1 sec.

To begin the system identification process, you import data into MATLAB. If you are planning to estimate linear models, remove the means from each of the measured input-output signals. In most cases, you can model linear dynamics around zero-mean levels.

As a final step in preprocessing, you split the data set into two halves; the first half is for estimating models, and the second half is for validating models. You should validate models using an independent data set, which might be a portion of the data from a single experiment or data from a different experiment.

Estimating a Simple ARX Model

To assess the data and the degree of difficulty in identifying a model, you first estimate the simplest, discrete-time model to get a relationship between *u(t)* and *y(t)* — the ARX model. This black-box approach does not require you to model the physics of your system. For an overview of supported model structures, see "Estimating Dynamic Models" on page 1-15.

For a single-input and single-output ARX model, the difference equation is:

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

To estimate this structure using System Identification Toolbox, you must specify three parameters:

- n_a as the number of poles
- n_b as the number of b parameters (equal to the number of zeros plus 1)
- n_k as the number of samples before the input affects the output of the system

System Identification Toolbox uses these parameters to compute delayed inputs and outputs in the difference equation. For example, suppose that nonparametric analysis and physical insight into your system leads you to conclude that your system might have two poles. Therefore, try modeling the data using two delayed outputs: *y(t-1)* and *y(t-2)* (assuming that the sample interval T=1). Because a parameters appear as coefficients of delayed outputs, this corresponds to specifying n_a as 2.

From the impulse-response plot, you learn that the output is delayed by two samples in response to a pulse input. Therefore, you infer that the time delay

n_k in the system is 2 sec. Because T=1 sec, it takes 2 time samples before a change in u affects y.

There is no straightforward way to estimate the number of zeros in the system. The number of zeros corresponds to the number of delayed inputs n_b minus 1. One guideline is to have (n_b+1) be less than or equal to n_a. Because you chose n_a to be 2, it is reasonable to try n_b equal to 3.

How System Identification Toolbox Estimates Parameters

Based on the values of n_a, n_b, and n_k, System Identification Toolbox computes the following transformation of inputs *u(t)* and outputs *y(t)* by shifting *u(t)* and *y(t)* values in time:

- *u(t-2)*
- *u(t-3)*
- *y(t-1)*
- *y(t-2)*

System Identification Toolbox substitutes these transformations into the difference equations to estimate the numerical coefficients $a_1 \ldots a_n$ and $b_1 \ldots b_n$.

For example, suppose that System Identification Toolbox estimates the following coefficients for a given data set:

$$y(t) - 1.5y(t-1) + 0.7y(t-2) = 0.9u(t-2) + 0.5u(t-3) + e(t)$$

Solving this equation for *y(t)* shows that the output at time t is a linear combination of past outputs and past inputs, as follows:

$$y(t) = 1.5y(t-1) - 0.7y(t-2) + 0.9u(t-2) + 0.5u(t-3) + e(t)$$

Validating the Model

System Identification Toolbox lets you create many models and then compare them to find the best one. One way you validate models is by generating a model-output plot and comparing the simulated output to the measured output in the validation data set. You can also generate residual-analysis plots. For more information, see “Validating Models” on page 1-36.

If you simulate your model with e=0 and find that there is poor agreement between simulated output and measured output, then you should try another model, such as an ARX model with different n_a, n_b, and n_k parameters, or a completely different model structure (for example, ARMAX, Box-Jenkins, or Output-Error model).

Using Your Model

Suppose that you are interested in identifying continuous-time models. This Toolbox lets you estimate a discrete-time structure of arbitrary order and then use `d2c` to transform to a continuous-time model.

After you select the simplest model that adequately represents the system dynamics, you can use the simulation or prediction functionality of System Identification Toolbox to simulate or predict the output, respectively. You can also perform numerical analysis of model parameters to compute the poles of the system.

If you are primarily interested in designing a controller for a dynamic system, you can import your model into other MathWorks products, such as Control System Toolbox or Simulink.

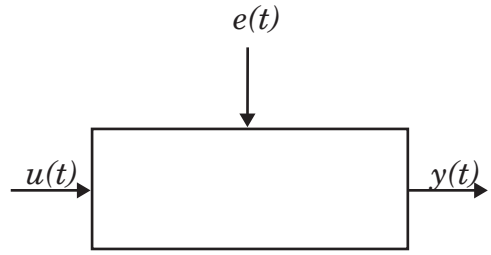
Estimating Dynamic Models

A *model* of a system is a tool that you use to answer questions about the system without having to perform experiments. For example, you might use a model to simulate the output of a system for a given input and analyze the system's response. Alternatively, you might be interested in using the model to predict future output of a system.

System Identification Toolbox lets you estimate models for dynamic systems using measured input and output data. In *dynamic systems*, the input causes the output to vary with time, and the output can depend on earlier inputs and outputs. In contrast to dynamic systems, *static systems* represent an instantaneous relationship between the input and output variables. A *time series* is a special type of dynamic system where only the output values are measured and the input values are not observed.

You get the best results when you work with at least two data sets: one that you use to estimate the models (*estimation data*), and the other you use to validate the models (*validation data*).

Models describe the relationship between one or more measured input signals, *u(t)*, and one or more measured output signals, *y(t)*. In real systems, there are additional inputs that you cannot measure or control, which affect the system's output. 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. Input and output signals can be measured in the time or frequency domain.

The most general description of a dynamic system is given by:

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

In this case, the output of a system *y(t)* is given by *g*, which might be a function of the inputs *u(t)*, time *t*, and system parameters θ. *v(t)* is the output noise.

For nonlinear models, *g* can take on a variety of forms. For more information, see "Nonlinear Black-Box Models" on page 1-32.

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

$$y = Gu + He$$

G is an operator that takes the input to the output and captures the system dynamics. *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*. System Identification Toolbox lets you choose whether to estimate the noise model.

The remainder of this section describes the types of models you can estimate using System Identification Toolbox:

- "Choosing Grey-Box or Black-Box Models" on page 1-17
- "Do I Need Linear or Nonlinear Models?" on page 1-18
- "Modeling Multiple-Output Systems" on page 1-19
- "Choosing Continuous-Time or Discrete-Time Models" on page 1-21
- "Estimating a Noise Model" on page 1-22
- "Supported Black-Box Models" on page 1-24

For a complete discussion of working with models in System Identification Toolbox, see the introduction in the *System Identification Toolbox User's Guide*.

Choosing Grey-Box or Black-Box Models

Before beginning system identification, decide whether to you want to estimate grey-box or black-box models. This section describes what you need to consider when making this decision in the following topics:

- "When to Estimate Grey-Box Models" on page 1-17
- "When to Estimate Black-Box Models" on page 1-18

After you choose between grey-box or black-box modeling, you can decide whether to estimate a linear or nonlinear model.

When to Estimate Grey-Box Models

If you understand the physics of your system and can represent the system using an ordinary differential equation (ODE), then you can use System Identification Toolbox to perform linear or nonlinear grey-box modeling. A *grey-box model* is a model where the mathematical structure of the model and possibly some of the parameters are already known. You capture the model ODE and the parameters you want to estimate in an m-file or MEX-file, and then use System Identification Toolbox objects, methods, and functions to estimate the model parameters.

Note You can only perform grey-box modeling using System Identification Toolbox objects, methods, and functions in the MATLAB Command Window. MEX-files are supported for nonlinear grey-box models only.

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, however, can be time consuming to set up and does require that you know the relationship between the system variables and the parameters. For detailed information about grey-box modeling, see the sections on estimating linear and nonlinear grey-box models in the *System Identification Toolbox User's Guide*.

When to Estimate Black-Box Models

In many real-world situations, it is too difficult to describe a system using known physical laws. In such cases, you can use System Identification Toolbox to perform black-box modeling. A *black-box model* is a flexible structure that is capable of describing many different systems and its parameters might not have any physical interpretation.

Black-box modeling has the following advantages over grey-box modeling:

- 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.

This Getting Started guide provides several tutorials on using System Identification Toolbox to estimate black-box models.

Do I Need Linear or Nonlinear Models?

After you decide whether to work with grey-box or black-box models, choose between linear and nonlinear models. System Identification Toolbox lets you estimate both linear and nonlinear grey-box and black-box models.

In practice, all systems are nonlinear and the output is a nonlinear function of the input variables. However, a linear model might accurately describe the system dynamics. Linear approximations are very useful because they are simple and provide good results in many situations. Therefore, estimate linear models first and see if these models are sufficiently accurate in representing the dynamics.

How can you tell when you need a nonlinear model? The following are useful guidelines:

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

  For example, you might be dealing with a process 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 and single-output data. However, it is often not possible to find simple transformations of measured signals, and if it is not, you then must estimate nonlinear models.

- You plot the response of the system to a specific input and notice that the responses are different depending on the input level or input sign. For example, you might see that the output response to an input step up is much faster than the response to a step down. This response behavior indicates that the system is nonlinear and you need a nonlinear model.

- You try many linear models of varying complexity, but the model output does not adequately reproduce the measured output. This inability of the model to reproduce measured output might be caused by noisy data or by nonlinear system behavior.

For more information about supported linear and nonlinear black-box models, see “Supported Black-Box Models” on page 1-24. For detailed information about grey-box modeling, see the sections on estimating linear and nonlinear grey-box models in the *System Identification Toolbox User’s Guide*.

Modeling Multiple-Output Systems

All System Identification Toolbox model types support estimation of multiple-input and single-output (MISO) models.

Modeling multiple-output systems is more challenging because input-output couplings require additional parameters to obtain a good fit, which requires

more complex models. In general, the model is better when more inputs are included, and worse when more outputs are included.

You can estimate multiple-output models both in the System Identification Tool GUI and in the MATLAB Command Window.

This section discusses the following topics:

- "Modeling Multiple-Output Models Directly" on page 1-20
- "Modeling Multiple Outputs as a Combination of Single-Output Models" on page 1-21

Modeling Multiple-Output Models Directly

Several types of System Identification Toolbox models support direct modeling of multiple-output systems. Use this approach when the outputs are likely to be coupled.

You can estimate the following types of models for multiple-output data:

- Impulse- and step-response models using correlation analysis.
- Frequency-response models using spectral analysis.
- Linear ARX models or state-space models.

Note State-space models are easier to work with because they require you to specify a single integer for the model order. In addition, you can estimate the equivalent of ARMAX and Output-Error (OE) multiple-output models using state-space model structures. For the ARMAX case, specify to estimate the K matrix for the state-space model. For the OE case, set K to zero.

- Nonlinear ARX and Hammerstein-Wiener models.

For more information about supported black-box model structures, see "Supported Black-Box Models" on page 1-24.

Modeling Multiple Outputs as a Combination of Single-Output Models

It is more difficult for models to explain the behavior of several outputs than a single output. If you get a poor fit estimating a multiple-output model directly, you can generate partial models, where each model represents the effect of all inputs on one of the outputs.

Use this approach when no feedback is present in the dynamic system and there are no couplings between the outputs. If you are unsure about the presence of feedback, use the `advice` command on the data set. For more information, see the reference pages.

To construct partial models, use subreferencing to create partial data sets, such that each data set contains all inputs and one output. Choose a different output for all partial data sets. For more information about creating partial data sets, see the following sections in the *System Identification Toolbox User's Guide*:

- For working in the System Identification Tool GUI, see "Creating Data Sets from Selected Channels".
- For working in the MATLAB Command Window, see "Subreferencing iddata Objects" and "Subreferencing idfrd Objects".

Estimate a model for each partial data set using any of the available model structures in System Identification Toolbox. After validating the single-output models, use vertical concatenation to combine these partial models into a single multiple-output model. You can try refining the multiple-output model using the original (multiple-output) data set.

For more information about concatenating models and refining estimated models, see the introduction to working with models in the *System Identification Toolbox User's Guide*.

Choosing Continuous-Time or Discrete-Time Models

System Identification Toolbox lets you estimate linear continuous-time models directly in two cases. You can estimate low-order transfer functions, called *continuous-time process models* for time- or frequency-domain data. You can

also estimate a continuous-time model of any structure for frequency-domain data.

To get a linear continuous-time model of arbitrary structure for time-domain data, you can use System Identification Toolbox to first estimate a discrete-time model and then use `d2c` to transform it to a continuous-time model.

For nonlinear models, you can only estimate discrete-time models using time-domain data.

Estimating a Noise Model

System Identification Toolbox lets you estimate a noise model for linear models structures.

Note Nonlinear ARX and Hammerstein-Wiener models do not produce parametric noise models.

This section discusses the following topics:

- "What Is a Noise Model?" on page 1-23
- "When to Estimate a Noise Model" on page 1-24

If you decide that a good noise model is important, choose the ARMAX, Box-Jenkins, or state-space model structures that include additional parameters for modeling noise. For more information about these model structures, see "Estimating Linear Nonparametric and Parametric Models".

What Is a Noise Model?

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

$$y = Gu + He$$

G is an operator that takes the input to the output and captures the system dynamics. e is an unmeasured input that is the noise source. H is an operator that describes how the system forms the additive noise from e and is called a *disturbance model*, or *noise model*.

System Identification Toolbox algorithms assume that the noise source e is *white noise*, which means that it is entirely unpredictable. In other words, it is not possible to guess the value of $e(t)$ regardless of how accurately you measured the past data, up to time t-1.

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 typically going to be 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—all that matters is the spectrum of He.

When to Estimate a Noise Model

You might estimate the noise model when:

- 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, *G*, by emphasizing during the estimation the frequencies that are least affected by noise.

Tip To see if estimating a noise model *H* might help you improve the dynamic model *G*, try comparing the output of the models with and without noise. If the simulations differ significantly, then you might need a noise model to improve the dynamic model.

You might omit estimating a noise model when you have a good signal-to-noise (SNR) ratio. With a good SNR, information about *G* in the data is not corrupted.

Supported Black-Box Models

If you choose to estimate black-box models, System Identification Toolbox provides the following five categories of black-box models:

- "Linear Nonparametric Models" on page 1-25
- "Linear Continuous-Time Process Models" on page 1-27
- "Linear Parametric Polynomial Models" on page 1-28
- "Linear Parametric State-Space Models" on page 1-31
- "Nonlinear Black-Box Models" on page 1-32

This section introduces some common terms for describing models and provides an overview the supported models, including whether you can estimate continuous-time or discrete-time models and any restrictions on the data.

Definitions of Terms Describing Models

Nonparametric models consist of data tables or curves and are not represented by a compact mathematical formula with adjustable parameters. Thus, nonparametric models do not impose a structure on your system. Typical nonparametric methods for linear models include *correlation analysis*, which estimates the impulse or step response of the system, and *spectral analysis*, which estimates the frequency response (periodogram) of the system.

Parametric models have a well-defined mathematical structure, and this structure is fit to the input-output data by adjusting the coefficient values, or *model parameters*. Parametric identification methods use numerical search to find the parameter values that correspond to the best agreement between simulated and measured output.

For linear parametric models, System Identification Toolbox provides a number of *model structures*, including several types of polynomial structures (ARX, ARMAX, Output-Error, and Box-Jenkins) and state-space structures. For nonlinear parametric models, System Identification Toolbox offers the nonlinear ARX and the Hammerstein-Wiener structures.

In practice, input and output signals are collected at specific time intervals, or *samples*. A *discrete-time model* expresses the relationship between the values of the signals at the sampling instants. Such models are typically described by difference equations. A *continuous-time model* describes the relationship between continuous-time input and output signals. You typically represent continuous-time systems using differential equations.

Linear Nonparametric Models

System Identification Toolbox supports two types of linear, nonparametric models, which are described in the following table.

Model Type	Description	Data Domain
Transient-response model using correlation analysis.	Estimates the impulse response g_k for a linear system described by: $$y(t) = \sum_{k=1}^{\infty} g_k u(t-k)$$	Both time- and frequency-domain data can be used for estimating g_k.
Frequency-response model using spectral analysis.	Estimates the frequency response *G(z)* for a linear system. For a discrete-time system sampled with a time interval *T*, the transfer function relates the Z-transforms of the input *U(z)* and output *Y(z)*: $$Y(z) = G(z)U(z)$$ The frequency function $G(e^{iwT})$ is the transfer function *G(z)* evaluated on the unit circle.	Both time- and frequency-domain data can be used for estimating *G(z)*.

There are two types of transient response for a dynamic model:

- Impulse response.
- Step response.

The *impulse response* is the output signal that results when the input is an impulse, defined as follows for a discrete model:

$$u(t) = 0 \quad t \neq 0$$
$$u(t) = 1 \quad t = 0$$

The step response is the output signal that results from a step input, defined as follows:

$$u(t) = 0 \quad t < 0$$
$$u(t) = 1 \quad t \geq 0$$

The response to an input *u(t)* is equal to the convolution of the impulse response, as follows:

$$y(t) = \int_0^t h(t-z) \cdot u(z) dz$$

The *frequency response* describes the steady-state response of a system as a function of frequency in terms of peak amplitude and phase. The response at a particular frequency represents the amplitude and phase of the response to a sinusoidal input. For a linear system, the response to a sinusoidal input is also a sinusoid with the same frequency. The transfer function magnifies the amplitude of the input and shifts its phase. You use a Bode plot to visualize the system's frequency response, which shows the amplitude change and the phase shift as a function of the sinusoid's frequency.

Nonparametric models serve well as preliminary models that you can use to analyze system characteristics. For example, estimating transient response provides insight into the rise time and settling time of the system response. Similarly, estimating frequency response might indicate the order of the system, locations of resonances and notches, crossover frequencies, and the bandwidth of the system. You can also use transient-response models for simulation.

Linear Continuous-Time Process Models

Continuous-time process models represent linear system dynamics in terms of a static gain, a time delay before the system output responds to the input, and characteristic time constants associated with poles, zeros, and the time delay. As the name implies, this type of model is continuous-time only.

The System Identification Tool lets you create different process-model structures for both time-domain and frequency-domain data by varying the number of poles, adding an integrator, or adding or removing a time delay or a zero. In this Toolbox, you can specify a first-, second-, or third-order model, and the poles can be real or complex (underdamped modes).

Note Continuous-time process models let you estimate the input delay.

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

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

Linear Parametric Polynomial Models

The simplest models are linear, discrete-time difference equations, such as the following:

$$y(t) + a_1 y(t - T) + a_2 y(t - 2T) = b_1 u(t - T) + b_2 u(t - 2T)$$

y(t) is the output, *u(t)* is the input, and T is the sampling interval. This Getting Started guide uses the time-shift operator q^{-1} to compactly represent difference equations, where $q^{-1}u(t) = u(t - T)$:

$$y(t) + a_1 q^{-1} y(t) + a_2 q^{-2} y(t) = \mathrm{b}_1 q^{-1} u(t) + b_2 q^{-2} u(t)$$

or

$$A(q)y(t) = B(q)u(t)$$

In this case, $A(q) = 1 + a_1 q^{-1} + a_2 q^{-2}$ and $B(q) = b_1 q^{-1} + b_2 q^{-2}$.

System Identification Toolbox supports several such linear polynomial model structures and the general representation for this family of polynomials is:

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

The polynomials A, B_i, C, D, and F_i contain the time-shift operator q. u_i is the ith input, nu is the total number of inputs, and nk_i is the ith input delay.

Note This form is completely equivalent to the Z-transform form: q corresponds to z.

After you select a model structure, you must choose a *model order*, which is one or more integers that define the complexity of the model. The exact definition of model order varies with the model structure, but is related to the number of poles, the number of zeros, and the response delay time—*dead time*—given by the number of samples before the output responds to the input.

For a single-input and single-output ARX model, which contains only the A and B polynomials, the difference equation is:

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

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.

System Identification Toolbox estimates the parameters $a_1 \ldots a_n$ and $b_1 \ldots b_n$ using the input and output data from the estimation data set.

If you have a specific structure in mind for your application, you can decide whether the dynamics and the noise have common or different poles. *A(q)* corresponds to poles that are common for the dynamic model and the noise model. Using common poles for dynamics and noise is useful when the disturbances enter the system at the input. F_i determines the poles unique to the system dynamics, and D determines the poles unique to the disturbances.

In continuous time, the general frequency-domain equation is written in terms of the Laplace transform variable s, which corresponds to differentiation:

$$A(s)Y(s) = \frac{B(s)}{F(s)}U(s) + \frac{C(s)}{D(s)}E(s)$$

In the continuous-time case, the underlying time-domain model is a differential equation and the model order integers represent the number of estimated numerator and denominator coefficients. For example, n_a=4 and n_k=2 correspond to the following model:

$$A(s) = s^4 + a_1 s^3 + a_2 s^2 + a_3$$
$$B(s) = b_1 s + b_2$$

The standard approach for obtaining continuous-time models is to first estimate polynomial models in discrete time and then use `d2c` to convert this discrete-time model to continuous-time form.

The following table summarizes the types of linear polynomial model structures supported in System Identification Toolbox.

Model Structure	Discrete-Time Form
ARX	$$A(q)y(t) = \sum_{i=1}^{nu} B_i(q)u_i\left(t - nk_i\right) + e(t)$$
ARMAX	$$A(q)y(t) = \sum_{i=1}^{nu} B_i(q)u_i\left(t - nk_i\right) + C(q)e(t)$$

Model Structure	Discrete-Time Form
OE (Output-Error)	$$y(t) = \sum_{i=1}^{nu} \frac{B_i(q)}{F_i(q)} u_i\left(t - nk_i\right) + e(t)$$ **Note** This form excludes a noise model. Thus, the white noise source *e(t)* affects only the output and corresponds to $H = 1$ in the general equation.
BJ (Box-Jenkins)	$$y(t) = \sum_{i=1}^{nu} \frac{B_i(q)}{F_i(q)} u_i\left(t - nk_i\right) + \frac{C(q)}{D(q)} e(t)$$

Linear Parametric State-Space Models

Continuous state-space models use state variables to describe a system by a set of first-order differential equations, rather than by one or more *n*th-order differential equations. State variables *x(t)* can be reconstructed from the measured input-output data need. State variable might not have physical meaning.

Discrete-time state-space models provide the same type of linear difference relationship between the inputs and the outputs as the ARX model, but they are rearranged such that there is only one delay in the expressions. The discrete-time state-space model structure is often written in the *innovations form* that describes noise:

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

The *order* of the state-space model is an integer equal to the dimension of *x(t)* and relates to the number of delayed inputs and outputs used in the corresponding linear difference equation.

In continuous-time the state-space description has the following form:

$$\dot{x}(t) = Fx(t) + Gu(t) + \tilde{K}e(t)$$
$$y(t) = Hx(t) + Du(t) + e(t)$$

The relationships between the discrete state-space matrices A, B, C, D, and K and the continuous-time state-space matrices F, G, H, D, and $\tilde{K}$ are given for piecewise-constant input, as follows:

$$A = e^{FT}$$
$$B = \int_0^T e^{F\tau} G d\tau$$
$$C = H$$

The exact relationship between K and $\tilde{K}$ is complicated. However, for short sampling intervals T, the following approximation works well:

$$K = \int_0^T e^{F\tau} \tilde{K} d\tau$$

Nonlinear Black-Box Models

System Identification Toolbox lets you estimate nonlinear discrete-time black-box models for both single-output and multiple-output time-domain data. You can choose from two types of nonlinear, black-box model structures:

- Nonlinear ARX models.
- Hammerstein-Wiener models.

To learn how to estimate nonlinear black-box models using the System Identification Tool GUI or commands in the MATLAB Command Window, see the sections on estimating nonlinear models in *System Identification Toolbox User's Guide*.

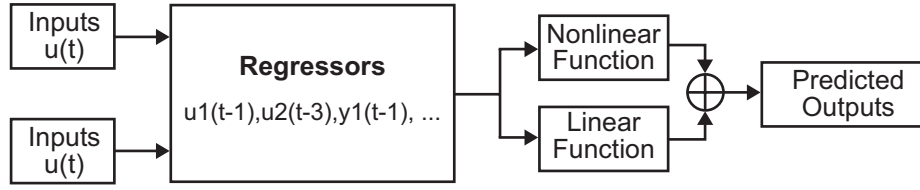
Note You can estimate nonlinear black-box models from input-output data only. These models do not support time-series data, where there is no input.

Nonlinear ARX model. Describes nonlinear structures using a parallel combination of nonlinear and linear blocks. The nonlinear and linear functions are expressed in terms of variables called regressors, which System Identification Toolbox calculates using the models orders you specify.

The model output is a function of the *regressors*, such that:

$$\hat{y} = g\left(y(t-1), u(t-1), y(t-2), \ldots\right)$$

The function g is a combination of a linear function and a nonlinear function. The nonlinear function might be a binary partition tree, a neural network, or a network based on wavelets. The following figure shows how the predicted output of the model is formed from the inputs and outputs.



System Identification Toolbox computes regressors by performing transformations of the measured input *u(t)* and output *y(t)* signals. For example, regressors can be delayed inputs and outputs, such as *u(t-1)* and *y(t-3)*. Regressors can also be nonlinear functions of inputs and outputs, such as *tan(u(t-1))* or *u(t-1)y(t-3)*. You can either use default regressors, or specify your own custom functions of input and output signals.

You choose a nonlinear structure that independently combines linear and nonlinear regressors and the structure of the nonlinearity itself, such as a binary partition tree or a network based on wavelets. System Identification Toolbox uses input-output data to find the linear and nonlinear mappings that give the best predicted outputs of the nonlinear model.

For an example of estimating nonlinear ARX models, see *System Identification Toolbox User's Guide*.

Hammerstein-Wiener model. Describes nonlinear structures using up to two static nonlinear blocks (no dynamics) in series with a linear block. Specifically, the input signal comes into a static nonlinearity, then goes into a linear dynamic system, and finally passes into a second static nonlinearity, as shown in the following figure.

In System Identification Toolbox, the linear block is a discrete-time transfer function and the nonlinear blocks are implemented using nonlinearity estimators.

For an example of estimating Hammerstein-Wiener models, see *System Identification Toolbox User's Guide*.

Validating Models

After you estimate a model, you need to determine if this model is sufficiently accurate for your application—*validate* the model. Typically, you validate each model directly after estimation to determine whether this model is accurate enough, or whether to try another model.

The section "Estimating Dynamic Models" on page 1-15 recommends that you designate two types of data sets in System Identification Toolbox: one for estimating the models (*estimation data*), and the other for validating the models (*validation data*). During model validation, System Identification Toolbox uses the validation data for comparing measured output to modeled output, computing residuals, and computing the prediction errors.

You can use either of the following approaches for validating models in System Identification Toolbox:

- "Comparing Model Output and Measured Output" on page 1-36
- "Analyzing Residuals" on page 1-40

For more information about these and other validation approaches, see the sections on plotting and validating models in the *System Identification Toolbox User's Guide*.

Comparing Model Output and Measured Output

System Identification Toolbox lets you validate a model by checking how well its simulated or predicted output matches the measured output.

The approach you use to validate the model output should match how you plan to use the model. If you plan to use the model in simulation applications, then compare the simulated model output to the validation data. However, if you plan to use the model for prediction, then compare the predicted output from the model to the measured output.

This section discusses the following topics:

- "Difference Between Simulation and Prediction" on page 1-37
- "Comparing Outputs in the System Identification Tool" on page 1-38

• "Comparing Output Using the compare Function" on page 1-39

Difference Between Simulation and Prediction

The main difference between simulation and prediction is whether System Identification Toolbox uses measured or computed previous outputs for computing the next output. Whether you have a continuous-time or a discrete-time model, System Identification Toolbox represents the model as a difference equation behind-the-scenes, where the current output is some function of the previous and current inputs and possibly the previous outputs.

When you *simulate* the model output, System Identification Toolbox computes the first output value using the initial conditions and the inputs. Then, System Identification Toolbox feeds this computed output into the difference equation for calculating the next output value. In this way, the simulation progresses using previously calculated outputs in the difference equation to produce the next output; with an *infinite prediction horizon*, the simulation has no limit on how far out in time it computes output values. Simulating models uses the input-data values from the validation data set to compute the output values.

Using a model for prediction is common in many controls applications where you want to predict the output a certain number of steps in advance and compute the control signal. When you use System Identification Toolbox to *predict* model output, the algorithm uses both the measured and the calculated output data values in the difference equation for computing the next output.

For one-step-ahead prediction, System Identification Toolbox uses all measured previous output values to compute the current output value. For example, for a model that has its current output dependent on two previous outputs, the algorithm uses the following steps to compute the output *y(4)*:

1. Computes the output *y(3)* using measured outputs $y_m(2)$ and $y_m(1)$.
2. Computes the output *y(4)* using the *measured* outputs $y_m(3)$ and $y_m(2)$ (not the computed *y(3)*.

For two-step-ahead prediction, System Identification Toolbox predicts the current and the previous output using measured output values up to two time

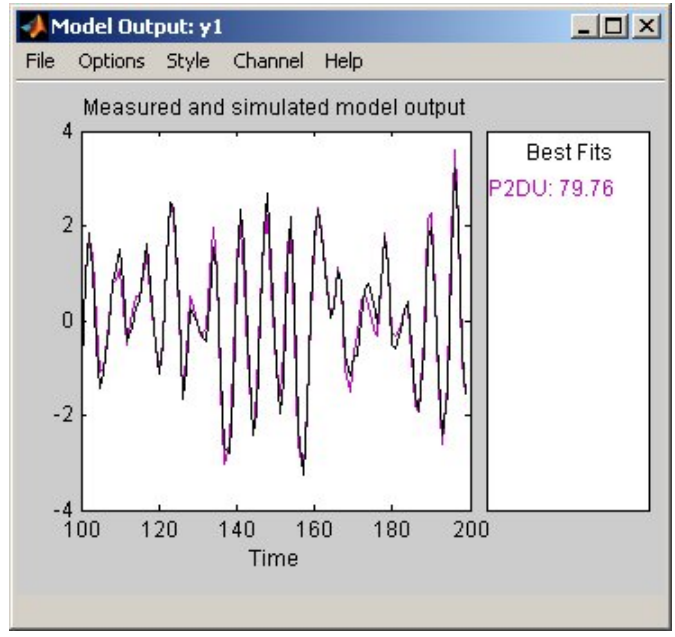
steps ago. For a model that has its current output depend on two previous outputs, the algorithm uses the following steps to compute the output *y(7)*:

1. Computes *y(6)* using the measured output values up to *t=5*.
2. Computes *y(7)* using the measured output $y_m(5)$ and the *computed* output *y(6)*.

Note When predicting output using your model, make sure that the validation data is collected under conditions similar to the estimation data with comparable signal-to-noise ratio.

Comparing Outputs in the System Identification Tool

If you are working in the System Identification Tool GUI, you can create a Model Output plot that looks similar to the one in the following figure.



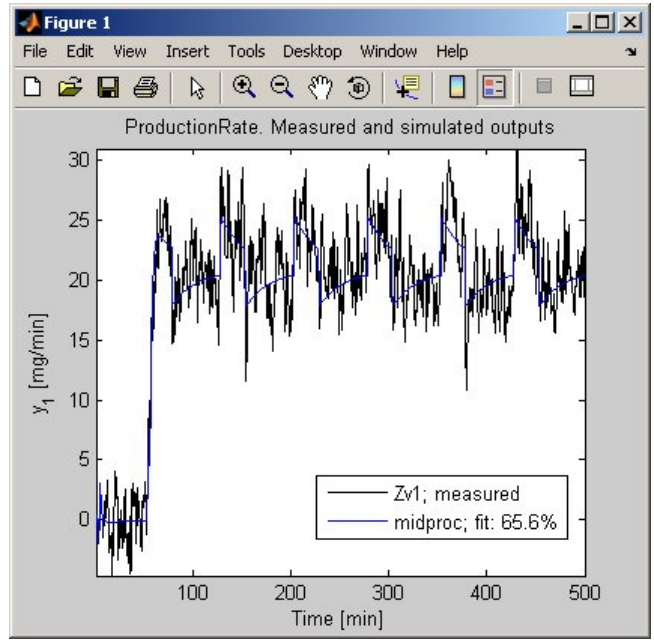
The next two chapters in this Getting Started guide provide examples of how to generate the Model Output plot using the System Identification Tool.

To generate this plot, System Identification Toolbox uses input data from the validation data set to simulate the output from the model. By default, the plot compares simulated and measured outputs. You can, however, use the **Options** menu to show predicted output instead of simulated output.

The **Best Fits** area of the Model Output plot shows the agreement (in percent) between the model output and the measured output. To get a sense for what the number means, consider that 100% corresponds to a perfect fit, and 0% indicates that the fit is no better than guessing the output to be a constant.

Comparing Output Using the compare Function

If you are working in the MATLAB Command Window, you can use the `compare` function to generate a plot that compares simulated or predicted model output to measured output, as shown in the following figure.



By default, the `compare` function assumes an infinite prediction horizon and calculates simulated output. However, if you specify a finite prediction horizon as a third argument in `compare`, the function compares predicted output to measured output instead.

Analyzing Residuals

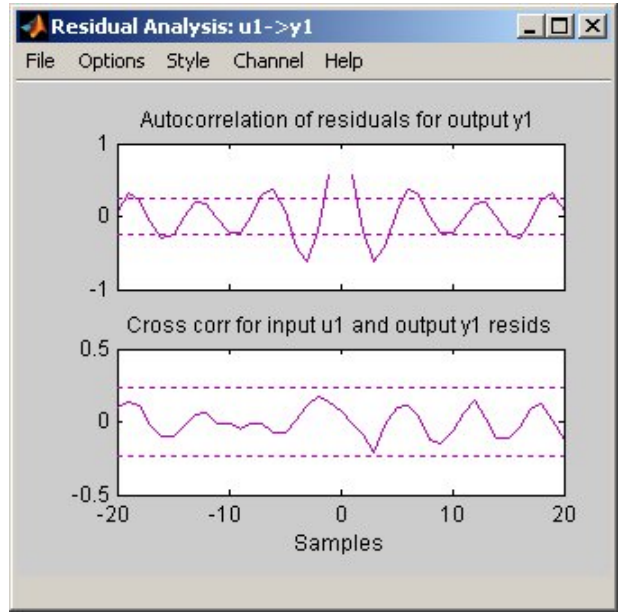
You can also test a model by checking the behavior of its residuals. *Residuals* are differences between the one-step-predicted output from the model and the measured output from the validation data. Thus, residuals represent the portion of the validation data not explained by the model.

Residual analysis consists of two tests: the whiteness test and the independence test.

Whiteness Test. According to the *whiteness test* criteria, a good model has the residual autocorrelation function inside the model confidence interval, indicating that the residuals are uncorrelated.

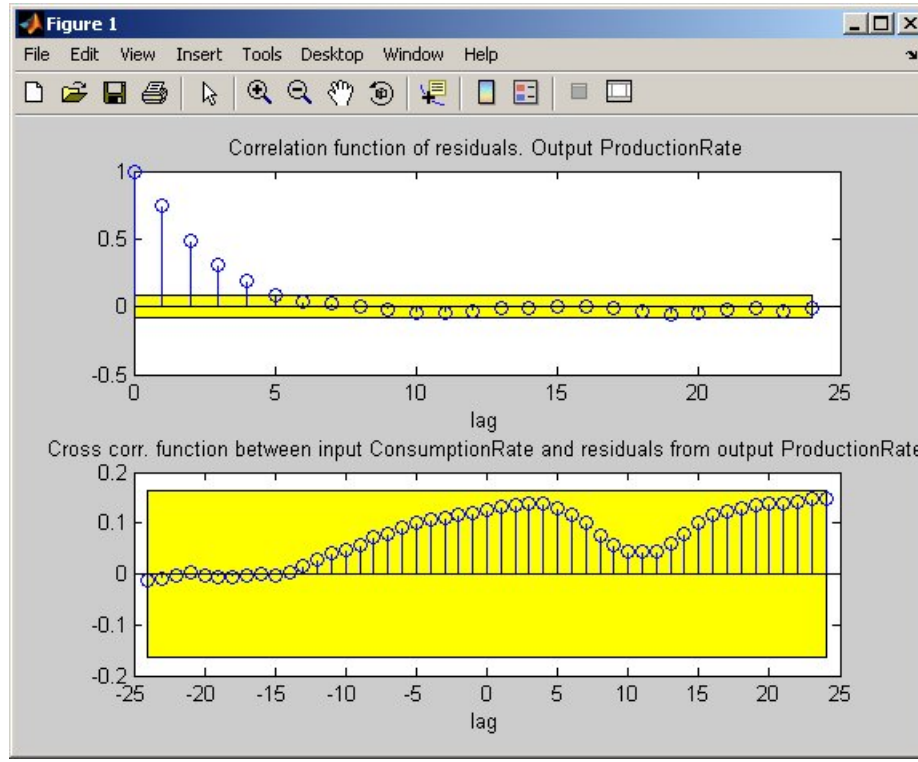
Independence Test. According to the *independence test* criteria, a good model has residuals uncorrelated with past inputs. Evidence of correlation indicates that the model does not describe how part of the output relates to the corresponding input. For example, a peak outside the confidence interval for lag k means that the output *y(t)* that originates from the input *u(t-k)* is not properly described by the model.

If you are working in the System Identification Tool GUI, you can create a Residual Analysis plot that looks similar to the following figure.



In the Residual Analysis plot, there are two sets of axes. The top axes show the autocorrelation of residuals (whiteness test). The horizontal dashed lines on the plot represent the model confidence interval. The horizontal scale is the number of *lags*, or the difference between correlated time steps. Any fluctuations within the confidence interval are considered random, indicating the residuals are mostly uncorrelated. The bottom axes show the cross-correlation of the residuals with past inputs (independence test).

If you are working in the MATLAB Command Window, use `resid` to perform residual analysis and generate a plot similar to the one in the following figure.



In this figure window, the confidence region displays as a yellow horizontal band. For more information about this command, see the reference pages.

Additional Resources

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, N.J., 1994.

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

- Ljung, L. *System Identification: Theory for the User*. 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.

2

Estimating Linear Models Using the System Identification Tool

About This Tutorial (p. 2-3)	Overview of the tutorial and the sample data.
Before You Begin (p. 2-4)	How to load the sample MAT-file into MATLAB workspace and open the System Identification Tool GUI.
Importing Data into the System Identification Tool (p. 2-6)	How to import time-domain data into the System Identification Tool.
Plotting and Preprocessing Data (p. 2-12)	How to create a time plot of the data, subtract the mean values of the input and the output, and split the data into two halves to use one half for model estimation, and the other half for model validation.
Removing Unused Data Sets and Saving the Session (p. 2-21)	How to save a System Identification Tool session, including imported data sets and generated models.
Estimating Preliminary Models (p. 2-23)	How to estimate models using Quick Start to assess the complexity of the data and the performance of several polynomial and state-space models, including the possible model orders and input-to-output delays.

Fine-Tuning Model Orders and Delays (p. 2-29)	How to use the interactive ARX Model Structure Selection dialog box to estimate a range of possible model orders and systematically fine-tune the model structure.
Exporting the Model to the MATLAB Workspace (p. 2-48)	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. 2-50)	How to export models to the LTI Viewer, which is available if you have installed Control System Toolbox.

About This Tutorial

By performing the steps in this tutorial, you get an overview of estimating linear models using the System Identification Tool graphical user interface (GUI). This tutorial is based on the example in *System Identification: Theory for the User*, Second Edition, by Lennart Ljung, Prentice Hall, 1999.

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

Note The tutorial uses time-domain data to demonstrate how you can estimate linear models. This workflow also applies to frequency-domain data. To learn more about frequency-domain data, see the chapter on representing data for system identification in the *System Identification Toolbox User's Guide*.

This tutorial describes the following tasks:

- "Loading Data into MATLAB Workspace" on page 2-4
- "Starting the System Identification Tool" on page 2-4
- "Importing Data into the System Identification Tool" on page 2-6
- "Plotting and Preprocessing Data" on page 2-12
- "Removing Unused Data Sets and Saving the Session" on page 2-21
- "Estimating Preliminary Models" on page 2-23
- "Fine-Tuning Model Orders and Delays" on page 2-29
- "Exporting the Model to the MATLAB Workspace" on page 2-48
- "Exporting the Model to the LTI Viewer" on page 2-50

For general information about using the System Identification Tool GUI, see the *System Identification Toolbox User's Guide*.

Before You Begin

Before you can perform the tasks described in this tutorial, you must do the following preparation:

- "Loading Data into MATLAB Workspace" on page 2-4
- "Starting the System Identification Tool" on page 2-4

Loading Data into MATLAB Workspace

Load sample data in `dryer2.mat` by typing the following command at the MATLAB prompt:

```
load dryer2
```

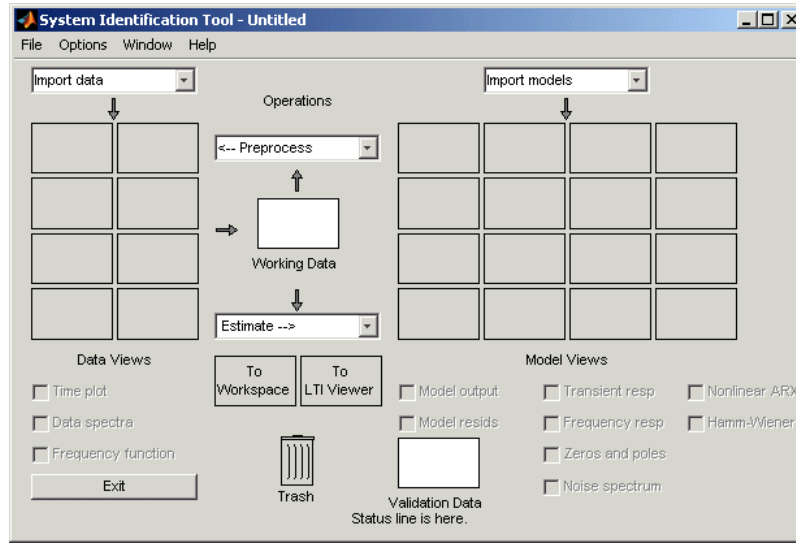
This command loads the data into MATLAB workspace as two column vectors, `u2` and `y2`, respectively. The variable `u2` is the input data and `y2` is the output data.

Starting the System Identification Tool

To open the System Identification Tool GUI, type the following command at the MATLAB prompt:

```
ident
```

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



Importing Data into the System Identification Tool

After opening the System Identification Tool window, as described in “Starting the System Identification Tool” on page 2-4, you can import the sample data for this tutorial.

This portion of the tutorial shows how to import single-input and single-output (SISO) data from a sample data file `dryer2.mat` that you loaded into MATLAB workspace. The input and output signals each contain 1000 data samples.

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

---

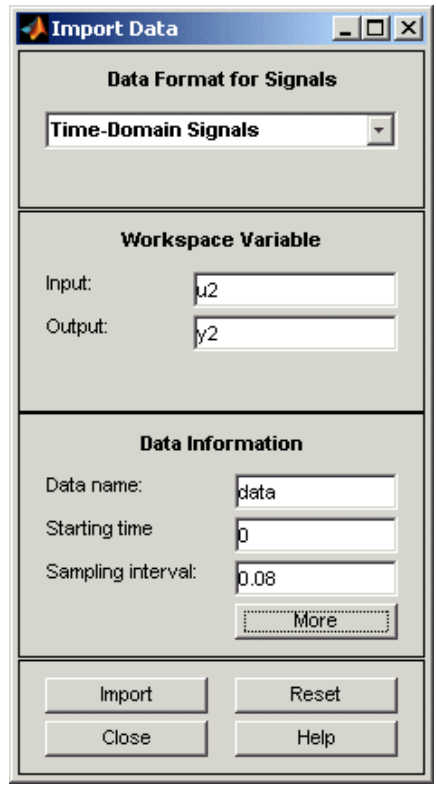
1 In the System Identification Tool window, select **Import data > Time domain data**.

2 In the Import Data dialog box, specify the following options:

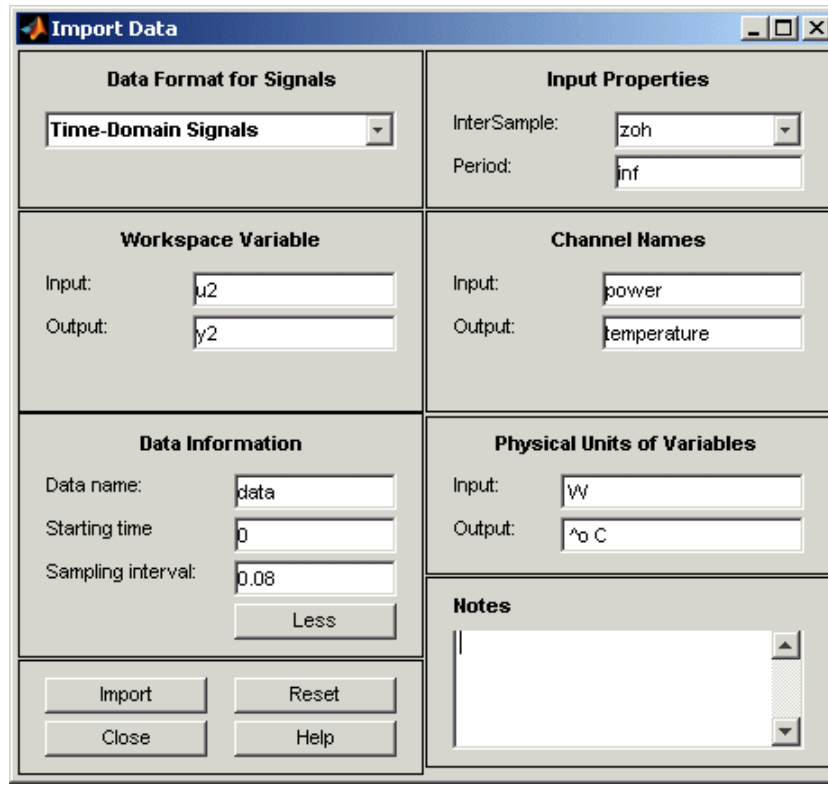
- **Input** — Enter `u2` as the name of the MATLAB variable that is the input signal.
- **Output** — Enter `y2` as the name of the MATLAB variable that is the output signal.
- **Data name** — Edit the default name of the data to `data`. This name labels the data in the System Identification Tool window 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.

  System Identification Toolbox uses the sampling interval during model estimation and to set the horizontal axis on time plots. If you transform a time-domain signal to a frequency-domain signal (see `fft`), the Fourier transforms are computed as Discrete Fourier Transforms (DFT) using this sampling interval.

The Import Data dialog box now looks like the following figure.



3 In the **Data Information** area, click **More** to expand the dialog box. Verify that the settings shown in the following figure appear in the Import Data dialog box.



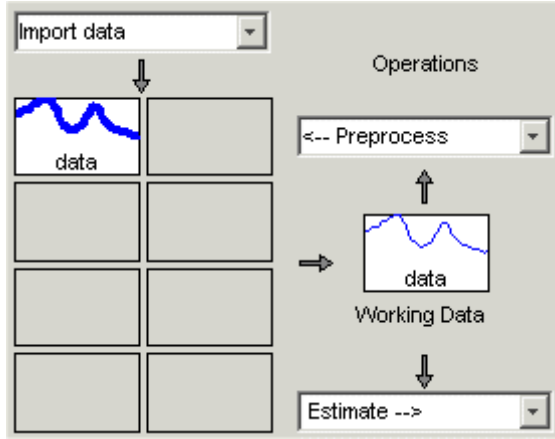
- **InterSample** — Specifies the behavior of the input signals between samples for transformations between discrete time and continuous time. It is set to `zoh` (zero-order hold) to maintain a piecewise-constant input signal between samples. Other possible values are:
  - `foh`: First-order hold maintains a piecewise-linear input signal between samples.
  - `bl`: Band-limited behavior specifies that the continuous-time input signal has zero power above the Nyquist frequency, which is equal to the inverse of the sampling interval.

  The algorithms for transforming between discrete-time and continuous-time models differ for different intersample settings (see the `d2c` and `c2d` reference pages for more information about transforming models).
- **Period** — `Inf` specifies a nonperiodic input. For a periodic input, enter the period of the input signal in your experiment.

  Note If your data is periodic, include a whole number of periods.

- **Input** channel name is set to `power`, and **Output** channel name is set to `temperature`. When you have multichannel input and output signals, specify the names of individual **Input** and **Output** channels, separated by commas. To learn how channel names can facilitate selecting data in plots, see the section on selecting channels in multivariate plots in the *System Identification User's Guide*.
- **Physical Units of Variables** — Contains a string for the **Input** (power in Watts, `W`) and the **Output** (temperature in degrees Celsius, `^oC`) channel for labeling plot axes. When you have multichannel input and output, specify the units of individual **Input** and **Output** separated by commas.
- **Notes** — Contains your comments about the origin and state of the data. For example, you might enter the experiment name, date, and a description of experimental conditions. When you estimate models from this data, the models inherit these notes.

4 Click **Import** to add the icon named `data` to the Data Board.



5 Click **Close** to close the Import Data dialog box.

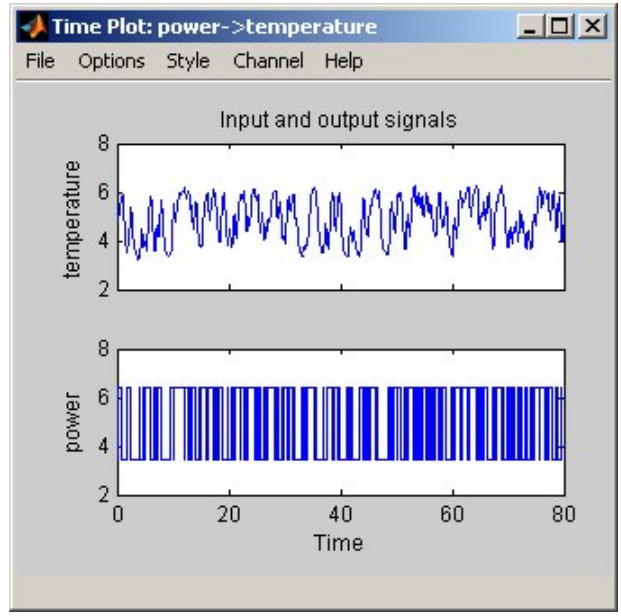
Plotting and Preprocessing Data

After importing data into the System Identification Tool, as described in "Importing Data into the System Identification Tool" on page 2-6, prepare the data for system identification.

This portion of the tutorial shows how to plot the data, subtract the mean values of the input and the output, and split the data into two halves. You use one half of the data for model estimation, and the other half of the data for model validation.

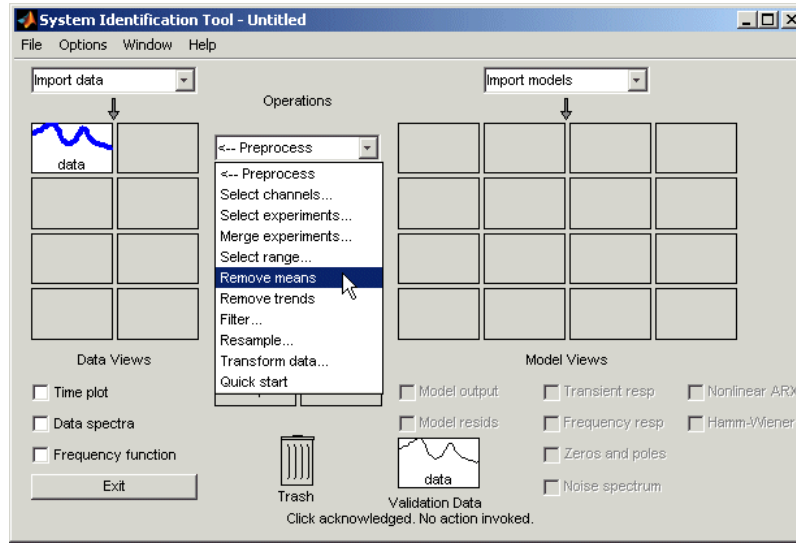
The following steps demonstrate the most common preprocessing operations. For information about other types of preprocessing, such as resampling and filtering the data, see the *System Identification Toolbox User's Guide*.

1 In the System Identification Tool window, select the **Time plot** check box to display the time plot.

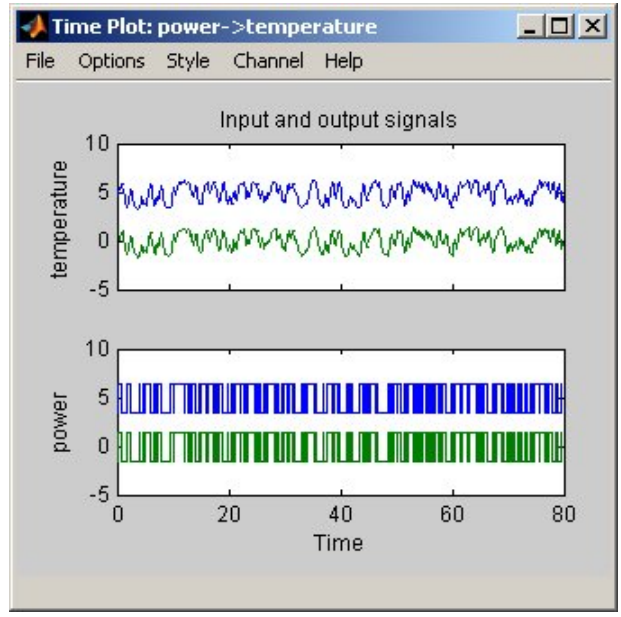


The top axes show the output data (temperature), and the bottom axes show the input data (power). Both the input and the output data have nonzero mean values.

2 In the System Identification Tool window, 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 Data Board with the default name `datad` (the suffix *d* means *detrend*), and updates the time plot to display both the original and the detrended data.



The detrended data has a zero mean value.

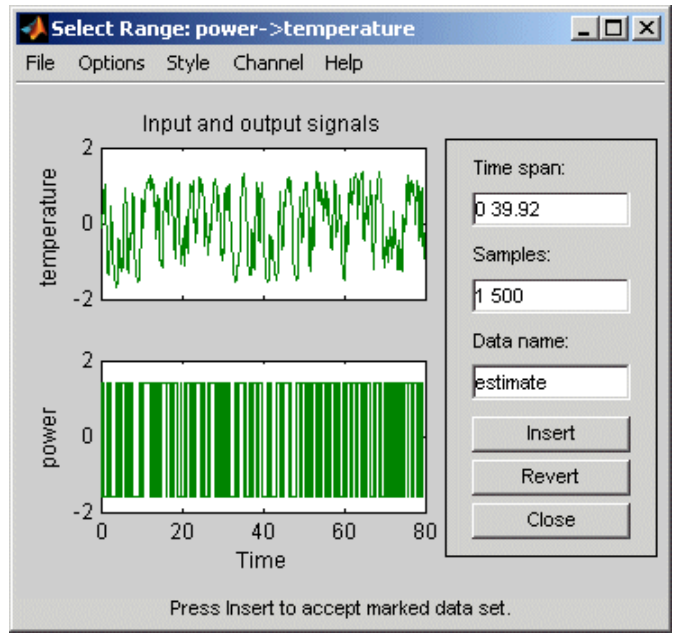
3 In the System Identification Tool window, drag the `datad` data set to the **Working Data** rectangle.

4 Select **Preprocess > Select range** to open the Select Range dialog box.

In this dialog, you can split the data into two halves and designate the first half for model estimation, and the second half for model validation, as described in the following steps.

5 In the Select Range dialog box, edit the **Samples** field to select the first 500 samples, as follows:

```
1  500
```

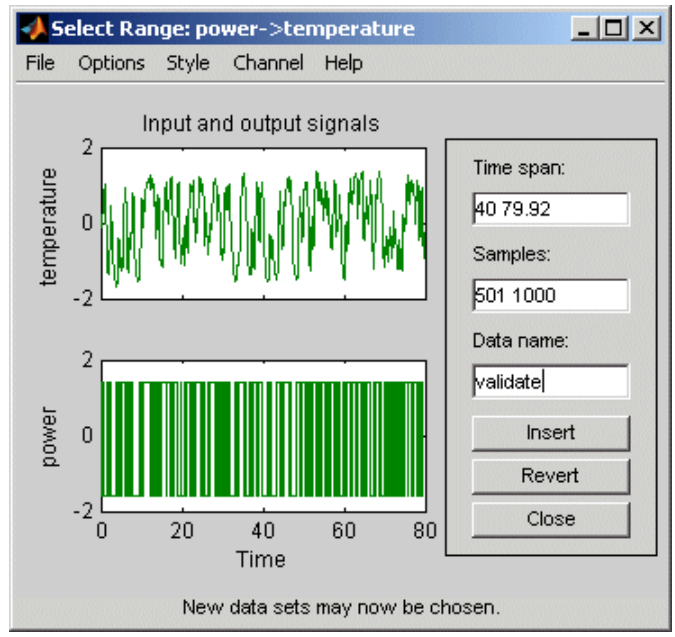


Tip You can also select data samples using the mouse by clicking and dragging a rectangular region on the plot. If you select samples on the input-channel axes, the corresponding region is also selected on the output-channel axes.

6 In the **Data name** field, type the name `estimate`, and click **Insert**. This action adds a new data set to the Data Board in the System Identification Tool window.

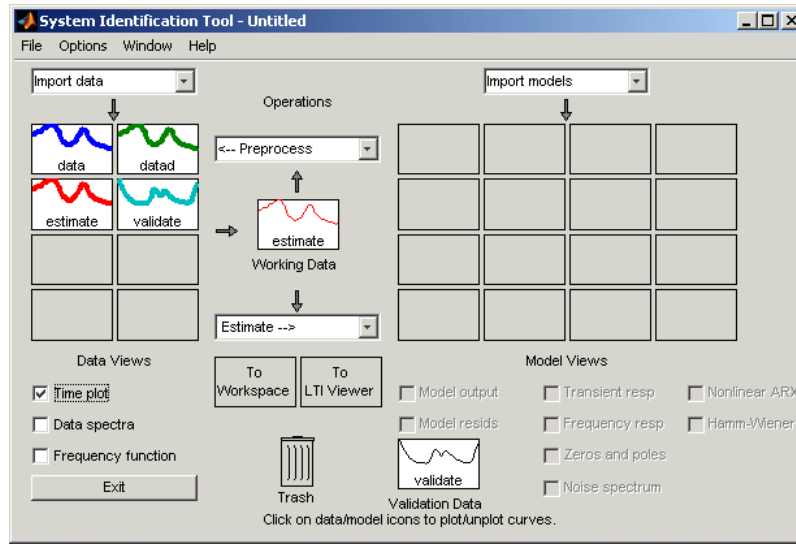
7 In the Select Range dialog box, edit the **Samples** field to select the last 500 samples, as follows:

```
501  1000
```



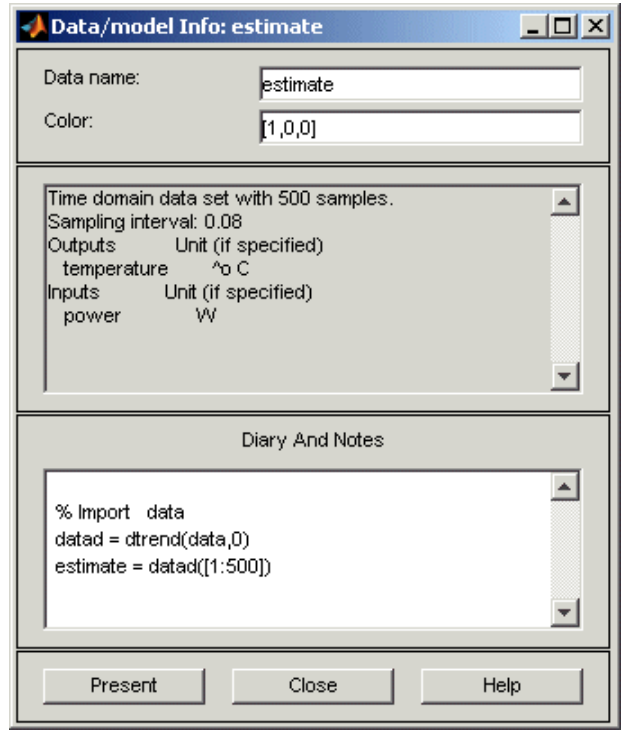
8 In the **Data name** field, type the name `validate`, and click **Insert**. This action adds a new data set to the Data Board.

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 window looks similar to the one shown in the following figure.



Note 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 do not need to split the data set you originally imported.

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



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

- Edit the name of the data set in the **Data name** field.
- Edit 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 is mixed into the overall color.
- In the noneditable area, view the total number of samples, the sampling interval, and the input and output channels 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 on these and other Toolbox commands, see the reference pages for each command.

As an alternative preprocessing shortcut, you can select **Preprocess > Quick start** to simultaneously perform the following four actions:

- Subtract the mean value from each channel.
- Split data into two halves.
- Specify the first half as estimation data for models (or **Working Data**).
- Specify the second half as **Validation Data**.

Removing Unused Data Sets and Saving the Session

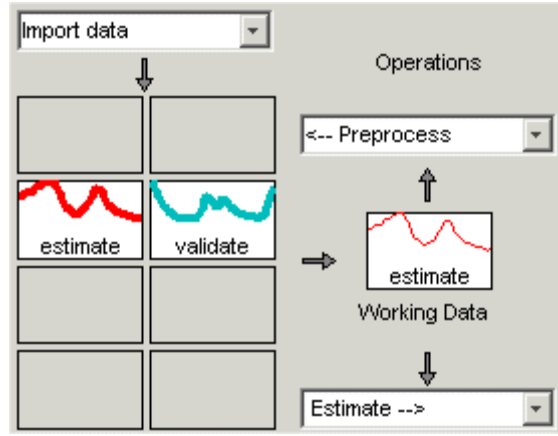
After you preprocess the data, as described in "Plotting and Preprocessing Data" on page 2-12, you may delete any data sets in the window that you do not need for estimation and validation, and save your session. You can open this session later and use it as a starting point for model estimation and validation without having to repeat 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 Data Board, and save the session for reuse.

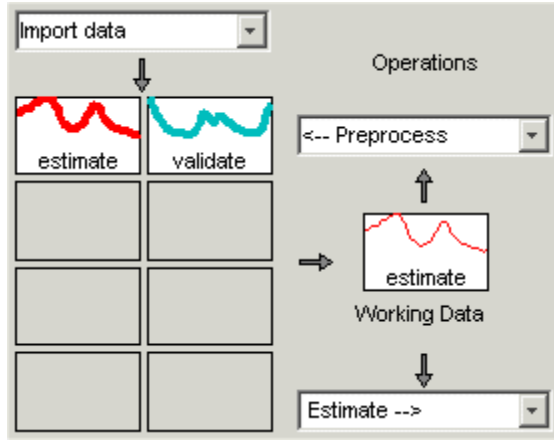
1. In the System Identification Tool window, drag and drop the `data` data set into the **Trash**.
2. Drag and drop the `datad` data set into the **Trash**.

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

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



3 Drag and drop the `estimate` and `validate` data sets to fill the empty rectangles at the top of the Data Board, 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, enter 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, enter the session name as an argument. For example:

```
ident('prep_data')
```

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

Estimating Preliminary Models

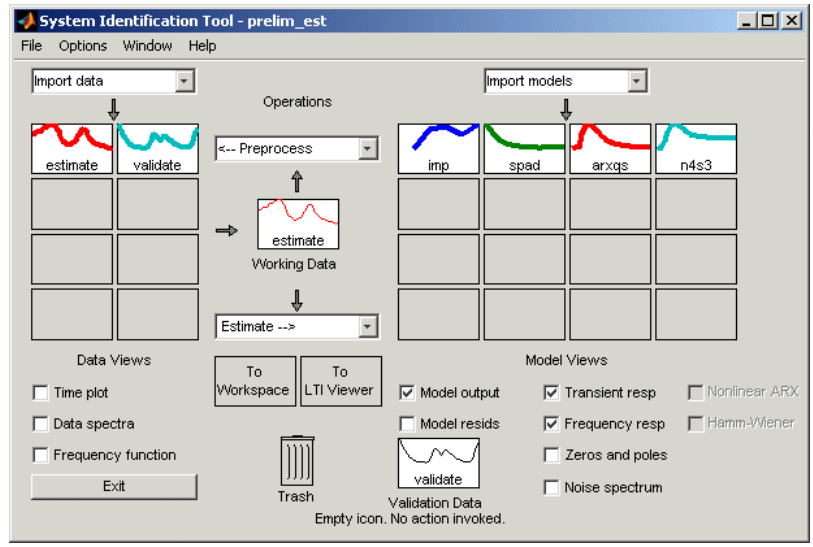
After preparing the data for estimation, as described in "Plotting and Preprocessing Data" on page 2-12, you can use the Quick Start feature of System Identification Toolbox to estimate and compare several types of models. Although these models might not be the final models you use to represent your system, you can use these models to assess whether linear modeling is sufficient. Preliminary models help gain insight into the possible delays and orders of the model that you can later refine.

This section discusses the following topics:

- "Using Quick Start to Estimate Preliminary Models" on page 2-23
- "Analyzing Preliminary Models" on page 2-25

Using Quick Start to Estimate Preliminary Models

In the System Identification Tool window, select **Estimate > Quick start** to estimate and plot the characteristics of four different types of models, as shown in the following figure.



Quick start always estimates the following four types of models and adds them to the Model Board with default names:

- `imp` — Step response by correlation analysis using `impulse`.

  This model is nonparametric (not expressed in terms of parameters) and computes the response for each time value, up to a maximum time determined by the algorithm.

- `spad` — Spectral analysis estimate of the frequency function using `spa`. The frequency function 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 ARX model calculated using `arx`.

  This model is parametric and has the following structure:

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

  Where *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. System Identification Toolbox estimates the parameters

  $a_1 \ldots a_n$ and $b_1 \ldots b_n$ using the input and output data from the estimation data set.

  In `arxqs`, n_a=n_b=4, and n_k is estimated from the step response model `imp`.

- `n4s3` — State-space model calculated using `n4sid`. The algorithm automatically selects the model order (in this case, 3).

  This model is parametric and has the following structure:

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

  Where *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. System Identification Toolbox estimates the state-space matrices A, B, C, D, and K.

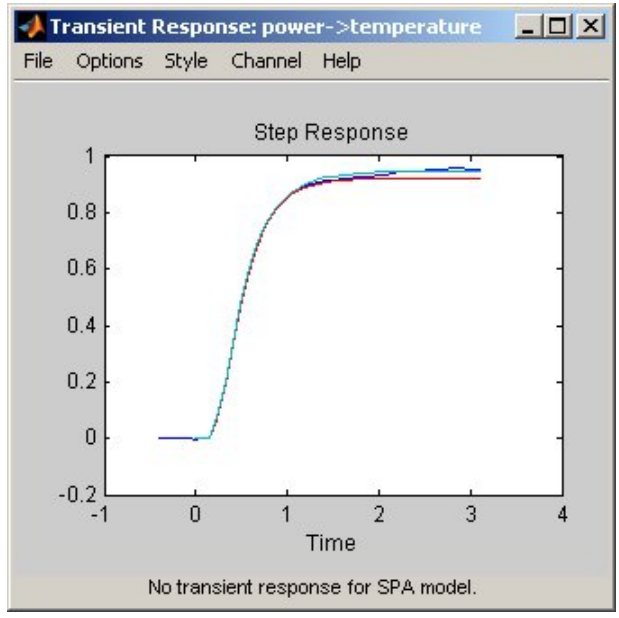
Analyzing Preliminary Models

Quick start also generates the following three plots of the preliminary models you created in "Using Quick Start to Estimate Preliminary Models" on page 2-23: step response, frequency response, and model output. To simulate or predict the model output, System Identification Toolbox uses the validation-data input as the input to the model, and plots the model output on top of the validation-data output. For more information about validating models, see "Validating Models" on page 1-36.

The step response and frequency response plots show agreement for the different models. Furthermore, 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.

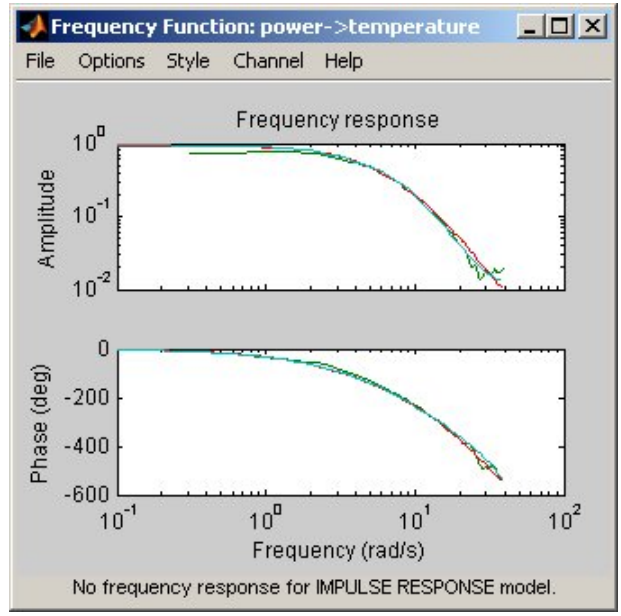
The step-response plot does not include the frequency-response model, `spad`, estimated using spectral analysis.

You can use impulse-response and step-response plots to estimate the dead time of linear systems. For example, the following step response plot shows a time delay of about 0.25 seconds before the system responds to the input. This response delay, or *dead time*, is approximately equal to about 3 samples because the system sampling interval is `0.08` s.



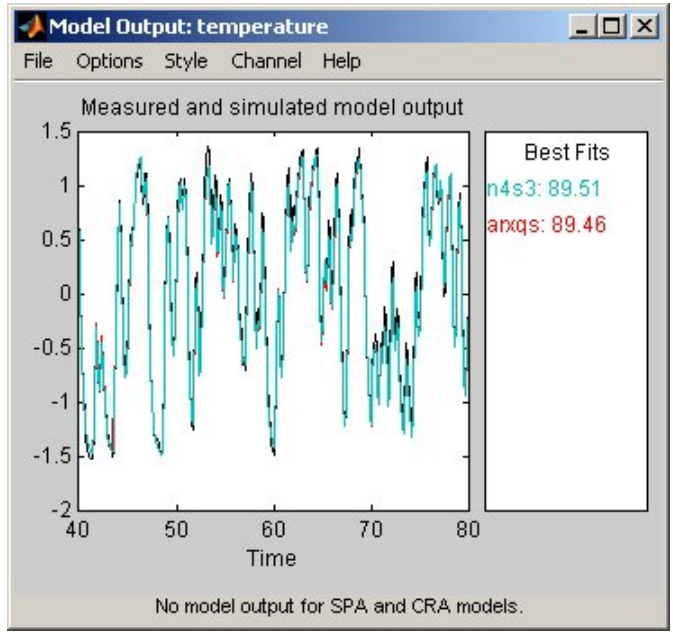
Step Response Plot for imp, arxqs, and n4s3

Also, notice that the frequency response plot does not include the impulse response model, `imp`, which is estimated using correlation analysis.



Frequency Response for Models spad, arxqs, and n4s3

By default, the Model Output plot compares the measured and the simulated output. The fit values for each model are summarized in the **Best Fits** area of the Model Output plot. The models in the **Best Fits** list are ordered from best at the top to worst at the bottom.



Measured Output and Model Output for arxqs and n4s3

Note If you want to compare predicted model output instead, you can select this option from the **Options** menu in the plot window.

Fine-Tuning Model Orders and Delays

In "Estimating Preliminary Models" on page 2-23, you estimated four different types of preliminary models to see if a linear model is sufficient in representing the dynamics of the Feedback Process Trainer. In this portion of the tutorial, you explore different model structures and orders to find the simplest model that adequately represents the dynamics.

System identification is typically an iterative process where you try different model structures and orders. You use System Identification Toolbox to quickly identify some initial orders and delays and then systematically explore these variations to find the best model for your application.

This tutorial focuses on linear parametric black-box ARX, ARMAX, and state-space model structures. A *black box* model has a structure that is not based on physical analysis, which is useful for modeling complex systems where you cannot mathematically account for some or all of the physical processes. *Parametric* means that the model structure is fit to the input-output data by adjusting coefficients, or model parameters. For an introduction to the model types this Toolbox supports, see "Estimating Dynamic Models" on page 1-15.

This section discusses the following tasks:

- "Estimating a Range of Best-Fit Orders" on page 2-29
- "Trying State-Space and ARMAX Structures" on page 2-34
- "Choosing the Best Model" on page 2-39

Estimating a Range of Best-Fit Orders

What are the reasonable model orders for your system? System Identification Toolbox lets you estimate multiple ARX models with different combinations of orders and delays and compare the performance of these models. You choose the model with the best performance and use it as an initial guess for further modeling.

This section describes the following topics:

- "About ARX Models" on page 2-30

• "Estimating Model Orders" on page 2-30

About ARX Models

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

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

y(t) represents the output at time *t*, *u(t)* represents the input at time *t*, n_a is the number of poles, n_b is the number of zeros plus 1, n_k is the number of samples before the input affects the system output, and *e(t)* is the white-noise disturbance. You must specify the model orders, and System Identification Toolbox estimates the parameters $a_1 \ldots a_n$ and $b_1 \ldots b_n$.

Estimating Model Orders

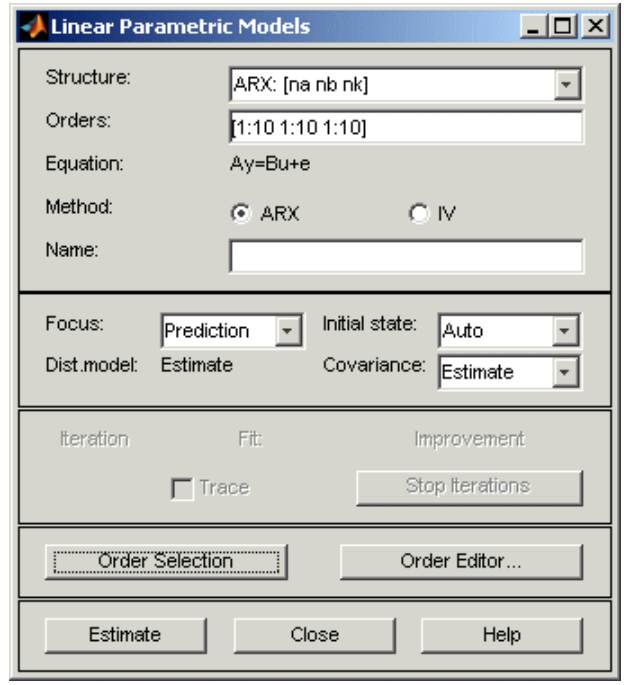
1 In the System Identification Tool window, select **Estimate > Linear parametric models** to open the Linear Parametric Models dialog box.

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

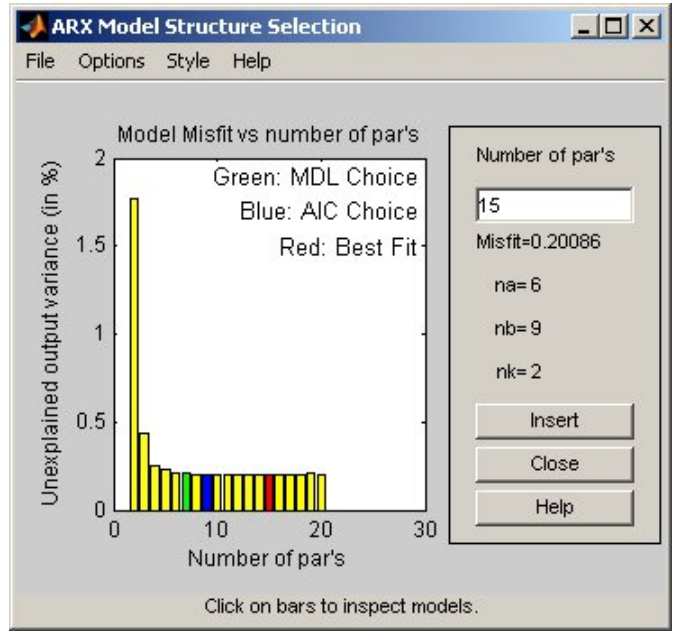
2 Edit the **Orders** field to specify that System Identification Toolbox try all combinations of poles, zeros, delays, where each value is between 1 and 10:

```
[1:10 1:10 1:10]
```

Tip Click **Order Selection** to automatically enter these values.



3 Click **Estimate** to open the ARX Model Structure Selection dialog box, 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 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. Thus, **Unexplained output variance (in %)** is the portion of the output not explained by the model.

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

- Green—Best fit minimizes Rissanen MDL criterion.
- Blue—Best fit minimizes Akaike AIC criterion.
- 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.

Click the red rectangle that represents overall best fit. The ARX Model Structure Selection dialog box shows that n_a=6, n_b=9, and n_k=2 (a delay of 2 samples) produces the best fit. This criterion corresponds to 15 parameters on the horizontal axis.

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

---

4 In the ARX Model Structure Selection dialog box, click any bar to view the orders that give the best fit corresponding to that sum of parameters.

For example, click the bar corresponding to 9 on the horizontal axis, which is green and represents the model orders that minimize the Rissanen MDL criterion. The area on the right is dynamically updated to show the orders and delays that give the best fit: n_a=5, n_b=4, and n_k=2.

5 In the ARX Model Structure Selection dialog box, 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.

System Identification Toolbox adds the model `arx692` to the Model Board in the System Identification Tool window and updates the plots to include the model's response.

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 2 samples.

6 In the ARX Model Structure Selection dialog box, 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 3 samples) into the Linear Parametric Models dialog box and performs the estimation.
- The model `arx223` is added to the Model Board in the System Identification Tool window and the plots are updated to include its response and output.

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

Trying State-Space and ARMAX Structures

By estimating 1000 ARX models with different combinations of orders, as described in "Estimating a Range of Best-Fit Orders" on page 2-29, you identified the number of poles, zeros and delays that provide a good starting point for systematically exploring different models. The ARX Model Structure Selection dialog box showed that the overall best fit for this system corresponds to a model with 6 poles, 9 zeros, and a delay of 2 samples. It also showed that a low-order model with n_a=2 (2 poles), n_b=2 (1 zero), and n_k=3 also provides a good fit. You can now use this information to explore other model structures.

This section discusses the following topics:

- "About State-Space and ARMAX Structures" on page 2-35
- "Estimating State-Space and ARMAX Models" on page 2-35

About State-Space and ARMAX Structures

In this section of the tutorial, you estimate a state-space model with the following general structure:

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

y(t) represents the output at time *t*, *u(t)* represents the input at time *t*, *x(t)* is the state values at time *t*, and *e(t)* is the white-noise disturbance. System Identification Toolbox estimates the state-space matrixes *A*, *B*, *C*, *D*, and *K*. To estimate a state-space model, you must specify a single integer as the model order. By default, the delay equals 1.

Next, you estimate ARMAX models with different orders. For a single-input and single-output system (SISO), the ARMAX model structure is:

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

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

To estimate an ARMAX model, you must specify the model orders, and System Identification Toolbox estimates the parameters $a_1 \ldots a_n$, $b_1 \ldots b_n$, and $c_1 \ldots c_n$.

Estimating State-Space and ARMAX Models

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

3 In the **Orders** field, enter `6` to create a sixth-order state-space model.

   This choice is based on the fact that the overall-best-fit ARX model has 6 poles.

4 Click **Estimate** to add a state-space model to the Model Board called `n4s6`.

5 In the **Structure** list, select `ARMAX: [na nb nc nk]` to estimate an ARMAX model.

6 In the **Orders** field, edit the orders *na*, *nb*, *nc*, and *nk* to these values:

```
2 2 2 2
```

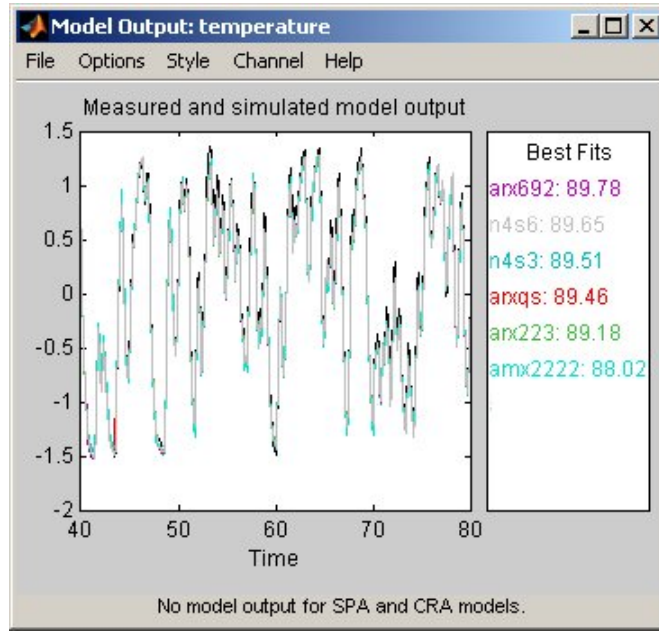
   The model name in the **Name** field is `amx2222`, by default.

7 Click **Estimate** to add the ARMAX model to the Model Board.

   Check the Model Output plot to verify that the new models are included.

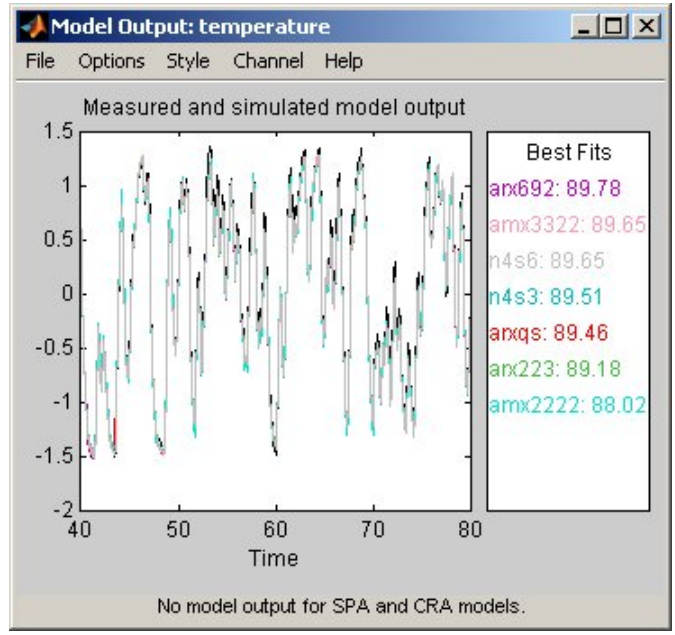
Tip If you closed the Model Output plot, you can generate it again by selecting the **Model output** check box in the System Identification Tool window. If the new model does not appear on the plot, click the model icon in the Model Board to make the model active.

The fit for `amx2222` is about 1% lower than the other models. Try a higher-order ARMAX model.



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

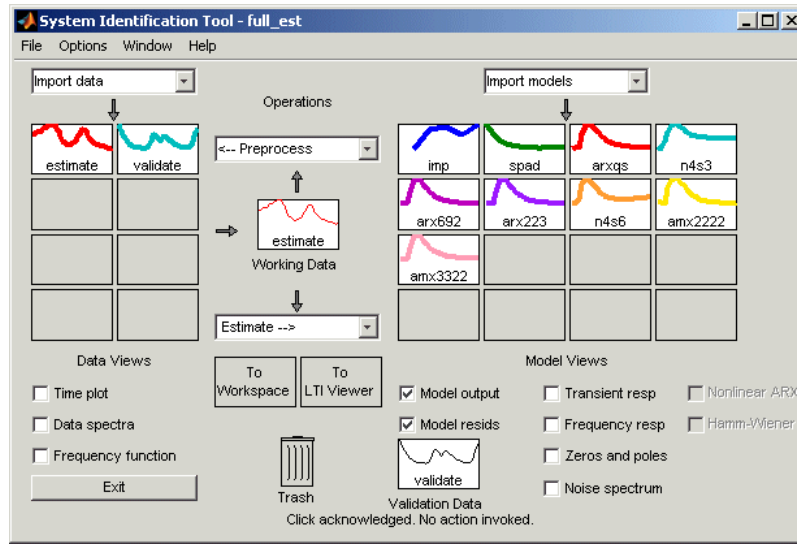
8 Repeat steps 6–7 using **Orders** `3 3 2 2`. These orders result in a model that fits the data almost as well as the higher order ARX model `arx692`.



Choosing the Best Model

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

In the previous steps of this tutorial, you generated several black-box parametric models. The following figure shows the System Identification Tool window, which include all of the models you estimated thus far.



This portion of the tutorial describes the following tasks:

- “Comparing Model Output” on page 2-39
- “Examining Model Residuals” on page 2-42
- “Getting the Model Parameters” on page 2-45

For more information about model validation, see “Validating Models” on page 1-36.

Comparing Model Output

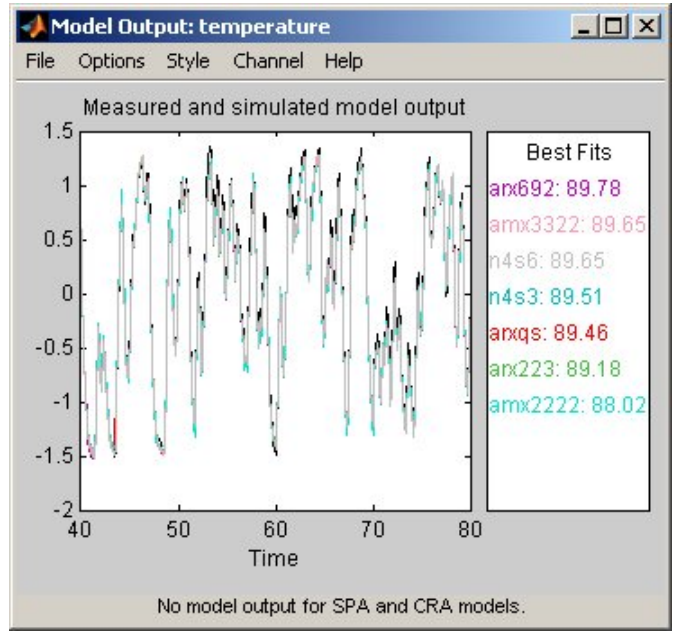
One way to test, or *validate*, a model is to check how well its simulated or predicted output matches the measured output. For more information about

this validation technique, see “Comparing Model Output and Measured Output” on page 1-36.

In System Identification Toolbox, you can quickly create a Model Output plot to compare the actual (measured) output and the model output. However, in this case, the Model Output plot should be already open and is automatically updated with the new models.

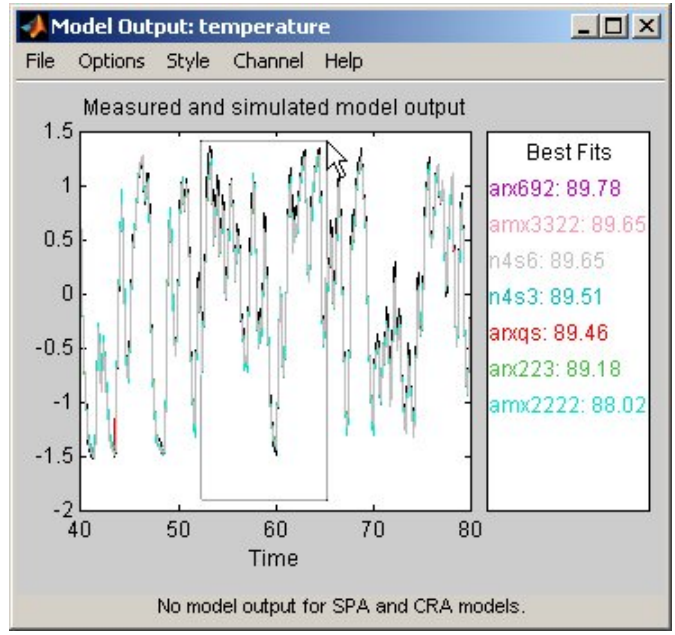
Tip If you closed the Model Output plot, you can generate it again by selecting the **Model output** check box in the System Identification Tool window. If the new model does not appear on the plot, click the model icon in the Model Board to include this model on plots.

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`.

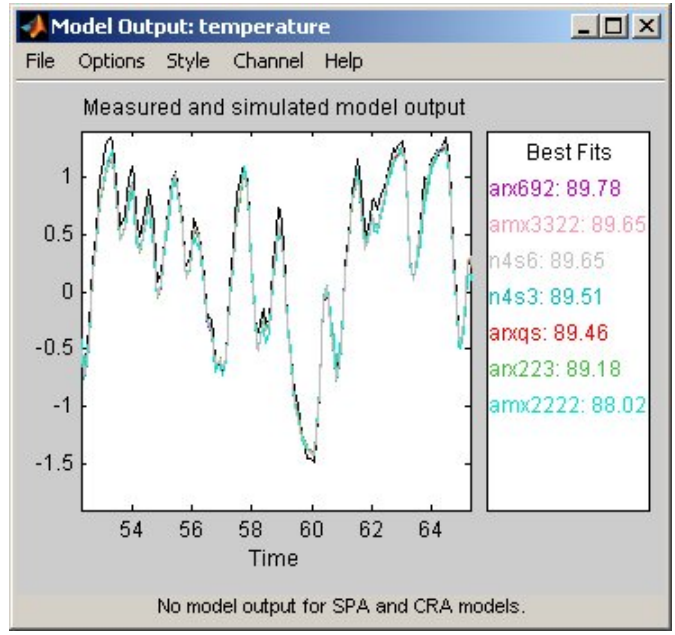


Note To validate your models using a different data set, you can drag and drop this data set into the **Validation Data** rectangle in the System Identification Tool window. This action automatically updates any open model views. If you change to validation data that is in frequency domain, rather than time domain, the Model Output plot updates to show 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 be in good agreement with the validation data.



Examining Model Residuals

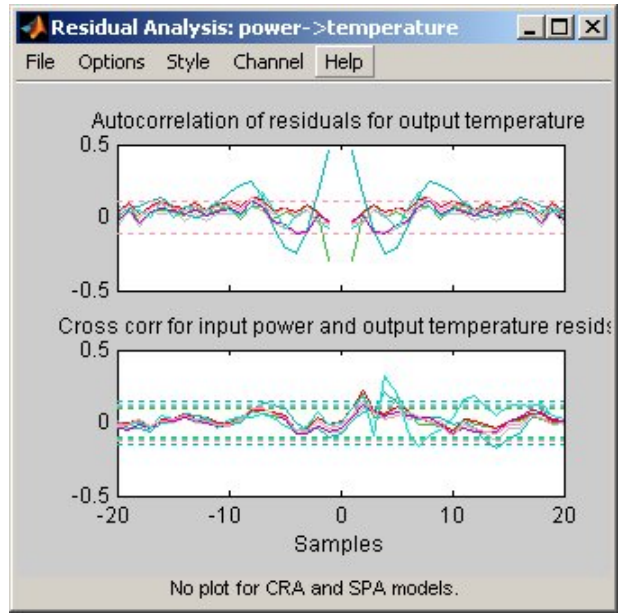
In addition to comparing model output to measured output, you can validate a model by checking the behavior of its residuals. For more information about residual analysis, see “Analyzing Residuals” on page 1-40.

To generate a Residual Analysis plot, select the **Model resids** check box in the System Identification Tool window. The Residual Analysis plot shows both the autocorrelation of residuals and the cross-correlation of residuals with the input. The horizontal scale is the number of lags, or the difference between the time steps that are correlated.

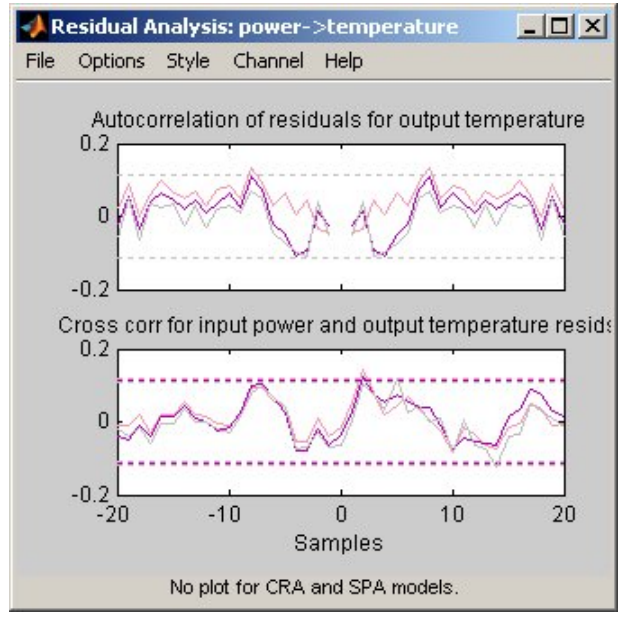
The top axes show the autocorrelation of residuals for the temperature output. The horizontal dashed lines on the plot represent the model confidence interval. Any fluctuations within the confidence interval are considered random. Two of the models, `n4s3` and `arx223`, produce residuals that enter

outside the confidence interval. It is desirable that the correlation function lie entirely between the confidence lines.

The bottom axes show the cross-correlation of the residuals with the input. The models `arxqs` and `amx2222` extend beyond the confidence interval and do not perform as well as the other models.



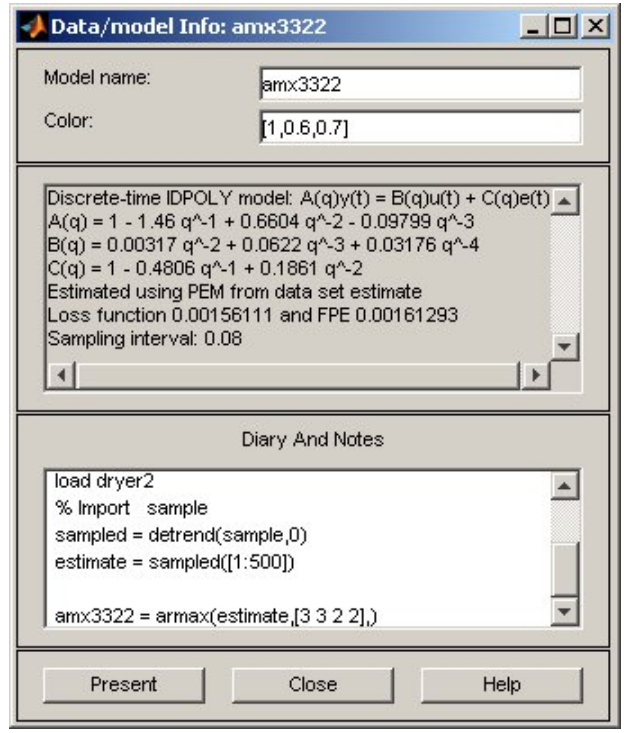
Remove models `n4s3`, `arx223`, `arxqs`, and `amx2222` from the Residual Analysis plot by clicking each model icon in the System Identification Tool window. The Residual Analysis plot now includes only the three models that pass the residual tests: `arx692`, `n4s6`, and `amx3322`.



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.

Getting the Model Parameters

You can view the numerical parameter values of the model `amx3322` by right-clicking the model icon in the Model Board. The Data/model Info dialog box opens.



The noneditable area of the dialog box lists the coefficients in the following format:

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

This notation implies that the ARMAX model structure is represented as:

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

This structure corresponds to this difference equation:

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

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.

According to the Data/model Info dialog box, the model parameters are:

```
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.

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

```
A(q) = 1 - 1.46(+-0.06003)q^-1
       + 0.6604(+-0.08906)q^-2
       - 0.09799(+-0.03519)q^-3
B(q) = 0.00317(+-0.001646)q^-2
       + 0.0622(+-0.002425)q^-3
       + 0.03176(+-0.005629)q^-4
C(q) = 1 - 0.4806(+-0.07558)q^-1
         + 0.1861(+-0.05597)q^-2
```

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

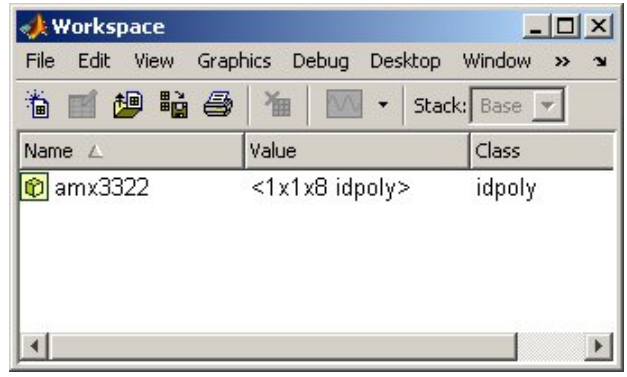
After you select a model, you might want to perform further analysis on this model using other MathWorks products. For example, if the model is a plant that requires a controller, you can import the model into the Control System Toolbox. To simulate your model in Simulink, perhaps as part of a larger dynamic system, you can import this model as a Simulink block.

To make your model available to Simulink and other Toolboxes, you must first export the model from the System Identification Tool to the MATLAB workspace.

Note The model you create in the System Identification Tool is not automatically available in MATLAB workspace. You must export it.

To export the `amx3322` model, drag it to the **To Workspace** rectangle in the System Identification Tool window.

The model appears in MATLAB workspace window.



This model is an object that belongs to the `idpoly` class.

After the model is in MATLAB workspace, you can perform other operation on the model. For example, if you have Control System Toolbox 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 installed on your computer, the **To LTI Viewer** rectangle displays in the System Identification Tool window.

The LTI Viewer is a graphical user interface for viewing and manipulating the response plots of linear models and displays the following plot types:

- Step and impulse responses
- Bode and Nyquist plots
- Nichols plots
- Singular values of the frequency response
- Pole/zero plots
- Response to a general input signal
- 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 window.

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

3

Estimating Continuous-Time Process Models Using the System Identification Tool

About This Tutorial (p. 3-3)	Overview of the tutorial and the sample data.
Before You Begin (p. 3-5)	How to load the sample MAT-file into MATLAB workspace and open the System Identification Tool GUI.
Importing Data into the System Identification Tool (p. 3-7)	How to import the `iddata` object from MATLAB workspace into the System Identification Tool.
Plotting and Preprocessing Data (p. 3-10)	How to create a time plot of the data, subtract the mean values of the input and the output, and split the data into two halves to use one half for model estimation, and the other half for model validation.
Estimating Second-Order Process Models with Complex Poles (p. 3-14)	Introduction to the Process Model dialog box and how to estimate a second-order process model with complex poles (underdamped modes).
Fine-Tuning the Process Model (p. 3-22)	Modifying the estimation algorithm to improve the model.

Getting the Model Parameters (p. 3-26)	How view estimated model parameters and history of operations on the model and the corresponding data.
Exporting the Model to the MATLAB Workspace (p. 3-28)	How to make the model available to operations in the MATLAB Command Window for further processing with this Toolbox or other MathWorks products.
Using Simulink with System Identification Toolbox (p. 3-29)	How to create and simulate a System Identification Toolbox model in Simulink.

About This Tutorial

This tutorial takes you through the steps of using the System Identification Tool graphical user interface (GUI) to estimate low-order, continuous-time process models. Such models are popular in many industries and apply to production environments and DC motors, for example.

Continuous-time process models estimate static gain, time delay before the system output responds to the input, and characteristic time constants associated with poles and zeros to describe the system dynamics. The parameters of such models have physical significance, unlike the parameters of the black-box models you explored in Chapter 2, “Estimating Linear Models Using the System Identification Tool”.

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 $Y(s)$:

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

In this tutorial, you estimate G using different process-model structures. The System Identification Tool lets you 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 model order you can specify in this Toolbox is 3, and the poles can be real or complex (underdamped modes).

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

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

The sample data you use in this tutorial is in `proc_data.mat`, which contains 200 samples of simulated single-input and 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/sec:

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

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. To learn more about frequency-domain data, see the chapter on representing data for system identification in the *System Identification Toolbox User's Guide*.

If you use Simulink, see the last section in this tutorial for information about bringing System Identification Toolbox models and data objects into the Simulink environment.

Before You Begin

Before you can perform the tasks described in this tutorial, you must do the following preparation:

- "Loading Data into the MATLAB Workspace" on page 3-5
- "Starting the System Identification Tool" on page 3-5

Loading Data into the MATLAB Workspace

Load sample data in `proc_data.mat` by typing the following command at the MATLAB prompt:

```
load proc_data
```

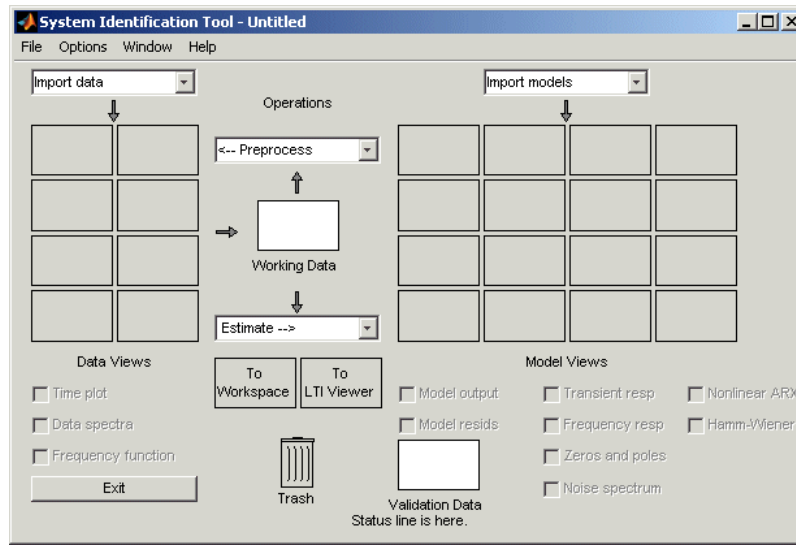
This command loads the data into MATLAB workspace as the `iddata` object `z`. For more information about `iddata` objects, see the section on creating `iddata` objects in the *System Identification Toolbox User's Guide*.

Starting the System Identification Tool

To open the System Identification Tool GUI, type the following at the MATLAB prompt:

```
ident
```

The default session name is `Untitled` and is displayed in the title bar, as shown in the following figure. For general information about this GUI, see the sections on using the System Identification Tool in the *System Identification Toolbox User's Guide*.



Importing Data into the System Identification Tool

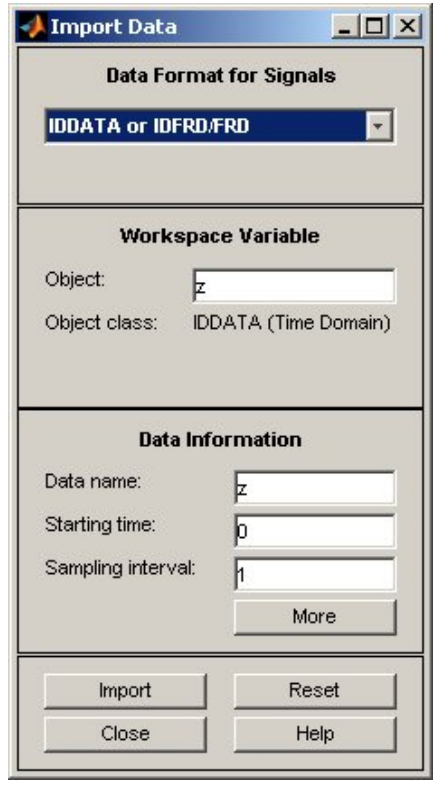
After opening the System Identification Tool window, as described in "Starting the System Identification Tool" on page 3-5, you import the data for this tutorial. This portion of the tutorial shows how to import single-input and single-output (SISO) data as an `iddata` object in `proc_data.mat`. The input and output signals each contain 200 data samples.

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

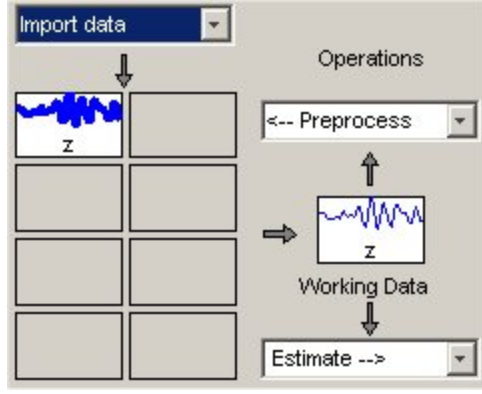
2 In the Import Data dialog box, specify the following options:

- **Object** — Enter `z` as the name of the MATLAB variable that is the time-domain data object. Press **Enter**.
- **Data name** — Use the default name `z`, which is the same as the name of the data object you are importing. This name labels the data in the System Identification Tool window 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 is the simulation sampling interval used to simulate the data.

  System Identification Toolbox uses the sampling interval during model estimation and to set the horizontal axis on time plots. If you transform a time-domain signal to a frequency-domain signal (see `fft`), the Fourier transforms are computed as Discrete Fourier Transforms (DFT) using this sampling interval.



3 Click **Import** to add the icon named z to the Data Board.



4 Click **Close** to close the Import Data dialog box.

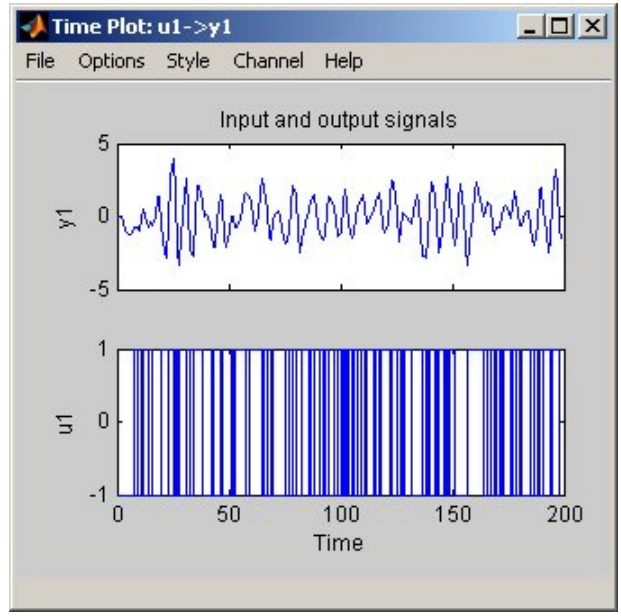
Plotting and Preprocessing Data

After importing data into the System Identification Tool, as described in "Importing Data into the System Identification Tool" on page 3-7, prepare the data for system identification.

This portion of the tutorial shows how to plot the data, subtract the mean values of the input and the output, and split the data into two halves. You use one half of the data for model estimation, and the other half of the data for model validation.

In most applications, you perform these common data preprocessing tasks. For information about other types of preprocessing, such as resampling and filtering the data, see the sections on plotting and preprocessing data in the *System Identification Toolbox User's Guide*.

1 In the System Identification Tool window, select the **Time plot** check box to display the time plot.



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

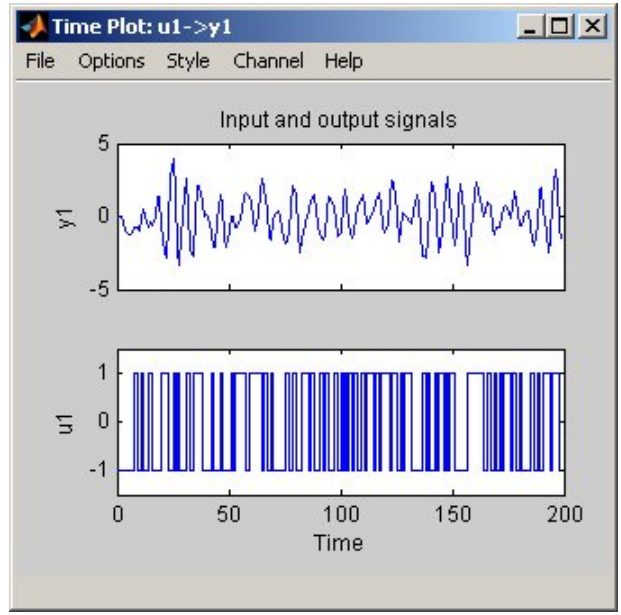
The next two steps demonstrate how to modify the axis limits in the plot.

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

3 In the Limits for Time Plot dialog box, set the new vertical axis limit of the input data channel **u1** to [-1.5 1.5]. Click **Apply** and then **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 Select **Preprocess > Quick start** to simultaneously perform the following four actions:

- Subtract the mean value from each channel.
- Split data into two halves.
- Specify the first half as estimation data for models (or **Working Data**).
- Specify the second half as **Validation Data**.

Estimating Second-Order Process Models with Complex Poles

After preparing the data for estimation, as described in "Plotting and Preprocessing Data" on page 3-10, you are ready to estimate process models.

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/sec:

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

This portion of the tutorial shows how to use the System Identification Tool GUI to create the following model structure and estimate its parameters:

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

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

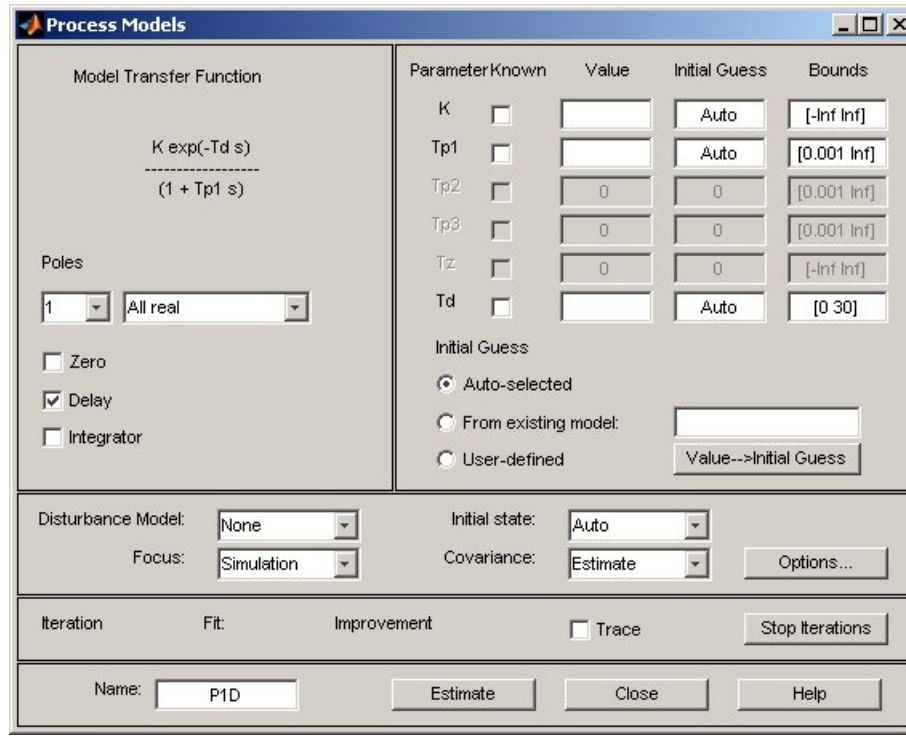
The remainder of this sections describes the following tasks:

- "Estimating an Initial Model" on page 3-14
- "Validating the Initial Model" on page 3-19

Estimating an Initial Model

The following procedure shows you how to create an initial model structure and estimate the parameters of this model. This model provides a good starting point for guiding the next step in the identification process.

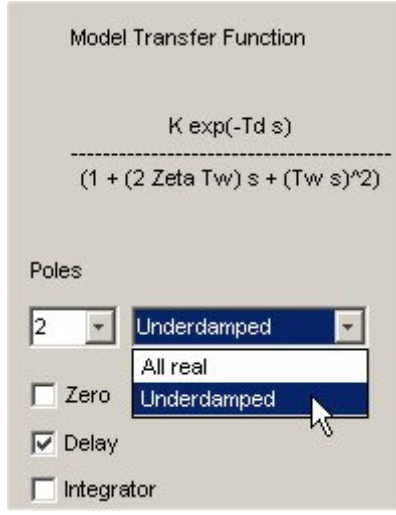
1 In the System Identification Tool window, select **Estimate > Process models** to open the Process Models dialog box.



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

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

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

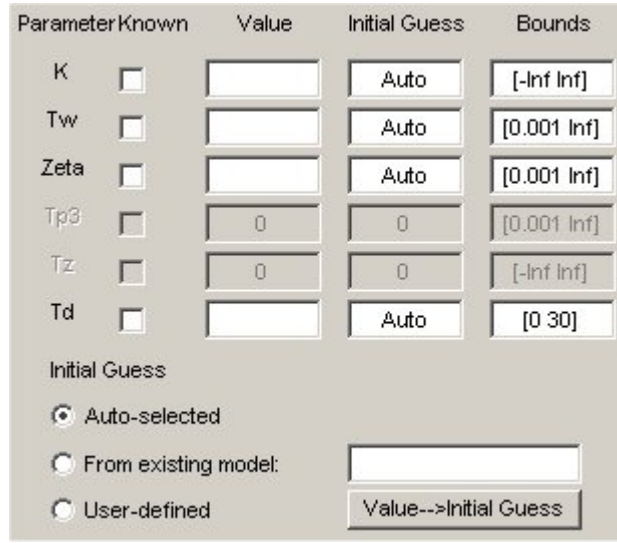


  The Parameter area now shows four active parameters: `K`, `Tw`, `Zeta`, and `Td`. By default, the model **Name** is set to the acronym `P2DU`, which indicates two poles (`P2`), the presence of a delay (`D`), and underdamped modes (`U`).

Note You can edit the model name. Choose a model name that is unique in the Model Board of the System Identification Tool window.

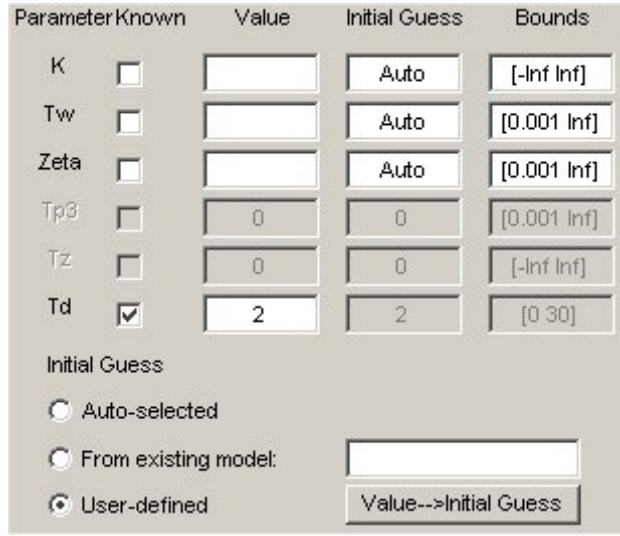
- Make sure that the **Zero** and **Integrator** check boxes are clear to exclude a zero and an integrator (self-regulating process) from the model.

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`.



4 (Optional) If you know a parameter value approximately, enter this value in the **Initial Guess** column of the Parameter table. The estimation algorithm uses this value as a starting point. If you know a parameter value exactly, enter this value in the **Initial Guess** column, and also select the corresponding **Known** check box in the table to fix its value.

For example, the following shows that the delay value `Td` is fixed at `2` s and is not estimated.



5 Keep the default **Bounds** values as the minimum and maximum values of each parameter.

When you know the range of possible values for a parameter, enter these values into the corresponding **Bounds** field to help the estimation algorithm. Otherwise, you should keep the default values.

6 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 `Simulation` is appropriate when your data has a high signal-to-noise ratio and 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, which are shown on plots as model confidence regions.

7 Click **Estimate**. This selection adds the model `P2DU` to the Model Board in the System Identification Tool window.

Validating the Initial Model

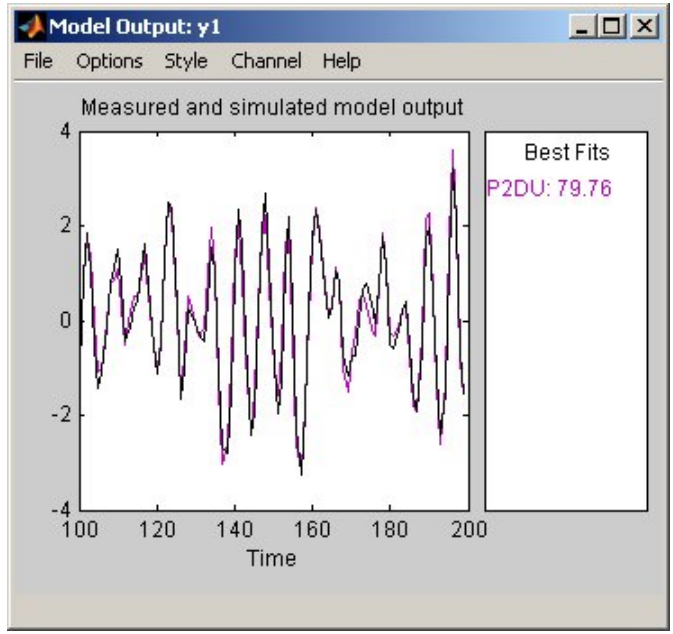
To examine the initial model you created in "Estimating an Initial Model" on page 3-14, you generate the following two plots using the validation data:

- Time plot comparing the model output and the measured output.
- Autocorrelation of the output residuals, and cross-correlation of the input and the output residuals.

Examining Model Output

One way to test, or *validate*, a model is to check how well its simulated or predicted output matches the measured output. For more information about this validation technique, see "Comparing Model Output and Measured Output" on page 1-36.

To generate the Model Output plot, select the **Model output** check box in the System Identification Tool window. To simulate the model output, System Identification Toolbox 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 is in good agreement 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.

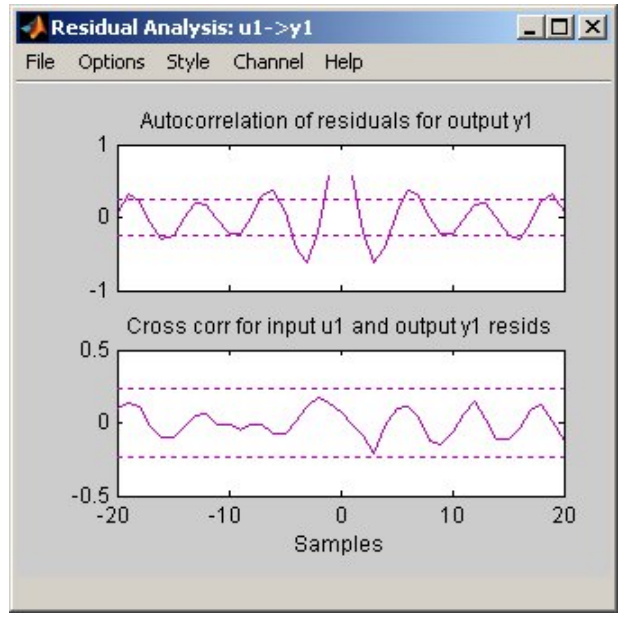
Examining Model Residuals

In addition to comparing model output to measured output, you can validate a model by checking the behavior of its residuals. For more information about residual analysis, see “Analyzing Residuals” on page 1-40.

System Identification Toolbox computes the model confidence interval when you set **Covariance** to `Estimate` in the Linear Parametric Models dialog box. A good model should have residual autocorrelation function within the confidence interval, indicating that the residuals are uncorrelated. Residuals must also be uncorrelated with past inputs. 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 the output *y(t)* that originates from the input *u(t-k)* is not properly described by the model.

To generate a Residual Analysis plot, select the **Model resids** check box in the System Identification Tool window. The Residual Analysis plot shows both the autocorrelation of residuals and the cross-correlation of residuals with the input.



The top axes show the autocorrelation of residuals for the temperature output (whiteness test). The horizontal dashed lines on the plot represent the model confidence interval. 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 random.

The residuals appear to be correlated, but there is no correlation between the residuals and the inputs. This result indicates that this process model is good, but that there might be a need for a noise model. Next, adjust the algorithm settings to improve the results.

Fine-Tuning the Process Model

In this portion of the tutorial, you modify the estimation algorithm settings to see if you can improve the model results. This section includes the following topics:

- "Estimating Models with Modified Settings" on page 3-22
- "Comparing Models" on page 3-23

Estimating Models with Modified Settings

Note The Process Models dialog box should still be open. If you closed it, repeat the process described in "Estimating an Initial Model" on page 3-14 before continuing.

---

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:

     $$y = Gu + He$$

     e is white noise.
   - **Name** — Edit the model name to `P2DUe1` to generate a model with a unique name in the Model Board.
2. Click **Estimate**.
3. In the Process Model 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 Model Board.

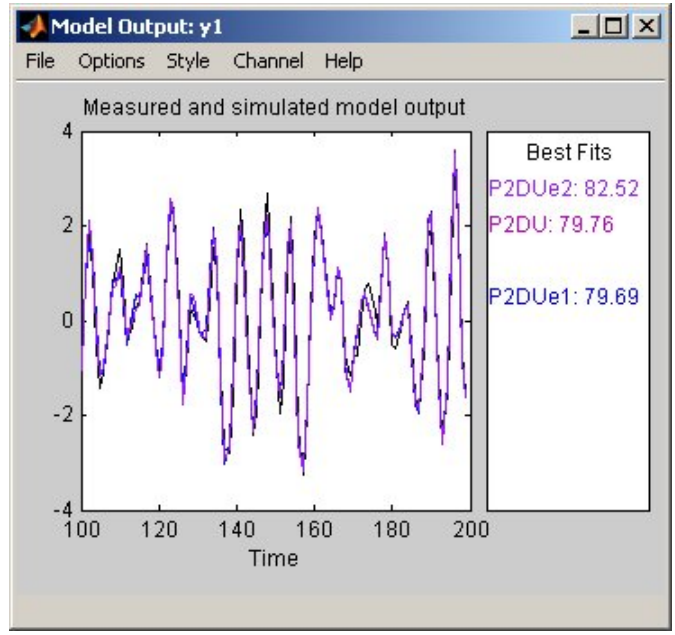
5 Click **Estimate**.

Comparing Models

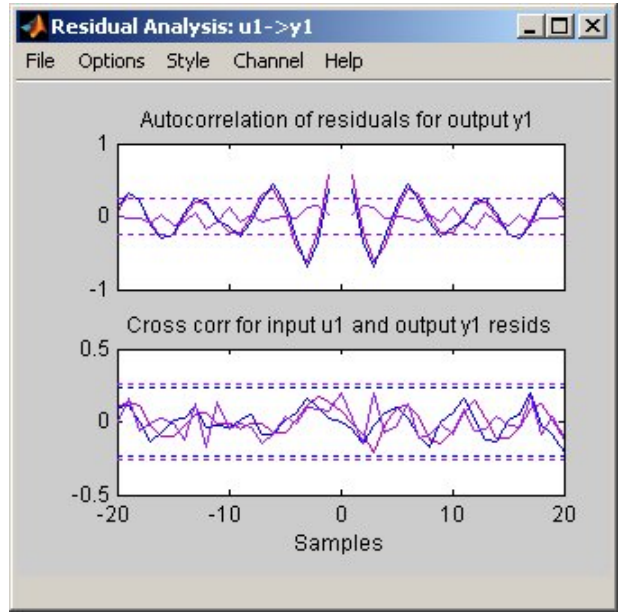
The Model Output and the Residual Analysis plots dynamically update to include the two new models. In this portion of the tutorial, you use these plots to compare the estimated models.

Note If you closed these plots, you can open them again by selecting the **Model output** and the **Model resids** check boxes in the System Identification Tools window.

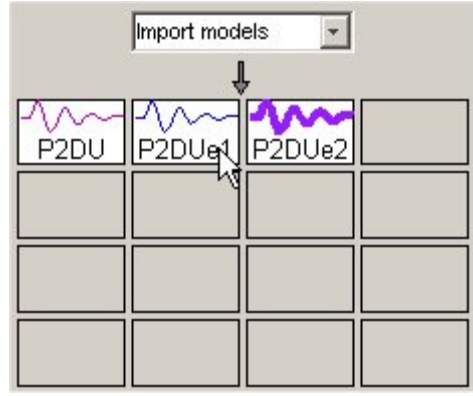
The following Model Output plot shows that the `P2DUe2` model has a slightly better performance than the other two models. However, all three models are in good agreement with the validation-data output.



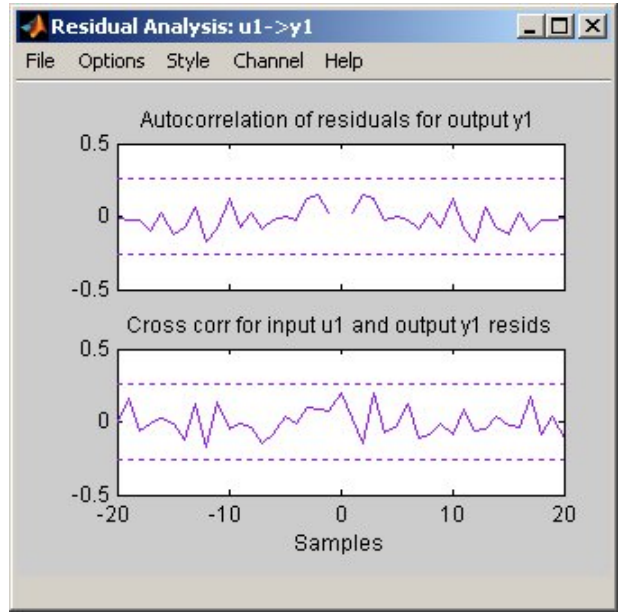
Futhermore, `P2DUe2` falls well within the confidence bounds on the Residual Analysis plot.



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 window.



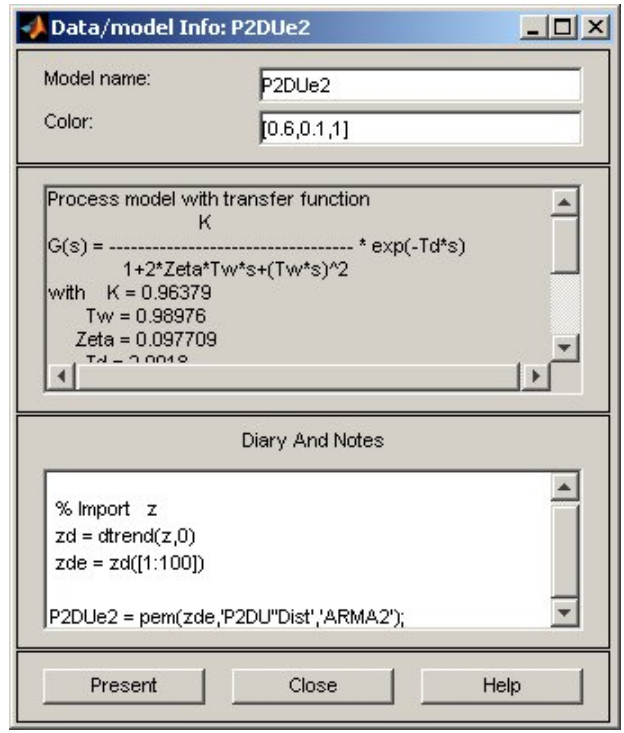
The Residual Analysis plot updates, as shown in the following figure.



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.

Getting the Model Parameters

You can view the numerical parameter values and other information about the model `P2DUe2` by right-clicking the model icon in the Model Board. The Data/model Info dialog box opens.



The noneditable area of the dialog box lists the model coefficients that correspond to the following model structure:

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

For the model `P2DUe2`, `K` is `0.96379`, `Tw` is `0.98976`, `Zeta` is `0.097709`, and `Td` is `2.0018`.

These coefficients are in good agreements 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:

```
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)
```

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

The Data/model Info dialog box does not show the uncertainties of the model parameters. 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. The uncertainty in the model parameters is computed to 1 standard deviation and appears after `+-` that is next to each parameter value:

```
K = 0.96379+-0.018245
Tw = 0.98976+-0.0055579
Zeta = 0.097709+-0.0064056
Td = 2.0018+-0.0025342
```

Exporting the Model to the MATLAB Workspace

After you select a model, you might want to perform further analysis on this model using other MathWorks products. For example, if the model is a plant that requires a controller, you can import the model into the Control System Toolbox. To simulate your model in Simulink, perhaps as part of a larger dynamic system, you can import this model as a Simulink block.

To make your model available to Simulink and other Toolboxes, you must first export the model from the System Identification Tool to the MATLAB workspace.

Note The model you create in the System Identification Tool is not automatically available in MATLAB workspace. You must export it.

To export the `P2DUe2` model, drag it to the **To Workspace** rectangle in the System Identification Tool window. The model now appears in MATLAB workspace window.

Note This model is an object that belongs to the `idproc` class.

Using Simulink with System Identification Toolbox

After you export a model from the System Identification Tool to the MATLAB workspace, as described in "Exporting the Model to the MATLAB Workspace" on page 3-28), you can bring this model into Simulink.

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

Note You must have completed the previous steps in this tutorial to proceed. Simulink must be installed to build the Simulink model.

This tutorial describes the following tasks:

- "Preparing Input Data in the MATLAB Workspace" on page 3-29
- "Building the Simulink Model" on page 3-30
- "Configuring Blocks and Simulation Parameters" on page 3-31
- "Running the Simulation" on page 3-35

Preparing Input Data in the MATLAB Workspace

In this tutorial, you use the input portion of the data object `z` from `proc_data.mat` as input to the model. You also use the process model `P2DUe2`. If you have completed all the steps in this tutorial, these variables are already in MATLAB workspace.

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

In the next step, you build the Simulink model.

Building the Simulink Model

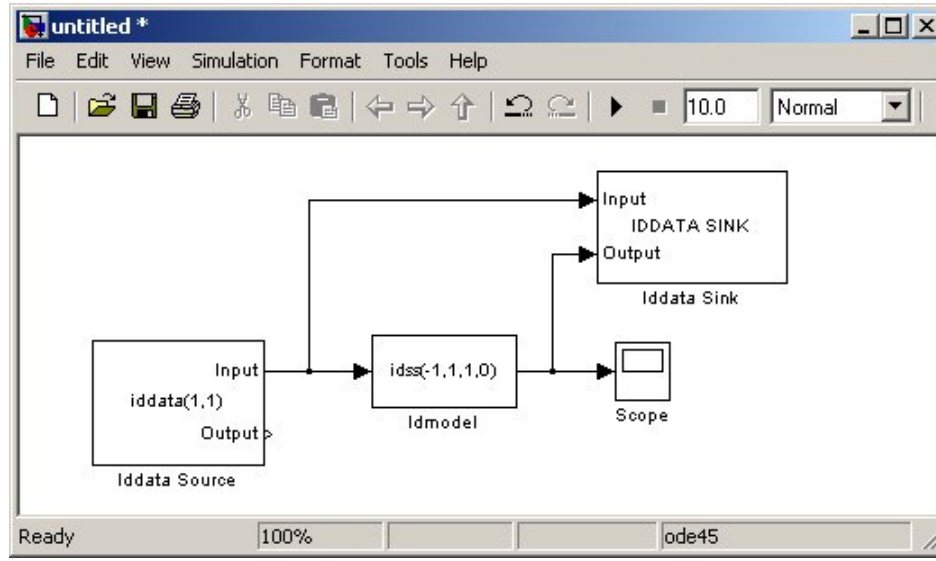
The following steps guide you through the process of adding blocks to a Simulink model. For more information about working with Simulink models, see the Simulink documentation.

1. In the MATLAB Command Window, type `simulink` at the MATLAB prompt.
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 System Identification Toolbox.

   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 Source block
   - Idmodel block
   - Iddata Sink 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 block to get information from MATLAB workspace and set the simulation time interval and duration.

Configuring Blocks and Simulation Parameters

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

- Get data from MATLAB workspace.
- Set the simulation parameters.

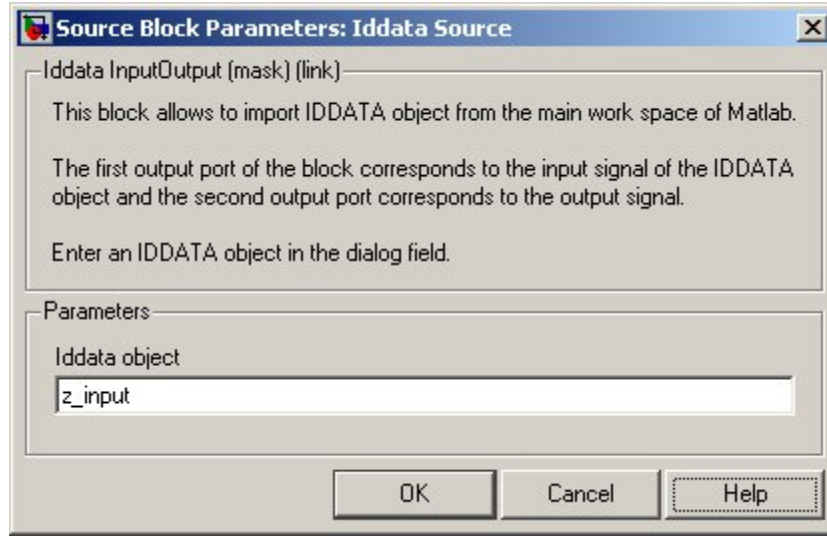
1 In the new model window, select **Simulation > Configuration Parameters**.

2 In the Configuration Parameters dialog box, enter 200 in the **Stop time** field. Click **OK**. This value sets the duration of the simulation to 200 seconds.

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

```
z_input
```

This variable represents the data object in MATLAB workspace that contains the input data. Click **OK**.

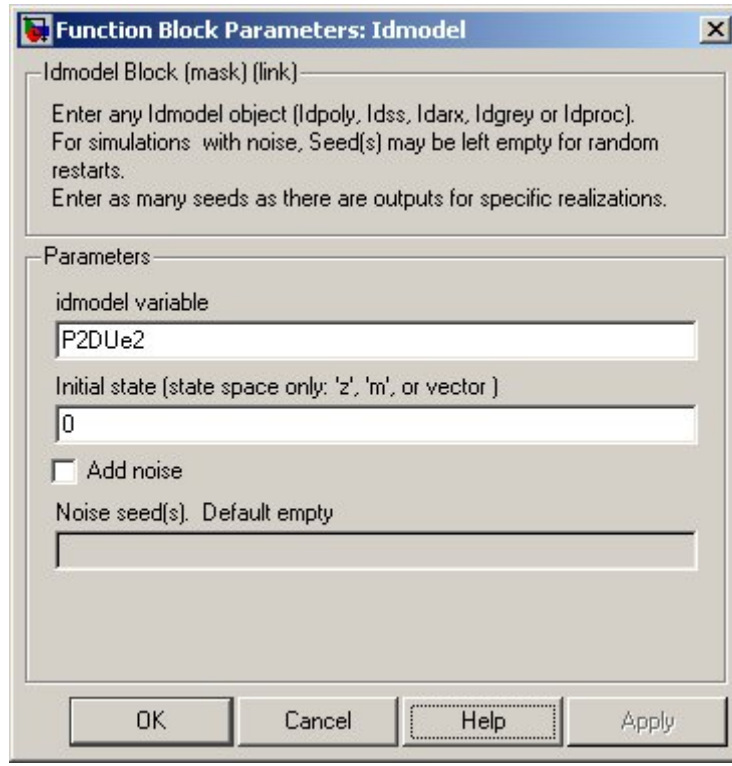


Tip As a shortcut, you can drag and drop the variable name from the MATLAB Workspace window to the **Iddata object** field.

4 Double-click the Idmodel block to open the Function Block Parameters: Idmodel dialog box. Then, type the following variable name in the **idmodel variable** field:

```
P2DUe2
```

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



5 Clear the **Add noise** check box to exclude noise from the system. Click **OK**.

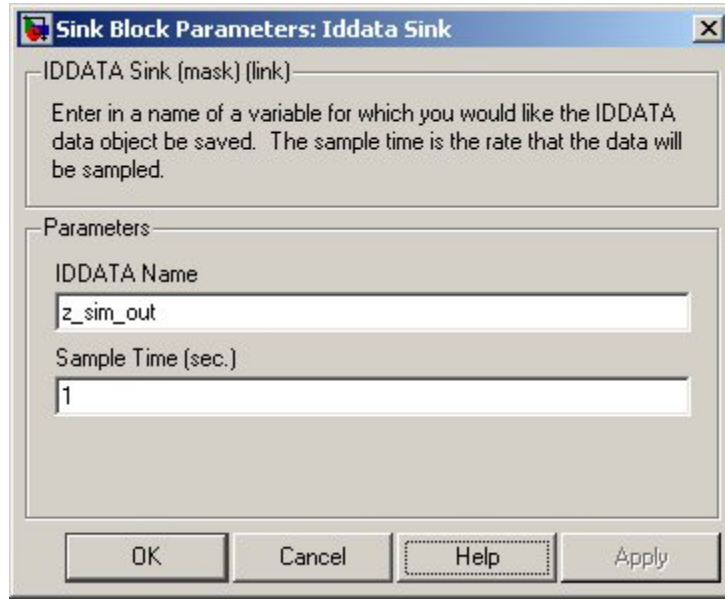
When **Add noise** is selected, Simulink derives the noise amplitude from the `NoiseVariance` property of the model and adds noise to the model input using `noisecnv`. The simulation propagates this noise according to the noise model H that you estimated with the system dynamics in System Identification Toolbox:

$$y = Gu + He$$

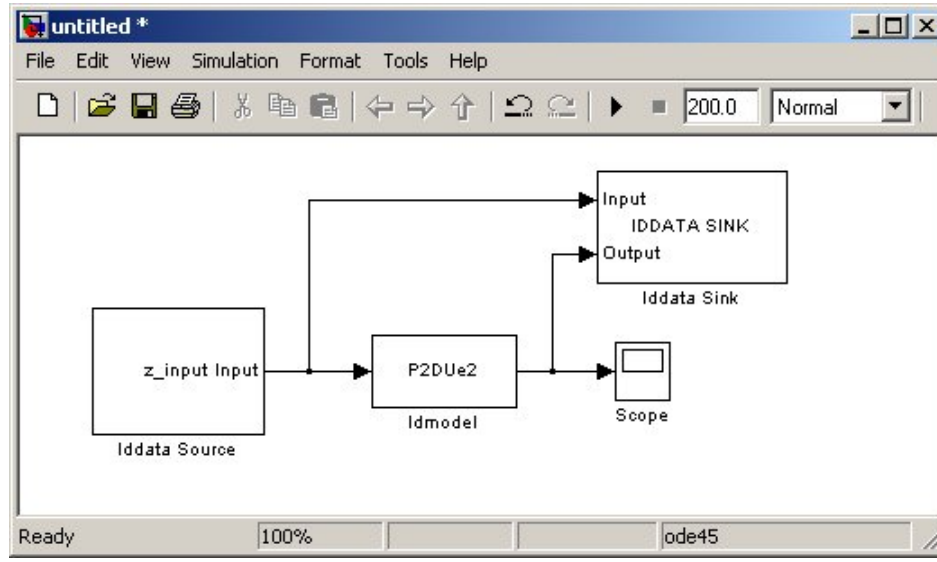
6 Double-click the Iddata Sink block to open the Sink Block Parameters: Iddata Sink dialog box. Then, type following variable name in the **IDDATA Name** field:

```
z_sim_out
```

7 Type `1` in the **Sample Time (sec.)** field to set the sampling time of the output data to be the same as the sampling time of the input data. Click **OK**.



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

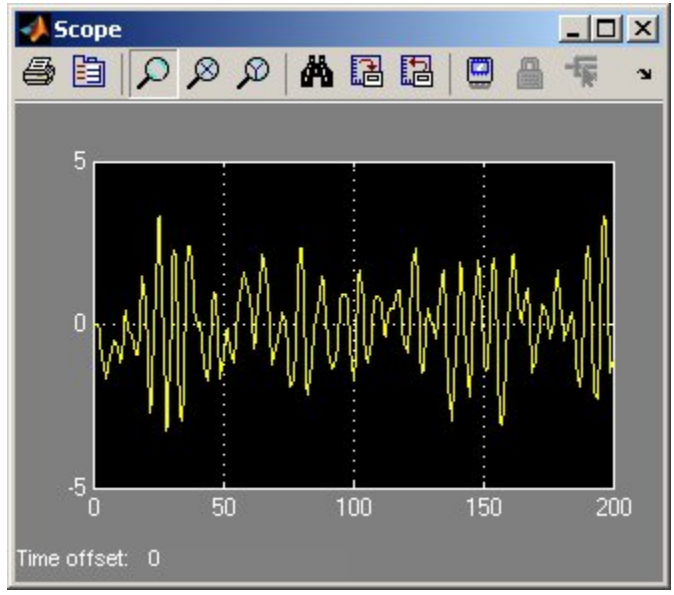


You are ready to run the simulation.

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 window, 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 is now available for further processing and exploration.

4

Estimating Linear Models in the MATLAB Command Window

About This Tutorial (p. 4-3)	Overview of the tutorial and the sample data.
Before You Begin (p. 4-5)	How to load the sample MAT-file into MATLAB workspace, plot the data, and remove equilibrium values from the data.
Representing Data for System Identification (p. 4-9)	How to create the data objects that let you manipulate the data and data properties using System Identification Toolbox functions and methods.
Estimating Nonparametric Models (p. 4-16)	How to estimate frequency-response and transient-response models to gain insight into the dynamic characteristics of the system.
Estimating Model Orders and Delays (p. 4-21)	How to estimate model orders and input-to-output delays using a simple ARX polynomial model.
Estimating Continuous-Time Process Models (p. 4-31)	How to estimate and validate a linear, continuous-time process model both with and without noise.

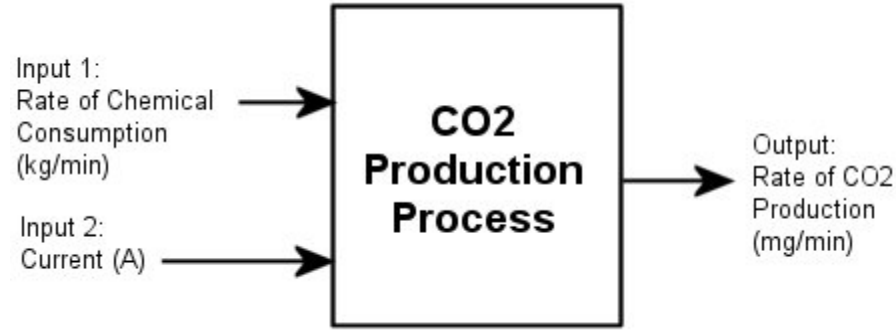
Estimating Black-Box Polynomial Models (p. 4-43)	How to estimate and validate ARX, state-space, and Box-Jenkins models.
Comparing Models (p. 4-52)	Using the `compare` function to plot the response of model output and measured output.
Using Models for Simulation and Prediction (p. 4-54)	Using `sim` and `predict` to simulate and predict model output, respectively.

About This Tutorial

By performing the steps in this tutorial, you get an overview of how to estimate linear models using System Identification Toolbox objects, functions, and methods in the MATLAB Command Window.

In this tutorial, you estimate linear black-box models to fit measured data for a hypothetical CO_2-production process. For definitions of a black-box model and related terminology, see "Definitions of Terms Describing Models" on page 1-25.

The sample data is provided in `co2data.mat`, which you install with the latest version of System Identification Toolbox. This MAT-file contains two-input and single-output (MISO) time-domain data from a steady-state process 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.



The workflow in this tutorial demonstrates system identification as an iterative process, which you begin by estimating a few simple models to get a range of possible model orders for your system. You use the order and delays of your preliminary models as a starting point for estimating other model structures. By validating the models as you go, you decide whether to adjust the orders and delays to fine-tune a specific model structure or try a different model structure.

Note This tutorial uses time-domain data to demonstrate how you can estimate linear models. This workflow also applies to frequency-domain data. To learn more about frequency-domain data, see the chapter on representing data for system identification in the *System Identification Toolbox User's Guide*.

Before You Begin

Before you can perform the tasks described in this tutorial, you must do the following preparation:

- "Loading Data into the MATLAB Workspace" on page 4-5
- "Plotting the Input and Output Data" on page 4-5
- "Removing Equilibrium Values from the Data" on page 4-7

Loading Data into the MATLAB Workspace

Load the sample data in `co2data.mat` by typing the following command at the MATLAB prompt:

```
load co2data
```

This command loads the following five variables into MATLAB workspace:

- `Input_exp1` and `Output_exp1` are the input and output data from the first experiment, respectively.
- `Input_exp2` and `Output_exp2` are the input and output data from the second experiment, respectively.
- `Time` is the time vector from 0 to 1000 minutes, increasing in equal increments of `0.5` min.

For both experiments, the input data consists of two columns of values: the first column is the rate of chemical consumption (in kilograms per minute), and the second column is the current (A). The output data is a single column of the rate of CO2 production (in milligrams per minute).

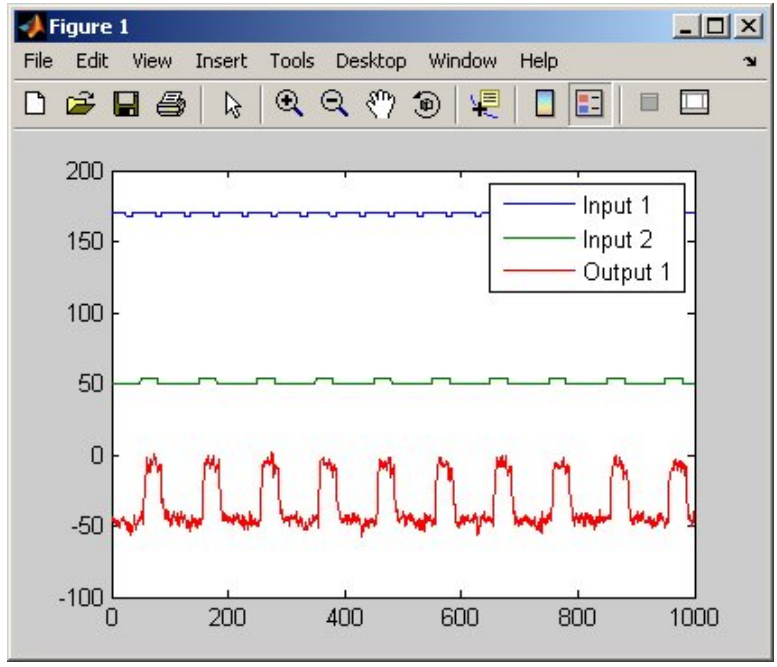
Plotting the Input and Output Data

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

```
plot(Time,Input_exp1,Time,Output_exp1)
legend('Input 1','Input 2','Output 1')
figure
plot(Time,Input_exp2,Time,Output_exp2)
```

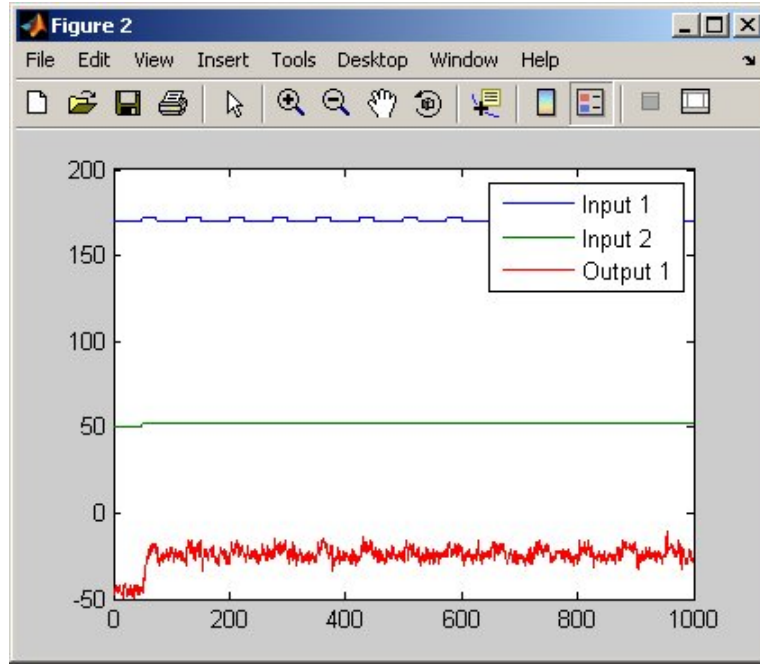
```
legend('Input 1','Input 2','Output 1')
```

In the first experiment, the operator applies a pulse wave to each input to perturb it from its steady-state equilibrium.



Input and Output Data from Experiment 1

In the second experiment, the operator applies a pulse wave to the first input and then applies a step signal to the second input.



Input and Output Data from Experiment 2

Removing Equilibrium Values from the Data

The data plots in the section “Plotting the Input and Output Data” on page 4-5 show that the inputs and the outputs have nonzero equilibrium values.

With steady-state data, you can, in most cases, estimate linear models for signals that were measured relative to zero. Thus, you can seek models around zero without modeling the absolute equilibrium levels in physical units.

Magnifying the plots for both Experiment 1 and Experiment 2 shows that the earliest moment the operator applies a disturbance to the inputs occurs after 25 minutes of steady-state conditions. This disturbance occurs after the first 50 samples.

Use the following commands to remove the equilibrium values from the inputs and the 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.

Representing Data for System Identification

After you load the sample data into MATLAB workspace, as described in "Before You Begin" on page 4-5, you bundle the measurement data and information about its characteristics into a single entity, called *data object*.

System Identification Toolbox data objects, including `iddata` and `idfrd`, encapsulate both data values and data properties. System Identification Toolbox commands let you 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.

Note You should work with two independent data sets: use one data set for model estimation and the other for model validation. When two independent data sets are not available, you can split one data set into two halves, assuming that each half contains enough information to adequately represent the system dynamics.

This section discusses the following tasks:

- "Creating iddata Objects" on page 4-9
- "Plotting the Data" on page 4-11

Creating iddata Objects

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

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

Use these commands to 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.

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 properties, use dot notation or the `set` command. For example, to assign channel names and units that label plot axes, type the following syntax at the MATLAB prompt:

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

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

Note Property names are not case sensitive.

For detailed information about `iddata` objects, see the sections on creating `iddata` objects in the *System Identification Toolbox User's Guide*.

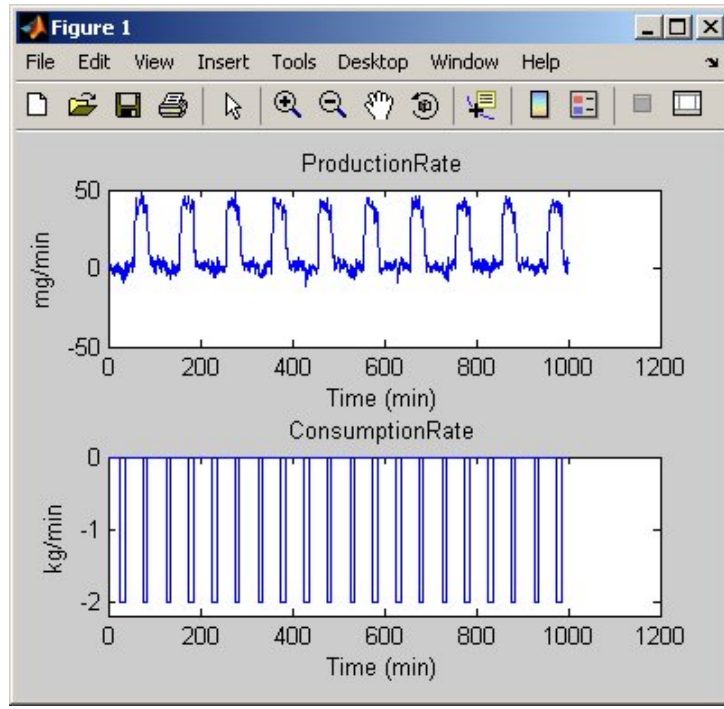
Plotting the Data

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

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

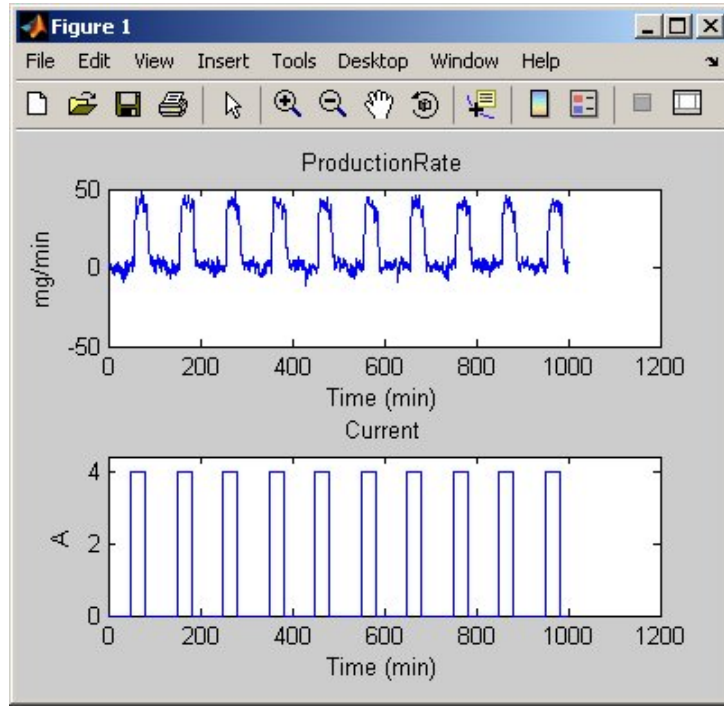
This opens the following plot. The bottom axes show the first input `ConsumptionRate`, and the top axes show the output `ProductionRate`.

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.



Input 1 and Output for ze

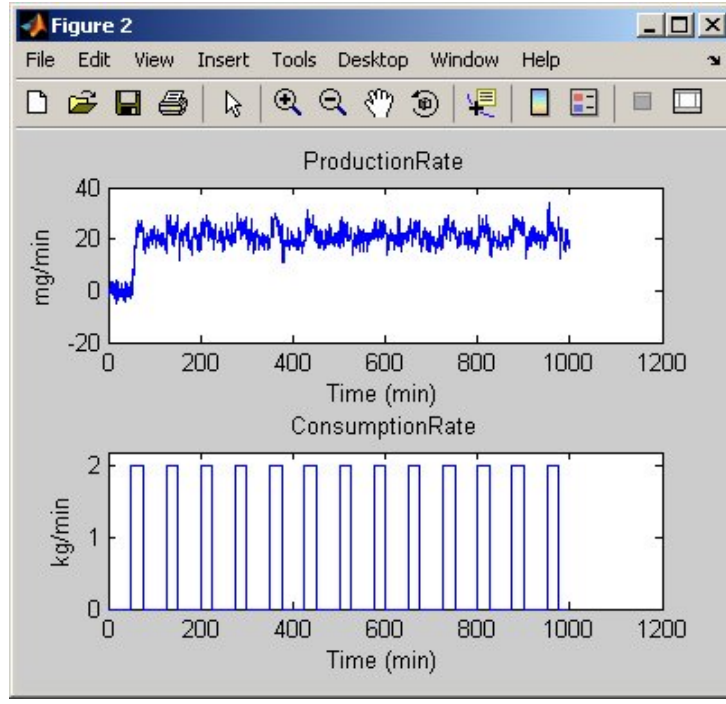
Tip When your data contains multiple inputs and outputs, press **Enter** to view the next input-output pair.



Input 2 and Output for ze

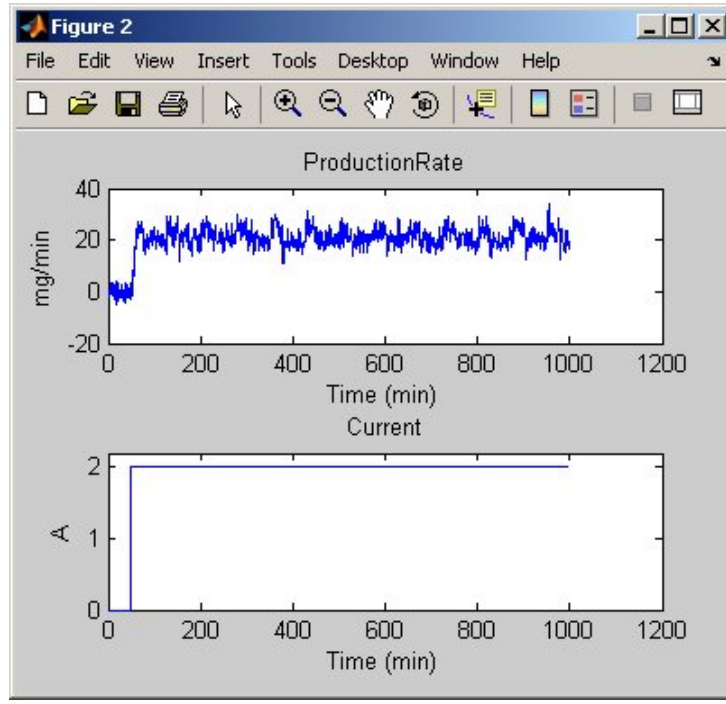
To plot the validation data in a new MATLAB Figure window, type the following commands at the MATLAB prompt:

```
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

Magnify the plots to see that the process amplifies the first input (`ConsumptionRate`) by a factor of 2, and amplifies the second input (`Current`) by a factor of 10.

Estimating Nonparametric Models

In this portion of the tutorial, you use the estimation data set `ze`, created in "Creating iddata Objects" on page 4-9, to estimate nonparametric models.

Nonparametric models include frequency-response and step-response models, which can help you gain insight into the dynamic characteristics of the system. These models are not represented by a compact mathematical formula with adjustable parameters. Instead, they consist of data tables. For an overview of nonparametric models, see "Linear Nonparametric Models" on page 1-25.

This section discusses the following tasks:

- "Estimating the Frequency Response" on page 4-16
- "Estimating the Step Response" on page 4-19

After you create the response plots, you find out the following information about the dynamics of the system:

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

Estimating the Frequency Response

To estimate the frequency response, use the `spa` command:

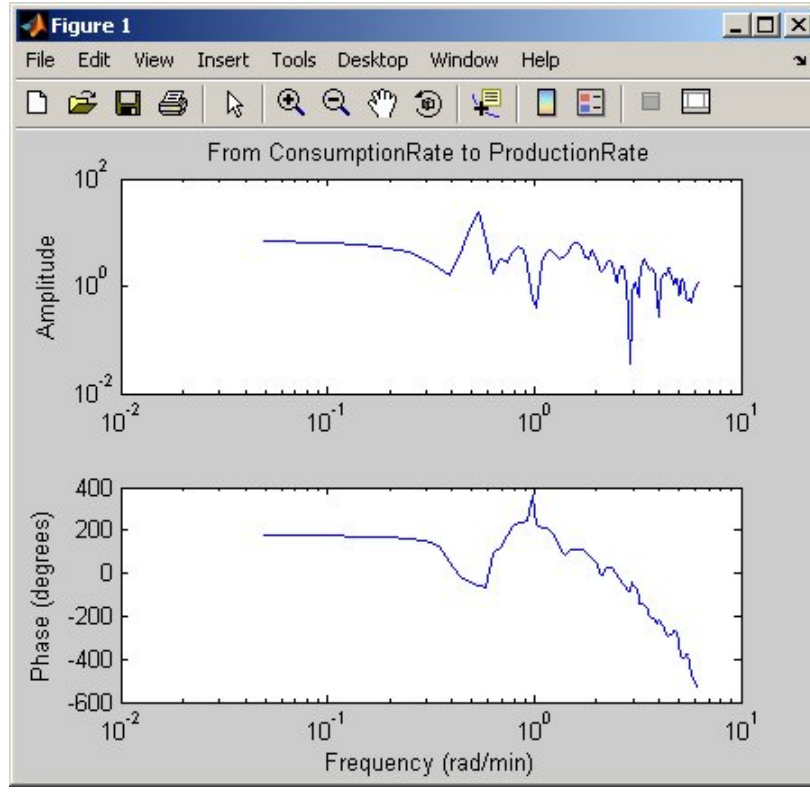
```
Ge=spa(ze);
```

Note System Identification Toolbox provides three functions for estimating the frequency response: `etfe` estimates the empirical transfer function using Fourier analysis, `spa` estimates the transfer function using spectral analysis for a fixed frequency resolution, and `spafdr` lets you specify a variable frequency resolution for estimating the frequency response.

To plot the frequency response on a Bode plot, type the following at the MATLAB prompt:

```
plot(Ge)
```

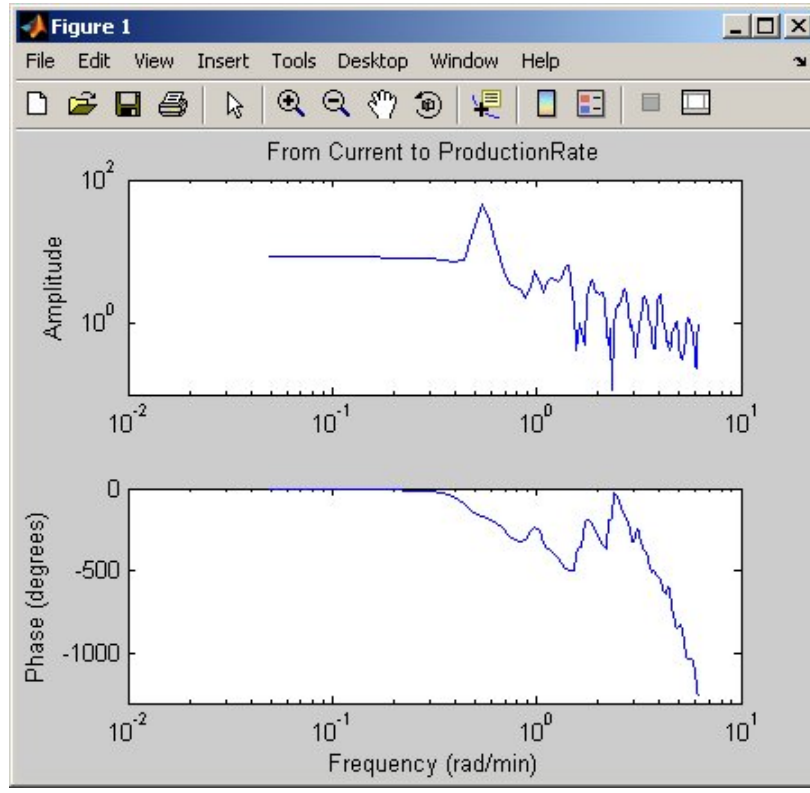
This command produces the following plot.



Frequency Response for the First Input-Output Path

The amplitude peaks at the frequency of about 0.7 rad/s, which suggests a resonance mode for the first input-to-output combination—`ConsumptionRate` to `ProductionRate`.

Only one input-output pair appears on the plot at a time. Thus, to view the second input `Current`, select the MATLAB Figure window, and press **Enter**. The input-output pair are 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-to-output combinations.

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

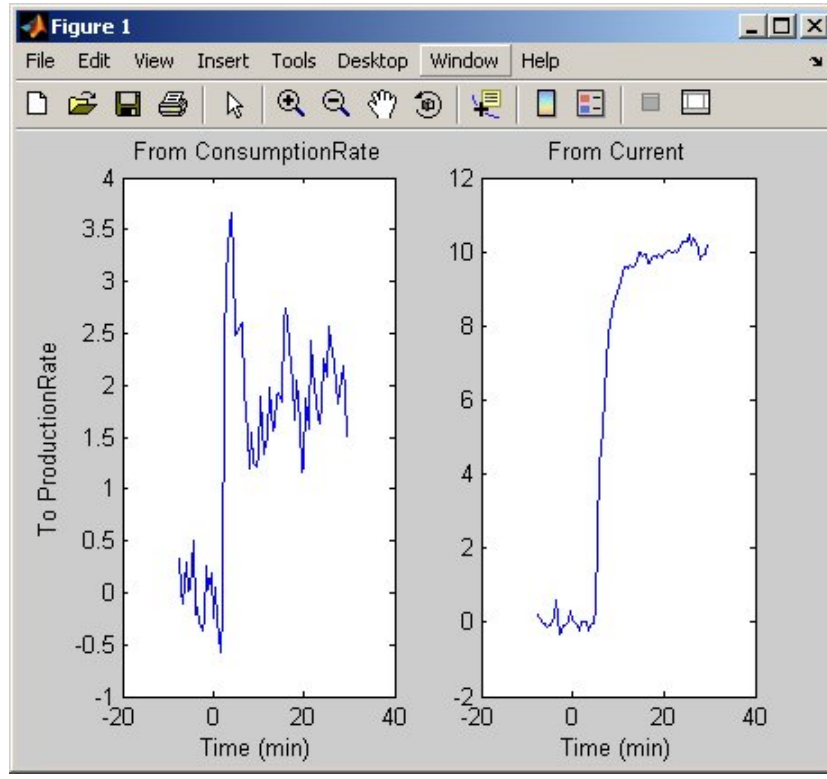
Estimating the Step Response

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-to-output combination suggests an overshoot, which indicates the presence of an underdamped mode (complex poles) in the physical process.

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

The plot indicates a nonzero delay in the system, which is consistent with the rapid phase roll-off you got in the Bode plot in “Estimating Nonparametric Models” on page 4-16.

Estimating Model Orders and Delays

In this portion of the tutorial, you estimate model orders and delays that are required inputs for estimating parametric models.

This section discusses the following tasks:

- "Estimating the Delay" on page 4-21
- "Estimating Model Orders Using a Simple ARX Structure" on page 4-23

Estimating the Delay

This section discusses the following topics:

- "Estimating Delay Using a Simple ARX Model" on page 4-21
- "Alternative Methods for Estimating Delay" on page 4-22

Estimating Delay Using a Simple ARX Model

In case of single-input systems, you can estimate the delay by using an impulse-response plot. However, in the case of multiinput systems, such as the one in this tutorial, you might not be able to tell which of the inputs caused the initial change in the output and you need a different approach.

The `delayest` function estimates the time delay in a dynamic system by estimating a low-order, discrete-time ARX model and treating the delay as an unknown parameter. The ARX model structure is one of the simplest black-box parametric structures, as described in "Linear Parametric Polynomial Models" on page 1-28.

In discrete-time, the ARX structure is a difference equation with the following form:

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

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

`delayest` assumes that n_a=n_b=2 and that there is no noise, and uses this information to estimate n_k.

To estimate the delay in this system, type the following at the MATLAB prompt:

```
delayest(ze)
```

MATLAB responds with the following:

```
ans =

     5    10
```

System Identification Toolbox gives two answers 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 `2.5`-min delay before the first input affects the output, and `5.0`-min delay before the second input affects the output.

Alternative Methods for Estimating Delay

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 off the time difference between the first change in the input and the first change in the output. This method is only practical for single-input and single-output

system; in the case of multiinput systems, you might not be able to tell which of the inputs caused the initial change in the output.

- Plot the impulse response of the data with a 1-standard-deviation confidence region. The time delay is equal to the first positive peak in the transient response magnitude that is greater than the confidence region for positive time values.

Estimating Model Orders Using a Simple ARX Structure

In this portion of the tutorial, you use `struc`, `arxstruc`, and `selstruc` to estimate a collection of ARX models with different combination of orders, and select the best orders based on the quality of the fit to the data. This approach provides a good initial guess of the model order that captures the system dynamics.

In this tutorial, there are two inputs to the system and one output. Therefore, you must estimate model orders for each input-output combination independently.

For each estimation, you use two independent data sets—the estimation data `ze` and the validation data `zv`.

You use the insights from nonparametric modeling to find the best model order. In "Estimating Nonparametric Models" on page 4-16, you saw that the first input-output pair might be characterized by a second-order response and that the second input-output pair might be characterized by a first-order response.

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.

Overview of the Process for Estimating Model Order

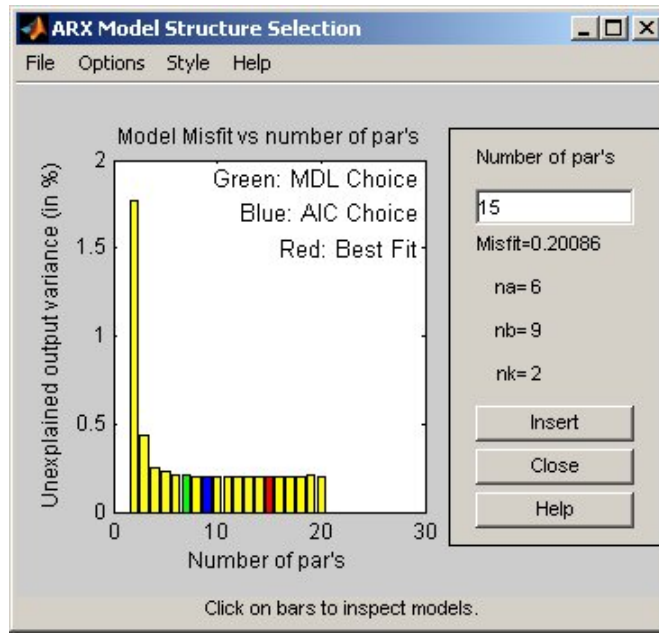
Model order is one or more integers that define the complexity of the model. In general, model order is related to the number of poles, the number of zeros, and the response delay (time in terms of the number of samples before the

output responds to the input). The specific meaning of model order depends on the model structure.

ARX model order corresponds to three integers: 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.

You use `struc`, `arxstruc`, and `selstruc` in combination to estimate model order. Each of these functions produces the following result:

- `struc` creates a matrix of possible 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 dialog box— shown in the following figure—to guide your choice of the model order with the best performance.



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

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

The vertical axis, called **Unexplained output variance (in %)**, is 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. In other words, **Unexplained output variance (in %)** is the portion of the output not explained by the model.

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

- Green minimizes Rissanen MDL criterion.
- Blue minimizes Akaike AIC criterion.
- Red minimizes the sum of the squares of the difference between the validation data output and the model output. This option is considered the overall best fit.

Model Order for the First Input-Output Combination

Nonparametric models in "Estimating the Frequency Response" on page 4-16 show that the response for the first input-output combination might be a second-order response. It makes sense to try the following order combinations, which are close to the estimated orders:

n_a=2:5

n_b=1:5

n_k=5

In "Estimating the Delay" on page 4-21, the delay for this input-output combination was estimated to be 5.

To estimate model order for the first input-output combination, use the following procedure.

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

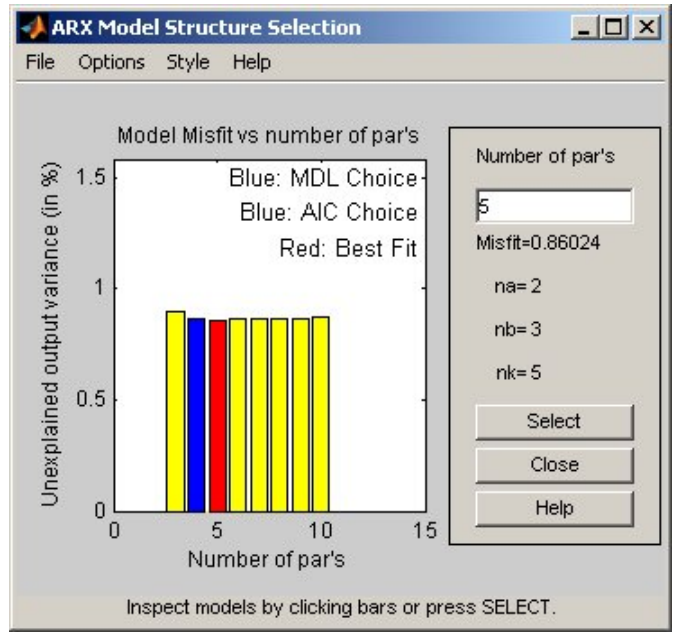
   ```
   NN1 = struc(2:5,1:5,5);
   ```

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 function opens the interactive ARX Model Structure Selection dialog box.



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.

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

Nonparametric models in the section "Estimating the Frequency Response" on page 4-16 show that the second input-to-output combination might have a first-order response. Therefore, it is reasonable to try the following range of orders:

n_a=1:3

n_b=1:3

n_k=10

In "Estimating the Delay" on page 4-21, the delay for this input-output combination was estimated to be 10.

To estimate model order for the second input-output combination, use the following procedure.

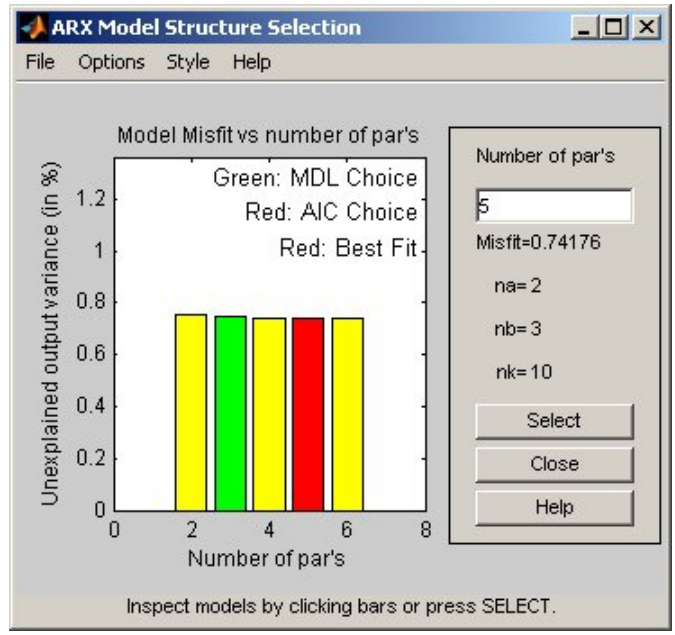
1 Use `struc` to create a matrix of possible model orders:

```
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 dialog box.



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 the rectangles is insignificant, Thus, all combinations of orders produce similar model performance. Therefore, you can choose the parameter combination that corresponds to the lowest model order.

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 Process Models

In this portion of the tutorial, you estimate a linear, continuous-time process model. System Identification Toolbox supports continuous-time process models with at most 3 poles (which might contain underdamped poles), one zero, a delay element, and an integrator.

To estimate this type of model, you use the information from "Estimating Model Orders and Delays" on page 4-21, where you estimated initial model orders and found the following results:

- For the first input-output combination: two poles (n_a=2), 1 zero (n_b=2), and a delay of 5 (n_k=5).
- For the second input-output combination: 1 pole (n_a=1), no zeros (n_b=1), and a delay of 10 (n_k=10).

This section discusses the following tasks:

- "Setting Up the Structure of the Process Model" on page 4-31
- "Estimating Model Parameters Using pem" on page 4-35
- "Validating the Process Models" on page 4-37
- "Estimating a Noise Model to Improve Results" on page 4-39

Before you begin, create a subset of 1000 samples from 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 section on subreferencing `iddata` objects in the "Subreferencing iddata Objects".

Setting Up the Structure of the Process Model

First, use the `idproc` constructor to create two process model structures:

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

The argument of `idproc` is a cell array that contains two strings, where each string specifies the structural elements of the corresponding model structure for each input-output combination:

- `'P2ZUD'` represents a transfer function with 2 poles (`P2`), 1 zero (`Z`), underdamped (complex-conjugate) poles (`U`) and a delay (`D`).
- `'P1D'` represents a transfer function with 1 pole (`P1`) and a delay (`D`).

To view the two resulting process-model structures, type the following command at the MATLAB prompt:

```
midproc0
```

MATLAB responds with the following output:

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

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


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


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

with   K = NaN
     Tp1 = NaN
      Td = NaN

This model was not estimated from data.
```

The parameters have `NaN` values because they are not yet estimated.

Based on the analysis in "Estimating the Delay" on page 4-21, you can set the delays to `2.5` min and `5` min for each of the two input-output combinations—as initial guesses. The following command sets the time delay property of the model object using dot notation:

```
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.

View all the properties of the `idproc` model object:

```
get(midproc0)
```

MATLAB responds with:

```
ans =
                Type: 'P2DZU,P1D'
                  Kp: [1x1 struct]
                 Tp1: [1x1 struct]
                 Tp2: [1x1 struct]
                 Tp3: [1x1 struct]
                  Tz: [1x1 struct]
                  Tw: [1x1 struct]
                Zeta: [1x1 struct]
                  Td: [1x1 struct]
         Integration: {'off'  'off'}
          InputLevel: [1x1 struct]
        InitialState: 'Auto'
    DisturbanceModel: {'None'  [1x0x0 idpoly]}
                  X0: [3x1 double]
                Name: ''
                  Ts: 0
           InputName: {2x1 cell}
           InputUnit: {2x1 cell}
          OutputName: {'y1'}
          OutputUnit: {''}
            TimeUnit: ''
     ParameterVector: [8x1 double]
               PName: {8x1 cell}
    CovarianceMatrix: []
       NoiseVariance: 1
          InputDelay: [2x1 double]
           Algorithm: [1x1 struct]
      EstimationInfo: [1x1 struct]
               Notes: {}
            UserData: []
```

For information about each parameter, see the `idproc` reference pages.

Estimating Model Parameters Using pem

After defining the process model structure, as described in "Setting Up the Structure of the Process Model" on page 4-31, you can use the `pem` (*prediction-error minimization*) method to estimate the parameters.

`pem` is an *iterative* estimation method, which mean that it uses an iterative nonlinear least-squares algorithm to minimize a cost function. The *cost function* is a function of the weighted sum of the squares of the errors.

Depending on its arguments, `pem` estimates different black-box polynomial models. You can use `pem` to estimate parameters for linear continuous-time process, state-space, ARX, ARMAX, Box-Jenkins, and Output-Error model structures.

To use `pem`, you must provide a model structure with unknown parameters and the estimation data as input arguments. In this case, use `midproc0` as the model structure and `Ze1` as the estimation data:

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

MATLAB responds with the following estimated parameters:

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

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

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


           K
G_2(s) = ---------- * exp(-Td*s)
          1+Tp1*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 process models let you estimate the delays. In this case, the estimated delay values `2.4739` and `4.8864` are different from the initial values `2.5` and `5`, respectively.

Tip When you know any of the parameter values, you can fix these parameters and only estimate the ones you do not know. For example, to fix the delay, you can assign a value to `Td` and use this command to avoid estimating it:

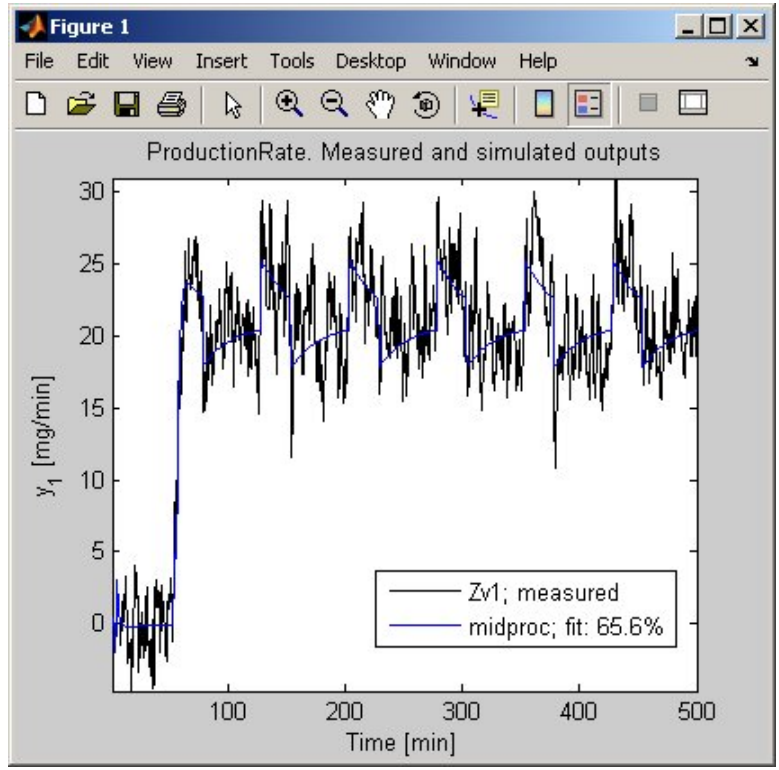
```
midproc.Td.status='fixed'
```

Validating the Process Models

One way to test, or *validate*, a model is to determine how well its simulated or predicted output matches the measured output. For more information about this validation technique, see “Comparing Model Output and Measured Output” on page 1-36.

In this section, you create a plot that compares the actual output and the model output using the `compare` function:

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



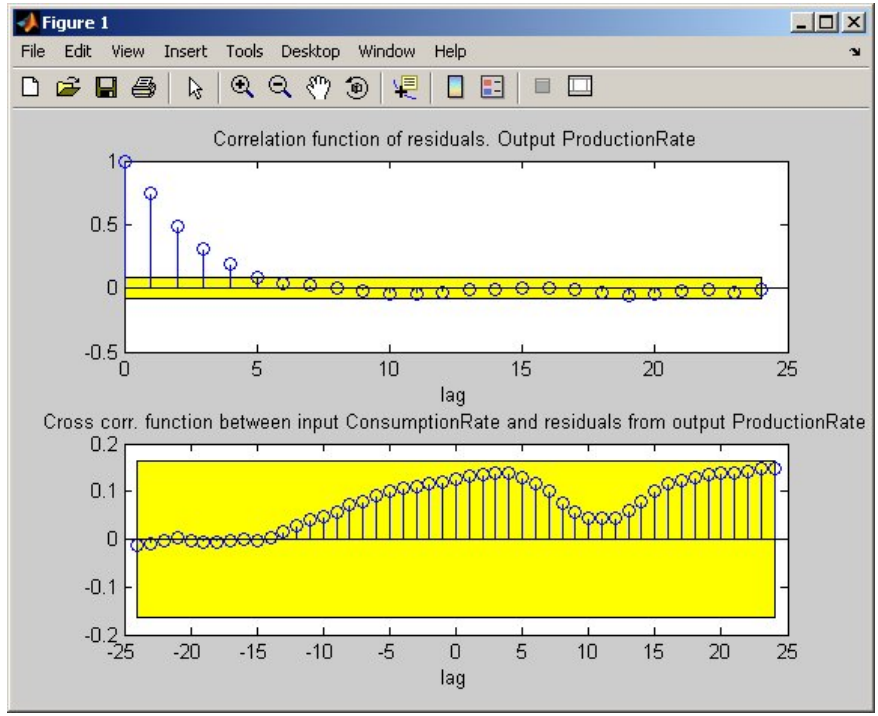
The preceding plot shows that the model output is in reasonable agreement with the measured output: there is an agreement of 65.6% percent between the model and the validation data.

You can also test a model by checking the behavior of its residuals. For more information about residual analysis, see “Analyzing Residuals” on page 1-40.

Use `resid` to perform residual analysis:

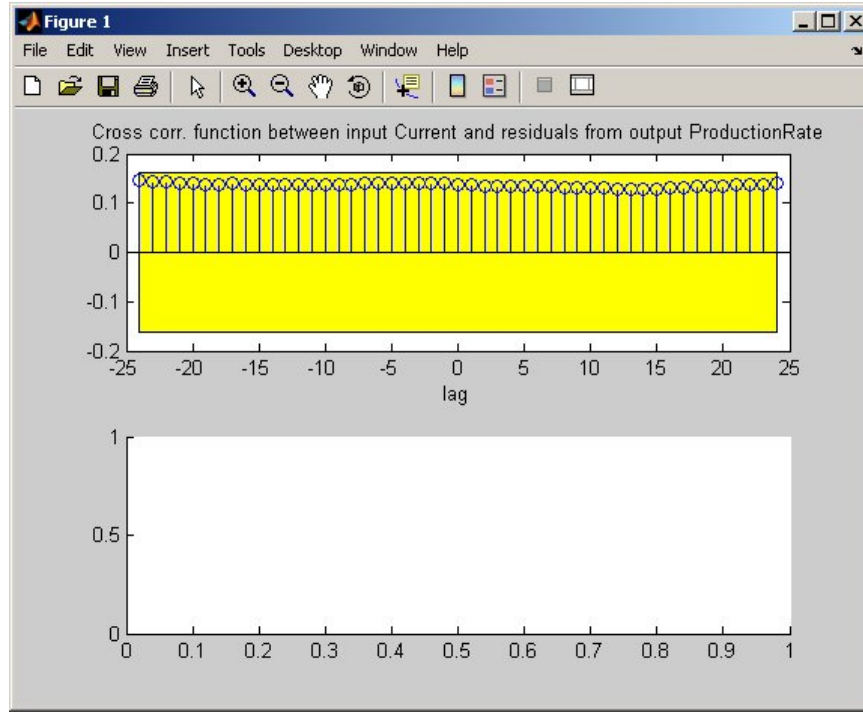
```
resid(Zv1,midproc0)
```

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 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 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 process models is accurate, but that there might be a need for a noise model. Next, adjust the algorithm settings and improve the results.

Estimating a Noise Model to Improve Results

In “Validating the Process Models” on page 4-37, you noticed that the process model performed well except that the it produced correlated residuals. This correlation of residuals indicates evidence of unmodeled dynamics, which

might be entering the system as an external disturbance. To capture such dynamics, you can estimate a noise model.

In this section, you estimate a noise model H as a continuous-time, first-order ARMA model and assume that e is white noise.

$$y = Gu + He$$

To specify computing the first-order ARMA noise model during estimation, specify the `DisturbanceModel` property as `'arma1'`:

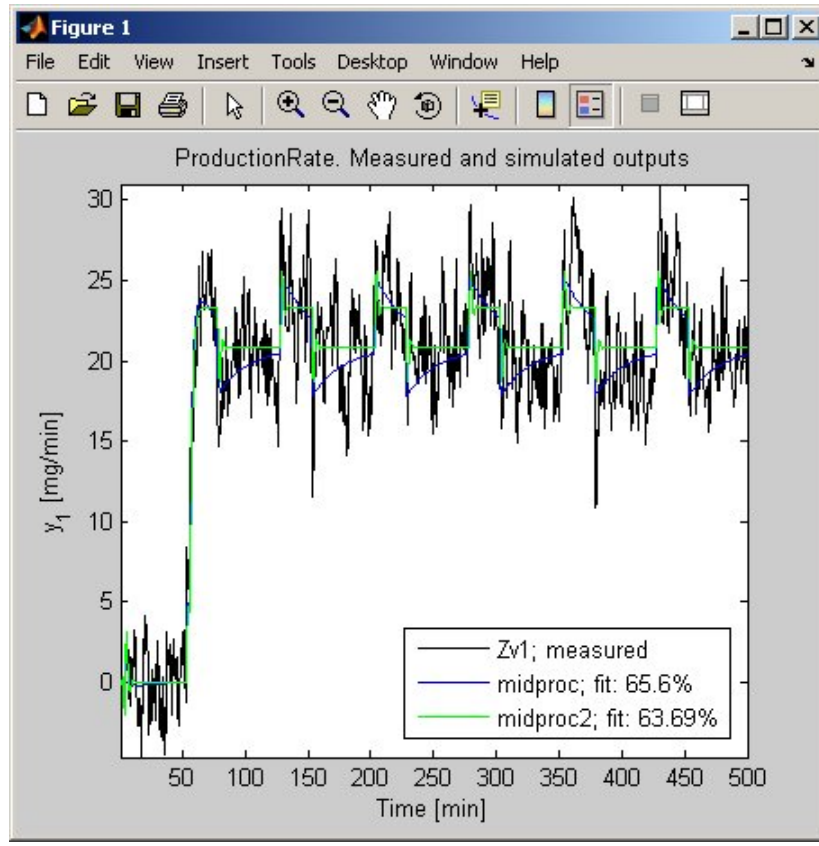
```
midproc2 = pem(Ze1,midproc0,'DisturbanceModel','arma1')
```

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

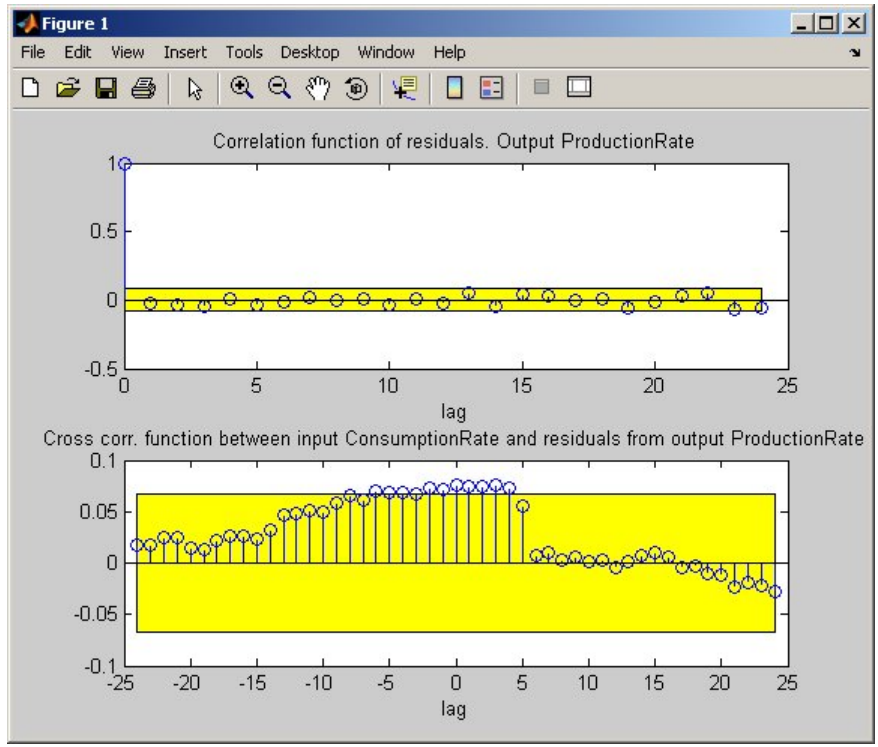
```
compare(Zv1,midproc,midproc2)
figure
resid(Zv1,midproc2)
```

The following plot shows that the model output is in reasonable agreement with the validation-data output: it is effectively unchanged from `midproc`, which does not include a noise model.



After MATLAB displays the first plot, press **Enter** to view the cross-correlation of the residuals with the second input.

The next plot shows that adding a noise model does improve the result by producing uncorrelated residuals: the top set of axes show that the autocorrelation values are inside the confidence bounds.



Estimating Black-Box Polynomial Models

In "Estimating Model Orders and Delays" on page 4-21, you estimated the model orders and found the following results:

- For the first input-output combination: two poles (n_a=2), 1 zero (n_b=2), and a delay of 5 (n_k=5).
- For the second input-output combination: 1 pole (n_a=1), no zeros (n_b=1), and a delay of 10 (n_k=10).

In this portion you of the tutorial, you estimate several different types of black-box polynomial models using these model-order estimates:

- "Linear ARX Model" on page 4-43
- "State-Space Model" on page 4-46
- "Box-Jenkins Model" on page 4-49

For an overview of supported black-box polynomial models, see "Supported Black-Box Models" on page 1-24.

Before you begin, create a subset of 1000 samples from 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 section on subreferencing `iddata` objects in the "Subreferencing iddata Objects".

Linear ARX Model

The ARX structure is given by the following general form:

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

To estimate an ARX model, you need to specify n_a as the number of poles, n_b as the number of b parameters (equal to the number of zeros plus 1), and n_k as the input delay.

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.

There are several ways to estimate an ARX model in System Identification Toolbox. Start by using `arx` to compute the polynomial coefficients using a fast, noniterative method:

```
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])
```

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

```
get(marx)
```

MATLAB returns the following model properties:

```
                     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 any of 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)
```

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

MATLAB returns the following output:

```
marx_poles =

    0.7751
    0.2585
   -0.0475
```

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

You can also use the iterative method `pem` to estimate an ARX model:

To estimate the ARX model using `pem`, enter:

```
marx_pem = pem(Ze1,'na',3,'nb',[2 1],'nk',[5 10]);
present(marx_pem)
```

In this case, MATLAB estimates the polynomials `A`, `B1`, and `B2` using the iterative method:

```
Discrete-time IDPOLY model: A(q)y(t) = B(q)u(t) + e(t)
A(q) = 1 - 0.9861 (+-0.03008) q^-1
         + 0.1512 (+-0.04243) q^-2
         + 0.009524 (+-0.02611) q^-3
B1(q) = 1.895 (+-0.1901) q^-5 - 1.594 (+-0.1898) q^-6
B2(q) = 1.826 (+-0.08123) q^-10
```

Commands `arx` and `pem` produce results that are in agreement within the parameter uncertainties.

State-Space Model

The discrete state-space model has the following structure, written in the *innovations form*:

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

The *order* of a state-space model is an integer equal to the dimension of *x(t)*, which is related to the number of delayed inputs and outputs used in the corresponding linear difference equation. For an overview of the black-box polynomial models supported by this Toolbox, see "Supported Black-Box Models" on page 1-24.

Similar to ARX, the state-space model structure is an excellent 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.

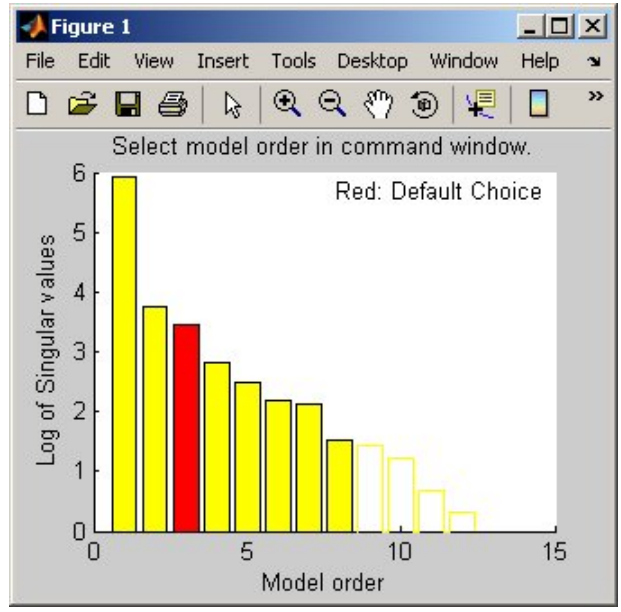
There are two methods to estimate a state-space model in System Identification Toolbox: noniterative (subspace) and iterative method. Start by using the fast, noniterative command `n4sid` to compute the coefficients.

The following command specifies a range of model orders and evaluates the performance of several state-space models (orders 2 to 8):

```
mn4sid = n4sid(Ze1,2:8,'nk',[5 10]);
```

This command opens the following plot, which recommends the most significant states. 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`.

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 following figure indicates that a third-order model might be sufficient.



System Identification Toolbox 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 the input delays.

Alternatively, you can use `pem` to estimate the state-space model. `pem` is an iterative method and has the following syntax:

```
pem(data,n,'nk',nk)
```

In this case, `data` is an `iddata` or `idfrd` object, and `n` and `nk` specify the model order and delay, respectively.

To estimate the state-space model using `pem`, type the following commands at the MATLAB prompt:

```
marx = pem(Ze1,3,'nk',[5 10]);
present(marx)
```

Box-Jenkins Model

A Box-Jenkins (BJ) model has the following general structure:

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

To estimate a BJ model, you need to specify the parameters n_b, n_f, n_c, n_d, and n_k. For an overview of the black-box polynomial models supported by System Identification Toolbox, see “Supported Black-Box Models” on page 1-24.

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.

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 at the MATLAB prompt and specify `nf=2`, `nb=2`, `nk=5` for the first input, and `nf=nb=1` and `nk=10` for the second input:

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

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 polynomials, as follows:

```
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.

You can separate the dynamics (measured components) and the noise using the following commands:

```
% Model with B and F polynomials only
mbj_meas = mbj('measure');
% Model with C and D polynomials only
mbj_noise = mbj('noise');
```

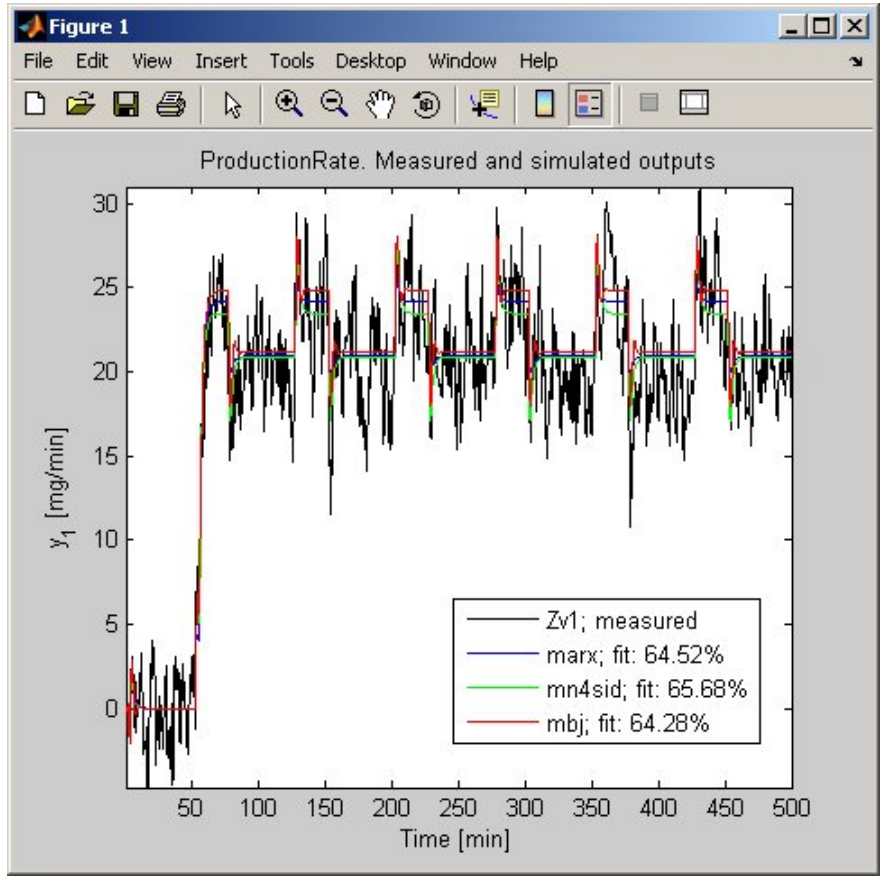
Separating the dynamics and the noise model is helpful in many applications. For example, in traditional control system design, you might only need the plant model and not the disturbance model.

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. After MATLAB displays the first plot, 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 black-box parametric models and adequately captures the process dynamics.

Using Models for Simulation and Prediction

After you estimate and validate a model, you can simulate the model output, use it to predict future outputs of a process, or insert the model into a control loop to design a controller.

In this portion of the tutorial, you use the continuous-time process model `midproc2`, created in "Estimating Continuous-Time Process Models" on page 4-31, to simulate and predict output. To learn more about the difference between simulation and prediction, see "Difference Between Simulation and Prediction" on page 1-37.

This section discusses the following topics:

- "Simulating the Model Response" on page 4-54
- "Predicting the Model Response" on page 4-56

Simulating the Model Response

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 code shows how to use the `idinput` and `iddata` commands to construct an input data set, and how to use `sim` to simulate the output:

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

To validate the model, simulate the model with different data sets and compare the model output to measured output. Carefully consider which initial conditions you use in the simulation. A system produces difference responses for different initial conditions, even when the input data to the model is the same. 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. If you do not specify the correct initial states, your model response does not match the measured output for your data.

Use `pe` to estimate the initial conditions `X0e` in the data set `Zv1`:

```
[Err,X0e] = pe(midproc2,Zv1,'estimate');
```

This function also computes the prediction errors `Err` between the simulated and the measured outputs.

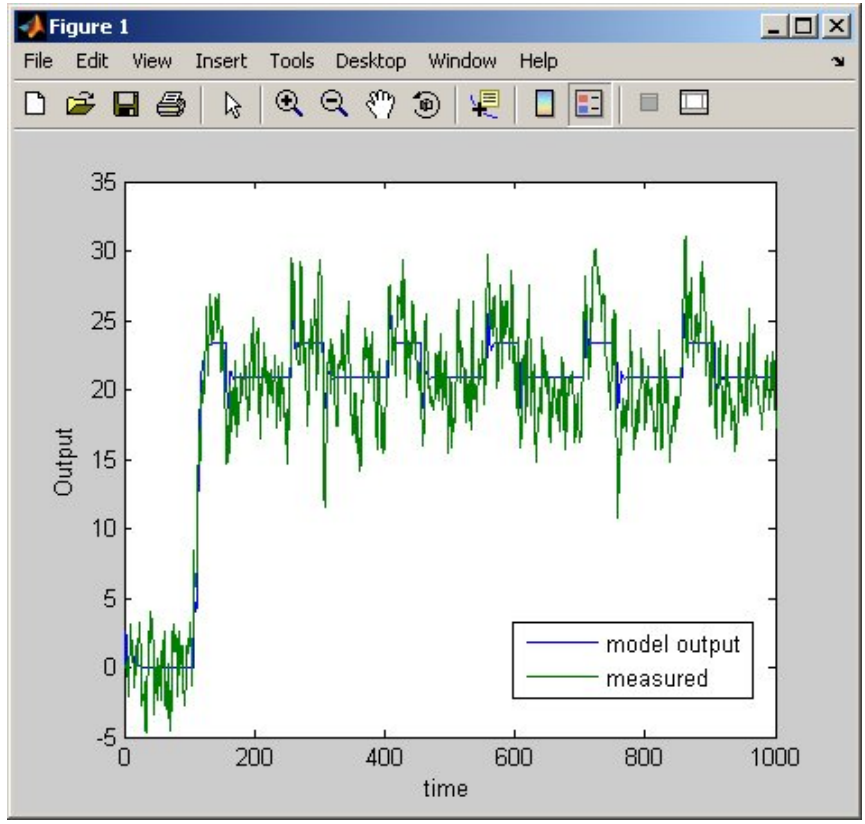
Next, simulate the model using the initial states estimated from the data:

```
ysim_2 = sim(midproc2,U,'InitialState',X0e);
```

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.



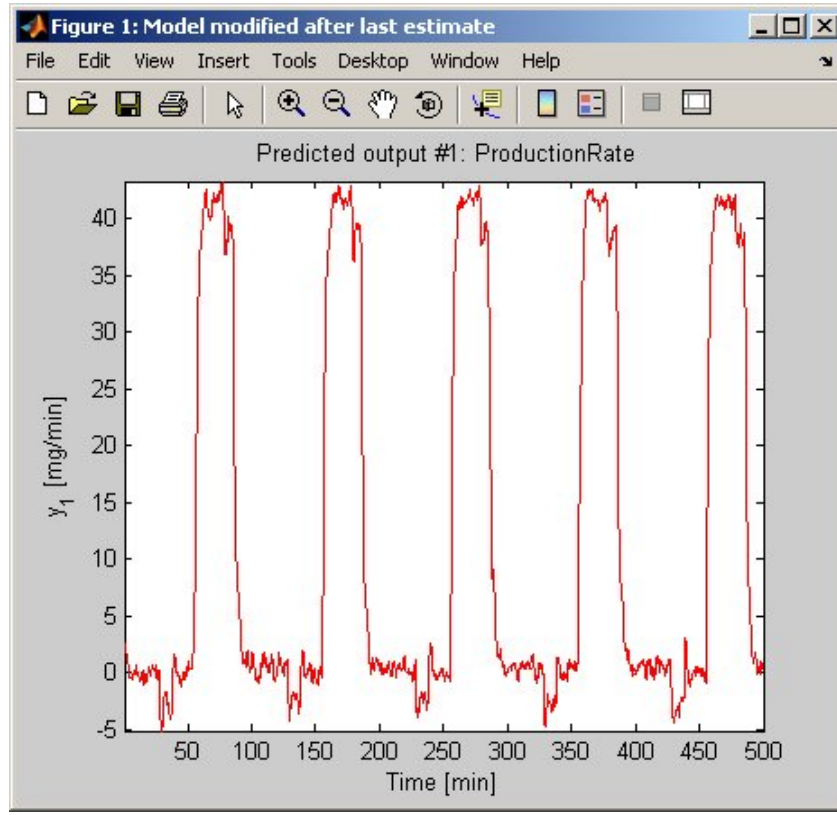
Predicting the Model Response

Many control-design applications require you to predict the response of a dynamic system.

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.

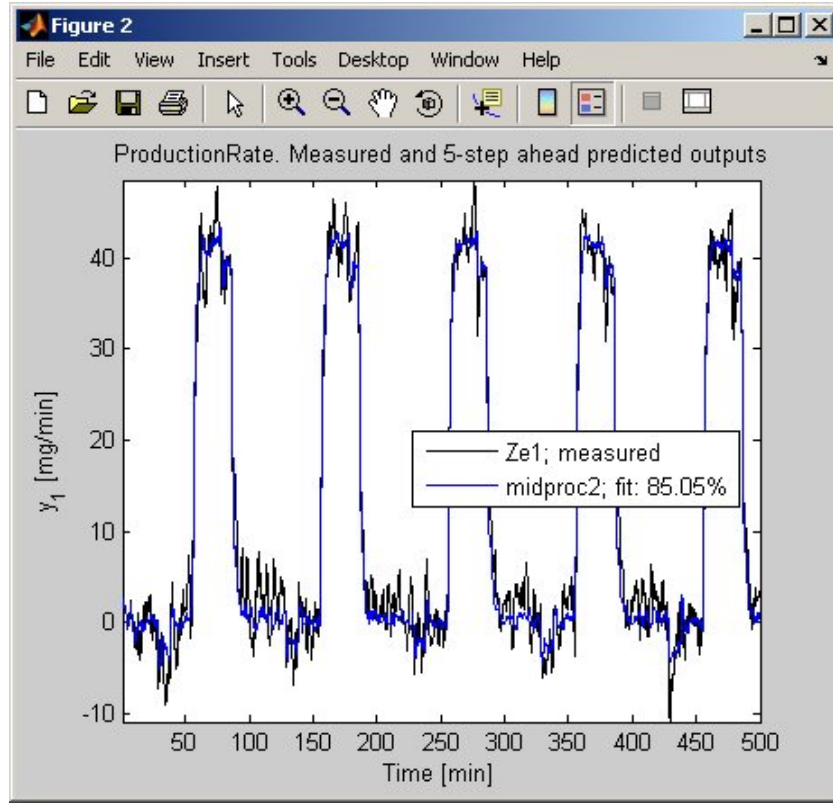


To compare the predicted output values with the measured output values, use the following command:

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

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

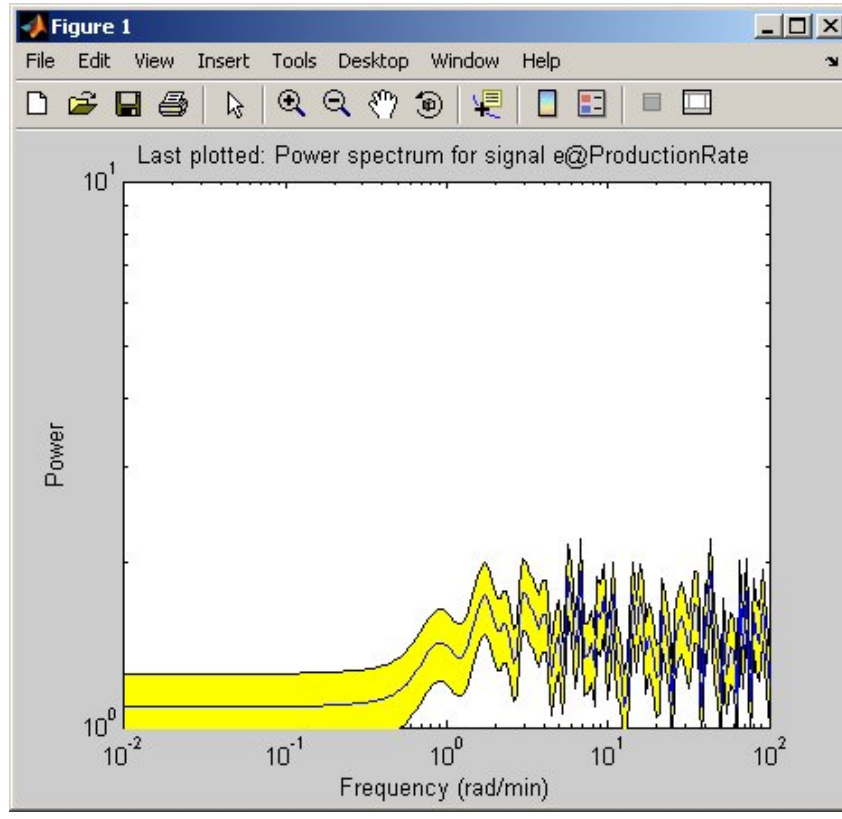
Note When you do not specify a third argument, as in "Simulating the Model Response" on page 4-54, `compare` assumes an infinite prediction horizon and simulates the model output.



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 higher at high frequencies because of the high-frequency nature of the disturbance.



Index

A

ARMAX
  definition 1-30
  estimating using the System Identification Tool 2-34
ARX
  definition 1-30
  estimating in MATLAB Command Window 4-43
  estimating using Quick Start 2-23

B

BJ model. *See* Box-Jenkins model
black-box model
  advantages 1-17
  estimating using Quick Start 2-23
  supported types 1-24
Box-Jenkins model
  definition 1-30
  estimating in MATLAB Command Window 4-43

C

continuous-time model 1-21 1-25
continuous-time process model. *See* process model

D

data
  creating `iddata` object 4-9
  estimation versus validation 1-15
  importing MAT–file into System Identification Tool 2-6
  importing object into System Identification Tool 3-7
  loading into MATLAB workspace 2-4
  plotting `iddata` object 4-9
  plotting in MATLAB Command Window 4-5
  plotting in the System Identification Tool 2-12
  preprocessing in the System Identification Tool 2-12
dead time 1-28
delay
  estimating in MATLAB Command Window 4-21
  estimating using the System Identification Tool 2-29
discrete-time model 1-21 1-25
disturbance. *See* noise
disturbance model. *See* noise model
dynamic system 1-15

E

estimation data 1-15
exporting models
  to LTI Viewer 2-50
  to the MATLAB workspace 2-48

F

frequency-response model
  definition 1-25
  estimating in MATLAB Command Window 4-16
  estimating using Quick Start 2-23

G

grey-box model 1-17

I

`iddata` object
  creating 4-9
  plotting 4-9
importing data

iddata object into System Identification Tool 3-7
MAT-file into System Identification Tool 2-6
impulse-response model
estimating using Quick Start 2-23
in MATLAB Command Window 4-16
independence test 1-40

L

linear model
estimating in MATLAB Command Window 4-3
estimating using the System Identification Tool 2-3
versus nonlinear 1-18
linear nonparametric model 1-25
loading data into MATLAB workspace 2-4
LTI Viewer 2-50

M

model
choosing grey-box or black-box 1-17
choosing linear or nonlinear 1-18
continuous time 1-21 1-25
definition 1-15
discrete time 1-21 1-25
estimating in MATLAB Command Window 4-3
estimating in the System Identification Tool 2-3
estimating low-order continuous-time 3-3
estimating using Quick Start 2-23
nonparametric 1-25
parametric 1-25
supported black-box 1-24
using 4-54
validating 1-36
model order
definition 1-28
estimating in MATLAB Command Window 4-21
estimating using the System Identification Tool 2-29
model parameters
viewing in the System Identification Tool 2-45

N

noise 1-15
noise model
definition 1-23
estimating for process model 3-22
when to estimate 1-24
nonlinear model
Hammerstein-Wiener model 1-32
nonlinear ARX model 1-32
when to estimate 1-18
nonparametric model 1-25
analyzing plots 2-25

O

OE model. *See* Output-Error model
order. *See* model order
Output-Error model
definition 1-30

P

parametric model
definition 1-25
plotting data
in MATLAB Command Window 4-5
in the System Identification Tool 2-12
plotting models
in LTI Viewer 2-50
polynomial models 1-28
prediction

in MATLAB Command Window 4-54
versus simulation 1-37
preprocessing data
in the System Identification Tool 2-12
process model
defining structure using idproc 4-31
definition 1-27
estimating in MATLAB Command Window 4-31
estimating noise models 3-22
estimating using the System Identification Tool 3-3

Q

Quick Start
for estimating models 2-23
for preprocessing data 2-20

R

removing data sets in the System Identification Tool 2-21
residuals
definition 1-40
plotting using the System Identification Tool 2-42

S

simulation
in MATLAB Command Window 4-54
using Simulink 3-29
versus prediction 1-37
Simulink 3-29
state-space model
definition 1-31
estimating in MATLAB Command Window 4-43
estimating using Quick Start 2-23
estimating using the System Identification Tool 2-34
step-response model
estimating using Quick Start 2-23
in MATLAB Command Window 4-16
System Identification Tool
estimating continuous–time process models 3-3
estimating linear models 2-3
estimating models using Quick Start 2-23
exporting models to LTI Viewer 2-50
exporting models to the MATLAB workspace 2-48
removing data sets 2-21
saving sessions 2-21
starting 2-4
versus MATLAB Command Window 1-8
System Identification Toolbox
about 1-2
demos 1-3
documentation 1-5
installation 1-3
related products 1-6
resources 1-43
using with Simulink 3-29
workflow 1-9

T

transient-response model
description 1-25
estimating in MATLAB Command Window 4-16
estimating using Quick Start 2-23
trash 2-21

V

validating models 1-36
analyzing residuals 1-40

comparing output 1-36
in MATLAB Command Window 4-52
using the System Identification Tool 2-39
validation data 1-15

W

whiteness test 1-40
workflow for System Identification Toolbox 1-9
example 1-11